دانشکده مهندسی برق و کامپیوتر

جزوه درسی

معماری نرم‌افزار

دکتر فریدون شمس

فهرست مطالب

فصل ۱- چرخه کاری معماری

۱-۱- مقدمه

دهه‌های متوالی، طراحان نرم‌افزار یاد می‌گرفتند که سیستم‌ها را منحصراً بر پایه تعاریف نیازمندیهای عملیاتی بسازند. بنابراین مستندات نیازمندیها به طراح داده می‌شد و طراح می‌بایست با طراحی قابل قبولی برمی‌گشت. امروزه، متدهای مدرن تولید نرم‌افزار، به سادگی و بکری این روش اقرار دارند و تمام چرخه‌های اطلاعات را از طراح به تحلیل‌گر (و برعکس) ایجاد می‌کنند، اما به این باور ضمنی نیز رسیده‌اند که طراحی، خود محصولی از بازه تشخیص نیازمندیهای عملیاتی سیستم است.

نکته دیگری که مورد توجه قرار گرفت، تشخیص و وجود نیازمندیهایی به غیر از نیازمندیهای فنی است. تشخیص اینکه نیازمندیهای دیگری که بر سیستم تاثیر می‌گذارند، چه هستند و پوشش آنها باید چگونه در نظر گرفته شود، سوالاتی هستند که در دهه‌های قبل بدون پاسخ مانده‌اند و جواب آن در وجود معماری است.

۱-۲- مثالی از عدم وجود معماری

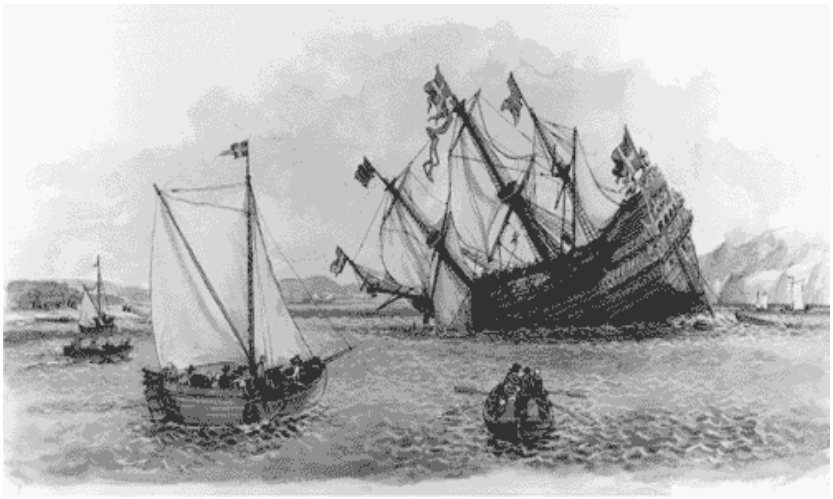
در سال ۱۶۲۰ بین سوئد و فنلاند جنگ سختی در گرفت. پادشاه سوئد – Gustavus Adolphus – مصمم شد تا جنگ را سریعاً خاتمه دهد، به همین دلیل تصمیم به ساخت یک ناو جنگی جدید گرفت که شبیه آن تا آن زمان دیده نشده بود. Vasa نام این ناو جنگی ترسناک بود که ۷۰۰ متر طول، قابلیت حمل بیش از ۳۰۰ سرباز و ۶۴ توپ در دو عرشه را داشت. شاه، بر قدرت تهاجمی آن اصرار بسیار داشت تا بتواند هر کشتی جنگی دیگری را با آن مورد تاخت و تاز قرار دهد.

برای ساخت این ناو جنگی، شاه معروفترین سازنده کشتی که یک فرد دانمارکی به‌نام Henrik Hybertsson را استخدام کرده بود. این فرد کشتی‌های بزرگی ساخته بود، اما هیچوقت کشتی جنگی به اندازه Vasa نساخته بود. در آن زمان عرشه‌ای با دو توپخانه بسیار کمیاب بود و البته در اندازه Vasa وجود نداشت. مانند همه معماران، Hybertsson سعی به ایجاد توازن بین خصوصیات و دغدغه‌ها نمود.

- ❖ باید ناوی جنگی را در زمان کم و با هزینه پائین آماده سازم
- ❖ سرعت درخواستی پادشاه را برآورده سازم

- ❖ قابلیت اعتماد و امنیت آن را تضمین دهم
- ❖ اما تجربه من نشان می‌دهد که یک عرشه باید توپخانه داشته باشد

این کار از نظر معماری غیرممکن می‌نمود اما با اصرار شاه این پروژه شروع شد و بالاخره در ۱۰ آگوست ۱۶۲۸ کار ساخت این ناو به اتمام رسید.



شکل ۱-۱- ناو جنگی Vasa

بعد از به آب انداختن این ناو جنگی و پس از اینکه اولین شلیک خود را انجام داد، به دو نیم تقسیم شد و بیش از ۱۵۰ نفر از خدمه آن غرق شدند. گفته می‌شود که Vasa «نامتناسب» ساخته شده بود. به عبارت بهتر، معماری آن دارای نقص بوده است.

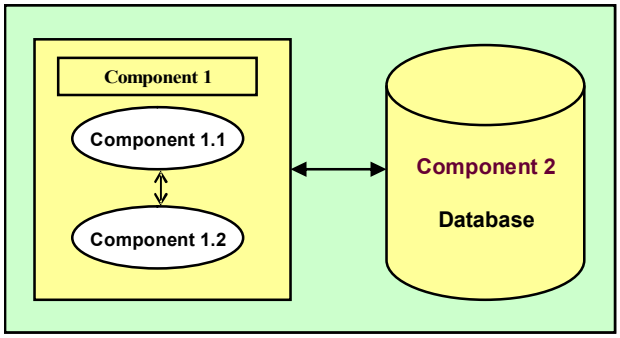
۱-۳- تاریخچه معماری

در طول تاریخ کامپیوتر، نرم‌افزار بسرعت پیچیده و پیچیده‌تر شده است. راه‌هایی برای مقابله با پیچیدگی‌ها نیز ایجاد شدند مثل: برنامه‌سازی ساخت یافته، ایده جامعیت مفهومی و غیره. در دهه ۷۰، Brooks راجع به اهمیت معماری نوشت، البته او بیشتر راجع به چیزی صحبت می‌کرد که امروزه ما آن را «واسط» می‌نامیم. در سال ۱۹۹۴ ، Dargan & Denning ، معماری نرم‌افزار را با اصول جدید نرم‌افزار توصیف کردند ولی توصیف آنها هم به نظر، بیشتر «متدی برای تولید» می‌رسد تا تعریف. بدین ترتیب تعریفی جامع از معماری تا اواسط دهه ۹۰ وجود نداشت. در سال ۱۹۹۶، Garlan و Show، «توجه به معماری بعنوان سطحی جدا در طراحی نرم‌افزار» را به عنوان ایده‌ای نو مطرح نمودند.

در واقع، پایه‌های مطالعه در حیطه معماری نرم‌افزار – با اینکه مدت زیادی از به رسمی شناخته شدنش نمی‌گذرد– در مطالعات راجع به ساختارهای نرم‌افزار شکل گرفت و برای اولین بار درسال ۱۹۶۸ در مقاله‌ای از Edsger Dijkstra مطرح شد. Dijkstra ، مطرح کرد که تمرکز و توجه به اینکه نرم‌افزار چگونه تقسیم‌بندی شده و ساختارش چگونه است، درست همانند کاری است که با یک برنامه ساده می‌کنیم تا نتیجه درستی بدهد و بسیار ارزشمند است. David Parnas، این خط فکری را گسترش داد و مسائلی از قبیل ماژول‌ها، پنهان‌سازی اطلاعات، ساختارهای نرم‌افزار و ... را مطرح کرد که همگی بر خصوصیاتی از نرم‌افزار تأکید دارند که در دیدی اقتصادی به تولید و نگهداری از سیستم تأثیر می‌گذارند.

۱-۴- تعریف معماری نرم‌افزار

معماری یک سیستم نرم‌افزاری، blue print سیستم در بالاترین سطح انتزاع است که مؤلفه‌های اصلی و مهمترین محاورات بین آنها را توصیف می‌کند. نرم‌افزار پیچیده نیاز به ساختار دارد و ساختار باید پیاده‌سازی و مستندسازی شود. معماری نقطه شروع طراحی نرم‌افزار است و به همین دلیل دیدی کلان از عناصر و ارتباطات آن ارائه می‌دهد. شکل ۱-۲ مجموعه‌ای از مولفه‌های اصلی و ارتباطات بین یک سیستم را نشان می‌دهد.



شکل ۱-۲- نمایش مولفه‌ها و ارتباطات در معماری

تعریف واژه‌ای مثل «معماری نرم‌افزار» کار ساده‌ای نیست. از شکل بالا تنها می‌توان برخی نتایج کلی را استنتاج کرد. معماری نرم‌افزار با بالاترین سطح طراحی یک سیستم سروکار دارد. می‌توان گفت معماری نرم‌افزار مجموعه‌ای از مؤلفه‌های مرتبط با هم است. این تعریف با فرض‌های اولیه نیاز به ابزاری گرافیکی دارد و می‌توان گفت با چنین ارائه‌ای بهتر قابل لمس است.

برای درک بهتر تعریفی از معماری ارائه می‌شود که در سراسر این متن قابل استفاده است. «معماری نرم‌افزار یک برنامه یا سیستم محاسباتی، ساختار یا ساختارهای آن سیستم محاسباتی است که خصوصیات قابل رویت از بیرون عناصر و ارتباطات بین آنها را نشان می‌دهد.»

با زبانی غیر رسمی می‌توان چنین گفت: «معماری نرم‌افزار، در رابطه با ساختارهای سیستم‌های نرم‌افزاری بزرگ است.» دید معمارانه به سیستم یک دید انتزاعی است که جدا از جزئیات پیاده‌سازی و الگوریتم نمایش داده شده است و تمرکز بر رفتار و محاورات مؤلفه‌ها (بصورت Black Box) دارد.

با این تعریف، نقش جدیدی به مجموعه توسعه‌دهندگان نرم‌افزار افزوده می‌شود که همان «معمار نرم‌افزار» است. وظیفه معمار نرم‌افزار، طراحی و مستندسازی معماری است تا به کمک آن بتوان طراحی جزئیات انجام شود. مطمئناً مستندسازی معماری صرفاً رسم یک نمودار نیست بلکه عناصر، ارتباطات و دیدگاه معمار باید به صورت کامل تشریح شود. معمار در هر سیستم نرم‌افزاری دیدگاه‌های خاصی را متناسب با نیازمندیها، اجبارها و قابلیت‌های مورد نیاز آن سیستم ارائه می‌دهد. این دیدگاه‌ها در هر سیستم می‌توانند متفاوت باشند. به همین خاطر، در صورتیکه دو معمار نرم‌افزار در دو سازمان متفاوت بر روی سیستمی که نیازمندی‌های مشابه دارند، کار کنند، حاصل کار آنها می‌تواند معماری مشابه‌ای نباشد.

۱-۵- شکل‌گیری معماری نرم‌افزار

معماری حاصل مجموعه‌ای از تصمیمات فنی و کاری است. از جمله موثرترین و اولین عوامل تاثیر گذار بر معماری می‌توان به نیازمندیهای سیستم اشاره نمود. نیازمندیهای سیستم از نقطه نظرات متفاوت سبب ایجاد تصمیم‌گیری‌های مختلف خواهند شد که هر یک از تصمیمات می‌توانند به نرم‌افزاری کارآمد یا غیرکارآمد منتهی شوند. عوامل مختلفی دیگری نیز در تصمیمات معماری تاثیر می‌گذارند که در قسمت بعدی به مطالعه آنها خواهیم پرداخت.

تصمیماتی که در معماری اخذ می‌شود در مورد سه موضوع اصلی هستند که عبارتند از:

- ❖ طراحی آن
- ❖ تحقق تاثیرات مختلف
- ❖ محیطی که معماری باید اجرا شود

۱-۶- عوامل تاثیرگذار بر معماری نرم‌افزار

بطور کلی می‌توان عوامل تاثیرگذار بر معماری نرم‌افزار را در چهار گروه ذیل دسته‌بندی نمود:

- **ذینفعان:** هر ذینفع داری علاقه‌مندی و اهداف خاص خود است که برخی از اهداف با یکدیگر منافات دارند. معماری نرم‌افزار با دانستن این اهداف سعی به ایجاد توازن و برآورده نمودن آنها می‌نماید. هر چند برآورده نمودن نظر همه اهداف ذینفعان در سطح ایده‌آل تقریباً ناممکن است، اما معمار می‌تواند با ایجاد توازن بین آنها، در سطح مناسبی آنها را برآورده سازد.
- **سازمان توسعه دهنده:** ساختار و ماهیت سازمان توسعه‌دهنده بر معماری تاثیر گذار است. برخی سازمان‌ها به سرمایه‌گذاری سریع می‌اندیشند و انتظارشان از نرم‌افزار بهره‌دهی کوتاه‌مدت است، در این سازمان‌ها معماری نرم‌افزار شکل چابک‌تری به خود می‌گیرد. سازمان‌هایی نیز وجود دارند که سرمایه‌گذاری طولانی مدت در نرم‌افزار مد نظر است که در این حالت معماری سعی به افزایش قابلیت استفاده مولفه‌ها می‌نماید.
- **تجربیات و پیش زمینه معمار:** تکرار نتایج خوب کسب شده از تجربیات معمار سبب اجتناب از شکست خواهد شد. هر تجربه خوب یا بد معمار می‌تواند بر معماری اثر داشته باشد.
- **محیط فنی:** استانداردهای عملی صنعتی یا فنون متداول مهندسی نرم‌افزار اثر مستقیمی بر معماری نرم‌افزار دارند. معماری سعی به استفاده از این استانداردها در توسعه نرم‌افزار می‌نماید.

علاوه بر عوامل ذکر شده، عوامل دیگری نیز به صورت ضمنی بر معماری تاثیر می‌گذارند که اثر آنها در شرایط خاص می‌تواند مهم باشد. شناسایی این عوامل، منبع و ماهیت آنها می‌تواند کمک شایانی به توسعه معماری نماید. در واقع، این معمار است که باید ماهیت، منبع و اولویت اجبارهای روی پروژه را هر چه زودتر درک نماید تا بتواند معماری پایاتری ارائه نماید. دو روش بازبینی معماری و استفاده از نمونه‌ها[1] برای این کار وجود دارد که در بخش بعدی به بررسی آنها خواهیم پرداخت.

۱-۷- چرخه کاری معماری

همانطور که ذینفعان، سازمان توسعه‌دهنده، محیط فنی و تجربه معمار بر معماری تاثیر می‌گذارند و سبب رشد آن می شوند، معماری نیز بر این عوامل تاثیر می‌گذارد و سبب تغییر آنها خواهد شد. درک

[1] Prototype

معمار از ایـن تـاثیر دو طرفه سبب رشد معمـاری و رشد عوامل موثر خواهد شد. مـی‌تـوان از دو بعـد ایـن تاثیر را پیگیری نمود:

- **تاثیر معماری بر سازمان توسعه دهنده**
  - ساختار و منابع سازمانی
    - واحدهای کاری مطابق با قسمت‌های معماری تنظیم می‌شوند.
    - زمانبندی پروژه‌ها از معماری تاثیر می‌پذیرند و در معماری زمانبنـدی مشـخص می‌شود.
    - بودجه پروژه‌ها توسط معماری تخمین زده می‌شوند.
  - اهداف سازمان
    - تجربه در ساخت نوع خاصی از سیستم
    - موفقیت در بازار
    - ارزیابی بازار
    - دارایی‌های خط تولید[1]
- **تاثیر معماری بر عوامل تاثیرگذار**
  - تاثیر معماری بر نیازمندی‌های مشتریان
    - دانش مشتریان برای درخواست خصوصیات خاصی در سیستم بعدی
    - حمایت از بروز رسانی، تطبیق و غیره
  - تاثیر معماری بر تجربه معمار
    - ایجاد سیستم، موفقیت و حتی شکست، تجربه معمار را تحت تاثیر قرار خواهد داد.
  - تاثیر معماری بر محیط فنی
    - برخی اوقات، سیستم یا معماری محیط فنـی را تحـت تـاثیر قـرار خواهـد داد. بدین ترتیب که برخـی اسـتانداردها و خصوصـیات فنـی را بـر محـیط اجبـار می‌نماید.

[1] Product Line

با توجه به مطالب ذکر شده می‌توان بیان کرد که معماری نرم‌افزار نتیجه تاثیرات فنی، حرفه و اجتماعی است و بر محیط فنی، حرفه و اجتماعی نیز تاثیر می‌گذارد. این تاثیرات دو جانبه که چرخه‌ای را از معماری بر عوامل و بالعکس ایجاد می‌نماید «چرخه کاری معماری» اطلاق می‌شود. معمار با شناخت چرخه کاری معماری می‌توان عواملی را شناسایی نماید که بر معماری تاثیر گذاشته و از آن تاثیر می‌پذیرند.

۱-۸- مراحل فرآیند معماری نرم افزار

مراحل آفرینش یک معماری نرم‌افزار، که با استفاده از آن معماری بتوان به طراحی تحقق بخشید و پیاده‌سازی و مدیریت ساخت سیستم نهایی را ایجاد کرد، شامل فعالیت‌هایی به شرح زیر است:

الف ـ ایجاد یک مورد کاری برای سیستم[1]

سوالاتی پیرامون اندازه هزینه برای تولید محصول، قیمت نهایی، میزان سود و محدودیت‌های محیطی سیستم و غیره، سوالاتی هستند که یک معمار به تنهایی می‌تواند به آنها جواب دهد و از طرفی بدون جواب این سوال‌ها، رسیدن به اهداف حرفه میسر نیست. به عنوان نمونه در متدلوژی USDP، در مرحله آغازین[2] فعالیتی بنام تشخیص موارد کاربری وجود دارد که رسیدن به این سطح را میسر می‌سازد.

ب ـ فهم نیازمندیها

راه‌های زیادی برای شناخت نیازمندی‌ها، از کاربر نهایی و خریداران سیستم، وجود دارد، همانند استفاده از سناریو در روش شیءگرا، یا استفاده از موارد کاربری برای تصویر نیازمندی‌ها. یک تکنیک پایه برای فهم نیازمندی‌های یک سیستم در حال تولید، شناخت محدوده تغییرات و تفاوت‌های آن سیستم در مقایسه با سیستم‌های مشابه قدیمی‌تر است. اینکار بدین سبب بسیار جا افتاده که امروزه سیستمی نمی‌توان پیدا کرد که خیلی از سیستم‌های مشابه دور باشد و شناخت نیازمندیهای یک سیستم با فهم خصوصیات و نیازمندی‌های سیستم‌های قبلی مشابه رابطه مستقیم دارد. محصول خروجی مرحله تحلیل، که دامنه مساله را مدلسازی می‌کند، تفاوت‌های بین سیستم‌های اجرایی مشابه را معرفی می‌کند.

تکنیک دیگری که به فهم نیازمندی‌ها کمک می‌کند، ایجاد نمونه‌هاست. نمونه‌ها می‌توانند برای مدلسازی رفتار مطلوب، طراحی واسط کاربری و آنالیز منابع کمک کنند. با اینکار می‌توان سیستم را به صورت واقعی جلوی چشمان سهامداران ایجاد کرد و تصمیماتی روی آن گرفت. گذشته از فهم نیازمندیها، کیفیت‌های

[1] Business Case
[2] Inception

مطلوب سیستم تحت ساخت، شکل معماری آن را می‌سازد. حال چگونه می‌توان بین نیازمندیهای متضاد، میانه روی کرد و به میزان و بالانسی مناسب در معماری رسید که با توجه به اولویت‌های نیازمندیها، ایجاد شده است. اینکار با نمونه‌سازی[1] میسر است.

ج ـ آفرینش، انتخاب و سفارشی‌سازی معماری

نباید متدی برای ایجاد معماری، از قبل دیکته شود. اگر شرایط و قیودی از قبل تعیین شده باشند، همیشه اولویت را به برخی کیفیت‌های خاص می‌دهند. این تأکید اولویت‌ها بصورت ضمنی است و اگر کیفیت‌های مورد نظر در این متد با نیازمندیهای سیستم مطلوب هماهنگ نباشند ایجاد مشکل خواهد کرد. در عوض بهتر است روشی ارائه شود که در آن روش، طراحی به صورتی انعطاف‌پذیر تولید شود و در طی تحلیل، با توجه به نیازمندیها، شکل محکم‌تری به خود بگیرد. در ابتدا چند مدل طراحی پیشنهاد می‌شود ولی بتدریج برخی از آنها رد می‌شوند. انتخاب منطقی بین این طراحی‌های کاندیدا، یکی از کارهای مهم و بزرگ معمار است.

د ـ نمایش و ارائه مدل معماری

برای اینکه یک معماری بتواند بعنوان ستون فقرات طراحی یک پروژه باشد، باید بتواند بین همه سهامداران، بسیار آشکار و نامبهم معرفی و ارائه شود. مدل نمایشی باید حاوی اطلاعات مفید، غیر مبهم و قابل خواندن توسط افراد در زمینه‌های مختلف باشد. خود معمار نیز باید مطمئن باشد که معماری ارائه شونده، نیازمندیهای کیفی و رفتاری مطلوب را برآورده می‌سازد. می‌توانیم مدل ارائه شونده را، ورودی برای مرحله تحلیل رسمی بدانیم (همانند شبیه‌سازی، وارسی یا نمونه‌سازی). معماری را می‌توان با استفاده از زبان‌هایی رسمی بنام زبان توصیف معماری (ADL) نمایش داد.

ه ـ تحلیل یا ارزیابی معماری

ADLها قابلیت‌های زیادی در تحلیل دارند ولی بیشتر تمرکزشان روی خصوصیات اجرایی سیستم است (کارایی، رفتار، الگوهای ارتباطی و غیره). تکنیک‌های تحلیل کمی، برای ارزیابی معیارهای غیر اجرایی در اختیار یک معمار است. به‌عنوان نمونه Maintainability که خود شامل وجوه: Portability، Reusability، Adaptability وغیره است که همگی خصوصیات سیستم در مقابل تغییرات را نشان می‌دهند.

[1] Prototyping

و ـ پیاده‌سازی سیستم بر پایه معماری

در این مرحله باید مطمئن باشیم که تولیدکنندگان سیستم، به ساختارها و پروتکل‌های ارتباطی براساس معماری، متعهد هستند. اولین قدم در این راه، داشتن یک معماری واضح، صریح و قابل تبادل است. بهتر است محیطی داشته باشیم که به تولیدکنندگان سیستم در شناخت و بکارگیری معماری کمک و راهنمایی شود. نهایتاً وقتی معماری ساخته و بکار گرفته شد، به فاز نگهداری منتقل می شود. باید در این فاز تضمین شود که معماری هنوز به شکل نمایش داده شده خود، متعهد است.

ی) تضمین همنوایی پیاده‌سازی با معماری

بعد از ایجاد و استفاده از معماری مرحله نگهداری شروع می‌شود تا مستندات و معماری بروز نگهداری شوند.

۱-۹- شناخت معماری خوب

واقعاً نمی‌توان یک معماری را بد یا خوب نامید، بلکه تنها می‌توان بیان نمود که اهداف خود را کمتر یا بهتر برآورده ساخته است. معماری می‌تواند مورد ارزیابی قرار گیرد و ارزیابی مشخص می‌کند که معماری چقدر خصوصیات و اهداف مورد نظرش را برآورده ساخته است. بطور کلی برای خوب بودن معماری می‌توان سه معیار کلان را مطرح نمود که اغلب مورد توجه قرار می‌گیرد:

- دستیابی به اهداف
- قابل انجام با بودجه معقول
- قابل انجام در زمان معقول

برای افزایش کیفیت معماری می‌توان برخی نکات پیشنهادی مربوط به فرآیند و ساختار را رعایت نمود که البته عدم رعایت آنها سبب خطا در معماری نمی‌شوند.

❖ **مفاهیم راهنما مربوط به فرآیند**

- معماری باید محصول یک معمار یا یک تیم معماری با رهبر مشخص باشد
- معمار (تیم معماری) باید نیازمندیهای وظیفه‌مندی و فهرست اولویت‌بندی شده‌ای از خصوصیات کیفی در دست داشته باشد
- معماری باید به خوبی و با حداقل یک دید ایستا و یک دید پویا مستند شود

- معماری باید بین ذینفعان توزیع شود، تا بتوانند حضور فعال در بازنگری آن داشته باشند
- معماری باید برای تحقق معیارهای عددی قابل دستیابی ارزیابی شود
- معماری باید به گونه‌ای باشد تا بتواند به صورت افزایشی پیاده‌سازی شود
- معماری باید منجر به بیان مجموعه‌ای از حیطه‌هایی که در آنها رقابت منابع وجود دارند و حل هر یک از آنها شود

❖ **مفاهیم راهنما مربوط به ساختار**

- معماری باید براساس ماژول‌های خوش تعریف بوده و انتساب وظیفه‌مندی به آنها بر پایه مخفی سازی اطلاعات و تفکیک وظایف باشد
- هر ماژول باید دارای یک رابط خوش تعریف باشد
- خصوصیات کیفی باید توسط تاکتیک‌های آشنا انجام شوند
- معماری نباید وابسته به نسخه خاصی از محصول تجاری یا ابزار باشد
- ماژول‌هایی تولید کننده داده باید از مصرف‌کننده‌ها مجزا شوند
- برای سیستم‌های موازی، معماری باید فرآیندها یا وظایف خوش تعریف را نمایش دهد
- هر وظیفه باید به پردازنده خاصی نسبت داده شود
- معماری باید مجموعه کوچکی از الگوهای تعاملی ساده را به طور مشخص نشان دهد

فهرست مطالب

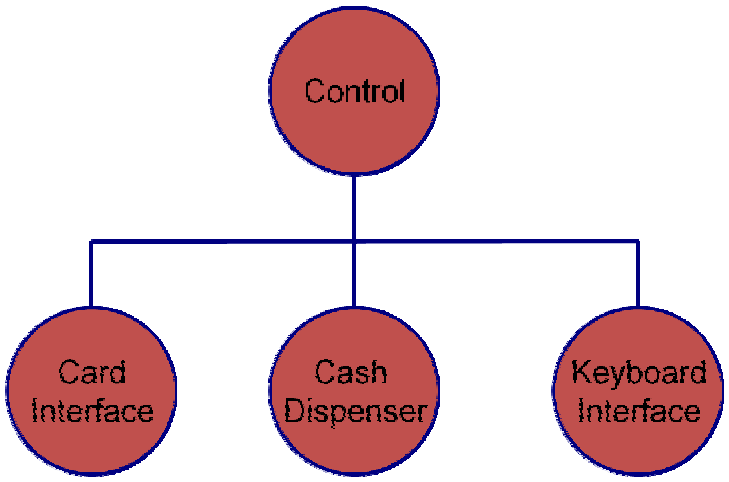
فصل دوم

معماری نرم‌افزار چیست؟

در این فصل مفهوم معماری از دیدگاه مهندسی نرم‌افزار بررسی خواهد شد. به عبارت بهتر، فواید حاصل از بکارگیری معماری نرم‌افزار در پروژه‌های توسعه نرم‌افزاری و همچنین نتایج مثبت به ارمغان آورده شده برای سازمان استفاده کننده از معماری نرم‌افزار تشریح می‌شوند.

۲-۱- معماری نرم‌افزار چه هست و چه نیست؟

برای درک معماری نرم‌افزار از یک مثال ساده شروع می‌کنیم. شکل ۲-۱ توصیفی از سیستم ATM است. هدف از این نمودار تشریح معماری سطح بالای سیستم است. این نوع نمودار، از جمله نمودارهای است که اغلب برای کمک به تشریح یک معماری مورد استفاده قرار می‌گیرد. حال سئوال مهم این است که دقیقا چه چیزهایی را از روی این نمودار می‌توان بیان کرد؟



شکل ۲-۱ : نمایش معماری سطح بالا ATM

تعدادی از جواب‌ها به شرح زیر هستند:

- سیستم متشکل از چهار عنصر است.
- عناصر Card Interface, Cash Dispenser and Keyboard Interface ممکن است ارتباط بیشتری با یکدیگر نسبت به عنصر Control داشته باشند زیرا نزدیک به یکدیگر هستند.
- تمامی عناصر با یکدیگر ارتباط دارند زیرا نمودار کاملا متصل است.

حال سئوال اساسی این است که آیا این نمودار یک معماری است؟ فرض کنید که معماری متشکل از مولفه‌ها و ارتباطات بین آنها باشد. بنابراین با توجه به این تعریف به نظر می‌رسد که این نمودار برای یک معماری کافی باشد. اما سئوال بعدی این است که چه چیزهایی را نمی‌توان از روی این نمودار بیان کرد؟ جواب این سئوال در عناوین زیر خلاصه می شود:

- *ماهیت هر یک از عناصر* چیست؟

ماهیت جداسازی عناصر چیست؟ آیا هر کدام بر روی یک پردازنده اجرا می‌شوند؟ آیا آنها در زمانهای مختلف اجرا می‌شوند؟ عناصر متشکل از پردازنده‌ها، برنامه‌ها یا هر دوی اینها هستند؟ آیا آنها روشی که منجر به تقسیم شده است را نشان می‌دهند؟ آیا آنها مفهوم جداسازی زمان اجرا را نمایش می‌دهند؟ آیا آنها شیء، پردازنده، برنامه‌های توزیع شده هستند؟

- *مسئولیت هر یک از عناصر* چیست؟

عناصر چه کارهایی را انجام می‌دهند؟ وظیفه آنها در سیستم چیست؟

- *مفهوم هر یک از ارتباطات* چیست؟

منظور از ارتباطات این است که آیا عناصر با یکدیگر گفتگو دارند، یکدیگر را کنترل می‌کنند، به یکدیگر داده ارسال می‌کنند، از یکدیگر استفاده می‌کنند، دیگری را فراخوانی می‌کنند، با یکدیگر همزمان می‌شوند، اطلاعاتی مخفی را بین یکدیگر به اشتراک می‌گذارند و یا ترکیبی از این ارتباطات را دارند؟ مکانیزم برقراری ارتباط چیست؟ چه اطلاعاتی از طریق مکانیزم ارتباطی رد و بدل می‌شوند؟

- *مفهوم هر یک از طرح‌بندی‌ها*[1] چیست؟

چرا Control در بالاترین سطح قرار دارد؟ آیا بدین معنی است که عناصر دیگر را فراخوانی می‌کند و عناصر دیگر اجازه فراخوانی آن را ندارند؟ آیا سه عنصر سطح پائین‌تر وظیفه واحد پیاده‌سازی را بر عهده دارد؟

با توجه به اینکه نمی‌توان پاسخ این سئوالات در مورد سیستم ATM را از روی نمودار ۲-۱ جواب داد بنابراین نمودار مذکور یک معماری نیست. تنها چیزی که می‌توان در مورد آن گفت این است که یک نقطه شروع برای معماری را نشان می‌دهد. پس معماری نرم‌افزار به صورت زیر تعریف می‌شود.

«معماری نرم‌افزار یک برنامه یا سیستم محاسباتی، ساختار یا ساختارهای آن سیستم محاسباتی است که خصوصیات قابل رویت از بیرون عناصر و ارتباطات بین آنها را نشان می‌دهد.»

برای اینکه بتوان درک خوبی از این تعریف داشته مفاهیم درون آن را با جزئیات بیشتری بررسی می‌شود:

[1] layout

الف) منظور از خصوصیات "قابل رویت از بیرون" فرضیاتی هستند که عناصر مختلف می‌توانند از عنصر دیگر داشته باشند از قبیل سرویس‌های فراهم شده، کارائی، خصوصیات، رسیدگی به خطاها[1]، منابع مشترک استفاده شده.

ب) منظور از "معماری عناصر نرم‌افزاری را نشان می‌دهد" اینکه معماری در بردارنده اطلاعاتی پیرامون چگونگی اتصال عناصر به یکدیگر است. این بدین معنی است که معماری عناصری را که در این ارتباطات نقشی ندارند، نادیده می‌گیرد.

تقریبا در سیستم‌های امروزی عناصر از طریق واسط‌ها[2] با یکدیگر ارتباط برقرار می‌کند و این واسط‌ها به دو بخش عمومی و اختصاصی تقسیم می‌کنند. معماری مرتبط با بخش عمومی این تقسیم‌بندی است. بخش اختصاصی (آنهایی که باید به صورت داخلی پیاده‌سازی شوند) در معماری لحاظ نمی‌شود.

ج) تعریف معماری مشخص می‌کند که سیستم‌ها می‌توانند بیش از یک ساختار داشته باشند و اینکه یک ساختار به تنهایی نمی‌تواند یک معماری در نظر گرفته شود. برای مثال، تمام پروژهای متوسط[3] به واحدهای پیاده‌سازی تقسیم می‌شوند و هر یک از این واحدها وظایف خاص خود را دارند و خیلی اوقات مبنای تخصیص کار برای تیم برنامه‌نویسی هستند. این نوع از عناصر دربرگیرنده برنامه‌ها و داده‌هایی هستند که نرم‌افزار در جهت پیاده‌سازی واحدها، آنها را فراخوانی یا مورد دستیبابی قرار می‌دهد و همچنین شامل داده‌ها و برنامه‌های اختصاصی هستند. بنابراین مشخص است که باید چندین نوع ساختار در معماری وجود داشته باشد تا بتوان این موارد را مشخص و نشان داد.

این نوع بخش‌بندی سیستم به ساختارهای مختلف یک تقسیم‌بندی استاتیک است زیرا متمرکز بر روی روشی است که در آن عملکرد سیستم به بخش‌هایی تقسیم شده و آنها نیز به تیم‌های توسعه تخصیص داده می‌شوند در حالی که ساختارهای دیگر بیشتر بر روی نحوه تعامل عناصر با یکدیگر در زمان اجرا یا اجرای توابع سیستم متمرکز هستند. به عنوان مثال، فرض کنید سیستم به صورت یک مجموعه از پردازش‌های موازی ایجاد شده باشد. پردازش‌هایی که در زمان اجرا وجود دارند و ارتباطات همگام‌سازی بین پردازش‌ها نوعی دیگر از ساختار را تشکیل می‌دهد (پویا) که اغلب برای تشریح سیستم استفاده می‌شود.

[1] Fault handling
[2] Interfaces
[3] Nontrivial

حال سئوال اینکه آیا هر کدام از این ساختارها به تنهایی یک معماری است؟ جواب منفی است حتی با وجود اینکه هر کدام از آنها اطلاعات وابسته به معماری را به تصویر می‌کشاند، زیرا معماری متشکل از این نوع ساختارها بعلاوه ساختارهای دیگری است. این موضوع نشان می‌دهد با توجه به اینکه معماری می‌تواند بیش از یک نوع ساختار داشته باشد، بیش از یک نوع عنصر (به عنوان مثال پردازش‌ها و واحدهای پیاده‌سازی)، بیش از یک نوع ارتباط بین عناصر (زیرقسمت و همگام‌سازی) و حتی بیش از یک زمینه (زمان توسعه یا اجرا) نیز می‌تواند وجود داشته باشد.همچنین اینکه تعریف معماری مشخص نمی‌کند که عناصر و ارتباطات در معماری چه چیزهایی هستند. آیا عنصر نرم‌افزاری شیء است؟ یک پردازش است؟ یک کتابخانه است؟ یک پایگاه‌داده است؟ یک محصول تجاری است؟ در واقع یک عنصر نرم‌افزاری می‌تواند هر کدام از این موارد و حتی بیشتر از اینها نیز در نظر گرفته شود.

د) تعریف بر این موضوع اشاره می‌کند که هر سیستم محاسباتی با نرم‌افزار یک معماری نرم‌افزار دارد زیرا هر سیستم می‌تواند توسط عناصر و ارتباطات بین آنها نمایش داده شود. در بسیاری از موارد که سیستم کوچک یا کم اهمیت است، خودش یک عنصر منفرد است. علاوه بر این اگر چه هر سیستم یک معماری دارد ولی این موضوع بیان کننده این نیست که معماری بر هر سیستمی لازم است.

متاسفانه معماری یک سیستم می‌تواند مستقل از شرح و یا مشخصاتش باشد که این امر نشان‌دهندهٔ اهمیت مستندسازی معماری[1] و ساخت مجدد معماری[2] است.

ح) "رفتار[3] هر عنصر بخشی از معماری است" البته منظور رفتاری است که می‌تواند از منظر سایر عناصر مشاهده و تشخیص داده شود. یک چنین رفتاری اجازه می‌دهد عناصر با یکدیگر ارتباط برقرار کنند و کاملا واضح است که بخشی از معماری است. بیان این منظور دلیل دیگری است بر این موضوع که چرا یک سری ترسیم جعبه–خط[4] به عنوان یک معماری در نظر گرفته نمی‌شوند. بعلاوه رفتار یک عنصر چگونگی ایجاد عنصر دیگر جهت برقراری ارتباط با آن و یا قابل قبول بودن یک سیستم به صورت کلی را تحت تاثیر قرار می‌دهد بنابراین چنین رفتاری بخشی از معماری نرم‌افزار خواهد بود.

[1] Architecture documentation
[2] Architecture reconstruction
[3] Behavior
[4] Line-box

در نهایت تعریف نسبت به اینکه آیا معماری برای یک سیستم خوب است یا بد، توجهی ندارد. این بدین معنی است که معماری می‌تواند، سیستم را از رسیدن به خواسته‌های خود، کارائی و نیازمندیهای چرخه حیات هم بازداشته و هم اجازه دستیابی به آنها را فراهم نماید. نکته مهم در ارتباط با این امر این است که روش سعی و خطا روش مناسبی برای انتخاب معماری مناسب برای یک سیستم نیست (یعنی اینکه یک معماری به صورت تصادفی انتخاب شود، یک سیستم از آن ساخته شد و امید به خوب بودن آن داشته باشد). لذا این موضوع اهمیت ارزیابی معماری و طراحی معماری را بیشتر می‌کند.

۲-۲- تعاریف دیگر از معماری

معماری نرم‌افزار یک نظم در حال رشد ولی جوانی است لذا برای آن تعریف واحد و قابل قبولی وجود ندارد. از طرف دیگر کمبودهایی در تعاریف ارائه شده برای آن وجود ندارد. بسیاری از این تعاریف معمولا دارای عناوین سازگار در بدنه خود هستند (ساختار، عناصر و ارتباطات بین آنها) ولی آنها آنقدر از نظر جزئیات گسترده هستند که امکان تبادل‌پذیری مفاهیم بین آنها وجود ندارد.

معماری نرم‌افزار از اصول طراحی و کنش‌هائی که هنگام کار بر روی یک سیستم انجام می‌شود نشات گرفته شده است. معماری نرم‌افزار هدفش انتزاعی کردن دارائیهای ذاتی سیستم است و باید محدوده وسیعی از فعالیت‌ها، روش‌ها، شیوه‌ها و نتایج را در ماهیت خود داشته باشد.

در ادامه تعدادی تعریف برجسته برای معماری را ذکر خواهیم کرد و توضیحاتی پیرامون هر یک ارائه خواهیم داد تا دید کاملتری از معماری حاصل شود. این تعاریف عبارت‌اند از:

- *معماری طراحی سطح بالاست.*

ماهیتا این نکته کاملا صحیح است که معماری یک طراحی سطح بالا است ولی هر طراحی سطح بالا را نمی‌توان یک معماری در نظر گرفت. همچنین درست است که معماری دارای طراحی است ولی کلیه وظایف مرتبط با طراحی، مربوط به معماری نیستند. مانند تصمیم‌گیری در مورد ساختار داده‌های مهمی که محصورسازی خواهند شد. البته واسط مرتبط با آن ساختار داده‌ها قطعا یک دغدغه مربوط به معماری است اما انتخاب واقعی (ساختار داده‌ی داخلی)آنها مربوط به معماری نیست.

- *معماری ساختار کلی سیستم است.*

این موضوع دلالت بر این دارد که سیستم یک ساختار دارد و واضح است که این موضوع صحیح نیست. ساختارهای مختلف، نقاط مهندسی مهم کمکی[1] فراهم می‌کنند تا بتوان یک سیستم را با خصوصیات کیفی که منجر به موفقیت یا شکست آن سیستم می‌شود، درهم آمیخت.

- *معماری ساختار مولفه‌های یک برنامه یا سیستم، ارتباط بین آنها، و مفاهیم و راهنمایهای کنترل کننده طراحی و تکامل آنها در زمان است.*

این تعریف یکی از تعاریف متمرکز بر فرآیند است که دربرگیرنده اطلاعات ثانونه از قبیل اصول و راهنماها است. بسیاری ادعا دارند که معماری شامل یک سری از نیازهای ذینفعان و استدلالی در مورد چگونگی برآورده کردن آن نیازها است. البته درست است که جمع‌آوری این نوع اطلاعات ضروری است و ماهیت یک تکنیک خوب حرفه‌ای را دارد اما نباید آنها را مستقیما بخشی از معماری در نظر گرفت. هر سیستم یک معماری دارد که می‌تواند مستقل از هر دانشی پیرامون فرآیندی که معماری بوسیله آن طراحی شده یا تکامل یافته، کشف و تحلیل شود.

- *معماری مولفه‌ها و اتصالها است.*

اتصالها بر یک مکانیزم زمان اجرا برای انتقال کنترل و داده پیرامون یک سیستم دلالت می‌کنند. بنابراین تعریف فوق بر روی ساختارهای معماری زمان اجرا متمرکز است. این تعریف باعث می‌شود ساختارهای معماری غیر زمان اجرا نقش درجه دوم[2] را داشته باشند. در اصل آنها درجه دوم نیستند ولی نقش غیر ناچیزی در به موفقیت رسیدن اهداف یک سیستم دارند (وقتی صحبت از "ارتباطات" بین عناصر می‌شود منظور هم رابطه زمان اجرا و هم رابطه غیر زمان اجرا است).

۲-۳- الگوهای معماری، مدل مرجع و معماری مرجع

بین طرح‌های جعبه-خط که نقاط شروع ساده‌ای هستند و معماریهای تکامل یافته (شامل کلیه اطلاعات مرتبط با سیستم) گروهی از گامهای میانی وجود دارند. هر گام نتیجه یک مجموعه از تصمیمات[3] و انتخاب‌های[4]

[1] Critical engineering leverage points
[2] Second-class
[3] Decisions
[4] Choices

مربوط به معماری را ارائه می‌کند. بعضی از این گامهای میانی در جایگاه خود خیلی مفید هستند. حال اجازه دهید سه موضوع کلی درباره معماری را تشریح کنیم که نقش مهمی در آن ایفا می‌کنند.

۱. *یک الگو معماری توصیفی از عناصر و انواع ارتباط بین آنها به همراه مجموعه‌ای از اجبارها در مورد چگونگی استفاده از آنها است.*

یک الگو می‌تواند به صورت یک مجموعه از اجبارها بر روی معماری در نظر گرفته شود و آن اجبارها یک مجموعه یا خانواده‌ای از معماریها را تعریف می‌کند که آنها را ارضاء[1] می‌کنند. برای مثال سرویس دهنده–سرویس گیرنده[2] یک الگوی معماری معمولی است. سرویس دهنده و سرویس گیرنده دو نوع عنصر هستند و هماهنگی آنها بر حسب قراردادی است که سرویس دهنده برای برقرای ارتباط با هر یک از سرویس گیرندگان استفاده می‌کند.

استفاده از عبارت سرویس گیرنده–سرویس دهنده فقط بر این دلالت دارد که سرویس گیرنده چند تا است اما سرویس گیرندگان مشخص نیستند و بحثی در ارتباط با چگونگی عملکرد و پیاده‌سازی قراردادهای مختلف طراحی شده برای هر یک از سرویس دهندگان و یا سرویس دهنده وجود ندارد. معماریهای بیشماری از الگوی سرویس گیرنده–سرویس دهنده وجود دارند اما همگی آنها با یکدیگر تفاوت دارند. یک الگوی معماری یک معماری نیست بلکه دربرگیرنده اجبارهای کارا (مناسب) بر روی معماری و در نتیجه بر روی سیستم است.

یکی از پرکاربردترین جنبه‌های الگوها آن است که خصوصیات کیفی مشهور را نمایش می‌دهند. این دلیلی است که مشخص می‌کند چرا یک معمار بجای انتخاب تصادفی یک معماری باید الگوی خاص را انتخاب کند. بعضی از الگوها راه حل مشخصی را برای مشکلات کارائی، امنیت و قابل دسترس بودن بالا ارائه می‌کنند. انتخاب الگوی معماری اغلب نخستین انتخاب مهم طراحی یک معمار است.

۲. *یک مدل مرجع، تقسیم وظیفه‌مندی[3] به همراه جریان داده بین بخش‌ها است.*

یک مدل مرجع تجزیه استاندارد یک مسئله مشخص به بخش‌هایی است که به صورت اشتراکی مسئله را حل می‌کنند. تجربه نشان داده است که مدلهای مرجع مربوط به دامنه‌های تکامل‌یافته هستند.آیا

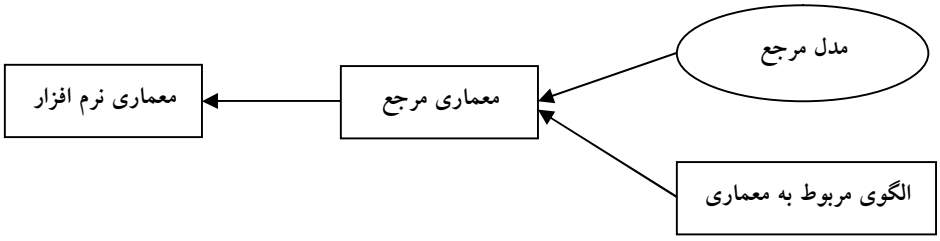
[1] Satisfy
[2] Client-Server
[3] Functionality

می‌توانید نام بخش‌های استاندارد یک کامپایلر یا یک سیستم مدیریت پایگاه داده را ذکر کنید؟ می‌توانید شرح دهید که چگونه بخش‌های مختلف آنها با یکدیگر در جهت تحقق اهدافشان همکاری می‌کنند؟ اگر جواب شما مثبت است بدین دلیل است که شما قبلا یک مدل مرجع از آن برنامه‌های کاربردی را یاد گرفته و در ذهن خود دارید.

۳. *یک معماری مرجع، مدل مرجع نگاشت داده شده به عناصر نرم‌افزاری (که به صورت اشتراکی وظیفه‌مندی مشخص شده در مدل مرجع را پیاده سازی می‌کنند) و جریانهای داده بین آنها است.*

از آنجائیکه مدل مرجع وظیفه‌مندی را تقسیم می‌کند معماری مرجع یک نگاشتی از آن وظیفه‌مندی به بخش‌های تجزیه شده سیستم است. نگاشت می‌تواند یک به یک باشد (اما ضرورتی ندارد). یک عنصر نرم‌افزاری می‌تواند قسمتی از یک وظیفه یا چندین وظیفه را پیاده‌سازی کند.

مدلهای مرجع، الگوهای معماری و معماریهای مرجع، معماری نیستند. آنها مفاهیم مفیدی هستند که عناصر یک معماری را بدست می‌آورند[1]. هر کدام از این مفاهیم نتیجه تصمیمات اولیه طراحی هستند. ارتباط بین این عناصر طراحی در شکل ۲-۲ به تصویر کشیده شده است.



شکل ۲-۲ : ارتباطات بین مدل‌های مرجع، الگوهای مربوط به معماری، معماریهای مرجع و معماریهای نرم‌افزار

[1] Capture

۲-۴- چرا معماری نرم‌افزار مهم است؟

سه دلیل اساسی برای اهمیت معماری نرم‌افزار وجود دارد که عبارتند از:.

۱. *ارتباط (مراوده) بین ذینفعان*

معماری نرم‌افزار یک انتزاع مشترک از سیستم ارائه می‌کند که بیشتر می‌تواند به عنوان پایه‌ای برای فهم دوجانبه، مذاکره، اجماع و ارتباط استفاده شود.

۲. *تصمیمات اولیه طراحی*

معماری نرم‌افزار اولین تصمیمات طراحی درباره یک سیستم را مشخص می‌کند و این الحاقهای اولیه هم خودشان و هم بخش‌های آتی فرآیند توسعه نرم‌افزار را تحت تاثیر قرار می‌دهند. همچنین معماری اولین نقطه‌ای در تصمیمات طراحی برای ساخت سیستم است که می‌تواند مورد تحلیل واقع شود.

۳. *انتزاع قابل انتقال از سیستم*

معماری نرم‌افزار یک مدل قابل فهم و نسبتا ساده ایجاد می‌کند تا به وسیله آن مشخص شود چگونه یک سیستم ساخته می‌شود و چگونه عناصر آن با یکدیگر کار می‌کنند. این مدل در سرتاسر سیستم قابل انتقال است. همچنین آن می‌تواند به دیگر سیستمهای که خصوصیات کیفیتی و نیازمندیهای وظیفه‌مندی یکسانی دارند اعمال شود و باعث افزایش قابلیت استفاده در مقیاس بزرگ شود.

۲-۴-۱- معماری وسیله‌ای برای ارتباط ذینفعان است

هر یک از ذینفعان سیستم –مشتری، کاربر، مدیر پروژه، برنامه‌نویس، آزمایش کننده و غیره– در ارتباط با خصوصیات متفاوت از سیستم هستند که به وسیله معماری تحت تاثیر قرار می‌گیرد. برای مثال، کاربر در ارتباط با این خصوصیت است که سیستم مطمئن و در صورت نیاز قابل دسترس باشد. مشتری در ارتباط با اینکه معماری در زمانبندی و بودجه مورد نظر پیاده‌سازی شود. مدیر نیز نگران است که آیا معماری اجازه خواهد داد تیمها به صورت مستقل کار کرده و در یک روش کنترل شده و منظم با یکدیگر ارتباط داشته باشند همچنین مدیر نگرانیهایی نیز در ارتباط با بودجه و زمانبندی دارد. معمار نیز در مورد استراتژیهایی نگران است که منجر به رسیدن به کلیه اهداف آنها می‌شود.

معمار یک زبان مشترک فراهم می‌کند که دغدغه‌های مختلف بتوانند از طریق آن بیان شده، با یکدیگر ارتباط داشته و حل شوند آن هم در سطحی که حتی برای سیستمهای پیچیده و بزرگ نیز قابل فهم باشد. بدون یک چنین زبانی فهم توانایی یک سیستم بزرگ به منظور اتحاذ تصمیمات اولیه‌ای که کیفیت و کارائی را تحت تاثیر قرار می‌دهد خیلی دشوار است.

۲-۴-۲- معماری مجموعه‌ای اولیه از تصمیمات طراحی را آشکار می‌کند

معماری نرم‌افزار مجموعه اولیه از تصمیمات طراحی سیستم را نشان می‌دهد. این تصمیمات اولیه از جمله تصمیماتی هستند که دستیابی صحیح به آنها سخت بوده و به ندرت در فرآیند توسعه تغییر پیدا می‌کنند و همچنین تاثیرات گسترده‌ای دارند.

- *معماری اجبارها[1] را روی پیاده‌سازی تعریف می‌کند*

یک پیاده‌سازی بیانگر یک معماری است اگر آن با تصمیمات طراحی مرتبط با ساختار مشخص شده به وسیله معماری مطابقت داشته باشد این بدین معنی است که پیاده‌سازی باید به عناصری تقسیم شود که قبلا در معماری مشخص شده است. این عناصر باید در یک روش مشخص شده در معماری با یکدیگر ارتباط داشته باشند و هر عنصر باید مسولیتی که در قبال سایر عناصر دارد مطابق آنچه که در معماری مشخص شده را انجام دهد.

تصمیمات تخصیص منابع همچنین می‌توانند پیاده‌سازی را محدود می‌کنند. اجبارها (محدودیت‌ها) یک نوع جداسازی دغدغه‌ها[2] را ممکن می‌سازند که اجازه می‌دهد مدیران بهترین تصمیمات را در استفاده از پرسنل و ظرفیت عملیاتی اتخاذ کنند.

- *معماری ساختار سازمانی را دیکته[3] می‌کند*

نه تنها معماری ساختار سیستمی را مشخص می‌کند که باید توسعه داده شود بلکه ساختار معماری، ترسیم کننده ساختار پروژه در حال توسعه نیز هست. روش معمولی برای تقسیم کار در یک سیستم بزرگ تخصیص کارهای متفاوت از سیستم به گروهای مختلف برای تحقق آن دسته از کارها است. این امر "ساختار تفکیک کار

[1] Constraint
[2] Separation of concerns
[3] Dictate

سیستم[1]" نامیده می‌شود. به دلیل اینکه معماری سیستم در برگیرنده تجزیه سطح بالا از سیستم است، معمولا به عنوان پایه‌ی ساختار تفکیک کار نیز در نظر گرفته می‌شود.

اما نکته مهم درباره معماری این است که وقتی بر روی یک معماری توافق شد دیگر تغییر آن به دلایل حرفه‌ای و مدیرتی دشوار خواهد بود و این همان دلیلی است که باعث می‌شود قبل از اینکه یک معماری برای یک سیستم قطعی شود ارزیابی جامعی از آن صورت بپذیرد.

– *معماری خصوصیات کیفی سیستم را فراهم کرده یا مانع آن می‌شود*

اینکه یک سیستم قادر خواهد بود خصوصیات کیفی مطلوب خودش را دارا باشد اساسا توسط معماری آن مشخص می‌شود. به منظور درک بهتر، موارد زیر را در نظر بگیرید:

- اگر سیستم شما نیاز به کارائی بالا داشته باشد ضروری است که رفتارهای مبتنی بر زمان عناصر و فرکانس و حجم ارتباطات بین عناصر را مدیریت کنید.
- اگر قابلیت اصلاح مهم است نیاز دارید که تخصیص مسئولیت‌ها به عناصر را طوری انجام دهید که تغییر در سیستم پیامدهای گسترده و زیادی نداشته باشد.
- اگر امنیت بالائی مد نظر باشد لازم است که ارتباط بین عناصر را و اینکه کدام عناصر به کدام اطلاعات دسترسی داشته باشند را مدیریت و محافظت کنید. همچنین ممکن است عناصر سفارشی‌شده جدیدی (مانند هسته مطمئن) به معماری معرفی کنید.
- اگر شما معتقد هستید که مقیاس‌پذیری در سیستم مورد نیاز است مجبور هستید دقت استفاده از منابع را مشخص کنید تا معرفی جایگزینی با ظرفیت بالا[2] را تسهیل ببخشید.
- اگر پروژه شما نیاز دارد به صورت تدریجی زیرمجموعه‌ای از سیستم را تحویل دهد لازم است به دقت، استفاده مولفه‌ها از یکدیگر (ارتباط بین مولفه‌ها) را مدیریت کنید.
- اگر شما می‌خواهید که عناصر سیستم در دیگر سیستمها نیز قابل استفاد باشند نیاز دارید که وابستگی بین عناصر را محدود کنید این امر در جهت اینکه وقتی یک عنصر را استخراج کردید آن عنصر به

[1] The work breakdown structure of a system
[2] Higher-capacity replacements

میزان زیادی وابستگی به محیطی خود نداشته باشد در واقع پارامترهای تاثیرگذار بر عناصری که وابسته به محیط هستند تا حد ممکن محدود شوند.

راهبردهای مربوط به این نوع خصوصیات کیفی و سایر خصوصیات کیفی کاملا مرتبط به معماری هستند. اما مهم است که بدانید معماری به تنهائی نمی‌تواند عملکرد و کیفیت را تضمین کند. البته تصمیمات پیاده‌سازی ضعیف همیشه می‌تواند یک طراحی معماری مناسب را خدشه دار کند. تصمیمات در کلیه مراحل چرخه حیات –از طراحی سطح بالا تا کدنویسی و پیاده‌سازی– کیفیت سیستم را تحت تاثیر قرار می‌دهند. بنابراین کیفیت همواره نشان دهنده عملکرد صحیحی از طراحی معماری نیست. برای تضمین کیفیت، یک معماری خوب لازم است ولی معماری به تنهائی آن کافی نیست.

– *پیش‌بینی سیستم از طریق مطالعه معماری*

در معماری این امر امکان‌پذیر است که گفته شود یک تصمیم معماری (خصوصیات کیفی) قبل از اینکه سیستم توسعه یافته و استقرار پیدا کند قابل دستیابی هستند یا نه؟ اگر پاسخ منفی بود باید یک معماری مناسب که آن خصوصیات مورد نظر را تحقق می‌بخشد انتخاب شود (انتخاب تصادفی یک معماری یا روش منجر به چنین حالتی می‌شود). همچنین امکان دارد که کیفیت یک سیستم فقط مبتنی بر ارزیابی معماری آن پیش‌بینی شود. تکنیک‌های ارزیابی معماری همچون ATAM یک چنین قابلیتی را فراهم می‌کند.

– *معماری امکان استدلال درباره تغییرات و مدیریت تغییرات را آسان می‌کند*

موسسه مهندسی نرم‌افزار به این حقیقت دست یافته است که ۸۰ درصد از هزینه عمومی سیستم نرم‌افزاری بعد از اولین استقرار[1] اتفاق می‌افتد. یک نتیجه از این آمار آن است که بسیاری از سیستم‌ها بر روی این مرحله یعنی معماری در حال فعالیت توسعه هستند.

هر معماری تغییرات ممکن را به سه دسته محلی، غیرمحلی و معماری تقسیم‌بندی می‌کند. یک تغییر محلی می‌تواند به وسیله تغییر یک عنصر منفرد انجام شود. یک تغییر غیرمحلی به تغییر چندین عنصر نیاز دارد ولی ساختارهای پایه‌ای معماری را دست نخورده باقی می‌گذارد (یعنی ساختار اصلی تغییر پیدا نمی‌کند). یک تغییر معماری روشهای اصلی که عناصر با یکدیگر ارتباط برقرار می‌کنند و الگوی معماری را تحت تاثیر قرار می‌دهد و احتمالا منجر به تغییرات کلی بر روی سیستم خواهد شد. مشخص است که تغییر محلی مطلوب‌ترین تغییر

[1] Deployment

است و همچنین یک معماری کارامد معماری است که احتمالی ترین تغییرات در آن آسانترین تغییرات قابل اعمال باشند.

– *معماری به نمونه‌سازی اکتشافی کمک می‌کند*

به محض اینکه معماری تعریف شد می‌تواند مورد تحلیل قرار گرفته و به عنوان یک اسکلت از سیستم نمونه‌سازی شود. این عمل به دو صورت به فرآیند توسعه کمک می‌کند.

۱. سیستم در ابتدای چرخه حیات محصول قابل اجرا است.
۲. یک حالت خاص از داشتن یک سیستم قابل اجرای اولیه آن است که مشکلات کارائی بالقوه می‌تواند در ابتدای چرخه حیات محصول شناسائی شود.

هر کدام از این مزیت‌ها میزان خطر در پروژه را کاهش می‌دهد. زمانی که معماری بخشی از یک خانواده از سیستمهای مرتبط باشد هزینه ایجاد یک چارچوب برای نمونه‌سازی می‌تواند بر روی توسعه سیستم‌های مختلف پخش شود.

– *معماری تخمین زمانبندی و هزینه را دقیق‌تر می‌کند*

تخمین هزینه و زمانبندی ابزار مدیریتی مهمی هستند که مدیر را قادر می‌سازند منابع ضروری را بدست آورده، و درک کند چه موقع پروژه مشکل دارد. تخمین هزینه، مبتنی بر درک بخش‌های سیستم است و فهم اینکه ماهیت هر بخش چیست، به عبارت دقیق‌تر مبتنی بر کل دانش سیستم است.

نکته مهم درباره تخمین این است که مدیر پروژه تنها زمانی قادر است تخمین دقیق‌تری از بخش‌های پروژه داشته باشد که آنها را به خوبی بشناسد. همچنین اینکه تعریف اولیه از معماری بدین معنی است که نیازمندیها برای یک سیستم بررسی و در بعضی حالات اعتبارسنجی نیز شده‌اند. هر چقدر دانش درباره یک سیستم بیشتر باشد میزان دقت تخمین نیز بیشتر خواهد بود.

۲-۴-۳- معماری به عنوان یک مدل قابل استفاده مجدد و قابل انتقال

قبلا اعمال استفاده مجدد در چرخه حیات توسعه نرم‌افزار بزرگترین مزیتی بود که می‌توانست حاصل شود. در حالی که استفاده مجدد در سطح کد سودمند است استفاده مجدد در سطح معماری اهرم بزرگی برای

سیستم‌هایی با نیازمندیهای مشابه فراهم می‌کند. نه فقط کد می‌تواند قابل استفاده مجدد باشد بلکه حتی نیازمندیهای موجود در معماری در مراحل اولیه نیز می‌تواند قابل استفاده مجدد باشد.

- معماری خط تولید یک معماری مشترکی به اشتراک می‌گذارد

معماری خط تولید یک مجموعه از سیستمهای شدیدا نرم‌افزاری است که یک مجموعه مشترک و مدیریت شده از ویژگیهایی که یک هدف خاص یا نیازهای یک بخش بخصوص تجاری را برطرف می‌کنند به اشتراک می‌گذارد و از یک مجموعه مشترک از دارائی‌های هسته[1] توسعه داده می‌شود. مهمترین این دارائی‌های هسته، معماری است که به منظور مدیریت نیازهای کل این مجموعه از سیستم‌ها طراحی شده است. معمارهای خط تولید یک معماری را انتخاب می‌کنند (یا یک خانواده از معماریهای خیلی نزدیک به یکدیگر) که اهداف کلیه اعضای خط تولید را به وسیله ایجاد تصمیمات اولیه معماری و همچنین سایر تصمیمات معماری که فقط به عضوهای خاص اعمال خواهد شد، برآورده کند. معماری مشخص می‌کند که چه چیزی برای کلیه اعضا خط تولید ثابت است و چه چیزیی متغییر می‌باشد. معماری خط تولید یک روش قدرتمند برای توسعه سیستمهایی فراهم می‌کند که مرتبه بزرگی از تلاشها را در زمان ورود به بازار، هزینه، بهره‌وری و کیفیت محصول دارند.

- معماری می‌تواند با استفاده از عناصر بزرگ و توسعه یافته خارجی ساخته شود

از آنجائیکه روشهای پیشین توسعه نرم‌افزار بر روی برنامه‌نویسی به عنوان فعالیت اصلی متمرکز هستند و پیشرفت کار را با معیارهای همچون تعداد خطوط کد می‌سنجند توسعه مبتنی بر معماری اغلب بر روی ترکیب و سوار کردن عناصر که بیشتر تمایل دارند به صورت جدا و حتی مستقل از سایر عناصر توسعه یابند، متمرکز است.

یک جنبه کلیدی از معماری سازماندهی ساختار عناصر، واسط‌ها و مفاهیم عملیاتی است. مهمترین اصل از این سازماندهی قابلیت تعویض‌پذیری[2] است. مولفه‌های آماده تجاری[3]، زیر سیستم‌ها و واسط‌های ارتباطی سازگار همگی مبتنی بر اصل قابلیت تعویض‌پذیری هستند. اما نکات زیادی در مورد توسعه نرم‌افزار از طریق ترکیب وجود دارد که هنوز بدون راه حل هستند. زمانی که مولفه‌های انتخابی برای زیرسیستم‌های مجزا مطرح

[1] Core assets
[2] Interchangeability
[3] Commercial off-the-shelf components

هستند و آن زیر سیستم‌ها با فرضیات معماری تداخل دارند تلاشهای مورد نیاز برای یکپارچه کردن وظایف زیرسیستم‌ها می‌تواند خیلی زیاد باشد.

- *محدود کردن انتخاب‌های مختلف طراحی*

اگر چه برنامه‌های کامپیوتری می‌توانند در روشهای مختلفی با یکدیگر ترکیب شوند ولی دست‌یافته‌هایی وجود دارد که به وسیله محدود کردن انتخاب‌های ممکن برای ارتباط و مشارکت برنامه‌ها با یکدیگر بدست می‌آیند. در واقع هدف از بیان این موضوع اینکه باید پیچیدگی طراحی سیستمی را که در حال ایجاد آن هستیم به حداقل برسانیم. مزیت‌های این عمل افزایش استفاده مجدد، طراحی‌های ساده و مرتب که خیلی قابل فهم هستند، توانائی تحلیل بالا، زمان انتخاب کوتاه و قابلیت تعویض پذیری بالا، هستند. خصوصیت‌های طراحی نرم‌افزار مبتنی بر انتخاب الگوی معماری است. الگوی انتخابی که برای یک مسئله خاص مناسب باشد باید پیاده‌سازی راه حل طراحی را نیز بهبود دهد.

- *معماری توسعه مبتنی بر قالب*[1] *را اجازه می‌دهد*

یک معماری تصمیمات طراحی درباره چگونگی ارتباط عناصر را در بر دارد که می‌تواند این تصمیمات جمع شده و یک مرتبه معماری نوشته شود. قالب‌ها می‌توانند برای دست یابی به مکانیزم ارتباطی بین عناصر مورد استفاده قرار بگیرند. برای نمونه، یک قالب می‌تواند اعلان‌ها را برای بخش عمومی یک عنصر کد کند در حالی که نتایج دست نخورده باقی می‌مانند یا اینکه قراردادی که عناصر برای ارتباط با سیستم اجرایی استفاده می‌کنند را کد کند.

- *یک معماری می‌تواند پایه‌ای برای آموزش باشد*

معماری شامل توصیفی از چگونگی ارتباط عناصر در جهت برآورده کردن رفتارهای مورد نیاز سیستم است که می‌تواند برای معرفی سیستم به اعضا جدید پروژه مورد استفاده قرار بگیرد. این عملکرد معماری، نگاه در مورد اینکه یکی از موارد استفاده مهم معماری نرم‌افزار حمایت از ارتباط بین ذینفعان مختلف است را تقویت می‌کند.

[1] Template-Based

۲-۵- معماری سیستم در مقابل معماری نرم‌افزار

سئوال این است که چرا درباره معماری نرم‌افزار صحبت می‌شود؟ مگر معماری سیستم مهم نیست؟ چه تفاوتی بین معماری نرم‌افزار و معماری سیستم وجود دارد؟ در حقیقت، تفاوت‌های خیلی کمی بین این دو مفهوم وجود دارد. ولی اینکه چرا بیشتر در مورد معماری نرم‌افزار صحبت می‌شود دلیلش اینکه خواهان آن هستیم که بر ماهیت حساس بودن تصمیمات نرم‌افزاری تاکید شود که یک معمار در مورد کلیه جنبه‌های کیفیت یک محصول اتخاذ می‌کند.

در ایجاد یک معماری نرم‌افزار ملاحضات سیستم نادیده گرفته می‌شود. برای مثال، اگر خواهان یک سیستم با کارائی بالا هستید نیاز است که ایده‌هایی در مورد خصوصیات فیزیکی سکوی سخت افزاری که سیستم در روی آن اجرا خواهد شد (سرعت پردازنده، سرعت دسترسی به دیسک، مقدار حافظه) و همچنین مشخصات دستگاهی که سیستم با آن در ارتباط است (سنسورها، دستگاه‌های ورودی/خروجی، فعال کننده‌ها) داشته باشید و علاوه بر اینها باید خصوصیات شبکه را نیز در نظر گرفته باشید (پهنای باند). اگر شما نیاز به سیستم با قابلیت اطمینان بالائی داشته باشید دوباره با دغدغه‌های سخت افزاری مانند نرخ خرابی، قابل دسترس بودن دستگاه‌ها و پردازنده‌های اضافی سروکار دارید ولی در بقیه مسائل مربوط به معماری نرم‌افزار ذهن معمار از کلیه ملاحضات سخت افزاری دور است.

بنابراین وقتی شما یک معماری نرم‌افزار طراحی می‌کنید احتمالا نیاز خواهید داشت که درباره کل سیستم فکر کنید یعنی هم سخت افزار و نرم‌افزار. با این وجود هنوز تاکید بر روی معماری نرم‌افزار به جای معماری سیستم است. حال دوباره سئوال این است که چرا؟ زیرا بسیاری از آزادی عمل‌های یک معمار متعلق به انتخاب‌های نرم‌افزار است نه انتخاب‌های سخت افزار. البته این صحیح نیست که انتخاب‌های سخت افزاری وجود ندارد اما این ممکن است خارج از انتخاب‌های یک معمار باشد. به همین دلیل بر روی بخش نرم‌افزاری معماری متمرکز می‌شویم زیرا جائی است که تصمیمات اساسی ایجاد می‌شود، جائی است که بیشترین آزادی عمل در آن وجود دارد و البته جائی است که گزینه‌های زیادی برای موفقیت سیستم در آن قابل کشف و تعریف کردن است.

۲-۶- دیدگاه‌ها و ساختارهای معماری

متخصص پوست، خون و عصب هر کدام دید متفاوتی از ساختار بدن انسان دارند. چشم پزشک و متخصص قلب بر روی زیربخش‌های بدن متمرکز هستند و روان‌شناس نیز با رفتارهای انسان سروکار دارد. اگر چه این دیدها به شکل متفاوت ترسیم کننده ماهیت بدن انسان می‌شوند و خصوصیات متفاوتی دارند اما ذاتا به هم مرتبط هستند. این دیدهای مختلف با یکدیگر معماری بدن انسان را نشان می‌دهند. در نرم‌افزار نیز چنین حالتی وجود دارد. یعنی اینکه به دلیل پیچیدگی موجود در ماهیت نرم‌افزار، در هر لحظه باید بر یکی از ساختارهای سیستم نرم‌افزاری متمرکز شویم. در ارائه معماری از دو مفهوم مرتبط با هم به نامهای دیدگاه[1] و ساختار[2] استفاده می‌شود که تعریف هر یک به شرح زیر است.

- **دیدگاه**

نمایشی از یک مجموعه‌ای منسجم از عناصر معماری است که به وسیله ذینفعان سیستم (معمار) ایجاد شده و خوانده می‌شود.

- **ساختار**

مجموعه‌ای از عناصر و ارتباط بین آنها است. در یک نوع ساختار فقط عناصر متعلق به آن نوع ساختار لحاظ می‌شود.

برای مثال، یک ساختار ماژول مجموعه از ماژول‌های سیستم و نحوه سازماندهی آن ماژول‌ها است. یک دید ماژول نمایشی از ساختارها است که به وسیله ذینفعان مستند شده و به وسیله ذینفعان سیستم استفاده می‌شود. ساختارهای معماری مبتنی بر گستردگی عناصری که نشان می‌دهند به سه گروه بزرگ تقسیم می‌شوند که عبارتند از :

- *ساختارهای ماژول*[3]

عناصر در این ساختار ماژول‌ها هستند که واحدهایی از پیاده‌سازی می‌باشند. ماژول‌ها یک روش مبتنی بر کد برای بررسی سیستم ارائه می‌کنند و بر بخش‌های وظیفه‌مندی سیستم تاکید دارند. این نوع

[1] view
[2] Structure
[3] Module structures

ساختار تاکید کمتری بر روی چگونگی آشکار شدن نتایج در زمان اجرا دارد. ساختار ماژول اجازه پاسخ‌دهی به سئوالاتی را می‌دهد که عبارتند از: وظیفه‌مندی اصلی تخصیص داده شده به هر یک از ماژول‌ها چیست؟ چه عناصر نرم‌افزاری اجازه استفاده شدن در ماژول را دارند؟ چه ماژول‌های با استفاده از رابطه عموی‌سازی و خصوصی‌سازی با ماژول‌های دیگر ارتباط دارند؟

- *ساختارهای مولفه و اتصال*[1]

عناصر در این ساختار مولفه‌های زمان اجرا و اتصالها هستند. ساختار مولفه و اتصال به پاسخ‌گویی به سئوالات از قبیل مولفه‌های اجرائی اصلی کدام هستند و چگونه تعامل برقرار می‌کنند؟ مخزن داده مشترک اصلی چه چیزی است؟ کدام بخش از سیستم تکرار شده است؟ چگونه داده‌ها در سرتاسر سیستم حرکت می‌کنند؟ کدام بخش‌های سیستم می‌توانند به صورت موازی اجرا شوند؟ چگونه می‌توان ساختار سیستم را زمانی که در حال اجرا است تغییر داد؟ کمک می‌کند.

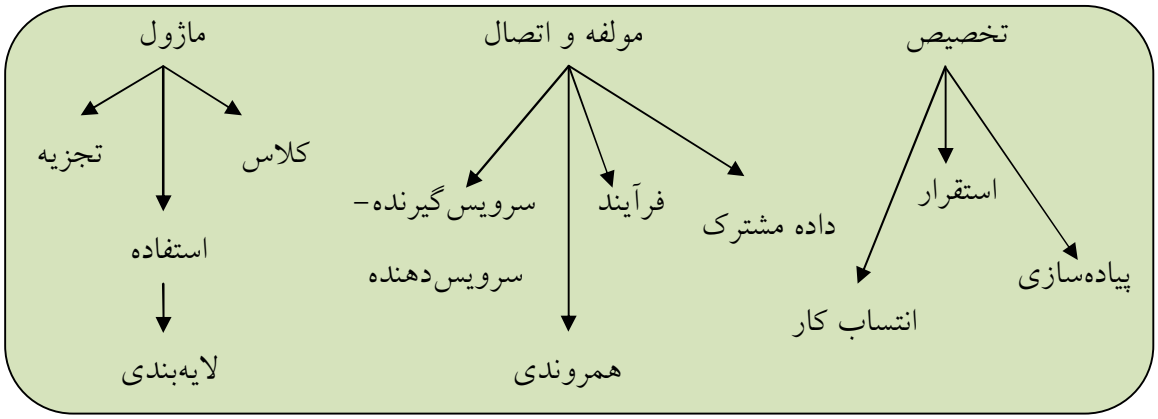
- *ساختارهای تخصیص*[2]

ساختار تخصیص ارتباط بین عناصر نرم‌افزاری و عناصر متعلق به یک یا چند محیط خارجی[3] را نشان می‌دهد که سیستم در آن ایجاد و اجرا شده است. این ساختار به سئوالاتی از این دست پاسخ می‌دهد: هر پردازنده چه عناصر نرم‌افزاری را اجرا می‌کند؟ در طول توسعه، آزمایش و ساخت سیستم هر عنصر در چه فایهایی ذخیره شده‌اند؟ چه عناصر نرم‌افزاری به چه تیم‌های توسعه تخصیص داده شده‌اند؟

۲-۶-۱- ساختارهای نرم‌افزار

دسته‌بندی ساختارهای نرم‌افزار در شکل ۲-۳ نمایش داده شده است که توضیحات پیرامون هر یک از این دسته‌بندی‌ها و ساختارهای درون آن در ادامه بیان خواهد شد.

[1] Component-and-connector structures
[2] Allocation structures
[3] External environments



شکل ۲-۳ : ساختارهای عمومی معماری نرم‌افزار

۲-۶-۱-۱- ماژول

ساختارهای مبتنی بر ماژول به شرح زیر هستند:

- تجزیه[1]

واحدها، ماژول‌هایی هستند که به وسیله رابطه "یک زیر ماژول است از" به ماژول‌های دیگر مرتبط می‌شوند که این موضوع نشان می‌دهد چگونه ماژول‌های بزرگتر به ماژول‌های کوچکتر تقسیم می‌شوند. ماژول‌ها اغلب فراورده‌های یکسانی دارند (مانند مشخصات ماژول، کد، طرح آزمایش و غیره). ساختار تجزیه قابلیت اصلاح بزرگی در سیستم به وسیله تضمین اینکه احتمالا تغییرات در داخل یک تعداد ماژول خواهد بود، ایجاد می‌کند. همچنین از آن به عنوان پایه‌ای برای ایجاد سازماندهی از پروژه شامل ساختار مستندسازی، یکپارچگی و طرح‌های آزمایش استفاده می‌شود.

- *استفاده*[2]

واحدهای این ساختار ماژول‌ها، رویه‌ها و منابع رابط‌های ماژول‌ها هستند. واحدها از طریق رابطه "استفاده می‌کند از" به واحدهای دیگر مرتبط می‌شوند و منظور این است که صحت واحدی که از واحد دیگر استفاده می‌کند به صحت واحد استفاده شده بستگی دارد. این ساختار به مهندس سیستم اجازه می‌دهد که به راحتی قابلیت جدید را به سیستم اضافه کند که این امر یک توسعه تدریجی را امکان‌پذیر می‌کند.

[1] Decomposition
[2] Uses

- لایه[1]

یک لایه مجموعه منسجم از وظیفه‌مندیهای یکسان را نشان می‌دهد. در واقع اگر رابطه *استفاده* را بتوان در یک روش مناسب مورد استفاده قرار داد به یک لایه‌بندی دست پیدا خواهیم کرد. در این ساختار هر لایه (مثلا nام) به سرویس‌های لایه پائین‌تر از خود (لایه n-1ام) دسترسی دارد. همچنین لایه‌ها به صورت انتزاعی (ماشین مجازی) طراحی می‌شوند که این امر باعث می‌شود مشخصات لایه پائین از دید لایه بالاتر پنهان باشد.

- *کلاس یا تعمیم*[2]

واحدها در این ساختار کلاس‌ها هستند که توسط رابطه "ارث‌بری می‌کند از[3]" یا "یک نمونه‌ای است از[4]" با یکدیگر ارتباط برقرار می‌کنند. این نوع ساختار از استدلال درباره رفتارهای مشترک و مشخص کردن تفاوت‌ها بین کلاس‌های مختلف حمایت می‌کند. همچنین ساختار کلاس اجازه می‌دهد که درباره استفاده مجدد و افزایش تدریجی تصمیم‌گیری شود.

۲-۶-۱-۲- مولفه و اتصال

ساختارهای مبتنی بر مولفه و اتصال به شرح زیر هستند:

- فرآیند یا فرآیندهای ارتباطی

واحدها، فرآیندها یا نخ‌ها هستند که به وسیله ارتباط، همزمانی و یا عملیات انحصاری به واحدهای دیگر متصل می‌شوند. این نوع ساختار با جنبه‌های اجرائی سیستم در ارتباط می‌باشد. ارتباط در این نوع ساختار "الصاق[5]" است که نشان می‌دهد چگونه اتصال‌ها و مولفه‌ها یکدیگر را کنترل می‌کنند، ساختار فرآیند در زمینه بررسی (اجرا) کارائی و قابلیت دسترسی یک سیستم مورد استفاده قرار می‌گیرد.

- همزمانی

ساختار همزمانی به تشخیص فرصت‌های موازی‌سازی و به اشتراک‌گذاری منابع کمک می‌کند. واحدهای این ساختار مولفه‌ها می‌باشند و اتصالها از نوع رشته‌های منطقی هستند. یک نخ منطقی

[1] Layered
[2] Class or generalization
[3] inherits-from
[4] is-an-instance-of
[5] Attachment

مجموعه‌ای از محاسبات پشت سر هم است که می‌تواند به نخی فیزیکی جداگانه در فرآیند طراحی تخصیص داده شود. ساختار همزمانی باعث می‌شود که نیازمندیهای اجرای همزمان شناسائی شوند.

- مخزن یا داده‌های مشترک

این ساختار دربرگیرنده مولفه‌ها و اتصالهایی است که برای ایجاد، ذخیره و دسترسی به داده‌های مانا مورد استفاده قرار می‌گیرند. این ساختار نشان می‌دهد که داده‌ها چگونه تولید و مصرف می‌شوند و همچنین به تضمین کارائی و یکپارچگی داده‌ها کمک می‌کند.

- سرویس‌گیرنده–سرویس‌دهنده

ساختار سرویس‌گیرنده–سرویس‌دهنده برای تشریح سیستم‌هایی مناسب است که در آن تعدادی سرویس‌دهنده و سرویس‌گیرنده با یکدیگر جهت دستیابی به اهداف سیستم مشارکت می‌کنند. واحدها در این ساختار سرویس‌گیرنده‌ها و سرویس‌دهنده‌ها هستند و اتصال‌ها قراردادهایی هستند که این واحدها از آن برای برقراری ارتباط و انجام کار سیستم استفاده می‌کنند. کاربرد این ساختار در جداسازی دغدغه‌ها برای توزیع‌پذیری فیزیکی و تنظیم بار کاری سیستم است.

۲-۶-۱-۳- تخصیص

ساختارهای مبتنی بر تخصیص به شرح زیر هستند:

- استقرار[1]

عناصر ساختار استقرار موجودیت‌های نرم‌افزاری، سخت افزاری و مسیرهای ارتباطی هستند که نشان می‌دهند چگونه نرم‌افزار به عناصر ارتباطی و پردازنده‌ها انتساب شده است. در این ساختار، رابطه‌ی "تخصیص داده شده به" نشان می‌دهد واحدهای فیزیکی عناصر نرم‌افزاری به چه چیزی تخصیص داده شده‌اند و رابطه "منتقل شده به" نیز برای تخصیص پویا استفاده می‌شود. این نوع ساختار به مهندس سیستم کمک می‌کند که پیرامون کارائی، یکپارچگی داده‌ها، قابلیت دسترس‌پذیری و امنیت سیستم استدلال‌های لازم را انجام دهد. ساختار استقرار یکی از ساختارهای جالب در مورد سیستمهای توزیع‌شده و موازی است.

[1] Deployment

- پیاده‌سازی[1]

این ساختار نشان می‌دهد چگونه عناصر نرم‌افزاری به ساختارهای فایل در طی فرآیند توسعه، به منظور فراهم آوردن یکپارچگی یا محیط کنترل پیکربندی سیستم نگاشت شده‌اند. این ساختار برای مدیریت فعالیت‌های توسعه و ساخت فرآیندها، حیاتی است.

- انتساب کار[2]

این ساختار برای تخصیص فعالیت‌های پیاده‌سازی و یکپارچگی به تیم‌های توسعه مناسب است. ساختار انتساب کار به کنترل عملکرد افراد کمک می‌کند و حتی مشخص می‌کند که تجارب مورد نیاز هر تیم چه چیزی باید باشد تا بتواند کار را خوب تحویل دهند. همچنین برای پروژه‌های بزرگ توزیع شده که کار به تیم‌های مختلف واگذار می‌شود این ساختار کاربرد مناسبی دارد.

در جدول ۲-۱ یک دید کلی و مختصر شده از ساختارهای سیستم نمایش داده شده است.

۲-۶-۲- ساختارهای مرتبط با یکدیگر

هر کدام از ساختارها عنوان شده، دیدهای مختلفی از سیستم ارائه می‌کنند و هر کدام در جایگاه خود مهم و مفید هستند. اگر چه این ساختارها دیدهای مختلفی از سیستم را نشان می‌دهد ولی در اصل آنها مستقل نیستند زیرا عناصر هر یک به عناصر دیگری وابسته است. به عنوان مثال یک ماژول در ساختار تجزیه ممکن است به یک یا چند عنصر ساختار مولفه و اتصال تبدیل شود. بنابراین نگاشت مختلفی می‌تواند بین ساختارها وجود داشته باشد. نکته مهم اینکه در تمام سیستم‌ها نیاز نیست که از کلیه ساختارها استفاده شود برای سیستم‌های کوچک انتخاب یک یا چند ساختار می‌تواند جوابگو باشد زیرا این ساختارها می‌توانند دربرگیرنده ساختارهای دیگر نیز باشند. در واقع در این نوع سیستم‌ها بعضی از ساختارها در صورت طراحی مانند هم خواهند شد لذا طراحی یک ساختار کفایت می‌کند. ولی برای سیستم‌های بزرگ باید تحلیل کافی صورت گرفته و بر حسب آنها ساختارهای مناسب انتخاب شوند. در سیستم‌های بزرگ هر ساختار ممکن است آنقدر بزرگ باشد که طراح مجبور باشد آنها را به زیرساختارهایی تبدیل کند. بنابراین انتخاب نوع و تعداد ساختارهای موجود در یک معماری رابطه مستقیم با ماهیت سیستم نرم‌افزاری و حجم و گستردگی آن دارد.

[1] Implementation
[2] Work assignment

۲-۶-۳- معماری ۱+۴

در سال ۱۹۹۵ آقای Kruchten در مقاله‌ای بیان کرد که هیچ کدام از ساختارها با یکدیگر تناقضی ندارند و همه آنها در جهت برطرف کردن نیازمندیهای سیستم هستند ولی بهتر است از موارد کاربری مهم به منظور انتخاب ساختارها استفاده شود. ایشان دیدگاهی پیرامون این موضوع مطرح کردند و آن را چهار بعلاوه یک نامید. دیدهای روش ۱+۴ به شرح زیر هستند:

- *منطقی*

شامل کلاسها و اشیاء است که در شیءگرائی جهت دستیابی به حداکثر تجرید استفاده می‌شود. در واقع همان ساختار ماژول است.

- *فرآیند*

نحوه همزمانی و توزیع وظیفه‌مندیها را نشان می‌دهد (ساختار مولفه‌ها و اتصال‌ها).

- *توسعه*

سازمان‌دهی ماژول‌ها، کتابخانه‌ها، زیرسیستم‌ها و واحدهای توسعه را نمایش می‌دهد در واقع ساختار تخصیص را نمایش می‌دهد.

- *فیزیکی*

نحوه نگاشت اجزا را به پردازنده‌ها و گره‌های ارتباطی مشخص می‌کند. این ساختار مشابه ساختار تخصیص است اما بیشتر جنبه استقرار مد نظر آن است.

- *سناریو*

به خودی خو یک دید نیست در واقع همان موارد کاربری است.

جدول ۲-۱: ساختارهای معماری سیستم

ساختار نرم‌افزار	رابطه‌ها	استفاده می‌شود برای
تجزیه	Is a submodule of; shares secret with	اختصاص منابع و ساختاردهی پروژه و برنامه‌ریزی، مخفی‌سازی اطلاعات، محصورسازی، کنترل پیکربندی
استفاده	Requires the correct presence of	مهندسی زیرمجموعه‌ها و گسترش‌ها
لایه	Requires the correct presence of; uses the services of; provides abstraction to	توسعه تدریجی، پیاده‌سازی سیستم‌ها بر پایه مفهوم ماشین مجازی (قابلیت حمل)
کلاس	Is an instance of; shares access methods of	طراحی سیستم‌های شیءگرا، توسعه سریع سیستم یا همان توسعه با استفاده از قالب‌های مشترک
سرویس‌گیرنده-سرویس‌دهنده	Communicates with; depends on	عملیات توزیع‌شده، جداسازی دغدغه‌ها، تحلیل کارائی، توازن بار،
فرآیند	Runs concurrently with; may run concurrently with; excludes; precedes;	تحلیل زمانبندی، تحلیل کارائی
همزمانی	Runs on the same logical thread	شناسائی محلهایی که رقابت منابع وجود دارد، محلهایی که نخها ممکن است تقسیم شوند، به هم متصل شوند، ایجاد یا از بین روند
مخزن داده	Produces data; consumes data	کارائی، یکپارچگی داده و قابلیت اصلاح
استقرار	Allocated to; migrates to	تحلیل کارائی، قابلیت دسترس‌پذیری و امنیت
پیاده‌سازی	Stored in	کنترل پیکربندی، یکپارچگی، فعالیت‌های آزمایش
انتساب کار	Assigned to	مدیریت پروژه، استفاده بهتر از تجارب، مدیریت دارائی‌ها

فهرست مطالب

فصل سوم

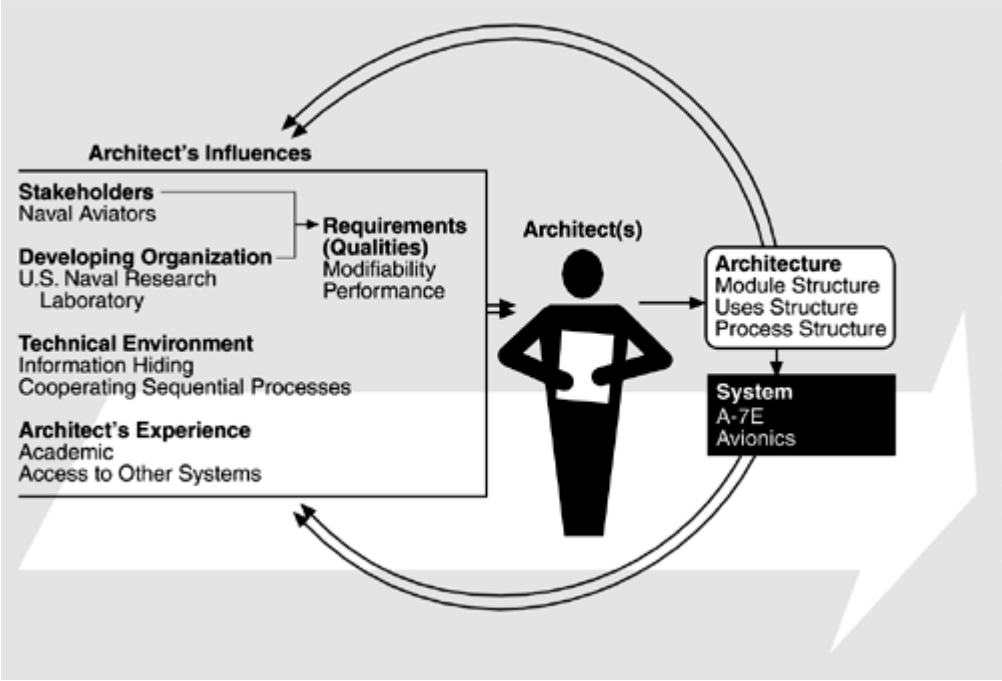
سیستم هوانوردی *A-7E*: مطالعه موردی برای بکارگیری ساختارهای معماری

در این فصل یک مطالعه موردی ارائه خواهد شد تا از طریق آن بتوان نشان داد معماری نرم‌افزار چگونه عناصر یک سیستم و ارتباطات بین آنها را نمایش می‌دهد. در واقع هدف بیان و به تصویر کشیدن این موضوع است که یک سیستم می‌تواند ساختارهای مختلفی داشته باشد و این ساختارها هر کدام از یک جنبه سیستم را توصیف می‌کنند.

۳-۱- چرخه حرفه معماری

ساخت سیستم هوانوردی *A-7E* در اوایل سال ۱۹۷۷ شروع شد. توسعه‌دهندگان این سیستم علاوه بر اینکه می‌خواستند نیازهای سازمان هوایی ایالات متحده را برآورده کنند سعی داشتند از طریق تولید یک چنین نرم‌افزاری نشان دهند که استراتژیهای مهندسی نرم‌افزار (مانند پنهان‌سازی اطلاعات و فرآیندهای ترتیبی همکار[1]) می‌توانند برای سیستم‌های بلادرنگ تعبیه شده درونی با کارایی بالا[2] نیز مفید باشند. شکل ۳-۱ دید کلی از چرخه حرفه معماری سیستم هوانوردی *A-7E* را نشان می‌دهد که بخشهای اصلی آن به شرح زیر هستند:

- **ذینفعان سیستم**، خلبانان نیروی هوایی هستند.
- **سازمان توسعه دهنده**، آزمایشگاه تحقیقاتی نیروی هوایی ایالات متحده است.
- **محیط فنی**، پنهان‌سازی اطلاعات و فرآیندهای ترتیبی همکار هستند.
- **تجارب معماری**، دانشگاهی و دسترسی به اطلاعات دیگر سیستم‌های مرتبط است.



شکل ۳-۱ : چرخه حرفه معماری سیستم هوانوردی *A-7E*

[1] Cooperating sequential processes
[2] High-performance embedded real-time systems

۳-۲- نیازمندیها و خصوصیات کیفی

در این بخش هدف پاسخ دادن به سئوالاتی از این قبیل است: سیستم چیست؟ چه خصوصیات کیفی برای دستیابی مهم هستند؟ و اینکه نرم‌افزار چه نقشی در انجام وظایف سیستم دارد؟

شکل ۳-۲ یک *A-7E Corsair II* را نشان می‌دهد. *A-7E Corsair II* یک جنگنده تک سرنشین است که در دهه‌های ۱۹۶۰، ۱۹۷۰ و ۱۹۸۰ مورد استفاده قرار گرفته شده است این جنگنده یکی از نخستین جنگنده‌های نظامی است که در آن از کامپیوتر در جهت کمک به خلبان و هدف‌گیری استفاده شده است.

شکل ۳-۲ : یک *A-7E Corsair II*

کامپیوتر داخلی *A-7E* یک ماشین *IBM* بدون نیاز به کامپایلر است که زبان برنامه‌نویسی آن اسمبلی است. در این کامپیوتر ثباتهایی جهت انتقال داده آنالوگ به دیجیتال و بالعکس در نظر گرفته شده است تا بدین طریق بتوان داده‌هایی را به دستگاه‌های *A-7E* ارسال یا از آن دریافت کرد. در *A-7E* نرم‌افزار مسئول خواندن سنسورها و به روز کردن صفحه مدیریت خلبان است (صفحه مدیریت خلبان، به خلبان کمک می‌کند که هدف‌گیری کرده و به سمت هدف شلیک کند). نرم‌افزار *A-7E* مانند نرم‌افزارهای مدرن امروزی عمل نمی‌کرد آن فقط بخشی از عملیات را مدیریت می‌کند.

مهمترین سنسورهایی که در سیستم *A-7E* وجود دارند و نرم‌افزار باید آنها را خوانده و مدیریت کند به شرح زیر هستند:

- سنسور فشار هوا (میله‌ای در جلوی هواپیما)
- رادار محاسبه فاصله تا زمین و سمت نجومی[1]
- رادار داپلر (محاسبه سرعت و زاویه)
- مجموعه‌ی اندازه‌گیری ماندی (اندازه‌گیری شتاب بین محورها و زمین)
- انواع اسلحه در هر بال
- مجموعه‌ای از سوئیچ‌ها در کابین خلبان

صفحه مدیریت خلبان نیز شامل بخش‌های زیر است:

- نمایش *Head-up (HUP)*
- نمایش حرکت روی نقشه
- نمایش صفحه کلید و کلیدهای الفبایی و عددی
- چراغ‌های هشدار
- صفحه مدرج موجود در کابین

خلبان هواپیما می‌تواند به روشهای مختلف بین نرم‌افزار و یک موقعیت زمینی ارتباط برقرار کند که این روشها عبارتند از:

۱. مکان نمای روی *HUD* را روی موقعیت مورد نظر برده و دکمه انتخاب[2] را فشار دهد.
۲. مکان نمای روی نقشه را روی موقعیت مورد نظر برده و دکمه انتخاب را فشار دهد.
۳. به صورت دستی رادار محاسبه تا زمین را به نقطه مورد نظر برده و دکمه انتخاب را فشار دهد.
۴. عرض و طول جغرافیایی را توسط صفحه کلید وارد نماید.

بعد از مشخص شدن محل، نرم‌افزار شروع به محاسبه اطلاعات مربوط به ناوبری[3] می‌کند و آن را در صفحه *Heads-up* نمایش می‌دهد. حال خلبان می‌تواند از میان حالت‌های ناوبری مشخص شده یکی را انتخاب کند که انتخاب آن بستگی به خروجی سنسورها در آن لحظه دارد (نرم‌افزار سیستم *A-7E* دسته کم با استفاده از پنج روش مستقیم و غیر مستقیم ارتفاع هواپیما را مشخص می‌کند).

[1] Forward-looking radar
[2] Designate
[3] Navigation

معماری که در این فصل ارائه خواهد شد معماری برای یک نرم‌افزار واقعی نیست بلکه خروجی پروژه طراحی شده بوسیله مهندسین نرم‌افزار نیروی هوائی امریکا است که از آن به عنوان تشریح کننده ایده خود استفاده کرده‌اند. خصوصیات کیفی که انتظار می‌رود سیستم دارای آن باشد شامل کارایی و قابلیت اصلاح برای تغییرات موردانتظار است. کارایی در ارتباط با بروزرسانی صفحه نمایش *A-7E* و یا محاسبات شلیک اسلحه (بر حسب ثانیه) است. قابلیت اصلاح نیز با تغییرات در نوع اسحله، سکو و غیره سروکار دارد.

۳-۳- معماری برای سیستم هوانوردی *A-7E*

معماری برای سیستم هوانوردی *A-7E* دربرگیرنده سه ساختار معماری است:

۱) تجزیه، که یک ساختار ماژولی است

۲) استفاده، که یک ساختار ماژولی است.

۳) فرآیند، که یک ساختار مولفه و اتصال است.

۳-۳-۱- ساختار تجزیه

در مواقعی که سیستم بزرگ است باید در نظر داشت که چگونه می‌توان سیستم را به واحدهای مجزا تقسیم کرد که این واحدهای مجزا بتوانند مستقل از هم پیاده‌سازی شوند و همچنین چگونه ماژولها با یکدیگر ارتباط برقرار کنند. همان طور که گفته شده است واحد ساختار تجزیه، ماژول است. ماژول می‌تواند به عنوان گروهی از رویه‌ها بعلاوه یک مجموعه از ساختار داده‌های اختصاصی در نظر گرفته شود.

در زمان پیش از پروژه *A-7E*، کارایی به عنوان یک هدف برجسته در سیستم‌های تعبیه شده[1] در نظر گرفته می‌شد، اما هدف از طراحی سیستم *A-7E* برقراری تعامل بین کارایی و قابلیت اصلاح بود. در واقع پروژه می‌خواست نشان دهد که می‌توان به کارایی بدون کاهش قابلیت اصلاح دست پیدا کرد.

۳-۳-۱-۱- پنهان‌سازی اطلاعات

تجزیه ماژول *A-7E* مبتنی بر پنهان‌سازی اطلاعات است. هدف از پنهان‌سازی اطلاعات، محصورسازی بخشهای مختلف سیستم در داخل ماژولهای مختلف است (پنهان‌سازی اطلاعات یکی از تاکتیک‌های مربوط به معماری است). پنهان‌سازی اطلاعات، به یک ماژول الزام می‌کند که فقط از طریق یک مجموعه تسهیل

[1] Embedded systems

کننده‌های عمومی (واسط‌ها) با بیرون تعامل برقرار کند. بنابراین هر ماژول یک سری رویه‌های دسترسی فراهم می‌کند که می‌تواند توسط سایر ماژولهای موجود در سیستم فراخوانی شود. رویه‌های دسترسی فقط مفهوم بین ماژولی[1] را برای برقراری ارتباط با اطلاعات محصور شده در درون یک ماژول فراهم می‌کنند. محصورسازی باعث می‌شود که در صورت وجود تغییراتی در سیستم، هر تغییر در داخل ماژول مرتبط با خود کنترل شود.

برای مثال اگر یک دستگاه مانند سنسور تعیین ارتفاع هواپیما احتمال داشته باشد در روند برنامه تولید یا توسعه جایگزین شود، بنابراین پنهان‌سازی اطلاعات جزئیات برقراری ارتباط با آن دستگاه را در درون ماژول پنهان می‌کند. واسط این ماژول احتمالا دربرگیرنده یک برنامه‌ای خواهد بود که آخرین مقادیر اندازه‌گیری شده به وسیله سنسور را باز می‌گرداند. دلیل آن نیز این است که کلیه سنسورها این قابلیت را به اشتراک می‌گذارند. بنابراین اگر سنسور جایگزین شود فقط بخش درونی ماژول نیاز به تغییر پیدا می‌کند و قسمت‌های دیگر نرم‌افزار بدون تغییر باقی خواهند ماند (واسط ماژول یک تجریدی برای سنسور به وجود می‌آورد).

یک ماژول می‌تواند شامل زیر ماژول باشد یا اینکه به عنوان یک واحد پیاده‌سازی مجزا در نظر گرفته شود. اگر ماژول شامل زیر ماژول باشد در این صورت راهنمائی‌هایی (توضیحاتی) برای زیر ماژولها ارائه می‌شود. مستندسازی ساختار تجزیه گاها راهنمای ماژول[2] نیز نامیده می‌شود. راهنمای ماژول، مسئولیت‌های هر ماژول را از طریق بیان کردن تصمیمات طراحی تعریف می‌کند که به وسیله آن ماژول محصور خواهند شد. هدف از راهنما، دستیابی به جداسازی دغدغه‌ها و همچنین کمک به مسئول نگهداشت سیستم است که درک کند چه ماژولهایی در اثر تغییرات یا اعلام یک خطا در سیستم تحت تاثیر قرار می‌گیرند. راهنما، معیارهای استفاده شده برای اختصاص یک مسئولیت ویژه به یک ماژول خاص را بیان می‌کند و آنها را در یک روشی مرتب می‌کند که بتوان اطلاعات مورد نیاز در مورد ماژولها را بدون بررسی مستندات نامربوط بدست آورد.

راهنما، به شکل درختی ساختار ماژول را نمایش می‌دهد و سیستم را به تعدادی بخشهای کوچک تبدیل می‌کند که این بخش‌ها مجددا به بخشهای کوچکتر دیگری تجزیه می‌شوند. این عمل تا زمانی ادامه پیدا می‌کند که بخشها به اندازه کافی کوچک شده باشند. هر گره غیر برگ در درخت یک ماژول را نشان می‌دهد که ترکیبی از ماژولهای فرزندش است. راهنما هیچ توضیحی درباره ارتباطات زمان اجرای میان ماژولها نمی‌دهد (آن هیچ

[1] Inter-module
[2] Module guide

اشاره‌ای ندارد در مورد اینکه چگونه ماژولها در زمان اجرای سیستم با یکدیگر ارتباط برقرار می‌کنند) بلکه به جای آن رابطه‌ی زمان طراحی بین واحدهای پیاده‌سازی را تشریح می‌کند. این عمل منجر به ساخت فاز طراحی سیستم می‌شود.

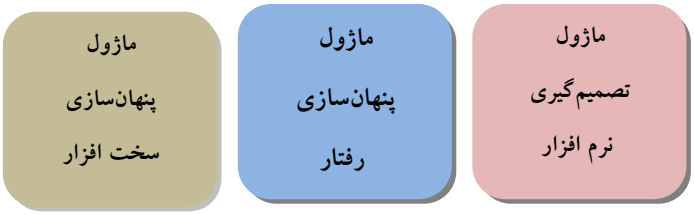
اعمال کردن اصول مطرح شده همیشه کار راحتی نیست. همه این اعمال حرکتی در جهت کاهش هرینه نرم‌افزار از طریق پیش‌بینی کردن تغییرات احتمالی هستند. پیش‌بینی تغییرات ضرورتا مبتنی بر تجربه، دانش حوزه کاربرد و درک خوب تکنولوژیهای سخت‌افزاری و نرم‌افزاری است. از آنجائی که طراح ممکن است کلیه تجارب مناسب و موفق را نداشته باشد بنابراین رویه‌های ارزیابی رسمی استفاده می‌شود که هدف از طراحی آنها استفاده کردن از تجارب دیگران است. جدول ۳-۱ نقش ساختار ماژولها در معماری *A-7E* را نشان می‌دهد.

جدول ۳-۱: چگونگی دستیابی به اهداف کیفی از طریق ساختار تجزیه ماژول *A-7E*

هدف	نحوه دستیابی
راحتی انجام تغییرات در : اسلحه‌ها، سکو، علائم و ورودی	پنهان سازی اطلاعات
درک تغییرات پیش بینی شده	رویه‌های ارزیابی رسمی به منظور استفاده از تجارب خبرگان حوزه
تخصیص تیم‌های کاری در جهت به حداقل رساندن تعاملات	ماژولهای ساخته یافته به صورت سلسله مراتبی (هر تیم کاری به یک ماژول سطح دوم و کلیه فرزندان آن نسبت داده می‌شود)

۳-۳-۱-۲- ساختار تجزیه ماژول *A-7E*

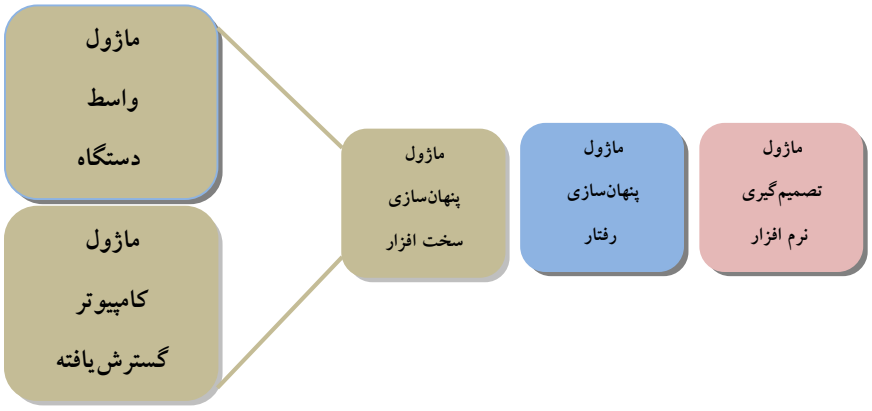
درخت تجزیه سیستم *A-7E* متشکل از سه ماژول سطح بالا است (شکل ۳-۳) این ماژولها از این نکته حاصل شده‌اند که در سیستمهای شبیه *A-7E*، تغییرات از سه ناحیه مهم –سخت‌افزار که نرم‌افزار با آن ارتباط برقرار کند، رفتارهای قابل رویت بیرونی سیستم و تصمیم‌گیری تحت حوزه طراح نرم‌افزار– نشات می‌گیرند.



شکل ۳-۳: سه ماژول سطح بالای سیستم *A-7E*

- *ماژول پنهان‌سازی سخت‌افزار*

ماژول پنهان‌سازی سخت‌افزار دربرگیرنده رویه‌های است که نیاز به تغییر در آنها وجود دارد و آن هم زمانیکه، سخت‌افزار موجود با یک نوع سخت‌افزار دیگر جایگزین شود که از نظر توانائی عمومی با یکدیگر یکسان هستند ولی واسط‌های سخت‌افزاری و نرم‌افزاری متفاوتی دارند. این ماژول، سخت‌افزار مجازی یا یک مجموعه دستگاه‌های تجریدی را پیاده‌سازی می‌کند که به وسیله سایر بخش‌های نرم‌افزاری استفاده می‌شود. اولین راز[1] (نکته برجسته) این ماژول، واسط‌های سخت‌افزاری و نرم‌افزاری و دومین نکته برجسته آن نیز ساختارها و الگوریتم‌های استفاده شده برای پیاده‌سازی سخت‌افزار مجازی است. یک زیر ماژول از ماژول پنهان‌سازی سخت‌افزار، ماژول کامپیوتر توسعه یافته[2] است که جزئیات پردازنده‌ها را مخفی می‌کند. شکل ۳-۴ زیر ماژولهای مربوط به ماژول پنهان‌سازی سخت افزار را نشان می‌دهد.

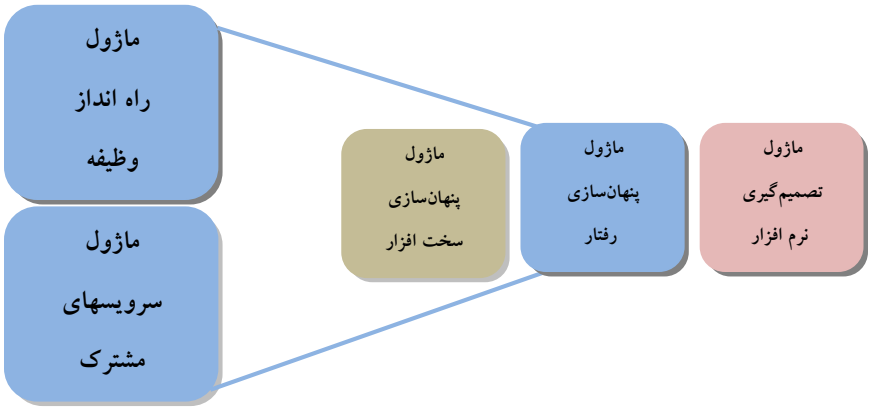


شکل ۳-۴: زیر ماژولهای ماژول پنهان‌سازی سخت افزار

- *ماژول پنهان‌سازی رفتار*[3]

ماژول پنهان‌سازی رفتار دربرگیرنده رویه‌های است که باید تغییر پیدا کند و آن هم زمانیکه، تغییری در نیازمندیها وجود دارد و آن تغییر رفتارهای مورد نیاز سیستم را تحت تاثیر قرار می‌دهد. نیازمندیهایی که منجر به تغییر رفتار سیستم می‌شوند نکته برجسته ماژول پنهان‌سازی رفتار است. شکل ۳-۵ زیرماژولهای مربوط به ماژول پنهان‌سازی رفتار را نشان می‌دهد.

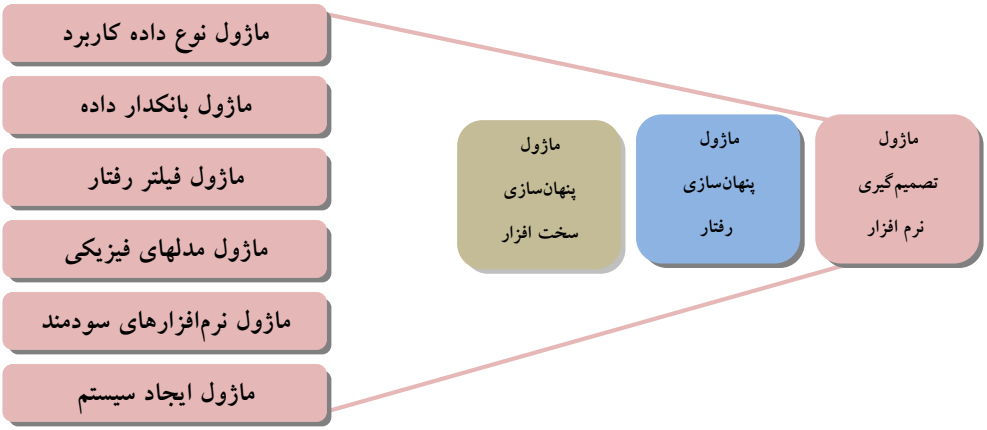
[1] Primary secrets
[2] Extended Computer (EC)
[3] Behavior-Hiding



شکل ۳-۵: زیرماژولهای ماژول پنهان‌سازی رفتار

- *ماژول تصمیم‌گیری نرم‌افزار*[1]

ماژول تصمیم‌گیری نرم‌افزار، تصمیمات نرم‌افزاری را پنهان می‌کند که مبتنی بر تئوری ریاضیات، قوانین فیزیک و ملاحضات برنامه‌نویسی از قبیل دقت و کارایی الگوریتم هستند. نکته برجسته این ماژول در مستند نیازمندیها تشریح نمی‌شود. این ماژول با ماژولهای دیگری تفاوت دارد زیرا نکته برجسته و واسط آنها توسط طراح‌ها تعیین می‌شود. تغییرات در این ماژولها بیشتر ناشی از تمایل برای بهبود کارایی و دقت است. شکل ۳-۶ زیرماژولهای مربوط به ماژول تصمیم‌گیری نرم افزار را نشان می‌دهد.



شکل ۳-۶: زیرماژولهای ماژول تصمیم‌گیری نرم افزار

[1] Software decision

راهنمای ماژول سعی دارد تا تشریح کند چگونه تداخلهای موجود بین دسته‌های مختلف (تصمیم‌گیری نرم‌افزاری و رفتار)، به وسیله مشخصات نیازمندیهای کامل و بدون ابهام حل شده و بعد از آن تجزیه سطح ثانویه انجام می‌شود. در ادامه چگونگی تجزیه شدن ماژول تصمیم‌گیری نرم‌افزار تشریح خواهد شد (ماژولهای سطح دوم ماژول تصمیم‌گیری نرم‌افزار).

- *ماژول نوع داده کاربرد*[1]

از نوع داده‌های فراهم شده به وسیله ماژول کامپیوتر توسعه یافته و همچنین نوع داده‌های سودمند برای کاربردهای هوایی حمایت می‌کند. این نوع داده‌ها نیاز به یک پیاده‌ساز وابسته به کامپیوتر ندارند. مثالهایی از این نوع داده‌ها فاصله، بازه زمانی و زاویه هستند. این نوع داده‌ها با استفاده از یک نوع داده اولیه فراهم شده به وسیله کامپیوتر توسعه یافته پیاده‌سازی می‌شوند.

نکات برجسته ماژول نوع داده کاربرد، داده‌های نمایش داده شده در متغییرها و رویه‌های استفاده شده برای پیاده‌سازی عملیات بر روی آن متغییرها هستند. واحدهای اندازه‌گیری (مانند رادیان و ثانیه) بخشی از نمایش بوده و پنهان هستند. در روند اجرا هر کجا لازم باشد ماژول عملیات تبدیل را برای دریافت یا ارسال مقدار واقعی در یک واحد اندازه‌گیری خاص را انجام می‌دهد.

- *ماژول بانکدار داده*[2]

بسیاری از داده‌ها به وسیله یک ماژول تولید شده و توسط ماژول دیگر مصرف می‌شود. در بسیاری از حالات نیز مصرف‌کننده باید مقداری را دریافت کند که از نظر عملیاتی کاملا به روز است. مدت زمانی که یک واحد داده[3] باید مجددا محاسبه (به روز) شود به وسیله خصوصیات هر دوی تولیدکننده (مانند هزینه تولید و نرخ تغییر مقدار) و مصرف‌کننده (مانند دقت مورد نیاز) مشخص می‌شود. ماژول بانکدار داده به عنوان یک "واسطه[4]" عمل می‌کند و مشخص می‌کند چه زمانی مقدار جدید برای این داده‌ها باید محاسبه شود. ماژول بانکدار داده مقادیر را از رویه‌های تولیدکننده دریافت می‌کند. رویه‌های

[1] Application data Type (AT)
[2] Data Banker (DB)
[3] Datum
[4] Middleman

مصرف‌کننده، داده‌ها را از رویه‌های دسترسی بانکدار داده بدست می‌آوردند. در بیشتر حالات مصرف‌کننده یا تولیدکننده نیاز دارد در صورت تغییر در سیاست به روزرسانی، اصلاح شود.

بانکدار داده، مقادیر کلیه داده‌ها را فراهم می‌کنند. این داده‌ها می‌تواند متعلق به وضعیت داخلی ماژولها یا وضعیت هواپیما باشد که برای گزارشگیری به کار می‌روند. بانکدار داده تا زمانی مورد استفاده قرار می‌گیرد که تولیدکننده و مصرف‌کننده ماژولهای جدا از هم بوده و یا اینکه هر دوی آنها زیرماژولهایی از یک ماژول بزرگ هستند. بانکدار داده زمانی که مصرف‌کننده نیاز به یک داده خاص از مقادیر ترتیبی تولید شده توسط یک تولیدکننده را داشته باشد و یا اینکه مقدار تولید شده توسط تولیدکننده، یک تابعی از مقادیر ورودی است (مانند تابع *sin(x)*) مورد استفاده قرار نمی‌گیرد.

انتخاب سیاست به روزرسانی باید مبتنی بر دقت نیازمندیهای مصرف‌کننده در نظر گرفته شود (مانند بیشترین مقدار زمان انتظار قابل قبول برای مصرف‌کننده یا اینکه چند وقت یکبار مصرف‌کننده نیاز به مقادیر دارد). این اطلاعات بخشی از اطلاعات معلوم شده برای پیاده‌سازی ماژول بانکدار داده هستند.

- *ماژول فیلتر رفتار*[1]

ماژول فیلتر رفتار دربرگیرنده مدلهای دیجیتالی از فیلترهای فیزیکی است. آنها توسط رویه‌های دیگر به کار گرفته می‌شوند تا داده‌های مشکلدار را فیلتر کنند. نکته برجسته اصلی این ماژول‌ها مدلهایی است که مبتنی بر مقادیر نمونه و خطاهای تخمین زده شده، مقادیر را تخمین می‌زنند. نکته برجسته دیگر آن، ساختار داده‌ها و الگوریتم‌های استفاده شده برای پیاده‌سازی این مدل‌ها است.

- *ماژول مدلهای فیزیکی*[2]

نرم‌افزار به تخمین کمیت‌هایی نیاز دارد که نمی‌توانند مستقیما اندازه‌گیری شود اما می‌توانند با استفاده از مدلهای ریاضی محاسبه شود. به عنوان مثال زمانی که یک گلوله سپری می‌کند تا به هدف برخورد کند. نکته برجسته اصلی ماژول مدلهای فیزیکی، مدلها هستند و نکته برجسته ثانویه آن نیز پیاده‌سازی مبتنی بر کامپیوتر آن مدلها است.

[1] Filter Behavior (FB)
[2] Physical Models (PM)

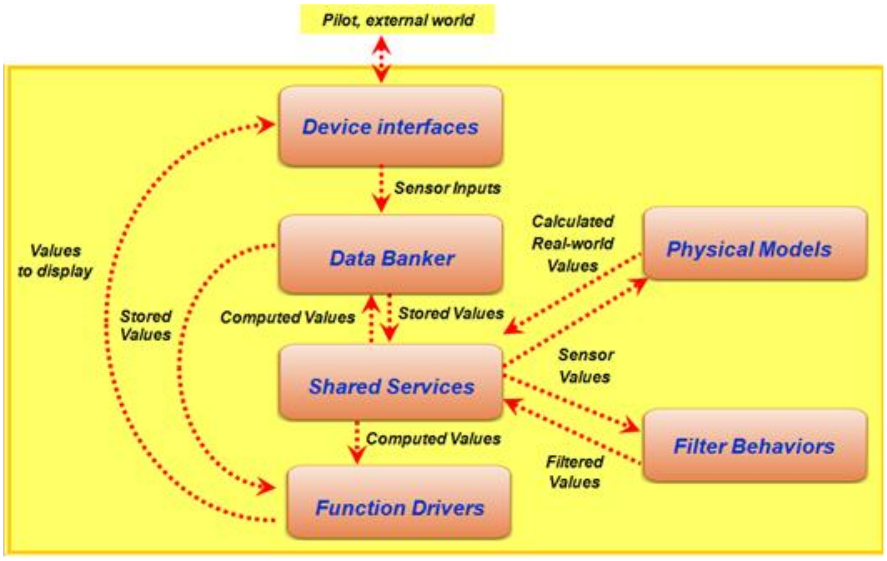
- *ماژول نرم‌افزارهای سودمند*[1]

ماژول نرم‌افزارهای سودمند دربرگیرنده برنامه‌های مستقل سودمندی است که اگر وجود نداشته باشند باید توسط هر کدام از برنامه‌نویسان برای کار خودشان نوشته شوند. برنامه‌های مستقل می‌توانند توابع محاسباتی ریاضی و ناظرهای منابع در نظر گرفته شوند. نکته بر جسته این ماژول، ساختار داده‌ها و الگوریتم‌هایی هستند که برای پیاده‌سازی رویه‌ها به کار می‌روند.

- *ماژول ایجاد سیستم*[2]

نکته برجسته ماژول ایجاد سیستم، تصمیماتی هستند که تا زمان ایجاد سیستم به تعویق انداخته می‌شوند. تصمیمات می‌تواند انتخاب مقادیر پارامترهای ایجاد سیستم یا انتخاب پیاده‌سازیهای مختلف یک ماژول در نظر گرفته شوند. نکته برجسته دیگر آن، روش استفاده شده برای تولید یک قالب قابل اجرا بر روی ماشین از طریق کد و نشان دادن تصمیمات به تعویق انداخته شده است. رویه‌ها در این ماژول بر روی کامپیوترهای *onboard* اجرا نمی‌شوند بلکه بر روی کامپیوتر معمولی اجرا می‌شوند تا کدی برای کامپیوترهای *onboard* تولید شود.

شکل ۳–۷ یک دید جریان داده بین زیر ماژولهای مربوط به ماژول تصمیم‌گیری نرم‌افزار را نشان می‌دهد.



شکل ۳–۷: دید جریان داده مربوط به ماژول تصمیم‌گیری نرم‌افزار

[1] Software Utility (SU)
[2] System generation

۳-۳-۲- ساختار استفاده

ساختار استفاده هیچ اطلاعاتی درباره زمان اجرای نرم‌افزار مشخص نمی‌کند. البته امکان این وجود دارد که یک حدس‌هایی از نحوه تعامل دو رویه در دو ماژول مختلف داشت ولی این اطلاعات به هیچ عنوان در تجزیه ماژول نشان داده نمی‌شوند. به منظور مشخص کردن نحوه ارتباطات میان رویه‌های مختلف در ماژولهای متفاوت می‌توان از ساختارهای دیگر استفاده کرد.

۳-۳-۲-۱ رابطه استفاده

مفهوم پشت ساختار استفاده، رابطه استفاده است. گفته می‌شود رویه A استفاده می‌کند از رویه B . یعنی اینکه رویه A برای اینکه نیازمندیهای خود را برآورده کند باید یک عملکرد صحیح از رویه B را داشته باشد. در عمل این ارتباط تقریبا شبیه رابطه فراخوانی است (البته نه کاملا). رویه A معمولا رویه B را برای استفاده فراخوانی می‌کند ولی دو حالت وجود دارد که در آن فراخوانی و استفاده با هم اختلاف دارند:

۱) رویه A نیاز به فراخوانی رویه B دارد اما نتیجه‌ی محاسباتی که به وسیله A انجام می‌شود هیچ وابستگی به نتیجه عملکرد B ندارد. رویه B باید فقط وجود داشته باشد تا رویه A عمل کند اما نیازی به آن در جهت صحیح اجرا شدن A نیست. در واقع رویه A رویه B را فراخوانی می‌کند ولی از آن استفاده نمی‌کند. به عنوان مثال رویه B می تواند یک کنترل کننده خطا باشد که حذف آن هیچ تاثیری بر نتیجه‌ی رویه A ندارد.

۲) رویه B عملکرد خود را بدون اینکه توسط رویه A فراخوانی شود انجام می‌دهد ولی رویه A از نتایج رویه B استفاده می‌کند. نتایج رویه B می‌تواند در یک مخزن داده ذخیره شود. به عنوان مثال B می‌تواند یک کنترل کننده وقفه باشد که رویه A فرض می‌کند آن وجود دارد و به صورت صحیح عمل می‌کند بنابراین رویه A از رویه B استفاده می‌کند ولی آن را فراخوانی نمی‌کند.

واحد ساختار استفاده، رویه دسترسی[1] است. ساختار استفاده به وسیله دیکته کردن اینکه کدام رویه‌ها اجازه دارند از رویه‌های دیگر استفاده کنند، تعریف می‌شود. ساختار استفاده معمولا با یک جدول مستندسازی می‌شود. در جدول هر سطر و ستون تداعی کننده یک رویه در سیستم است. بنابراین اگر عنصر (m,n) در

[1] Access procedure

جدول مقدار صحیح را داشته باشند نشان دهنده آن است که رویه m از رویه n استفاده می‌کند. جدول ۳-۲ نقش ساختار استفاده را در معماری سیستم *A-7E* نشان می‌دهد.

جدول ۳-۲: چگونگی دستیابی به اهداف کیفی از طریق ساختار استفاده *A-7E*

هدف	نحوه دستیابی
ساخت افزایشی و آزمایش توابع سیستم	استفاده از ساختار استفاده برای برنامه‌نویسان، که رویه‌هایی قابل استفاده توسط هر یک از آنها را محدود می‌کند
طراحی برای تغییر سکو	محدود کردن تعداد رویه‌هایی که مستقیما از سکو استفاده می‌کنند
تولید راهنمای استفاده با گستردگی قابل مدیریت	هر کجا مناسب باشد یک رابطه استفاده بین ماژولها تعریف شود

جدول ۳-۳: بخشی از مشخصات ساختار استفاده سیستم *A-7E*

رویه‌های درحال استفاده: رویه A در ...	اجازه دارد استفاده کند از هر رویه در ...
EC: Extended Computer Module	*None*
DI: Device Interface Module	*EC.DATA, EC.PGM, EC.IO, EC.PAR, AT.NUM, AT.STE, SU*
ADC: Air Data Computer	*PM.ECM*
IMS: Inertial Measurement Set	*PM.ACM*
FD: Function Driver Module	*EC.DATA, EC.PAR, EC.PGM, AT.NUM, AT.STE, SU, DB.SS.MODE, DB.SS.PNL.INPUT, DB.SS.SYSVAL, DB.DI*
ADC: Air Data Computer Functions	*DB.DI.ADC, DI.ADC, FB*
IMS: IMS Functions	*DB.DI.IMS, DI.IMS*
PNL: Panel Functions	*EC.IO, DB.SS.PNL.CONFIG, SS.PNL. FORMAT, DI.ADC, DI.IMS, DI.PMDS, DI.PNL*
SS: Shared Services Module	*EC.DATA, EC.PGM, EC.PAR, AT.NUM, AT.STE, SU*
PNL: Panel I/O Support	*DB.SS.MODE, DB.DI.PNL, DB.DI.SWB, SS.PNL.CONFIG, DI.PNL*
AT: Application Data Type Module	*EC.DATA, EC.PGM*
NUM: Numeric Data Types	*None additional*
STE: State Transition Events	*EC.PAR*

۳-۳-۲-۲- ساختار استفاده *A-7E*

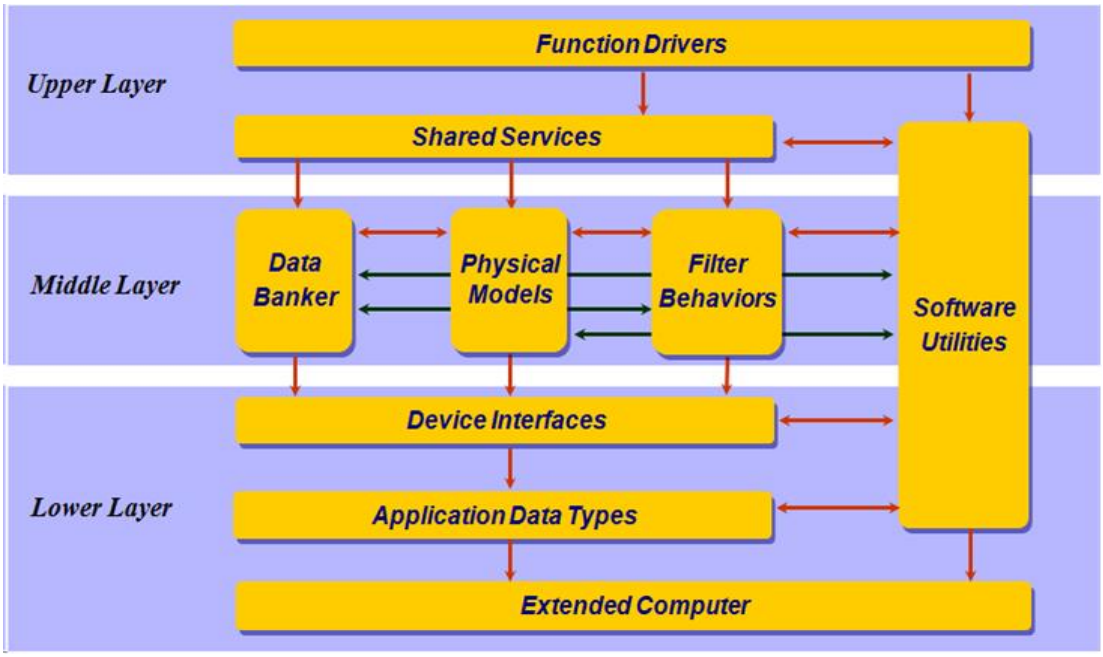
ساختار استفاده ابتدا در مشخصه سیستم مستند شده و رابطه "اجازه استفاده[1]" را نشان می‌دهد ولی نهایتا ساختار اصلی استفاده بعد از پیاده‌سازی استخراج می‌شود. جدول ۳-۳ بخشی از ساختار استفاده سیستم *A-7E* را نشان می‌دهد. از جدول ۳-۳ موضوعات زیر قابل استنتاج است:

- هیچ رویه‌ای در ماژول *EC* اجازه استفاده از رویه‌های ماژولهای دیگر را ندارد ولی همه ماژولهای دیگر می‌توانند از آن استفاده کنند.
- رویه‌ها در ماژول *AT* فقط اجازه دارند که از رویه‌های ماژول *EC* استفاده کنند.
- رویه‌ها در ماژول *DI* فقط اجازه استفاده از رویه‌های ماژولهای *EC, AT, PM* را دارند.
- رویه‌های *FD, SS* می‌توانند از رویه‌های ماژولای *DB, EC, AT, DI* استفاده کنند.
- هیچ رویه‌ای نمی‌تواند از رویه‌های *FD* استفاده کند.
- فقط یک رویه *FD* می‌تواند از یک رویه *SS* استفاده کند.

بعد از مشخص کردن رابطه‌های استفاده چیزی که حاصل می‌شود یک سیستم تقسیم‌بندی شده به لایه‌ها است. در این لایه‌بندی، ماژول *EC* پائین‌ترین لایه و ماژول *AT* نیز درست لایه بالائی ماژول *EC* را تشکیل می‌دهد. این دو لایه یک ماشین مجازی را تشکیل می‌دهند که در آن هر رویه در یک لایه اجازه دارد از رویه‌های موجود در لایه خود یا لایه‌های پائین‌تر استفاده کند.

در بالاترین لایه‌ها ماژولهای *FD* و *SS* قرار دارند که آزادی عمل زیادی برای استفاده از تسهیلات سیستم در جهت انجام وظایف خود دارند. در لایه‌های میانی نیز ماژولهای *PM, FB* و *DB* قرار دارند. ماژول *SU* نیز به صورت موازی با این ساختار قرار گرفته و اجازه دارد از هر رویه‌ای (به جز رویه‌های متعلق به ماژول *FD*) در جهت انجام کار خود استفاده کند. شکل ۳-۸ یک دید کلی از لایه‌های حاصل شده برای سیستم *A-7E* را نشان می‌دهد.

[1] Allowed-to-use



شکل ۳-۸: لایه‌های پدیدار شده از ساختار استفاده در سیستم *A-7E*

لایه‌بندی از ساختار استفاده حاصل می‌شود ولی ساختار استفاده یک جانشین برای لایه‌بندی نیست زیرا لایه‌بندی یک دید از ساختار استفاده است. به عنوان مثال در سیستم *A-7E* یک ماژول *FD* خاص، از یک مجموعه خاص از عملیات متعلق به *SS, DB, PM, DI, AT* و *EC* استفاده خواهد کرد بنابراین مجموعه رویه‌های مشتق شده در این روش یک زیر مجموعه را تشکیل می‌دهند.

ساختار استفاده همچنین یک تصویری را فراهم می‌کند که از طریق آن می‌توان درک کرد که چگونه ماژولها در زمان اجرا در جهت انجام وظایف ارتباط برقرار می‌کنند. به عنوان مثال، رویه *FD* مقدار خروجی مرتبط با خروجی یک دستگاه را کنترل می‌کند. در واقع *FD* داده‌ها را (از طریق *DB*) از تولیدکنندگان داده بازیابی می‌کند قوانین لازم برای محاسبه مقدار صحیح را به آنها اعمال می‌کند و آن مقدار را از طریق فراخوانی واسط‌های دستگاه مورد نظر به آنها ارسال می‌کند. داده‌ای که *FD* دریافت می‌کند می‌تواند از منابع زیر بدست آید:

- رویه‌های واسط دستگاه
- رویه‌های مدلهای فیزیکی
- رویه‌های سرویس‌های به اشتراک گذاشته شده

بعد از اینکه ساختار استفاده طراحی شد، پیاده‌سازان می‌دانند به چه واسطهایی برای کار خودشان نیاز دارند. و بعد از اینکه پیاده‌سازی کامل شد مستند ساختار استفاده می‌تواند کاملا مشخص شود یعنی زیر مجموعه‌ها کاملا مشخص خواهند شد.

۳-۳-۳- ساختار فرآیند

سومین ساختار معماری مهم برای سیستم *A-7E* ساختار فرآیند است. اگر چه کامپیوتر به کار گرفته شده در سیستم هواپیما تک پردازنده است ولی ماژول کامپیوتر توسعه یافته یک واسط برنامه‌نویسی مجازی را فراهم می‌کند که توانائی چند پردازنده‌ای را نمایان می‌سازد. در واقع نرم‌افزار به صورت فرآیندهای ترتیبی همکار پیاده‌سازی می‌شود که با یکدیگر در جهت استفاده از منابع مشترک هماهنگ می‌شوند. فرآیند یک مجموعه از گام‌های برنامه‌نویسی است که در جهت پاسخ به یک رویداد مکررا تکرار می‌شوند. هر فرآیند نخ کنترل[1] خودش را دارد و می‌تواند خودش را در زمان انتظار برای یک رویداد به حالت معلق ببرد.

فرآیندها به دو دلیل در سیستم *A-7E* در نظر گرفته شده‌اند. اولی برای *FD* ها است که مقادیر خروجی نرم‌افزار *avionics* را محاسبه کنند (یعنی مورد نیاز هستند که به صورت دوره‌ای یا در پاسخ به یک رویداد اجرا شوند). دومی اینکه فرآیندها روشی برای پیاده‌سازی رویه‌های دسترسی هستند. اگر مقدار برگشت داده شده به وسیله یک رویه خیلی پرهزینه از نظر محاسباتی باشد یک برنامه‌نویس برای اینکه بتوان مقدار داده خروجی را به موقع تحویل دهد می‌تواند مقادیر داده را توسط فرآیندی در پشت زمینه محاسبه کند و به محض فراخوانی رویه دسترسی، مقدار را برگرداند.

ساختار فرآیند دربرگیرنده مجموعه‌ای از فرآیندها در نرم‌افزار است. رابطه در این ساختار به صورت "همزمان شدن با[2]" تعریف می‌شود که مبتنی بر رویدادی است که یک فرآیند تولید می‌کند و چندین فرآیند دیگر منتظر آن هستند. این رابطه به عنوان یک ورودی اصلی برای فعالیت زمانبندی استفاده می‌شود. جدول ۳-۴ نقش ساختار فرآیند در معماری سیستم *A-7E* را نشان می‌دهد.

[1] Thread of control
[2] Synchronizes-with

جدول ۳-۴: چگونگی دستیابی به اهداف کیفی از طریق ساختار فرآیند *A-7E*

هدف	نحوه دستیابی
نگاشت ورودی به خروجی	هر فرآیند به صورت چرخه‌ای پیاده‌سازی شده که نمونه‌گیری می‌کند، ورودی را مشخص می‌کند، محاسبه می‌کند و خروجی را نمایش می‌دهد
نگهداری محدودیت‌های بلادرنگ	فرآیند از طریق ساختار فرآیند شناسائی می‌شود و سپس زمانبندی برون خطی انجام می‌شود
فراهم کردن به موقع نتایج محاسبات زمانبر	محاسبات را در پشت زمینه انجام می‌دهد و نتیجه به روز شده را در هنگام فراخوانی نشان می دهد.

ساختار فرآیند زمانی ایجاد می‌شود که ساختارهای دیگر طراحی شده باشند. در سیستم *A-7E* رویه‌های *FD* به صورت فرآیند پیاده‌سازی می‌شوند و فرآیندهای دیگر نیز در پشت زمینه محاسبات زمانبر را محاسبه می‌کنند. از ساختار فرآیند دو نوع اطلاعات قابل استخراج است. اولی اینکه چه رویه‌های در بدنه هر فرآیند وجود دارد و دومی اینکه کدام فرآیندها نمی‌توانند به صورت همزمان اجرا شوند.

فهرست مطالب

فصل چهارم

درک خصوصیات کیفی

در این فصل هدف درک چگونگی تشریح خصوصیات کیفی است که یک سیستم در جهت برآورده کردن اهداف خود باید دارای آنها باشد. بدین منظور ابتدا ارتباط بین معماری و خصوصیات کیفی بررسی شده سپس انواع خصوصیات کیفی معرفی خواهند شد و نهایتا نشان داده خواهد شد که چگونه خصوصیات کیفی می‌توانند به کار گرفته شوند تا اینکه بتوان نیازمندیهای کیفی یک سیستم را بدست آورد.

۴-۱- مقدمه

همان طور که در چرخه حرفه معماری بیان شد توجه کردن به حرفه باعث مشخص شدن کیفیت‌هایی[1] می‌شود که لازم است معماری نرم‌افزار با آنها مطابقت داشته باشد. این کیفیت‌ها فراتر از نیازمندیهایی هستند که مبنای قابلیت‌ها، سرویس‌ها و رفتار سیستم است. در واقع وظیفه‌مندی و خصوصیات کیفی در رابطه تنگاتنگی با یکدیگر قرار دارند. بنابراین اگر فرض شود که وظیفه‌مندیها تنها مسائل مهم در شمای تولید و توسعه یک سیستم نرم‌افزاری هستند در آن صورت کاملا مشخص است که نگاه کوته نظرانه‌ای به مسئله وجود دارد.

سیستم‌ها مرتبا طراحی مجدد می‌شوند ولی نه به خاطر اینکه نیازهای وظیفه‌مندی خود را پوشش دهند بلکه به دلیل اینکه مشکلاتی در زمینه نگهداری، قابلیت حمل‌پذیری، مقیاس‌پذیری، سرعت پاسخگوی ضعیف و امنیت دارند. به عبارت بهتر، به دلیل اینکه خصوصیات کیفی در این سیستم‌ها مشخص نشده و یا هماهنگی کارایی بین خصوصیات کیفی و نیازهای وظیفه‌مندی برقرار نشده است این سیستم‌ها مجددا طراحی می‌شوند. این طراحی مجدد تا زمانی که نتوان خصوصیات کیفی مورد نظر سیستم‌ها را برآورده کرد ادامه پیدا خواهد یافت. بنابراین کاملا مشخص است که خصوصیات کیفی در موفقیت و بکارگیری یک سیستم نقش بسزایی دارند.

روند این فصل تمرکز بر روی فهم چگونگی بیان خصوصیات کیفی است که یک سیستم باید دارای آن باشد. بدین منظور ابتدا ارتباط بین خصوصیات کیفی و معماری نرم‌افزار از طریق بررسی دقیق خصوصیات کیفی بحث می‌شود. سپس بیان خواهد شد که چگونه این خصوصیات کیفی باید به کار گرفته شوند تا بتوان نیازمندیهای کیفی برای یک سیستم را مشخص کرد.

۴-۲- وظیفه‌مندی و معماری

وظیفه‌مندی و خصوصیات کیفی متعامد هستند (در تئوری) زیرا در صورت اینکه این دو متعامد نبودند انتخاب عملکرد[2]، سطح امنیت یا کارایی یا دسترس‌پذیری و یا قابلیت استفاده را دیکته می‌کرد (اصل تعامد[3] پیشنهاد می‌دهد که عملکردهای مستقل به وسیله سازوکارهای مستقل کنترل شوند). در یک نظر، مشاهد می‌شود

[1] Qualities
[2] Function
[3] Orthogonality principle

که امکان آن وجود دارد تا بتوان به صورت مستقل یک سطح دلخواه از خصوصیات کیفی و نیازهای وظیفه‌مندی را انتخاب کرد (پیاده‌سازی کرد). اما این نکته بدین معنی نیست که هر سطح از خصوصیات کیفی با هر عملکردی قابل دستیابی است. به عنوان مثال، کارکردن با تصاویر گرافیکی پیچیده و یا مرتب کردن بانک‌های اطلاعاتی بسیار حجیم ممکن است با کارایی بالایی انجام نشود اما چیزی که امکان دارد این است که برای هر کدام از عملکردها می‌توان یک سطح نسبی از کیفیت‌ها را مشخص کرد. در واقع یک نوع تعادل بین آنها ایجاد کرد.

حال سئوال این است که وظیفه‌مندی چیست؟ وظیفه‌مندی توانایی سیستم برای انجام کاری است که به خاطر آن ایجاد شده است. یک وظیفه[1] (تکلیف) برای اینکه بتواند کار خود را به طور کامل به اتمام برساند نیازمند آن است که بسیاری از عناصر سیستم به صورت هماهنگ با یکدیگر عمل کنند (مانند لوله‌کشها، نقاش‌ها و ... که با یکدیگر همکاری می‌کنند تا یک ساختمان ساخته شود). اگر عناصر در یک سیستم به مسئولیت‌های درستی تخصیص داده نشوند و یا قابلیت‌های کافی برای هماهنگ شدن با سایر عناصر را نداشته باشند در آن صورت سیستم نمی‌تواند وظیفه‌مندی درستی را از خود نشان دهد.

وظیفه‌مندی می‌تواند با استفاده از هر ساختاری حاصل شود. در حقیقت اگر وظیفه‌مندی تنها نیازمندی بود، سیستم می‌توانست به صورت یک ماژول واحد با هیچگونه ساختاری داخلی وجود داشته باشد و جایگزین ترکیب ماژولهای مختلف با اهداف گوناگون شود. در این صورت وظیفه‌مندی تا حد زیادی مستقل از ساختار می‌شد. اما معماری نرم‌افزار تخصیص ساختار را تحمیل می‌کند زیرا که خصوصیات کیفی از اهمیت زیادی در توسعه سیستم برخوردار هستند.

۴-۳- معماری و خصوصیات کیفی

خصوصیات کیفی در واقع نیازهای غیروظیفه‌مندی هستند که نقش بسیار مهمی در تعیین سبک معماری دارند. خصوصیات کیفی در نرم‌افزار به صورت یکسری پارامترهای فنی در نظر گرفته می‌شوند و معمار بایستی این خصوصیات را به صورت قابل فهم برای سایر سهامداران فهرست کند تا آنها بتوانند در مورد این فهرست اظهار نظر کنند. اظهار نظر سهامدارن نهایتا منجر به این می‌شود که سیستم دارای کیفیت مورد نظر باشد.

[1] Task

معماری اولین قدم تولید نرم‌افزار است که خصوصیات کیفی در آن قابل ردیابی هستند. خصوصیات کیفی در کلیه مراحل طراحی، پیاده‌سازی و استقرار مطرح می‌شوند (هیچ خصوصیت کیفی فقط وابسته به طراحی یا پیاده‌سازی یا استقرار نیست) و در نتیجه چنانچه توسط معماری پشتیبانی شوند قابلیت ردیابی آنها راحت‌تر خواهد بود. در معماری قسمتی از خصوصیات کیفی مدنظر قرار می‌گیرد (در ارتباط با معماری هستند) که مربوط به کلیه مراحل طراحی، پیاده‌سازی و انتقال است. به عنوان مثال، در طراحی واسط کاربر اینکه از *checkbox* استفاده شود یا *radio box* جزء معماری محسوب نمی‌شود (غیرمعمارانه) زیرا تنها در ارتباط با مرحله طراحی است. درمقابل، اینکه سیستم توانایی لغو کردن[1]، خنثی کردن[2] اعمال و یا دوباره استفاده کردن از اطلاعات قبلی را داشته باشد مربوط به معماری است زیرا که این عملکردها از نتیجه همکاری بخش‌های مختلف حاصل می‌شوند (معمارانه).

به عنوان مثال دیگر، خصوصیت قابلیت اصلاح را در نظر بگیرید. خصوصیت قابلیت اصلاح متشکل از دو بخش است، بنابراین می‌تواند هم معمارانه و هم غیر معمارانه در نظر گرفته شود. زیرا بخش اول که در ارتباط با نحوه تقسیم‌بندی وظیفه‌مندیها است، مربوط به معماری می‌شود ولی بخش دوم که در ارتباط با چگونگی پیاده‌سازی تکنیک‌های کد است مربوط به معماری نمی‌شود. در ضمن باید به این مساله نیز توجه داشت که معماری به تنهایی توانایی دستیابی به خصوصیات کیفی را ندارد بلکه ساختار یا پایه‌ای را برای دستیابی به خصوصیات کیفی فراهم می‌کند (ایجاد می‌کند).

در مورد سیستم‌های پیچیده نیز ذکر این نکته مهم است که دستیابی به کلیه اهداف کیفی در سیستم‌های پیچیده غیرممکن است زیرا خصوصیات کیفی بر یکدیگر تاثیرگذار هستند و این معمار است که باید فهرستی از اولویت‌ها برای خصوصیات کیفی را فراهم نموده و بر اساس آن بین خصوصیات کیفی موازنه ایجاد کند. موازنه بدین معنی است که ممکنه همه خصوصیات کیفی را نتوان همزمان فراهم کرد (پشتیبانی کرد) ولی باید بررسی کرد که کدام یک از شرایط برای سیستم از اهمیت بیشتری برخوردار هستند. مثلا در سیستمی که هم خصوصیت کیفی کارایی و هم خصوصیت کیفی قابلیت استفاده مجدد مطرح است باید بررسی شود که آیا کارایی اهمیت بیشتری دارد یا قابلیت استفاده مجدد.

[1] Cancel
[2] Undo

۴-۴- خصوصیات کیفی

در *IEEE 1990* کیفیت به صورت زیر تعریف شده است:

۱. میزان موفقیتی که یک سیستم، مولفه و یا فرآیند در رویارویی با مجموعه‌ای از نیازمندیهای مشخص بدست می‌آورد.

۲. میزان موفقیتی که یک سیستم، مولفه و یا فرآیند در رویارویی با مجموعه ای از نیازهای مورد انتظار کاربران بدست می آورد.

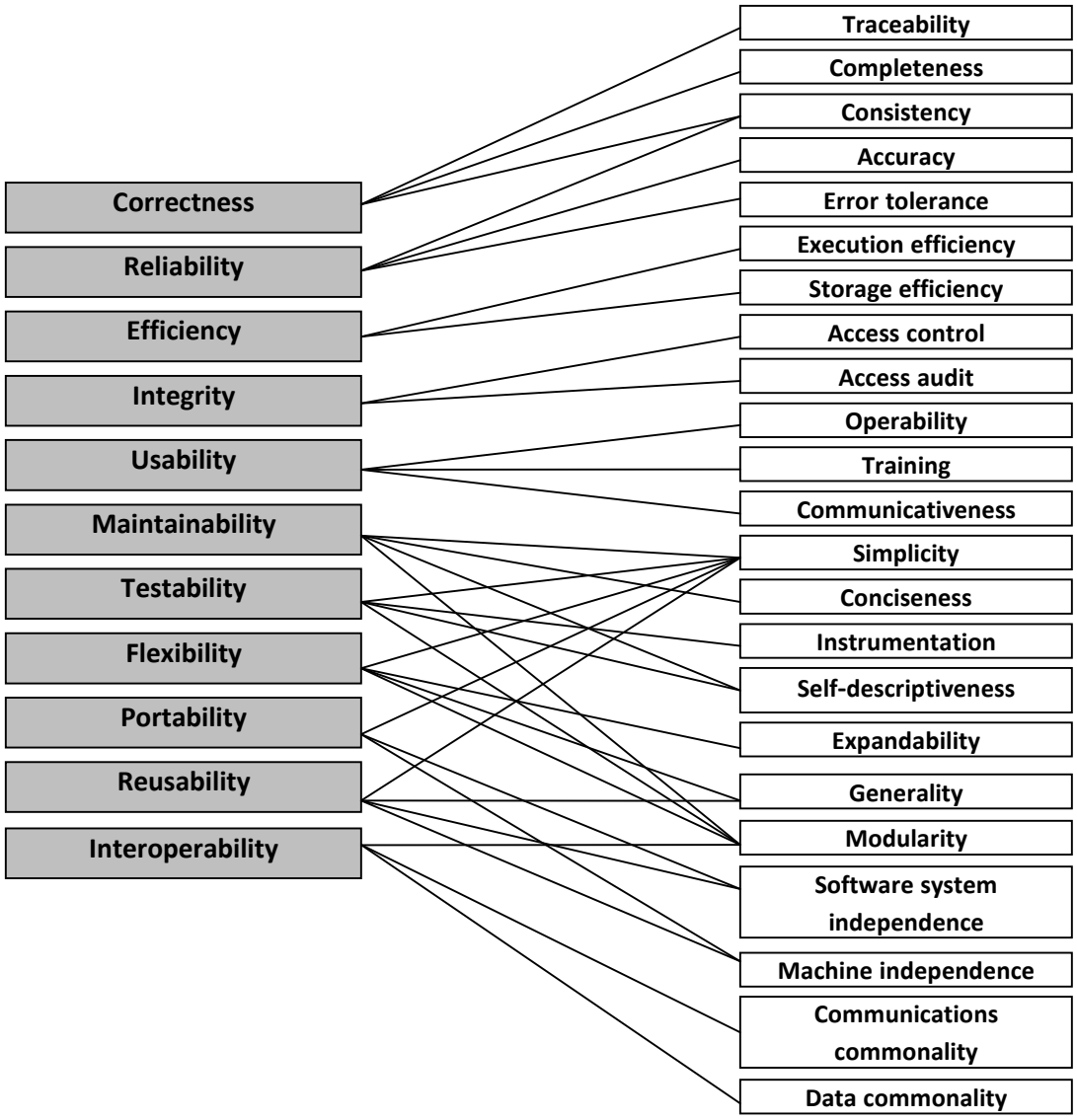
از دهه ۷۰ میلادی به بعد تلاشهای زیادی در جهت ارائه مدلهای گوناگون از خصوصیات کیفی صورت گرفته است تا بتوان به کمک آنها خصوصیات کیفی را به ویژگیهایی اندازه‌پذیر و قابل لمس در سیستم تبدیل کرد. از این مدلها می‌توان به دسته‌بندیهای ارائه شده توسط *McCall* در سال ۱۹۷۷، *Boehm* در سال ۱۹۷۸، *ISO* در سال ۱۹۹۲، *IEEE* در سال ۱۹۹۸ و *Bass* در سال ۲۰۰۳ اشاره کرد. دسته‌بندیهای متنوع و گوناگون دیگری نیز وجود دارند که محققین بنا به مورد در زمینه کارهای خود ارائه داده‌اند که برخی از آنها عبارتند از:

- دسته‌بندی ارائه شده توسط *Smith* در سال ۲۰۰۲ در مورد کارایی
- دسته‌بندی ارائه شده توسط *Laprie* در سال ۱۹۹۲ در مورد قابلیت اطمینان و قابلیت دسترسی
- دسته‌بندی ارائه شده توسط *Randell* در سال ۱۹۹۵ در مورد ایمنی

۴-۴-۱- مدلهای کیفی

مدلهای کیفی در معماری نرم‌افزار عموماً به صورت دسته‌بندی از خصوصیات کیفی و ارتباطات میان آنها هستند که برای مشخص نمودن و ارزیابی نیازهای غیروظیفه‌مندی بکار می‌روند. اکثر این مدلها در دو سطح به صورت خصوصیات خارجی قابل مشاهده و خصوصیات داخلی قابل اندازه‌گیری ارائه شده‌اند. همانطور که در شکل ۴-۱ نشان داده شده است در دهه ۱۹۷۰ *Cavano, McCall* یک چارچوب کیفی برای کیفیت محصولات ارائه کردند که در واقع از سه دیدگاه منشاء گرفته بود:

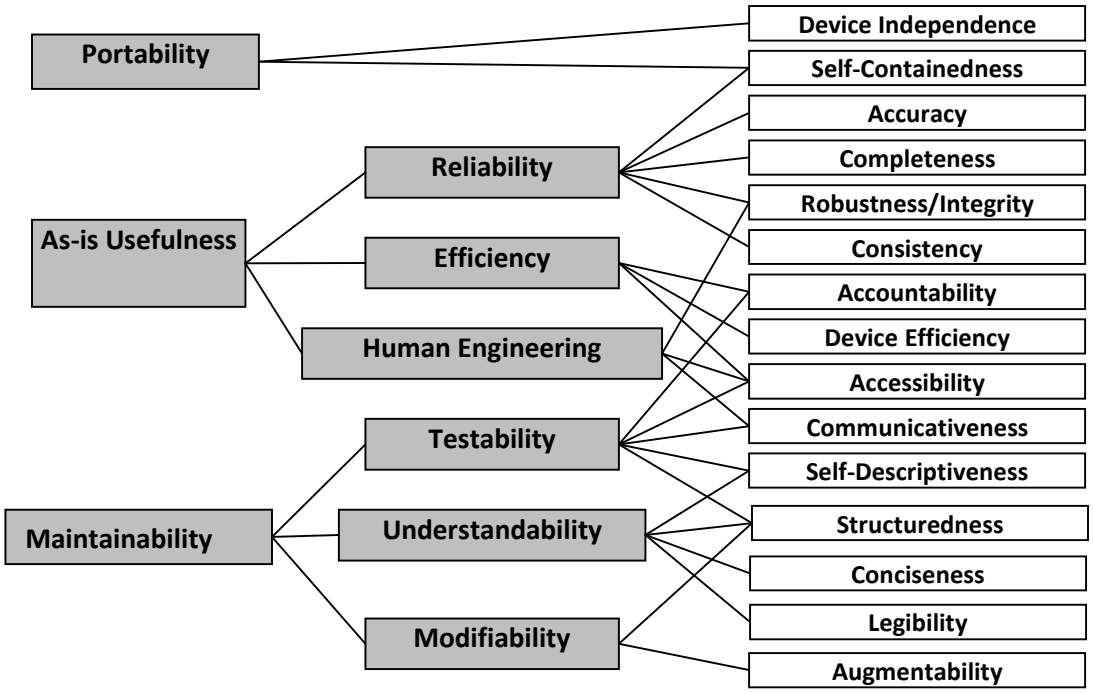
- عملکرد
- تجدید نظر
- تحول



شکل ۴-۱ : چارچوب کیفیتی *Mc Call*

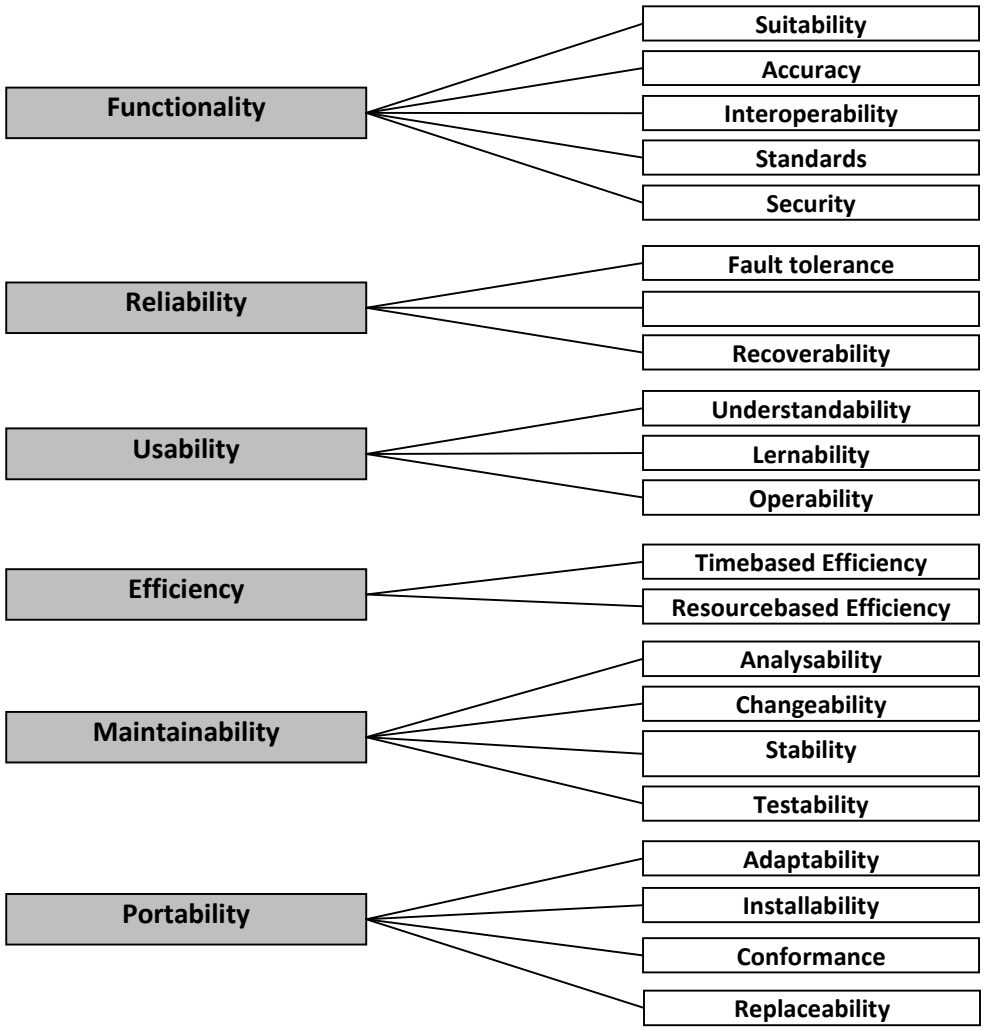
همزمان با این چارچوب، *Boehm* مطابق شکل ۴-۲ مدل کیفی را ارائه نمود که از سه نقطه نظر زیر خصوصیات کیفی را مورد بررسی قرار داده بود:

- کاربر نهایی
- کاربران در مکانهای مختلف (قابلیت حمل)
- کاربران در زمانهای مختلف (قابلیت نگهداری)



شکل ۴-۲: مدل کیفیتی *Boehm*

توسعه بیشتر مدلهای کیفی منجر به ارائه یک استاندارد توسط *ISO* شد و در *ISO 9126* مدل کیفی در دو سطح به صورت خصوصیات و زیر خصوصیات تعریف شد. یکی از تفاوتهای کلیدی و اصلی این مدل کیفی با سایر مدلها در این بود که در آن هر زیر خصوصیت تنها توسط یک خصوصیت در سطح بالاتر تحت تأثیر قرار می‌گرفت در حالیکه در سایر مدلها اینگونه نبود. این مدل در شکل ۴-۳ ارائه شده است.



شکل ۴-۳: مدل کیفیتی استاندارد *ISO/ 9126*

اما آنچه که به طور وسیع و گسترده در معماری نرم افزار مورد استفاده قرار می‌گیرد، دسته‌بندی است که توسط *Bass, Clements, Kazman* ارائه شده است. در این دسته‌بندی خصوصیات کیفی بر اساس اینکه در زمان اجرا قابل مشاهده هستند یا خیر، در دو دسته قرار می‌گیرند:

- *خصوصیات کیفی قابل مشاهده در زمان اجرا*: این خصوصیات نشان می‌دهند که در طول مدت اجرا، یک سیستم چقدر خوب می‌تواند نیازمندیهای رفتاری خودش را تأمین کند. یعنی به لحاظ رفتاری معین می‌کند که آیا سیستم نتایج را برآورده می‌کند و آیا این نتایج را در زمان درست بر آورده می‌سازد

یا خیر؟ به عبارت بهتر نرم‌افزار باید اجرا شود تا مشخص شود که در اثر اجرای آن چنین خصوصیاتی فراهم می‌شوند یا خیر؟ این خصوصیات عبارتند از :

- کارایی
- امنیت
- قابلیت دسترسی
- قابلیت عملکرد یا و ظیفه‌مندی[1]
- قابلیت استفاده

- *خصوصیات کیفی غیر قابل مشاهده در زمان اجرا*: این خصوصیات به گونه‌ای هستند که در زمان اجرا نمی‌توان تشخیص داد که به آنها دست یافته‌ایم یا خیر. به عبارت دیگر نمی‌توان آنها را در زمان اجرا دید و باید بعداً ارزیابی کرد. این دسته از خصوصیات نشان می‌دهند که جمع‌آوری سیستم و آزمایش و طراحی سیستم با چه میزان سهولت و راحتی انجام می‌شود. همه خصوصیات کیفی در یک مقطع خودشان را نشان نمی‌دهند و بعضی در مرحله تحلیل، بعضی در مرحله طراحی و غیره خود را نشان می دهند. این خصوصیات عبارتند از:
  - قابلیت اصلاح
  - قابلیت حمل
  - قابلیت یکپارچگی[2]
  - قابلیت استفاده مجدد[3]
  - قابلیت آزمایش

از دیگر دسته‌بندیها، دسته‌بندی ارائه شده توسط *Jakobson, Booch, Rumbaugh* در سال ۱۹۹۹ است که به صورت زیر، نیازمندیهای غیروظیفه‌مندی را لیست کرده اند:

- کارایی
- قابلیت اطمینان

[1] Functionality
[2] Integritability
[3] Reusability

- قابلیت نگهداری
- قابلیت گسترش[1]
- قابلیت دسترسی
- دقت
- امنیت
- اولویت
- سودمندی برای کاربر[2]
- قابلیت استفاده
- زمان بازیافت
- استفاده حافظه[3]
- راحتی فراگیری

از دیگر مدلهای کیفی ارائه شده می‌توان به مدل پیشنهادی *Grady* که *FURPS* نامیده می‌شود و شامل قابلیت عملکرد، قابلیت استفاده، قابلیت اطمینان، کارایی و قابلیت پشتیبانی[4] است و همچنین مدل پیشنهادی *Dromey* که در واقع یک مدل هشت‌تایی شامل شش خصوصیت کیفی سطح بالا در مدل *ISO 9126* به همراه قابلیت استفاده مجدد و بلوغ فرآیندی[5] است، اشاره کرد.

هر دو دسته خصوصیات کیفی معرفی شده (چه قابل مشاهده در زمان اجرا و چه غیر قابل مشاهده در زمان اجرا) مربوط به خود سیستم و برنامه کاربردی هستند و به آنها خصوصیات کیفی مرتبط با سیستم می‌گویند ولی یک سری خصوصیات کیفی مرتبط با حرفه نیز وجود دارند که تحت عنوان خصوصیات کیفی تجاری از آنها یاد می‌شود که آنها نیز در معماری اثرگذار هستند. همچنین نوع دیگری از خصوصیات کیفی در معماری نرم‌افزار هستند که مربوط به معماری می‌باشند. در بخشهای بعدی به توضیح این سه دسته از خصوصیات کیفی پرداخته خواهد شد.

[1] Extensibility
[2] Usefulness, value to users
[3] Memory usage
[4] Supportability
[5] Process maturity

۴-۵- خصوصیات کیفی سیستمی

خصوصیات کیفی سیستمی از دهه ۱۹۷۰ مورد توجه انجمن نرم‌افزار قرار گرفت. دسته‌بندیها و تعاریف متعدد و متنوعی در مورد خصوصیات کیفی ارائه شده است که در بخش قبل تعدادی از آنها ذکر شدند. اما از دیدگاه معمار سه مسئله مهم در مورد خصوصیات کیفی وجود دارد:

۱. *تعاریف ارائه شده برای یک خصوصیت کیفی عملیاتی نیست*. به عنوان مثال، بی معنی است که گفته شود یک سیستم اصلاح‌پذیر خواهد بود. زیرا هر سیستم نسبت به یک مجموعه از تغییرات قابلیت اصلاح را دارد و نسبت به یک مجموعه دیگر قابلیت اصلاح را ندارد. این موضوع در مورد سایر خصوصیات کیفی نیز صادق است.
۲. *مشخص نیست یک جنبه خاص به کدام خصوصیت کیفی تعلق دارد*. به عنوان مثال آیا خرابی سیستم جنبه‌ای از قابلیت دسترسی است یا جنبه‌ای از امنیت و یا جنبه‌ای از قابلیت استفاده است؟ در واقع خرابی سیستم می‌تواند جنبه‌ای از تمامی خصوصیات کیفی مذکور در نظر گرفته شود.
۳. *هر خصوصیت مجموعه فرهنگ لغات خاص خود را دارد*. به عنوان مثال خصوصیت کارایی شامل مفهوم رویداد است که در یک سیستم رخ می‌دهد. خصوصیت امنیت دربردارنده‌ی مفهوم حمله، خصوصیت قابلیت دسترسی شامل مفهوم خرابی و خصوصیت قابلیت استفاده دربردارنده‌ی مفهوم ورودی کاربر است که در کل همگی آنها به یک مضمون واحد اشاره دارند.

یک راه حل برای دو مسئله اول، استفاده از سناریوهای خصوصیات کیفی است. سناریوها به عنوان ابزاری برای مشخص کردن خصوصیات کیفی هستند. یک راه حل برای مسئله سوم نیز ارائه یک توصیف مختصر از هر خصوصیت است. قابل ذکر است که خصوصیات کیفی متعلق به دسته خصوصیات کیفی سیستمی شامل قابلیت دسترسی، قابلیت اصلاح، کارایی، امنیت، قابلیت آزمایش و قابلیت استفاده هستند.

۴-۵-۱- سناریوهای خصوصیات کیفی

هر سناریوی خصوصیات کیفی، یک نیازمندی مرتبط با خصوصیات کیفی[1] است. یک سناریوی خصوصیت کیفی از شش بخش تشکیل شده است:

[1] Quality-attribute-specific requirement

- منبع تحریک[1]

منبع تحریک شامل بعضی از موجودیت‌ها (مانند انسان، سیستم کامپیوتری یا هر محرک دیگر) است که یک تحریک را ایجاد می‌کند. به عبارت بهتر مولد یک تحریک است.

- محرک[2]

محرک شرایطی است که اگر در سیستم ایجاد شود لازم است که مورد بررسی قرار بگیرد.

- محیط[3]

تحریک در وضعیت‌ها و شرایط خاص و معینی روی می‌دهد. به عنوان مثال، زمانی که یک تحریک رخ می‌دهد ممکن است سیستم در حال اجرا با بارکاری زیاد یا در هر وضعیت دیگری قرار داشته باشد.

- فرآورده[4]

فرآورده موجودیتی است که هدف تحریک است. فرآورده ممکن است که کل سیستم یا بخشی از آن در نظر گرفته شود.

- پاسخ[5]

پاسخ عبارت است از فعالیتی که سیستم بعد از تحریک شدن انجام می‌دهد.

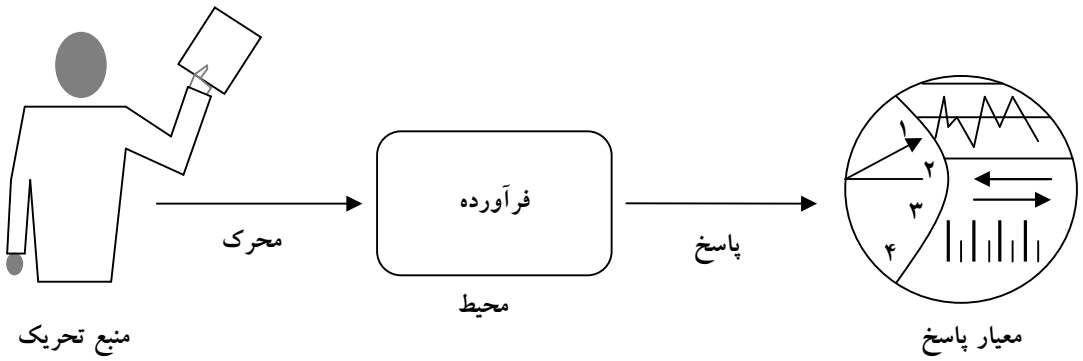
- معیار پاسخ[6]

وقتی که پاسخی داده می‌شود آن باید به روش مشخصی اندازه‌گیری شود تا نیازمندیهای مورد نظر بتواند مورد آزمایش قرار بگیرند.

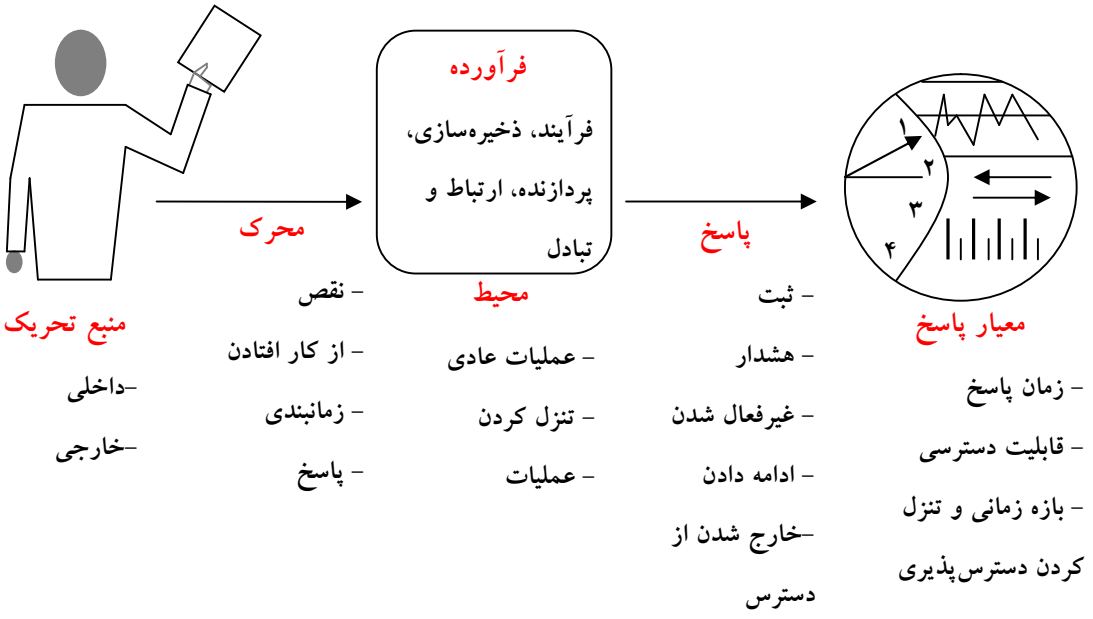
در حالت کلی دو نوع سناریوی خصوصیات کیفی وجود دارد. اولی سناریوهای خصوصیات کیفی عمومی هستند (سناریوهای عمومی) که مستقل از سیستم بوده و می‌توانند در ارتباط با هر سیستمی باشند. دیگری سناریوهای خصوصیات کیفی عینی[7] (سناریوهای عینی) هستند که برای سیستم‌های خاص مشخص می‌شوند. مشخصات خصوصیات کیفی به صورت مجموعه‌ای از سناریوهای عمومی بیان می‌شود که برای تبدیل کردن

[1] Source of stimulus
[2] Stimulus
[3] Environment
[4] Artifact
[5] Response
[6] Response measure
[7] Concrete

مشخصات خصوصیات کیفی به نیازمندیهای خصوصیات کیفی مرتبط با یک سیستم خاص، نیاز به سناریوهای عینی است. شکل ۴-۴ بخشهای یک سناریوی خصوصیات کیفی را نشان می‌دهد.



شکل ۴-۴: بخشهای اصلی سناریوی خصوصیات کیفی



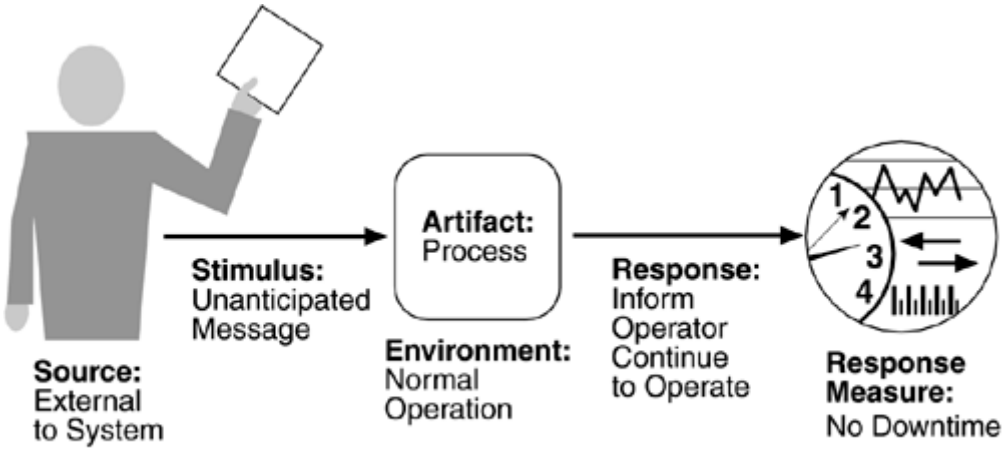
شکل ۴-۵: سناریوی عمومی قابلیت دسترسی

۴-۵-۱-۱- سناریوی قابلیت دسترسی

سناریوی عمومی برای خصوصیت کیفی قابلیت دسترسی در شکل ۴-۵ نمایش داده شده است. در شکل ۴-۵ هر شش بخش مربوط به سناریو قابل مشاهده است. از این سناریو می‌توان برای مشخص کردن سناریوهای عینی استفاده کرد. البته همه سناریوهای عینی نیاز ندارند که کلیه بخش‌های مربوطه را داشته باشند. بخش‌های مورد نیاز برای سناریوهای عینی نتایج سناریو و نوع آزمایشی است که باید انجام شود تا مشخص گردد سناریو قابل دستیابی است.

یک نمونه از سناریوی قابلیت دسترسی که از سناریوی عمومی مشخص شده در شکل ۴-۵ مشتق شده است در شکل ۴-۶ به تصویر کشیده شده است. در این سناریو کلیه قسمت‌هایی که مورد نیاز بوده با مقادیر موردنظر مقداردهی شده است. این سناریو به صورت ذیل تعریف می‌شود:

"در هنگام عملیات عادی، یک فرآیند به صورت ناگهانی و غیره منتظره یک پیام را دریافت می‌کند، فرآیند به اپراتوری دریافت کننده پیام اطلاع می‌دهد و به عملیات خود بدون هیچ وقفه‌ای ادامه می‌دهد."



شکل ۴-۶: یک سناریو عینی برای قابلیت دسترسی

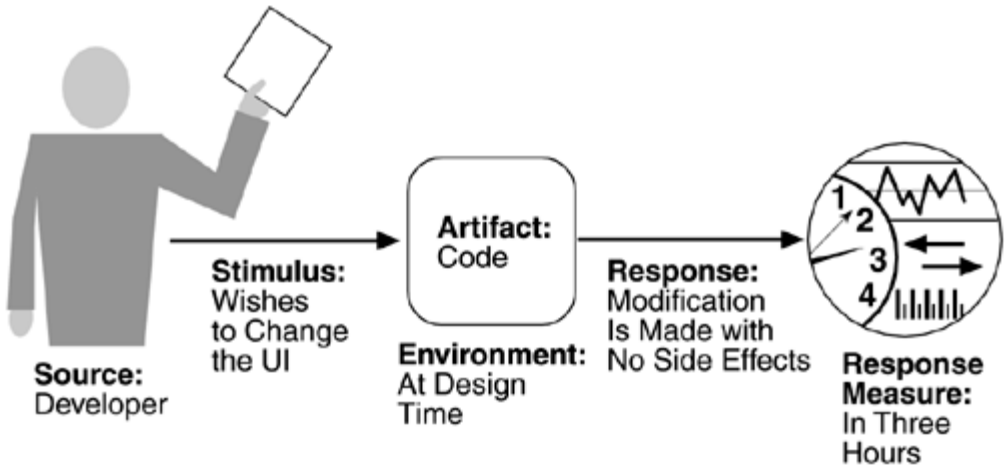
تشخیص منبع تحریک در مورد پاسخ‌های مختلف از اهمیت بالایی برخوردار است. بنابراین در هر سناریو باید مشخص شود که هر پاسخ مربوط به کدام محرک است. به عنوان مثال، وقتی یک سناریو امنیت بررسی می‌شود بسیار مهم است که مشخص شود که منبع تحریک معتبر است یا نه؟ زیرا به ازای هر کدام پاسخ‌های

مختلفی باید در نظر گرفته شود. علاوه بر منبع تحریک، محیط نیز ممکن است بر روی پاسخ تاثیر بگذارد. به عنوان مثال وقتی یک رویداد در سیستم رخ می‌دهد با توجه به اینکه اگر سیستم در حالت بار اضافه[1] باشد ممکن است پاسخهای متفاوتی دریافت شود.

۴-۵-۱-۲- سناریوی قابلیت اصلاح

یک سناریوی ساده مربوط به قابلیت اصلاح به صورت زیر است:

" یک توسعه‌دهنده خواستار تغییر واسط کاربر به منظور تغییر رنگ پیش زمینه به رنگ آبی است. این تغییر به کد مربوط به زمان طراحی اعمال خواهد شد. اعمال کردن تغییر و آزمایش آن بدون هیچ گونه تاثیر جانبی بر رفتار سیستم کمتر از سه ساعت به طول خواهد انجامید". شکل ۴-۷ سناریوی مربوطه را نشان می‌دهد.



شکل ۴-۷: یک سناریوی نمونه برای قابلیت اصلاح

در یک سیستم، یک مجموعه از سناریوهای عینی می‌تواند به عنوان نیازمندیهای خصوصیات کیفی استفاده شود. در این مجموعه هر سناریو به اندازه کافی دربردارنده اطلاعاتی است که قابل فهم برای معمار است و جزئیات پاسخ نیز کاملا آشکار و مشخص هستند در جهت اینکه بتوان آزمایش کرد که آیا سیستم به پاسخ مورد نظر خود دست یافته است یا نه؟ همچنین اگر هنگام استخراج نیازمندیها، یک نیازمندی از دو سناریو حاصل شود می‌توان یکی از آنها را حذف کرد.

[1] Overloaded

در مبحث مربوط به بیان سناریوها، برای هر خصوصیت یک جدول در نظر گرفته می‌شود که در آن برای هر شش بخش موجود در سناریو مقادیر ممکن که مستقل از سیستم هستند، مشخص شده است. یک سناریوی خصوصیات کیفی عمومی به وسیله انتخاب یک مقدار بر هر عنصر تولید می‌شود و یک سناریوی عینی نیز به عنوان بخشی از نیازمندیهای استخراج شده، به وسیله انتخاب یک یا چند موجودیت از ستون‌های جدول و مشخص کردن نتایج آنها ایجاد می‌شود. سناریوهای عینی نقش یکسانی با موارد کاربری دارند. موارد کاربری باعث مشخص کردن نیازهای وظیفه‌مندی می‌شوند ولی سناریوهای عینی باعث مشخص کردن نیازمندیهای خصوصیات کیفی می‌شوند.

۴-۶- سناریوهای خصوصیات کیفی در عمل

سناریوهای عمومی چارچوبی برای تولید تعدادی از سناریوهای مستقل از سیستم و مرتبط با خصوصیت کیفی فراهم می‌کنند. هر کدام از این سناریوها پتانسیل بالقوه‌ای دارند ولی مناسب برای هر سیستم در حال توسعه نیستند. برای اینکه سناریوهای عمومی قابل بکارگیری (مفید) در یک سیستم خاص باشند باید آنها را با توجه به خصوصیات سیستم مورد نظر ایجاد کرد (یعنی آنها را خاص سیستم مورد نظر تعریف کرد). ساختن یک سناریوی عمومی خاص به معنی اینکه سناریوی عمومی را به عبارات کاملا مرتبط با سیستم خاص ترجمه کرد. به عنوان مثال یک سناریوی عمومی به صورت ذیل تعریف می‌شود: "یک درخواست وجود دارد که به منظور ایجاد یک تغییر در وظیفه‌مندی دریافت می‌شود و تغییر باید در یک زمان بخصوص در درون فرآیند توسعه در بازه یک دوره زمانی خاص انجام شود.". بنابراین یک نسخه خاص سیستم می‌تواند به صورت زیر در نظر گرفته شود: "یک درخواست برای اضافه کردن یک مرورگر جدید به سیستم مبتنی بر وب دریافت می‌شود و آن تغییر باید در دو هفته انجام شود."

یک سناریوی عمومی واحد ممکن است تعداد زیادی نسخه خاص سیستم داشته باشد. به عنوان مثال همان سیستمی که باید از مرورگر جدید پشتیبانی می‌کرد ممکن است نیاز به یک نوع جدید تصویری داشته باشد که باید توسط سیستم پشتیانی شود. در نتیجه برای این حالت نیز باید یک سناریوی خاص سیستم وجود داشته باشد.

حال در ادامه به بررسی نحوه ایجاد سناریوهای عمومی برای خصوصیات کیفی قابلیت دسترسی، قابلیت اصلاح، کارایی، امنیت، قابلیت آزمایش و قابلیت استفاده پرداخته می‌شود.

۴-۶-۱- قابلیت دسترسی

خصوصیت کیفی قابلیت دسترسی در ارتباط با شکست سیستم و پیامدهای همراه با آن است. شکست سیستم زمانی رخ می‌دهد که سیستم سرویسهای پیش فرض خود را به طور مناسب ارائه نکند. همچنین وقتی شکست روی می‌دهد نتایج آن توسط کاربران سیستم قابل مشاهده است. حوزه‌های مرتبط با این خصوصیت کیفی عبارتند از :

- چگونه شکست سیستم تشخیص داده می‌شود.
- چگونه شکست سیستم رخ می دهد.
- زمانیکه یک شکست رخ می‌دهد چه اتفاقی می‌افتد.
- تا چه مدت سیستم در حالت عملیاتی نیست (در صورت شکست).
- چگونه می‌توان جلوی شکست را گرفت.
- زمانیکه یک شکست رخ می دهد چه نوع اخطارهای لازم است.

باید توجه کرد که میان دو مفهوم شکست[1] و نقص[2] تفاوت وجود دارد. یک نقص در صورت اینکه مورد بررسی قرار نگیرد و برطرف نشود منجر به شکست می‌شود. یک شکست توسط کاربر سیستم قابل مشاهده است در حالی که نقص توسط کاربر قابل مشاهده نیست. زمانی که یک نقص توسط کاربر قابل مشاهده می‌شود در واقع آن تبدیل به شکست شده است. به عنوان مثال، یک خرابی می‌تواند انتخاب الگوریتم اشتباه برای یک محاسبه باشد که نهایتا نتیجه اشتباه حاصل شده توسط الگوریتم اشتباه منجر به شکست می‌شود.

یکی از مهمترین مفاهیم در زمانیکه سیستم دچار شکست می‌شود مدت زمان لازم برای تعمیر است. از آنجایی که شکست سیستم توسط کاربران قابل مشاهده است بنابراین زمان برطرف کردن شکست برابر است با زمان بین مشاهده شدن شکست توسط کاربر تا زمانی که دیگر شکست قابل مشاهده نیست (سیستم تعمیر می‌شود).

[1] Failure
[2] Fault

تفاوت بین شکست و نقص ارائه کننده مفهوم راهبردهای تعمیر خودکار[1] است. معنی راهبرد تعمیر خودکار اینکه در زمان اجرای سیستم اگر نقصی روی دهد، سیستم بدون آنکه شکست را نشان دهد (دچار شکست شود) آن نقص را اصلاح کند. بنابراین با این راهبرد، دیگر در سیستم شکست وجود نخواهد داشت.

در دسترس بودن یک سیستم برابر است با احتمال آنکه سیستم در زمان مورد نیاز قابل استفاده باشد. این تعریف به صورت زیر نیز قابل بیان است:

$$\alpha = \frac{\text{میانگین زمان شکست}^{2}}{\text{میانگین زمان اصلاح}^{3}+\text{میانگین زمان شکست}}$$

بر اساس این تعریف وقتی گفته می‌شود سیستم دارای دسترس‌پذیری ۹۹.۹٪ است به معنی این است که سیستم به احتمال ۰.۱٪ در زمانی که مورد نیاز است قابل استفاده نیست (یعنی نمی‌تواند سرویس دهد). هنگام محاسبه قابلیت دسترسی معمولا زمانهایی که سیستم باید در حالت خارج از سرویس باشد در نظر گرفته نمی‌شود زیرا در چنین زمانهایی طبق تعریف، سیستم مورد نیاز نیست.

۴-۶-۱-۱- سناریوهای عمومی قابلیت دسترسی

قسمت‌های مختلف یک سناریوی قابلیت دسترسی به شرح زیر تعریف می‌شود (شکل ۴-۵):

- منبع تحریک: بین تحریک‌های داخلی و خارجی مشخص کننده شکست و نقص، تفاوت وجود دارد زیرا پاسخ سیستم به هر یک از این تحریک‌ها می‌تواند متفاوت باشد.
- محرک: یک نقص که می‌تواند محرک در نظر گرفته شود در یکی از دسته‌های زیر قرار می‌گیرد:
  - *Omission* : یک مولفه شکست خورده که نمی‌تواند به ورودی پاسخ دهد.
  - *Crash* : مولفه مکررا دچار نقص از نوع *omission* می‌شود.
  - *Timing* :یک مولفه پاسخ می‌دهد ولی پاسخ زود یا دیر تولید می‌شود.
  - *Response* : یک مولفه پاسخ می‌دهد ولی پاسخ آن صحیح نیست.

[1] Automatic repair strategies
[2] Mean time to failure
[3] Mean time to repair

- *فرآورده:* فرآورده مشخص کننده منبعی است (مانند پردازشگر، کانال ارتباطی، فرآیند یا انباره) که نیاز دارد تا حد بالایی قابل دسترس باشد.
- *محیط:* وضعیت سیستم وقتی نقص یا شکست اتفاق می‌افتد ممکن است پاسخ سیستم را تحت تاثیر قرار دهد. به عنوان مثال اگر سیستم از قبل دچار چندین نقص شده و در حال اجرا در یک حالت غیرطبیعی است احتمالا بهتر است که سیستم را کاملا خاموش کرد اما اگر اولین نقص مشاهده شده باشد می‌توان بعضی ملاحظات را در مورد زمان پاسخ یا نحوه عملکرد سیستم، در نظر گرفت.
- *پاسخ:* برای یک شکست واکنشهای متفاوتی می‌تواند وجود داشته باشد. این واکنش‌ها می‌توانند ثبت کردن شکست، هشدار دادن به کاربر یا دیگر سیستم‌ها، خاموش کردن سیستم‌های خارجی و یا دسترس‌پذیر نبودن در طول تعمیر در نظر گرفته شوند.
- *معیار پاسخ:* معیار پاسخ می‌تواند درصد قابلیت دسترسی، مدت زمان تعمیر و زمانها یا دوره‌هایی که سیستم در آنها قابل دسترس است، در نظر گرفته شود.

جدول ۴-۱ مقدار ممکن برای هر بخش از سناریوی قابلیت دسترسی را نشان می‌دهد.

جدول ۴-۱: تولید سناریوی عمومی قابلیت دسترسی

قسمت سناریو	مقادیر ممکن
منبع	خارج از سیستم، داخل سیستم
محرک	خرابی: *omission, crash, timing, response*
فرآورده	پردازشگرهای سیستم، کانالهای ارتباطی، انبار داده، فرآیندها
محیط	عملیات طبیعی، حالت تنزل[1] (سیستم با ویژگیهای کم)
پاسخ	سیستم باید رویداد را تشخیص داده و حداقل یکی از اقدامات زیر را انجام دهد: – ثبت رویداد – آگاه کردن عناصر مناسب از جمله کاربر و دیگر سیستم‌ها که می‌توانند منابع رویدادها را غیر فعال کنند – در دسترس نبودن برای مدت مشخص (این مدت به بحرانی بودن سیستم بستگی دارد) – ادامه عملکرد خود در حالت طبیعی یا تنزل
معیار پاسخ	زمان دسترس‌پذیری: بازه زمانی که سیستم باید در دسترس باشد زمان تعمیر: بازه زمانی که سیستم می‌توان در حالت تنزل باشد.

[1] Degraded mode

۴-۶-۲- قابلیت اصلاح

قابلیت اصلاح در ارتباط با هزینه تغییرات است. این خصوصیت کیفی بر گرفته از دو مقوله زیر است:

۱. چه چیزی می‌تواند تغییر کند؟ هر جنبه سیستم می‌تواند دارای تغییر باشد. متداولترین تغییرات وظیفه‌هایی است که سیستم انجام می‌دهد. از دیگر جنبه‌های قابل تغییر سیستم می‌توان به سکویی که سیستم بر روی آن قرار دارد (مانند سخت افزار و سیستم عامل)، محیطی که در آن سیستم عمل می‌کند (مثلا، سیستمی که سیستم مورد نظر باید با آن ارتباط داشته باشد یا قراردادهایی که سیستم از آنها برای ارتباط با محیط استفاده می‌کند)، کیفیت‌هایی که سیستم ارائه می‌دهد (کارایی، قابلیت اطمینان و حتی تغییرات آینده سیستم) و ظرفیت سیستم (تعداد کاربران پشتیبانی شده، تعداد عملیات همزمان) اشاره کرد. تغییرات در سیستم می‌تواند شامل حذف، اضافه یا تغییر یکی از جنبه‌های مربوط به سیستم در نظر گرفته شود.

۲. چه زمانی تغییر ایجاد می‌شود و چه کسی این تغییر را ایجاد می‌کند؟ در گذشته متداولترین تغییرات در سطح کد برنامه بود. این بدین معنا است که ایجاد کننده سیستم مجبور به اعمال تغییر بود سپس این تغییر باید آزمایش می‌شد و نهایتا سیستم در یک نسخه جدید استقرار پیدا می‌کرد. اما امروزه سئوال در مورد اینکه چه زمان تغییر ایجاد شود با این سئوال که چه کسی باید تغییر را اعمال کند در هم آمیخته شده است.

به عنوان مثال این که یک کاربر نهایی محافظ صفحه نمایش[1] را تغییر می‌دهد کاملا واضح است که کاربر تغییری در یکی از جنبه‌ها سیستم اعمال می‌کند و روشن است که نوع و گروه این تغییر با تغییری که بتوان سیستم را روی وب نیز داشت، فرق می‌کند. بنابراین تغییرات می‌تواند در پیاده‌سازی (به وسیله تغییر کد برنامه)، در زمان کامپایل (با به کار بردن گزینه‌های زمان کامپایل)، در زمان ایجاد (با انتخاب کتابخانه‌ها)، در زمان پیکربندی نصب[2] (به وسیله یک سری از تکنیک‌ها از جمله منتسب کردن پارامترها) یا در طول اجرا (به وسیله انتساب پارامترها) انجام شود. همچنین یک تغییر ممکن است توسط کاربر یا مدیر سیستم ایجاد شود.

[1] Screen saver
[2] configuration setup

بنابراین زمانی که تغییری مشخص شد، پیاده‌سازی جدید برای آن تغییر باید طراحی، پیاده‌سازی، آزمایش و مستقر شود. کلیه این فعالیت‌ها مستلزم وقت و هزینه است که هر دو آنها با استفاده از روش‌ها خاص قابل اندازه‌گیری است.

۴-۶-۲-۱- سناریوهای عموی قابلیت اصلاح

قسمت‌های مختلف یک سناریوی عمومی قابلیت اصلاح به شرح زیر است (شکل ۴-۷):

- منبع تحریک: منبع تحریک مشخص می‌کند چه کسی تغییرات را ایجاد کرده است (توسعه‌دهنده، مدیر سیستم یا کاربر نهایی). بنابراین کاملا مشخص است که باید مکانیزمهایی وجود داشته باشد که به کاربر نهایی و مدیر سیستم اجازه دهد که سیستم را تغییر دهد. به عنوان مثال در شکل ۴-۷ تغییر به وسیله توسعه‌دهنده اعمال شده است.
- محرک: این بخش مشخص کننده تغییراتی است که ایجاد شده است. یک تغییر می‌تواند اضافه یا حذف کردن یک تابع و یا اینکه اصلاح یک تابع از قبل موجود باشد. همچنین تغییرات می‌تواند در خصوصیات کیفی، ظرفیت و تعداد کاربران نیز وجود داشته باشد.
- *فرآورده*: فرآورده مشخص می‌کند چه چیزی تغییر پیدا می‌کند (مانند وظیفه‌مندی سیستم، سکوی سیستم، واسط کاربری سیستم، محیط سیستم و یا سیستمی که با سیستم موجود در تعامل است). به عنوان مثال در شکل ۴-۷ تغییر مربوط به واسط کاربری است
- *محیط*: این بخش مشخص می‌کند که چه زمانی تغییر می‌تواند ایجاد شود (مانند زمان طراحی، زمان کامپایل، زمان ایجاد و یا زمان اجرا)
- *پاسخ*: هر کسی که می‌خواهد تغییر را انجام دهد باید درک کند که چگونه آن را باید ایجاد، آزمایش و مستقر کند.
- *معیار پاسخ*: همه پاسخ‌های ممکن نیازمند صرف هزینه و زمان است. بنابراین زمان و هزینه دو معیار مهم هستند و همیشه هم امکان‌پذیر نیست که بتوان زمان را پیش‌بینی کرد. ولی معیارهای دیگری همچون تعداد پیمانه‌های تحت تاثیر[1] وجود دارد که می‌توان از آنها استفاده کرد.

جدول ۴-۲ مقادیر ممکن برای هر بخش از سناریوی قابلیت اصلاح را نشان می‌دهد.

[1] Number of modules effected

جدول ۴-۲: تولید سناریوی عمومی قابلیت اصلاح

قسمت سناریو	مقادیر ممکن
منبع	کاربر نهایی، توسعه‌دهنده، مدیر سیستم
محرک	نیاز به اضافه، حذف و یا تغییر دادن وظیفه‌مندی، خصوصیات کیفی و یا ظرفیت سیستم
فرآورده	واسط کاربری سیستم، سکو، محیط، سیستمی که با سیستم هدف در تعامل است
محیط	در زمان اجرا، کامپایل، ساخت و طراحی
پاسخ	– یافتن مکانهایی از معماری که باید تغییر کنند – ایجاد تغییر بدون تحت تاثیر قرار دادن دیگر وظیفه‌مندیها – آزمایش سیستم – مستقر ساختن تغییرات
معیار پاسخ	هزینه بر حسب تعداد عناصر تحت تاثیر قرار گرفته، کار انجام شده و هزینه مالی، میزان تاثیر بر دیگر وظایف یا خصوصیات کیفی سیستم

۴-۶-۳- کارایی

خصوصیت کیفی کارایی در ارتباط با زمان است. رویدادها (وقفه‌ها، پیام‌ها، درخواست‌هایی از سوی کاربران یا گذشت زمان) در سیستم رخ می‌دهند و سیستم باید به آنها پاسخ دهد. انواع مختلفی از مشخصات مربوط به پاسخ و رویداد رسیده (به سیستم) وجود دارد اما اساسا کارایی به این مسئله می‌پردازد که اگر یک رویداد رخ دهد چه مدت زمان طول خواهد کشید که پاسخ سیستم تولید شود (سیستم پاسخ دهد).

یکی از مواردی که سبب پیچیدگی کارایی می‌شود تعداد زیاد منابع و الگوهای ورودی است. رویدادها می‌توانند از درخواستهای کاربر، از سیستم‌های دیگر یا از داخل سیستم صادر شوند. یک سیستم مالی مبتنی بر وب، رویدادها را از کاربرانش دریافت می‌کند (احتمالا به تعداد ده‌ها، صدها و هزاران بار). برای این سیستم پاسخ می‌تواند تعداد تراکنشهایی باشد که می‌تواند در هر دقیقه پردازش کند. در هر صورت، الگوی رویدادها و الگوی پاسخ می‌تواند مشخص شود. این مشخصات زبانی را تشکیل می‌دهند که سناریوهای عمومی کارایی با استفاده از آن ساخته می‌شوند.

یک سناریوی کارایی با ورود یک درخواست به سیستم، در جهت انجام دادن یک سرویس آغاز می‌شود. به منظور برآورده کردن درخواست منابع زیادی از سیستم مصرف می‌شود و ممکن است در حالیکه سیستم به یک

سرویس پاسخ می‌دهد به درخواست‌های دیگر نیز پاسخ دهد. الگوی ورود رویداد می‌تواند به صورت دوره‌ای[1] یا آماری[2] در نظر گرفته شود. یک الگوی ورود رویداد دوره‌ای ممکن است هر ۱۰ میلی ثانیه اتفاق بیفتد. ورود رویداد دوره‌ای معمولا در سیستم‌های بلادرنگ کاربرد دارد. الگوی ورود آماری بدین معنی است که ورود رویدادها با توجه به یک توزیع احتمال است. رویدادها همچنین می‌توانند به صورت پراکنده[3] رخ دهند. یعنی اینکه الگوی ورودی قابل بیان بر اساس مشخصات الگوهای دوره‌ای یا آماری نیست. پاسخ سیستم به یک تحریک در خصوصیت کیفی کارائی می‌تواند در قالب مشخصه‌های زیر تعریف شود:

- دوره عکس العمل[4] : زمان بین ورود تحریک و پاسخ سیستم به آن
- آخرین مهلت[5] در پردازش: به عنوان مثال در کنترل موتور وقتی که سیلندر در یک موقعیت ویژه قرار می‌گیرد، سوخت باید آتش شود.
- توان عملیاتی[6] سیستم: به عنوان مثال تعداد تراکنش‌هایی که سیستم در یک ثانیه پردازش می‌کند را می‌توان توان عملیاتی سیستم در نظر گرفت.
- بی‌ثباتی پاسخ[7]: تغییر در دوره عکس العمل
- تعداد رویدادهایی که به علت مشغول بودن سیستم پردازش نمی‌شوند.
- داده‌هایی که به علت مشغول بودن سیستم از دست می‌روند.

۴-۶-۳-۱- سناریوهای عمومی کارایی

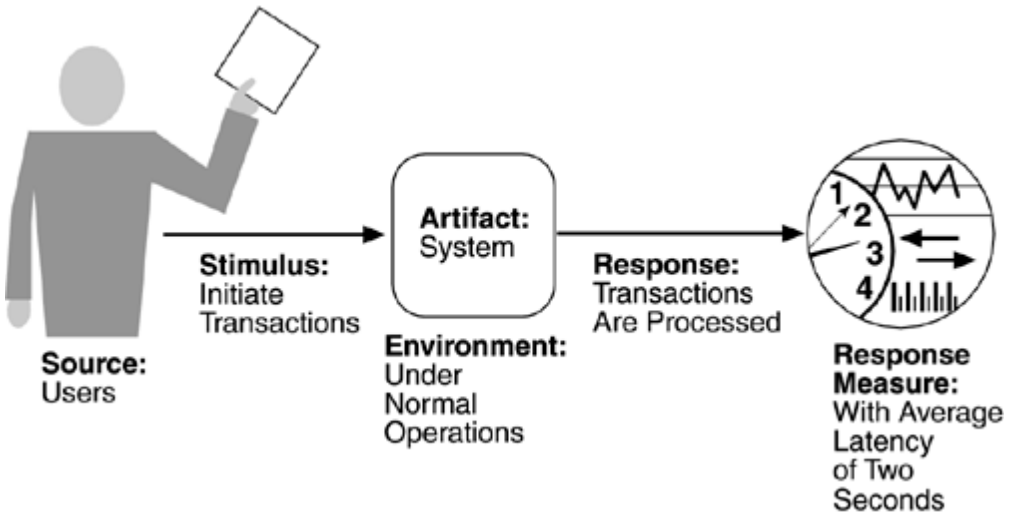
بخش‌های مختلف سناریوی عمومی کارایی به شرح زیر است که یک مثال نشان دهنده آن در شکل ۴-۸ نشان داده شده است. در سناریوی ارائه شده، کاربران ۱۰۰۰ تراکنش را در دقیقه به صورت آماری تحت عملیات نرمالسازی آغاز می‌کنند و این تراکنشها با یک دوره عکس العمل متوسط دو ثانیه‌ای پردازش می‌شوند.

- منبع تحریک: محرک‌ها می‌توانند از منابع خارجی (احتمالا چند منبع) یا منابع داخلی وارد سیستم شوند. در مثال فوق منبع تحریک مجموعه‌ای از کاربران است.

[1] Periodic
[2] Stochastic
[3] Sporadic
[4] Latency
[5] Deadline
[6] Throughput
[7] Jitter of response

- *محرک:* محرک‌ها، ورود رویدادها هستند. الگوی ورود می‌تواند به صورت دوره‌ای، آماری یا پراکنده در نظر گفته شود. در مثال مذکو محرک به صورت آماری با شروع ۱۰۰۰ تراکنش در دقیقه است.
- *فرآورده:* فرآورده در سناریوی کارایی سرویس‌های سیستم است
- *محیط:* سیستم می‌تواند در حالت عملیاتی مختلفی همچون نرمال، حساس[1] و اضافه بار باشد.
- *پاسخ:* سیستم رویدادهای ورودی را پردازش می‌کند. این عمل می‌تواند موجب تغییر در محیط سیستم شود (به عنوان مثال ممکن است حالت سیستم از طبیعی به اضافه بار تغییر کند). در مثال مذکور تراکنش‌ها پردازش می‌شوند.
- *معیار پاسخ:* معیار پاسخ شامل اندازه‌گیری زمانی است که طول می‌کشد تا رویدادهای ورودی پردازش شوند (دوره عکس العمل یا آخرین مهلت که بر اساس آن رویداد باید پردازش شوند)، تنوع در این زمان (بی‌ثباتی)، تعداد رویدادهایی که در یک بازه زمانی خاص می‌تواند پردازش شود (توان عملیاتی) یا یک دسته از رویدادها که نمی‌توانند پردازش شوند (نرخ خرابی، از دست رفتن داده) است. در مثال مذکور تراکنش‌ها باید در یک دوره عکس العمل متوسط دو ثانیه‌ای پردازش شوند.

جدول ۴-۳ عناصر سناریوی عمومی کارایی را نشان می‌دهد



شکل ۴-۸: یک سناریوی نمونه برای خصوصیت کیفی کارایی

[1] Emergence

جدول ۴-۳: تولید سناریوی عمومی کارایی

قسمت سناریو	مقادیر ممکن
منبع	یکی از منابع مستقل، احتمالا در داخل سیستم
محرک	ورود دوره‌ای رویدادها، ورود آماری رویدادها و ورود پراکنده رویدادها
فرآورده	سیستم
محیط	حالت طبیعی، حالت اضافه بار
پاسخ	تحریک پردازش‌ها، تغییر سطح سرویس
معیار پاسخ	دوره عکس العمل، آخرین مهلت، توان عملیاتی، بی‌ثباتی، نرخ از دست رفتن رویداد، میزان از دست رفتن داده

۴-۶-۴- امنیت

امنیت واحدی از میزان قابلیت مقاومت سیستم در برابر استفاده غیرمجاز بوده آن هم در صورتیکه به ارائه سرویس به کاربران مجاز خللی وارد نشود. تلاش برای از بین بردن امنیت، حمله نامیده می‌شود. حمله به شکلهای مختلفی می‌تواند وجود داشته باشد. حمله می‌تواند یک تلاش غیرمجاز برای دستیابی به داده‌ها، سرویس‌ها، به منظور تغییر داده و یا بازداشتن کاربران مشروع از دریافت سرویس باشد. همچنین حمله شامل موضوعاتی چون سرقت پول به وسیله انتقال الکترونیکی با تغییر داده‌های حساس، سرقت شماره کارت اعتباری به وسیله خراب کردن فایلها روی سیستم‌های کامپیوتری و یا حمله توسط ویروسها است. امنیت می‌تواند یک سیستم فراهم کننده عدم انکار[1]، محرمانگی[2]، جامعیت، اطمینان، قابلیت دسترسی و ممیزی[3] در نظر گرفته شود. در ادامه برای هر یک از عبارات معرفی شده تعریف و مثالی ارائه می‌شود.

- *عدم انکار*

ویژگی است که یک تراکنش (مانند دستیابی به داده‌ها، سرویس‌ها و یا تغییر آنها) نمی‌تواند توسط هیچ یک از اجرا کننده‌های آن انکار شود. این بدین معنی است که اگر شما بر روی شبکه اینترنت کالایی را سفارش دادید نمی‌توانید آن را انکار کنید.

[1] Nonrepudiation
[2] Confidentiality
[3] Auditing

- *محرمانگی*

ویژگی است که داده‌ها و سرویس‌ها از دستیابی غیرمجاز محافظت می‌شوند. به عنوان مثال یک شخص نفوذگر نمی‌تواند به بازگشت مالیات بر درآمد شما دسترسی داشته باشد.

- *جامعیت*[1]

ویژگی است که باعث می‌شود داده‌ها و سرویس‌ها به صورت مورد نظر ارائه شوند. به عنوان مثال نمره یک دانش‌آموز بعد از اینکه استاد آن را مشخص کرد تغییر نکرده است.

- *اطمینان*[2]

ویژگی است که اطمینان می‌دهد که شرکت کنندگان در یک تراکنش همانهایی هستند که ادعا می‌کنند. این بدین معنی است که وقتی مشتری یک شماره کارت اعتباری را به یک موسسه اینترنتی ارسال می‌کند موسسه همانی است که مشتریان تصور می‌کنند.

- *قابلیت دسترسی*

ویژگی است که بیان می‌کند سیستم برای کاربران مشروع در دسترس خواهد بود. این بدین معنی است که یک حمله نمی‌تواند شما را از سفارش یک کتاب باز دارد.

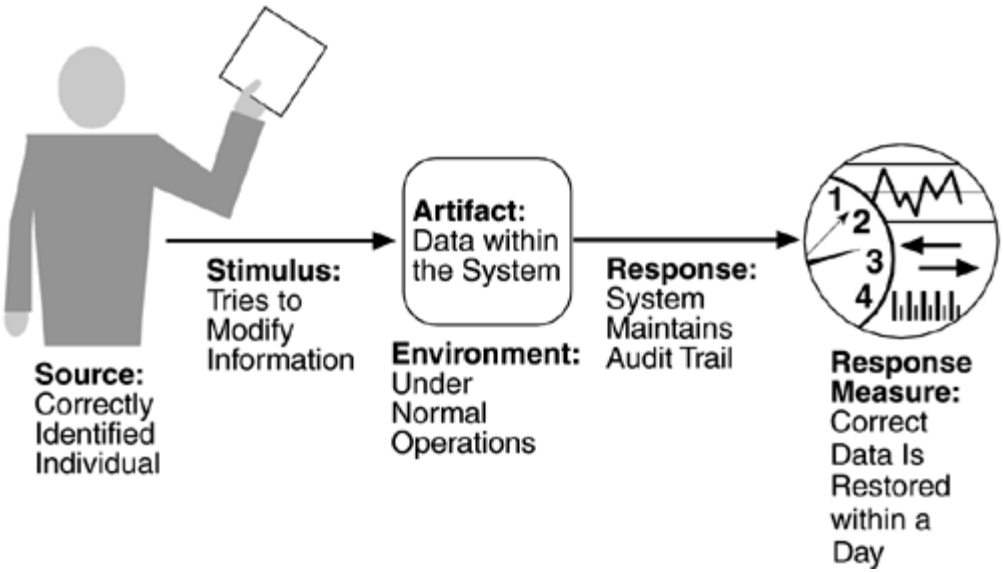
- *ممیزی*

ویژگی است که سیستم فعالیت‌هایی را ردیابی می‌کند تا بتواند در شرایط خاص اثر آنها را خنثی کند. این بدین معنا است که اگر شما پولی را از یک حساب به حساب دیگر منتقل کردید سیستم یک ثبت از این انتقال را حفظ می‌کند.

۴-۶-۴-۱- سناریوهای عمومی امنیت

در شکل ۴-۹ یک نمونه از سناریوی عمومی کارایی را نمایش داده شده است. در شکل ۴-۹ یک منبع تحریک سعی دارد که داده را از خارج سیستم تغییر دهد پس سیستم یک ممیزی نگهداری می‌کند و داده تغییر یافته بعد از یک روز به مقدار قبلی خود بازگردانده می‌شود.

[1] Integrity
[2] Assurance



شکل ۴-۹: یک سناریوی نمونه برای خصوصیت کیفی کارایی

بخشهای مختلف یک سناریوی عمومی کارایی به شرح زیر هستند:

- منبع تحریک: منبع حمله انسان یا سیستم است. منبع یا از قبل شناسائی شده (به صورت صحیح یا ناصحیح) یا اینکه هنوز شناسائی نشده است. اگر منبع حمله، پرانگیزه[1] است در این صورت فعالیت‌های دفاعی مانند "ما می‌دانیم که شما چه کسی هستید و شما را مورد پیگرد قانونی قرار می‌دهیم" احتمالا کارساز نخواهد بود. در چنین حالاتی انگیزه کاربر مهم است.

اگر منبع به منابع وسیعی دسترسی دارد (مثل یک دولت) در این صورت اقدامات دفاعی بسیار مشکل خواهد بود. حمله یک نوع دستیابی غیرمجاز، تغییر سرویس یا عدم اجازه دستیابی به سرویس نیز بشمار می‌رود. مشکلات در زمینه امنیت، اجازه دسترسی دادن به کاربران قانونی و تعیین مشروعت آنهاست. اگر منظور از امنیت تنها جلوگیری از دستیابی به یک سیستم باشد در این صورت عدم اجازه دادن به کلیه دستیابی‌ها به سیستم یک اقدام دفاعی موثر در این مورد است.

[1] High motivated

- محرک: حمله یا تلاشی برای از بین بردن امنیت است. محرک را می‌توان یک شخص یا سیستم غیرمجاز که سعی در نمایش اطلاعات، تغییر و یا حذف اطلاعات، دستیابی به سرویس‌های سیستم یا کاهش دسترس‌پذیری سرویس‌های سیستم در نظر گرفت.
- *فرآورده*: هدف حمله می‌تواند سرویس‌های سیستم و یا داده‌های درونی سیستم باشد.
- *محیط*: حمله می‌تواند زمانی که سیستم بر خط یا برون خط است انجام شود. همچنین حمله می‌تواند از پشت یک دیوار آتشی یا از یک شبکه باز اتفاق بیفتد.
- *پاسخ*: استفاده از سرویس‌ها بدون تشخیص هویت یا جلوگیری از کاربران قانونی در استفاده از سرویس‌ها، هدفی متفاوت از دیدن داده‌های حساس یا تغییر آنها است. بنابراین سیستم باید کاربران قانونی را شناسائی کرده و به آنها مجوز دستیابی به داده‌ها و سرویس‌ها را اعطا کند و از دستیابی کاربران غیرقانونی جلوگیری کرده و دستیابی غیرمجاز را گزارش دهد.

سیستم نه تنها باید دسترسی کاربران قانونی را فراهم کند بلکه باید از ارائه خدماتی از جمله اعطا و بازپس‌گیری حقوق دستیابی نیز حمایت کند. یکی از تکنیک‌ها در جلوگیری از حمله ایجاد ترس مجازات شدن به وسیله نگهداری یک رد ممیزی از تلاشهای دسترسی و تغییرات است. یک تکنیک رد ممیزی همچنین در اصلاح یک حمله موفق نیز مفید است. در شکل ۴-۴ یک رد ممیزی نگهداری شده است.

- *معیار پاسخ*: معیارهای پاسخ سیستم شامل دشواری برخورد کردن با حملات متعدد و سخت بودن بازیابی تاثیرات آنها است. در مثال مذکور، رد ممیزی این امکان را می‌دهد که حساب‌هایی که از آنها پول اختلاس شده است به حالت اولیه خود بازگردانده شود. البته اختلاس کننده همچنان پول را در اختیار دارد و باید مورد پیگرد قانونی قرار داده شده و پول از او باز پس گرفته شود اما این مقوله خارج از حیطه سیستم کامپیوتری است.

جدول ۴-۴ نشان دهنده سناریوهای عمومی امنیت است.

جدول ۴-۴: تولید سناریوهای عمومی امنیت

قسمت سناریو	مقادیر ممکن
منبع	فرد یا سیستمی که به درستی یا به اشتباه شناسائی شده یا شناسائی نشده و به منابع به صورت محدود یا گسترده دسترسی دارد. همچنین اینکه سیستم یا شخص می‌تواند داخلی یا خارجی باشد.
محرک	تلاش می‌کند تا داده‌ها را نمایش دهد، انها را حذف کند یا تغییر دهد، به سرویسهای سیستم دسترسی داشته باشد، دسترسی به سرویسهای سیستم را کاهش دهد.
فرآورده	سرویسهای سیستم، داده‌های درون سیستم
محیط	برخط یا برون خط، متصل یا گسسته
پاسخ	– کاربر تشخیص هویت می‌شود – هویت کاربر پنهان می‌ماند – دستیابی به داده‌ها و سرویسها بلوکه می‌شود – اجازه دستیابی به داده‌ها و سرویس‌ها داده می‌شود – اعطا یا بازپس‌گیری حقوق دستیابی – ثبت دستیابی/تغییر یا تلاش برای دستیابی/تغییر داده‌ها و سرویسها به وسیله شناسه – ثبت داده‌ها به شکل غیرقابل خواندن – تشخیص درخواست اولویت بالا برای سرویسها و آگاه کردن کاربر یا سیستم دیگر و محدود کردن دستیابی به سرویسها
معیار پاسخ	– زمان/میزان کار/منابع لازم برای اندازه گیری امنیت – احتمال تشخیص حمله – احتمال شناسائی افراد مسئول در مورد حمله یا دستیابی/تغییر داده‌ها و سرویسها – درصد سرویسهایی که حتی با وجود حمله در دسترس هستند – بازیابی داده‌ها و سرویسها – میزان خرابی داده‌ها و سرویسها و میزان دستیابی‌های مشروع که از آنها جلوگیری شده است

۴-۶-۵- قابلیت آزمایش

قابلیت آزمایش نرم‌افزار به میزان سهولتی که نرم‌افزار نقص‌های خود را از طریق آزمایش نشان می‌دهد، اشاره دارد. حداقل ۴۰٪ هزینه ایجاد سیستم‌های نرم‌افزاری مربوط به هزینه انجام آزمایش است بنابراین اگر معمار نرم‌افزار بتوان این هزینه را کاهش دهد تاثیر بسزایی در هزینه تولید خواهد داشت.

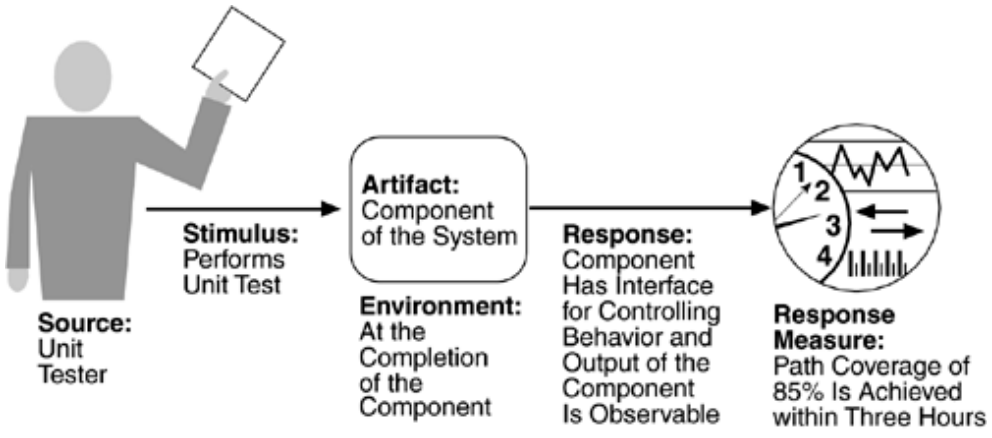
قابلیت آزمایش ارجاع به احتمال شکست سیستم در اجرای بعدی در زمان آزمایش دارد آن هم با فرض اینکه حداقل یک نقص در سیستم وجود دارد. در عمل، محاسبه این احتمال ساده نبوده بنابراین در موقع بحث پیرامون معیارهای پاسخ از معیارهای دیگر استفاده می‌شود. به منظور آزمایش صحیح باید حالت هر مولفه و ورودیهای آن را کنترل کرده و سپس خروجی‌های آن را مشاهده و مورد بررسی قرار داد که معمولا این عمل از طریق یک نرم‌افزار کنترل کننده آزمایش[1] انجام می‌شود.

عمل آزمایش آخرین مرحله از بخش‌های مختلف چرخه حیات سیستم است که توسط انواع توسعه‌دهندگان، آزمایشگرها و بازبین کننده‌ها انجام می‌شود. در واقع قسمت‌های مربوط به کد، طراحی و یا کل سیستم می‌توان مورد آزمایش قرار بگیرد. معیار پاسخ برای قابلیت آزمایش نیز میزان کارا بودن آزمایش‌ها در کشف نقص‌ها و مدت زمان لازم برای انجام یک آزمایش مطلوب برای پوشش یک سطح مناسب از عملکردهای سیستم، در نظر گرفته می‌شود.

۴-۶-۵-۱- سناریوهای عمومی قابلیت آزمایش

شکل ۴-۱۰یک مثال از سناریوی قابلیت آزمایش در ارتباط با کارایی آزمایش واحد[2] را نشان می‌دهد. سناریو به این صورت است که: آزمایش کننده واحد یک آزمایش بر روی یک مولفه کامل شده سیستم انجام می‌دهد که واسطهایی را برای کنترل رفتارش و مشاهده خروجی خود فراهم کرده است، ۸۰٪ پوشش مسیر در عرض سه ساعت انجام می‌شود.

[1] Test harness
[2] Unit test



شکل ۴-۱۰: سناریوی نمونه برای قابلیت آزمایش

قسمت‌های سناریوی عمومی قابلیت آزمایش به شرح زیر هستند:

- منبع تحریک: عمل آزمایش توسط آزمایش کننده واحد[1]، آزمایش کننده یکپارچگی[2]، آزمایش کننده سیستم[3] یا کاربر انجام می‌شود. یک آزمایش طراحی می‌تواند توسط توسعه‌دهندگان یا گروه‌های خارجی دیگر انجام شود. در مثال مذکور آزمایش توسط یک آزمایش کننده انجام می‌شود.
- محرک: محرک در آزمایش این است که یک فرسنگ‌شمار[4] در فرآیند تولید ملاقات شود (رسیدن به فرسنگ شما). این می‌تواند تکمیل یک تحلیل یا طراحی، تکمیل یک کد مانند کد کلاس، تکمیل یک زیر سیستم و یا تکمیل کل سیستم در نظر گرفته شود. در مثال مذکور آزمایش با کامل شدن یک واحد کد آغاز می‌شود.
- *فرآورده*: طراحی، قسمتی از کد و یا کل سیستم می‌تواند فرآورده‌های آزمایش در نظر گرفته شود. در مثال مذکور یک واحد کد آزمایش می‌شود.
- *محیط*: آزمایش می‌تواند در زمان طراحی، ایجاد، کامپایل و یا زمان استقرار انجام شود. در مثال مذکور آزمایش در زمان توسعه انجام می‌شود.

[1] Unit testers
[2] Integration testers
[3] System testers
[4] Milestone

- *پاسخ:* از آنجایی که قابلیت آزمایش مربوط به قابلیت مشاهده[1] و قابلیت کنترل[2] است پاسخ دلخواه آن است که سیستم برای آنجام آزمایش مورد نظر قابل کنترل بوده و پاسخ برای هر آزمایش قابل مشاهده باشد. در مثال مذکور واحد می‌تواند کنترل شده و پاسخ آن مشاهده شود.
- *معیار پاسخ:* معیارهای پاسخ شامل درصدی از دستورات که در یک آزمایش اجرا شده‌اند، طول بزرگترین زنجیره آزمایش[3] (معیاری برای سنجش دشواری آزمایش) و تخمین احتمال پیدا کردن نقص‌های اضافی است. در مثال مذکور معیار آزمایش درصد پوشش دستورات اجرای است.

جدول ۴-۵ نشان دهنده جدول تولید سناریو عمومی قابلیت آزمایش است.

جدول ۴-۵: تولید سناریوهای عمومی قابلیت آزمایش

قسمت سناریو	مقادیر ممکن
منبع	- توسعه‌دهنده واحد - یکپارچه‌ساز افزایشی[4] - بازبین‌کننده سیستم - آزمایش‌کننده پذیرش مشتری[5] - کاربر سیستم
محرک	تحلیل، معماری، طراحی، کلاس، کامل شدن یکپارچگی زیر سیستم، تحویل سیستم
فرآورده	بخشی از طراحی، بخشی از کد، کل برنامه
محیط	در زمان طراحی، در زمان توسعه، در زمان کامپایل، در زمان استقرار
پاسخ	فراهم کردن دسترسی به مقادیر حالت، فراهم کردن مقادیر محاسبه شده، آماده کردن محیط آزمایش
معیار پاسخ	- درصد دستورات اجرا شده - احتمال شکست در صورتیکه نقص وجود داشته باشد - زمان انجام آزمایشها - طول بزرگترین زنجیره وابستگی یک آزمایش - مدت زمان لازم برای آماده کردن محیط آزمایش

[1] Observability
[2] Controllability
[3] Longest test chain
[4] Increment integrator
[5] Client acceptance tester

۴-۶-۶- قابلیت استفاده

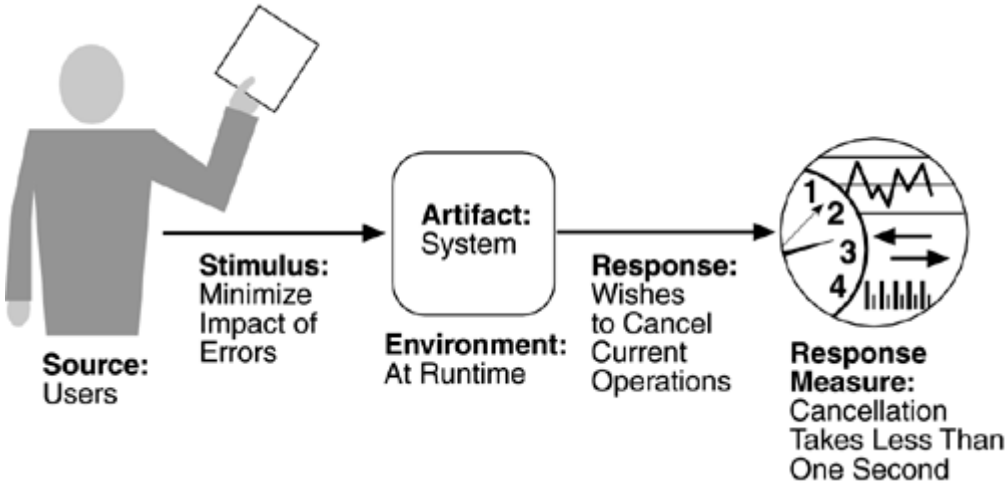
قابلیت استفاده در ارتباط با این موضع است که تا چه اندازه برای کاربر آسان است که یک وظیفه دلخواه خود را انجام دهد و انواع پشتیبانی‌هایی که سیستم در جهت انجام آن فراهم می‌کند، چیست؟ این موضع به حوزه‌های زیر تقسیم می‌شود:

- *یادگیری ویژگیهای سیستم*[1]: اگر کاربر با یک سیستم خاص یا یک جنبه خاص از سیستم نا آشنا است در این صورت سیستم چه کاری می تواند انجام دهد تا عمل یادگیری آسانتر شود؟
- *استفاده از یک سیستم به صورت کارا*[2]: سیستم چه کاری می‌تواند انجام دهد که کاربر را در انجام عملیاتش کارآمدتر کند؟
- *کاهش تاثیر خطاها*[3]: سیستم چه کاری می‌تواند انجام دهد که خطای کاربر دارای کمترین تاثیر باشد؟
- *تطبیق سیستم با نیاز کاربر*[4]: چگونه می‌توان کاربر (یا خود سیستم) را به گونه‌ای تنظیم کرد که وظایف کاربر با آسانی بیشتر انجام شود.
- *افزایش اطمینان و رضایت*[5]: سیستم چه کاری می‌تواند انجام دهد تا به کاربر اطمینان خاطر دهد که عمل صحیح انجام شده است؟

۴-۶-۶-۱- سناریوهای عمومی قابلیت استفاده

شکل ۴-۱۱ یک مثال از سناریوی قابلیت استفاده را نمایش می‌دهد. در این سناریو یک کاربر خواهان کاهش تاثیر خطایی است که از لغو یک عملیات در زمان اجرا ناشی می‌شود. لغو کردن عملیات باید در زمان کمتر از یک ثانیه انجام شود.

[1] Learning system features
[2] Using a system efficiently
[3] Minimizing the impact of errors
[4] Adapting the system to user needs
[5] Increasing confidence and satisfaction



شکل ۴-۸: سناریوی نمونه برای قابلیت استفاده

قسمت‌های سناریوی عمومی قابلیت استفاده به شرح زیر هستند:

- *منبع تحریک*: کاربر نهایی همیشه منبع تحریک است.
- *محرک*: کاربر نهایی است که می‌خواهد از سیستم به صورت کارا استفاده کند، استفاده از سیستم را فراگیر کند، تاثیر خطاها را حداقل کند و هنگام کار با سیستم احساس راحتی کند.
- *فرآورده*: فرآورده همیشه خود سیستم است.
- *محیط*: فعالیت‌های کاربر که مربوط به قابلیت استفاده است همیشه در زمان اجرا یا در زمان پیکربندی سیستم انجام می‌شود.
- *پاسخ*: سیستم باید ارائه کننده ویژگیهای مورد نیاز به کاربر باشد یا نیازهای کاربران را پیش‌بینی کند.
- *معیار پاسخ*: پاسخ سیستم مبتنی بر فاکتورهای زمان فعالیت، تعداد خطاها، تعداد مسائل حل شده، رضایت کاربر، بهره‌وری دانش کاربر[1]، نسبت عملیات موفق به کل عملیات و میزان زمان/داده از دست رفته هنگام وقوع خطا، اندازه‌گیری می‌شود.

جدول ۴-۶ تولید سناریوی عمومی قابلیت استفاده را نشان می‌دهد.

[1] Gain of user knowledge

جدول ۴-۶: تولید سناریوهای عمومی قابلیت استفاده

قسمت سناریو	مقادیر ممکن
منبع	کاربر نهایی
محرک	– یادگیری ویژگیهای سیستم – استفاده از سیستم به صورت کارا – کاهش تاثیر خطاها – تطبیق دادن سیستم – احساس راحتی هنگام کار با سیستم
فرآورده	سیستم
محیط	در زمان اجرا یا در زمان پیکربندی
پاسخ	سیستم فراهم کننده یک یا چند مورد از پاسخهای زیر است: – حمایت از"یادگیری ویژگیهای سیستم": راهنمای سیستم حساس به زمنیه، واسط آشنا به کاربر و واسط کاربری که در یک زمینه ناآشنا قابل استفاده است. – حمایت از "استفاده کارا از سیستم": تجمیع داده‌ها، استفاده مجدد از داده‌ها و فرمانهای از قبل وارد شده، پشتیبانی از جستجوی کارا در یک صفحه، نماهای متفاوت با عملیات سازگار، جستجوی فراگیر و فعالیت‌های همزمان – حمایت از "کاهش تاثیرات خطاها": حنثی کردن، لغو کردن، بازیابی از حالت شکست، تشخیص و اصلاح خطای کاربر، بازیافتن کلمه عبور فراموش شده، بازبینی منابع سیستم – حمایت از "تطبیق سیستم": قابلیت سفارشی‌سازی و بین المللی کردن – حمایت از "احساس راحتی": نمایش حالت سیستم و هماهنگ کارکردن با کاربر
معیار پاسخ	زمان وظیفه، تعداد خطاها، تعداد مسائل حل شده، رضایت کاربر، نسبت عملیات موفق به کل عملیات و میزان از دست رفتن زمان/داده

تا به اینجا خصوصیات کیفی سیستمی به صورت کلی معرفی شدند. تعداد دیگری از خصوصیات کیفی در تحقیقات و منابع استاندارد مهندسی نرم‌افزار مشاهده می‌شوند که در اینجا ذکر نشده‌اند زیرا بسیاری از آنها در همین سناریوهای تعریف شده می‌گنجند. برای مثال قابلیت توسعه خصوصیت مهمی است که در این منابع به چشم می‌خورد اما در بخش خصوصیات کیفی این خصوصیت با خصوصیت کیفی قابلیت تغییر پوشانده می‌شود (مثلا تعداد کاربرانی که سیستم از آنها پشتیبانی می‌نماید). قابلیت حمل نیز که از دیگر خصوصیات مهم است به عنوان یک سکویی از قابلیت تغییر در نظر گرفته شده است. با توجه به کلیه مطالب مطرح شده، اگر در

یک سازمانی یک خصوصیت کیفی از اهمیت بالایی برخوردار است در آن صورت برای آن باید سناریوی جداگانه‌ای ایجاد شود و شش قسمت در نظر گرفته شده برای هر سناریو برای آن بررسی و ایجاد شود.

۴-۶-۷- مفاهیم ارتباطی با سناریوهای عمومی

یکی از کاربردهای سناریوهای عمومی فراهم آوردن ارتباط بین ذینفعان است. از آنجایی که هر خصوصیت کیفی برای خودش یک سری لغاتی دارد که از طریق آن مفاهیم پایه خود را تشریح می‌کند و همچنین عبارت‌های متفاوتی نیز دارند که به رویداد یا مفاهیم یکسان اشاره می‌کنند، بنابراین این امر موجب برقراری ارتباط نامناسب بین ذینفعان می‌شود. برای مثال در موقع بحث بر روی خصوصیت کیفی کارایی یک ذینفع ذکر می‌کند که کاربر در هنگام برخورد با تاخیر یک پاسخ نمی‌داند که آیا آن ناشی از کار خودش است یا سیستم؟ و یا اینکه باید کاری را انجام دهد یا نه؟ در جهت تسهیل درک یک چنین مباحثی باید تصمیمات معماری مورد بحث قرار بگیرند مخصوصا پیرامون مصالحه‌های که از قبل انجام شده است.

جدول ۴-۷ انواع محرکهای ممکن برای خصوصیات کیفی به همراه مفاهیم متفاوت برای هر یک را نشان می‌دهد. بعضی از محرک‌ها در زمان طراحی اتفاق می‌افتند و بعضی‌ها هم در زمان قبل از اجرا به وقوع می‌پیوند. مشکل اصلی در اینجا برای یک معمار درک اینکه کدام یک از محرک‌ها منجر به رخ دادن حالات یکسان می‌شوند، کدام یک تجمیعی از دیگر محرک‌ها است و کدام یک مستقل هستند. بنابراین یک مرتبه که ارتباطات روش و مشخص شوند معمار می‌تواند هر یک از آنها را به ذینفعان خود مرتبط کرده و با استفاده از زبان آشنا برای خودشان تشریح کند.

نکته دیگر اینکه امکان مشخص کردن ارتباط بین محرک‌ها به صورت عمومی وجود ندارد زیرا آنها وابسته به محیط هستند. یک کارایی می‌تواند اتمیک یا می‌تواند تجمیعی از رخدادهای سطح پائین باشد، یک شکست ممکن است یک رویداد منفرد از کارایی و یا مجموعی از آنها در نظر گرفته شود. برای مثال یک چنین حالتی زمانی اتفاق بیفتد که یک سرویس‌دهنده و سرویس‌گیرنده بین خودشان چندین پیام را ردوبدل می‌کنند پس هر کدام از اینها یک رویداد اتمیک از دیدگاه کارایی هستند.

جدول ۴-۶: محرک‌های خصوصیات کیفی

خصوصیات کیفی	محرک‌ها
قابلیت دسترسی	رویداد غیرمنتظره، عدم رخ داد رویداد مورد انتظار
قابلیت اصلاح‌پذیری	درخواست برای اضافه کردن/حذف/تغییر وظیفه‌مندی، سکو، خصوصیت کیفی یا ظرفیت
کارایی	دوره‌ای، آماری، پراکنده
امنیت	هنگام تلاش برای نمایش، تغییر، حذف، دسترسی و کاهش قابلیت سرویس‌های سیستم
قابلیت آزمایش	کامل شدن فاز توسعه سیستم
قابلیت استفاده	نیاز داشتن به یادگیری خصوصیات سیستم، استفاده از کارآمدی سیستم، کاهش تاثیر خطاها، تطبیق سیستم و احساس راحتی

۴-۷- خصوصیات کیفی حرفه

علاوه بر خصوصیات کیفی که به صورت مستقیم در یک سیستم اعمال می‌شوند تعداد اهدافی کیفیتی حرفه وجود دارد که معماری سیستم را تشکیل می‌دهد. این اهداف حول هزینه، زمانبندی، بازار و بازاریابی هستند.

- *زمان عرضه محصول به بازار:* اگر یک فشار رقابتی وجود داشته باشد و یا فرصت کمی برای تولید یک سیستم یا محصول وجود داشته باشد زمان توسعه و تولید از اهمیت زیادی برخوردار می‌شود. در این مورد مسئله این است که از عناصر موجود استفاده شود و یا آنها خریداری شوند. اغلب اوقات با استفاده از عناصر از پیش ساخته شده زمان تولید کاهش می‌یابد (مانند استفاده از محصول با تولید انبوه تجاری *(COTS[1])* یا استفاده از عناصر موجود در پروژه‌های قبلی). توانایی اضافه کردن یا استقرار یک زیر مجموعه از سیستم به نحوه‌ی تجزیه سیستم به عناصر مختلف بستگی دارد.
- *هزینه و سودآوری :* به طور طبیعی برای هر سیستم بودجه‌ای در نظر گرفته می‌شود که نباید هزینه‌های مربوط به تولید از آن تجاوز نمایید. معماریهای مختلف منجر به هزینه تولید متفاوتی می‌شوند. مثلاً معماریهایی که تکیه بر تکنولوژیهای جدیدتر دارند به نسبت معماریهای سنتی‌تر هزینه بیشتری خواهند داشت.

[1] Commerical Off-the-Shelf

- *طول عمر تعین شده برای سیستم:* اگر یک سیستم بخواهد طول عمر زیادی داشته باشد خصوصیاتی نظیر قابلیت اصلاح، قابلیت حمل و قابلیت توسعه و گسترش در محیطهای مختلف از اهمیت بیشتری برخوردار می‌شوند. لذا از یک طرف ایجاد زیر ساختهایی که بتوانند چنین خصوصیاتی را پشتیبانی نمایند مانند ساختن لایه‌ای متفاوت برای برآوردن قابلیت حمل و قابلیت اصلاح، زمان زیادی برای تولید نیاز دارد و از طرف دیگر یک محصول با قابلیت اصلاح، حمل و توسعه طول عمر بیشتر خواهد داشت و با ارزش‌تر خواهد بود.
- *بازار فروش نهایی:* برای رسیدن به نرم‌افزاری با فروش انبوه یا نرم‌افزارهای چند منظوره باید سکویی که سیستم بر روی آن اجرا می‌شود را به گونه‌ای در نظر گفت که نیازهای آینده بازار را بتوان تامین کند. بنابراین قابلیت حمل و وظیفه‌مندی، کلیدی برای موفقیت یک محصول در بازار هستند. همچنین خصوصیات کیفی دیگر مانند کارایی، قابلیت اطمینان و قابلیت استفاده در موفقیت پروژه نقش دارند. به منظور ورود به یک بازار بزرگ با مجموعه‌ای از محصولات مرتبط نیاز به یک روش خط تولید است که در آن یک سری نیازمندهای هسته سیستم در نظر گرفته شوند و سایر نیازمندها به عنوان لایه‌ای در اطراف آنها خصوصیات جدید فراهم شوند.
- *زمانبندی تحویل قسمت‌های مختلف:* اگر یک محصول با یک سری نیازمندیهای اولیه و بسیاری از ویژگیهای مرتبط که باید بعدا اضافه شوند در نظر گرفته شود انعطاف‌پذیری و قابلیت سفارشی‌سازی معماری از اهمیت بیشتری برخوردار می‌شود. مخصوصا اینکه سیستم باید با استفاده از گسترشهای ساده و حداقل تناقضات ذهنی ساخته شود.
- *یکپارچگی با سیستم‌های موروثی:* اگر سیستم جدید بخواهد با سیستم‌های موجود تجمیع شود، توجه زیادی به مکانیزمهای تعریف درست تجمیع لازم است. این خصلتی است که به اندازه فروش مهم است و در عین حال به طور ضمنی با مسائلی که معماری ایجاب می‌کند نیز در ارتباط است

۴-۸- خصوصیات کیفی معماری

علاوه بر خصوصیات کیفی سیستم و خصوصیات مرتبط با محیط حرفه که سیستم بر اساس آن تولید می‌شود خصوصیات کیفی دیگری وجود دارند که مستقیما با خود معماری در ارتباط هستند و رسیدن به آنها از اهمیت زیادی برخوردار است.

- *جامعیت مفهومی*[1]: جامعیت مفهومی یک نمای زیربنایی است که طراحی سیستم در سطوح مختلف را یکپارچه و متحد می‌کند.
- *کمال و صحت*[2]: کمال و صحت برای معماری ضروری است زیرا به کلیه نیازمندهای سیستم و محدودیت‌های منابع زمان اجرا اجازه تحقق یافتن را می‌دهد.
- قابلیت ساخت[3]: قابلیت ساخت اجازه می‌دهد که سیستم به وسیله تیم توسعه در دسترس و در یک روش زمانبندی شده کامل شود (در زمان مقرر) و برای تغییرات آتی در مسیر تولید سیستم راه را باز بگذارد. این مسئله به تجزیه کردن ماژولها به صورت مناسب و تخصیص آنها به تیم‌های تولید و محدود کردن ارتباط بین ماژولها مربوط می‌شود. در واقع هدف اصلی ایجاد توازن در کار است و معمولا به دلیل اینکه قابلیت ساخت با لغات هزینه و زمان اندازه‌گیری می‌شود رابطه‌ای بین قابلیت ساخت و ماژولهای هزینه مختلف وجود دارد.

[1] Conceptual integrity
[2] Correctness and completeness
[3] Buildability

فهرست مطالب

فصل پنجم

دستیابی به خصوصیات کیفی

در این فصل هدف مشخص کردن یک سری گامها، راهبردها و تاکتیک‌ها در جهت دستیابی به خصوصیات کیفی (مطرح شده در فصل چهارم) در سطح معماری است. به عبارت بهتر هدف بررسی ارتباط بین تصمیمات معماری و نیازمندیهای کیفی است.

۵-۱- مقدمه

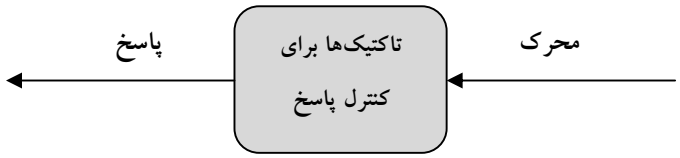
در فصل چهارم یک سری خصوصیات کیفی معرفی شد که این خصوصیات کیفی به نوبه خود در قالب سناریوها بیان شدند. سناریوها اگر چه باعث شناسائی نیازمندیهای خصوصیات کیفی می‌شوند ولی به فهم چگونگی دستیابی به خصوصیات کیفی هیچ کمکی نمی‌کنند (نیازمندیهای کیفی پاسخهایی از نرم‌افزار را مشخص می‌کنند که در قالب آنها اهداف تجاری[1] تحقق می‌یابد). به منظور دستیابی به خصوصیات کیفی نکته مهم تاکتیک‌هایی[2] است که به وسیله معمار استفاده می‌شوند.

مجموعه‌ای از تاکتیک‌ها الگویی را ایجاد می‌کند که با استفاده از آن معمار، طراحی سیستم را انجام می‌دهد (معمار ممکن است برای طراحی سیستم از چندین الگوی طراحی استفاده کند). به عنوان مثال در نظر بگیرید که هدف تجاری ایجاد یک خط تولید باشد. در جهت دستیابی به آن هدف باید اجازه تغییرپذیری در کلاس‌ها یا توابع ویژه داده شود. معمار باید قبل از تصمیم‌گیری در مورد مجموعه‌ای از الگوها برای دستیابی به تغییرپذیری مطلوب، بررسی کند که چه ترکیبی از تاکتیک‌ها باید برای تغییرپذیری اعمال شوند. زیرا تاکتیک‌های انتخاب شده، در آینده تصمیمات معماری را هدایت خواهند کرد (یک الگوی معماری یا راهبرد مجموعه‌ای از تاکتیک‌ها را پیاده‌سازی می‌کند).

تاکتیک یک تصمیم معماری است که کنترل پاسخ یک خصوصیت کیفی را تحت تاثیر قرار می‌دهد و مجموعه‌ای از تاکتیک‌ها یک راهبرد معماری را تعریف می‌کنند. الگوی معماری نیز مجموعه‌ای از تاکتیک‌هایی است که در یک روش خاص در کنار هم قرار گرفته‌اند.

یک طراحی سیستم متشکل از مجموعه‌ای از تصمیمات معماری است. بعضی از این تصمیمات کمک می‌کنند تا پاسخ خصوصیات کیفی کنترل شوند (که تاکتیک نامیده می‌شوند) و تصمیمات دیگر نیز دستیابی به قابلیت عملکرد سیستم را تضمین می‌کنند. این ارتباط در شکل ۵-۱ نشان داده شده است. همان طور که در شکل ۵-۱ مشخص است محرک[3] بر یک تاکتیک یا مجموعه‌ای از تاکتیک‌ها اعمال می‌شود و آن تاکتیک‌ها پاسخ مورد نیاز و مطلوب را تولید می‌کنند.

[1] Business goals
[2] Tactics
[3] Stimulus



شکل ۵-۱: ارتباط بین تاکتیک و خصوصیت کیفی

هر تاکتیک یک گزینه طراحی برای معمار است. برای مثال یکی از تاکتیک‌ها برای افزایش قابلیت دسترسی سیستم، افزونگی است. بنابراین افزونگی یکی از گزینه‌هایی است که معمار برای افزایش قابلیت دسترسی سیستم در اختیار دارد. معمولا قابلیت دسترسی بالا از طریق تاکتیک افزونگی به تاکتیک دیگری جهت همزمانی نیاز دارد تا تضمین کند نسخه‌هایی که می‌توانند در صورت خرابی به کار گرفته شوند همگی به روز هستند. با توجه به این مثال می‌توان دو نکته مهم برای تاکتیک‌ها ذکر کرد:

۱. *تاکتیک‌ها می‌توانند تاکتیک‌های دیگر را بهبود دهند*

افزونگی یک تاکتیک است و می‌توان آن را در درون (با کمک) افزونگی داده و افزونگی محاسبات بهبود داد. در نتیجه هر دو نوع افزونگی داده و افزونگی محاسبات نیز تاکتیک خواهند بود. علاوه بر این طراح می‌تواند دوباره هر کدام از این تاکتیک‌ها را بهبود بخشد تا به یک مجموعه‌ای از تاکتیک منسجم‌تر دست پیدا کند.

۲. *الگوها، تاکتیک‌ها را بسته‌بندی[1] می‌کنند*

یک الگویی که از قابلیت دسترسی حمایت می‌کند مطمئنا دربرگیرنده هر دو نوع تاکتیک افزونگی و همزمانی است و امکان این نیز وجود دارد که از نسخه‌های منسجم‌تری از این نوع تاکتیک‌ها استفاده کند.

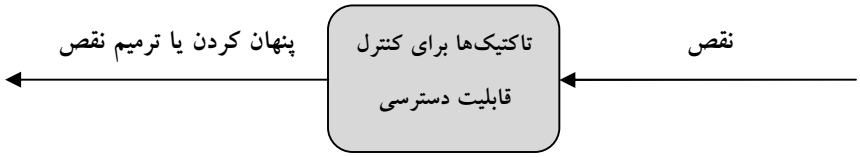
در ادامه برای هر شش خصوصیت کیفی که در فصل چهار معرفی شدند یک سری از تاکتیک‌ها معرفی می‌شود و برای هر تاکتیک نیز توضیحاتی ارائه خواهد شد. سازماندهی نمایش تاکتیک‌ها برای هر خصوصیت کیفی به صورت سلسله مراتبی خواهد بود تا مسیری (حالتی) ایجاد شود که بتوان به راحتی تاکتیک مناسب برای هر خصوصیت کیفی را شناسائی کرد.

[1] Package

۵-۲- تاکتیک‌های قابلیت دسترسی

یک شکست[1] در سیستم زمانی اتفاق می‌افتد که سیستم دیگر سرویسی که در مشخصاتش ذکر شده است را ارائه نکند. وقوع شکست در سیستم توسط کاربر قابل مشاهده است و نقص[2] نیز یکی از عوامل اصلی پدید آورنده شکست در سیستم است (به منظور خوانایی بهتر از این به بعد به جای شکست از کلمه خطا استفاده خواهد شد ولی فراموش نشود که بین نقص، خطا[3] و شکست تفاوت وجود دارد).

تاکتیک‌هایی که برای قابلیت دسترسی مطرح خواهند شد پیرامون جلوگیری از تبدیل شدن نقص به خطا یا حداقل کم کردن تاثیرات آن هستند. این موضوع در شکل ۵-۲ به تصویر کشیده شده است.



شکل ۵-۲: هدف از تاکتیک‌های قابلیت دسترسی

کلیه روشها مرتبط با قابلیت دسترسی به روشهایی از نوع افزونگی، روشهایی از نوع نظارت بر سلامت[4] برای تشخیص نقص و روشهایی از نوع ترمیم[5] برای زمانی که نقص اتفاق می‌افتد، تقسیم می‌شوند. در جهت بررسی این روشها ابتدا تشخیص خطا سپس ترمیم نقص و نهایتا جلوگیری از نقص تشریح خواهند شد.

۵-۲-۱- تشخیص خطا

سه مورد از تاکتیک‌ها برای تشخیص خطاها صدا/انعکاس[6]، ضربان قلب[7] و استثنائات[8] هستند که توضیح پیرامون هر یک به شرح زیر است:

[1] Failure
[2] Fault
[3] Error
[4] Health monitoring
[5] Recovery
[6] Ping/echo
[7] Heartbeat
[8] Exceptions

- *صدا/انعکاس یا Ping/echo*

در این تاکتیک یک مولفه پیامی را تولید می‌کند و انتظار دارد که از مولفه مورد نظر خود در یک زمان مشخص پاسخی را دریافت کند. این تاکتیک می‌تواند در بین گروهی از مولفه‌ها که به صورت دو به دو مسئول یک وظیفه هستند به کار گرفته شود. همچنین سرویس‌گیرنده‌ها با استفاده از این تاکتیک می‌توانند تضمین کنند که یک شیء سمت سرور یا یک مسیر ارتباطی با سرور در یک کارایی مطلوب عمل می‌کند. تشخیص دهنده‌های نقص *"ping/echo"* می‌توانند در یک روش سلسله مراتبی سازماندهی شوند که در آن تشخیص دهنده‌های سطح پائین، فرآیندهای نرم‌افزاری متعلق به یک پردازنده را *ping* می‌کنند و تشخیص دهنده‌های سطح بالا، تشخیص دهنده‌های سطح پائین را *ping* می‌کنند. این نوع سازماندهی از پهنای باند ارتباطی کمتری نسبت به حالتی که در آن یک تشخیص دهنده کل فرآیندها را *ping* می‌کند، برخودار است.

- *ضربان قلب*

در این روش یک مولفه به صورت دوره‌ای یک ضربان قلب (پیام) را منتشر می‌کند و مولفه دیگر منتظر دریافت آن می‌شود. اگر ضربان قلب توسط مولفه منتظر پیام به صورت دوره‌ای دریافت نشود مولفه منتظر پیام متوجه می‌شود که یک خطا اتفاق افتاده است و مولفه مربوط به اصلاح نقص را فراخوانی می‌کند. یک ضربان قلب می‌تواند در بردارنده داده نیز باشد. یعنی علاوه بر اینکه مولفه گیرنده متوجه می‌شود که مولفه مورد نظر خود فعال است داده‌های مورد نیاز خود را نیز از این طریق دریافت می‌کند. به عنوان یک مثال، سیستم *ATM* را در نظر بگیرید که در آن ماشین می‌تواند همراه با پیامی که مشخص کننده فعال بودن خودش است (ضربان قلب) به صورت دوره‌ای لیستی از آخرین تراکنشهای انجام شده را به سرور ارسال کند. بنابراین دیگر این پیام ارسالی فقط یک ضربان قلب نیست زیرا شامل داده‌هایی است که پردازش می‌شود.

- *استثنائات*

در این تاکتیک از استثناء استفاده می‌شود. نحوه عملکرد بدین صورت است که به محض تشخیص یک نقص، یک استثناء صادر می‌شود. بروز استثناء باعث می‌شود سیستم از بروز نقص آگاه شده و حالت خود را به قبل از بروز استثناء تغییر دهد. نهایتا اینکه بازگرداندن سیستم به حالت قبل از بروز نقص باعث جلوگیری از بروز خطا در سیستم می‌شود.

تاکتیک‌های ضربان قلب و *ping/echo* در میان فرآیندهای مجزا عمل می‌کنند ولی تاکتیک استثناء درون یک فرآیند عمل می‌کند. در تاکتیک استثناء، کنترل کننده استثناء معمولا یک تبدیل معنایی انجام می‌دهد که این عمل باعث تبدیل شدن نقص به قالبی می‌شود که می‌تواند پردازش شود.

۵-۲-۲- ترمیم نقص

ترمیم نقص متشکل از فرآیندهای آماده‌سازی برای ترمیم و اعمال ترمیم در سیستم است که بعضی از تاکتیک‌ها در این زمینه به شرح زیر هستند:

- *رای‌گیری*[۱]

در این تاکتیک فرآیندها بر روی پردازنده‌های افزونه در حال اجرا هستند (تعداد پردازنده‌ها زیاد هستند) که هر کدام ورودی یکسان را دریافت می‌کنند و خروجی تولید کرده و آن را به رای‌دهنده ارسال می‌کنند. رای دهنده اگر انحرافی[۲] یا اشتباهی در رفتار هر پردازنده دریافت کنند آن را به عنوان یک نقص در نظر می‌گیرد. الگوریتم رای‌گیری می‌تواند مبتنی بر "قانون اکثریت[۳]" یا "مولفه‌های برتر[٤]" و یا سایر الگوریتم‌ها باشد. این تاکتیک به منظور تصحیح عملیات دارای نقص الگوریتم و یا خطای یک پردازنده استفاده می‌شود و اغلب نیز در سیستم‌های کنترل به کار گرفته می‌شود. اگر تمام پردازنده‌ها یک الگوریتم واحد را به کار ببرند افزونگی فقط نقص یک پردازنده را به جای نقص الگوریتم تشخیص می‌دهد، بنابراین اگر پیامد یک خطا زیاد باشد مولفه‌های افزونه می‌توانند متفاوت در نظر گرفته شوند.

گستره این تنوع[۵] بدین صورت است که نرم‌افزار برای هر مولفه افزونه، توسط تیم‌های مختلف توسعه داده شده و بر روی سکوهای نامشابه اجرا شوند و برای گستره‌های کمتر می‌توان از یک مولفه نرم‌افزار منفرد روی سکوهای نامشابه استفاده کرد. در هر صورت تنوع نیازمند هزینه زیاد برای توسعه و نگهداری است و کاربرد آن نیز در موارد خاص است.

[1] Voting
[2] Deviant
[3] Majority rules
[4] Preferred component
[5] Driversity

- *افزونگی فعال*[1]

در این تاکتیک همه مولفه‌های افزونه به صورت موازی به رویدادها پاسخ می‌دهند. در نتیجه همه آنها در وضعیت مشابه قرار دارند (به روز هستند). این تاکتیک یکی از پاسخ‌های دریافتی از مولفه‌ها را فقط مورد استفاده قرار داده (معمولا اولین مولفه‌ای که پاسخ داد) و مابقی پاسخ‌ها را نادیده می‌گیرد. وقتی یک نقص رخ می‌دهد در زمان بسیار کوتاهی که در حد چند میلی ثانیه است پشتیبان فعال می‌شود (زیرا پشتیبان در آن لحظه فعال است) و سرویس‌های مربوطه را پاسخ می‌دهد. افزونگی فعال اغلب در سیستم‌های سرویس‌گیرنده و سرویس‌دهنده به کار گرفته می‌شود، مانند سیستم‌های مدیریت پایگاه‌داده.

- *افزونگی غیرفعال*[2]

در این تاکتیک یک مولفه به رویدادها جواب می‌دهد و بعد مولفه‌های دیگر را از بروزرسانی‌های لازم آگاه می‌کند تا آنها تغییرات را در خود اعمال کنند. وقتی یک نقص اتفاق می‌افتد قبل از اینکه سیستم، سرویس بازیابی از حالت نقص را شروع کند باید مطمئن شود که وضعیت پشتیبان به روز است. این تاکتیک نیز در سیستم‌های کنترلی مورد استفاده قرار می‌گیرد، مخصوصا در حالتی که ورودی از طریق کانالهای ارتباطی یا از طریق سنسورها است و ضرورت ایجاب می‌کند که در صورت وقوع نقص، مولفه اصلی به مولفه پشتیبان انتقال صورت بپذیرد. در مورد همگام‌سازی این تاکتیک نیز باید مولفه اصلی شروع کننده همگام‌سازی باشد و برای تضمین همزمان نیز می‌تواند از انتشارهای اتمیک[3] استفاده کند.

- یدک[4]

در این تاکتیک یک سکوی محاسباتی یدکی جانشین[5] طوری تنظیم می‌شود که جانشین مولفه‌های زیادی شود که عمل خطا در آنها اتفاق افتاده است. بدین منظور باید عمل راه اندازی مجدد سیستم انجام شود تا پیکربندی مورد نظر به سیستم اعمال شود (یدک فرآیند شروع به کار خاص خود را دارد). نحوه عملکرد بدین صورت است که یک نقطه وارسی از وضعیت سیستم در یک رسانه ماندگار

[1] Active redundancy
[2] Passive redundancy
[3] Atomic broadcasts
[4] Spare
[5] Standby spare computing platform

به صورت دوره‌ای نگهداری می‌شود و تمامی تغییرات در وضعیت سیستم در آن رسانه ثبت می‌شود، به محض وقوع خطا حالت مناسب تشخیص داده شده و یدک جاگزین آن می‌شود. انجام این عمل ممکن است در حدود چند دقیقه طول بکشد.

- *عملیات سایه‌سازی*[1]

یک مولفه از قبل شکست خورده می‌تواند در حالت سایه برای یک مدت کوتاه در حال اجرا باشد تا مطمئن شود که همان رفتاری را دارد که قبل از حالت ترمیم داشت.

- *همگام‌سازی مجدد حالت*[2]

تاکتیک‌های افزونگی فعال و غیر فعال نیاز دارند تا مولفه جایگزین قبل از شروع سرویس خود به روز باشد و لذا همزمانی ضرورت دارد. روش به روزرسانی بستگی به مدت زمان قابل تحمل برای غیرفعال بودن سیستم، اندازه به روزرسانی و تعداد پیامهای مورد نیاز برای به روزرسانی دارد.

- *نقطه وارسی/عقبگرد*[3]

نقطه وارسی یک ذخیره‌سازی از وضعیت سازگار سیستم در رسانه ماندگار است. ثبت حالت سازگار می‌تواند به صورت دوره‌ای یا در موقع پاسخ دادن به یک رویداد خاص انجام شود. بعضی اوقات سیستم در یک روش غیر معمول دچار خطا می‌شود که وضعیت خطا نیز در آن قابل تشخیص است در این حالت باید سیستم به آخرین حالت پایدار خود با توجه به آخرین نقطه وارسی برگردانده شود.

سه تاکتیک عملیات سایه‌سازی، همگام‌سازی مجدد حالت و نقطه وارسی/عقبگرد متکی بر جایگزینی مجدد مولفه[4] هستند. یعنی زمانی که یک خطا در مولفه افزونه اتفاق می‌افتد آن مولفه بعد از اصلاح می‌تواند دوباره به سیستم معرفی شود.

۵-۲-۳- جلوگیری از بروز نقص

تعدادی از تاکتیک‌های مربوط به جلوگیری از بروز نقص به شرح زیر هستند:

[1] Shadow operation
[2] State resynchronization
[3] Checkpoint/rollback
[4] Component reintroduction

- *خارج کردن از سرویس*[1]

این تاکتیک یک مولفه را از عملکرد در سیستم خارج می‌کند تا بدین صورت از بروز خطاها در سیستم جلوگیری کند. به عنوان یک مثال راه اندازی مجدد مداوم یک مولفه برای جلوگیری از نشت حافظه در جهت اینکه سیستم دچار خطا نشود، یک نوع خارج کردن مولفه از سرویس است. عمل خارج کردن مولفه از سرویس می‌تواند به صورت خودکار یا دستی در نظر گرفته شود. در صورت استفاده از حالت اتوماتیک باید یک راهبرد معماری برای آن در نظر گرفته شود.

- *تراکنش‌ها*

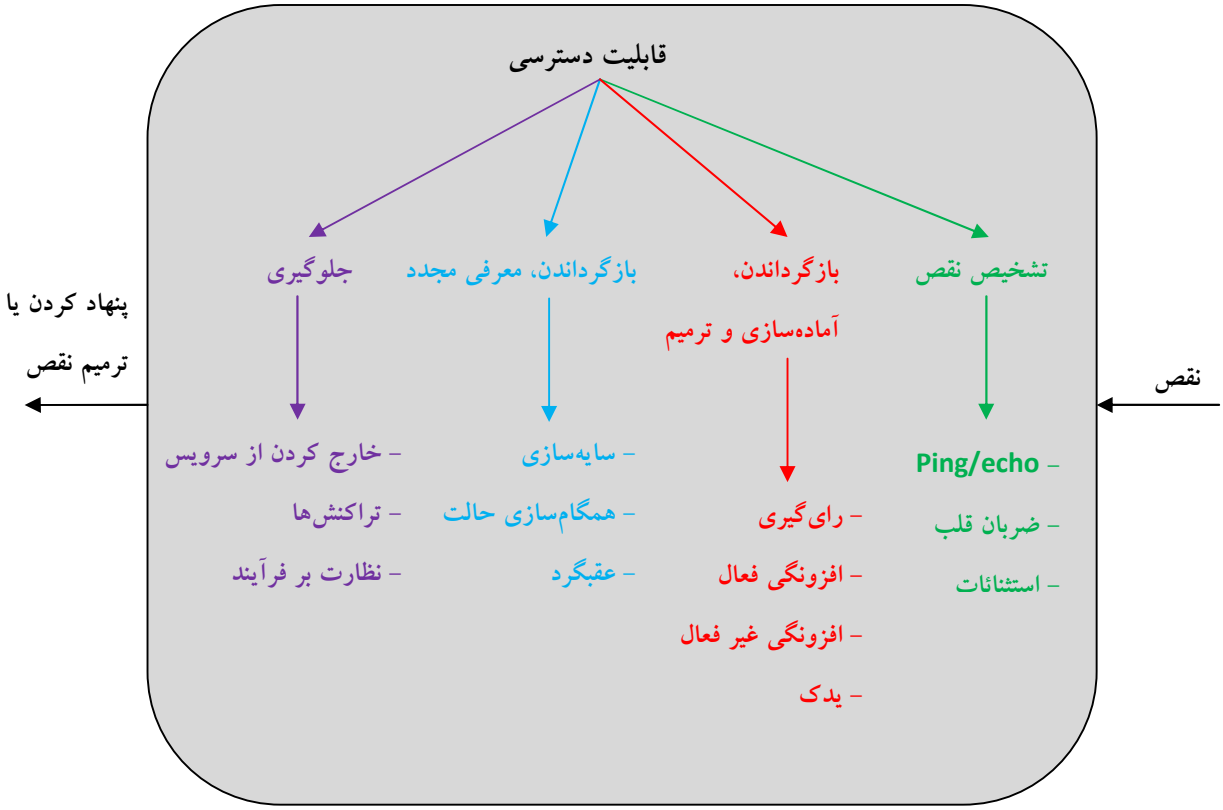
یک تراکنش مجموعه‌ای از گامهای ترتیبی است به صورتی که در یک لحظه مشخص بتوان این گامها را نادیده در نظر گرفت. به بیان بهتر اینکه کلیه دستورات درون یک تراکنش باید کاملا اجرا شوند و اگر خطایی در یک دستور درون تراکنش وجود داشته باشد کلیه دستورات دیگر اعم از اجرا شده و نشده نادیده گرفته شوند. بنابراین هدف از تراکنش، جلوگیری از تصادم اجرای چندین دستور به صورت همزمان بر روی یک داده مشترک و همچنین از بین بردن اثرات ناشی از ناقص اجرا شدن یک دستور در مجموعه دستورات است.

- *نظارت بر فرآیند*[2]

در این تاکتیک به محض اینکه نقصی در یک فرآیند کشف شد، فرآیند نظارت می‌تواند فرآیند دارای مشکل را حذف کرده و یک نمونه جدید از آن ایجاد کند و آن را در حالت مناسب قرار دهد.

شکل ۵-۳ یک دسته‌بندی کلی از تاکتیک‌های مربوط به قابلیت دسترسی را نشان می‌دهد.

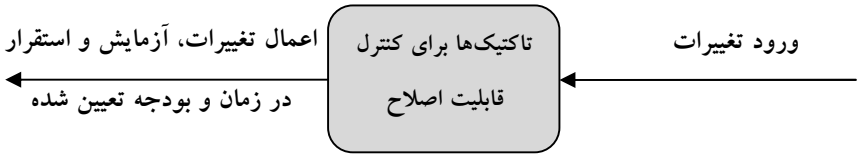
[1] Removal from service
[2] Process monitor



شکل ۵-۳: خلاصه‌ای از تاکتیک‌های قابلیت دسترسی

۵-۳- تاکتیک‌های قابلیت اصلاح[1]

هدف از تاکتیک‌های قابلیت اصلاح (تغییرپذیری)، کنترل زمان و هزینه پیاده‌سازی، آزمایش و استقرار تغییرات است. شکل ۵-۴ این رابطه را به خوبی نشان می‌دهد.



شکل ۵-۴: هدف از تاکتیک‌های قابلیت اصلاح

تاکتیک‌های مربوط به قابلیت اصلاح بر حسب اهدافشان دسته‌بندی می‌شوند. دسته اول که "محلی‌سازی تغییرات[2]" نامیده می‌شود هدفش کاهش تعداد ماژولهایی است که در اثر یک تغییر به طور مستقیم تحت تاثیر

[1] Modifiability tactics
[2] Localize modifications

قرار می‌گیرند. دسته دوم "جلوگیری از تاثیرات موجی[1]" نامیده می‌شود که هدفش محدود کردن تغییرات در ماژولهای محلی است. دلیل تفاوت قائل شدن بین این دو حالت اینکه در دسته اول ماژولهایی وجود دارند که مستقیما تحت تاثیر یک تغییر قرار می‌گیرند (یعنی مسئولیت آنها تغییر می‌کند) ولی در دسته دوم ماژولهایی وجود دارند که به صورت غیرمستقیم تحت تاثیر قرار می‌گیرند (یعنی مسئولیت ماژولها تغییر پیدا نمی‌کند ولی پیاده‌سازی آن ماژولها باید تغییر پیدا کنند تا اثر تغییر در یک ماژول مورد نظر دیده شود). دسته سوم "به تعویق انداختن زمان انقیاد[2]" نامیده می‌شود که هدفش کنترل زمان و هزینه استقرار است. حال در ادامه تاکتیک‌های مربوط به هر دسته‌بندی عنوان شده و توضیحاتی پیرامون نحوه عملکرد هر یک ذکر خواهد شد.

۵-۳-۱- محلی‌سازی تغییرات

اگرچه ارتباط مستقیمی بین تعداد ماژولهای تحت تاثیر یک مجموعه از تغییرات و هزینه پیاده‌سازی آنها وجود ندارد ولی محدود کردن تغییرات به یک مجموعه کوچکی از ماژولها عموما هزینه را کاهش می‌دهد. هدف از تاکتیک‌های موجود در این دسته‌بندی این است که در زمان طراحی طوری مسئولیت‌ها به ماژولهایی تخصیص داده شوند که تغییرات آتی آنها محدود به یک حوزه باشد. یعنی اینکه کلیه تغییرات در داخل حوزه هر ماژول اعمال و مدیریت شود. تاکتیک‌های مربوط به این دسته‌بندی به شرح زیر هستند:

- *حفظ انسجام یا پیوستگی معنایی*[3]

انسجام معنایی به ارتباطات میان مسئولیت‌ها در یک ماژول اشاره دارد. در واقع هدف این است که تضمین شود تمامی مسئولیت‌ها با یکدیگر بدون وابستگی به ماژولهای دیگر عمل (کار) می‌کنند. برای دستیابی به این هدف مسئولیت‌هایی که انسجام معنایی دارند باید در درون یک ماژول قرار بگیرند. معیارهای وابستگی[4] و انسجام[5] یک تلاش برای اندازه‌گیری انسجام معنایی هستند. انسجام معنایی باید نسبت به یک سری از تغییرات از قبل پیش‌بینی شده اندازه‌گیری شود. یک زیر تاکتیک برای این موضوع تجرید سرویسهای مشترک[6] است.

[1] Prevent the ripple effect
[2] Defer binding time
[3] Maintain semantic coherence
[4] Coupling
[5] Cohesion
[6] Abstract common services

فراهم کردن سرویسهای عمومی از طریق ماژولهای خاص معمولا به عنوان عملی در جهت حمایت از قابلیت استفاده مجدد در نظر گفته می‌شود. اما نکته مهم اینکه تجرید سرویسهای مشترک از قابلیت اصلاح نیز پشتیبانی می‌کند. اگر سرویسهای مشترک تجرید شده باشند تغییرات در آنها یک بار اعمال می‌شود و نیازی نیست که این تغییرات در ماژولهای دیگری که از آنها استفاده می‌کنند دوباره اعمال شوند. همچنین استفاده از این سرویسها، سایر کاربران را تحت تاثیر قرار نخواهد داد. بنابراین چنین تاکتیکی هم از محلی‌سازی تغییرات و هم جلوگیری از تاثیرات موجی حمایت می‌کند. مثالهایی از سرویهای مشترک تجرید شده عبارتند از: استفاده از چارچوب‌های کاربردی و نرم‌افزارهای میان‌افزار.

- *پیش‌بینی تغییرات مورد انتظار*[1]

در نظر گرفتن یک سری تغییرات فرضی می‌تواند راهی را برای ارزیابی تخصیص مسئولیت‌های انجام شده فراهم کند. در این تاکتیک اصلی‌ترین سئوال اینکه آیا برای هر تغییر یک تجزیه پیشنهادی وجود دارد که مجموعه‌ای از ماژولهایی که نیاز است تغییر پیدا کنند را محدود کند؟ یا اینکه می‌توان این سئوال را در نظر گرفت که آیا انجام تغییرات مختلف پایه‌ای، ماژولهای یکسانی را تحت تاثیر قرار می‌دهد؟

سئوال دیگر اینکه چرا این تاکتیک متفاوت از تاکتیک انسجام معنائی در نظر گرفته می‌شود؟ تاکتیک تخصیص مسئولیت‌ها مبتنی بر انسجام معنایی، در نظر می‌گیرد که تغییرات پیش‌بینی شده به صورت معنائی، منسجم خواهند بود. تاکتیک پیش‌بینی تغییرات مورد انتظار در ارتباط با انسجام مسئولیت‌های مربوط به یک ماژول نیست بلکه با حداقل کردن تاثیرات تغییرات در ارتباط است. در حقیقت این تاکتیک به سختی قابل استفاده است زیرا نمی توان تمام تغییرات را پیش‌بینی کرد به همین دلیل این تاکتیک همراه با تاکتیک انسجام معنایی به کار گرفته می‌شود.

- *عمومی‌سازی ماژول*[2]

ایجاد یک ماژول به صورت خیلی عمومی اجازه می‌دهد که ماژول محدوده گسترده‌تری از عملکردها را مبتنی بر ورودی محاسبه کند (انجام دهد). ورودی می‌تواند به عنوان یک زبان تعریف شده برای

[1] Anticipate expected changes
[2] Generalize the module

ماژول در نظر گرفته شود که این عمل در حالت ساده می‌تواند یک سری از پارامترهای ثابت در نظر گرفته شود و یا در حالت پیچیده‌تر یک مفسر، که انتخاب و ایجاد پارامترها به عنوان یک برنامه در زبان مفسر باشد. بنابراین با عمومی‌سازی، تغییرات در یک ماژول می‌تواند به احتمال زیاد با تنظیم مناسب زبان ورودی برطرف شود و نیازی نباشد که خود ماژول دستخوش تغییر شود.

- *محدود کردن گزینه‌های ممکن*

تغییرات (مخصوصا در یک خط تولید) می‌تواند دامنه متغییری داشته باشد و لذا ماژولهای زیادی را تحت تاثیر قرار دهد. محدود کردن گزینه‌های ممکن برای تغییرات، تاثیرات ناشی از تغییرات را کاهش خواهد داد. برای مثال یک نقطه تغییر در خط تولید می‌تواند اجازه تغییر پردازنده باشد و لذا محدود کردن استفاده از پردازنده‌های هم خانواده گزینه‌های ممکن برای تغییرات و لذا تاثیرات ناشی از آن را محدود می‌کند.

۵-۳-۲- جلوگیری از تاثیرات موجی

تاثیر موجی یک تغییر، یعنی ایجاد یک تغییر در ماژول دیگری که به طور مستقیم تحت تاثیر آن تغییر نیست. به عنوان مثال اگر ماژول A تغییر پیدا کند تا بتواند یک تغییر (اصلاح) خاص را برآورده کند در اثر تاثیرات موجی ماژول B نیز تغییر پیدا می‌کند و این تغییر B فقط به سبب آن است که ماژول A تغییر پیدا کرده است. دلیل این نوع تغییرات، وابسته بودن B در بعضی از حالات به A است. پس قبل از اینکه به بررسی تاکتیک‌های این دسته پرداخته شود انواع وابستگیهای ممکن بین ماژولها معرفی می‌شود. وابستگیهای ممکن بین ماژولها به شرح زیر هستند:

۱. *نحو داده و سرویس* [1]

داده: فرض کنید برای اینکه B به طور صحیح اجرا شود نیازمند داده‌ای است که این داده می‌تواند توسط A تولید شود، در این صورت باید نوع داده تولید شده توسط A با نوع داده قابل استفاده برای B سازگار باشد.

[1] Syntax of data and service

سرویس: فرض کنید برای اینکه B به طور صحیح اجرا شود نیازمند سرویسی است و این سرویس می‌تواند توسط A فراهم شود در این صورت امضاء سرویس فراهم شده به وسیله A باید با امضاء سرویس مورد نیاز B سازگار باشد.

۲. *معنایی داده و سرویس*[1]

داده: برای اجرای صحیح B، معنای داده تولید شده به وسیله A و مصرف شده به وسیله B باید با فرضیات B سازگار باشد.

سرویس: برای اجرای صحیح B، معنای سرویس تولید شده به وسیله A و مصرف شده به وسیله B باید با فرضیات B سازگار باشد.

۳. *ترتیبی داده و سرویس*[2]

داده: B برای اجرای صحیح خود باید داده‌ای به صورت ترتیبی و آن هم تولید شده توسط A را دریافت کند. برای مثال زمانی که یک بسته توسط گیرنده دریافت می‌شود عنوان بسته داده باید نسبت به بدنه بسته داده در حالت تقدم قرار گرفته باشد.

سرویس: برای اجرای صحیح B، A باید قبلا در یک محدودیت زمانی مشخص اجرا شده باشد. برای مثال A نباید بیش از ۵ میلی ثانیه قبل از اجرای B اجرا شده باشد.

۴. *شناسه واسط A*[3]

A می‌تواند چندین واسط داشته باشد برای اینکه B بتواند به صورت صحیح اجرا شود شناسه‌ی واسط باید با فرضیات B سازگار باشد.

۵. *محل B (در زمان اجرا)*[4]

برای اینکه B به طور صحیح اجرا شود محل زمان اجرای A باید با فرضیات B سازگار باشد. برای مثال B می‌تواند فرض کند که A یک فرآیند متفاوت بر روی همان پردازنده‌ای است که خودش بر روی آن در حال اجرا است.

[1] Semantics of data and service
[2] Sequence of data and service
[3] Identity of an interface of A
[4] Location of A (runtime)

۶. *کیفیت سرویس یا داده تولید شده به وسیله* A[1]

برای اینکه B به طور صحیح اجرا شود بعضی از خصوصیات دربرگیرنده کیفیت داده و سرویس فراهم شده به وسیله A باید با فرضیات B سازگار باشند. برای مثال، داده فراهم شده به وسیله یک سنسور باید دقت لازم را داشته باشد تا الگوریتم موجود در B بتواند به درستی کار کند.

۷. *وجود* A[2]

برای اینکه B به طور صحیح اجرا شود A باید موجود باشد. برای مثال، اگر B سرویسی را از شیء A درخواست کند و شیء A وجود نداشته باشد و یا اینکه امکان ایجاد آن به صورت پویا ممکن نباشد در این صورت B نخواهد توانست به صورت صحیح اجرا شود.

۸. *رفتار منبعی* A[3]

برای اینکه B به طور صحیح اجرا شود رفتار منبعی A باید با فرضیات B سازگار باشد. این می‌تواند به صورت استفاده منبعی از A (A از حافظه مشابه B استفاده می‌کند) یا مالکیت منبع (B یک منبع رزرو می‌کند که A فکر می‌کند صاحب آن است) باشد.

با توجه به مشخص شدن انواع وابستگی‌ها، حال می‌توان تاکتیک‌های موجود برای جلوگیری از انتشار موجی را بررسی کرد. قابل توجه است که هیچ کدام از تاکتیک‌هایی که معرفی خواهند شد ضرورتا از انتشار موجی تغییرات معنایی جلوگیری نمی‌کنند. در روند تشریح تاکتیک‌های مربوطه، ابتدا تاکتیک‌هایی که در ارتباط با واسطهای مربوط به یک ماژول خاص هستند بررسی شده (پنهان‌سازی اطلاعات[4] و نگهداری واسطهای موجود[5]) و بعد شکستن زنجیره وابستگی‌ها[6] معرفی می‌شود (استفاده از واسطه[7]).

- *پنهان‌سازی اطلاعات*

پنهان‌سازی اطلاعات عبارت است از تجزیه مسئولیت‌ها یک موجودیت به قسمت‌های کوچک و مشخص کردن اینکه کدام اطلاعات اختصاصی و کدام اطلاعات عمومی هستند. مسئولیت‌های عمومی

[1] Quality of service/data provided by A
[2] Existence of A
[3] Resource behavior of A
[4] Hide information
[5] Maintain existing interfaces
[6] Breaks a dependency chain
[7] Intermediary

از طریق واسط‌های خاص قابل دسترسی هستند. در واقع هدف اینکه تغییرات را درون یک ماژول ایزوله کرده و از انتشار تغییرات به دیگر ماژولها جلوگیری شود. این تاکتیک یک روش قدیمی برای جلوگیری از انتشار تغییرات است که به شدت وابسته به تاکتیک "پیش‌بینی تغییرات مورد انتظار" است و دلیلش هم اینکه آن از تغییرات پیش‌بینی شده به عنوان یک پایه تجزیه استفاده می‌کند.

- *نگهداری واسط‌های موجود*[1]

اگر B وابسته به نام و امضاء واسط A باشد ثابت نگاه داشتن واسط و گرامر آن باعث می‌شود که B بدون تغییر باقی بماند. البته اگر B یک وابستگی معنایی با A داشته باشد این تاکتیک ضرورتا درست کار نخواهد کرد زیرا تحت پوشش قرار دادن تغییرات در معنا داده و سرویس خیلی مشکل است. همچنین پوشش دادن وابستگی‌ها بر روی کیفیت یک داده یا سرویس، منبع مورد استفاده یا مالکیت منبع نیز کار مشکلی است. ثابت نگاه داشتن واسط می‌تواند از طریق جدا کردن واسط از پیاده‌سازی حاصل شود. این مکانیزم اجازه ایجاد واسط‌های تجریدی را می‌دهد که از طریق آن می‌توان تغییرات[2] را تحت پوشش قرار داد. تغییرات می‌توانند در داخل مسئولیت‌های موجود قرار بگیرند یا با یک پیاده‌سازی جدید ادغام شوند و آن پیاده‌سازی جانشین پیاده‌سازی قبلی واسط شود.

الگوهایی که این تاکتیک را پیاده‌سازی می‌کنند عبارتند از:

اضافه کردن واسط‌ها: بیشتر زبانها اجازه واسط‌های متعدد را می‌دهند. به تازگی سرویسها و داده‌های قابل مشاهده می‌توانند از طریق واسط‌های جدید قابل دسترس باشند که این امر اجازه می‌دهد واسط‌های موجود (قبلی) بدون تغییر بمانند و در نتیجه امضاء یکسان را فراهم کنند.

اضافه کردن آداپتورها[3]: اضافه کردن یک آداپتور به A باعث می‌شود که A توسط آداپتور پوشش داده شده و امضاء اصلی A توسط آن فراهم شود.

فراهم کردن یک *Stub* : اگر تغییرات در سیستم دربرگیرنده حذف A باشد در این صورت، فراهم کردن یک *stub* برای A اجازه می‌دهد که B بدون تغییر باقی بماند و آن هم درصورتی که B فقط وابسته به امضاء A باشد.

[1] Maintain existing interfaces
[2] Variations
[3] Adapter

- *محدود کردن مسیرهای ارتباطی*[1]

محدود کردن مسیرهای ارتباطی یعنی محدود کردن ماژول‌ها به اینکه فقط یک ماژول، داده را به اشتراک بگذارد. به عبارت دیگر یعنی کاهش تعداد ماژولهایی که داده تولید شده به وسیله یک ماژول مشخص را مصرف می‌کنند و نیز کاهش تعداد ماژولهایی که داده مصرف شده به وسیله ماژول را تولید می‌کند. این عمل تاثیر موجی را کاهش می‌دهد زیرا تولید و مصرف داده، وابستگی تعریف می‌کند که باعث انتشار موجی می‌شود.

- *استفاده از یک واسطه*[2]

اگر B هر نوع وابستگی با A به غیر از وابستگی معنائی داشته باشد امکان این وجود دارد که یک واسطه بین A و B قرار داده شود که فعالیت مرتبط با وابستگی را مدیریت کند. انواع واسطه‌های ممکن به شرح زیر هستند:

داده: مخزنها[3] (غیرفعال و تخته سیاه) به عنوان یک واسطه بین مصرف کننده و تولید کننده داده عمل می‌کنند. مخزنها می‌توانند گرامر تولید شده به وسیله A را به فرضیات B تبدیل کنند. بعضی الگوهای *publish/subscribe* می‌توانند گرامر را به فرضیات B تبدیل کنند. الگوهای *MVC* و *PAC* نیز داده‌های موجود در یک فرمالیسم[4] (وسیله ورودی و خروجی) را می‌تواند به دیگری تبدیل کنند.

سرویس: نمای خارجی[5]، پل[6]، میانجی[7]، راهبرد، پراکسی[8] و الگوهای کارخانه[9] کلا واسطه‌ای فراهم می‌کنند که گرامر یک سرویس را به دیگری تبدیل می‌کند. لذا آنها کلا برای جلوگیری از انتشار تغییرات A در B مورد استفاده قرار می گیرند.

شناسه واسط A: الگوی دلال[10] می‌تواند تغییرات در شناسه یک واسط را پنهان کند (پوشش دهد). اگر B وابسته به شناسه یک واسط از A باشد و شناسه آن واسط A تغییر کند در این صورت با اضافه کردن

[1] Restrict communication paths
[2] Use an intermediary
[3] Repositories
[4] Formalism
[5] Facade
[6] bridge
[7] Mediator
[8] Proxy
[9] Factory
[10] Broker

آن شناسه به دلال و داشتن دلالی که با شناسه جدید از A ارتباط برقرار کند B می‌تواند بدون تغییر باقی بماند.

محل A : یک سروریس دهنده نام[1] اجازه می‌دهد که محل A تغییر پیدا کند بدون اینکه B تحت تاثیر قرار بگیرد. در این روش A مسئول اینکه محل جاری خود را در سرور نام ثبت کند و B نیز باید محل A را از سروریس دهنده نام دریافت کند.

رفتار منبعی A یا کنترل منبع به وسیله A : مدیر منبع یک واسطه‌ای است که مسئولیت تخصیص منابع را بر عهده دارد. مدیرهای منابع خاص می‌توانند برآورده کردن تمام درخواست‌ها را در یک محدودیت مشخص تضمین کنند البته بدین منظور باید کنترل منبع به مدیر منبع داده شود.

وجود A :الگوی کارخانه توانائی این را دارد که نمونه‌هایی ایجاد کند که مورد نیاز هستند بنابراین وابستگی B به وجود A به وسیله عملکرد کارخانه ارضاء می‌شود.

۵-۳-۳- به تعویق انداختن زمان انقیاد

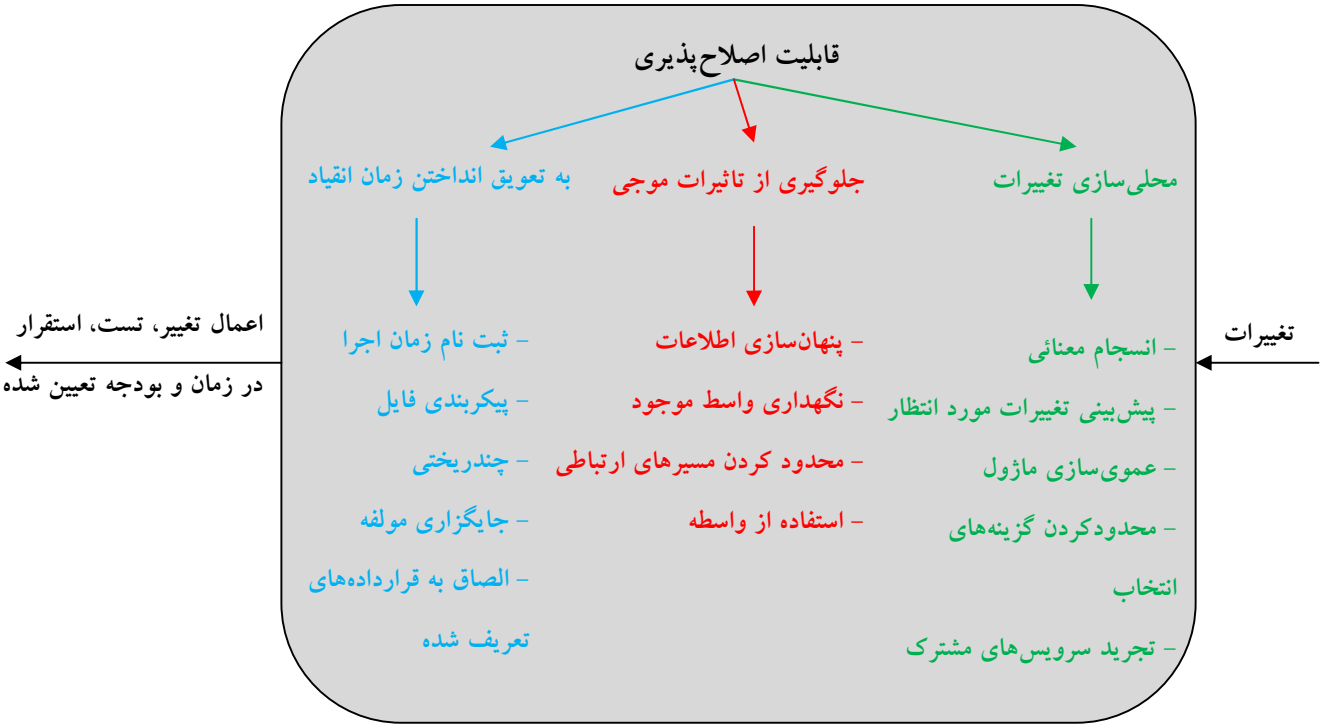
دو دسته تاکتیک تشریح شده در بخش‌های قبلی هدفشان به حداقل رساندن تعداد ماژولهایی است که باید اصلاح شوند تا تغییر مورد نظر پیاده‌سازی شود. بعضی سناریویهای تغییر وجود دارند که شامل دو عنصر هستند (مانند استقرار و اجازه دادن به غیر توسعه دهندگان تا تغییرات را اعمال کنند) که در آنها قابلیت اصلاح با استفاده از کاهش تعداد ماژولها ارضاء نمی‌شود. به تعویق انداختن زمان انقیاد این سناریوها را حمایت می‌کند که آن مستلزم هزینه اضافی است که باید صرف فراهم کردن ساختار اضافی برای حمایت از انقیاد دیرهنگام شود. تصمیمات مربوط به انقیاد می‌توانند محدود به زمانهای مختلف در زمان اجرا باشند (تصمیماتی مطرح است که استقرار را تحت تاثیر قرار می‌دهند). استقرار سیستم به وسیله یک سری فرآیندها دیکته می‌شود. زمانی که یک تغییر به وسیله یک توسعه‌دهنده اعمال می‌شود معمولا یک آزمایش و یک فرآیند توزیع شده وجود دارد که تاخیر زمانی بین ایجاد تغییرات و قابل دسترس بودن آن تغییرات برای کابر نهایی توسط آنها تعیین می‌شود. انقیاد سیستم در زمان اجرا به معنی اینکه سیستم برای آن انقیاد آماده است و کلیه آزمایش‌ها و گامهای توزیع شده کامل شده‌اند. به تعویق انداختن زمان انقیاد به کاربر یا مدیر سیستم اجازه می‌دهد تغییراتی را ایجاد کنند و

[1] Name server

یا ورودی را فراهم کند که رفتار سیستم را تحت تاثیر قرار دهد. تعدادی از تاکتیک‌ها که در زمان بارگذاری یا اجرا تاثیراتی دارند به شرح زیر هستند:

- ثبت نام در زمان اجرا[1] از عملیات *plug and play* حمایت می‌کند، که این مستلزم هزینه سرباز اضافی برای مدیریت ثبت نام است. برای مثال، ثبت نام *publish/subscribe* می‌تواند در زمان اجرا یا زمان بارگذاری پیاده‌سازی شود.
- پیکربندی فایلها تمایل دارند که در زمان شروع پارامترها را مقداردهی کنند.
- چندریختی اجازه انقیاد دیرهنگام فراخوانی متدها را می‌دهد.
- جایگزاری مولفه اجازه انقیاد زمان بارگذاری را می‌دهد.
- الصاق به قراردادهای تعریف شده[2] اجازه انقیاد زمان اجرا فرآیندهای مستقل را می‌دهد.

خلاصه‌ای از تاکتیک‌های قابلیت اصلاح در شکل ۵-۵ به تصویر کشیده شده است



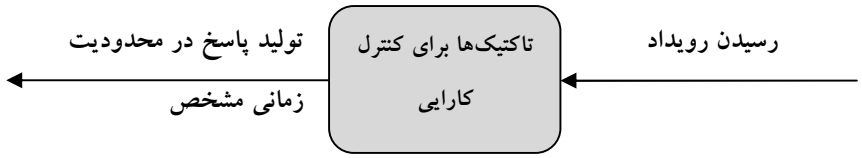
شکل ۵-۵: خلاصه‌ای از تاکتیک‌های قابلیت اصلاح

[1] Runtime registration
[2] Adherence to defined protocols

۵-۴- تاکتیک‌های کارایی

هدف از تاکتیک‌های کارایی ایجاد یک پاسخ به رویدادهای سیستم در یک زمان محدود است. رویداد می‌تواند منفرد یا به صورت جریانی[1] باشد که شروع کننده یک درخواست است تا آن درخواست منجر به انجام یک محاسبه شود. یک رویداد می‌تواند رسیدن یک پیام، منقضی شدن یک بازه زمانی، تشخیص یک تغییر وضعیت خاص در محیط سیستم و غیره در نظر گرفته شود.

با وقوع یک رویداد، سیستم رویداد مربوطه را پردازش کرده و پاسخ مناسب را تولید می‌کند. در واقع تاکتیک‌های کارایی مدت زمانی را که طول می‌کشد یک پاسخ برای یک رویداد تولید شود را کنترل می‌کنند. این مفهوم به وضوح در شکل ۵-۶ به تصویر کشیده شده است. اصولا بین زمانی که یک رویداد تولید شده به سیستم وارد می‌شود و زمانی که پاسخی از جانب سیستم برای آن رویداد تولید می‌شود فاصله زمانی وجود دارد که به آن تاخیر[2] گفته می‌شود. بعد از اینکه یک رویداد وارد سیستم شد سیستم شروع به پردازش آن رویداد می‌کند و یا اینکه پردازش به دلایلی مسدود[3] می‌شود. این مسئله نشان دهنده این است که دو مفهوم (عنصر) مهم در مدت زمان پاسخ تاثیر دارند که عبارتند از: مصرف منبع[4] و زمان مسدود شدن[5].



شکل ۵-۶: هدف از تاکتیک‌های کارایی

۱. مصرف منبع

منابع شامل پردازنده، رسانه‌های ذخیره‌سازی داده، پهنای باند شبکه ارتباطی و حافظه هستند اما منبع می‌تواند شامل موجودیت‌های خاصی نیز باشد که در هر سیستم در زمان طراحی تعریف می‌شوند. رویدادها می‌توانند انواع مختلفی داشته باشند که هر کدام از یک سری پردازش ناشی می‌شوند. برای

[1] Stream
[2] Latency
[3] Blocked
[4] Resource consumption
[5] Blocked time

مثال یک پیام می‌تواند از یک مولفه‌ای که در شبکه قرار گرفته تولید شده و به مولفه دیگر برسد. این فرآیند شامل قرار گرفتن پیام در بافر، انتقال دادن آن، پردازش بر حسب یک الگوریتم خاص، انتقال به خروجی، قرار گرفتن در بافر خروجی، ارسال به سمت مولفه گیرنده است. کلیه این عملیات در کل تاخیر پردازش یک رویداد را تشکیل می‌دهند به عبارت بهتر در ایجاد آن تاخیر شرکت می‌کنند.

۲. *زمان مسدود شدن*

یک محاسبه می‌تواند هنگامی که از یک منبع استفاده می‌کند، مسدود شود. دلایل مسدود شدن می‌تواند رقابت برای منبع، قابل دسترس نبودن منبع و وابسته بودن محاسبه به نتایجی از دیگر محاسبات باشد که هنوز تولید نشده‌اند.

- رقابت برای منابع : شکل ۵-۶ نشان می‌دهد که رویدادها به سیستم وارد می‌شوند. رویدادها می‌توانند در یک جریان منفرد (تکی) یا جریان چندگانه باشند. جریانهای چندگانه بر روی یک منبع رقابت می‌کنند و یا اینکه رویدادهای منفرد مختلف در این جریان بر روی یک منبع رقابت را انجام می‌دهد. همگی این رقابت‌ها منجر به تاخیر می‌شوند. به صورت کلی هر چه قدر میزان رقابت برای یک منبع بیشتر باشد میزان تاخیر نیز افزایش خواهد یافت. با این حال، میزان تاخیر بستگی به این دارد که چگونه رقابت‌ها، حکمیت شوند و همچنین اینکه چگونه یک مکانیزم حکمیت[1] با درخواست‌های منفرد برخورد کند.
- قابل دسترس بودن منبع: حتی در صورت نبودن یک رقابت نیز ممکن است یک محاسبه به دلیل عدم در دسترس بودن منبع مسدود شود. قابل دسترس نبودن ممکن است بدلیل برون خط[2] بودن منبع، خطا در مولفه‌های آن و یا دلایل دیگر اتفاق بیفتد. در هر حالت معمار باید محل‌هایی که ممکن است منبع در آنجا قابل دسترس نبوده و باعث افزایش تاخیر شود را شناسائی کند.
- وابستگی به دیگر محاسبات: یک محاسبه ممکن است مجبور باشد به دلیل همزمان شدن با نتایج دیگر محاسبات و یا دریافت نتایجی از محاسبه‌ای که به وسیله خودش شروع شده است

[1] Arbitration mechanism
[2] Offline

در حالت انتظار به سر ببرد. به عنوان مثال، یک محاسبه ممکن است اطلاعات را از دو منبع مختلف بخواند در این صورت اگر این دو منبع به صورت ترتیبی خوانده شوند تاخیر ایجاد شده نسبت به حالتی که آن دو منبع به صورت موازی خوانده می‌شوند بیشتر خواهد بود.

با توجه به این توضیحات، به تشریح سه دسته‌بندی از تاکتیک‌های مربوط به کارایی پرداخته می‌شود که عبارتند از : درخواست منبع[1]، مدیریت منبع و حکمیت منبع[2].

۵-۴-۱- درخواست منبع

جریانهای رویداد، منشاء درخواست منبع هستند. دوخصوصیت از یک درخواست عبارت است از فاصله زمانی بین رویدادها در یک جریان منبع (جریان درخواست منبع) و اینکه چه تعدادی از یک منبع به وسیله هر درخواست مصرف می‌شود. یکی از تاکتیک‌ها در جهت کم کردن زمان تاخیر، کاهش منابع مورد نیاز برای پردازش یک رویداد است. راه‌های انجام آن به شرح زیر هستند:

- *افزایش بهره‌وری محاسبات*

اولین گام در پردازش یک رویداد یا پیام اعمال کردن الگوریتم است. بهبود الگوریتم استفاده شده در مناطق بحرانی مطمئنا تاخیر زمانی را کاهش خواهد داد. گاها یک منبع ممکن است بر دیگری ترجیح داده شود. برای مثال، داده‌های میانی می‌توانند در یک مخزن نگهداری شوند یا مبتنی بر زمان و فضای قابل دسترس، داده‌ها از نو تولید شوند. این تاکتیک معمولا به پردازنده‌ها اعمال می‌شود اما می‌توان از آن برای دیسک‌ها نیز استفاده کرد.

- *کاهش سربار محاسباتی*

اگر درخواستی برای یک منبع وجود نداشته باشد نیازهای پردازشی کاهش می‌یابند. به عبارت دیگر استفاده از تاکتیک‌های خاص می‌تواند باعث کاهش یا افزایش سربار محاسباتی شود. به عنوان مثال استفاده از واسطه‌ها میزان منابع قابل استفاده در هنگام پردازش یک جریان رویداد را افزایش می‌دهد.

تاکتیک دیگر برای کم کردن زمان تاخیر، کاهش تعداد رویدادهایی است که پردازش می‌شود. این تاکتیک می‌تواند با یکی از روشهای زیر تحقق پیدا کند:

[1] Resource demand
[2] Resource arbitration

- *مدیریت نرخ رویداد*

اگر امکان آن وجود داشته باشد که تناوب نمونه‌گیری که به منظور نظارت بر متغییرهای محیطی استفاده می‌شود را کاهش داد، درخواست‌ها نیز می‌توانند کاهش پیدا کنند. این مکانیزم گاهی اوقات امکان‌پذیر است و شرط آن اینکه سیستم فوق مهندسی[1] شده باشد. بعضی زمانها هم نرخ نمونه‌گیری بالای غیرضروری استفاده می‌شود تا دوره هماهنگ[2] بین جریانهای چندگانه ایجاد کنند. یعنی بعضی جریان از رویدادها، فوق نمونه‌گیری[3] می‌شوند تا اینکه بتوانند هماهنگ شوند.

- *کنترل تناوب نمونه‌گیری*

اگر هیچ کنترلی بر ورود رویدادهای خارجی تولید شده وجود نداشته باشد نمونه‌گیری از صف درخواست‌ها می‌تواند در تناوب پائینی انجام شود که این عمل منجر به این خواهد شد که بعضی از درخواست‌ها از بین بروند.

تاکتیک‌های دیگر برای کاهش و مدیریت درخواست‌ها شامل کنترل کردن استفاده از منابع است. این تاکتیک‌ها به شرح زیر هستند:

- *زمان اجرا محدود*

هدف از این تاکتیک محدود کردن مقدار زمان اجرایی استفاده شده برای پاسخ به یک رویداد است. این عمل گاهی اوقات مفید و گاهی اوقات بی اثر است. برای الگوریتم‌های وابسته به داده که به صورت تکراری عمل می‌کنند محدود کردن تعداد تکرارها یک روش محدود کردن زمان اجرا است.

- *اندازه صف محدود*[4]

این تاکتیک حداکثر تعداد ورودی به صف‌ها را کنترل می‌کند.

۵-۴-۲- مدیریت منابع

اگرچه درخواست برای منابع ممکن است قابل کنترل نباشد ولی مدیریت منابع بر روی زمان پاسخ تاثیر بسازی دارد. بعضی از تاکتیک‌های مدیریت منابع به شرح زیر هستند:

[1] Overengineering
[2] Harmonic periods
[3] Oversampling
[4] Bound queue sizes

- *معرفی همزمانی*

اگر درخواست‌ها بتوانند به صورت موازی پردازش شوند زمان مسدود شدن کاهش پیدا می‌کند. همزمانی می‌تواند به وسیله پردازش جریانهای مختلف از رویدادها روی نخهای متفاوت و یا به وسیله نخهای اضافی برای مجموعه فعالیت‌های مختلف فرآیند نشان داده شود. در زمان همزمانی، تخصیص نخها به منابع به صورت مناسب (توازن بار) به منظور حداکثر کردن تاثیر همزمانی از اهمیت بالایی برخوردار است.

- *نگه داشتن چندین نسخه از داده یا محاسبات*

سرویس‌گیرنده‌ها در یک الگوی سرویس‌دهنده/سرویس‌گیرنده، نسخه‌های المثنی از محاسبات هستند. هدف از نسخه‌های المثنی کاهش رقابت‌هایی است که ایجاد می‌شود. رقابت زمانی ایجاد می‌شود که همه مولفه‌ها بر روی یک سرور مرکزی قرار گرفته باشند. عملیات نهان‌سازی[1] تاکتیکی است که در آن داده‌ها بر روی مخزن‌هایی با سرعت‌های مختلف یا مخزنهای مجزا تکرار می‌شوند تا رقابت کاهش پیدا کند. از آنجائی که داده‌های نهان شده یک نسخه از داده‌های موجود هستند سازگاری و همزمانی نسخه‌ها به عنوان یک مسئولیت سیستم باید در نظر گفته شود.

- *افزایش منابع قابل دسترس*[2]

پردازنده‌های سریع، پردازنده‌های اضافی، حافظه اضافی و شبکه‌های سریع کلا عوامل بالقوه‌ای برای کاهش زمان تاخیر هستند. معمولا هزینه یکی از فاکتورهای تاثیرگذار در انتخاب منابع است اما افزایش منابع به صورت قطعی تاکتیکی برای کاهش زمان تاخیر است.

۵-۴-۳- حکمیت منابع

هر زمان رقابت بر سر منابع (پردازنده‌ها، حافظه‌های کمکی و شبکه‌ها) وجود داشته باشد منابع باید زمانبندی شوند. هدف معمار فهم خصوصیات هر منبع استفاده شده و انتخاب راهبرد زمانبندی سازگار با آن است. یک سیاست زمانبندی به صورت مفهومی متشکل از دو بخش است: بخش اول تخصیص اولویت و بخش دیگر توزیع است.

[1] Caching
[2] Increase available resources

تمام سیاست‌های زمانبندی، تخصیص اولویت را انجام می‌دهند. گاها فرآیند اولویت‌دهی، ساختار ساده‌ای همچون اولین ورودی/اولین خروجی[1] دارد. در دیگر حالت‌ها، یک زمان انقضاء و یا اهمیت معنایی به هر درخواست الصاق می‌شود. معیارهای رقابت برای زمانبندی عبارتند از: استفاده بهینه از منابع، اهمیت درخواست، حداقل کردن تعداد منابع استفاده شده، حداقل کردن تاخیر، حداکثر کردن توان‌کاری، جلوگیری از گرسنگی به منظور تضمین انصاف و غیره. بنابراین معمار نیاز دارد که از این معیارهای تداخلدار و تاثیرات آنها بر روی تاکتیک‌های انتخاب شده آگاهی داشته باشد. یک جریان رویداد اولویت بالا زمانی می‌تواند توزیع شود که منبع تخصیص داده شده به آن در دسترس باشد. گاهی اوقات این عمل به قبضه کردن کاربر فعلی منبع بستگی دارد.

بعضی از سیاست‌های زمانبندی عمومی عبارتند از:

۱. *اولین ورودی/اولین خروجی*

صفهای *FIFO* با کلیه درخواست‌های منابع به صورت یکسان برخورد می‌کند و به نوبت منابع را در اختیار آنها قرار می‌دهند. مسئله‌ای که در *FIFO* وجود دارد اینکه یک درخواست جدید دقیقا پشت سر درخواست قبلی در صف قرار می‌گیرد و اگر یک درخواست زمان پاسخ زیادی داشته باشد کلیه درخواست‌ها در صف باید منتظر بمانند. در واقع به صورت کامل برابری در این سیاست رعایت می‌شود. البته این مسئله یک مشکل محسوب نمی‌شود اما اگر بعضی از درخواست‌ها اولویت بالایی نسبت به بقیه داشته باشند این الگوریتم نمی‌تواند راهکاری را برای آنها فراهم کند.

۲. *زمانبندی اولویت ثابت*[2]

این سیاست برای هر درخواست که به منبع می‌رسد یک اولویت بخصوص اختصاص می‌دهد و منبع را با توجه به اولویت‌ها اختصاص می‌دهد. این راهبرد تضمین می‌کند که سرویس بهتر برای درخواست‌هایی با اولویت بالا داشته باشد اما امکان سرویس دادن به یک اولویت پایین را نیز می‌پذیرد. بنابراین ممکن است که یک درخواست مدت زیادی منتظر سرویس بماند چرا که پشت سر تعداد زیادی درخواست با اولویت بالا قرار گرفته است. سه راهبرد اولویت‌بندی عمومی وجود دارد که عبارتند از:

[1] First In First Out (FIFO)
[2] Fixed-priority scheduling

اهمیت معنائی: به هر جریان یک اولویت به صورت ثابت با توجه به بعضی از خصوصیات دامنه وظیفه‌ای که جریان را تولید کرده است اختصاص داده می‌شود. این نوع زمانبندی در سیستم‌های کامپیوتری بزرگ استفاده می‌شود که خصوصیات دامنه، زمان راه‌اندازی وظیفه[1] است.

یکنواختی فرجه[2]: یکنواختی فرجه یک تخصیص اولویت ایستا است که اولویت بالا را به جریانهایی با کوتاه‌ترین فرجه می‌دهد. این سیاست زمانی استفاده می‌شود که جریانی از اولویت‌های مختلف با فرجه بلادرنگ باید زمانبندی شوند.

یکنواختی نرخ[3]: یکنواختی نرخ یک تخصیص اولویت ایستا برای جریانهای دوره‌ای است. در این سیاست بالاترین اولویت به جریانهای با کوتاه‌ترین دوره تخصیص داده می‌شود. این سیاست، زمانبندی یک مورد خاص از یکنواختی فرجه است اما این روش نسبتا مشهورتر است و توسط سیستم عامل‌ها نیز پشتیبانی می‌شود.

۳. *زمانبندی اولویت پویا*

نوبت چرخشی[4]: نوبت چرخشی یک راهبرد زمانبندی است که درخواست‌ها را مرتب می‌کند و سپس در زمانی که امکان تخصیص وجود دارد منبع را به درخواست بعدی با توجه به نوبت، تخصیص می‌دهد. یک شکل خاص از نوبت چرخشی اجرایی چرخشی[5] است که عملیات تخصیص در فواصل زمانی ثابت انجام می گیرد.

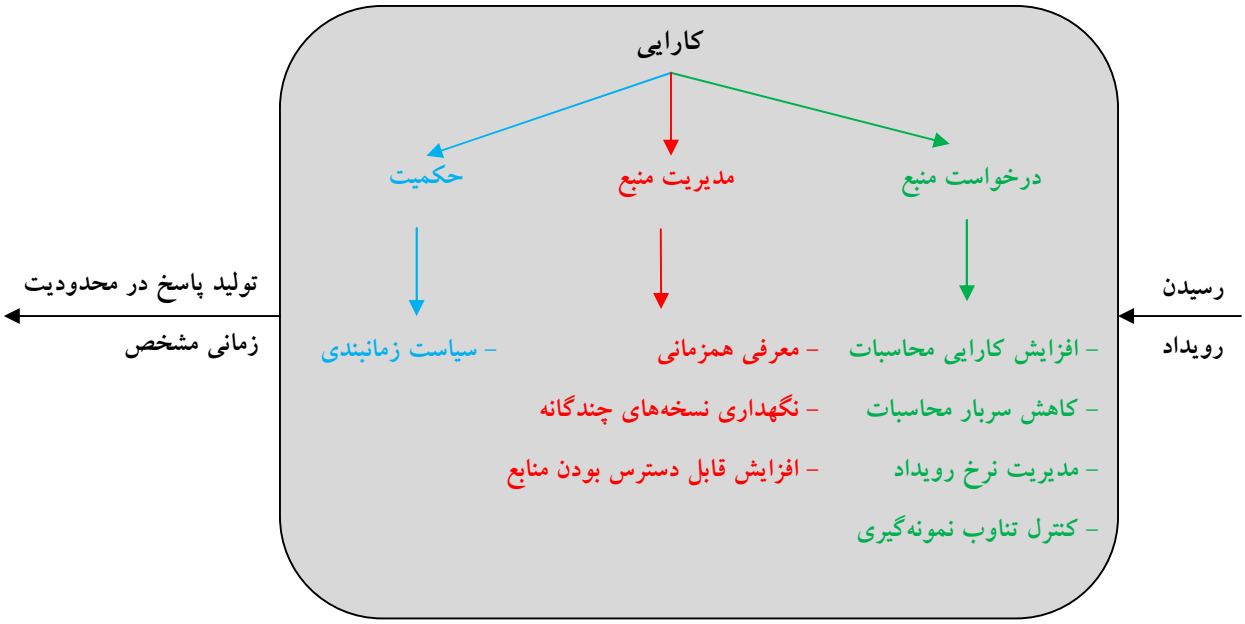
زودترین فرجه اول[6]: اولویت‌ها را بر اساس درخواست‌های معلق با زودترین فرجه تخصیص می‌دهد.

۴. *زمانبندی ایستا*

یک زمانبندی اجرائی چرخشی یک راهبرد زمانبندی است که نقاط انحصاری و ترتیب تخصیص منابع به صورت برون خطی انجام می‌گیرد.

تاکتیک‌های مربوط به کارایی در شکل ۵–۷ خلاصه شده است.

[1] The time of task initiation
[2] Deadline monotonic
[3] Rate monotonic
[4] Round robin
[5] Cyclic executive
[6] Earliest deadline first



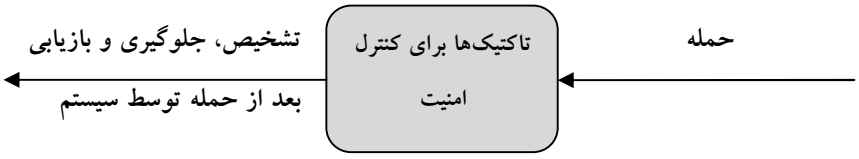
شکل ۵-۷: خلاصه‌ای از تاکتیک‌های کارایی

۵-۵- تاکتیک‌های امنیت

تاکتیک‌های مربوط به امنیت را می‌توان در سه گروه دسته‌بندی کرد:

- تاکتیک‌های مربوط به جلوگیری از حمله
- تاکتیک‌های مربوط به کشف حمله
- تاکتیک‌های مربوط به بازیابی بعد از حمله

به عنوان مثال قفل گذاشتن بر روی در خانه تاکتیک جلوگیری از حمله است، قرار دادن یک سنسور حرکتی در خانه تاکتیک کشف حمله و داشتن بیمه خانه تاکتیک بازیابی بعد از حمله به شمار می‌رود. شکل ۵-۸ اهداف تاکتیک‌های امنیت را نشان می‌دهد.



شکل ۵-۸: هدف از تاکتیک‌های امنیت

۵-۵-۱- جلوگیری از حمله‌ها

عدم انکار[1]، محرمانگی، یکپارچگی و اطمینان از جمله خصوصیات امنیت هستند. تاکتیک‌هایی که در ادامه تشریح خواهند شد می‌توانند در ترکیب با یکدیگر منجر به تحقق این خصوصیات شوند.

- *اعتبارسنجی کاربران*

اعتبارسنجی تضمین می‌کند که کاربر یا کامپیوتر راه دور واقعا همان کسی است که باید باشد. کلمه عبور، کلمه‌های عبور یک مرتبه‌ای[2]، گواهینامه‌های دیجیتال و شناسه‌های بیولوژی همگی یک اعتبارسنجی را فراهم می‌کنند.

- *مجوزدهی به کاربران*

مجوزدهی تضمین می‌کند که کاربر معتبر اجازه دسترسی و تغییر داده یا سرویس را دارد. این عمل معمولا از طریق یک سری الگوهای کنترل دسترسی در سیستم مدیریت می‌شود. کنترل دسترسی می‌تواند به وسیله کاربر یا کلاس کاربر[3] انجام شود (اعمال شود). کلاسهای کاربران می‌توانند به وسیله گروه‌های کاربران، نقشهای کاربران یا لیست‌هایی از کاربران منفرد تعریف شوند.

- *حفظ محرمانگی داده*

داده‌ها باید از دسترسی نامعتبر محافظت شوند. محرمانگی معمولا از طریق اعمال بعضی الگوریتمهای رمزگذاری بر روی داده یا پیوندهای ارتباطی[4] حاصل می‌شود. رمزگذاری محافظت (اطمینان) بیشتری برای حفظ ماندگاری داده نسبت به روش مجوزدهی فراهم می‌کند. پیوندهای ارتباطی معمولا فرآیند کنترل مجوز را ندارند بنابراین رمزنگاری تنها روشی برای انتقال داده از طریق پیوندهای ارتباط عمومی است. پیوند می‌تواند از طریق شبکه خصوصی مجازی[5] یا پروتکل‌های امن مانند *SSL*[6] برای پیوندهای مبتنی بر وب پیاده‌سازی شود. رمزنگاری می‌تواند متقارن (یعنی دارا بودن کلیدهای یکسان) یا نامتقارن (یعنی دارا بودن کلیدهای عمومی و خصوصی) در نظر گرفته شود.

---

[1] Nonrepudiation
[2] One-time passwords
[3] User class
[4] Communication links
[5] Virtual Private Network (VPN)
[6] Secure Sockets Layer

- *حفظ یکپارچگی*

داده باید به همان شکلی که مدنظر است تحویل داده شود (انتقال داده شود). برای تضمین این ویژگی، داده می‌تواند اطلاعاتی اضافی رمزگذاری شده در درون خود داشته باشد (مانند ***checksum*** یا نتایج ***hash***). این اطلاعات اضافی می‌توانند همراه با داده اصلی یا مستقل از آن رمزگذاری شوند.

- *نمایش محدود*[1]

حمله‌ها معمولا مبتنی بر یک ضعف از سیستم هستند که از آن ضعف سوء استفاده می‌شود تا به کلیه داده‌ها و سرویسها در سیستم میزبان حمله شود. معمار می‌تواند تخصیص سرویس‌ها به میزبان را طوری طراحی کند تا سرویس‌های محدودی برای هر میزبان قابل دسترس باشد.

- *دسترسی محدود*

دیوارهای آتشی دسترسی را مبتنی بر منبع پیام یا درگاه مقصد محدود می‌کنند. پیام‌هایی که از یک منبع نامشخص می‌آیند می‌توانند یک شکلی از حمله باشند. محدود کردن منبع پیام همیشه امکان‌پذیر نیست. به عنوان مثال، یک وب سایت عمومی می‌تواند انتظار داشته باشد که یک دسترسی از منبع نامشخص داشته باشد.

۵-۵-۲- تشخیص حمله‌ها

تشخیص حمله معمولا از طریق یک سیستم تشخیص نفود انجام می‌شود[2]. این نوع سیستم برای عملکرد خود از عمل مقایسه الگوهای ترافیک شبکه با داده‌های یک پایگاه‌داده خاص استفاده می‌کنند. در مورد تشخیص سوء استفاده، سیستم الگوی ترافیک را با الگوهای تاریخی حمله‌های مشهور مقایسه می‌کند. در مورد تشخیص وضعیت غیرعادی، سیستم الگوی ترافیک را با خط مبنای تاریخی خودش مقایسه می‌کند. خیلی اوقات به منظور انجام مقایسه، بسته‌ها باید فیلتر شوند. عمل فیلتر کردن می‌تواند بر پایه قرارداد، پرچم‌های *TCP*[3]، اندازه بارمفید[4]، منبع یا مقصد آدرس و شماره درگاه انجام شود.

[1] Limit exposure
[2] Intrusion dectection system
[3] Transmission Control Protocol
[4] Payload sizes

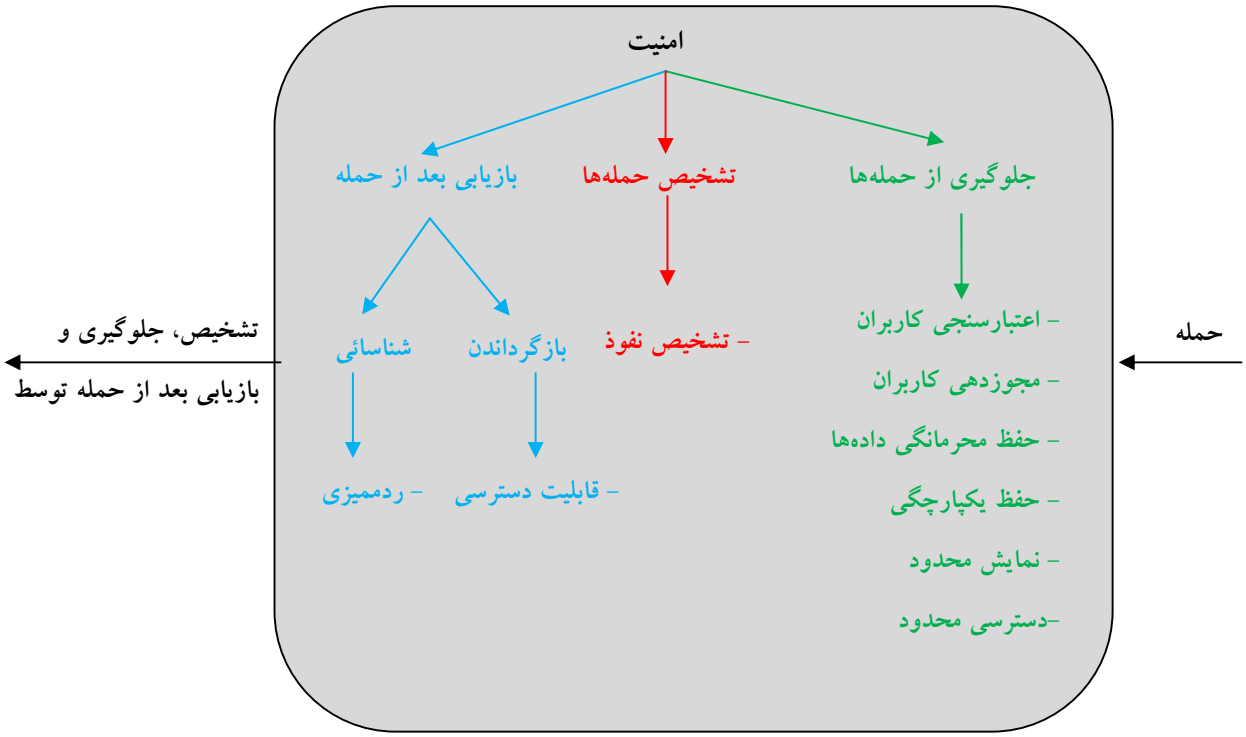
تشخیص دهنده‌های نفوذ باید موارد زیر را داشته باشند: مجموعه‌ای از سنسورها برای تشخیص حمله، مدیریت‌هایی برای ترکیب سنسورها، پایگاه‌داده‌هایی برای مرتب‌سازی رویدادها برای تحلیل نهائی، ابزارهایی برای تحلیل و گزارش‌گیری برون خط و یک کنسول کنترل در جهت اینکه تحلیل کننده بتواند کنش‌های تشخیص دهنده نفوذ را تغییر دهد.

۵-۵-۳- بازیابی بعد از حمله

تاکتیک‌های مربوط به بازیابی بعد از حمله می‌توانند به دو دسته تقسیم شود دسته اولی تاکتیک‌هایی هستند که در ارتباط با بازگرداندن وضعیت سیستم به حالت قبل از حمله هستند و دسته دوم در ارتباط با شناسائی شناسه حمله کننده است (به منظور اهداف پیشگیری یا پیگرد قانونی). تاکتیک‌های استفاده شده برای بازیابی سیستم یا داده به یک حالت صحیح با تاکتیک‌های استفاده شده برای قابلیت دسترسی هم پوشانی دارند زیرا هر دو اینها در ارتباط با بازیابی حالت سازگار سیستم از یک حالت ناسازگار است. تنها تفاوت این است که در نگهداری نسخه‌های افزونه از داده‌های مدیریتی سیستم مانند کلمه عبور، لیست کنترل دسترسی و سرویس‌های نام حوزه[1] و داده‌های مربوط به نمایه[2] کاربران توجه زیادی می‌شود.

تاکتیک مورد نیاز برای شناسائی حمله کننده، نگهداری یک رد ممیزی[3] (دنبال ممیزی) است. رد ممیزی شامل یک نسخه از تمام تراکنش‌های اعمال شده بر روی داده به همراه اطلاعات مربوط به تشخیص هویت است. اطلاعات ممیزی می‌تواند برای ردگیری فعالیت‌هایی که یک حمله کننده انجام داده، جلوگیری کردن از عدم انکار حمله کننده و پشتیبانی از بازبابی سیستم استفاده شود. رد ممیزیها گاها خودشان هدف حمله هستند بنابراین باید در یک روش مطمئن از آنها نگهداری شود. شکل ۵-۹ خلاصه‌ای از تاکتیک‌های امنیت را نشان می دهد.

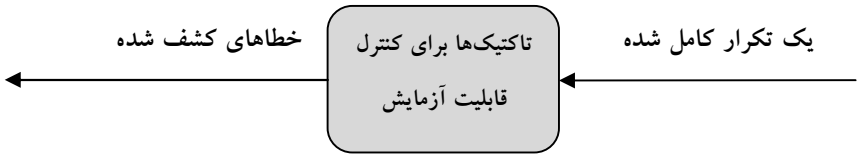
[1] Domain name services
[2] Profile
[3] Audit trail



شل ۵-۹: خلاصه‌ای از تاکتیک‌های امنیت

۵-۶- تاکتیک‌های قابلیت آزمایش

هدف از تاکتیک‌های قابلیت آزمایش سهل‌تر نمودن آزمایش سیستم است و آن هم برای زمانیکه یک تکرار از توسعه سیستم به پایان می‌رسد. شکل ۵-۱۰ استفاده از تاکتیک‌ها برای قابلیت آزمایش را نشان می‌دهد.



شکل ۵-۱۰: هدف از تاکتیک‌های قابلیت آزمایش

تاکتیک‌های مطرح شده در این بخش در ارتباط با آزمایش سیستم در زمان اجرا هستند. هدف از آزمایش سیستم کشف نقص‌های آن است که این امر مستلزم فراهم آوردن ورودی به سیستم و دریافت خروجی است. اجرای رویه‌های آزمایش مستلزم نرم‌افزاری است که ورودیهایی برای سیستم در حال آزمایش فراهم کند و بعد

خروجی حاصل از آن ورودی‌ها را جمع‌آوری کند، این نرم‌افزار کنترل کننده آزمایش[1] نامیده می‌شود. نکته مهمی که می‌توان در مورد طراحی و تولید کنترل کننده آزمایش ذکر کرد اینکه در بعضی از سیستمها انجام آن زمانبر و پرهزینه است. تاکتیک‌های مربوط به قابلیت آزمایش به دو دسته "فراهم کردن ورودی و جمع آوری خروجی" و "نظارت داخلی[2]" تقسیم می‌شوند.

۵-۶-۱- ورودی و خروجی

تاکتیک‌های مربوط به مدیریت ورودی و خروجی آزمایش به شرح زیر هستند:

- *ثبت/برگشت به عقب*[3]

ثبت و برگشت به عقب به هر دو عمل جمع آوری اطلاعات مبادله شده بر روی واسط و استفاده از آن در کنترل کننده آزمایش اطلاق می‌شود. اطلاعات در حال گذر از واسط در هنگام اجرای عادی سیستم در یک مخزن ذخیره می‌شود و خروجی توسط یک مولفه و ورودی نیز توسط مولفه دیگر نمایش داده می‌شود. ثبت این اطلاعات اجازه می دهد که ورودی آزمایش برای یک مولفه تولید شود و خروجی آزمایش برای مقایسه بعدی ذخیره شود.

- *جدا کردن واسط از پیاده‌سازی*

جداکردن واسط از پیاده‌سازی اجازه جانشینی پیاده‌سازیهای[4] متفاوت را برای اهداف آزمایشی مختلف فراهم می‌سازد. پیاده‌سازیهای ریشه‌ای[5] اجازه می‌دهند که بقیه سیستم در غیاب مولفه ریشه فرض شده آزمایش شود. جانشین کردن یک مولفه‌ی بخصوص اجازه می دهد مولفه‌ای که جانشین می‌شود به عنوان یک کنترل کننده برای بقیه سیستم عمل کند.

- *مسیرهای دسترسی/واسط‌های ویژه*[6]

داشتن واسط‌های آزمایش ویژه اجازه می‌دهد که مقادیر متغییر مولفه بدون وابستگی به اجرای سیستم از طریق کنترل کننده آزمایش جمع آوری شود. برای مثال فوق داده می‌تواند از طریق واسط خاصی

[1] Test harness
[2] Internal monitoring
[3] Record/playback
[4] Substitution of implementations
[5] Studding
[6] Specialize access routs/interfaces

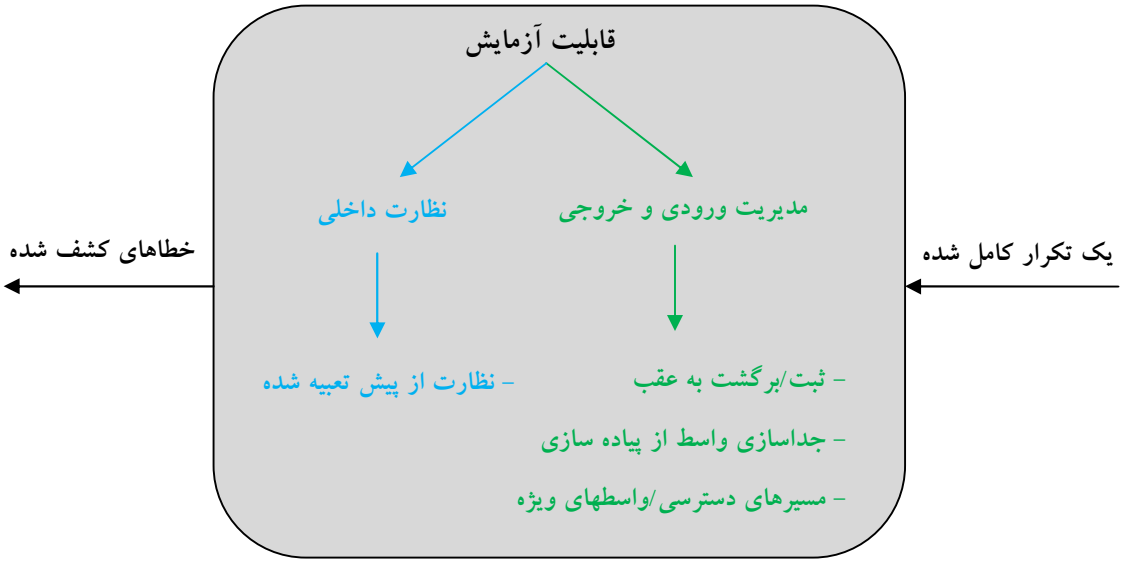
قابل دسترس باشد که با استفاده از آن کنترل کننده آزمایش باید فعالیت‌های خودش را به انجام برساند (یا از آن استخراج کند). مسیرهای دسترسی و واسطهای ویژه باید از مسیرها دسترسی و واسطهای مورد نیاز برای وظیفه‌مندیهای سیستم جدا نگهداری شوند. داشتن یک سلسله مراتبی از واسطها در معماری نشان می‌دهد که موارد تست در هر سطح معماری می‌توانند اعمال شوند (آزمایش وظیفه‌مندی به منظور مشاهده پاسخ‌ها است).

۵-۶-۲- نظارت داخلی

یک مولفه می‌تواند تاکتیک‌هایی بر پایه وضعیت داخلی پیاده‌سازی کند تا فرآیند آزمایش را پشتیبانی نماید.

- *نظارت‌های از پیش تعبیه شده*

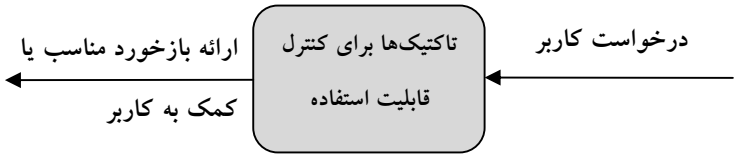
مولفه می‌تواند وضعیت، کارایی، ظرفیت، امنیت و اطلاعات دیگر قابل دسترس از طریق واسط را نگهداری کند. این واسط می‌تواند یک واسط ثابت از مولفه باشد یا اینکه واسطهای موقتی که از طریق تاکتیک‌های خاصی مانند برنامه‌نویسی جنبه‌گرا یا ماکروهای پیش پردازش شده ایجاد می‌شوند. یک تاکتیک عمومی، ذخیره کردن رویدادها در زمانی است که وضعیت‌های نظارت فعال شده است. وضعیت‌های نظارت می‌تواند تلاشهای لازم برای آزمایش را افزایش دهد زیرا آزمایش ممکن است بعد از تمام شدن نظارت مجبور باشد تکرار شود. افزایش قابلیت رویت در فعالیت‌های مولفه معمولا سنگین‌تر از هزینه آزمایش اضافی است. شکل ۵-۱۱ خلاصه‌ای از تاکتیک‌های استفاده شده برای قابلیت آزمایش را نشان می دهد.



شکل ۵-۱۱: خلاصه‌ای از تاکتیک‌های قابلیت ازمایش

۵-۷- تاکتیک‌های قابلیت استفاده

قابلیت استفاده در ارتباط است با میزان راحتی کاربر در انجام یک عمل مشخص و پشتیبانی‌هایی که یک سیستم برای کاربر فراهم می‌کند. دو گروه از تاکتیک‌ها برای پشتیبانی از قابلیت استفاده وجود دارند که هر کدام برای گروه خواصی از کاربران در نظر گرفته شده‌اند. گروه اول تاکتیک‌های زمان اجرا هستند که دربرگیرنده فعالیت‌هایی هستند که از کاربر در زمان اجرای سیستم پشتیبانی می‌کنند. گروه دوم در ارتباط با ماهیت تکراری بودن طراحی واسط کاربری و پشتیبانی از توسعه‌دهنده واسط، در زمان طراحی است. تاکتیک‌های مربوط به قابلیت استفاده به شدت در ارتباط با تاکتیک‌های مربوط به قابلیت اصلاح هستند. شکل ۵-۱۲ هدف از تاکتیک‌های قابلیت استفاده در زمان اجرا را نشان می‌دهد.



شکل ۵-۱۲: هدف از تاکتیک‌های قابلیت استفاده در زمان اجرا

۵-۷-۱- تاکتیک‌های زمان اجرا

وقتی سیستم شروع به اجرا شدن می‌کند قابلیت استفاده از طریق بازخوردهایی از جانب کاربر، به عنوان کسی که سیستم برای آن کاری را انجام می‌دهد، افزایش می‌یابد. همچنین قابلیت استفاده از طریق فراهم کردن دستورات مبتنی بر قابلیت استفاده[1] نیز افزایش می‌یابد. برای مثال لغو، برگشت به وضعیت قبل، موافقت[2] و نشان دادن چند دید، از تصحیح خطا یا اجرا کردن مناسب عملیات به وسیله کاربر حمایت می‌کند.

محققین در تعاملات انسان با کامپیوتر عبارات "ابتکار عمل با کاربر[3]"، "ابتکار عمل با سیستم" و "ابتکار عمل ترکیبی[4]" را برای شرح دادن اینکه کدام یک از انسان یا کامپیوتر در انجام عملیات اصلی پیش قدم است و اینکه چگونه تعاملات را پیش می‌رود، به کار می‌برند. به عنوان مثال زمان لغو یک دستور، کاربر فرمان لغو را صادر می‌کند بنابراین ابتکار عمل با کاربر است (یعنی شروع کننده کاربر است) و سیستم نیز به آن پاسخ می‌دهد. در روند لغو کردن، سیستم ممکن است یک صفحه پیشرفت را نشان دهد که این نیز یک حالتی است که ابتکار عمل با سیستم است (یعنی شروع کننده سیستم است). بنابراین لغو کردن یک ابتکار عمل ترکیبی است.

از عبارات ابتکار عمل با سیستم یا کاربر برای دسته‌بندی تاکتیک‌هایی استفاده می‌شود که معمار می‌تواند با استفاده از آنها به سناریوهای گوناگون دست پیدا کند.

هنگامی که کاربر ابتکاری را انجام می‌دهد معمار برای آن یک پاسخ مانند دیگر بخش‌های وظیفه‌مندی طراحی می‌کند. معمار باید مسئولیت‌های سیستم برای پاسخ دادن به دستورات کاربر را کاملا مشخص کند. حال دوباره دستور لغو را در نظر بگیرید زمانی که کاربر دستور لغو را صادر می‌کند سیستم باید در حال گوش دادن به دستور باشد (بنابرانی مسئولیتی در اینجا وجود دارد، باید یک گوش دهند ثابت وجود داشته باشد و آن هم نباید به وسیله کنش‌های دیگر مسدود شود)، سپس دستور لغو را از بین ببرد و تمامی منابع در حال استفاده به وسیله دستور لغو شده را آزاد کند. نهایتا مولفه‌هایی که در ارتباط با دستور لغو است باید از این امر آگاه شوند تا اینکه بتوانند کنش‌های مناسب را انجام دهند.

[1] Usability-based commands
[2] Aggregate
[3] User initiative
[4] Mixed initiative

زمانی که سیستم یک ابتکاری را انجام می‌دهد آن باید متکی بر یک سری اطلاعات (مانند مدل) پیرامون کاربر، وظیفه در حال انجام توسط کاربر و وضعیت خود سیستم باشد. هر مدل، نیازمند انواع مختلفی از ورودیها است تا اینکه بتواند عمل ابتکاری خود را آغاز کند. تاکتیک‌های ابتکار با سیستم، آنهایی هستند که مدلهایی را شناسایی می‌کنند تا به وسیله آن سیستم پیش بینی کند که آیا رفتار متعلق به خودش است یا در ارتباط با کاربر است. محصورسازی این اطلاعات، معمار را قادر خواهد ساخت که به سهولت مدل‌ها را دستکاری کند. دستکاریها می‌تواند به صورت پویا مبتنی بر رفتار گذشته کاربر یا برون خط در طول توسعه، در نظر گرفته شوند.

- *نگهداری یک مدل از وظیفه*[1]

در این مورد، مدل نگهداری شده مربوط به وظیفه‌ای است که در سیستم در حال انجام شدن است. این مدل کمک می کند که سیستم بتواند ایده‌هایی در مورد اینکه کاربر در حال حاضر مشغول انجام چه کاری است را داشته باشد و اینکه بر اساس همان مدل کمک‌ها و راهنمایی‌هایی برای کاربر فراهم آورد. برای مثال دانستن اینکه هر جمله معمولا با یک کاراکتر حروف بزرگ شروع می‌شود به یک برنامه کاربردی اجازه می‌دهد که کاراکتر حروف کوچک در آن موقعیت را اصلاح کند.

- *نگهداری یک مدل از کاربر*

در این مورد، مدل نگهداری شده مربوط به کاربر است. این مدل دانش کاربر از سیستم، رفتارهای کاربر بر حسب زمان پاسخ مورد انتظار و دیگر جنبه‌های خاص مربوط به کاربر و کلاس کاربر را مشخص می‌کند. برای مثال، نگهداری یک مدل کاربر به سیستم اجازه می‌دهد که در صفحه‌ای که توسط کاربر در حال خواندن است حرکت آرامی داشته باشد تا آن صفحه سریع و قبل از خوانده شدن ناپدید نشود و یا اینکه به صفحه دیگر انتقال نیابد.

- *نگهداری یک مدل از سیستم*

در این مورد، مدل نگهداری شده مربوط به سیستم است. این مدل رفتارهای مورد انتظار سیستم را در جهت اینکه بازخوردهای مناسب به کاربر داده شود، مشخص می‌کند. مدل سیستم حالت‌هایی مانند اینکه چقدر زمان طول خواهد کشید که فعالیت جاری به پایان برسد را پیش‌بینی می‌کند.

[1] Maintain a model of the task

۵-۷-۲- تاکتیک‌های زمان طراحی

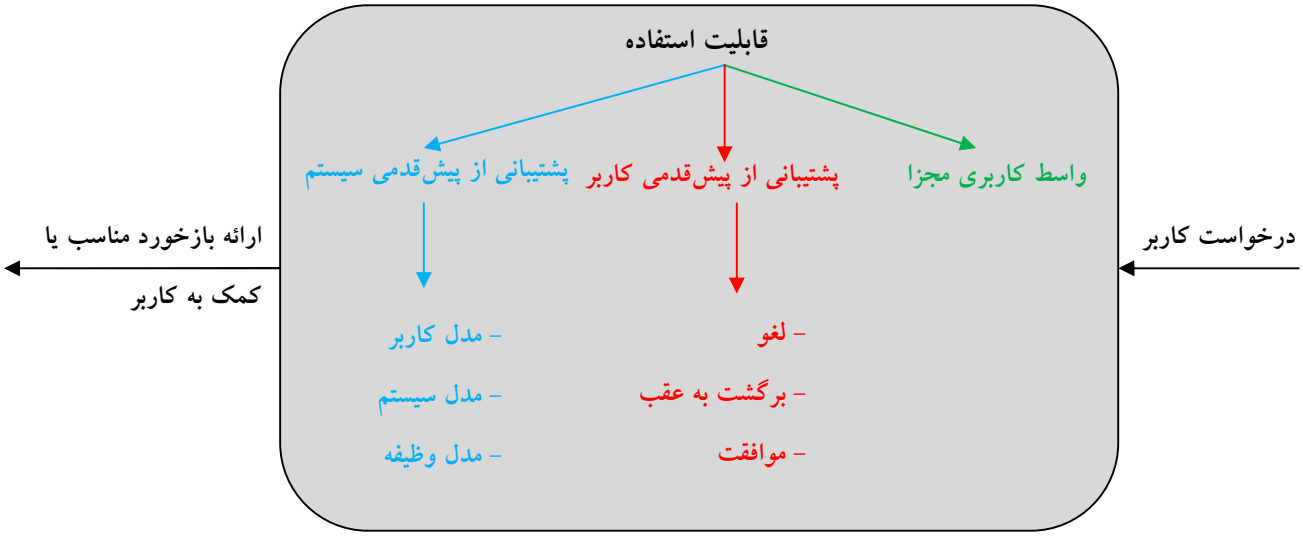
واسط‌های کاربری معمولا به صورت مکرر در فرآیند آزمایش اصلاح می‌شوند. یعنی اینکه، مهندس قابلیت استفاده اصلاحیه‌هایی به توسعه‌دهندگان در ارتباط با طراحی واسط کاربری موجود می‌دهد و توسعه‌دهندگان آن تغییرات را در واسط کاربری اعمال می‌کنند. این عمل منجر به یک تاکتیکی می‌شود که حالت پالایش شده تاکتیک انسجام معنایی برای قابلیت اصلاح است.

- *جدا کردن واسط کاربری از مابقی برنامه کاربردی*

محلی کردن تغییرات پیش‌بینی شده اساس و پایه انسجام معنایی است. از آنجایی که انتظار می‌رود واسط کاربری مکررا در طول توسعه و بعد از استقرار تغییر پیدا کند نگهداری کد مربوط به واسط کاربری به صورت مجزا باعث می‌شود که تغییرات در داخل آن متمرکز شوند. الگوهای معماری نرم‌افزار که به منظور پیاده‌سازی این تاکتیک و حمایت از قابلیت اصلاح واسط کاربری توسعه داده شده‌اند عبارتند از:

Model-View-Controller, Presentation-Abstraction-Control, Seeheim, Arch/Slinky

شکل ۵-۱۳ خلاصه‌ای از تاکتیک‌های زمان اجرا برای دستیابی به قابلیت استفاده را نشان می دهد.



شکل ۵-۱۳: خلاصه‌ای از تاکتیک‌های زمان اجرای برای قابلیت استفاده

۵-۸- رابطه تاکتیک‌ها با الگوهای معماری

یک معمار با استفاده از تاکتیک‌ها می‌تواند به ویژگیهای خاص مورد انتظار در سیستم دست پیدا کند. در حقیقت معمار یک الگو یا مجموعه‌ای از الگوهای طراحی شده را انتخاب می‌کند که این الگوها در درون خود یک یا بیش از یک تاکتیک را تحقق می‌دهند. این موضوع می‌تواند توسط الگوی طراحی شئ فعال[1] تشریح شود.

الگوی طراحی شیء فعال اجرای متد را از درخواست متد جدا می‌کند تا همزمانی را افزایش داده و دسترسی همگام شده به اشیائی که مقیم در نخ کنترل خودشان هستند را ساده کند.

الگو طراحی شیء فعال متشکل از شش عنصر است:

۱. *پراکسی*: واسطی فراهم می‌کند تا سرویس‌گیرنده‌ها متدهای قابل دسترس عمومی بر روی شئ فعال را درخواست کنند.
۲. یک *درخواست متد*: واسطی برای اجرا متدها از یک شیء فعال فراهم می‌کند.
۳. یک *لیست فعالسازی*: بافری برای درخواست متدهای معوقه فراهم می‌کند.
۴. یک *زمانبند*: مشخص می‌کند کدام درخواست‌های متد به عنوان درخواست بعدی اجرا شود.
۵. یک *خادم*[2]: رفتار و وضعیت مدل شده را به عنوان یک شیء فعال تعریف می‌کند.
۶. یک *آینده*[3]: اجازه می‌دهد سرویس‌گیرنده‌ها نتایج درخواست متد خود را بدست آوردند.

دلیل ایجاد این الگو، افزایش همزمانی است (که یک هدف مربوط به کارایی است). بنابراین هدف اصلی از آن، پیاده‌سازی تاکتیک "معرفی همزمانی" مربوط به کارایی است. دیگر تاکتیک‌هایی که این الگو شامل آنها می‌شود به شرح زیر هستند:

- *پنهان سازی اطلاعات (مربوط به قابلیت اصلاح)*:

هر عنصر مسئولیت‌هایی را که می‌خواهد به آنها دست یابد انتخاب می‌کند و دست یافته‌هایش را در پشت واسط پنهان می‌کند.

[1] Active object design pattern
[2] Servent
[3] Future

- *واسطه (مربوط به قابلیت اصلاح)*:

پراکسی به عنوان یک واسطه عمل می‌کند که تغییرات در فراخوانی متد را بافر خواهد کرد.

- *زمان انقیاد (مربوط به قابلیت اصلاح)*:

الگوی شئ فعال فرض می‌کند که درخواست برای شیء در زمان اجرا به شیء می‌رسد. به هر حال، انقیاد سرویس‌گیرنده به پراکسی بر حسب زمان انقیاد، باز باقی می‌ماند.

- *سیاست زمانبندی (مربوط به کارایی)*:

زمانبند بعضی سیاست‌های زمانبندی را پیاده‌سازی می‌کند.

هر الگو چندین تاکتیک را پیاده‌سازی می‌کند و معمولا هم در ارتباط با خصوصیات کیفی متفاوت است. همچنین هر پیاده‌سازی از الگو، انتخاب‌هایی در مورد تاکتیک‌ها ایجاد می‌کند. برای مثال یک پیاده‌سازی می‌تواند یک ثبت وقایع[1] برای درخواست‌های انجام شده از شیء فعال به منظور حمایت از بازیابی در نظر بگیرد، یک رد ممیزی را نگهداری کند و یا اینکه قابلیت آزمایش را پشتیبانی کند.

فرآیند تحلیل برای معمار دربرگیرنده فهم همه تاکتیک‌هایی است که در یک پیاده‌سازی درگیر هستند و فرآیند طراحی دربرگیرنده ایجاد یک تصمیم قوی[2] پیرامون اینکه چه ترکیباتی از تاکتیک‌ها منجر به این خواهد شد که سیستم به اهداف خود دست یابد.

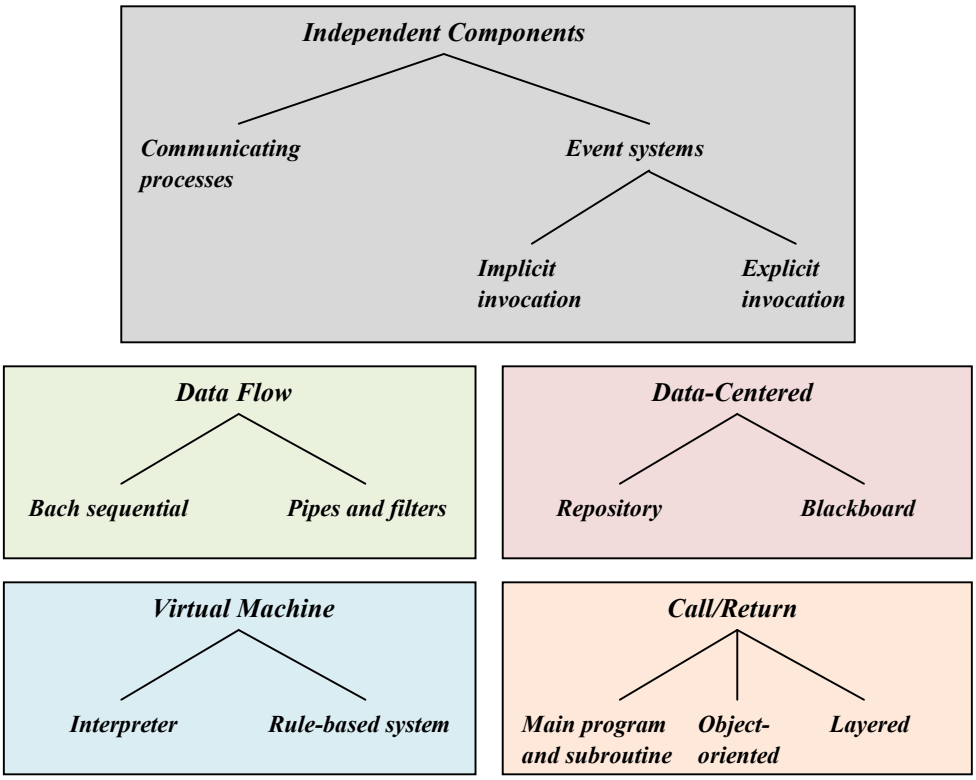
۵-۹- سبک‌ها[3] و الگوهای[4] معماری

الگوی معماری در نرم‌افزار که به سبک معماری نیز مشهور است مشابه سبک معماری در ساختمان‌سازی است (مانند سبک معماری گوتیک یا سبک یونانی). الگوی معماری متشکل از یک تعداد ویژگیهای کلیدی و قوانینی برای ترکیب آنها در جهت حفظ یکپارچگی معماری است. یک الگوی معماری با موارد زیر مشخص می‌شود:

[1] Log
[2] Judicious choice
[3] styles
[4] Patterns

- یک مجموعه از انواع عناصر (مانند یک مخزن داده یا مولفه که یک تابع ریاضی را محاسبه می‌کند)
- یک نقشه توپولوژی از عناصر شامل روابط داخلی آنها
- یک مجموعه از اجبارهای معنائی (به عنوان مثال فیلترها در سبک *pipe-and-filter* مبدل ضعیف داده هستند. آنها به صورت افزایشی جریان ورودی را به جریان خروجی تبدیل می‌کنند اما هیچ کنترلی بر روی نرخ بالا دستی یا نرخ پایین دستی عناصر ندارند)
- یک مجموعه از مکانیزم‌های تعاملی (مانند فراخوانی زیرروال، مشترک یک رویداد[1]، تخته سیاه) که مشخص می‌کنند چگونه عناصر از طریق توپولوژی تعیین شده هماهنگ می‌شوند.

شکل ۵-۱۴ یک دسته‌بندی از الگوهای معماری را نشان می‌دهد که با استفاده از رابطه *"is-a"* سازماندهی شده‌اند.



شکل ۵-۱۴: دسته‌بندی از الگوهای معماری

[1] Event-subscriber

۵-۹-۱- سبک مولفه‌های مستقل[1]

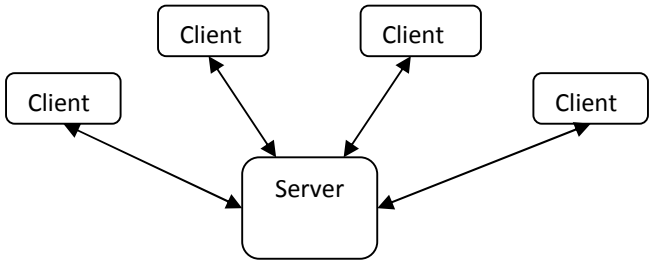
هدف از سبک مولفه‌های مستقل دستیابی به قابلیت اصلاح از طریق جداسازی بخش‌های مختلف محاسباتی و قابلیت توسعه است. این سبک از فرآیندهای مختلف تشکیل شده است که از طریق ارسال پیام با یکدیگر ارتباط برقرار می‌کنند. فرآیندها به یکدیگر داده ارسال می‌کنند ولی یکدیگر را مستقیما کنترل نمی‌کنند. پیامها می‌توانند به یک مشترک مشخص فرستاده شوند یا می‌تواند بین مشترک‌های غیرمشخص مبادله شوند.

۵-۹-۱-۱- سبک فرآیندهای ارتباطی[2]

در این سبک تعاملات بین مولفه‌ها از طریق ارسال پیام صورت می‌پذیرد. مولفه‌ها واحدهای موازی مانند فرآیند یا نخ هستند. اتصال‌دهنده‌ها نیز ارسال پیام یا هماهنگ‌ساز هستند. این سبک بیشتر در سرویس‌های مربوط به سیستم عامل یا در سیستم‌های موازی به کار می‌رود. یک زیر سبک از سبک فرآیندهای ارتباطی سبک سرویس‌گیرنده و سرویس‌دهنده است. تمرکز این زیر سبک بر روی سرویس‌های متفاوتی است که سرویس‌گیرنده‌ها می‌خواهند اجرا نمایند. این سبک معماری، در مواقعی کاربرد بیشتری پیدا می‌کند که تجهیزات سخت‌افزاری به صورت تعدادی از کامپیوترهای محلی سازماندهی شده‌اند و یک منبع مرکزی مانند درخت فایل، پایگاه داده و یا یک منبع محاسباتی قوی موجود است.

سرویس‌دهنده می‌تواند داده را به یک یا بیشتر از یک سرویس‌گیرنده منتقل کند. سرویس‌گیرنده به سرویس‌دهنده تقاضا می‌دهد اگر سرویس‌دهنده سنکرون عمل کند در همان زمان که داده را ارسال می‌کند کنترل را به مشتری برمی‌گرداند و اما اگر آسنکرون عمل کند فقط داده را به سرویس‌گیرنده می‌دهد. هدف اصلی در این سبک علاوه بر قابلیت اصلاح، رسیدن به مقیاس‌پذیری بالا است. شمای کلی سبک سرویس‌گیرنده سرویس‌دهنده در شکل ۵-۱۵ نشان داده شده است.

[1] Independent components
[2] Communicating processes



شکل ۵-۱۵: سبک سرویس‌گیرنده و سرویس‌دهنده

۵-۹-۱-۲- سیستم‌های برپایه رویداد[1]

سیستم‌های بر پایه رویداد زیر سبک‌هایی هستند که در آنها، کنترل بخشی از مدل است. هر مولفه داده‌هایی را که می‌خواهد با محیطش به اشتراک بگذارد اعلام می‌کند و بقیه‌ی مولفه‌ها می‌توانند در کلاس داده آن مقداری را ثبت کنند (مولفه‌ها مقادیری را ثبت می‌کنند که می‌خواهند دریافتشان کنند). اگر اینکار را انجام دهند هر زمان داده‌ای پدیدار شد فراخوانی می‌شوند و داده را دریافت می‌کنند. در این روش، مدیریت پیام ممکن است مولفه‌هایی که به آنها پیام ارسال می‌کند را کنترل نکند. این سبک برای این مهم است که هر مولفه را از شناسائی مولفه‌های دیگر بی نیاز می‌کند. همچنین مولفه‌ها می‌توانند به طور موازی کار کنند و فقط در زمان نیاز با یکدیگر مقادیری از داده را رد و بدل کنند.

سبک فراخوانی ضمنی[2]:

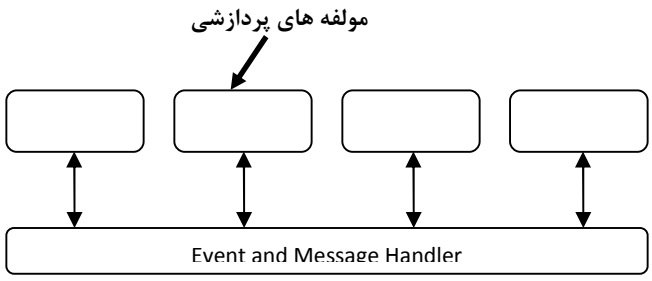
عموما در سیستم‌هایی که واسطهای مولفه‌ها، مجموعه‌ای از رویه‌ها و توابع را فراهم می‌آورند مولفه‌ها به صورت فراخوانی صریح و روشن آن روتین‌ها با هم ارتباط برقرار می‌کنند. اما نوع دیگری که تحت عنوان فراخوانی و یا درخواست ضمنی از آن یاد می‌شود به گونه‌ای است که بجای آنکه فراخوانی رویه‌ها مستقیما انجام شود یک مولفه می‌تواند درخواست خود مبنی بر یک رویداد را با کمک رویه‌ای مبنی بر آن رویداد در سیستم ثبت کند. هنگامی که آن رویداد منتشر شد خود سیستم تمام رویه‌هایی که برای آن رویداد ثبت شده بودند را فراخوانی می‌نماید. بنابراین انتشار یک رویداد به صورت ضمنی موجب فراخوانی رویه‌هایی در دیگر مولفه‌ها می‌شود. مولفه‌ها در سبک فراخوانی ضمنی ماژولهایی هستند که واسطهای آنها مجموعه‌ای از رویه‌ها و

[1] Event systems
[2] Implicit invocation

رویدادها را فراهم می‌آورند. رویه‌ها برای فراخوانی معمولی کاربرد دارند و علاوه بر آن مولفه‌ها می‌توانند رویه‌های خود را با رویدادهایی در سیستم ثبت نماید. این باعث می‌شود که آن رویه‌ها با انتشار و رخ دادن رویدادهای مربوط به خود در زمان اجرا فراخوانده شوند. بنابراین اتصال‌دهنده‌ها در فراخوانی ضمنی شامل فراخوانی رویه‌ها و یا انتشار رویدادها هستند.

نکته مهم این سبک در آن است که انتشار دهندگان رویدادها نمی‌دانند که کدام مولفه‌ها با آن رویدادها درگیر هستند و از طرفی مولفه‌ها نمی‌توانند فرضیاتی از نتیجه اجرای رویدادها و یا پردازش آنها داشته باشند. بنابراین به این دلیل است که اکثر سیستم‌های فراخوانی ضمنی دارای فراخوانی صریح هم هستند و به این صورت حالت تبادل یا ارتباط را کامل می‌کنند. (فراخوانی نرمال رویه). در این سبک می‌توان مولفه‌ای را بدون اینکه بر واسطهای دیگر مولفه‌ها تأثیری داشته باشد جایگزین نمود. از طرف دیگر در این سبک مولفه‌ها کنترلی ندارد بر روی اتفاقاتی که در سیستم رخ می‌دهد. به عنوان مثال هنگامی که مولفه‌ای رویدادی را منتشر می‌کند درک و دیدی از مولفه‌های دیگر ندارد که به آن پاسخ می‌دهند و همچنین نمی‌داند که چه موقع عملیات آنها پایان یافته است. بعضی مواقع داده از طریق رویدادها فرستاده می‌شوند. زمانی که رویدادها در یک مخزن مشترک قرار می‌گیرند تا مبادله شوند کارایی عمومی سیستم و مدیریت منابع وضع وخیمی پیدا می‌کند.

مزایا این سبک حمایت از قابلیت استفاده و سادگی آن است و معایب آن نیز اینکه در این سبک کنترل سیستم‌ها سخت بوده و اینکه ترتیب اجرا قابل کنترل نیست. شمای کلی این سبک در شکل ۵-۱۶ نشان داده شده است (نمونه کاربردی از این سبک سیستم *X windows* است).

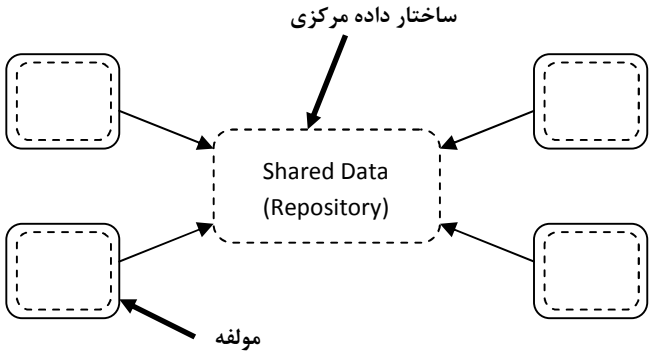


شکل ۵-۱۶: سبک فراخوانی ضمنی

۵-۹-۲- سبک متمرکز بر روی داده[1]

عبارت "متمرکز روی داده" اشاره به سیستم‌هایی دارد که در آنها دستیابی و بهنگام‌سازی داده‌ها، توصیفی مناسب از عملکرد سیستم است. در این سبک دو نوع کاملا مجزا از مولفه‌ها وجود دارد. اولی یک ساختار داده مرکزی است که وضعیت اخیر داده‌ها را نمایش می‌دهد و دیگری مجموعه‌ای مستقل از مولفه‌ها است که با منبع داده در تعاملند. تعاملات میان منبع داده و مولفه‌های خارجی آن، در میان سیستم‌های گوناگون متفاوت و متنوع است. اتصال‌دهنده‌ها در این سبک همان ارتباطات میان مولفه‌های مستقل با ساختار داده مرکزی به منظور تبادل داده و کنترل هستند.

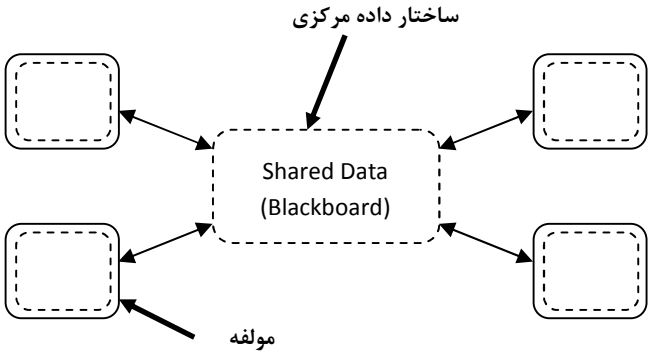
در این سبک مولفه‌ها روی مجموعه رشته‌ی کنترلی مستقلی اجرا می‌شوند. داده مشترکی که توسط کلیه مولفه‌ها دستیابی می‌شود می‌تواند به صورت یک منبع غیرفعال (مخزن[2]) و یا یک منبع فعال (تخته سیاه[3]) عمل کند. در حالتی که منبع داده غیر فعال است این سبک به سبک مخزن معروف است. در این حالت ارتباط یک طرفه‌ای میان مولفه‌ها و منبع داده وجود دارد. شمای کلی این سبک در شکل ۵-۱۷ نشان داده شده است.



شکل ۵-۱۷: سبک متمرکز روی داده (مخزن)

در حالتی که منبع داده فعال است این سبک به سبک تخته سیاه معروف است. در این حالت منبع داده یا همان تخته سیاه در زمان تغییرات در داده‌ها، برای مولفه‌ها پیام می‌فرستد. پس ارتباط میان مولفه‌ها و تخته سیاه دو طرفه است. شمای کلی این سبک در شکل ۵-۱۸ نشان داده شده است.

[1] Data-centered
[2] Reposaitory
[3] Blackboard



شکل ۵-۱۸: سبک متمرکز روی داده (تخته سیاه)

این سبک معماری همواره در حال گسترش و ارتقاء اهمیت است و این بخاطر وجود راه حلی ساختاری برای رسیدن به قابلیت یکپارچگی و یا قابلیت تجمیع پذیری است. مزیت عمده این سبک اینکه مولفه‌ها به صورت مستقل از هم وجود دارند و داده مشترک نیز بخشی مستقل از مولفه‌ها است بنابراین مقایس‌پذیر است و مولفه‌های جدید می‌توانند به راحتی اضافه شوند.

سبک متمرکز بر روی داده قابلیت اصلاح بالایی دارد و این بخاطر امکان تغییر عملکرد هر یک از مولفه‌ها بدون داشتن اثراتی روی بقیه مولفه‌ها است. در این سبک چنانچه بین مولفه‌ها اتصال برقرار باشد باعث کاهش قابلیت اصلاح می‌شود ولی در عوض کارایی را افزایش می دهد.

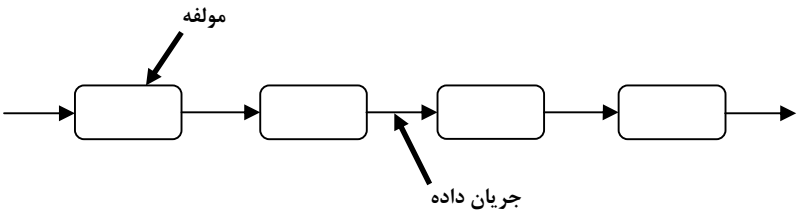
۵-۹-۳- سبک جریان داده[1]

در این سبک یک سری از تبدیلات روی قطعاتی متوالی از داده‌های ورودی انجام می‌شود. داده به سیستم وارد می‌شود و تا زمانی که به مقصد نهایی برسد در میان مولفه‌ها جریان می‌یابد. مقصد نهایی می‌تواند مخزنی از اطلاعات یا یک خروجی در نظر گرفته شود. سبک جربان داده شامل دو زیر سبک دسته‌ای ترتیبی[2] و لوله و فیلتر[3] است هدف از سبک جریان داده رسیدن به ویژگی استفاده مجدد و قابلیت اصلاح است.

در سبک دسته‌ای ترتیبی، گام‌های پردازش که همان مولفه‌ها هستند برنامه‌های مستقلی می‌باشند. در این سبک فرض بر این است که هر گام قبل از شروع گام بعدی اجرایش به پایان می‌رسد. اتصال‌دهنده‌ها در این سبک،

[1] Data flow
[2] Batch sequential
[3] Pipe and filter

اتصالات بین گامهای پردازش هستند که به منظور انتقال داده گامها را به یکدیگر متصل می نمایند. بنابراین به کمک این اتصال‌دهنده‌ها هر دسته‌ای از داده‌ها می‌تواند بین گام‌ها انتقال داده شود. فرم این سبک فرم کلاسیک پردازش داده است. شمای کلی این سبک در شکل ۵-۱۹ نشان داده شده است.



شکل ۵-۱۹: سبک جریان داده (دسته‌ای ترتیبی)

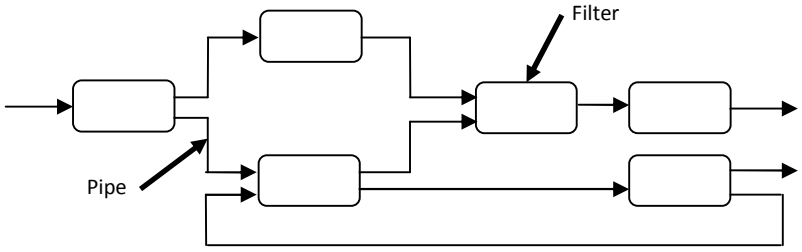
در سبک لوله و فیلتر هر مولفه مجموعه‌ای از ورودی‌ها و خروجی‌ها دارد. هر مولفه جریان‌های داده‌ای را از ورودی‌های خود می‌خواند و یکسری تبدیلات و تغییرات داخلی روی آنها ایجاد می‌نماید سپس جریانهای داده‌ای در خروجی خود فراهم می‌آورد که نحوه تحویل دادن آن داده‌ها بر اساس قواعد خاصی مشخص می‌شود. فیلترها در این سبک همان مولفه‌ها هستند و لوله‌ها در این سبک اتصال‌دهنده‌ها را تشکیل می‌دهند. اتصال‌دهنده‌ها در این سبک حالتی به خود نمی‌گیرند و فقط به عنوان یک کانال برای جریان داده مورد استفاده قرار می‌گیرند. در واقع اتصال‌دهنده‌ها خروجی‌های یک مولفه را به ورودی‌های مولفه دیگر منتقل می‌کنند. نحوه و چگونگی ترکیب شدن لوله‌ها و فیلترها با یکدیگر براساس قواعدی است که روی این سبک حاکم‌اند.

در این سبک فیلترها باید به صورت موجودیت‌های مستقل عمل کنند مخصوصاً آنها نمی‌توانند با یکدیگر وضعیت مشترک داشته باشند. هر فیلتر از ماهیت فیلتر قبلی و بعدی خود در جریان انتقال داده اطلاعی ندارد و مشخصات هر فیلتر محدود می‌شود به هر آنچه که در ورودی خود دارد و هر آنچه که در خروجی خود بوجود خواهد آورد. در این سبک لوله‌ها تنها مسیرهایی هستند که مولفه‌ها می‌توانند با یکدیگر ارتباط برقرار نمایند و همچنین اینکه کنترل به صورت توزیع شده است و هر مولفه تنها زمانی اجرا می‌شود که داده برای پردازش خود داشته باشد.

این سبک این امکان را برای طراح فراهم می‌کند تا بتواند یک برداشت کلی از رفتار سیستم و ورودی و خروجی آن داشته باشد و سیستم را به صورت ترکیب ساده‌ای از رفتارهای مولفه‌ها یا همان فیلترها مورد

بررسی قرار دهد. از آنجا که فیلترها به صورت جعبه سیاه دیده می‌شوند قابلیت اصلاح و استفاده مجدد بالایی دارند. لوله‌ها و فیلترها می‌توانند به صورت سلسله مراتبی ساخته شوند و هر ترکیبی از لوله‌ها و فیلترها به عنوان یک فیلتر بسته‌بندی شود. همچنین با توجه به اینکه هر فیلتر می‌تواند ورودی خود را به تنهایی و جدا از بقیه سیستم پردازش کند، امکان توازی و توزیع شدن وجود دارد.

از طرف دیگر در این سبک بخاطر شکسته شدن مسأله به این شکل، مفهوم و دید پردازش دسته‌ای القا می‌شود، بنابراین برای کاربردهای محاوره‌ای خوب نیست و با توجه به اینکه ترتیب دادن به فیلترها مشکل است و فیلترها نمی‌توانند برای حل مسئله با هم محاوره داشته باشند بنابراین کارایی پایین می‌آید. شمای کلی این سبک در شکل ۵-۲۰ نشان داده شده است.



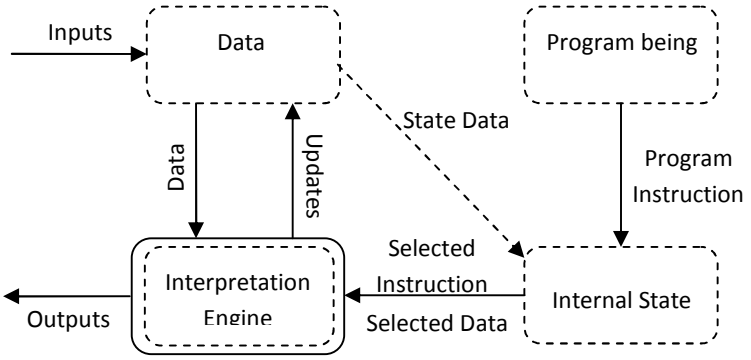
شکل ۵-۲۰: سبک جریان داده (لوله و فیلتر)

۵-۹-۴- سبک ماشین مجازی[1]

این سبک، عملکردی از یک برنامه قابل اجرا را که برای سخت‌افزار یا نرم‌افزاری ناآشنا است شبیه‌سازی می‌کند. شبیه‌سازی به روشهای مختلفی انجام می‌شود. سبک ماشین مجازی می‌تواند سکوهایی که هنوز ساخته نشده‌اند را شبیه‌سازی کند (مانند سخت‌افزار جدید) و یا اینکه می‌تواند حالتهای که ممکن است در عمل بسیار پیچیده، پر هزینه و خطرناک باشند را شبیه‌سازی نماید. (مثل سیستم‌های بحرانی). بنابرانی هدف اصلی از سبک ماشین مجازی رسیدن به قابلیت حمل بالا است. زیرسبک‌های سبک ماشین مجازی مفسرها[2] و سیستم‌های قاعده‌مند[3] هستند. به عنوان مثال زبان جاوا روی ماشین مجازی جاوا اجرا می‌شود که اجازه می‌دهد که زبان جاوا مستقل از سکو باشد.

[1] Virtual machine
[2] Interpreter
[3] Rule-based system

در زیر سبک مفسر، موتور تفسیر دستوری را از برنامه در حال تفسیر انتخاب می‌کند حالات داخلی خود را بهنگام می‌کند و منطبق بر دستورات، داده‌های برنامه را بهنگام می‌سازد. اجرای برنامه توسط مفسر، انعطاف پذیری یا وفق‌پذیری[1] را همراه با قابلیت داشتن وقفه، پرس‌وجو از برنامه و معرفی تغییرات در زمان اجرا را به ارمغان می‌آورد ولی بخاطر داشتن محاسبات اضافی درحین اجرا، کارایی کاهش می یابد. شمای کلی این سبک در شکل ۵-۲۱ نشان داده شده است.



شکل ۵-۲۱: سبک ماشین مجازی

۵-۹-۵- سبک فراخوانی و بازگشت[2]

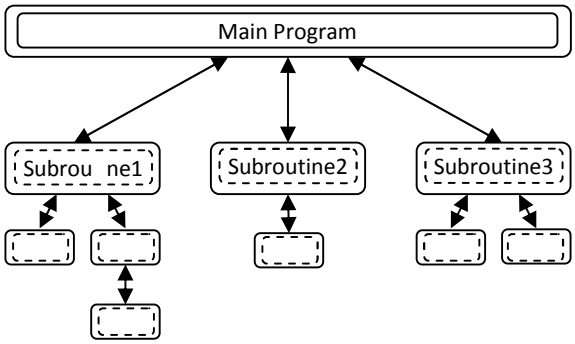
هدف اصلی از این سبک دستیابی به قابلیت اصلاح و مقیاس‌پذیری است. سبک فراخوانی و بازگشت وقتی استفاده می‌شود که ترتیب محاسبات ثابت است. این سبک به سه زیر سبک تقسیم می‌شود.

۵-۹-۵-۱- سبک برنامه اصلی و زیرروال[3]

این سبک همان برنامه‌سازی به روش کلاسیک را نشان می‌دهد. هدف از آن تقسیم کردن برنامه به قطعات کوچکتر به منظور دستیابی به قابلیت اصلاح بالاتر است. یک برنامه در این سبک به صورت سلسله مراتبی تقسیم می‌شود و دربرگیرنده یک رشته واحد از کنترل است. هر مولفه در این سلسله مراتب رشته کنترل را از پدر خود بدست می‌آورد و به فرزندان خود می‌دهد. مولفه‌ها در این سبک همان قطعات کوچک برنامه در طول تقسیم هستند و فراخوانی‌های میان این قطعات، که به نوعی آنها را به یکدیگر مربوط می‌سازند همان

[1] Adaptability
[2] Call and return
[3] Main program and subroutine

اتصال‌دهنده‌ها هستند. جریانهای کنترل در این سبک می‌توانند همراه با جریان داده نیز باشند. مزایای این سبک وجود یک منبع مشترک با دسترسی ترتیبی و داشتن یک جریان طبیعی از داده‌ها و الگوریتم‌های پردازش مستقل است. با این وجود، دادن تغییر کلی در آن مشکل بوده و تغییر دادن یا اضافه کردن یک ماژول همه ماژولهای سیستم را تحت تاثیر قرار می‌دهد. شمای کلی این سبک در شکل ۵-۲۲ نشان داده شده است.



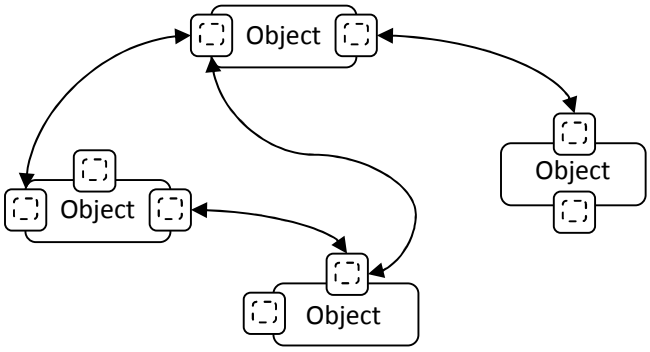
شکل ۵-۲۲: سبک برنامه اصلی و زیرروال

۵-۹-۵-۲- سبک شیء گرا

دراین سبک سیستم به صورت مجموعه‌ای از اشیاء با نوع داده مجرد در نظر گرفته می‌شود. هر شیء واسطهایی برای دیگر اشیاء در سیستم فراهم می‌کند. ارتباط بین اشیاء در این سیستم از طریق فرستادن پیام است. این سبک تکیه بر بسته‌بندی داده و داشتن دانشی در مورد چگونگی انجام عملیات و دستیابی بر داده‌ها دارد. بسته‌بندی شامل محصورسازی داده‌ها و پنهان‌سازی رموز داخلی داده‌ها از محیط است. به عبارت دیگر، اشیاء که در این سبک مولفه‌ها هستند داده را در خود محصورسازی نموده‌اند و مولفه‌ها یا اشیاء دیگر تنها از طریق فراخوانی واسطهای آن مولفه‌ها می‌توانند به داده‌های آنها دسترسی پیدا کنند. اتصال‌دهنده‌ها در این سبک درخواست‌های انجام یک سرویس در یک مولفه از سوی مولفه دیگر هستند.

در سبک شیء گرا از آنجایی که عملیات داخلی هر شیء مجصورسازی شده است لذا به راحتی می‌توان آن عملیات تغییر را داد بدون اینکه بر اشیاء دیگر تأثیر بگذارد. در نتیجه قابلیت استفاده مجدد و قابلیت اصلاح افزایش می‌یابد. به عنوان مثال کاربر یک سرویس نیازی ندارد بداند که چگونه یک سرویس کار می‌کند. می‌توان گفت خاصیت اصلی که وجه تمایز نمونه شئ‌گرا و انواع داده‌های مجرد است، وراثت و چند ریختی است.

از طرفی از آنجا که در این سبک هر شیء برای برقراری ارتباط با بقیه اشیاء از فراخوانی رویه استفاده می‌کند لذا دانستن هویت بقیه اشیاء برای آن شیء لازم است. بنابراین چنانچه هویت شیء تغییر کند لازم است تا بقیه اشیاء نیز از این موضوع آگاه شوند. شمای کلی این سبک در شکل ۵-۲۳ نشان داده شده است.



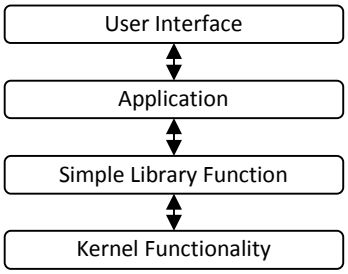
شکل ۵-۲۳: سبک شیءگرا

۵-۹-۵-۳- سبک لایه‌ای[1]

در این سبک، مولفه‌ها هر یک به لایه‌ای انتساب داده می‌شوند تا محاورات بین مولفه‌ها کنترل شود. در مدل محض از این سبک هر لایه فقط با لایه‌های مجاور خود ارتباط دارد. هر لایه بالاتر روی لایه قبلی خود ساخته می‌شود و لایه پایینی‌اش را پنهان می‌سازد و سرویس‌هایی را برای لایه‌های بالاتر فراهم می‌کند. پائین ترین لایه، عملیات هسته‌ای و مرکزی را فراهم می‌کند.

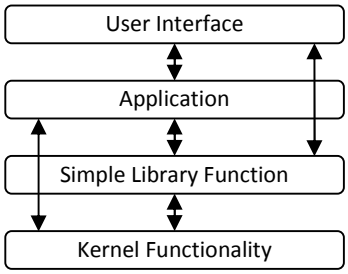
استفاده از سبک لایه‌ای باعث می‌شود که طراحی بر اساس افزایش سطوح تجریدی صورت گیرد و پیچیدگی یک مسأله به صورت مراحل پشت سر هم در آید. همچنین در این سبک تغییر در یک لایه منجر به تغییر در حداکثر دو لایه می‌شود و این باعث می‌شود که تغییرات ناشی از تغییر یک لایه به حداقل برسد و در نتیجه قابلیت استفاده مجدد و قابلیت اصلاح افزایش می‌یابد. از طرف دیگر به راحتی امکان‌پذیر نیست که کلیه سیستم‌ها را در قالب سبک لایه‌ای پیاده کرد و اگر هم این امر امکان‌پذیر باشد نیاز به کارایی باعث می‌شود که لایه‌ها با هم ارتباطات بیشتری داشته باشند. مزایای این سبک اینکه ترتیب لایه‌ها با سطوح مرتب شده و وابستگی بین سرویس‌ها وجود دارد. شمای کلی این سبک در شکل ۵-۲۴ نشان داده شده است.

[1] Layered



شکل ۵-۲۴: سبک لایه‌ای محض

در عمل، اغلب سیستم‌های لایه‌ای محض نیستند و عملیات در یک لایه، امکان ایجاد ارتباط با عملیات هر لایه‌ای را دارد. این امکان پلزنی لایه‌ای[1] نامیده می‌شود و هنگامی مورد استفاده قرار می‌گیرد که مسأله کارایی در زمان اجرا مورد نظر باشد. شمای کلی سبک لایه‌ای با این امکان در شکل ۵-۲۵ نشان داده شده است.



شکل ۵-۲۵: سبک لایه‌ای با پلزنی لایه‌ای

[1] Layer bridging

فهرست مطالب

فصل ششم

کنترل ترافیک هوایی: مطالعه موردی برای دستیابی به قابلیت دسترسی در طراحی

در این فصل یک مطالعه موردی مربوط به سیستم کنترل ترافیک هوایی مورد بررسی قرار خواهد گرفت که در آن مشخص خواهد شد که چگونه معماری، راهبردها و تاکتیک‌های خاص را در جهت تحقق نیازهای وظیفه‌مندی و خصوصیات کیفی به کار می‌برد.

۶-۱- مقدمه

سیستم کنترل ترافیک هوایی[1] از جمله نرم‌افزارهای کاربردی است که نیاز شدیدی به آن وجود دارد. این سیستم به شدت بلادرنگ بوده (یعنی باید زمانبندی مربوط به آخرین مهلت[2] (فرجه) همیشه رعایت شود)، از نظر ایمنی حالت بحرانی دارد (یعنی اگر سیستم درست عمل نکند زندگی انسانها به خطر می‌افتد)، شدیدا مورد توجه عموم مردم بوده و توزیع‌پذیری در آن زیاد است (یعنی سیستم نیازمند کنترل‌کننده‌های زیادی است که باید با یکدیگر در جهت کنترل هواپیما در سیستم هوایی همکاری کنند). علاوه بر این خصوصیات که سیستم باید دارای آن باشد مساله ساخت و نگهداری یک چنین سیستم هوایی امن و قابل اطمینان مطرح می‌شود که نیازمند هزینه مالی زیادی است که باید توسط ذینفعان آن فراهم شود (سیستم کنترل ترافیک هوایی یک سیستم چند میلیون دلاری است). در امریکا رفت و آمدهای هوایی به وسیله سازمان مدیریت هوایی فدرال[3] کنترل شده که نماینده دولت بوده و مسئولیت آن حفظ امنیت بخش هوایی کشور است (سازمان مدیریت هوایی فدرال در واقع مشتری سیستمی است که در این فصل بررسی خواهد شد).

از زمانی که هواپیما فرودگاه مبدا را ترک کرده و به فردوگاه مقصد می‌رسد با تعداد زیادی از کنترل کننده‌های ترافیک هوایی سروکار دارد که آن را در بخشهای مختلف هوایی در جهت یک پرواز امن راهنمایی می‌کنند. در این سیستم، کنترل کننده‌های زمینی مسئول هماهنگی هواپیماها در باند فرودگاه هستند و برجهای مراقبت نیز پرواز هواپیماها را در مناطق هوایی متعلق به خود کنترل می‌کنند. مراکز مسیریابی بین‌راهی[4] یا همان کنترل کننده‌های بین‌راهی نیز بخش‌های مختلف آسمان را کنترل می‌کنند در واقع هواپیماها را در خارج از نقاط تحت کنترل برجهای مراقبت، هدایت می‌کنند (آسمان امریکا به ۲۲ منطقه هوایی تقسیم شده است).

سیستمی که در این فصل مورد مطالعه قرار خواهد گفت، *ISSS*[5] نامیده می‌شود که توسعه یافته سخت‌افزاری و نرم‌افزاری برای ۲۲ مرکز کنترل بین‌راهی در امریکا است. در واقع *ISSS* یک بخشی از یک سیستم بزرگ دولتی به نام سیستم اتوماسیون پیشرفته[6] است که هدف از آن نصب سیستم‌های ارتقاء یافته در برج‌های مراقبت،

[1] Air Traffic Control (ATC)
[2] Deadline
[3] Federal Aviation Administration (FAA)
[4] En route centers
[5] Initial Sector Suite System
[6] Advanced Automation System (AAS)

ابزارهای کنترل زمینی و نیز مراکز کنترل هواپیمایی مربوط به گذر از اقیانوس‌ها[1] است. اینکه سیستم *ISSS* یک زیر بخشی از سیستم بزرگتری است به شدت معماری آن را تحت تاثیر قرار می‌دهد. زیرا باید عناصر و طراحی سیستم سازگار با سیستم اصلی در نظر گرفته شوند و همچنین اینکه سیستمهای مختلفی که جزو سیستم اصلی هستند عناصری مشترکی دارند که باید در طراحی *ISSS* مورد توجه قرار بگیرند (مانند واسط سیستم‌های رایویی یا واسط پایگاه داده طرح پرواز). بنابراین طراحی *ISSS* به صورت گسترده تحت تاثیر نیازمندیهای مربوط به کلیه زیر سیستم‌ها است.

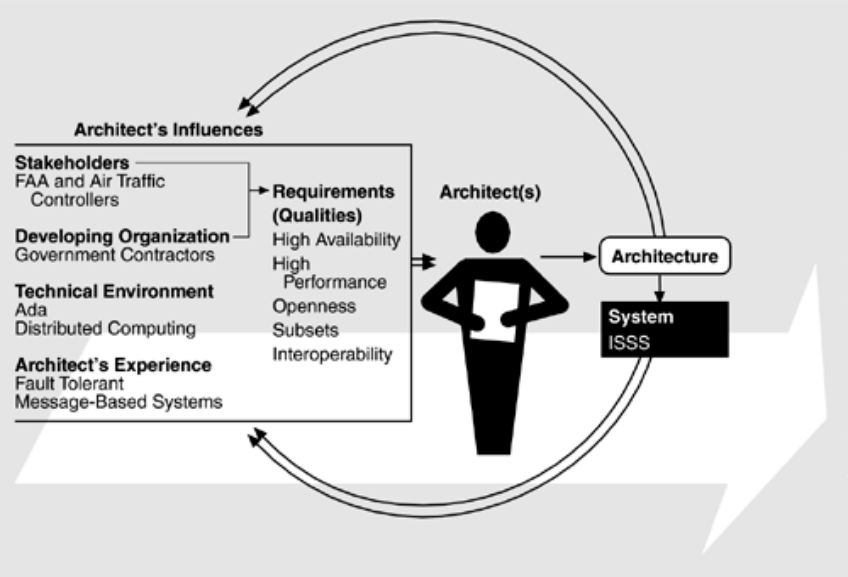
در این فصل یک مطالعه موردی که بخشی از یک سیستم هوایی است مورد بررسی قرار خواهد گرفت (با در نظر گرفتن دیدهای مشخص شده در فصل دوم و تاکتیک‌های مطرح شده در فصل پنجم) و مشخص خواهد شد که چگونه معماری ویژگیهای کلیدی در جهت برآورده ساختن درخواست‌ها و نیازهای پویا و گوناگون سیستم فراهم می‌کند.

۶-۲- چرخه حرفه معماری سیستم *ISSS*

شکل ۶-۱ چرخه حرفه معماری سیستم کنترل ترافیک هوایی را نشان می‌دهد. در شکل ۶-۱ بخشهای مختلف چرخه حرفه معماری به صورت زیر بیان شده‌اند:

- **کاربران نهایی**، کنترل کننده‌های ترافیک هوایی فدرال هستند.
- **توسعه‌دهنده**، یک شرکت بزرگی است که بسیاری از سیستم‌های شدیدا نرم‌افزاری[2] مهم را برای دولت آمریکا فراهم می‌کند (مشتری سازمان مدیریت هوایی فدرال است).
- **محیط فنی**، دربرگیرنده *Ada* به عنوان زبان پیاده‌سازی برای سیستم‌های نرم‌افزاری بزرگ و محاسبات توزیع[3] شده است.
- **تجارب معمار**، در زمینه تحمل‌پذیری خطا[4] و سیستم‌های مبتنی بر پیام[5] است.

[1] Transoceans
[2] Software-intensive systems
[3] Distributed computing
[4] Fault tolerant
[5] Message-based systems



شکل ۶-۱: چرخه حرفه معماری مربوط به سیستم کنترل ترافیک هوایی

۶-۳- نیازمندیها و خصوصیات کیفی

در نظر بگیرید که سیستم کنترل ترافیک هوایی سیستمی با اهمیت بالا برای عموم مردم، بسیار مهم از نظر تجاری و غیرنظامی و دارای حساسیت بالا برای دولت است و همچنین که اگر سیستم درست کار نکند زندگی انسان‌ها کاملا به خطر خواهد افتاد. بنابراین دو نیازمندی کیفی مهم مبتنی بر این خواست‌ها به شرح زیر هستند:

۱. *قابلیت دسترسی بالا*: به این معنی است که سیستم نباید برای مدتی طولانی عاجز از انجام عملیات باشد. قابلیت دسترسی واقعی مورد نیاز برای سیستم *ISSS*، ۰.۹۹۹۹۹ است. این بدین معنی است که سیستم باید برای مدت کمتر از پنج دقیقه در سال غیرقابل دسترس باشد (اگر سیستم بتواند در زمان کمتر از ۱۰ ثانیه از حالت شکست خارج شده و به ارائه سرویس خود بپردازد زمان آن شکست به عنوان زمان غیرقابل دسترس برای سیستم در نظر گرفته نمی‌شود).

۲. *کارایی بالا*: به این معنی است که سیستم مجبور است عملکرد تعداد زیادی از هواپیماها را (۲۴۴۰ هواپیما) بدون از دست دادن هیچ کدام از آنها مورد پردازش قرار دهد. بنابراین شبکه این سیستم مجبور است توانایی انتقال بار بالا را داشته باشد و نرم‌افزار نیز مجبور است که عملیات مربوط را سریع و آن هم به صورت قابل پیش‌بینی محاسبه کند.

علاوه بر این دو خصوصیت کیفی ذکر شده، نیازمندیهای مهم دیگری نیز وجود دارند که با وجود داشتن اهمیت کمتر برای امنیت هواپیما و مسافران، تاثیری بسزایی در شکل‌گیری معماری و مبانی پیش زمینه ایجاد معماری دارند. این نیازمندیها به شرح زیر هستند:

- باز بودن، به این معنی است که سیستم مجبور است با مولفه‌های نرم‌افزاری توسعه‌یافته به صورت تجاری شامل عملکردهای مربوط به سیستم کنترل ترافیک هوایی و سرویس‌های پایه محاسباتی از قبیل بسته‌های نمایش گرافیکی، تعامل داشته باشد.
- توانایی سیستم برای توسعه به صورت تدریجی و تکاملی (یعنی به صورت زیرسیستمی) در جهت فائق آمدن بر مشکلات بودجه‌ای
- توانایی ایجاد تغییرات در عملکرد سیستم و کنترل به روز رسانی‌ها در سخت‌افزار و نرم‌افزار (مانند پردازنده جدید، دستگاه‌ها و راه‌اندازهای جدید، نسخه جدید از کامپایلر *Ada*)
- توانایی عمل کردن با واسط‌های مبهم متعلق به سیستمهای خارجی (سخت‌افزار و نرم‌افزار)، سیستم‌های قدیمی و سیستم‌هایی که در آینده تولید خواهند شد.

سیستم *ISSS* باید کلیه نیازمندیهای متعلق به بسیاری از ذینفعان را برآورده کند، مخصوصا نیازمندیهای مربوط به کنترل کننده‌ها که کاربران نهایی سیستم هستند. اما نکته‌ای وجود دارد که منجر به غیر عادی شدن این سیستم می‌شود و آن این است که کنترل کننده‌ها می‌توانند سیستم را نپذیرند، حتی اگر سیستم تمامی نیازمندیهای خود را برآورده کند. بنابراین کلیه این مسائل حاکی از مهم بودن فرآیند تشخیص نیازمندیها و طراحی سیستم است. سیستم *ISSS* یک سیستم بزرگ است و بعضی از دلایلی که منجر به آن می‌شوند که این سیستم بزرگ در نظر گرفته شود به شرح زیر هستند:

- سیستم *ISSS* طوری طراحی شده است تا از ۲۱۰ کنسول در هر مرکز مسیریابی بین‌راهی حمایت کند. هر کنسول دربرگیرنده پردازنده خاص خودش است (*IBM RS/6000*).
- نیازمندی سیستم *ISSS* ایجاب می‌کند که به طور همزمان از ردیابی ۴۰۰ تا ۲۴۴۰ هواپیما پشتیبانی کند.
- برای تسهیل در دریافت سیگنال از ۱۶ تا ۴۰ رادار استفاده می‌کند.
- هر مرکز ۶۰ تا ۹۰ مکان کنترل دارد (هر مکان هم دارای یک یا چندین کنسول است).
- کد پیاده‌سازی آن که به زبان *Ada* است در حدود یک میلیون خط است.

با توجه به نکات ذکر شده، به صورت خلاصه سیستم *ISSS* باید توانایی انجام عملیات زیر را داشته باشد:

- دریافت گزارش نهایی رادار که در سیستم کنترل ترافیک هوایی (سیستم کامپیوتر میزبان) ذخیره شده است.
- تبدیل گزارشهای رادار به منظور اینکه آنها را نمایش داده یا به کنسولها منتشر شوند (هر کنسول توانایی انتخاب هر نوع گزارش و نمایش هر بخش از گزارش را دارد).
- کنترل کردن هشدارهای تداخلدار (احتمال برخورد هواپیما) یا دادههای دیگری که به وسیله کامپیوتر میزبان ارسال شدهاند.
- ارتباط با میزبان برای ورود و بازیابی برنامههای پرواز
- توانایی کنترل و نظارت گسترده اطلاعات (نظیر مدیریت شبکه)، برای اینکه به مدیران سایت اجازه بازپیکربندی آنی سیستم در هنگام نصب داده شود.
- توانایی ضبط اطلاعات برای پخش مجدد در آینده
- فراهم نمودن ابزارهای رابط کاربر گرافیکی مناسب در کنسولها
- فراهم نمودن قابلیت پشتیبانگیری کمینه[1] در هنگام بروز خطا در میزبان، شبکه ارتباطی اصلی یا سنسورهای رادار اصلی

۶-۴- راه حل معمارانه

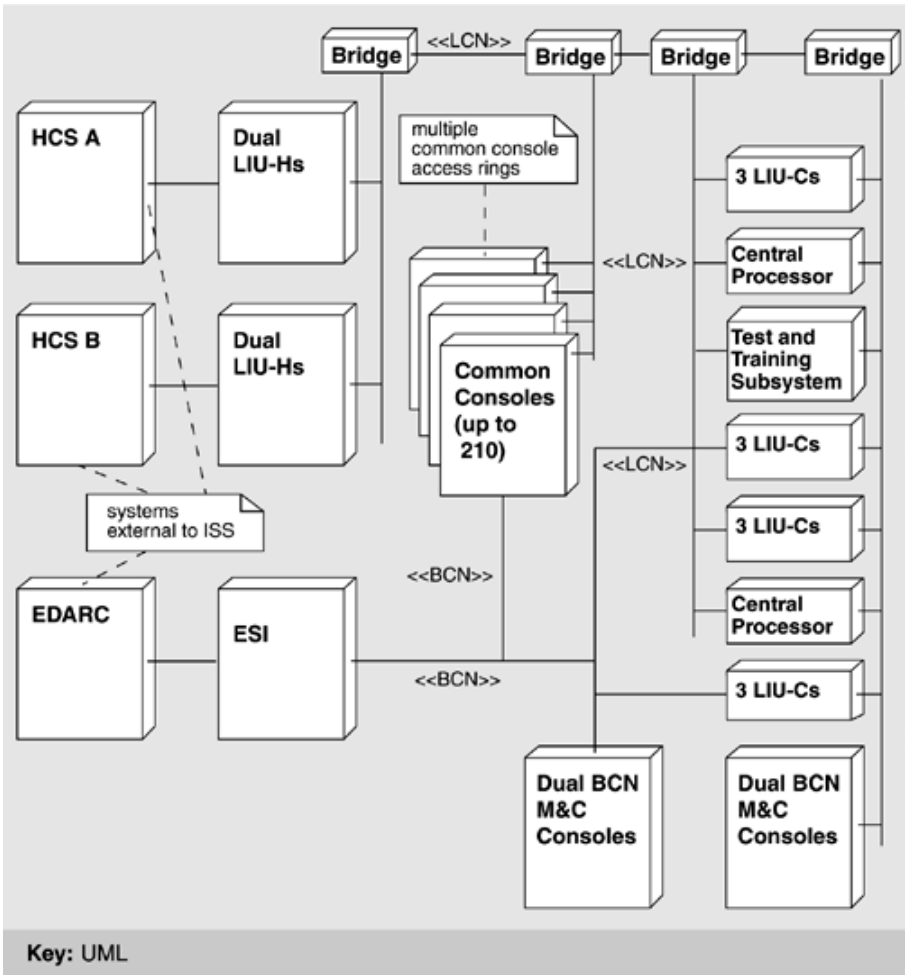
همچنانکه معماری، رفتار، کارایی، تحملپذیری خطا و قابلیت نگهداری سیستم را تحت تاثیر قرار میدهد خود نیز به وسیله نیازمندیهای دقیق در این حوزهها شکل میگیرد. در مورد سیستم *ISSS* نیز مهمترین پیشران، قابلیت دسترسی بالا است (عدم دسترسی سیستم در مجموع باید کمتر از ۵ دقیقه در سال باشد). بنابراین چنین نیازمندی نسبت به سایر نیازمندیهای سیستم *ISSS* ، تاثیر زیادی بر روی تصمیمات معماری خواهد گذاشت.

در ادامه برای به تصویر کشاندن معماری سیستم *ISSS* ابتدا محیط فیزیکی میزبان[2] نرمافزار تشریح خواهد شد سپس یک سری از دیدهای معماری مشخص شده و تاکتیکهای مربوط به آنها تشریح میشوند.

[1] Reduced
[2] Physical environment hosting

۶-۴-۱- دید فیزیکی *ISSS*

ISSS یک سیستم توزیع شده است که دربردارنده تعدادی عناصر است که از طریق یک شبکه محلی به یکدیگر متصل هستند. شکل ۶-۲ یک دید فیزیکی از سیستم *ISSS* را نشان می‌دهد.



شکل ۶-۲: دید فیزیکی سیستم *ISSS*

در شکل ۶-۲ هیچ نمایشی از سیستم‌های پشتیبان، واسط‌های آنها و همچنین ساختار نرم‌افزار وجود ندارد. عناصر اصلی دید فیزیکی و نقش‌هایی که هر کدام دارند به شرح زیر است:

- سیستم کامپیوتر میزبان که بخش اصلی سیستم خودکار مسیریابی بین‌راهی است. در هر مرکز مسیریابی بین‌راهی دو کامپیوتر میزبان وجود دارد که یکی به عنوان کامپیوتر اصلی است و دیگری به عنوان کامپیوتر پشتیبان تلقی می‌شود. کامپیوتر پشتیبان در صورت خرابی کامپیوتر اصلی به صورت خودکار

وارد عمل می‌شود و درخواست‌ها را پاسخ می‌دهد. کامپیوتر میزبان، پردازش مربوط به هر دو مورد از داده‌های برنامه پرواز و مراقبت را انجام می‌دهد. داده‌های مراقبت بر روی صفحه نمایش (کنسول) استفاده شده به وسیله کنترل کننده، نمایش داده می‌شود و داده‌های پرواز نیز در صورت لزوم بر روی نوار چاپگر هواپیما چاپ می‌شوند.

- کنسولهای مشترک[1]، ایستگاه‌های کنترل کننده ترافیک هوایی هستند. آنها اطلاعات موقعیتی هواپیما و برچسب‌های داده‌ای مرتبط را در قالب یک دید خاص و داده‌های برنامه پرواز را در قالب نوارهای الکترونیکی نمایش می‌دهند. به علاوه به کنترل کننده‌ها اجازه می‌دهند که داده‌های پرواز را تغییر داده و نیز اطلاعات نمایش داده شده و قالب آنها را کنترل کنند.
- کنسولهای مشترک به کامپیوتر میزبان از طریق شبکه ارتباط محلی[2] (شبکه اصلی *ISSS*) وصل می‌شوند. هر میزبان از طریق واسط دوتایی به نام *LIU-H*[3] به شبکه ارتباط محلی متصل می‌شود.
- شبکه ارتباط محلی از چهار شبکه حلقوی مبتنی بر نشانه و آن هم به صورت موازی، برای افزونگی و برقراری تعادل در بار کلی سیستم تشکیل شده است. یکی از شبکه‌ها عمل انتشار داده‌های مراقبت به کلیه پردازنده‌ها را بر عهده دارد، یک شبکه برای برقراری ارتباط نقطه به نقطه بین دو پردازنده استفاده می‌شود، شبکه دیگر نیز کانالی را ایجاد می‌کند تا داده‌های قابل نمایش از کنسولهای مشترک به واحدهای ضبط انتقال داده شوند تا در آنجا ذخیره شده و در صورت نیاز دوباره پخش شوند و نهایتا نیز یکی از شبکه‌ها به عنوان شبکه یدکی[4] استفاده می‌شود. پلها[5] ارتباط بین شبکه اصلی و شبکه‌هایی که به صورت حلقوی هستند را امکان‌پذیر می‌نمایند. آنها همچنین قابلیتی را فراهم می‌کنند که در صورت خرابی یکی از شبکه‌های حلقوی، شبکه یدکی جایگزین آن شده و یا اینکه مسیریابی‌های جایگزینی را ایجاد می‌کنند.

[1] Common consoles
[2] Local Communicating Network (LCN)
[3] LCN Interface Units (LIU-H)
[4] Spare
[5] Bridges

- کانال دستیابی به رادار پیشرفته یا *EDARC*[1] یک نمایش پشتیبان[2] از موقعیت هواپیما ایجاد کرده و همچنین اطلاعات بلوک داده پرواز را محدود به کنسولهای نمایش بین‌راهی می‌کند. *EDARC* زمانی که داده‌های نمایشی فراهم شده به وسیله میزبان از بین می‌رود، مورد استفاده قرار می‌گیرد. این ابزار داده‌های خام پردازش نشده ضروری رادار و واسطهای لازم برای پردازنده *ESI*[3] را فراهم می‌آورد.
- شبکه ارتباطی پشتیبان[4] یک شبکه اترنت با استفاده از قرارداد *TCP/IP* است که برای وظایف دیگر سیستم (علاوه بر واسط *EDARC*) مورد استفاده قرار می‌گیرد. بعلاوه به عنوان یک شبکه پشتیبان برای حالت‌های خاص که در آن شبکه ارتباط محلی با شکست مواجه می‌شود مورد استفاده قرار می‌گیرد.
- هر دوی شبکه ارتباطی محلی و شبکه ارتباطی پشتیبان به کنسول نظارت و کنترل[5] مرتبط شده‌اند. این کنسول به پرسنل پشتیبان سیستم یک دید کلی از وضعیت سیستم می‌دهد و به آنها اجازه می‌دهد که عملیات سیستم را کنترل کنند. کنسول نظارت و کنترل، یک کنسول معمولی است که شامل نرم‌افزار خاصی در جهت حمایت از عملیات مربوط به نظارت و کنترل بوده و همچین دارای توابع مدیریتی قابل دسترس سطح بالا (عمومی) است.
- زیر سیستم آزمایش و آموزش، قابلیتی را برای آزمایش سخت‌افزارها و نرم‌افزارهای جدید فراهم می‌آورد. این زیر سیستم اجازه می‌دهد که کاربران تازه کار بدون تداخل با اهداف سیستم کنترل ترافیک هوایی، آموزشهای لازم را دریافت نمایند.
- پردازنده‌های مرکزی از نوع ابر پردازنده‌ها هستند که قابلیت ضبط و پخش مجدد داده‌ها را برای سیستم *ISSS* فراهم می‌کنند.

۶-۴-۲- دید تجزیه ماژول

عناصر ماژول نرم‌افزار عملیاتی *ISSS*، آیتم‌های پیکربندی نرم‌افزار کامپیوتر[6] نامیده می‌شوند (در استاندارد توسعه نرم‌افزارهای دولتی تعریف شده است و استفاده از آن به وسیله مشتریان الزامی شده است). *CSCIs*

[1] Enhanced Direct Access Radar Channel (EDARC)
[2] Backup display
[3] External Systems Interface
[4] Backup communicating Network (BCN)
[5] Monitor-and-Control (M&C)
[6] Computer Software configuration Items (CSCIs)

کاملا به تخصیص کار مربوطه می‌شوند و تیم‌های بزرگ مسئول طراحی، ساخت و آزمایش آنها هستند. معمولا بعضی عناوین منسجم در ارتباط با هر *CSCIs* وجود دارد مانند بسته‌ها، فرآیندها و غیره. در سیستم *ISSS* پنج *CSCIs* وجود دارد که عبارتند از:

۱. *مدیریت نمایش*[۱]، مسئول تولید و نگهداری نمایش‌ها بر روی کنسولهای مشترک
۲. *سرویس‌های سیستمی مشترک*[۲]، مسئول فراهم کردن برنامه‌های کاربردی سودمند در نرم‌افزار کنترل ترافیک هوایی
۳. *ضبط، تحلیل و پخش مجدد*[۳]، مسئول بدست آوردن نشست‌های[۴] سیستم کنترل ترافیک هوایی برای تحلیل‌های بعدی
۴. *اصلاح سیستم هوایی ملی*[۵]، فراهم کردن یک نسخه تغییریافته از نرم‌افزار برای مقیم شدن بر روی یک میزبان
۵. سیستم *عامل IBM AIX*، فراهم کردن محیط سیستم عامل برای نرم‌افزارهای عملیاتی

CSCIs سیستم *ISSS*، واحدهای مربوط به مستندات قابل تحویل نرم‌افزار و همچنین خود نرم‌افزار را شکل می‌دهند. در واقع آنها در زمانبندی فرسنگ شمارهای پروژه ذکر می‌شوند و هر یک مسئول بخشی از عملکرد سیستم *ISSS* هستند.

دید تجزیه ماژول سیستم *ISSS* چندین تاکتیک قابلیت اصلاح را معرفی می‌کند (در فصل پنجم تشریح شدند). "انسجام معنایی" تاکتیکی برای تخصیص مسئولیت‌های خوش تعریف و بدون هم‌پوشانی برای هر *CSCIs* است. سرویسهای مشترک سیستم منعکس کننده تاکتیک "تجرید سرویسهای مشترک" است. ضبط، تحلیل و پخش مجدد منعکس کننده تاکتیک "ضبط و پخش مجدد" برای قابلیت آزمایش است. منابع هر *CSCI* از طریق واسطهای نرم‌افزاری خوش تعریف که بازتاب کننده "پیش‌بینی تغییرات مورد انتظار"، "عمومی‌سازی ماژول" و "نگهداری ماندگاری واسط[۶]" است، قابل دسترس هستند.

[1] Dislay management
[2] Common system services
[3] Recording, analysis, and playback
[4] Sessions
[5] National airspace system modification
[6] Maintaining interface stability

۶-۴-۳- دید فرآیند

پایه (مبنا) همزمانی در سیستم *ISSS* عناصری به نام کاربردها[1] هستند. یک کاربرد مسئول یک فرآیند بوده (با توجه به مفهوم فرآیندهای ترتیبی همکار دایجسترا) و در مرکز روشی است که طراحان سیستم *ISSS* برای اعمال تحمل‌پذیری خطا استفاده کرده‌اند. همچنین کاربرد به صورت یک واحد *"main"* در *Ada* پیاده‌سازی شده (یعنی یک فرآیند قابل زمانبندی به وسیله سیستم عامل) و بخشی از یک *CSCI* را شکل می‌دهد. نحوه ارتباط کاربردها با یکدیگر نیز مبتنی بر ارسال پیام است.

سیستم *ISSS* طوری ساخته شده است تا بتواند بر روی تعداد زیادی از پردازنده‌ها اجرا شود. در این سیستم، پردازنده‌ها به صورت منطقی با یکدیگر ترکیب شده‌اند تا یک گروه پردازشگر را تشکیل دهند و هدف از آن این است که میزبان، کپی‌هایی از یک یا چند کاربرد ایجاد کند (این مفهوم برای تحمل‌پذیری خطا و قابلیت دسترسی بسیار مهم است). همواره یکی از کپی‌ها اصلی و مابقی فرعی هستند، لذا کپی‌های مختلف از یک کاربرد به یک فضای آدرس اصلی[2] یا فضای آدرس جانشین[3] اشاره دارند.

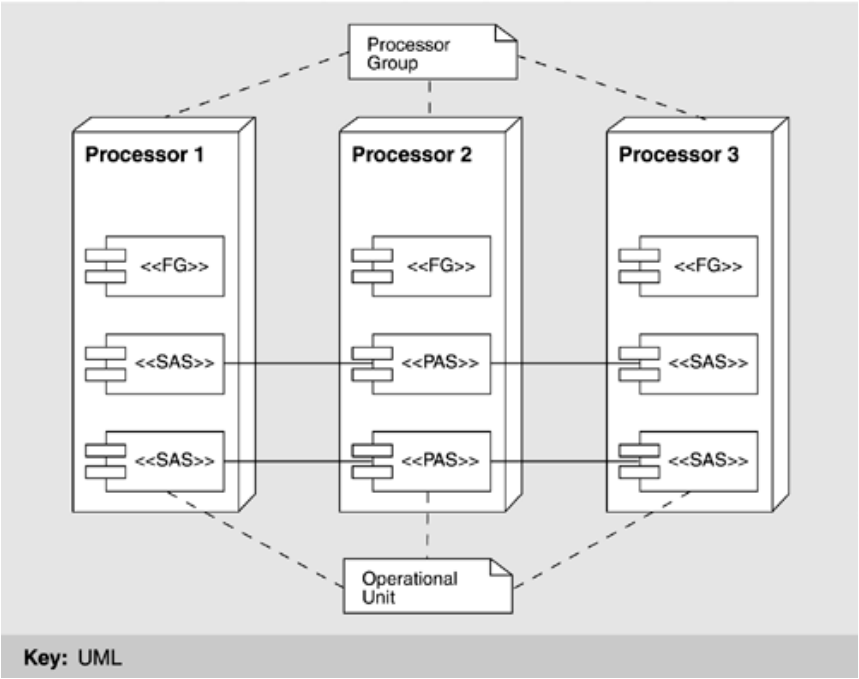
مجموعه‌ای از یک فضای آدرس اصلی بعلاوه فضای آدرس جانشین مربوط به آن یک واحد عملیاتی نامیده می‌شود. یک واحد عملیاتی کاملا در داخل پردازنده‌های یک گروه پردازنده است (یک گروه می‌تواند متشکل از چهار پردازنده باشد). بخشهایی از سیستم *ISSS* هم که مبتنی بر روش تحمل‌پذیری خطا ساخته نمی‌شوند به سادگی و مستقلا در پردازنده‌های مجزا اجرا می‌شوند که اصطلاحا به آنها گروهای عملکردی[4] گفته می‌شود. این گروه‌های عملکردی در صورت نیاز بر روی هر پردازنده قرار گرفته و حالت خود را نیز حفظ می‌کنند. بنابراین با توجه به مطالب ذکر شده یک کاربرد می‌تواند واحد عملیاتی یا گروه عملکردی در نظر گرفته شود. برای پشتیبان‌گیری از عملکرد یک کاربرد دو حالت وجود دارد یکی پشتیبان‌گیری از یک یا چند کپی فرعی و دیگری نگهداری وضعیت و داده کپی اصلی است. واحدهای عملیاتی، یک چنین ساختاری را در ماهیت خود دارند ولی گروه‌های عملکردی فاقد آن هستند.

[1] Applications
[2] Primary Address Space (PAS)
[3] Standby Address Space (SAS)
[4] Functional groups

یک کاربرد به صورت یک واحد عملیاتی پیاده‌سازی می‌شود البته اگر نیازمندیهای قابل دسترس آن، این امر را دیکته کنند، در غیر این صورت به شکل یک گروه عملکردی پیاده‌سازی می‌شود.

کاربردها در یک روش مبتنی بر سرویس‌دهنده و سرویس‌گیرنده با یکدیگر ارتباط برقرار می‌کنند. نحوه عملکرد بدین صورت است که تراکنش سرویس‌گیرنده، یک پیام درخواست سرویس به سرویس‌دهنده ارسال می‌کند و سرویس‌دهنده نیز به آن پاسخ می‌دهد. در داخل یک واحد عملیاتی نیز، فضای آدرس اصلی پیام‌های تغییر وضعیت خود را به هر یک از آدرسهای فضای جانشین ارسال می‌کند. آنها نیز با توجه به زمان دریافت یا دیگر علائم تشخیص می‌دهند که فضای آدرس اصلی دچار شکست شده یا نه؟ و در صورت نیاز یکی از آنها تبدیل به فضای آدرس اصلی می‌شود.

شکل ۶-۳ به طور خلاصه نمایش می‌دهد که چگونه فضاهای آدرس اصلی و جانشین در یک کاربرد با یکدیگر هماهنگ می‌شوند تا اینکه یک قابلیت پشتیبان فراهم کرده و چگونگی ارتباط خود را برای گروه‌های پردازشگر معلوم کنند.



شکل ۶-۳: گروه‌های عملکردی، واحدهای عملیاتی، گروه‌های پرازشگر و فضاهای آدرس اصلی و جانشین

زمانی که یک گروه عملکردی یک پیام را دریافت می‌کند، فقط نیاز دارد که وضعیت خود را به صورت مناسب به روز کرده و پاسخی ارسال کند. در مورد واحدهای عملیاتی نیز معمولا فضای آدرس اصلی یک واحد

عملیاتی به نیابت از کل واحد عملیاتی پیام را دریافت کرده و به آن پاسخ می‌دهد. سپس آن واحد عملیاتی باید هم حالت خود و هم حالت فضای آدرس جانشین را به روز کند که این کار نیازمند ارسال پیامهای اضافی است. گاها اتفاق می‌افتد که یک شکست در فضای آدرس اصلی رخ می‌دهد که در آن صورت یک تعویض صورت می‌پذیرد. نحوه انجام یک تعویض به صورت زیر است:

۱. یک فضای آدرس جانشین به یک فضای آدرس اصلی ترفیع پیدا می‌کند.
۲. فضای آدرس اصلی جدید با سرویس‌گیرنده‌های آن واحد عملیاتی ارتباط برقرار می‌کند (یعنی مشخص می‌کند که از این به بعد سرویس‌دهنده جدید من هستم) و سپس به درخواست‌های رسیده پاسخ می‌دهد.
۳. یک فضای آدرس جانشین ایجاد می‌شود تا اینکه جایگزین فضای آدرس اصلی قبلی شود.
۴. فضای آدرس جانشین جدید خودش را به فضای آدرس اصلی جدید معرفی می‌کند تا اینکه پیامهای خود را به آن ارسال کرده و همیشه به روز باشد.

اگر یک شکست در داخل فضای آدرس مجازی رخ دهد در آن صورت یک فضای جدید در داخل پردازنده دیگر ایجاد شده و شروع به فعالیت می‌کند. در واقع ابتدا با فضای آدرس اصلی ارتباط برقرار نموده و هماهنگ می‌شود و سپس داده‌ها را دریافت می‌کند.

برای اضافه کردن یک واحد عملیاتی جدید نیز گام‌های زیر به کار گرفته می‌شوند:

- شناسایی داده‌های ورودی ضروری و محلی که مقیم هستند.
- شناسایی واحدهای عملیاتی که نیازمند داده خروجی از واحد عملیاتی جدید هستند.
- وارد کردن الگوهای ارتباطی واحد عملیاتی جدید در یک گراف غیرچرخشی به صورتی که گراف باز هم غیر چرخشی باقی بماند تا در آینده بن بست اتفاق نیفتد.
- طراحی پیامها برای دستیابی به جریان داده‌های مورد نیاز
- شناسایی داده‌های حالت درونی[1] که باید برای گرفتن نقطه وارسی استفاده شوند و همچنین داده‌های حالتی که باید در به روزرسانی بین فضای آدرس اصلی و فضای آدرس جانشین در نظر گرفته شوند.
- تقسیم کردن داده‌های حالت به پیامهایی که خیلی خوب بر روی شبکه قرار بگیرد.

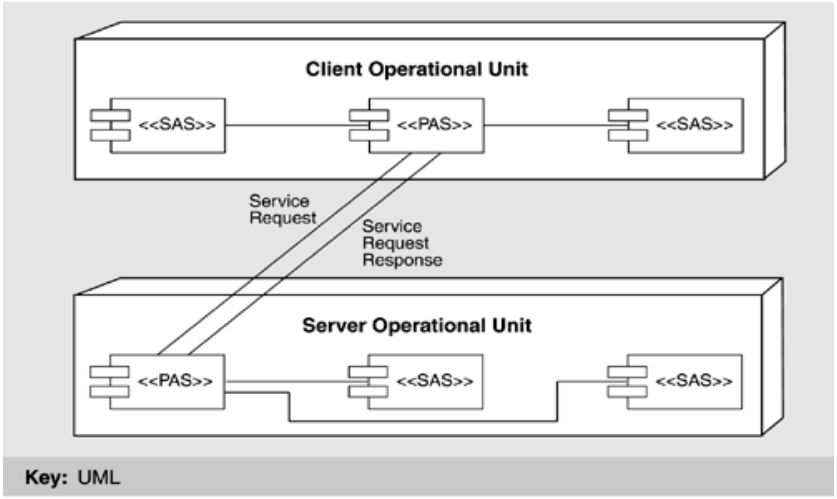
[1] Internal state data

- تعریف انواع پیامهای ضروری
- برنامه‌ای برای تعویض در حالت شکست (طرحهای به روز رسانی برای تضمین کامل بودن وضعیت)
- تضمین سازگار بودن داده‌ها در حالت تعویض
- تضمین اینکه گامهای پردازشی منفرد در زمان کمتر از یک "تپش قلب" کامل می‌شوند.
- طرح‌ریزی قرارداد قفل‌گذاری داده و به اشتراک گذاری داده با دیگر واحدهای عملیاتی

با توجه به مطالب ذکر شده پیرامون دید فرآیند، تاکیتک‌های مورد استفاده شده برای این دید عبارتند از: "همگام‌سازی مجدد حالت"، "سایه"، "افزونگی فعال" و "خارج کردن از سرویس".

۶-۴-۴- دید سرویس‌گیرنده و سرویس‌دهنده

به دلیل اینکه کاربردها در دید فرآیند با یک روش مبتنی بر سرویس‌گیرنده و سرویس‌دهنده با یکدیگر ارتباط برقرار می‌کنند، لازم است که یک دید سرویس‌گیرنده و سرویس‌دهنده از سیستم *ISSS* نیز ارائه شود. این دید در شکل ۶-۴ به تصویر کشیده شده است.



شکل ۶-۴: کاربردها به صورت سرویس‌گیرنده‌ها و سرویس‌دهنده‌ها

در این سیستم سرویس‌دهنده‌ها و سرویس‌گیرنده‌ها خیلی دقیق طراحی شده‌اند تا اینکه واسطها سازگاری داشته باشند. این قابلیت به وسیله استفاده از قرارداد انتقال ساده پیام برای تعامل فراهم شده است. با توجه به مطالب ذکر شده تاکتیک‌های قابلیت اصلاح استفاده شده عبارتند از: " نگهداری ماندگاری واسط"، "جایگزینی مولفه" و " الصاق به قراردادهای تعریف شده".

۶-۴-۵- دید کد[1]

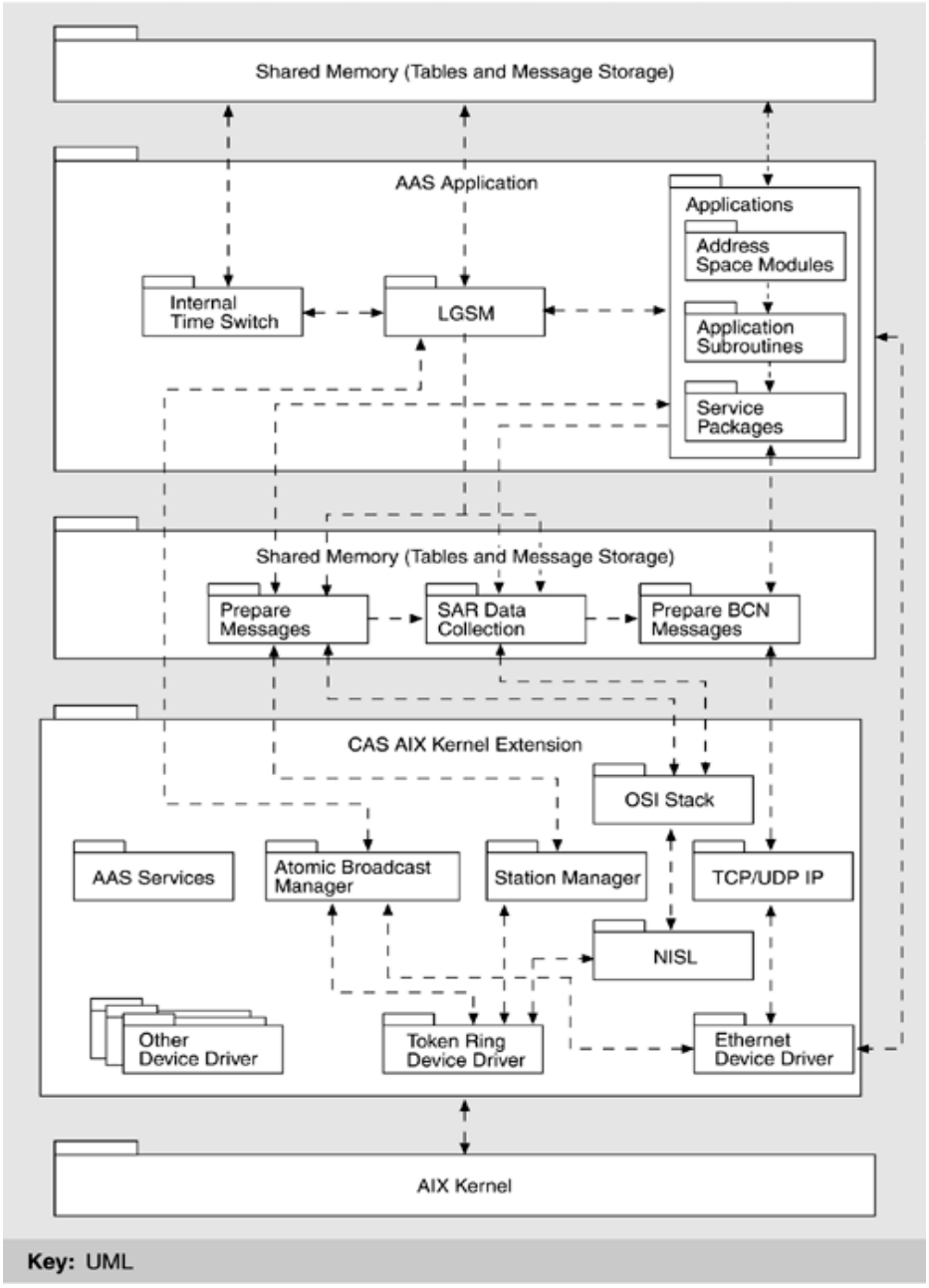
یکی از دیدهایی که در فصل دوم بررسی نشده ولی اغلب اوقات در معماری سیستم‌های بزرگ ظاهر می‌شود، دید کد است. دید کد نشان می‌دهد چگونه عملکرد سیستم به واحدهای کد نگاشت می‌شود. در *ISSS* یک برنامه *Ada* از یک یا چند فایل منبع تشکیل می‌شود و معمولا متشکل از تعدادی زیر برنامه است که گاها بعضی از آنها در داخل بسته‌های قابل کامپایل به صورت مجزا قرار دارند. بنابراین *ISSS* از یک چنین برنامه‌هایی تشکیل شده است که بسیاری از آنها در یک روش سرویس‌گیرنده و سرویس‌دهنده با یکدیگر ارتباط برقرار می‌کنند.

یک برنامه *Ada* می‌تواند متشکل از یک یا چند وظیفه باشد. هر وظیفه در *Ada* موجودیتی است که توانایی اجرا به صورت همروند با دیگر وظایف را دارد. در واقع وظایف، دید کد همروندی فرآیندهایی هستند که در دید فرآیند تشریح شده‌اند (وظایف *Ada* به وسیله سیستم زمان اجرای *Ada* مدیریت می‌شود). کاربردها یعنی واحدهای عملیاتی و گروه‌های عملکردی به بسته‌های *Ada* تجزیه می‌شوند که بعضی از آنها دربرگیرنده فقط تعاریف نوع و بعضی‌ها هم کاربردهای قابل استفاده مجدد هستند (بسته‌بندی کردن یک فعالیت طراحی است که پنهان‌سازی اطلاعات و تجرید را دربر گرفته و به وسیله طراح ارشد واحد عملیاتی انجام می‌شود).

۶-۴-۶- دید لایه‌بندی

لایه زیربنایی عملیات برنامه‌های کاربردی سیستم کنترل ترافیک هوایی بر روی *ISSS*، یک سیستم عامل تجاری یونیکس به نام *AIX* است. البته یونیکس کلیه سرویسهای مورد نیاز برای حمایت از یک سیستم تحمل‌پذیر خطای توزیع شده (مانند *ISSS*) را فراهم نمی‌کند. بنابراین سرویسهای سیستمی جدیدی به آن افزوده شده است. شکل ۶-۵ یک مجموعه از لایه‌های کلی محیط نرم‌افزار در یک پردازنده معمولی *ISSS* را نمایش می‌دهد.

[1] Code view



شکل ۶-۵: لایه‌های معماری نرم‌افزار *ISSS*

دو ردیف بالایی بر روی *AIX*، گسترش‌هایی به *AIX* را نشان می‌دهند که در داخل فضای آدرس هسته *AIX* اجرا می‌شوند. به دلیل نیازمندیهای مربوط به کارایی و قابلیت سازگاری با سیستم عامل *AIX*، این گسترش‌ها عموما برنامه‌های کوچک نوشته شده به زبان *C* هستند و از آنجایی که آنها در داخل فضای هسته اجرا می‌شوند

شکست در این برنامه‌ها به شدت می‌تواند موجب آسیب‌دیدگی *AIX* شود، لذا این برنامه‌ها باید نسبتا کوچک و کاملا عاری از خطا باشند (برنامه‌های قابل اطمینان منعکس کننده تاکتیک "نمایش محدود" هستند).

مدیر انتشار تجزیه‌ناپذیر[1] یک نقش کلیدی در ارتباطات بین ماژولهای مدیریت قابلیت دسترسی[2] محلی در درون یک "مجموعه بخش"[3] بازی می‌کند تا اینکه قابلیت دسترسی مناسب عملکردها را مدیریت کند. مدیر ایستگاه، سرویسهای *datagram* را بر روی شبکه ارتباطی محلی فراهم می‌کند و به عنوان نماینده سرویسهای مدیریت شبکه ارتباطی محلی، ارائه خدمات می‌دهد. زیرلایه‌های کارت شبکه نیز توابع مشابه‌ای برای انتقال پیامهای نقطه به نقطه فراهم می‌آوردند و همچنین اطلاعات شبکه‌ای خود را با مدیر ایستگاه به اشتراک می‌گذارد. دو لایه بعدی گسترشهای سیستم عامل را نشان می‌دهند که بیرون از فضای آدرس هسته اجرا می‌شوند. بنابراین در صورت شکست نمی‌توانند مستقیما به *AIX* آسیب برسانند. این برنامه‌ها معمولا در زبان *Ada* نوشته می‌شوند.

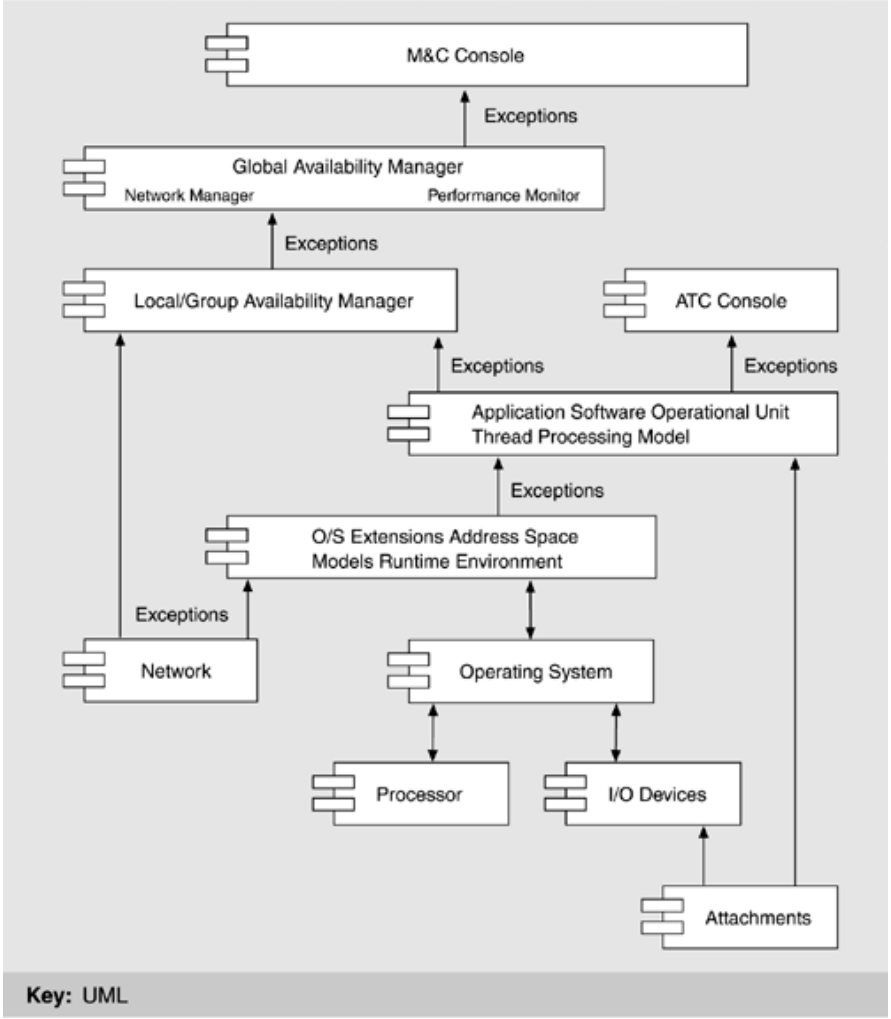
آماده کننده پیام‌ها[4]، پیام‌های شبکه ارتباط محلی برای برنامه‌های کاربردی را مدیریت می‌کند. آماده کننده پیام‌های شبکه ارتباطی پشتیبان[5] نیز یک چنین عملکردی را در جهت ارسال کردن پیام‌ها بر روی شبکه ارتباط پشتیبان انجام می‌دهند. یکی از عملکردهای این برنامه‌ها تشخیص این موضوع است که کدام یک از کپی‌های چندگانه از یک برنامه کاربرد در یک "مجموعه بخش"، اصلی است تا اینکه پیامها توسط آن دریافت شوند. مدیریت قابلیت دسترسی محلی نیز اطلاعات کنترلی مورد نیاز را برای انجام این شناسائی فراهم می‌کند.

در بالاترین لایه نیز کاربردها قرار دارند. مدیر قابلیت دسترسی محلی و برنامه‌های همزمان‌سازی زمان داخلی[6]، سرویس‌های سیستمی سطح کاربرد هستند. مدیر قابلیت دسترسی محلی مسئول مدیریت بارگذاری اولیه، خاتمه یافتن و قابلیت دسترسی برنامه‌های کاربردی است. مدیر با هر فضای آدرس در روی پردازنده خودش ارتباط برقرار می‌کند تا اینکه عملیات آن را کنترل کرده و وضعیت آن را بررسی کند. همچنین با مدیریت قابلیت دسترسی بر روی پردازنده‌های دیگر که در داخل "مجموعه بخش" خودشان قرار دارند ارتباط

[1] Atomic Broadcast Manager (ABM)
[2] Local Availability Manager
[3] Sector suite
[4] Prepare messages
[5] Prepare BCN Messages
[6] Internal Time Synchronization

برقرار می‌کند تا قابلیت دسترسی مجموعه‌ای از عملکردها (شامل تعویض شدن از یک برنامه کاربردی اصلی به یک کپی پشتیبان در زمان مناسب) را مدیریت می‌کند. مدیریت قابلیت دسترسی محلی با کاربرد مربوط به مدیریت قابلیت دسترسی سراسری[1] مقیم بر روی کنسولهای نظارت و کنترل ارتباط برقرار می‌کند تا وضعیت را گزارش و دستورات کنترلی از آنها دریافت کند. برنامه همزمان‌سازی زمان داخلی، زمان پردازنده را با زمان پردازنده‌های *ISSS* که برای عملکرد مناسب توابع مدیریت قابل دسترسی حیاتی هستند، همزمان می‌کند.

شکل ۶-۶ یک دید مولفه و رابط برای تحمل‌پذیری خطا در سیستم *ISSS* را نشان می‌دهد.



شکل ۶-۶: دید مولفه و اتصال برای تحمل‌پذیری خطا

[1] Global Availability Management

۶-۴-۷- دید تحمل‌پذیری خطا

نیازمندیهای قابلیت دسترسی بالا برای *ISSS* منجر به این می‌شود که تحمل‌پذیری خطا تبدیل به یک نقش بسیار مهمی در طراحی سیستم شود. زیرا راه‌اندازی مجدد سیستم در صورت شکست به هیچ عنوان قابل قبول نیست بنابراین یکی از راه‌های مناسب تعویض مولفه‌ها در زمانی است که سیستم در حالت انتظار برای عملیات[1] است. با توجه به این موضوع، به همان میزانی که طراحی سیستم پیشرفت می‌کند و تمزکز بر روی این خصوصیت کیفی افزایش پیدا می‌کند، ساختار معماری جدیدی به نام سلسله مراتب تحمل‌پذیری خطا ایجاد می‌شود (شکل ۶-۶). این ساختار نشان می‌دهد که چگونه نقص‌ها کشف، قرنطینه و نهایتا بازیابی می‌شوند. از آنجایی که طرح فضای آدرس مجازی/اصلی خطاهای درون یک کاربرد را شناسائی و بازیابی می‌کند، سلسله مراتب تحمل‌پذیری خطا نیز طوری طراحی شده است تا خطاهایی که از تعامل چندین کاربرد ناشی می‌شود را شناسایی و بازیابی کنند. سلسله مراتب تحمل‌پذیری خطا سطوح مختلفی را برای تشخیص نقص و بازیابی آن فراهم می‌کند. هر سطح به طور غیرهمزمان عملیات زیر را انجام می‌دهد

- خطاها را در خودش، همکاران و سطوح پائین‌تر شناسایی می‌کند.
- استثنائات در سطوح پائین را مدیریت می‌کند.
- تشخیص، بازیابی، گزارش و یا صدور استثنائات را انجام می‌دهد.

هر سطح در سلسله مراتب تحمل‌پذیری خطا هدفش افزایش قابلیت دسترسی به کمک لایه‌های پائین‌تر است. سطوح مربوط به تحمل‌پذیری خطا به شرح زیر هستند:

- فیزیکی (شبکه، پردازنده، دستگاه‌های ورودی و خروجی)
- سیستم عامل
- محیط زمان اجرا
- کاربرد
- قابلیت دسترسی محلی
- قابلیت دسترسی سراسری

[1] Standby

- کنترل و نظارت سیستم

تشخیص و قرنطینه کردن نقص‌ها در سیستم در هر سطح از سلسله مراتب انجام می‌شود. تشخیص نقص به کمک آزمایشهای تعبیه شده درونی[1]، اتمام زمان رویداد[2]، آزمایش شبکه، قرارداد عضویت گروه[3] و عکس العمل اشخاص به هشدارها و شاخص‌ها[4]، انجام می‌شود. بازیابی نقص نیز در هر سطح از سلسله مراتب انجام می‌شود و می‌تواند خودکار یا دستی باشد. برای مدیرهای قابلیت دسترسی سراسری و محلی، روشهای بازیابی مبتنی بر جدول خاصی هستند.

در فضای آدرس اصلی نیز چهار روش برای بازیابی وجود دارد که نوع روش به کار گرفته شده به وضعیت عملیاتی جاری و تصمیم مدیر قابلیت دسترسی محلی بستگی دارد (مدیر با استفاده از جداول تصمیم‌گیری می‌کند). این روشها عبارتند از:

- در حالت تعویض، فضای آدرس جانشین بلافاصله با فضای آدرس اصلی جایگزین می‌شود.
- یک راه‌اندازی گرم، از نقطه وارسی داده استفاده می‌کند
- یک راه‌اندازی سرد از داده‌های پیش فرض استفاده می‌کند که در آن صورت، حالت سیستم نیز از بین می‌رود.
- یک عملکرد سریع[5] برای گذر به یک داده منطقی یا سازگار استفاده می‌شود.

افزونگی به وسیله سخت‌افزارهای شبکه (پلها، شبکه ارتباط محلی و شبکه ارتباط پشتیبان)، پردازنده (چهار پردازنده برای یک گروه پردازنده و افزونگی ضبط داده‌ها) و نرم‌افزار (چندین فضای آدرس برای یک واحد عملیاتی) فراهم می‌شود. علاوه بر تاکتیک‌های قابلیت استفاده که در دید فرآیند مطرح شدند دید تحمل‌پذیری خطا از تاکتیک‌های "صدا/انعکاس" و "تپش قلب" برای تشخیص شکست، "استثناء" برای انتقال خطاها به یک محل مناسب برای اصلاح و "یدکی" برای بازیابی استفاده می‌کند.

[1] Built-in tests
[2] Event time-outs
[3] Group membership protocol
[4] Human reaction to alarms and indicators
[5] Cutover

۶-۴-۸- ارتباط بین دیدها

اگرچه دیدها باعث می‌شوند که درک یک سیستم به خوبی صورت بپذیرد ولی درک عمیق سیستم اغلب از بررسی ارتباط بین دیدها، مخصوصا نگاشت بین آنها حاصل می‌شود. در *ISSS*، *CSCIs*ها عناصری در دید تجریه ماژول هستند که از کاربردها تشکیل شده‌اند. کاربردها به نوبه خود در دید فرآیند و دید سرویس‌گیرنده و سرویس‌دهنده ظاهر می‌شوند. کاربردها با برنامه‌ها و بسته‌های *Ada* پیاده‌سازی شده و در دید کد نمایش داده می‌شوند که آنها نیز به نوبه خود به نخ‌ها تبدیل می‌شوند و نخ‌ها نیز عناصر دید همزمانی هستند. دید لایه‌بندی نیز عملکردهای تخصیص داده شده به ماژولها در دید تجزیه را تشریح می‌کند و مشخص می‌کند که هر ماژول اجازه انجام چه کاری را دارد. نهایتا یک دید خاص متمرکز بر روی دستیابی به یک خصوصیت کیفی زمان اجرا به نام دید تحمل‌پذیری خطا تعریف می‌شود که از عناصر دیدهای فرآیند، لایه‌بندی و ماژول استفاده می‌کند.

۶-۴-۹- سازگاری داده‌ها

ISSS از تاکتیک "فایلهای پیکربندی" مربوط به قابلیت تغییر استفاده گسترده‌ای می‌کند و آن را سازگاری داده می‌نامد. سازگاری داده یک راه حل خوب برای تغییر سیستم در مورد نیازمندیهای خاص محل، اولویت‌های خاص مرکز و کاربر، تغییرات پیکربندی، تغییرات نیازمندی و دیگر جنبه‌های نرم‌افزار که انتظار می‌رود با گذشت زمان و در هنگام استقرار تغییر پیدا کنند، ارائه می‌نماید. در اصل نرم‌افزار طوری طراحی می‌شود که پارامترهای ورودی خود را خوانده و برحسب مقادیر آن پارامترها، رفتارهای خاصی را انجام دهد. برای مثال، یک تغییر نیازمندی به منظور جدا کردن داده موجود در یک پنجره سیستم کنترل ترافیک هوایی به دو پنجره می‌تواند به وسیله تغییر سازگاری داده و چندین خط از کد انجام شود.

نکات منفی مربوط به سازگاری داده این است که مکانیزم پیچیده‌ای را برای پشتیبانی کننده‌ها ارائه می‌کند. برای مثال، اضافه کردن یک دستور جدید به سیستم عملا یک کار ساده‌ای است اما پیاده‌سازی این قابلیت انعطاف‌پذیر در حقیقت خود فرآیند پیچیده‌ای است.

از طرفی تعاملات پیچیده می‌تواند بین بخش‌های مختلف سازگاری داده‌ها اتفاق بیفتد که این امر باعث تحت تاثیر قرار گرفتن صحت و درستی می‌شود (هیچ گونه مکانیزم خودکار یا نیمه خودکاری به منظور جلوگیری از این نوع تاثیرات ناسازگار کننده وجود ندارد).

نهایتا اینکه سازگاری داده به شدت فضای حالت را که در عملیات نرم‌افزار باید به صورت صحیح انجام شود افزایش داده و این امر منجر می‌شود که سیستم به منظور صحت و درستی، بیشتر مورد آزمایش قرار بگیرد.

۶-۴-۱۰- قالب کد برای *ISSS*

همان طور که قبلا تشریح شد فضای آدرس اصلی-ثانویه (جانشین) به منظور دستیابی به تحمل‌پذیری خطا معرفی شده‌اند (کپی‌هایی از نرم‌افزار که در پردازنده‌های مختلف ذخیره شده‌اند). نحوه عملکرد بدین صورت است که کپی اصلی همیشه در حال اجرا است و در بازه‌های زمانی خاص اطلاعاتی در مورد وضعیت خود به کپی‌های دیگر ارسال می‌کند تا آنها کاملا به روز شوند. نحوه پیاده‌سازی این کپی‌ها به گونه‌ای است که آنها کاملا از یک کد منبع به دست می‌آیند. به منظور پیاده‌سازی این حالت شرکت توسعه‌دهنده یک قالب کد استانداردی برای هر کابرد تعریف کرد که این قالب در شکل ۶-۷ به تصویر کشیده شده است.

```
terminate:= false
initialize application/application protocols

ask for current state (image request)
Loop
  Get_event
  Case Event_Type is

  -- "normal" (non-fault-tolerant-related) requests to perform actions;
  -- only happens if this unit is the current primary address space
  when X=> Process X
        Send state data updates to other address spaces
  when Y=>Process Y
        Send state data updates to other address spaces
  ...
  when Terminate_Directive => clean up resources; terminate := true

  when State_Data_Update => apply to state data
  -- will only happen if this unit is a secondary address space, receiving
  -- the update from the primary after it has completed a "normal" action

  -- sending, receiving state data
  when Image_Request => send current state data to new address space
  when State_Data_Image => Initialize state data

  when Switch_Directive => notify service packages of change in rank

  -- these are requests that come in after a PAS/SAS switchover; they
  -- report services that they had requested from the old (failed) PAS
  -- which this unit (now the PAS) must complete. A,B, etc. are the names
  -- of the clients.
  when Recon_from_A=>reconstitute A
  when Recon_from_B=>reconstitute B
  ...
```

```
  when others=>log error
  end case
exit when terminate
end loop
```

شکل ۶-۷: قالب ساختار کد برای کاربردهای تحمل‌پذیری خطا در سیستم *ISSS*

ساختار قالب کد دربرگیرنده یک حلقه است که منطبق بر آن به کلیه رویدادهای وارده، سرویس داده می‌شود. اگر رویداد باعث شود که کاربرد یک عمل نرمال را انجام دهد در آن صورت کنش مناسب انجام می‌گیرد. بعد از انجام کنش مناسب، سرویس‌دهی مربوط به فعالیت به روز رسانی بخش ثانویه انجام می‌شود.

رویدادهای دیگر نیز مربوط به انتقال حالت یا به روزرسانی داده‌ها هستند. آخرین مجموعه از رویدادها نیز این است که فضای آدرس اصلی تغییر پیدا کرده بنابراین فضای آدرس جدید از سرویس‌گیرنده می‌خواهد که درخواست‌های بدون پاسخ خود را مجددا مطرح کرده تا بتواند به آنها پاسخ دهد.

قالب کد یک استدلال معمارانه است زیرا باعث می‌شود اضافه کردن کاربردهای جدید با کمترین تاثیرگذاری بر کار اصلی مکانیزم تحمل‌پذیری خطا انجام شود. توسعه‌دهندگان و پشتیبان‌ها نیاز ندارند که اطلاعات کامل در مورد مکانیزم مدیریت پیام‌ها داشته باشند (یک دید تجریدی کافی است) و یا اینکه نگران آن باشند که آیا کاربردشان قابلیت تحمل‌پذیری خطا را دارد یا نه؟ زیرا این عمل در سطح بالاتر کنترل شده است.

قالب کد باعث بهبود تاکتیک "تجرید سرویس‌های عمومی" می‌شود، بدین صورت که بخش‌هایی از هر کاربرد که مشترک است در قالب قرار داده می‌شوند. این تاکتیک به چندین تاکتیک دیگر مربوط است که عبارتند از:

- "پیش‌بینی تغییرات مورد انتظار"
- "انسجام معنایی"
- عمومی‌سازی ماژول"
- نگهداری ماندگاری واسط"
- "الصاق به قراردادهای تعریف شده"

جدول ۶-۱ تاکتیک‌هایی که معماری نرم‌افزاری سیستم *ISSS* در جهت دستیابی به اهداف کیفی خود از آنها استفاده کرده را نمایش می‌دهد.

جدول ۶-۱: چگونگی دستیابی سیستم *ISSS* به خصوصیات کیفی مورد نیاز

هدف	چگونگی دستیابی	تاکتیک‌های استفاده شده
قابلیت دسترسی بالا	افزونگی سخت‌افزاری (پردازنده و شبکه)، افزونگی نرم‌افزاری (تشخیص و بازیابی نقص به صورت لایه‌ای)	همزمان‌سازی مجدد حالت، سایه، افزونگی فعال، خارج کردن از سرویس، نمایش محدود، صدا/انعکاس، تپش قلب، استثناء و یدک
کارایی بالا	چندپردازنده‌های توزیع شده، تحلیل قابلیت زمانبندی رو به جلو و مدلسازی شبکه	معرفی همزمانی
بازبودن	پوشش و لایه‌بندی واسط	تجرید سرویس‌های مشترک، نگهداری ماندگاری واسط
قابلیت اصلاح	سازگاری داده‌ها بر اساس قالب‌ها و مبتنی بر جدول، تخصیص دقیق مسئولیت‌ها به ماژولها، محدود کردن استفاده از واسط‌های خاص	تجرید سرویسهای مشترک، انسجام معنایی، نگهداری ماندگاری واسط، پیش بینی تغییرات مورد انتظار، عمومی‌سازی ماژولها، جایگذاری مولفه‌ها، الصاق به قراردادهای تعریف شده، فایلهای پیکربندی
توانایی برای جداکردن زیرسیستمها	جداسازی مناسب دغدغه‌ها	تجرید سرویسهای مشترک
قابلیت تعامل‌پذیری	عملکرد و ارتباطات مبتنی بر پیام	الصاق به قراردادهای تعریف شده، نگهداری ماندگاری واسط

فهرست مطالب

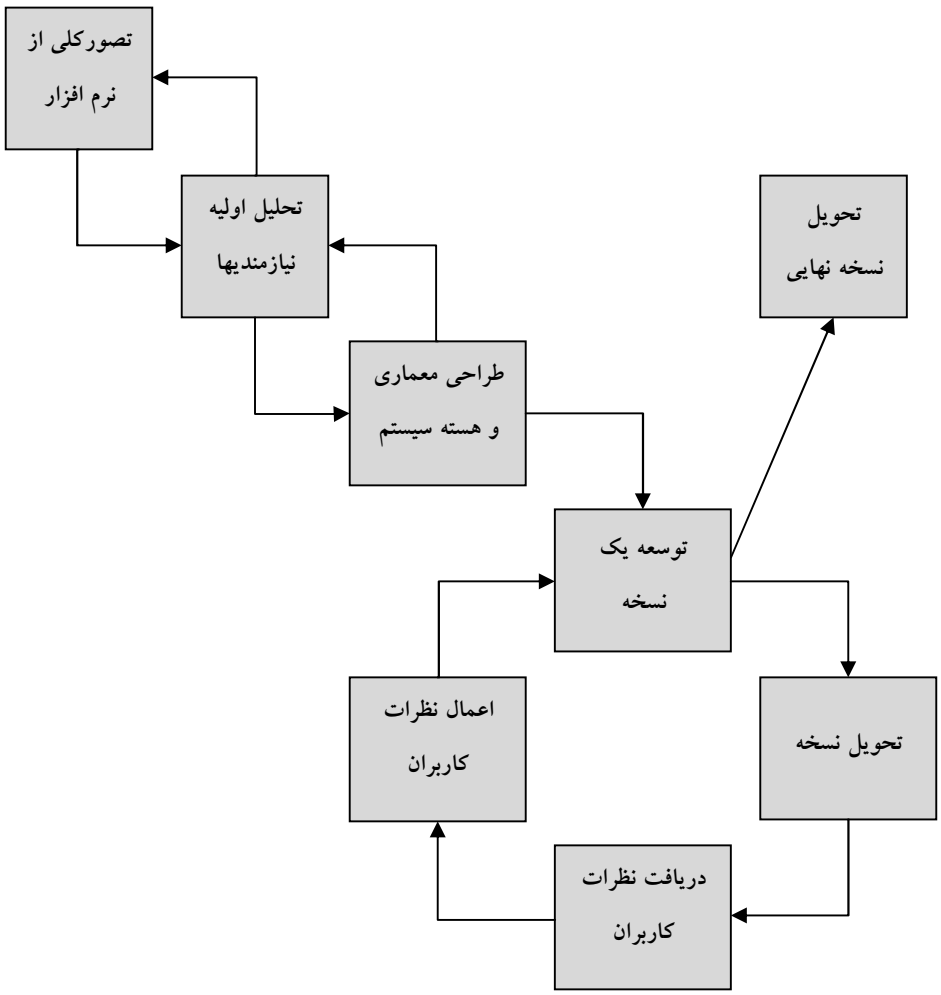
فصل هفتم

طراحی معماری

در این فصل طراحی معماری (طراحی ویژگی‌رانه) تشریح خواهد شد و مشخص می‌شود که چه زمانی معمار می‌تواند طراحی معماری را شروع کند، همچنین برای داشتن یک طراحی خوب چه گامهایی باید انجام شده و چه فاکتورهایی باید در نظر گرفته شوند. نهایتا نیز ذکر خواهد شد که چگونه طراحی معماری می‌تواند به تشکیل ساختار تیم توسعه و ایجاد یک اسکلت اولیه از سیستم کمک کند.

۷-۱- معماری در چرخه حیات نرم‌افزار

هر سازمانی که معماری نرم‌افزار را به عنوان پایه فرآیند توسعه نرم‌افزاری خود در نظر می‌گیرد باید جایگاه آن را در چرخه حیات مشخص کند. مشخص کردن جایگاه معماری به نحوه دستیابی به خصوصیات کیفی و همچنین چگونگی تعامل با مشتریان برای درک نیازمندیهای آنها بستگی دارد. مدلهای مختلفی از چرخه حیات وجود دارد اما یکی از این مدلها که جایگاه معماری را در نقطه میانی چرخه حیات تولید نرم‌افزار در نظر می‌گیرد چرخه حیات تحویل تکاملی[1] است که در شکل ۷-۱ نشان داده شده است.



شکل ۷-۱: چرخه حیات تحویل تکاملی

[1] Evolutionary delivery life cycle

هدف از مدل نشان داده شده در شکل ۷-۱ دریافت بازخورد از مشتری و اعمال آن در چندین نسخه مختلف قبل از تحویل نسخه نهایی است. این مدل امکان اضافه کردن عملکردهای جدید را در هر بار تکرار می‌دهد و همچنین اجازه می‌دهد قبل از نسخه نهایی -اگر در سیستم یک سری خصوصیات به صورت مناسب پیاده‌سازی شدند- یک نسخه محدود شده که امکان استفاده از آن وجود دارد را بتوان ارائه کرد.

۷-۱-۱- زمان شروع طراحی

مدل چرخه حیات نشان می‌دهد که طراحی معماری یک فرآیند تکراری[1] با تحلیل نیازمندیهای اولیه است و کاملا مشخص است که اگر ایده و تصور درستی در خصوص نیازمندیهای سیستم وجود نداشته باشد نمی‌توان طراحی را آغاز کرد. در واقع برای شروع طراحی باید نیازمندیهای کلی سیستم را کشف کرد.

معماری با توجه به مجموعه نیازهای وظیفه‌مندی، کیفی و حرفه شکل می‌گیرد. نیازمندیهای شکل‌دهنده معماری، پیشرانهای معماری[2] نامیده می‌شوند. به عنوان مثال پیشرانهایی که در تشکیل معماری *A-7E* در فصل سوم به کار گرفته شدند قابلیت تغییرپذیری و کارایی بودند. معماری سیستم کنترل ترافیک هوایی نیز که در فصل ششم مطرح شده است به وسیله نیازمندیهای قابلیت دسترسی شکل گرفته است. همچنین در نرم‌افزار شبیه‌ساز پرواز که در فصل هشتم ارائه می‌شود مشاهده خواهد شد که معماری بر اساس نیازمندیهای (پیشرانهای) کارایی و قابلیت تغییرپذیری تشکیل می‌شود.

تعیین پیشرانهای معماری با شناخت بالاترین اولویت اهداف تجاری صورت می‌پذیرد. بدین منظور باید اهداف تجاری به سناریوهای کیفی یا موارد کاربری تبدیل شوند (روش تحلیل مصالحه معماری[3] که در فصل یازده ارائه خواهد شد از یک درخت سودمندی برای کمک به تبدیل پیشرانهای حرفه به سناریوی کیفی استفاده می‌کند) و سپس با توجه به فهرست به وجود آمده، آنهایی انتخاب شوند که بیشترین تاثیر را بر معماری دارند. معمولا تعداد پیشرانهای معماری خیلی محدود بوده و کمتر از ده در نظر گرفته می‌شوند.

بعد از اینکه پیشرانهای معماری مشخص شدند طراحی معماری می‌تواند شروع شود. در طول طراحی معماری فرآیند تحلیل نیازمندیها تحت تاثیر سئوالات مطرح شده قرار می‌گیرد و این مراحل به صورت تکراری

[1] Iterating
[2] Architectural drivers
[3] Architectural Tradeoff Analysis Method (ATAM)

آنقدر انجام می‌شود تا اینکه بتوان به معماری مناسب دست پیدا کرد (پیکانی که در شکل ۷-۱ از بخش طراحی معماری خارج و وارد بخش تحلیل نیازمندیهای اولیه شده، نشان دهنده فرآیند تکراری بودن عملیات مربوط به طراحی است).

۷-۲- طراحی معماری

در این بخش یک روشی به نام طراحی ویژگی‌رانه[1] برای طراحی معماری معرفی می‌شود که هم نیازهای کیفی و هم نیازهای وظیفه‌مندی را برآورده می‌کند. طراحی ویژگی‌رانه مجموعه‌ای از سناریوهای کیفی را به عنوان ورودی می‌پذیرد و از طریق دانش موجود در مورد نحوه دستیابی به خصوصیات کیفی، طراحی معماری را انجام می‌دهد. از روش طراحی ویژگی‌رانه می‌توان به عنوان یک گسترش برای اغلب روشهای توسعه نرم‌افزاری استفاده کرد. به عنوان مثال، *RUP* را در نظر بگیرید این روش توسعه چندین مرحله دارد که اجرای آنها منجر به طراحی سطح بالای معماری می‌شود و بعد از طراحی سطح بالا نیز مشخص کردن جزئیات طراحی و پیاده‌سازی آغاز می‌شود. اعمال نمودن طراحی ویژگی‌رانه به آن باعث تغییر گامهای طراحی سطح بالای معماری می‌شود ولی پس از آن، فرآیند عادی تعریف شده در *RUP* دنبال می‌شود.

۷-۲-۱- طراحی ویژگی‌رانه

طراحی ویژگی‌رانه روشی برای تعریف معماری نرم‌افزار است. مبنای این روش، فرآیند تجزیه‌ای[2] بر روی خصوصیات کیفی است که نرم‌افزار باید آنها را برآورده کند. در واقع طراحی ویژگی‌رانه، یک فرآیند تجزیه‌ای بازگشتی است که در هر مرحله تاکتیک‌ها و الگوهای معماری را برای برآورده کردن مجموعه‌ای از سناریوهای کیفی انتخاب می‌کند و سپس وظایف را به انواع ماژول‌هایی که به وسیله الگوها مشخص شده‌اند، تخصیص می‌دهد. طراحی ویژگی‌رانه در چرخه حیات بعد از تحلیل نیازمندیها قرار دارد و زمانی می‌تواند آغاز شود که پیشرانهای معماری مشخص شده باشند.

خروجی طراحی ویژگی‌رانه، چندین سطح اول دید تجزیه ماژول یک معماری و سایر دیدهای مرتبط است (کلیه جزئیات دیدها از طریق بکارگیری طراحی ویژگی‌رانه قابل شناسایی نیستند). خروجی حاصل، نخستین

[1] Attribute-Driven Design (ADD)
[2] Decomposition process

خروجی معماری در جریان فرآیند طراحی است و بنابراین دانه‌بندی آن درشت است. با این حال برای دستیابی به خصوصیات کیفی مناسب وجود آن ضروری است، زیرا یک چارچوب کلی برای دستیابی به نیازهای وظیفه‌مندی فراهم می‌کند.

تفاوت بین معماری حاصل از طراحی ویژگی‌رانه و معماری آماده برای پیاده‌سازی، در جزئیات تصمیمات طراحی است که برای به کارگیری و پیاده‌سازی معماری مورد نیاز هستند. برای مثال تصمیم‌گیری در مورد استفاده از الگوی طراحی شیءگرای خاص یا بخش خاصی از میان‌افزار که همراه با آن محدودیت‌های معماری وارد می‌شود. نکته قابل توجه در مورد معماری طراحی شده به وسیله طراحی ویژگی‌رانه آن است که ممکن است تصمیم‌گیریهای لازم پیرامون معماری، برای داشتن انعطاف‌پذیری بیشتر به تاخیر انداخته شوند.

فرآیندهای طراحی مختلفی وجود دارند که می‌توانند با استفاده از سناریوهای عمومی (تشریح شده در فصل ۴)، تاکتیک‌ها و الگوها (تشریح شده در فصل ۵) ایجاد شوند. هر فرآیند درباره چگونگی تقسیم کردن کار طراحی و ماهیت فرآیند طراحی، پیش فرض‌های مختلف و خاص خودش را دارد.

به منظور تشریح طراحی ویژگی‌رانه، یک معماری خط تولید برای بازکننده‌های درب گاراژ در داخل سیستم اطلاعات خانگی[1] طراحی می‌شود. درب بازکن مسئول بالابردن و پایین آوردن درب به وسیله کلید، دستگاه کنترل از راه دور یا سیستم اطلاعات خانگی است. همچنین باید امکان تشخیص مشکلات درب بازکن از داخل سیستم اطلاعات خانگی وجود داشته باشد.

۷-۲-۱-۱- ورودی نمونه

ورودی طراحی ویژگی‌رانه مجموعه‌ای از نیازمندیها است. طراحی ویژگی‌رانه نیازهای وظیفه‌مندی (مخصوصا آنهایی که با مورد کاربری بیان می‌شوند) و محدودیت‌ها را به عنوان ورودی سیستم –مانند سایر روشهای طراحی– در نظر می‌گیرد، با این حال این روش در نحوه برخورد با نیازمندیهای کیفی با دیگر روشها تفاوت دارد. طراحی ویژگی‌رانه باعث می‌شود که نیازمندیهای کیفی به عنوان مجموعه‌ای از سناریوهای کیفی خاص سیستم در نظر گرفته شوند. سناریوهای عمومی به عنوان ورودی برای فرآیند نیازمندیها عمل می‌کنند و فهرستی را فراهم می‌کنند که در توسعه سناریوهای خاص سیستم استفاده می‌شود. سناریوهای خاص سیستم باید به جزئیاتی تبدیل شوند که برای سیستم ضروری هستند.

[1] Home information system

در مثال مربوط به خط تولید برای بازکننده درب گاراژ تعدادی از سناریوها حذف شده است زیرا آنها هیچ تاثیری در فرآیند طراحی ندارند ولی سناریوهای کیفی باقیمانده و مهم به شرح زیر هستند:

- ابزارها و کنترل کننده‌های لازم برای باز و بسته کردن درب مرتبط با محصولات مختلف در یک خط تولید متفاوت هستند و می‌توانند شامل کنترل کننده‌هایی باشند که در داخل یک سیستم اطلاعات خانگی استفاده می‌شوند. معماری تولید شده برای یک مجموعه خاص از کنترل کنندها باید به طور مستقیم از معماری خط تولید مشتق شود.
- پردازنده‌های استفاده شده برای محصولات گوناگون، متفاوت خواهد بود. بنابراین معماری تولید شده برای هر پردازنده خاص باید مستقیما از معماری خط تولید مشتق شود.
- اگر شخص یا شیء در هنگام پائین آمدن یا بالا رفتن درب گاراژ تشخیص داده شود پائین آوردن یا بالا بردن درب باید در عرض ۰.۱ ثانیه متوقف شود و عمل معکوس انجام شود.
- بازکننده درب گاراژ باید قابل دسترس از طریق سیستم اطلاعات خانگی باشد تا بتوان باز یا بسته بودن درب را تشخیص داده و همچنین آن را مدیریت کرد. این عملکرد می‌تواند با استفاده از یک قرارداد خاص محصول[1] بدست آید. بنابراین بایستی امکان تولید مستقیم یک معماری که بازگو کننده این قرارداد باشد را فراهم آورد.

۷-۲-۱-۲- آغاز طراحی ویژگی‌رانه

طراحی ویژگی‌رانه به شناسایی پیشرانها وابسته است و به محض اینکه کلیه آنها شناسایی شدند می‌تواند شروع شود. البته در طول طراحی، تشخیص اینکه کدام پیشرانها کلیدی هستند می‌تواند منجر به درک بهتر نیازمندیها یا تغییر بعضی از نیازمندیها شود. همچنین اینکه فرآیند طراحی زمانی هم که تعدادی از پیشرانها مشخص شده‌اند می‌تواند آغاز شود ولی به شرط اینکه اطمینان کافی در مورد آن پیشرانها وجود داشته باشد. حال در ادامه هر یک از گامهای مورد نیاز برای طراحی ویژگی‌رانه به صورت جزئی تشریح می‌شود و بعد از آن برای هر کدام از این گامها توضیحات کامل بیان خواهد شد.

[1] Product-specific diagnosis protocol

۱. *انتخاب ماژول برای تجزیه*

ماژولی که عملیات تجزیه با آن شروع می‌شود در واقع کل سیستم است. کلیه نیازمندیها برای این ماژول باید در دسترس باشد (محدودیت‌ها، نیازهای وظیفه‌مندی و نیازهای کیفی).

۲. *پالایش ماژولها*

أ. پیشرانهای معماری را از مجموعه سناریوهای کیفی عینی و نیازهای وظیفه‌مندی انتخاب کنید. در واقع این گام مشخص می‌کند که چه چیزی برای تجزیه مهم است.

ب. الگوی معماری را انتخاب کنید که پیشرانهای معماری را برآورده می‌کند. بدین منظور می‌توان الگوی معماری را مبتنی بر تاکتیک‌هایی که برای دستیابی به پیشرانها استفاده می‌شود، ایجاد یا انتخاب کرد.

ت. ماژولها را معرفی کرده و وظیفه‌مندیهای مشخص شده به وسیله موارد کاربری را به آنها تخصیص دهید و سپس با استفاده از چندین دید آنها را نمایش دهید.

ث. واسطهای ماژولهای فرزند را تعریف کرده (عمل تجزیه، ماژولها، محدودیت‌های موجود و انواع تعاملات بین آنها را معین می‌کند) و این اطلاعات را در مستند واسطهای مربوط به هر ماژول معین کنید.

ج. سناریوهای کیفی و موارد کاربری را پالایش و بازبینی کنید و محدودیت‌های آنها را به ماژولهای فرزند اعمال نمائید. این گام تصدیق می‌کند که هیچ چیز مهمی فراموش نشده است و ماژولهای فرزند را برای تجزیه بیشتر یا پیاده‌سازی آماده می‌کند.

۳. *تکرار گامهای بالا برای هر ماژولی که نیاز به تجزیه بیشتر دارد.*

۷-۲-۲-۳- انتخاب ماژول برای تجزیه

سیستم، زیرماژول و زیرسیستم همگی ماژول به شمار می‌آیند. فرآیند تجزیه عموما با خود سیستم شروع می‌شود که بعد آن سیستم به زیرسیستم تجزیه می‌شود و نهایتا در صورت نیاز به تجزیه بیشتر هر زیرسیستم به زیر ماژولها (ماژولهای فرزند) تبدیل می‌شود. در مثال درب بازکن گاراژ، بازکننده درب گاراژ سیستم است و یک محدودیت در این سطح آن است که دربازکن باید با سیستم اطلاعات خانگی ارتباط برقرار کند.

۷-۲-۲-۴- انتخاب پیشرانهای معماری

همان طور که قبلا ذکر شد پیشرانهای معماری از نیازهای وظیفه‌مندی و کیفی تشکیل شده‌اند که معماری یا یک ماژول خاص را شکل می‌دهند. پیشرانها از میان نیازمندیهای سیستم که اولویت بالایی دارند بدست می‌آیند. در مثال درب بازکن گاراژ چهار سناریوی مشخص شده در بخش ورودی نمونه، پیشرانهای معماری هستند. در سیستمی که این مثال مبتنی بر آن است تعدادی از خصوصیات کیفی نیز وجود دارند (کشف شدند) که عبارتند از: نیازمندی برای کارایی بلادرنگ، قابلیت تغییرپذیری برای حمایت از خط تولید. علاوه بر اینها، نیازمندی برای تشخیص برخط نیز باید توسط سیستم حمایت شود، پس تمامی این نیازمندیها باید در تجزیه اولیه سیستم برآورده شوند.

تشخیص پیشرانهای معماری همیشه یک فرآیند بالا به پائین نیست. گاهی اوقات بررسی کاملا جزئی برای درک نیازمندیهای خاص مورد نیاز است. برای مثال، برای تشخیص اینکه کارایی یک پیامد برای پیکربندی خاص سیستم است ممکن است نیاز به یک پیاده‌سازی نمونه‌ای از یک بخش سیستم داشته باشیم. در مثال درب بازکن گاراژ نیز تشخیص اینکه نیازمندی کارایی یک پیشران معماری است یا نه؟ باید مکانیزمهای استفاده شده برای باز کردن در توسط درب بازکن گاراژ و سرعت پردانده آن مورد بررسی قرار بگیرد.

در هنگام تجزیه در روش طراحی ویژگی‌رانه می‌توان صورت مسئله را طوری تغییر داد که امکان تأمین نیازمندیهای مهم فراهم شود زیرا نیازمندیهای کم اهمیت در داخل (محدودیت‌های) نیازمندیهای مهم برآورده می‌شوند. این خصوصیت مهمترین تفاوت بین طراحی ویژگی‌رانه و دیگر روشهای طراحی معماری است.

۷-۲-۲-۵- انتخاب الگوی معماری

برای هر خصوصیت کیفی تاکتیک‌هایی (و الگوهایی که این تاکتیک‌ها را پیاده‌سازی می‌کنند) وجود دارند که می‌توانند در طراحی معماری برای دستیابی به یک کیفیت مشخص مورد استفاده قرار بگیرند. هر تاکتیک طوری طراحی شده است که یک یا چند خصوصیت کیفی را تحقق ببخشد، اما الگوهایی که تاکتیک‌ها در داخل آن قرار دارند بر روی خصوصیات کیفی دیگر نیز تاثیر دارند. در طراحی معماری یک ترکیبی از تاکتیک‌های مختلف استفاده می‌شود تا بتوان به یک تعادل بین خصوصیات چندگانه مورد نیاز دست پیدا کرد. دستیابی به نیازهای وظیفه‌مندی و کیفی در جریان گام پالایش مورد تحلیل قرار می‌گیرد.

هدف از مرحله انتخاب الگوی معماری، بنا نهادن یک الگوی معماری کلی متشکل از انواع ماژولها است. این الگوی موردنظر پیشرانهای معماری را برآورده کرده و به وسیله ترکیب تاکتیک‌های مختلف انتخاب شده، ساخته می‌شود. در انتخاب یک تاکتیک دو فاکتور اصلی تاثیرگذار است. اولین فاکتور، خود پیشرانها هستند و دومین فاکتور تاثیرات جانبی است که الگوی پیاده‌سازی کننده تاکتیک بر روی خصوصیات کیفی دیگر دارد. برای مثال یک تاکتیک برای دستیابی به قابلیت تغییرپذیری استفاده از مفسر است. اضافه کردن یک زبان مشخصات تفسیری[1] به سیستم باعث می‌شود که ایجاد توابع جدید یا تغییر یک تابع از قبل موجود به آسانی انجام شود. همچنین ماکروی ضبط و اجرا[2]، مثالی از یک مفسر است به علاوه *HTML* نیز زبان مفسری است که شکل ظاهری یک صفحه وب را مشخص می‌کند.

با وجود اینکه استفاده از مفسر یک تاکنیک عالی برای دستیابی به قابلیت تغییرپذیری در زمان اجرا است اما تاثیر منفی شدیدی بر روی کارایی دارد. تصمیم‌گیری در مورد استفاده از مفسر به اهمیت قابلیت تغییرپذیری نسبت به کارایی وابسته است. البته گاهی یک تصمیم‌گیری می‌تواند طوری در نظر گرفته شود که از مفسر برای یک بخشی از الگوی کلی استفاده شود و برای سایر بخش‌ها از تاکتیک‌های دیگر استفاده شود.

در مثال درب بازکن گاراژ، با بررسی قابلیت دسترسی به این نتیجه می‌توان دست پیدا کرد که کارایی و قابلیت تغییرپذیری جزو خصوصیات کیفی حیاتی سیستم به شمار می‌روند. تاکتیک‌های قابلیت تغییرپذیری "محلی‌سازی تغییرات"، "جلوگیری از تاثیرات موجی"، "به تاخیر انداختن زمان انقیاد" و غیره هستند. از آنجایی که سناریوهای قابلیت تغییرپذیری برای سیستم فرضی کاملا با تغییراتی که در زمان طراحی اتفاق خواهند افتاد در ارتباط هستند بنابراین تاکتیک اصلی "محلی‌سازی تغییرات" است.

تاکتیک‌های "انسجام معنایی" و "پنهان‌سازی اطلاعات" نیز انتخاب شده و با یکدیگر در جهت ایجاد یک ماشین مجازی ترکیب می‌شوند تا تاثیرات بر حوزه‌های دیگر را کاهش دهند. تاکتیک‌های کارایی "درخواست منابع" و "حاکمیت بر منابع" هستند و از میان آنها نیز "افزایش بهره‌وری محاسبات[3]" و "انتخاب سیاست زمانبندی[4]" انتخاب می‌شود. با توجه به نکات بیان شده تاکتیک‌های حاصل به شرح زیر هستند:

[1] Interpreted specification language
[2] Macro recording and execution
[3] Increase computational efficiency
[4] Choose scheduling policy

- *پنهان‌سازی اطلاعات و انسجام معنایی*

مسئولیت‌های مجزا، با واسط کاربر، ارتباطات و سنسورهای درون خود ماژولها سروکار دارند. این ماژولها ماشین‌های مجازی نامیده می‌شوند. دلیل استفاده از ماشین مجازی نیز آن است که انتظار می‌رود که کاربر، ارتباطات و سنسورها مربوط به ماژولها به دلیل محصولات متفاوتی که از معماری مشتق می‌شوند، تغییر پیدا کنند. همچنین بدین منظور می‌توان مسئولیت‌های مرتبط با فعالیت نظارت را نیز از سایرین جدا کرد.

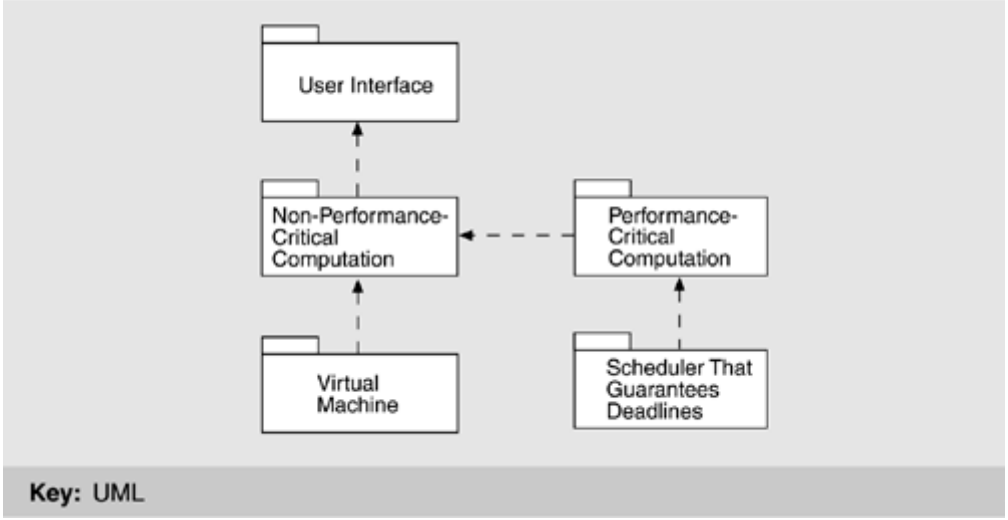
- *افزایش بهره‌وری محاسبات*

محاسباتی که از نظر کارایی بسیار مهم هستند باید تا حد ممکن کارآمد شوند.

- *زمانبندی عاقلانه*[1]

محاسباتی که از نظر کارایی بسیار مهم هستند باید طوری زمانبندی شوند که دستیابی به زمان آخرین فرصت (فرجه) در آنها تضمین شود.

شکل ۷-۲ یک الگوی معماری ترکیب شده از تاکتیک‌ها ذکر شده را نشان می‌دهد. از این تاکتیک‌ها می‌توان انواع مختلفی از الگوها را ایجاد کرد ولی الگوی ایجاد شده در شکل ۷-۲ به واقعیت خیلی نزدیکتر است.



شکل ۷-۲: الگوی معماریی که تاکتیک‌های مختلف را برای دستیابی به پیشرانهای درب بازکن گاراژ به کار می‌برد

[1] Schedule wisely

۷-۲-۲-۶- معرفی ماژولها و انتصاب وظیفه‌مندیها به ماژولها با استفاده از دیدهای چندگانه

در بخش قبلی تشریح شد که چگونه پیشرانهای کیفی معماری، ساختار تجریه ماژول را از طریق استفاده از تاکتیک‌ها مشخص می‌کنند. در حقیقت در آن گام نوع ماژولهای مشخص شده در گام تجزیه تعریف می‌شود. حال در ادامه به بررسی این موضوع پرداخته می‌شود که چگونه ماژولها معرفی می‌شوند:

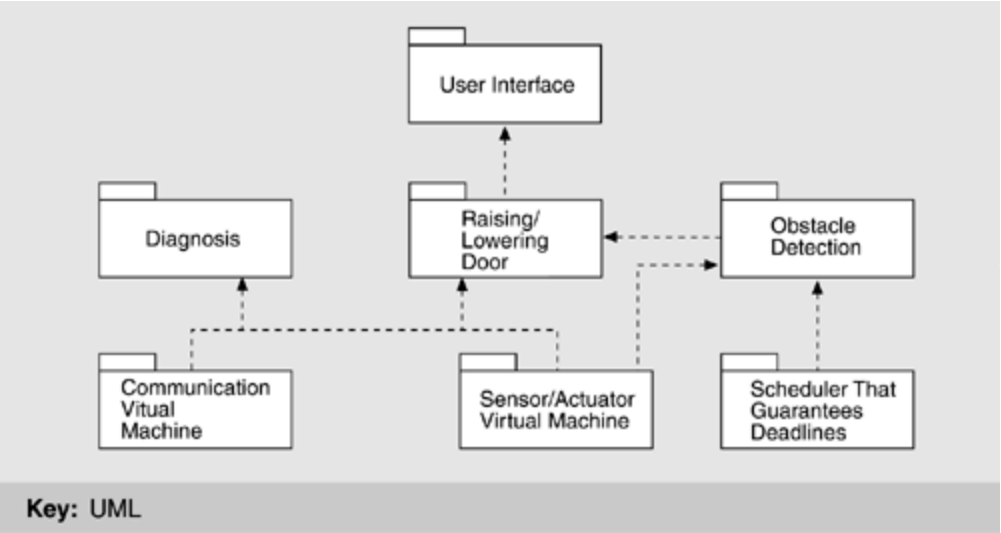
معرفی ماژولها

در شکل ۷-۲ یک مولفه محاسباتی غیرحساس از نظر کارایی[1] معرفی شده است که در سطح بالایی ماشین مجازی اجرا می‌شود (ماشین مجازی مسئولیتش مدیریت ارتباطات و تعاملات سنسورها است). نرم‌افزاری که در سطح بالایی یک ماشین مجازی اجرا می‌شود معمولا یک برنامه کاربردی است.

در یک سیستم منسجم، اصولا بیش از یک ماژول وجود دارد (برای هر گروه از عملکردها یک ماژول و برای انواع مشخص شده در الگوها نیز نمونه‌های مختلف وجود خواهد داشت) و معیارهای استفاده شده برای تخصیص عملکردها به آنها نیز مشابه معیارهای است که در روشهای طراحی مبتنی بر عملکردها از قبیل روشهای طراحی شیءگرا استفاده می‌شود.

در مثال درب بازکن گاراژ، مدیریت تشخیص مانع و متوقف کردن باز کننده درب گاراژ به بخش حساس از نظر کارایی[2] اختصاص می‌یابد زیرا این عملکرد دارای فرجه زمانی است. مدیریت باز و بسته کردن درب نیز به دلیل اینکه دارای فرجه زمانی نیست به بخش غیر حساس از نظر کارایی تخصیص داده می‌شود. قابلیت تشخیص نیز همچنین غیر حساس از نظر کارایی است. بنابراین ماژول غیرحساس از نظر کارایی در شکل ۷-۲ با ماژولهای تشخیص و باز و بسته کردن درب جایگزین می‌شود که این حالت در شکل ۷-۳ به تصویر کشیده شده است. همچنین چندین مسئولیت دیگر از ماشین مجازی نیز به نامهای خواندن و ارتباط سنسورها و کنترل محرک شناسایی شده‌اند که اینها نیز منجر به نمونه‌هایی از ماشین مجازی می‌شود که در شکل ۷-۳ نشان داده شده‌اند. نتیجه حاصل از این گام یک تجزیه قابل قبول از یک ماژول است که در بخش بعدی بررسی خواهد شد که به چه میزان این تجزیه موجب دستیابی به عملکردهای مورد نیاز سیستم شده است.

[1] Non-performance crtitical computation
[2] Performance-crtitical



شکل ۷-۳: تجزیه سطح اول بازکننده درب گاراژ

تخصیص وظیفه‌مندیها

بکارگیری موارد کاربری که وابسته با ماژولهای اصلی هستند به معمار کمک می‌کند که درک کاملی از عملکردهای توزیع شده داشته باشد. این امر همچنین منجر به اضافه کردن یا حذف ماژولهای فرزند در جهت تحقق کلیه عملکردهای مورد نیاز می‌شود. اما نهایتا هر مورد کاربری متعلق به یک ماژول اصلی باید به وسیله یک سری مسئولیت‌ها در داخل ماژولهای فرزند نمایش داده شود.

انتصاب مسئولیت‌ها به ماژولهای فرزند در تجزیه منجر به کشف اطلاعات ضروری قابل تبدیل می‌شود که این موضوع یک رابطه تولیدکننده/مصرف‌کننده بین آن ماژولها ایجاد می‌کند. در این قسمت از طراحی خیلی مهم نیست که مشخص شود چگونه این تبدیلات باید انجام شود. به عنوان مثال سئوالاتی وجود دارند که باید بعد از فرآیند طراحی به آنها پاسخ داده شود از جمله اینکه آیا اطلاعات وارد یا خارج می‌شود؟ آیا اطلاعات به صورت پیام منتقل می‌شود یا از طریق پارامتر؟ در واقع در این مرحله از طراحی، تنها خود اطلاعات و نقش مصرف‌کننده و تولیدکننده مهم است. این موضوع نشان دهنده یک مثال از نوع مشکلاتی است که در طراحی ویژگی‌رانه بدون حل باقی می‌ماند و در طول طراحی که به بررسی جزئیات پرداخته می‌شود، حل می‌شود. بعضی تاکتیک‌ها، الگوهای خاصی از تعامل بین انواع ماژولها را معرفی می‌کنند. به عنوان مثال تاکتیکی که از

نوع میانجی *publish/subscribe* استفاده می‌کند یک الگو را معرفی می‌کند که در آن الگوی *publish* برای یکی از ماژولها و الگوی *subscribe* برای دیگر ماژولها استفاده می‌شود.

نمایش معماری با دیدها

در طراحی ویژگی‌رانه یک دید از هر سه گروه اصلی از دیدها (ماژول تجزیه، همزمانی و استقرار) برای شروع طراحی کافی است. روش طراحی ویژگی‌رانه به یک دید خاص وابسته نیست و اگر نیاز به نمایش یک جنبه خاص داشته باشد (مانند اشیاء زمان اجرا) دیدهای دیگر می‌توانند معرفی شوند. حال در ادامه تشریح خواهد شد که چگونه طراحی ویژگی‌رانه از این سه دید عمومی استفاده می‌کند:

- *دید تجزیه ماژول*: فراهم کننده محدودیت‌هایی برای مسئولیت‌هایی است که کشف می‌شوند. رابطه جریان داده اصلی در میان ماژولها نیز از طریق این دید شناسایی می‌شود.
- *دید همزمانی*: می‌تواند جنبه‌های پویای سیستم مانند فعالیت‌های موازی و همگام‌سازی را مدل کند. این نوع مدلسازی می‌تواند به شناسایی مشکلات مربوط به رقابت منابع، موقعیت‌های ایجاد کننده بن‌بست، پیامدهای مربوط به سازگاری داده و غیره کمک کند. مدلسازی همزمانی سیستم احتمالا منجر به کشف مسئولیت‌های جدید برای ماژولهایی می‌شود که در دید ماژول ثبت شده‌اند. این نکته همچنین می‌تواند منجر به کشف ماژولهای جدید از قبیل مدیریت منابع و حل پیامدهای دسترسی همزمان به داده‌های کمیاب و غیره شود.

  دید همزمانی یکی از دیدهای مولفه و اتصال است. مولفه‌ها نمونه‌هایی از ماژولها در دید تجزیه ماژول و اتصالات نشان‌دهنده نخهای مجازی[1] هستند. یک نخ مجازی تشریح کننده یک مسیر اجرایی از طریق سیستم یا بخشی از آن است. نخ مجازی نباید با نخ سیستم عامل یکی در نظر گرفته شود زیرا نخ سیستم عامل بر دیگر خصوصیات همچون تخصیص حافظه و پردازنده دلالت می‌کند که در سطح طراحی از اهمیت خاصی برخوردار نیستند. با این وجود، بعد از اینکه تصمیم‌گیری در مورد سیستم عامل و استقرار ماژولها به واحدهای پردازشی تمام شد نخهای مجازی به نخهای سیستم عامل تبدیل (نگاشت) می‌شوند که این عمل در جریان طراحی تفصیلی انجام می‌شود.

[1] Virtual threads

اتصالات در دید همزمانی، آنهایی هستند که با نخهایی همچون "همزمانی با[1]"، "شروع[2]"،" لغو" و "ارتباط با" سروکار دارند. یک دید همزمانی نشان‌دهنده نمونه‌های ماژولهای موجود در دید تجزیه ماژول است به عبارت بهتر این نمایش به عنوان روشی برای درک نگاشت بین این دو دید به کار می‌رود. مشخص کردن این نگاشت خیلی مهم است زیرا که مشخص می‌شود یک نقطه همزمانی در کدام ماژول قرار گرفته است و بدین صورت می‌توان مسئولیت مورد نظر را به ماژول مناسب تخصیص داد. برای درک همزمانی در یک سیستم، موارد کاربری زیر می‌تواند کاربرد داشته باشد:

– *دو کاربر کارهای یکسانی را در زمان یکسان انجام می‌دهند*. این مورد کاربری به تشخیص مشکلات رقابت منابع و یکپارچگی داده کمک می‌کند. به عنوان مثال در سیستم درب بازکن گاراژ، یک کاربر می‌تواند درب را با کنترل ببند، درحالی که کاربر دیگر با استفاده از کلید آن را باز می‌کند.

– *یک کاربر چندین کار را به صورت همزمان انجام می‌دهد*. این مورد کاربری به تشخیص مشکلات کنترل فعالیت‌ها و تبدیل داده‌ها کمک می‌کند. به عنوان مثال در سیستم درب بازکن گاراژ، یک کاربر می‌تواند فعالیت تشخیص را انجام دهد در حالی که همزمان درب را باز می‌کند.

– *راه‌اندازی سیستم*. این مورد کاربری یک دید کلی خوب از فعالیت‌های اجرایی ثابت در سیستم و چگونگی آغاز شدن آنها می‌دهد. همچنین در مورد تصمیم‌گیری پیرامون راهبرد راه‌اندازی نیز کمک می‌کند به عنوان مثال اینکه آیا کلیه فرآیندهای مربوط به راه‌اندازی به صورت موازی یا ترتیبی انجام شوند. به عنوان نمونه در سیستم درب بازکن گاراژ، می‌توان این سئوالات را در ذهن داشت که آیا شروع به فعالیت کردن سیستم وابسته به قابل دسترس بودن سیستم اطلاعات خانگی است یا نه؟ آیا سیستم همیشه در حال اجرا است، منتظر یک سیگنال است و یا اینکه قبل از شروع فرآیند باز و بسته کردن فعال و بعد از آن متوقف می‌شود؟

– *خاموش کردن سیستم*. این مورد کاربری به روشن کردن موضوعاتی همچون دستیابی و نگهداری سیستم در یک وضعیت سازگار کمک می‌کند.

[1] Sysnchronizes with
[2] Starts

– در مثال سیستم درب بازکن گاراژ، یک نقطه همزمانی در ماشین مجازی سنسور/محرک وجود دارد. بخش حساس به کارایی باید از سنسور نمونه‌گیری کرده و همچنین باید درب را بالا و پائین ببرد. این موضوع قابل قبول است که بخش حساس به کارایی ماشین مجازی سنسور/محرک در زمانی که یک کنش برای بخش بالا و پائین کننده درب انجام می‌دهد، متوقف نماید. بنابراین نیاز به یک مکانیزم همزمان‌سازی برای ماشین مجازی سنسور/محرک وجود دارد.

همزمانی همچنین می‌تواند یک نقطه تغییر در نظر گرفته شود (در فصل مربوط به معماری خط تولید تشریح خواهد شد). برای بعضی محصولات در یک خط تولید یک مقداردهی اولیه ترتیبی می‌تواند خوب کار کند در حالی که برای محصولات دیگر این مقداردهی باید به صورت موازی انجام شود. بنابراین اگر مکانیزم تجزیه تکنیک‌هایی را برای تغییر روشهای مقداردهی ارائه نکند در این صورت باید تغییراتی در روش تجزیه صورت بپذیرد.

- دید *استقرار*: اگر چندین پردازنده یا سخت‌افزار خاص در سیستم استفاده شود مسئولیت‌های اضافی ممکن است ایجاد شود. استفاده از یک دید استقرار کمک می‌کند تا استقراری طراحی شود که از دستیابی به خصوصیات کیفی مطلوب حمایت کند.

دید استقرار که از نخهای مجازی درون دید همزمانی ایجاد شده است به نخهای مجازی درون پردازنده خاص و پیام‌هایی تبدیل می‌شود که بین پردازنده‌ها حرکت می‌کنند تا ورودی بعدی در ترتیب کنشها را مقداردهی اولیه کند. بنابراین دید استقرار پایه‌ای برای تحلیل ترافیک شبکه است.

دید استقرار همچنین به تصمیم‌گیری در مواردی که چندین نمونه از برخی ماژولها مورد نیاز است، کمک می‌کند. برای مثال یک نیازمندی قابلیت اطمینان ممکن است باعث شود که عملکردهای حیاتی بر روی پردازنده‌های مختلف تکرار شود. یک دید استقرار همچنین از استدلال پیرامون استفاده از سخت‌افزارهایی با کاربرد خاص نیز حمایت می‌کند.

انتقال یک نخ مجازی از یک پردازنده به پردازنده دیگر مسئولیت‌هایی را برای ماژولهای مختلف تولید می‌کند. به عنوان مثال این انتقال می‌تواند دلالت بر یک نیازمندی ارتباطی بین پردازنده‌ها نماید و اینکه بعضی از ماژولها نیز باید مسئول مدیریت ارتباطات باشند. بنابراین چنین مسئولیتی باید در دید تجزیه ماژول در نظر گرفته شود. در مثال درب بازکن گاراژ، پیامدهای استقرار ناشی از تقسیم

مسئولیت‌ها بین سیستم درب بازکن گاراژ و سیستم اطلاعات خانگی است. در واقع سئوال اصلی این است که کدام یک مسئول تشخیص هویت یک درخواست از راه دور است یا اینکه قرارداد ارتباطی بین آن دو چیست؟

۷-۲-۲-۷- تعریف واسط‌های زیر ماژولها (ماژولهای فرزند)

در طراحی ویژگی‌رانه، واسط یک ماژول سرویسها و خصوصیات فراهم شده و مورد نیاز را نشان می‌دهد. واسط از امضاء[1] متفاوت است. در واقع واسط مستند می‌کند که چه چیزهایی می‌تواند استفاده شود و اینکه آن سرویس به چه چیزهایی وابسته است. تحلیل و مستندسازی تجزیه برحسب ساختار (دید تجزیه ماژول)، پویایی[2] (دید همزمانی) و زمان اجرا (دید استقرار)، فرضیات تعاملی برای ماژولهای فرزند را آشکار می‌کند که این فرضیات باید در واسط ماژولها مستند شود. دید ماژول اطلاعات زیر را مستند می‌کند:

- اطلاعات مربوط به مصرف‌کنندگان/تولیدکنندگان
- الگوی تعاملی که ماژولها مبتنی بر آن سرویس داده و یا سرویس دریافت می‌کنند.

دید همزمانی اطلاعات زیر را مستند می‌کند:

- تعاملات بین نخهایی که منجر به واسط‌های ماژولها می‌شوند.
- اطلاعاتی پیرامون اینکه آیا یک مولفه فعال است یا نه؟ برای مثال، آیا مولفه نخ در حال اجرا دارد یا نه.

دید استقرار اطلاعات زیر را مستند می‌کند:

- نیازمندیهای سخت‌افزاری، به عنوان مثال سرعت محاسبات یک پردازنده مجبور است حداقل ۱۰ مگا بیت بر ثانیه باشد.
- نیازمندیهای ارتباطی، به عنوان مثال اطلاعات نباید بیش از یک بار در هر ثانیه به روز شوند.

۷-۲-۲-۸- پالایش و بازبینی موارد کاربری و خصوصیات کیفی

در این مرحله باید ماژولهایی که مبتنی بر دید تجزیه بدست آمده‌اند، بازبینی شوند و ماژولهای فرزند نیز باید برای تجزیه سطح بعدی آماده شوند.

[1] Signature
[2] Dynamism

نیازمندیهای وظیفه‌مندی

هر ماژول فرزند مسئولیت‌هایی دارد که از ملاحظات مربوط به تجزیه نیازهای وظیفه‌مندی مشتق می‌شوند. این مسئولیت‌ها برای هر ماژول می‌تواند به موارد کاربری تبدیل شوند. راه دیگر برای تعریف موارد کاربری تجزیه و پالایش موارد کاربری اصلی (پدر) است. برای مثال، یک مورد کاربری که کل سیستم را مقداردهی اولیه می‌کند به چندین زیرسیستم تجزیه می‌شود که در مجموع عمل مقدادهی اولیه را انجام می‌دهند. این روش قابلیت ردیابی[1] دارد زیرا تحلیل‌گر می‌تواند مراحل پالایش را دنبال کند.

در سیستم درب بازکن گاراژ، مسئولیت‌های اولیه برای باز کننده درب عبارتند از: باز و بستن درب در هنگام درخواست است که این درخواست می‌تواند از راه دور یا به صورت محلی باشد، متوقف کردن در در مدت ۰.۱ ثانیه زمانی که یک مانع تشخیص داده می‌شود و ارتباط با سیستم اطلاعات خانگی و پشتیبانی از تشخیص‌های از راه دور. این مسئولیت‌ها به داخل گروه‌های عملکردی مرتبط با ماژولها، تجزیه می‌شود که به شرح زیر هستند:

- *واسط کاربری*، تشخیص درخواست‌های کاربر و تبدیل آنها به قالب مورد نیاز ماژول بالا/پائین بر درب
- *ماژول بالا/پائین بر درب*، کنترل محرک برای بالا یا پائین بردن درب و متوقف کردن درب وقتی درب کاملا باز یا بسته شد.
- *تشخیص مانع*، تشخیص اینکه یک مابع در میان درب وجود دارد بنابراین باید حرکت آن متوقف شود و عمل عکس اتفاق بیفتد (یعنی اگر در حال باز بودن باشد متوقف شده و بسته شود).
- *ماشین مجازی ارتباطات*، مدیریت کلیه ارتباطات با سیستم اطلاعات خانگی
- *ماشین مجازی سنسور/محرک*، مدیریت کلیه تعاملات بین سنسورها و محرک‌ها
- *زمانبند*، تضمین اینکه تشخیص دهنده مانع فرجه زمانی خود را برآورده خواهد کرد.
- *تشخیص*، مدیریت تعاملات با سیستم اطلاعات خانگی که نیاز به بررسی و تشخیص حالت درب دارد.

[1] Traceability

محدودیت‌ها

محدودیت‌های ماژول اصلی (پدر) می‌تواند به وسیله یکی از روشهای زیر برآورده شود:

- *تجزیه محدودیت را برآورده می‌کند.* برای مثال محدودیت استفاده از یک سیستم عامل می‌تواند به وسیله تعریف یک ماژول فرزند برآورده شود یا اینکه محدودیت برآورده شده است و نیازی به انجام هیچ کاری نیست.
- *محدودیت به وسیله یک ماژول فرزند منفرد برآورده می‌شود.* برای مثال محدودیت استفاده از قرارداد خاص می‌تواند به وسیله تعریف یک ماژول فرزند محصور شده برای آن قرارداد، برآورده شود.
- *محدودیت به وسیله چندین ماژول فرزند برآورده می‌شود.* برای مثال استفاده از وب نیازمند دو ماژول (سرویس‌گیرنده و سرویس‌دهنده) برای پیاده‌سازی قراردادهای ضروری مورد نیاز است. به هر حال محدودیت، مبتنی بر تجزیه و هماهنگی فرزندان برآورده می‌شود.

در مثال درب بازکن گاراژ، یک محدودیت آن است که ارتباط با سیستم اطلاعات خانگی باید پیوسته حفظ شود. ماشین مجازی ارتباطات در صورت عدم قابل دسترس بودن ارتباطات آن را تشخیص می‌دهد، بنابراین این محدودیت به وسیله یک ماژول منفرد برآورده می‌شود.

سناریوهای کیفی

سناریوهای کیفی می‌بایستی که پالایش شده و به ماژولهای فرزند تخصیص داده شوند.

- یک سناریوی کیفی می‌تواند کاملا از طریق تجزیه و بدون هیچ تاثیر اضافی برآورده شود و بعد از برآورده شدن نیز می‌تواند علامتگذاری شود تا مشخص شود برآورده شده است.
- یک سناریوی کیفی می‌تواند به وسیله تجزیه جاری برآورده شود ولی منجر به محدودیت‌هایی بر روی ماژولهای فرزند شود. برای مثال، استفاده از لایه‌بندی می‌تواند یک سناریوی قابلیت تغییرپذیری خاصی را برآورده کند که در آن صورت الگوی استفاده شده برای فرزندان محدود خواهد شد.
- تجزیه می‌تواند هیچ ارتباطی با سناریوهای کیفی نداشته باشد. برای مثال، یک سناریوی قابلیت استفاده وابسته به بخشی از واسط کاربری است که آن نیز هنوز بخشی از تجزیه نیست، لذا این سناریو باید به یک ماژول فرزند تخصیص داده شود.

- سناریوی کیفی نمی‌تواند با تجزیه جاری برآورد شود. اگر سناریوی کیفی برآورده نشده مهم باشد در این صورت عمل تجزیه باید دوباره انجام شود در غیر این صورت، دلایل مربوط به اینکه تجزیه از این سناریو حمایت نمی‌کند باید ثبت شود. این حالت معمولا حاصل مصالحه بین سناریوهای کیفی مخصوصا سناریوهای کیفی با اولویت بالا است.

به عنوان مثال در سیستم دربازکن گاراژ، سناریوهای کیفی شناسایی شده طبق روش زیر پالایش یا برآورده شده‌اند:

- دستگاه‌ها و کنترل کننده‌ها برای باز و بستن کردن درب، در میان محصولات مختلف متعلق به یک خط تولید متفاوت هستند. آنها دربرگیرنده کنترل کننده‌هایی از داخل سیستم اطلاعات خانگی هستند. این سناریو به ماژول واسط کاربری محول شده است.
- پردازنده استفاده شده در محصولات مختلف، متفاوت خواهد بود. معماری خاصی برای هر محصول باید مستقیما از معماری خط تولید حاصل شود. این سناریو به کلیه ماژولها محول شده است. هر ماژول مسئول ویژگیهای خاص پردازنده می‌شود که توسط کامپایلرهای استاندارد حمایت نمی‌شود.
- اگر یک مانع (انسان یا شیء) در زمان باز شدن یا بسته شدن درب شناسایی شود عمل باز یا بسته شدن باید در عرض ۰.۱ ثانیه متوقف شود. این سناریو به زمانبند و ماژول تشخیص دهنده مانع محول شده است.
- بازکننده درب باید به منظور تشخیص و مدیریت از طرف سیستم اطلاعات خانگی منطبق بر یک قرارداد تشخیص خاص محصول، قابل دسترس باشد. این سناریو به دو ماژول تشخیص و ارتباطات محول شده است. ماژول ارتباطات مسئول قرارداد استفاده شده برای ارتباط با سیستم اطلاعات خانگی است و ماژول تشخیص نیز مسئول مدیریت سایر تعاملات مربوط به فعالیت تشخیص است.

در پایان این گام یک ماژول به تعدادی ماژول فرزند تجزیه می‌شود که هر ماژول فرزند مجموعه‌ای از مسئولیت‌ها (یک مجموعه از موارد کاربری، سناریوهای کیفی و یک مجموعه از محدودیت‌ها) را دارد. این ماژولها به همراه مسئولیت‌های خودشان برای شروع تکرار بعدی کافی هستند.

۷-۳- شکلی‌دهی به ساختار تیم

بعد از اینکه نخستین سطوح ساختار تجزیه ماژول معماری پدیدار شدند آن ماژولها می‌توانند به تیم‌های توسعه تخصیص داده شوند. حاصل این کار، دید تخصیص کار است که در فصل دوم به آن اشاره شد. این دید یا ماژولها را به واحدهای توسعه از قبل موجود تخصیص می‌دهد یا یک واحد توسعه جدید تعریف می‌کند.

تاثیر معماری بر توسعه ساختارهای سازمانی کاملا واضح است. وقتی که معماری یک سیستم در حال ساخت مورد پذیرش قرار می‌گیرد تیم‌های توسعه ایجاد می‌شوند تا بر روی ماژولهای اصلی فعالیت کنند یا ماژولها اصلی به تیم‌های توسعه از قبل موجود اختصاص داده می‌شود و همچنین یک ساختار تفکیک کار نیز ایجاد می‌شود که مشخص کننده وظایف این تیم‌ها است. سپس هر تیم تمرینات[1] (اصول) داخلی مربوط به خود را شروع می‌کند. برای سیستم‌های بزرگ، تیم‌های توسعه می‌توانند متعلق به شرکت‌های تابعه یا همکار باشند. اصول داخلی هر تیم می‌تواند در تابلو اعلانات و صفحات وب برای ارتباط، با هدف مشخص کردن قرارداد نامگذاری فایلها و سیستم پیکرندی فایلها در نظر گرفته شود. کلیه این اصول می‌تواند از گروهی به گروه دیگر متفاوت باشد (مخصوصا برای سیستم‌های بزرگ). همچنین تضمین کیفیت و رویه‌های آزمایش برای هر گروه مشخص می‌شود و هر گروه نیاز دارد که ارتباطات و هماهنگی‌هایی با گروه‌های دیگر برقرار کند. بنابراین تیم‌ها در داخل یک سازمان بر روی ماژولها کار می کنند.

در داخل یک تیم نیاز به ارتباطات قوی وجود دارد (اطلاعات زیاد در قالب تصمیمات طراحی جزئی به صورت ثابت به اشتراک گذاشته می‌شود) و بین تیم‌ها نیز باید ارتباطات کمی وجود داشته باشد. دستیابی به این نکات مستلزم این است که عمل جداسازی دغدغه‌ها به صورت مناسب در سیستم انجام شده باشد.

ساخت سیستم‌ها زمانی خیلی پیچیده می‌شود که معیارهای مشخص شده برای طراحی در نظر گرفته نشوند. در حقیقت ساختار تیم و کنترل تعاملات تیم‌ها اغلب فاکتورهای خیلی مهمی محسوب می‌شوند که موفقیت یک پروژه بزرگ را تحت تاثیر قرار می‌دهند. اگر تعاملات بین تیم‌ها نیازمند پیچیدگی باشد یا تعاملات بین عناصری که آنها ایجاد می‌کنند بدون دلیل پیچیده است و یا نیازمندیهایی برای آن عناصر قبل از برقراری ارتباط

[1] Practices

مورد نظر است، در این حالت نیاز به یک رابطه قوی بین تیم‌های توسعه است (تیم‌های توسعه باید مشابه با سیستم‌های نرم‌افزاری انسجام بیشتر ولی وابستگی کمی داشته باشند).

حال سئوال این است که چرا ساختار تیم منعکس کننده ساختار تجزیه ماژول است؟ اولا، پنهان‌سازی اطلاعات درون یک تیم انجام می‌پذیرد و ثانیا هر تیم توانایی انجام وظایف در ناحیه[1] خاصی را دارد و هر ماژول نیز باید مربوط به ناحیه خاصی باشد (ناحیه به معنی حیطهٔ با تخصص یا تجربه خاص است). در زیر مثالهایی از ناحیه ارائه شده است

- ماژول یک لایه واسط کاربری از سیستم است. واسط برنامه‌نویسی کاربردی که به دیگر ماژولها ارائه می‌شود مستقل از دستگاه‌های واسط کاربری خاصی است که برای ارائه اطلاعات به کاربران انسانی استفاده می‌شود زیرا آنها ممکن است تغییر پیدا کنند. ناحیه در اینجا نمایشی از این دستگاه‌ها است.
- ماژول، یک زمانبند فرآیند[2] است که تعداد پردازنده‌های قابل دسترس و همچنین الگوریتم زمانبندی را مخفی می‌کند.
- ماژول، یک ماژول مدلهای فیزیکی از معماری *A-7E* است. آن ماژول معادله‌ای که مقادیر پیرامون محیط فیزیکی را محاسبه می‌کند، محصور می‌کند. ناحیه اینجا تحلیل عددی و ارتباطات فضایی است.

کارآمدترین روش برای استفاده از نیروهای انسانی آن است که نیروها بر حسب خبرگیشان به تیم‌ها تخصیص داده شوند. که این امر را فقط ساختار ماژول امکان‌پذیر می‌کند (سازمانها گاها تیم‌هایی را تشکیل می‌دهند که مستقل از ساختارهای معماری هستند).

۷-۴- ساخت اسکلت سیستم

وقتی که طراحی معماری به صورت مناسب انجام شد و تیم‌ها آماده برای شروع ساخت آن شدند یک اسکلت سیستم می‌تواند ایجاد شود. هدف در این مرحله آن است که قابلیت زیربنایی برای پیاده‌سازی یک وظیفه‌مندی سیستم به صورت موثر فراهم شود. اصول مهندسی نرم‌افزار کلاسیک، نادیده گرفتن[3] بخشهایی از کد را پیشنهاد می‌دهد تا قسمت‌های مختلف سیستم بتوانند به صورت مجزا اضافه شده و مستقلا آزمایش شوند. اما سئوال این

[1] Domain
[2] Process scheduler
[3] Stubbing out

است که کدام بخشها باید نادیده گرفته شود؟ جواب این است که استفاده از معماری به عنوان یک راهنما، ترتیبی از پیاده‌سازی‌ها را کاملا مشخص خواهد کرد.

ابتدا بخشهایی از نرم‌افزار باید پیاده‌سازی شوند که با مولفه‌های تعاملی و اجرایی معماری ارتباط دارند. این امر ممکن است نیاز به ایجاد یک زمانبند در سیستم‌های بلادرنگ داشته باشد، پیاده‌سازی موتور قوانین[1] در یک سیستم مبتی بر قانون باشد، پیاده‌سازی مکانیزم‌های همگام‌سازی فرآیند در یک سیستم چند فرآیندی یا پیاده‌سازی هماهنگ کننده سرویس‌گیرنده و سرویس‌دهنده در یک سیستم سرویس‌دهنده و سرویس‌گیرنده در نظر گرفته شود.

اغلب مکانیزم‌های تعاملی پایه به وسیله یک میان افزار فراهم می‌شوند که در این حالت، مکانیزم تعاملی به جای اینکه پیاده‌سازی شود، نصب می‌شود. در سطح بالایی این ساختار تعاملی و ارتباطی ممکن است توابع ساده که بیشتر مورد استفاده قرار می‌گیرند نصب شوند. بعد از انجام این کار یک سیستم در حال اجرا بدست می‌آید که فقط با خودش در حال ارتباط است. این سیستم حاصل در واقع زیربنایی است که مشخص می‌کند بر روی آن کدام وظیفه‌مندیها می‌توانند اضافه شود. در این حالت می‌توان به راحتی وظیفه‌مندیهای که باید به سیستم اضافه شوند را انتخاب کرد. انتخاب مربوط به وظیفه‌مندی سیستم می‌تواند مبتنی بر کم کردن خطر به وسیله برآورده کردن وظیفه‌مندیهای حوزه‌های خیلی مشکل‌ساز باشد، همچنین مبتنی بر سطوح و نوع کارکنان قابل دسترس باشد و یا وظیفه‌مندیهای که در صورت پیاده‌سازی باعث می‌شوند نرم‌افزار خیلی سریع در بازار به موفقیت دست پیدا کند. به علاوه وقتی که عناصر مرحله بعدی توسعه انتخاب شدند می‌توان از ساختار "استفاده" بهره برد تا اینکه مشخص شود کدام مولفه‌ها باید به صورت صحیح اجرا شوند تا بتوان یک وظیفه‌مندی را بطور کامل پیاده‌سازی کرد.

بنابراین هر چه قدر این فرآیند ادامه پیدا می‌کند سیستم به صورت تدریجی بزرگ و بزرگتر می‌شود تا اینکه کلیه بخش‌ها پیاده‌سازی می‌شوند. در این روند هیچ نگرانی درباره یکپارچگی وجود ندارد زیرا در هر روند تدریجی توسعه (تکرار) می‌توان به راحتی منبع نقص را شناسایی کرد. بودجه و زمانبندی نیز خیلی خوب قابل پیش‌بینی است زیرا هر تکرار به صورت یک سیستم کوچک در نظر گرفته می‌شود. در این روش، ریشه‌ها[2] به

[1] Rule engine
[2] Stubs

واسط‌هایی که در نسخه نهایی وجود خواهند داشت تبدیل می‌شوند بنابراین آنها می‌توانند به درک و آزمایش تعاملات بین مولفه‌ها حتی در صورت عدم حضور وظیفه‌مندیهای اولویت بالا کمک کنند. این عمل از بعد دیگر منجر به یک مشکل می‌شود زیرا واسط‌هایی که اولین تیم توسعه‌دهنده تعریف کرده است مجبورند برای کلیه سیستم‌ها و زیرسیستم‌های بعدی استفاده شوند که برای حل این مشکل می‌تواند درباره نحوه ارتباط زیرسیستم‌ها از قبل مصالحه کرد.

فهرست مطالب

فصل هشتم

شبیه‌ساز پرواز. مطالعه موردی در معماری برای دستیابی به قابلیت تجمیع‌پذیری

در این فصل یک مطالعه موردی به نام شبیه‌ساز پرواز مورد بررسی قرار خواهد گرفت. هدف از تشریح این مطالعه موردی آشنا شدن با نحوه طراحی معماری برای یک سیستم است. در روند طراحی معماری این مطالعه موردی بیان خواهد شد که به چه صورت باید سیستم را به بخشهای مختلف تقسیم کرده و سپس وظیفه‌مندیها را به آنها تخصیص داد تا بتوان نیازمندیهای سیستم را برآورده کرد.

۸-۱- مقدمه

شبیه‌سازهای پرواز امروزی در میان پیچیده‌ترین سیستمهای نرم‌افزاری موجود قرار دارند. آنها خیلی توزیع‌شده و وابستگی شدیدی به زمانبندی دارند، همچنین باید این امکان را داشته باشند که با هر تغییر در ابزار و یا محیطی که آنها پشتیبانی می‌کنند، سازگار شوند. به عبارت بهتر، باید ساختارشان طوری باشد که به روز رسانی‌های مکرر را به راحتی قبول کنند. ایجاد و نگهداری یک چنین سیستمهای بزرگ چالشهای توسعه نرم‌افزاری مهمی را در طراحی آنها ارائه می‌کنند که این چالشها عبارتند از:

- *کارایی شدیدا بلادرنگ*[1]
- *قابلیت تغییرپذیری* به منظور تطبیق با تغییرات در نیازمندیها، هواپیماها و محیط‌های شبیه‌سازی شده
- *قابلیت مقیاس‌پذیری در وظایف*[2] (شکلی از قابلیت تغییرپذیری)، به منظور گسترش سیستم در جهت حمایت بیشتر از دنیای واقعی و بهبود دقت شبیه‌سازی

قابلیت یکپارچه‌سازی (تجمیع‌پذیری) به این موضوع اشاره دارد که به چه سهولتی مولفه‌های توسعه یافته مجزا، از جمله مولفه‌هایی که توسط تیم‌ها یا شرکت‌های مختلف توسعه داده شده‌اند، می‌توانند با یکدیگر در جهت برآورده کردن نیازمندیهای نرم‌افزار کار کنند. مشابه سایر خصوصیات کیفی، در جهت برآورده کردن قابلیت تجمیع‌پذیری می‌توان از تاکتیک‌های طراحی شده بدین منظور استفاده کرد (بعضی از این تاکتیک‌ها با تاکتیک‌های قابلیت تغییرپذیری یکی هستند). این تاکتیک‌ها عبارتند از:

- *پایدار، ساده و کوچک نگاه داشتن واسط‌ها*
- *پایبند بودن*[3] *به قراردادهای تعریف شده*
- *استفاده از واسط‌های نسخه‌بندی شده*[4]، استفاده از این تاکتیک اجازه اعمال کردن گسترشها[5] را می‌دهد در حالی که به عناصر موجود اجازه می‌دهد که تحت قوانین اصلی به کار خود ادامه دهند.

[1] Hard real-time performance
[2] Scalability
[3] Adhering
[4] Versioned
[5] Extensions

در بخشهای بعدی تعدادی از چالشهای مربوط به شبیه‌ساز پرواز مورد بررسی قرار خواهد گرفت و یک الگوی معماری به نام مدل ساخت‌یافته برای برآورده کردن نیازمندیهای این سیستم معرفی خواهد شد که تاکید آن بر روی موضوعات زیر است:

- سادگی و شباهت زیرساختارهای سیستم
- جداکردن راهبردهای انتقال کنترل و پیام از محاسبات
- راهبردهای مربوط به هماهنگی سیستم (ناشی از گستردگی سیستم)
- شفافیت طراحی[1]

کلیه این اصول منجر به یک الگوی معماری می‌شود که قابلیت تجمیع‌پذیری بالایی را برای یک سیستم فراهم می‌کند (این الگو، ترکیبی از چندین الگوی اولیه و پایه است).

۸-۲- چرخه حرفه معماری

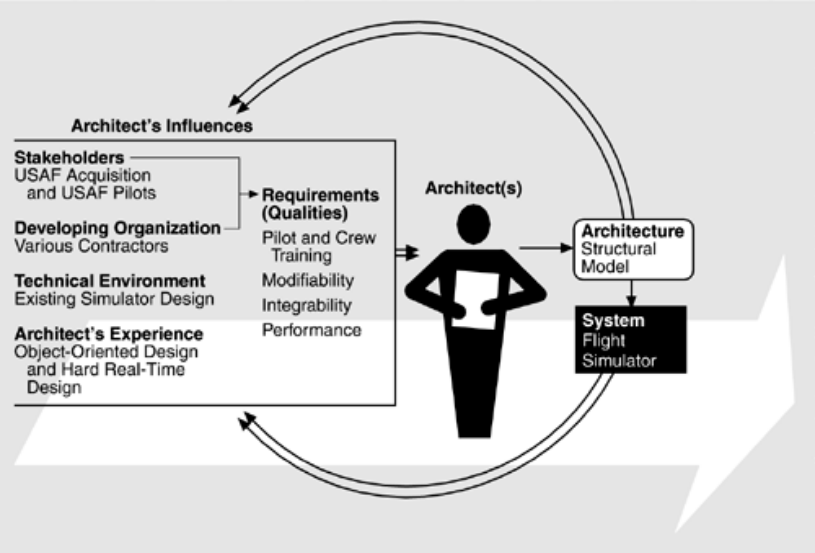
شکل ۸-۱ چرخه حرفه معماری را برای شبیه‌ساز پرواز مبتنی بر مدل ساختاری[2] نشان می‌دهد. شبیه‌سازی که در این فصل تشریح خواهد شد توسط نیروی هوایی آمریکا تهیه شده است. کاربران آن خلبانان و کارکنان هواپیماهای خاصی هستند که در حال آزمایش می‌باشند. از شبیه‌سازهای پرواز برای آموزش خلبانان در عملیات هوایی و کارکنان در عملیات با اسلحه‌های مختلف استفاده می‌شود. بعضی از شبیه‌سازها تنها برای یک کاربر طراحی شده است در حال که بیشتر آنها برای آموزش چندین کاربر به طور همزمان جهت ماموریت‌های مشترک طراحی شده‌اند.

شبیه‌سازهای پرواز به وسیله شرکت‌های مختلف ایجاد می‌شوند که از طریق فرآیند مناقصه انتخاب شده‌اند. سیستمهای شبیه‌ساز، بزرگ (در حدود ۱.۵ میلیون خط کد)، دارای طول عمر بالا (هواپیماهای شبیه‌سازی شده حداقل ۴۰ سال طول عمر دارند)، به شدت بلادرنگ و نیازمند دقت بالایی هستند (هواپیمای شبیه‌سازی شده باید دقیقا شبیه یک هواپیما واقعی در موقعیت‌های مختلف عمل کند. به عنوان مثال، در حالت پرواز عادی، مانورهای هوایی یا در حالتی که نقصی در تجهیزات وجود دارد).

[1] Transparency
[2] ABC for Structural-Model-based flight simulators

با توجه به مطالب ذکر شده، به صورت خلاصه عناصر چرخه حرفه معماری شبیه‌ساز پرواز به شرح زیر هستند:

- **کاربران**، خلبانان
- **مشتری**، نیروی هوایی امریکا
- **توسعه‌دهنده**، شرکت‌های مختلف
- **محیط فنی**، طرح شبیه‌سازهای موجود
- **تجربه معمار**، طراحی شیءگرا و طراحی سیستمهای بلادرنگ



شکل ۸-۱: چرخه حرفه معماری شبیه‌ساز پرواز

زمان ایجاد الگوی مدل ساختاری به سال ۱۹۸۷ بر می‌گردد که در آن موقع نیروی هوایی شروع به تحقیق پیرامون کاربرد تکنیک‌های طراحی شیءگرا کرد. شبیه‌سازهای پرواز الکترونیکی از سال ۱۹۶۰ وجود دارند و محرک تحقیق پیرامون طراحی شیءگرا نیز مشکلات مربوط به طراحی شبیه‌سازهای از قبل موجود بود. این محرک‌ها دربرگیرنده مشکلات ساخت (فاز استقرار فرآیند توسعه با بزرگتر و پیچیده شدن سیستم به صورت نمایی افزایش پیدا می‌کرد) و مشکلات مربوط به چرخه حیات بودند (هزینه بعضی تغییرات در سیستم برابر با هزینه ساخت کل سیستم بود).

۸-۳- نیازمندیها و خصوصیات کیفی

شبیه‌ساز آموزش پرواز دربرگیرنده سه نقش اصلی است که به شرح زیر هستند:

۱. *آموزش خلبانان و کارکنان*، خلبانان در داخل یک سکوی متحرک قرار می‌گیرند که اطراف آنها با تجهیزات واقعی مربوط به هواپیما احاطه شده است و یک صفحه نمایش بزرگ در جلوی آنها وجود دارد که چیزهای بیرونی قابل مشاهده در هنگام پرواز واقعی را شبیه‌سازی کرده و نمایش می‌دهد. هدف از یک شبیه‌ساز پرواز آموزش خلبانان و کارکنان این است تا آموزش ببینند که چگونه خلبانان هواپیما را کنترل کنند، چگونه مانورهای هوایی را انجام دهند (مانند سوخت‌گیری هوایی) و چگونه در موقعیت‌های خاص مثل حمله عکس العمل نشان دهند.

۲. *شبیه‌ساز محیط پرواز*، معمولا محیط یک مدل کامپیوتری است که بر حسب نوع تمرین یا آموزش خلبانان یا کارکنان تغییر پیدا می‌کند. محیط دربرگیرنده وضعیت هوا، تهدیدات، اسلحه‌ها و غیره است. برای مثال اگر هدف از آموزش، تمرین فرآیند سوخت‌گیری هوایی است بنابراین شبیه‌ساز در جریان هوا تلاطم ایجاد می‌کند تا فرآیند سوخت‌گیری واقعی در هوا را شبیه‌سازی کند.

۳. *شبیه‌ساز تعلیم‌دهنده پرواز (تعلیم‌دهنده آموزش)*، معمولا یک تمرین آموزشی هدف و ویژگیهای خاص خود را دارد. در طول تمرین، تعلیم‌دهنده مسئولیت نظارت بر نحوه کارایی و عملکرد خلبانان یا کارکنان را بر عهده داشته و باید شرایط محیطی خاصی که برای آن تمرین در نظر گرفته شده است را برای شبیه‌ساز تعریف می‌کند. گاهی اوقات این شرایط محیطی به یک نوشته[1] تبدیل می‌شود و برای سایر دانش آموختگان فقط در سیستم بارگذاری می‌شود. شرایط محیطی، دربرگیرنده عملکرد نامناسب تجهیزات، مورد حمله قرار گرفتن از سمت دشمن، مشکلات آب و هوایی از قبیل تلاطم‌های هوایی ایجاد شده به وسیله طوفانها و غیره هستند. تعلیم‌دهنده، کنسول جداگانه‌ای برای نظارت بر نحوه عملکرد خلبانان و کارکنان دارد، همچنین تعلیم‌دهنده از طریق این کنسول می‌تواند عملکردهای نامناسب را به شبیه ساز وارد کرده و محیط را کنترل کند.

شکل ۸-۲ یک مجموعه از شبیه‌سازهای پرواز امروزی را نشان می‌دهد.

[1] Script

شکل ۸-۲-: شبیه‌سازهای پرواز امروزی

۸-۳-۱- استفاده از مدلها

مدلهای استفاده شده در هواپیما و محیط قادر هستند که با هر دقتی شبیه‌سازی شوند. به عنوان یک مثال از متغییر بودن دقت، مدلسازی فشار هوا را در نظر بگیرید. یک مدل ساده آن است که فشار هوا فقط به وسیله ارتفاع تحت تاثیر قرار بگیرد و یک حالت پیچیده‌تر نیز آن است که فشار هوا تحت تاثیر ارتفاع و الگوهای آب و هوای محلی[1] باشد. مدلسازی الگوهای آب و هوا قدرت محاسباتی زیادی را می‌طلبد اما این اجازه را می‌دهد که در مورد آب و هوا پیش‌بینی‌هایی را انجام داد. یک مدل پیچیده‌تر از حالت قبلی نیز وجود دارد که در آن فشار هوا تحت تاثیر ارتفاع، الگوهای آب و هوای محلی و رفتار هواپیماهای است که در مجاور آن پرواز می‌کند (یکی از آشفتگیها در وضعیت هوا می‌تواند ناشی از هواپیماهایی باشند که اخیرا از مسیری هوایی عبور کرده‌اند).

از آنجا که بخش آموزش خلبانان و کارکنان هواپیما یکی از مهمترین بخشهای یک شبیه‌ساز پرواز است همواره تمرکز شدیدی بر روی دقت و کیفیت آن وجود دارد زیرا هر چقدر دقت و متغییر بودن فرآیند شبیه‌سازی بیشتر باشد مهارت‌های خلبانان و کارکنان هواپیما نیز افزایش خواهد یافت. بنابراین کارایی یکی از مهمترین نیازمندیهای کیفی برای شبیه‌ساز پرواز است.

[1] Local weather patterns

۸-۳-۲-حالات اجرا

شبیه‌ساز پرواز در چندین حالت می‌تواند اجرا شود که عبارتند از :

- *حالت عملیات*، اجرا عملکردهای معمولی شبیه‌ساز به عنوان یک ابزار آموزشی
- *حالت پیکربندی*، هنگامی که قرار است تغییراتی به جلسه آموزشی جاری[1] اعمال شود شبیه‌ساز به عنوان یک پیکربند استفاده می‌شود. برای مثال، یک خلبان در حال آموزش را در نظر بگیرید که تعلیم‌دهنده آموزش می‌خواهد آن را به حالت سوخت‌گیری در حال پرواز تغییر حالت دهد در این صورت شبیه‌ساز در حالت پیکربندی قرار می‌گیرد.
- *توقف*، شبیه‌سازی در حال اجرا را متوقف می‌کند.
- *پخش مجدد*، برای اجرای شبیه‌سازی بدون تعامل با خلبان استفاده می‌شود. به عنوان مثال، خلبان یک فرآیند آموزشی را انجام داده است و پخش مجدد، همان پرواز را دوباره و آن هم بدون نیاز به خلبان اجرا می‌کند تا خلبان با دیدن دوباره پرواز، اشتباهات خود را مشاهده کند و گاها در همین فرآیند می‌تواند کنترل را دوباره در دست گرفته و عکس العمل جدیدی را از خود نشان دهد (بدین منظور می توان از تاکتیک ضبط و پخش مجدد معرفی شده در فصل پنج استفاده کرد).

شبیه‌سازی که در این فصل تشریح خواهد شد دارای یک سری ویژگیهایی است که این ویژگیها به شرح زیر هستند:

۱. *محدودیت‌های کارایی بلادرنگ*[2]

شبیه‌سازهای هواپیما باید در یک نرخ نمایش ثابت[3] اجرا شوند تا اینکه بتوانند دقت لازم را تضمین کنند. نرخ نمایش یک تصویر لحظه‌ای در زمان است. زمانی که یک تعداد مناسب از نمایش‌ها در یک بازه زمانی مشخص در نظر گرفته شوند یک کاربر می‌تواند یک حرکت پیوسته را مشاهده کند. حالت‌های مختلف نمایش حرکت، نیازمند نرخ نمایش متفاوتی است. معمولا نرخ نمایش شبیه‌سازها بین ۳۰ تا ۶۰ هرتز است که در داخل هر نرخ نمایش باید کلیه عملیات اجرا شده و کامل شوند.

[1] Current training session
[2] Real-time performance constraints
[3] Fixed frame rates

۲. *توسعه و تغییرات مداوم*

شبیه‌ساز آموزش خلبانان و کارکنان به این دلیل وجود دارد که آموزش بر روی وسیله واقعی بسیار پرهزینه و خطرناک است. به منظور فراهم کردن یک تجربه پرواز واقعی، شبیه‌ساز پرواز باید مانند یک وسیله پرواز واقعی عمل کند. همچنین هواپیماهای نظامی و شخصی، پیوسته در حال تغییر و به‌روزرسانی هستند، بنابراین نرم‌افزار شبیه‌سازی نیز به طور مداوم در حال تغییر و به روزرسانی است. علاوه بر اینها، شبیه‌سازها به طور پیوسته در حال گسترش هستند تا از موقعیت‌های جدید منجمله مشکلاتی (عملکرد نامناسب) که ممکن است برای هواپیما اتفاق بیفتد (مانند حالتی که بالگردهای نظامی در درون شهر پرواز می‌کنند) حمایت کنند.

۳. *اندازه بزرگ و پیچیدگی زیاد*

شبیه‌سازهای پرواز معمولا از ده هزاران خط کد منبع برای ایجاد شبیه‌سازهای آموزشی ساده و میلیونها خط کد منبع برای شبیه‌سازهای آموزشی پیچیده و چند نفره تشکیل شده‌اند.

۴. *توسعه در مکانهای مختلف*

شبیه‌سازهای پرواز نظامی به دو دلیل به صورت توزیع شده توسعه می‌یابند، یکی به دلیل تکنیکی و دیگری به دلیل سیاسی است. از نظر تکنیکی چون بخشهای مختلف توسعه نیازمند خبرگی خاص هستند بنابراین، راهبرد شرکت اصلی توسعه‌دهنده آن است که از شرکت‌های متخصص مختلف در هر حوزه خاص استفاده کند. از نظر سیاسی، چون کارهایی که در آن از تکنولوژی بالایی استفاده می‌شود (از جمله توسعه شبیه‌ساز) موقعیت سیاسی خوبی را ایجاد می‌کنند، بنابراین کلیه سیاستمداران سعی می‌کنند که بخشی از کار را در حوزه خود داشته باشند.

علاوه بر اینها، دو مشکل اصلی دیگر پیرامون شبیه‌سازهای پرواز قدیمی وجود داشت که باعث شد نیروی هوایی به دنبال طراحی جدید برای شبیه‌سازهای خود باشد این مشکلات عبارتند از :

۱. *پرهزینه بودن تغییر، آزمایش و اشکالزدایی*

پیچیدگی نرم‌افزار شبیه‌سازی پرواز، ماهیت بلادرنگ بودن آن و گرایش برای تغییرات مداوم در کل باعث می‌شود که هزینه آزمایش، یکپارچه‌سازی و تغییر نرم‌افزار از هزینه توسعه آن فراتر رود.

همچنین یکی از پیامدهای ناشی از افزایش پیچیدگی بالا رفتن هزینه یکپارچه‌سازی است، لذا یکپارچه‌سازی به عنوان یک پیشران معماری برای این سیستم تلقی می‌شود.

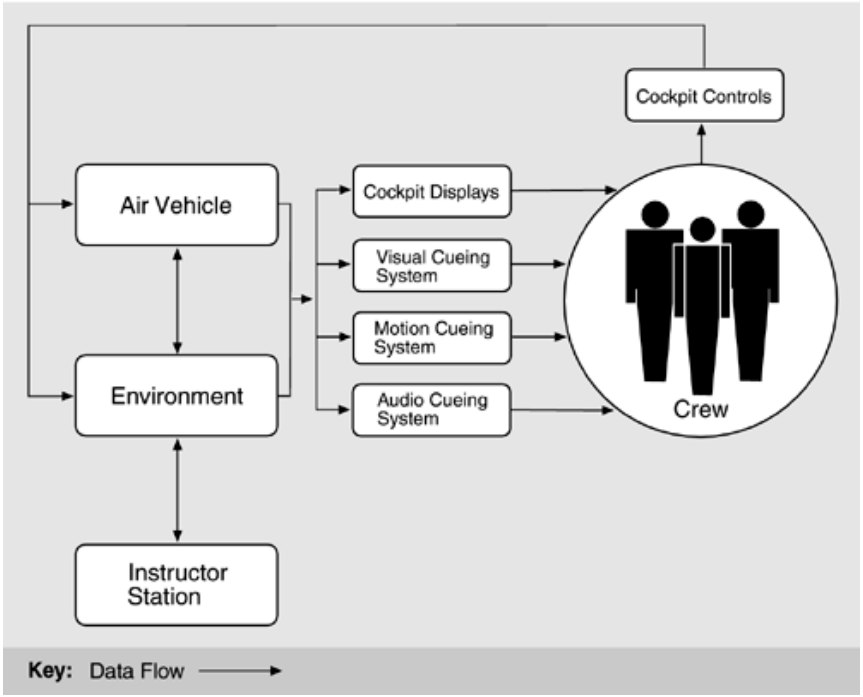
۲. *عدم وجود نگاشت روشن بین ساختار نرم‌افزار و ساختار هواپیما*

طراحی سنتی شبیه‌سازهای پرواز مبتنی بر پیروی از حلقه‌های کنترل در یک چرخه است. برای مثال فرض کنید که خلبان، هواپیما را به سمت راست هدایت می‌کند این عمل ایرودینامیک را تحریک کرده و باعث چرخش هواپیما به سمت راست می‌شود. در این حالت یک مدل، ارتباط بین کنترل کننده‌ها، سطوح، ایرودینامیک و جهت‌یابی هواپیما را بازتاب می‌کند. در معماری اصلی شبیه‌ساز، عملکرد تغییر جهت به ماژولی به نام "جهت" تخصیص داده شده است و به همین ترتیب سایر عملکردهای درگیر این تغییر جهت به ماژولهای خاصی اختصاص یافته‌اند (مانند ماژول جهت‌یابی، ماژول ایرودینامیک و غیره).

بدین منظور راهبرد پایه بررسی فعالیت‌های خلبان، مدلسازی مولفه‌های مربوط به هر فعالیت و نگاه‌داشتن کلیه محاسبات مربوط به هر وظیفه به صورت محلی است. راهبرد مطرح شده بیشترین کارایی را دارد زیرا هر وظیفه در یک ماژول مجزا مدلسازی می‌شود و بنابراین کمترین انتقال داده برای انجام محاسبات اتفاق می‌افتد. مشکل مربوط با این معماری آن است که مولفه‌های فیزیکی یکسانی در چندین مدل و لذا چندین ماژول ارائه می‌شوند. قرار گرفتن این نوع مولفه‌ها باعث افزایش ارتباطاتی می‌شوند که برای قابلیت یکپارچه‌سازی و تغییرپذیری مشکل‌ساز هستند.

۸-۴- راه حل معمارانه

شکل ۸-۳ یک مدل مرجع برای شبیه‌ساز پرواز را نشان می‌دهد. سه نقش اصلی (وسیله هوایی، محیط و تعلیم‌دهنده) که برای شبیه‌ساز پرواز در بخش قبلی مشخص شدند تعاملات بین کارکنان و سیستمهای مختلف آموزش را نشان می‌دهند. معمولا تعلیم‌دهنده در یک سکوی سخت‌افزاری متفاوت از مدل وسیله هوایی قرار دارد. مدل محیط نیز می‌تواند در یک سکوی سخت‌افزاری متفاوت یا اینکه در همان سکوی سخت‌افزاری تعلیم‌دهنده قرار داده شود.



شکل ۸-۳: مدل مرجع برای شبیه‌ساز پرواز

تقسیم‌بندی منطقی بین ایستگاه تعلیم‌دهنده و دیگر دو بخش شبیه‌ساز کاملا آشکار است. ایستگاه تعلیم‌دهنده دربرگیرنده بخش کنترل کننده مربوط به تعلیم‌دهنده و ناظر کنش‌های کارکنان است. دو بخش دیگر نیز عمل شبیه‌سازی را انجام می‌دهند. تقسیم‌بندی بین وسیله هوایی و محیط کاملا واضح نیست. برای مثال اگر یک وسیله هوایی از یک اسلحه استفاده کند منطقا تا زمانی که اسلحه از وسیله هوایی جدا نشده است بخشی از وسیله هوایی به شمار می‌رود در حالی که در آن لحظه جزئی از محیط نیز هست. به محض اینکه عمل شلیک انجام شود ایرودینامیک اسلحه تحت تاثیر هواپیما قرار می‌گیرد، بنابراین هر مدلسازی از ایرودینامیک باید به صورت منسجم به وسیله هوایی وابسته باشد. اگر اسلحه همیشه بخشی از محیط در نظر گرفته شود مدلسازی نیازمند ارتباط منسجم بین وسیله هوایی و محیط است. اگر اسلحه به عنوان بخشی از وسیله هوایی مدل شود و زمانی که شلیک شد به محیط بازگردانده شود کنترل اسلحه می‌بایستی میان این دو رد و بدل شود.

۸-۴-۱- رفتار زمان در شبیه‌ساز پرواز

همان طور که در فصل پنجم نیز ذکر شد مدیریت منابع یکی از تاکتیک‌های دستیابی به کارایی است. در یک شبیه‌ساز بلادرنگ، مهمترین منبع برای مدیریت، خود زمان است. شبیه‌ساز پرواز بازتاب کننده دنیای واقعی

است. شبیه‌ساز این عمل را از طریق ایجاد رفتارهای مبتنی بر زمان دنیای واقعی انجام می‌دهد. بنابراین وقتی خلبان در شبیه‌ساز یک سوئیچ را فعال می‌کند، شبیه‌ساز باید پاسخ مناسب را در همان زمان و مشابه پاسخ دنیای واقعی فراهم کند (پاسخ داده شده نباید خیلی سریع و یا خیلی کند باشد زیرا هر دو حالت باعث می‌شود شبیه‌ساز مانند دنیای واقعی عمل نکند).

دو روش کاملا متفاوت برای مدیریت زمان در شبیه‌ساز پرواز وجود دارد که یکی مدیریت دوره‌ای و دیگری مدیریت مبتنی بر رویداد است که هر دو اینها در شبیه‌ساز استفاده می‌شود. مدیریت زمان دوره‌ای در بخشهایی که باید کارایی بلادرنگ حفظ شود (مانند وسیله هوایی) به کار گرفته می‌شود و مدیریت زمان مبتنی بر رویداد نیز در بخش‌هایی که مدیریت بلادرنگ زیاد مهم نیست (مانند ایستگاه مربی) استفاده می‌شود.

۸-۴-۱-۱- مدیریت زمان دوره‌ای

برنامه مدیریت زمان دوره‌ای یک برش زمانی ثابت مبتنی بر نرخ نمایش دارد که پایه زمانبندی فرآیندهای سیستم است. این برنامه معمولا از روش زمانبندی چرخشی غیرانحصاری استفاده می‌کند که به وسیله تکرار رویه‌های زیر انجام می‌شود:

- زمان شبیه‌سازی شده اولیه را مقداردهی می‌کند.
- دو مرحله بعدی را تا زمانی که جلسه کامل شود، ادامه می‌دهد

۱. هر فرآیند برای یک برش زمانی ثابت فراخوانی می‌شود. هر فرآیند وضعیت داخلی خود را مبتنی بر زمان جاری شبیه‌ساز محاسبه کرده و آن را گزارش می‌کند. این عمل تضمین می‌کند که عملیات فرآیند، در برش زمانی بلادرنگ تمام خواهد شد.
۲. به زمان شبیه‌سازی شده به میزان یک برش اضافه می‌شود.

شبیه‌ساز مبتنی بر مدیریت دوره‌ای زمان، قادر است که زمان شبیه‌سازی شده و زمان واقعی را همگام کند. این همگام‌سازی زمانی انجام می‌شود که هر فرآیند بتواند حالت خود را برای دوره زمانی بعدی در داخل برش زمانی تخصیص داده شده به آن گسترش دهد. معمولا این عمل به وسیله سازگار کردن مسئولیت‌های فرآیندهای منفرد انجام می‌شود (مسئولیت‌ها به اندازه کافی کوچک هستند تا در برش زمانی اختصاص داده شده

محاسبه شوند). همچنین اینکه تعیین تعداد پردازنده‌های مورد نیاز برای تضمین توان محاسباتی در جهت انجام کلیه فرآیندها در زمان برش تعیین شده بر عهده طراح است.

۸-۴-۱-۲- مدیریت زمان مبتنی بر رویداد

برنامه مدیریت زمان مبتنی بر رویداد مشابه زمانبندی مبتنی بر وقفه در بسیاری از سیستم عامل‌ها است. این زمانبندی از طریق تکرار حلقه گام‌های زیر انجام می‌شود:

- یک رویداد شبیه‌سازی شده به صف رویداد اضافه می‌شود.
- در حالی که رویدادی در صف رویدادها وجود دارد
  - رویداد با کوچکترین زمان شبیه‌سازی شده را انتخاب کن
  - زمان شبیه‌سازی شده را برابر با زمان رویداد انتخاب شده قرار بده
  - فرآیند مربوط به رویداد جاری را فراخوانی کنید. این فرآیند ممکن است رویدادی را به صف رویدادها اضافه نماید.

در این حالت زمان شبیه‌سازی به وسیله فرآیندهای تولید کننده رویداد پیش می‌رود و شبیه‌ساز فقط رویداد بعدی را برای پردازش انتخاب می‌کند. در شبیه‌سازهای کاملا مبتنی بر رویداد، زمان شبیه‌سازی شده ممکن است خیلی سریع (مانند شبیه‌ساز بازی جنگی) یا کند (شبیه‌ساز مهندسی) از زمان واقعی پیش برود.

۸-۴-۱-۳- سیستمهای مدیریت زمان ترکیبی

ایستگاه تعلیم‌دهنده معمولا مبتنی بر رویداد است و وسیله هوایی نیز به صورت دوره‌ای زمانبندی می‌شود. مدل محیط نیز می‌تواند یکی از دو حالت دوره‌ای یا رویدادی باشد. بنابراین وابستگی بین وسیله هوایی و محیط می‌تواند منجر به نوع متفاوتی از زمانبندی شود. شبیه‌سازهای پرواز باید شبیه‌سازی زمان دوره‌ای (از قبیل مدل وسیله هوایی) را با شبیه‌سازی مبتنی بر رویداد (از قبیل مدل محیط آن هم در بعضی حالات) و همچنین سایر فعالیت‌های مبتنی بر رویداد که دوره‌ای نیستند (از قبیل ارتباط با ایستگاه تعلیم‌دهنده یا تنظیم کردن یک سوئیچ توسط خلبان)، ترکیب کند. سیاست‌های زمانبندی زیادی از دیدگاه هر فرآیند می‌تواند از ترکیب این نوع زمانبندی‌ها حاصل شود.

یک سیاست ساده برای مدیریت رویدادها در داخل پردازنده زمانبندی شده به صورت دوره‌ای آن است که پردازش دوره‌ای بلافاصله بعد از مرحله همگام‌سازی آغاز و قبل از هر پردازش غیردوره‌ای کامل شود. پردازش غیردوره‌ای در داخل بازه‌های محدود انجام می‌شود که در طول آن تعداد زیادی از پیام‌ها می‌توانند دریافت و پردازش شوند. پیام‌هایی که در طول بازه مشخص پردازش نمی‌شوند باید به فرجه زمانی بعدی انتقال داده شوند. ارتباط بین بخشهایی از سیستم که مبتنی بر رویداد زمانبندی شده است با بخشهایی که به صورت دوره‌ای زمانبندی شده‌اند به صورت غیردوره‌ای است. همچنین ارتباط بین بخشهایی که به صورت دوره‌ای زمانبندی شده‌اند با بخشهایی که به صورت رویداد هستند مبتنی بر رویداد است. با توجه به این پیچیدگی موجود در زمانبندی سیستمهای شبیه‌ساز پرواز، در ادامه الگوی معماری پیشنهاد خواهد شد که این مشکلات را حل می‌کند.

۸-۴-۲- الگوی معماری مدل ساختاری

مدل ساختاری متشکل از مجموعه‌ای از انواع عناصر و پیکربندی برای هماهنگی آنها در زمان اجرا است. همان طور که قبلا ذکر شد مدل وسیله هوایی می‌تواند بر روی چندین پردازنده قرار بگیرد. بنابراین عناصر مدل ساختاری وسیله هوایی باید به صورت داخلی در میان پردازنده، مدل محیط و بخشهای تعلیم‌دهنده شبیه‌ساز که بر روی پردازنده‌های مختلف هستند، هماهنگ شوند. الگوی معماری مدل ساختاری در بالاترین سطح از بخش‌های زیر تشکیل شده است:

- *اجرایی*[1]

بخش اجرایی، پیامدهای مربوط به هماهنگی را مدیریت می‌کند. این پیامدها عبارتند از: زمانبندی بلادرنگ زیرسیستمها، همگام‌سازی بین پردازنده‌ها، مدیریت رویداد از سمت ایستگاه تعلیم‌دهنده و کاربر، اشتراک‌گذاری و یکپارچه‌سازی داده.

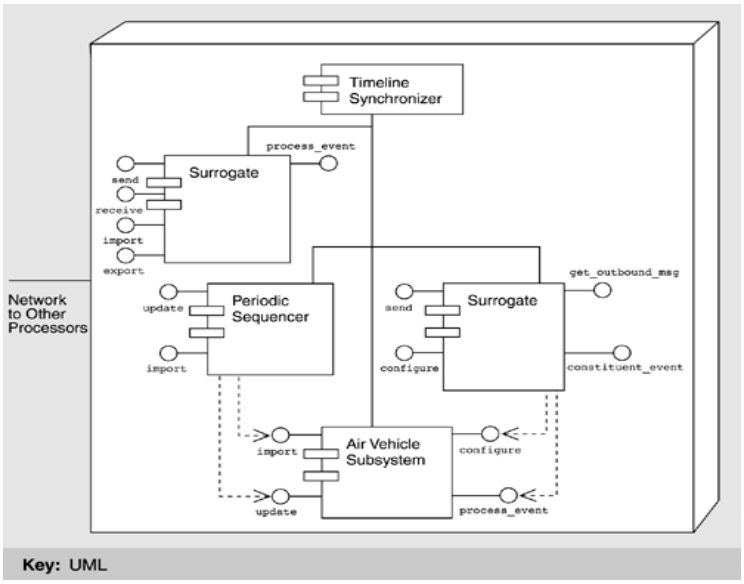
- *کاربرد*[2]

بخش کاربرد، محاسبات شبیه‌ساز از قبیل مدلسازی وسیله هوایی را کنترل می‌کند. عملکردهای کاربرد نیز به وسیله زیرسیستمها و فرزندانشان پیاده‌سازی می‌شوند.

[1] Executive
[2] Application

۸-۴-۲-۱- مدلهای اجرایی وسیله هوایی

شکل ۸-۴ مدل ساختاری وسیله هوایی با جزئیات مربوط به الگوی اجرایی را نشان می‌دهد. ماژولها در بخش اجرایی، همگام‌کننده خط زمان[1]، مرتب‌کننده دوره‌ای[2]، کنترل کننده رویداد[3] و جانشینها[4] (برای دیگر بخشهای شبیه‌ساز) هستند.



شکل ۸-۴: الگوی مدل ساختاری یک پردازنده سیستم وسیله هوایی با تمرکز بر روی بخش اجرایی

همگام‌کننده خط زمان:

همگام‌کننده خط زمان، مکانیزم اصلی زمانبندی برای مدل وسیله هوایی و نگهدارنده زمان داخلی و وضعیت جاری شبیه‌ساز است. همگام‌کننده خط زمان، داده و کنترل را به سه عنصر دیگر منتقل کرده و از آنها داده و کنترل را دریافت می‌کند. همچنین زمان واقعی را با بخشهای دیگر شبیه‌ساز هماهنگ می‌کند. نهایتا اینکه همگام‌کننده خط زمان، یک سیاست زمانبندی برای هماهنگی پردازشهای دوره‌ای و غیردوره‌ای پیاده‌سازی می‌کند.

[1] Timeline Synchronizer
[2] Periodic Sequencer
[3] Event Handler
[4] Surrogates

مرتب‌کننده دوره‌ای:

مرتب‌کننده دوره‌ای به منظور هدایت کردن کلیه پردازش‌های انجام شده به وسیله زیرسیستمهای شبیه‌ساز استفاده می‌شود. این عمل شامل فراخوانی زیرسیستمها به منظور انجام عملیات دوره‌ای مبتنی بر زمانبندی ثابت است. مرتب‌کننده دوره‌ای دو عملیات را برای هماهنگ‌کننده خط زمان فراهم می‌کند. یکی درخواست عملیات import است که در آن مرتب‌کننده دوره‌ای زیرسیستمها را فراخوانی می‌کند تا عملیات ورود خودشان را انجام دهند، دیگری نیز درخواست‌های عملیات update است که در آن مرتب‌کننده دوره‌ای از زیرسیستمها درخواست می‌کند که عملیات "به روزرسانی" را انجام دهند.

مرتب‌کننده دوره‌ای برای اینکه پردازش خودش را هدایت کند به دو قابلیت نیاز دارد. اولین قابلیت آن است که بتواند دانش مربوط به یک زمانبند را سازماندهی کند. دومین قابلیت نیز فراخوانی واقعی زیرسیستمها از طریق عملیات دوره‌ای و آن هم از طریق بعضی مکانیزمهای توزیع[1] است.

کنترل کننده رویداد[2]:

کنترل کننده رویداد، کلیه پردازش‌های غیر دوره‌ای انجام شده به وسیله زیرسیستمها را هماهنگ می‌کند. این عمل به وسیله درخواست عملیات غیر دوره‌ای متعلق به کنترل کننده دوره‌ای انجام می‌شود. کنترل کننده رویداد چهار عملیات را برای کنترل کننده خط زمان فراهم می‌کند که عبارتند از:

- configure، برای شروع یک ماموریت جدید آموزشی استفاده می‌شود

- constituent_event، برای زمانی استفاده می‌شود که یک رویداد برای نمونه خاصی از یک ماژول در نظر گرفته شده است.

- get_outbound_msg، به وسیله هماهنگ‌کننده خط زمان برای هدایت پردازش‌های غیردوره‌ای، آن هم در حالت‌های عملیاتی سیستم از قبیل operate (که غالبا دوره‌ای هستند) استفاده می‌شود.

- send، به وسیله زیرسیستمها برای ارسال رویدادها به دیگر کنترل کنندگان زیرسیستمها و همچنین ارسال پیام به دیگر سیستمها استفاده می‌شود.

[1] Dispatching
[2] Error handler

جانشین:

جانشین‌ها یک کاربرد از تاکتیک "استفاده از میانجی" هستند که مسئول ارتباطات سیستم-به-سیستم بین مدل وسیله هوایی و مدل محیط یا ایستگاه تعلیم‌دهنده هستند. جانشین‌ها از جزئیات فیزیکی سیستم که به وسیله آن ارتباطات برقرار می‌شوند، آگاه هستند. برای مثال، ایستگاه تعلیم‌دهنده بر وضعیت داده در مدل وسیله هوایی نظارت داشته و آن را به تعلیم‌دهنده نشان می‌دهد. در واقع جانشین، اطلاعات صحیح را زمانی که کنترل را از پردازنده می‌گیرند جمع‌آوری کرده و آن را به ایستگاه تعلیم‌دهنده ارسال می‌کند. از طرف دیگر تعلیم‌دهنده ممکن است بخواهد یک حالت خاص را برای خلبان در داخل شبیه‌ساز تنظیم کند، پس این یک رویداد خواهد بود که به وسیله جانشین دریافت شده و به پردازنده رویداد برای توزیع به زیرسیستمهای مناسب انتقال داده می‌شود.

استفاده از جانشین‌ها روشی است که در آن هر دو کنترل کننده رویداد و زمانبند دوره‌ای می‌توانند از جزئیات ایستگاه تعلیم‌دهنده یا سکویی که مدل محیط در آن در حال عملیات است، مخفی نگاه داشته شود. تمام دانش مرتبط با سیستم در جانشین‌ها جاسازی شده‌اند. بنابراین وقتی تغییری در این سکوها اتفاق بیفتد بیشترین انتشار تغییرات فقط در حد یک جانشین در مدل وسیله هوایی خواهد بود.

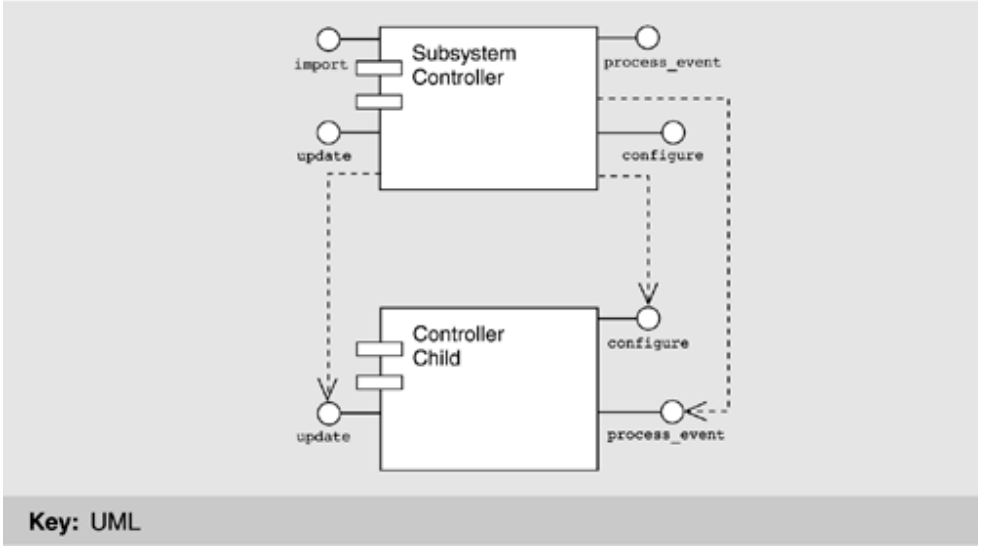
۸-۴-۲-۲- مدلهای کاربرد وسیله هوایی

شکل ۸-۵ انواع ماژولهایی را نشان می‌دهد که در زیر بخش کاربرد مدل ساختاری وسیله هوایی وجود دارد. برای وسیله هوایی فقط دو ماژول کنترل به نامهای کنترل کننده زیرسیستم[1] و کنترل کننده فرزند[2] وجود دارد. کنترل کنندگان زیرسیستمها، داده‌ها را به دیگر کنترل کنندگان زیرسیستمها و همچنین فرزندان خودش انتقال می‌دهند. آنها همچنین کنترل را از والدین خود دریافت و به والدین خود باز می‌گردانند. این محدودیت‌ها بر روی انتقال داده و کنترل، باعث ایجاد یک کنترل کننده فرزند می‌شود.

استدلال پیرامون این موضوع را می‌توان بدین صورت بیان کرد که هدف از این محدودیت‌ها کمک کردن به تغییرپذیری و یکپارچه‌سازی از طریق حذف وابستگی‌های موجود بین نمونه فرزند با دیگر ماژولها است. در

[1] Subsystem controller
[2] Controller child

واقع فقط فرزند با ماژول پدر خود در ارتباط خواهد بود (این یک حالتی است که در آن از تاکتیک "ارتباطات محدود" استفاده شده است).



شکل ۸-۵: انواع ماژولهای کاربرد

کنترل کننده زیرسیستم:

کنترل کنندگان زیرسیستمها به منظور متصل کردن یک مجموعه از عملکردهای مرتبط به فرزندان استفاده می‌شوند تا اینکه دو فعالیت زیر را انجام دهند:

- دستیابی به شبیه‌سازی زیرسیستم به صورت کلی
- کنترل غیرمستقیم و ارتباطات غیردوره‌ای بین سیستم و زیرسیستمها

کنترل کنندگان زیرسیستمها همچنین مسئول مشخص کردن این موضوع هستند که چگونه از قابلیت فرزندان خودشان استفاده کنند تا اینکه عملکردهای خاص تعلیم‌دهنده از قبیل عملکردهای نامتعارف و تنظیمات مربوط به پارامترها را برآورده کنند.

به دلیل اینکه الگوی مدل ساختاری، ارتباطات بین کنترل کنندگان فرزندان را محدود می‌کند، کنترل کننده زیرسیستم باید قابلیتی را فراهم کند تا ارتباط منطقی بین فرزندان خودش و فرزندان متعلق به سایر زیرسیستمها ایجاد شود. ارتباطات درونی[1] ورودی خروجی‌های تولید شده زیرسیستم را فراهم می‌کنند که برای الگوریتمهای

[1] Inbound

شبیه‌سازی فرزندان زیرسیستم موردنیاز هستند. ارتباطات خارجی[1] نیز یک چنین نیازهایی را ولی این بار برای فرزندان سایر زیرسیستمها و جانشین‌ها فراهم می‌کنند. این نوع ارتباطات به عنوان مجموعه‌ای از نام‌ها در نظر گرفته می‌شوند که به وسیله آنها کنترل کننده سیستم به صورت داخلی به داده‌های در نظر گرفته شده برای خروجی خود ارجاع می‌کند. وقتی یک چنین نامی خوانده یا نوشته می‌شود ارتباطات مناسب فرض می‌شوند که وجود دارد. اینکه واقعا چگونه این ارتباطات ایجاد می‌شوند زمانی مشخص می‌شود که طراحی به صورت کامل انجام شود. کنترل کننده سیستم علاوه بر ایجاد ارتباط بین فرزندان خودش و فرزندان متعلق به سایر زیرسیستمها، به عنوان یک میانجی بین فرزندان خودش عمل می‌کند زیرا محدود کردن ارتباطات دلالت بر این امر دارد که فرزندان اجازه ندارند که مستقیما با یکدیگر ارتباط برقرار کنند.

همان طور که قبلا ذکر شد شبیه‌ساز پرواز می‌تواند در چندین حالت قرار بگیرد. عمل انتقال به یک حالت اجرایی خاص از طریق یک اجراکننده صورت می‌پذیرد. سپس اجرا کننده حالت جاری خودش را به کنترل کننده سیستم اعلام می‌کند.

دو حالتی که در اینجا بررسی می‌شوند حالت "عملیات"[2] و "حالت تثبیت"[3] هستند. حالت "عملیات" کنترل کننده سیستم را ایجاد می‌کند تا عملیات معمولی خودش را که در ارتباط با شبیه‌ساز است انجام دهد. حالت "تثبیت" نیز به کنترل کننده سیستم اطلاع می‌دهد که عملیات خودش را در یک روش کنترل شده به پایان برساند. این روش کنترل شده به شرح زیر است:

- بازیابی و ذخیره‌سازی محلی ارتباطات داخلی تحت کنترل مستقیم یک اجرا کننده انجام می‌شود. یک چنین قابلیتی پیامدهای حاصل از سازگاری داده و انسجام زمانی را حل می‌کند.
- تثبیت الگوریتمهای شبیه‌سازی فرزندانش تحت کنترل نمونه‌های اجرایی.

کنترل کنندگان زیرسیستم باید قادر به انجام فعالیت‌های زیر باشند:

- معرفی خودشان و هر یک از فرزندانشان به یک مجموعه شرایط اولیه در پاسخ به یک رویداد
- مسیریابی درخواست‌ها برای عملکردهای نامتعارف و تنظیم پارامترهای شبیه‌ساز مبتنی بر دانش و آگاهی از قابلیت‌های فرزند

[1] Outbound
[2] Operate state
[3] Stabilize

کنترل کنندگان زیرسیستم می توانند از پیکربندی مجدد پارامترهای حمایت کنند. کنترل کنندگان زیرسیستمها این قابلیت‌ها را از طریق عملیات دوره‌ای و غیردوره‌ای قابل دسترس برای مرتب‌کننده دوره‌ای و کنترل کننده رویداد تحقق می‌بخشند. کنترل کنندگان زیرسیستم باید از دو عملیات دوره‌ای import, updateحمایت کنند و می توانند دو عملیات دیگر به نامهای process_event و configure را نیز پشتیبانی کنند.

- Update

عملیات update باعث می‌شود که کنترل کننده زیرسیستم، پردازش غیردوره‌ای متناسب با وضعیت عملیاتی جاری سیستم که به صورت پارامتر ورودی فراهم شده را انجام دهد. در حالت "عملیات"، عملیات update باعث می‌شود که کنترل کننده زیرسیستم از طریق ارتباطات ورودی، ورودی‌های مورد نیاز برای فرزندان خودش را فراهم کند تا فرزندانش بتوانند در یک ترتیب معنایی اجرا شده و تغییرات در سراسر آنها پخش شود. این عمل باعث تولید خروجی می‌شود که مورد نیاز ارتباطات خروجی زیرسیستمهای دیگر می‌شود. در حالت "تثبیت"، عملیات update بدین منظور به کار می‌رود که از کنترل کننده زیرسیستم درخواست می شود که یک تکرار از الگوریتم تثبیت را انجام دهد و بررسی کند که آیا معیارهای تثبیت تعریف شده برآورده شده‌اند یا نه. عملیات update یک پارامتر خروجی فراهم می‌کند که از طریق آن تشخیص دهد که آیا کنترل کننده سیستم، سیستم را به عنوان یک حالت پایدار در نظر گرفته است یا نه؟

- Import

از عملیات import برای صدور درخواستی که کنترل کننده زیرسیستم، ارتباطات ورودی خاص خودش را به وسیله خواندن مقادیر و ذخیره کردن محلی مقادیر به منظور استفاده در عملیات update بعدی، بهره برده می‌شود.

- process_event

عملیات process_event در حالت‌های عملیاتی استفاده می‌شود که اساسا دوره‌ای هستند. به عنوان مثال عملیات مربوط به درخواست از کنترل کننده زیرسیستم تا به یک رویداد پاسخ دهد. رویداد به وسیله پارامتر ورودی برای عملیات فراهم می‌شود. چندین رویداد از ایستگاه تعلیم‌دهنده که در این دسته‌بندی قرار می گیرد عبارتند از : process_malfunction, set_parameter, hold_parameter

- configure

عملیات configure برای تنظیم وضعیت‌های عملیاتی سیستم (مانند مقداردهی اولیه) که اساسا غیردوره‌ای هستند، استفاده می‌شود. این عملیات برای ایجاد یک مجموعه شرایط نامگذاری شده از قبیل پیکربندی دستگاه‌های آموزشی یا ماموریت‌های آموزشی استفاده می‌شود. اطلاعاتی که کنترل کننده زیرسیستم نیاز دارد تا یک حالت را ایجاد کند می‌تواند از طریق یک پارامتر ورودی، محلی در حافظه اصلی یا ثانویه و یا پایگاه‌داده بازیابی شوند. برای کامل کردن عملیات، کنترل کننده سیستم عملیات مربوط به فرزندان خودش را فراخوانی می‌کند تا منجر به این شود که فرزندان شرایط یا همان حالت‌ها را ایجاد کنند.

کنترل کننده فرزند:

کنترل کننده فرزند مدل وسیله هوایی می‌تواند شبیه‌سازی واقعی مولفه‌های هواپیما از قبیل پمپ هیدرولیک، تقویت‌کننده الکتریکی یا مخزن بنزین باشد. به علاوه آنها می‌توانند از مدلهای خاص شبیه‌ساز از قبیل نیرو، زمان، وزن و معادلات حرکت پشتیبانی کنند. بدون توجه به اینکه چه عملکرد خاصی شبیه‌سازی می‌شود، کنترل کنندگان فرزند همگی به عنوان یک نوع ماژول در نظر گرفته می‌شوند. معمولا کنترل کنندگان فرزند از شبیه‌سازی یک بخش منفرد یا شیء در داخل بعضی عملکردهای سطح پایین پشتیبانی می‌کنند. هر فرزند یک الگوریتم شبیه‌سازی فراهم می‌کند که وضعیت خود را مبتنی بر موارد زیر مشخص می‌کند:

- حالت قبلی‌اش
- ورودی‌هایی که ارتباطش را با فرزندان منطقاً مجاور نشان می‌دهد
- فاصله‌های زمانی تخمینی

یک فرزند به محض اینکه از طرف کنترل کننده سیستم فراخوانی می‌شود تا ورودی‌های مورد نیاز را فراهم کرده و خروجی فرزندان را دریافت کند، وضعیت خود را مشخص می‌کند. به این قابلیت اصطلاحا به روزرسانی گفته می‌شود. یک فرزند توانایی تولید خروجی‌های غیرعادی را دارد تا بتواند عملکرد نادرست را گزارش دهد. علاوه بر تغییرات مدلسازی در شرایط عملیاتی عادی (از قبیل فرسودگی) که می‌تواند منجر به عملکرد نامناسب در گذر زمان شود، به وسیله کنترل کنندگان زیرسیستم می‌توان به آنها اعلام کرد که عملکرد نامناسب از خود نشان دهند.

یک کنترل کننده فرزند می‌تواند به پارامتر یک شبیه‌ساز مقدار خاصی را بدهد. پارامترهای شبیه‌ساز نامهای خارجی برای پارامترهای کارایی و معیارهای تصمیم‌گیری استفاده شده در الگوریتم شبیه‌سازی کنترل کننده فرزند هستند. هر فرزند می‌تواند خودش را به صورت بعضی حالات مشخص معرفی کند. مشابه سایر قابلیت‌های فرزند، معرفی (مقداردهی اولیه به منظور قرار گرفتن در یک حالت مشخص) و مقداردهی به پارامترها باید از جانب کنترل کننده زیرسیستم درخواست شود.

قابلیت به روزرسانی، عملکردهای نامناسب، مقداردادن به پارامترها و مقداردهی اولیه می‌تواند بر حسب چگونگی استفاده کنترل کننده زیرسیستم از فرزندان متفاوت باشد. به عنوان مثال کنترل کننده زیرسیستم می‌تواند از فرزند بخواهد که به صورت دوره‌ای عمل به روزرسانی خود را انجام دهد. کنترل کنندگان فرزندان این قابلیت‌ها را از طریق یک مجموعه از عملیات دوره‌ای یا غیردوره‌ای که برای کنترل کننده زیرسیستم قابل دسترس است، پشتیبانی می‌کنند. به عنوان مثال، update یک عملیات دوره‌ای استفاده شده برای کنترل اجرای دوره‌ای الگوریتم شبیه‌سازی است. فرزند داده‌های خارجی را دریافت کرده و خروجی خود را از طریق پارامترها تولید می‌کند. همچنین دو عملیات غیر دوره‌ای process_event و configure نیز فراهم شده توسط فرزندان هستند.

کلیه تعاملات میان فرزندان به وسیله کنترل کننده زیرسیستم انجام می‌شود که مبتنی بر این است که چگونه از عملیات فرزند استفاده شود تا اینکه به نیازهای شبیه‌ساز که به زیرسیستم تخصیص داده شده، به صورت کامل دست پیدا کرد. این عملکرد دربرگیرنده موضوعات زیر است:

- انتشار دوره‌ای تغییروضعیت فرزندان از طریق استفاده از عملیات update
- ایجاد ارتباطات منطقی بین فرزندان از طریق پارامترهای ورودی و خروجی عملیات
- ایجاد ارتباطات منطقی بین فرزندان و سایر بخشهای شبیه‌ساز با استفاده از ارتباطات داخلی و خارجی زیرسیستم

۸-۴-۳- اسکلت سیستم

تا اینجا یک چارچوب ساختاری برای شبیه‌ساز پرواز تشکیل شده است ولی هیچ یک از جزئیات آن کاملا مشخص نشده است. در واقع این چارچوب، یک چارچوب عمومی برای شبیه‌سازی است که می‌تواند برای

بالگرد و حتی راکتورهای هسته‌ای نیز استفاده شود. با توجه به مطالب ذکر شده فرآیند ایجاد یک شبیه‌ساز که بتواند بخوبی عمل کند، مستلزم تکمیل اسکلت با زیرسیستم و کنترل کنندگان فرزندان مناسب برای هر وظیفه خواهد بود. مشخص است که می‌توان یک شبیه‌ساز کامل را که متشکل از میلیون‌ها خط کد برنامه است به صورت کامل با شش نوع ماژول کنترل کننده زیرسیستم، کنترل کننده فرزند، همگام‌کننده خط زمان، مرتب‌کننده دوره‌ای، کنترل کننده رویداد و جانشینها تشریح کرد. تشریح سیستم شبیه‌سازی پرواز به وسیله این ماژول منجر به این می شود که معماری را بتوان به سادگی ایجاد، درک، یکپارچه، توسعه و تغییر داد.

۸-۴-۴- تخصیص وظیفه‌مندی به کنترل کنندگان فرزندان

با توجه به اینکه الگوی معماری که به وسیله آن مدل وسیله هوایی ساخته می‌شود، تشریح شد، ولی هنوز باید وظیفه‌مندیها به نمونه‌های ماژولها در الگو تخصیص داده شوند. برای مشخص کردن جزئیات مربوط به هواپیما که باید شبیه‌سازی شود از تعریف نمونه‌های کنترل کنندگان زیرسیستمها استفاده می‌شود. تقسیم بندی واقعی به سیستم روی هواپیما، پیچیدگی هواپیما و نوع آموزشهای طراحی شده برای شبیه‌ساز وابسته است.

در این بخش یک تقسیم‌بندی ساده تشریح شده است. بدین منظور وظیفه‌مندی مبتنی بر ساختار فیزیکی هواپیما میان کنترل کنندگان فرزندان تقسیم می‌شود. برای انجام این عمل از روش تجزیه شیءگرا استفاده می‌شود که دارای خصوصیاتی به شرح زیر است:

- ارتباط بین تقسیم‌بندی هواپیما و شبیه‌ساز را حفظ می‌کند. این امر یک مجموعه از مدلهای مفهومی را فراهم می‌کند که می‌توانند به راحتی به دنیای واقعی نگاشت شوند. فهم چگونگی اینکه بخش‌ها با یکدیگر در هواپیما ارتباط برقرار می‌کند به فهم چگونگی تعامل در شبیه‌ساز کمک می‌کند. همچنین باعث می‌شود که درک کاربر از شبیه‌ساز ساده شود زیرا آنها با هواپیما (دامنه مساله) آشنا هستند و می توانند به راحتی دانش خود را به این حوزه انتقال دهند.
- تجربه قبلی در مورد شبیه‌سازها نشان داده است که تشخیص یک تغییر در هواپیما با تقسیم‌بندی هواپیما بسیار راحت است. بنابراین محل تغییر در شبیه‌ساز مربوط به محل یکسانی در تقسیم‌بندی هواپیما است که این امر باعث محلی کردن تغییرات در شبیه‌ساز می‌شود. همچنین این امر را سهل‌تر می‌کند که بتوان به راحتی درک کرد که چگونه تغییرات در هواپیما شبیه‌ساز را تحت تاثیر قرار

می‌دهد. لذا این امر باعث می شود که به راحتی بتوان زمان و هزینه مورد نیاز برای پیاده‌سازی تغییرات را مشخص کرد.

- تعداد و اندازه واسط‌های شبیه‌ساز کاهش می‌یابد. این امر ناشی از انسجام معنایی قوی در تقسیم‌بندی است که بزرگترین واسط ها را به جای پخش کردن در میان چندین تقسیم‌بندی در داخل تقسیمات قرار می‌دهند.
- محلی کردن عملکردهای نامناسب، از طریق مرتبط کردن آنها به بخش‌های خاص تجهیزات هواپیما قابل دسترس است. تحلیل عملکردهای نامناسب از طریق سروکار داشتن با نگاشت فیزیکی بسیار راحت است.

در تقسیم کردن مساله مدلسازی وسیله هوایی به داخل واحدهای قابل مدیریت، بدنه هواپیما بخش اصلی را تشکیل می‌دهد. با توجه به تقسیم‌بندی انجام شده برای سیستم هوایی می‌توان عناصر زیر را مشخص کرد:

- *سینتیک*[1] *یا وابسته به حرکت*

  عناصری که با نیروهای وارده بر بدنه هواپیما سروکار دارند.

- *سیستمهای هواپیما*[2]

بخشی که با سیستمهای مشترک در ارتباط است و برای هواپیما توانایی‌های گوناگونی فراهم می‌کند یا انرژی را درون بدنه هواپیما توزیع می‌کند.

- *سیستمهای هوایی*[3]

اشیایی هستند که یک سری کمک‌های فرعی برای پشتیبانی از هواپیما فراهم می‌کنند اما به طور مستقیم درگیر سینتیک مدل وسیله هوایی یا عملیات پایه‌ای سیستم‌های پرواز مانند رادار نیستند.

- *محیط*

اشیاء مرتبط با محیط که مدل وسیله هوایی در داخل آنها عمل می‌کند.

۸-۴-۵- تجزیه گروه‌ها

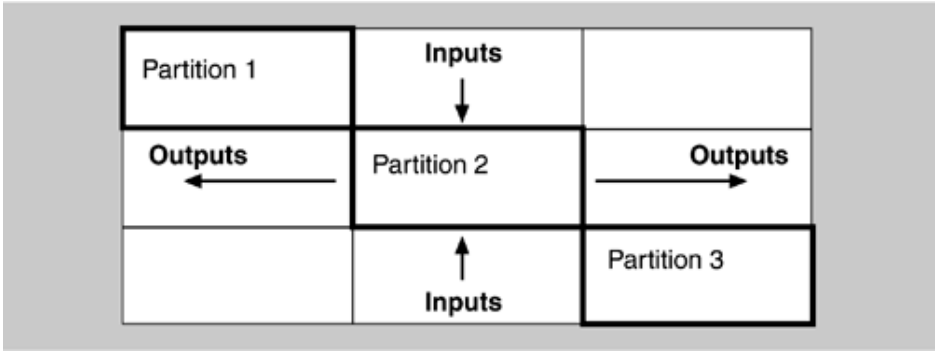
[1] Kinetics
[2] Aircraft systems
[3] Avionics

دانه درشت‌ترین عنصر تجزیه در مدل سیستم هوایی، گروه است. گروه‌ها به سیستمها تجزیه می‌شوند و سیستمها نیز به نوبه خود به زیرسیستمها تجزیه می‌شوند. زیرسیستمها، نمونه‌های کنترل کنندگان زیرسیستمها را فراهم می‌کنند. گروه‌ها و سیستمها به صورت مستقیم در معماری نمایش داده نمی‌شوند (کنترل کننده گروه وجود ندارد). آنها به منظور سازماندهی وظیفه‌مندی تخصیص داده شده به نمونه‌های مختلف کنترل کنندگان زیرسیستم استفاده می‌شوند. عمل تجزیه مربوط به گروه‌ها از طریق جداول n-square مدیریت می‌شود.

۸-۴-۵-۱- جداول n-square

یک روش برای نمایش اطلاعات مربوط به واسط‌ها در سیستم، جداول n-square است. با استفاده از این جداول می‌توان نمایش داد که چگونه قسمت‌های تقسیم شده به یکدیگر مرتبط هستند. از طرفی به دلیل اینکه بعضی فاکتورها در تصمیمات تقسیم‌بندی مبنایی برای تقسیم‌بندی واسط‌ها نیز هستند، جداول n-square برای ارزیابی این تصمیمات مفیدند. جداول یک روش خوب برای بدست آوردن ورودی‌ها و خروجی‌های ماژولها هستند و می‌توانند تجریدهای استفاده شده در بخش‌های مختلف طراحی را تشریح کنند.

یک مثال از جدول n-square در شکل ۸-۶ نشان داده شده است. سلولهای نمایش داده شده در قطر اصلی تقسیم‌بندیهای سیستم را نشان می‌دهند. ورودی‌های توسط ستونهای بالایی و پایینی هر قسمت نشان داده می‌شوند و خروجی‌ها نیز توسط سطرهای مربوط به هر قسمت نمایش داده می‌شوند. جریان داده از یک قسمت به قسمت دیگر به صورت راست، سپس پائین ، چپ و بالا است.



شکل ۸-۶: جدول n-square

شکل ۸-۷ جدول n-square مربوط به واسط‌های گروه‌های شناسایی شده برای شبیه‌ساز پرواز را نمایش می‌دهد. واسط‌های خارجی مدل وسیله هوایی برای سادگی از جدول حذف شده‌اند. عناصر داده‌ای نشان داده شده بر روی این جدول یک مجموعه تجمیعی از داده‌ها هستند که به منظور سادگی بدین صورت نشان داده شده‌اند (در جدول نوع واسط‌ها و نام آنها مشخص نشده است).

کلیه مدلهای وسیله هوایی، مربوط به ساختار هواپیما نیستند. مدلهای ایرودینامیک دیدی از اصول فیزیک هستند که در قالب تعاملات وسیله با محیط ظاهر می‌شوند. تقسیم‌بندی این حوزه متکی بر مدلهای ریاضی و موجودیت‌های فیزیکی است که پویایی وسیله را بیان می‌کنند. تقسیم‌بندی صحیح مبتنی بر مدلهای ریاضی که کل هواپیما را تحت تاثیر قرار می‌دهند بسیار مشکل‌تر از تقسیم‌بندی مبتنی بر ساختارهای فیزیکی هواپیما است.

Kinetics Group	Loads	Vehicle State Vector	Vehicle Position
Power	**Aircraft Systems Group**	Power	
Inertial State	Loads	**Avionics Group**	Ownship Emissions
Atmosphere, Terrain, and Weather Data		Environment Emitter Data	**Environment Group**

شکل ۸-۷: جدول n-square برای گروه‌های مدل وسیله هوایی

۸-۴-۶- تجزیه گروه‌ها به سیستمها

مرحله بعد از تخصیص وظیفه‌مندیها، تجزیه گروه‌ها به سیستمها است. سیستم و گروه می‌توانند به عنوان واحدهای مورد نیاز برای عمل یکپارچه‌سازی در نظر گرفته شوند (وظیفه‌مندی سیستم یک راه حل نسبتا کاملی برای یک مجموعه‌ای از مشکلات شبیه‌ساز است). سیستمها و گروه‌ها به عنوان نقاط بسیار مهم برای انجام عمل آزمایش و اعتبارسنجی تلقی می‌شوند. گروه‌های بدست آمده برای یک سیستم به صورت مجموعه‌ای از کد ماژولهای پیاده‌سازی شده به وسیله یک مهندس یا گروهی از مهندسین هستند.

۸-۴-۶-۱ سیستمها در گروه سینیتک

در این گروه عناصری هستند که مستقیما در کنترل حرکت وسیله، مدلسازی تعاملات وسیله و کنترل سطوح با محیط ارتباط دارند. سیستمهای شناسایی شده برای این گروه عبارتند از:

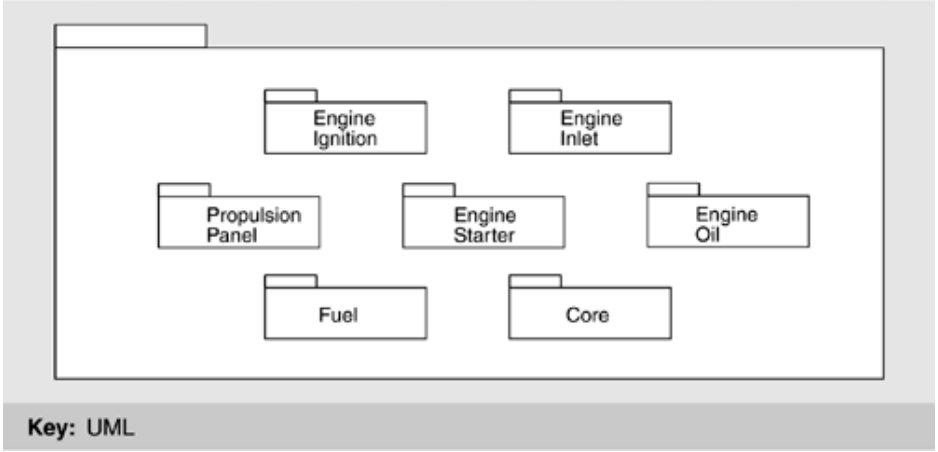
- بدنه هواپیما[1]
- نیروی محرکه[2]
- پاگردان هواپیما[3]
- کنترل کنندگان هواپیما[4]

کلیه زیرسیستمها در سیستم نیروی محرکه که در شکل ۸-۸ نشان داده شده‌اند با مدل موتور هواپیما در ارتباط هستند. چندین موتور به وسیله ایجاد چندین مجموعه از متغییرهای حالت و تکرار نمونه‌های اشیاء مناسب، کنترل می‌شوند. هدف اصلی این سیستم محاسبه فشار موتور، زمان حاصل از چرخش بخشهای موتور، نیروها و زمانهای حاصل از توزیع انبوه سوخت است. در شکل ۸-۸، سیستم سوخت هواپیما گروه‌بندی شده است زیرا واسط اصلی آن در ارتباط با موتورها است. سیستم سوخت هواپیما فشارهای وارد بر بدنه هواپیما به دلیل حرکت بنزین در داخل محفظه بنزین و تاثیر گرانشی حجم بنزین را محاسبه می‌کند.

تا به اینجا عملیات مربوط به تقسیم‌بندی و تخصیص آنها به سیستمها و کنترل کنندگان آنها انجام شده است. بنابراین برای اینکه معماری کامل شود باید عملیات زیر انجام بپذیرد:

- شناسایی نمونه‌های کنترل کنندگان فرزندان برای زیرسیستم نیروی محرکه
- تجزیه سایر گروه‌ها، سیستمها و زیرسیستمهای آنها

[1] Airframe
[2] Propulsion
[3] Landing gear
[4] Flight controls



شکل ۸-۸: زیرسیستم نیروی محرکه

فهرست مطالب

فصل نهم

مستندسازی معماری نرم‌افزار

در این فصل به منظور مستندسازی معماری نرم‌افزار یک قالب استاندارد معرفی خواهد شد که در آن مشخص شده است که چه اطلاعاتی باید در مستند معماری قرار داده شود. همچنین راهنمایی‌هایی در مورد چگونگی بدست آوردن اطلاعات مورد نیاز جهت قرار گرفتن در مستند معماری ارائه خواهد شد. نهایتا نیز نقش *UML* در مستندسازی معماری بررسی خواهد شد.

۹-۱- مقدمه

معماری نرم‌افزار نقش مرکزی در توسعه سیستم و سازماندهی فرآورده‌های آن دارد. در واقع معماری به عنوان یک طرح جامع و کلی برای پروژه و توسعه سیستم در نظر گرفته می‌شود. معماری کارهایی که باید توسط تیم طراح و پیاده‌سازان انجام شود را تعریف می‌کند. همچنین به عنوان عامل اصلی تحقق خصوصیات کیفی از قبیل کارایی، قابلیت تغییرپذیری و امنیت به شمار می رود.

معماری یک فرآورده برای تحلیل اولیه است تا بدین صورت تضمین شود که روش طراحی منجر به سیستم قابل قبول خواهد شد. علاوه بر این، معماری ساختارهای کلیدی را نگهداری می‌کند که به فعالیت‌های (تلاشهای) مربوط به درک و نگهداری سیستم بعد از توسعه کمک می‌کند.

مستندسازی معماری یک مرحله اصلی در شکل‌گیری معماری است. اگر کسی نتواند معماری (حتی معماری خوب و کامل) را درک کند یا معماری منجر به این شود که ذینفعان کلیدی معماری درک اشتباهی از آن داشته باشند در این صورت آن معماری هیچ فایده‌ای ندارد. برای ایجاد یک معماری قوی باید آن را با جزئیات کافی و بدون ابهام توضیح داد و همچنین به صورتی آن را سازماندهی کرد که دیگران خیلی سریع بتوانند اطلاعات مورد نیاز خود را از آن بدست آورند. در صورت ایجاد یک معماری بدون این خصوصیات، تلاشهای انجام شده بیهوده خواهد بود زیرا معماری بدست آمده قابل استفاده نخواهد بود. با توجه به این موضوعات، در ادامه تشریح خواهد شد که چه اطلاعاتی مهمی باید در معماری در نظر گرفته شوند و راهنمایی‌هایی نیز پیرامون نحوه بدست آوردن آن اطلاعات داده خواهد شد.

۹-۲- کاربردهای مستندسازی معماری[1]

همچنان که معماری برای یک سیستم به نیازمندیهای سیستم وابسته است، مستندسازی برای یک معماری نیز وابسته به نیازمندیها است. برای مستندسازی نمی‌توان اندازه یا حجم مستندات را به صورت یک استاندارد تعریف کرد (یعنی یک اندازه مشخص از مستندات که همه معماریها باید نه کمتر و نه بیشتر از آن مستند تولید کنند). مستندسازی باید به اندازه کافی انتزاعی باشد تا توسط کارمندان (توسعه‌دهندگان) جدید به سرعت درک شود و به اندازه کافی نیز باید همراه با جزئیات باشد تا بتواند به عنوان یک برنامه کلی برای تحلیل در نظر

[1] Uses of architectural documentation

گرفته شود. به عنوان مثال، مستند معماری برای تحلیل‌گر امنیت می‌تواند خیلی متفاوت‌تر از مستند معماری باشد که به یک پیاده‌ساز داده می‌شود.

مستند معماری هم تجویزی[1] و هم تشریحی[2] است. تجویزی و تشریحی بودن مستند بدین معنی است که مستند برای بعضی افراد مشخص می‌کند که چه محدودیت‌هایی باید بر روی تصمیمات در نظر گرفته شود تا بتوانند به طور صحیح اعمال شوند و برای بعضی افراد نیز تشریح می‌کند که تصمیمات اتخاذ شده قبلی پیرامون سیستم چه نکاتی هستند. کلیه این توصیفات حکایت از آن دارد که ذینفعان مختلف مستندسازی، نیازمندیهای گوناگونی دارند (مانند انواع مختلف اطلاعات، سطوح مختلف اطلاعات، رفتارهای مختلف از اطلاعات). بنابراین مستند مربوط به سیستم باید طوری تولید شود که به ذینفعان کمک کند تا سریعا اطلاعات موردعلاقه خود را با کمترین اطلاعات نامربوط بدست آورند و این بدان معنی است که برای ذینفعان مختلف باید مستندات متفاوت تولید شود. به عبارت بهتر باید یک مجموعه مستند واحد با یک نقشه تولید شود که به ذینفعان مختلف کمک کند در داخل مستند سریعا اطلاعات مورد نظر خود را پیدا کنند (از طریق نقشه در داخل معماری ناوبری کنند).

یکی از مهمترین قوانین عمومی مستندسازی به خصوص در مورد مستندسازی معماری آن است که مستند بر حسب دیدگاه یا نقطه نظر خواننده نوشته شود. مستندی که به راحتی نوشته شود ولی به راحتی قابل خواندن نباشد نمی‌تواند مورد استفاده قرار بگیرد. درک این موضوع که ذینفعان چه کسانی هستند و اینکه چگونه آنها می‌خواهند از مستند استفاده کنند، می‌تواند عاملی خوبی برای تولید و سازماندهی یک مستند قابل دسترس و قابل استفاده برای آنها باشد. همان طور که قبلا (در فصل دوم) ذکر شده است معماری وسیله‌ای برای ارتباط بین ذینفعان است که در این میان مستندسازی معماری، آن ارتباطات را سهل‌تر می‌کند. بعضی مثالها پیرامون ذینفعان معماری و اطلاعاتی که آنها انتظار دارند در مستندات معماری پیدا کنند در جدول ۹-۱ نشان داده شده است. علاوه بر اینها ذینفعان به دو دسته فصلی[3] و جدید[4] تقسیم می‌شوند. یک ذینفع جدید اطلاعاتی را می‌خواهد که از نظر محتوا مشابه با اطلاعاتی است که ذینفع فصلی می‌خواهد ولی با این تفاوت

[1] Perpective
[2] Descriptive
[3] Seasoned
[4] New

که حجم آن کم و خیلی مقدماتی‌تر است. همچنین اینکه مستندسازی معماری یک روش کلیدی برای آموزش افرادی است که نیاز به دید کلی از سیستم دارند. مانند توسعه‌دهندگان جدید، بانی‌های مالی جدید[1]، بازبین کننده‌های پروژه و غیره.

جدول ۹-۱: ذینفعان و نیازهای ارتباطی ارائه شده به وسیله معماری

ذینفع	نوع استفاده از معماری
معمار و مهندسین نیازمندیها	برای بحث و گفتگو در مورد نیازمندیهای متضاد و ایجاد توازنی بین آنها
معمار وطراحان بخشهای تشکیل دهنده سیستم	برای حل رقابت منابع و برقراری کارایی و دیگر هزینه‌های مربوط به مصرف منابع در زمان اجرا (خصوصیات کیفی)
پیاده‌سازها	برای تهیه اجبارهای غیرقابل تغییر در فعالیت‌های توسعه پائین دست
آزمایش‌کننده‌ها و یکپارچه سازها	برای تعیین رفتار صحیح در تست جعبه سیاه توسط مولفه‌هایی که باید یکدیگر را کامل کنند
طراحان دیگر سیستمهایی که با سیستم در ارتباطند	برای تعیین قسمت‌هایی که تحت تاثیر تغییرات آینده هستند
متخصص خصوصیات کیفی	برای تهیه مدلی که ابزارهای تحلیل مانند شبیه‌سازها و غیره بتوانند از آن استفاده نمایند (تحلیل معماری)
مدیران	برای آماده‌سازی تیم توسعه مطابق با کارهای منتسب شده، برنامه‌ریزی منابع پروژه و در جریان بودن روند کار انجام شده توسط تیم‌های مختلف
مدیران خط تولید	برای تعیین اینکه آیا نسخه جدیدی از محصول در نظر گرفته شده است یا خیر
تیم تضمین کیفیت	برای اجرای آزمایش هماهنگی بین پیاده‌سازی و تجویزهای معماری

۹-۳- انتخاب دیدهای مرتبط

مهمترین مفهوم مرتبط با مستندسازی معماری نرم‌افزار، دید[2] است. دید یک مجموعه منسجم از عناصر معماری را نمایش می‌دهد که به وسیله ذینفعان نوشته و توسط ذینفعان خوانده می‌شود. معماری یک موجودیت پیچیده‌ای است که نمی‌توان آن را در یک روش تک بعدی ساده تشریح کرد، بنابراین نیاز به دیدهای مختلفی وجود دارد که هر کدام از یک بعد معماری را تشریح کنند.

[1] Funding sponsors
[2] View

مستندسازی معماری شامل مستندسازی دیدهای مرتبط (مناسب) و اضافه کردن مستنداتی است که به بیش از یک دید اعمال می‌شوند. اصول بیان شده پیرامون مستندسازی بسیار ارزشمند هستند زیرا باعث می‌شوند مسئله مستندسازی معماری به سه بخش قابل ردیابی[1] –انتخاب دیدهای مرتبط، مستندسازی هر یک از دیدها و مستندسازی اطلاعاتی که به بیش از یک دید اعمال می‌شوند– تقسیم شود. در ادامه، بخش انتخاب دیدهای مرتبط تشریح شده و دو بخش دیگر نیز در قسمت‌های بعدی تشریح خواهد شد.

به منظور تولید یک بسته کامل مستندسازی که بتواند کلیه نیازمندیهای ذینفعان را پوشش دهد، معمار باید ابتدا به خوبی ذینفعان را شناسایی کند و بعد با استفاده از برنامه و نقشه آنها، مستند موردنظر را تولید کند. معماری می‌تواند به عنوان دستورالعمل عملیاتی برای پیاده‌سازها، یک نقطه شروع برای درک و بازیابی دارایی[2] نرم‌افزار، به صورت یک طرح کلی برای برنامه‌ریزی پروژه و غیره در نظر گرفته شود که همه آنها مبتنی بر خواسته‌ها و انتظارات ذینفعان هستند. در واقع این اهداف نشان دهنده این هستند که چه دیدهایی باید در مستند نیازمندیها وجود داشته باشند. (ذینفعان، به مستندسازی نیاز دارند تا بتوانند از طریق آن به اهداف خود دست پیدا کنند). به همین صورت خصوصیات کیفی خیلی مهم برای ذینفعان در توسعه سیستم نیز انتخاب دیدهای مورد نیاز برای مستندسازی را تحت تاثیر قرار می‌دهد.

به عنوان مثال، دید لایه‌بندی اطلاعاتی درباره قابلیت جابجایی سیستم فراهم می‌کند و دید استقرار نیز اجازه می‌دهد که درباره کارایی و قابلیت اطمینان سیستم استدلالهایی انجام شود. این خصوصیات کیفی در مستند معماری به وسیله تحلیل‌گرانی (احتمالا معمار) مورد ارجاع قرار می‌گیرد که نیاز دارند معماری را بررسی کنند تا مطمئن شوند معماری خصوصیات کیفی مورد نیاز را برآورده می‌کند. به طور خلاصه، دیدهای گوناگون اهداف و کاربردهای متفاوتی را پوشش می‌دهند. این موضوع اساسا نشان می‌دهد که چرا دیدها یا مجموعه دیدهای خاص برای معماری به هیچ عنوان پیشنهاد نمی‌شوند.

در واقع دیدهایی که باید مستند شود مبتنی بر چگونگی استفاده ذینفعان از مستند است. دیدهای متفاوت، عناصر و رابطه‌های مختلفی از سیستم را به تصویر می‌کشانند. جدول ۹–۲ یک مجموعه از ذینفعان و انواع دیدهایی که آنها فکر می‌کنند برای هدفشان مفید است (یعنی از آن دیدها استفاده می‌کنند) را نشان می‌دهد.

[1] Traceable
[2] Asset

همان طور که در فصل دوم ذکر شد، دیدها می‌توانند به سه دسته‌ی ماژول، مولفه و اتصال و تخصیص تقسیم شوند که این سه دسته‌بندی نشان دهنده این موضوع است که معماران نیاز دارند به سه روش مختلف در مورد نرم‌افزار خودشان فکر کنند:

۱. چگونه سیستم به صورت یک مجموعه از *واحدهای پیاده‌سازی* ساخته شود؟
۲. چگونه سیستم به صورت مجموعه‌ای از عناصر که *رفتارها* و *تعاملات زمان اجرا* دارند، ساخته شود؟
۳. چگونه سیستم به *ساختارهای غیرنرم‌افزاری* در محیط خودش مرتبط شود؟

جدول ۹-۲: ذینفعان و مستند معماری مرتبط با آنها

	دیدهای ماژول				دیدهای C&C[1]	دیدهای تخصیص	
ذینفع	**تجزیه**	**استفاده از**	**کلاس**	**لایه‌بندی**	**انواع مختلف**	**استقرار**	**پیاده‌سازی**
مدیر پروژه	*S*	*S*		*S*		*d*	
اعضای تیم توسعه	*d*	*d*	*d*	*d*	*D*	*S*	*S*
آزمایش کننده‌ها و یکپارچه سازها		*d*	*d*		*S*	*S*	*S*
نگهدارنده‌ها	*d*	*d*	*d*	*d*	*D*	*S*	*S*
سازنده کاربرد خط تولید		*d*	*S*	*o*	*S*	*S*	*S*
مشتری					*S*	*o*	
کاربرنهایی					*S*	*S*	
تحلیل‌گر	*d*	*d*	*S*	*d*	*S*	*d*	
حمایت کننده ساختار پایه	*S*	*S*		*S*		*S*	*d*
ذینفعان جدید	×	×	×	×	×	×	×
معماران فعلی و آینده	*d*	*d*	*d*	*d*	*D*	*d*	*S*

کلید:

جزئیات کامل =*d* – کمی جزئیات =*s* – اطلاعات کلی =*o* – هر چیزی = ×

[1] Component and connector

دیدها به سادگی یک مجموعه از عناصر سیستم و ارتباطات بین آنها را نشان می‌دهند، بنابراین عناصر و رابطه‌هایی که به نظر می‌رسد برای گروهی از ذینفعان مفید و سودمند است یک دیدگاه معتبر تولید می‌کند. در ادامه یک رویه سه مرحله‌ای برای انتخاب دیدهای یک پروژه ارائه خواهد شد که مراحل آن به شرح زیر است:

۱. *تولید یک فهرست دید نامزد*

این مرحله با تولید جدول دید/نیازمندیها (مانند جدول ۹-۲) آغاز می‌شود. مطمئنا فهرست ذینفعان (تعداد سطرها) در جدول برای هر پروژه متفاوت خواهد بود و تعداد ستونهای جدول نیز مبتنی بر دیدهایی خواهد بود که به سیستم اعمال می‌شوند. بعضی دیدها (مانند دید تجزیه و استفاده از) به هر سیستم اعمال می‌شوند در حالی که دیدهای دیگر (مانند لایه‌بندی و بسیاری از دیدهای مولفه و اتصال از قبیل داده مشترک و سرویس‌گیرنده و سرویس‌دهنده) فقط به سیستمهایی که مبتنی بر روش خودشان طراحی شده‌اند، اعمال می‌شوند. به عنوان مثال اگر یک سیستم قرار است به صورت سرویس‌گیرنده و سرویس‌دهنده پیاده‌سازی شود در این صورت از دید سرویس‌گیرنده و سرویس‌دهنده استفاده می‌شود. نهایتا بعد از اینکه سطرها و ستونهای جدول مشخص شدند باید ستونهای جدول پر شوند تا مشخص شود هر ذینفع به چه میزان اطلاعات از هر دید نیاز دارد.

۲. *ترکیب دیدها*

فهرست دید نامزد مشخص شده در مرحله اول احتمالا منجر به تعداد زیادی از دیدهای غیرعملی خواهد شد. به منظور کاهش اندازه فهرست به مقدار قابل مدیریت، ابتدا دیدهایی در جدول مورد بررسی قرار می گیرد که نیاز به اطلاعات کلی در مورد آنها است یا اینکه ذینفعان کمتری در ارتباط با آنها وجود دارند. سپس بررسی می‌شود که کدام یک از این دیدها می‌توانند در قالب دید دیگر پوشش داده شوند. بعد از مشخص شدن این نوع دیدها، دیدهایی که توسط دیدهای قوی پوشش داده می‌شوند، حذف می‌شوند. در واقع دیدها حذف نمی‌شوند بلکه با دید دیگری ترکیب می‌شوند. به عنوان مثال، برای پروژه‌های کوچک و متوسط، دید پیاده‌سازی به راحتی توسط دید تجزیه ماژول پوشش داده می‌شود. دید تجزیه ماژول همچنین می‌تواند دربرگیرنده دیدهای لایه‌بندی و استفاده نیز باشد. دید استقرار نیز معمولا ترکیبی از کلیه دیدهای مولفه و اتصال (برای مثال دید فرآیند) است که مولفه‌های تخصیص داده شده به سخت‌افزار را نمایش می‌دهد.

۳. *اولویت‌بندی دیدها*

بعد از مرحله دوم یک مجموعه مناسب از دیدها برای نمایش به ذینفعان وجود دارد. در این مرحله نیاز است که مشخص شود کدام یک از دیدها باید اول تولید شوند. چگونگی تصمیم‌گیری پیرامون این موضوع به جزئیات خاص هر پروژه بستگی دارد. این نکته را هم در نظر داشته باشید که هیچ اجباری وجود ندارد که قبلا از شروع ساخت یک دید، دید یا دیدهای دیگر کامل شده باشند. برای ایجاد کلیه دیدهای یک معماری می‌توان از اطلاعات سطح بالا شروع کرده و به صورت تدریجی آنها را کامل کرد.

ایجاد تدریجی دیدهای معماری دو دلیل مهم دارد. اول اینکه مدیران یک شرکت یا سازمان خواهان آن هستند که به صورت مستمر خروجی معماری را ببینند تا بتوانند نظرات خود را ارائه کنند، این موضوع منجر به توسعه تدریجی معماری می‌شود. دلیل دوم نیز آن است که در روند ساخت دیدها، ذینفعان ممکن است انتظارات خود را تغییر دهند، بنابراین با روش تدریجی ساخت دید می‌توان تا حدی مشکلات ناشی از تغییرات را کاهش داد.

۹-۴- مستندسازی دید

هیچ قالب استاندارد صنعتی برای مستندسازی دید وجود نداد. اما در اینجا یک قالب استاندارد هفت بخشی تشریح خواهد شد که خیلی خوب می‌توان در عمل از آن استفاده کرد (یعنی در صنعت استفاده و آزمایش شده است). پیش از هر چیز، وقتی هر بخشی انتخاب می‌شود تا در مستند قرار بگیرد باید مطمئن شویم که سازماندهی استاندارد در آن رعایت شده است. تخصیص اطلاعات خاص به بخشهای خاص به نویسنده مستند کمک خواهد کرد که به راحتی تکمیل بودن آن بخش یا وظیفه را تشخیص دهد. همچنین به خوانندگان مستند کمک خواهد کرد که اطلاعات مورد نیاز خود را بلافاصله پیدا کنند. بخشهای قالب استاندارد پیشنهادی برای مستندسازی معماری به شرح زیر هستند:

۱. *نمایش اولیه*[1]

نمایش اولیه، عناصر و رابطه بین آنها را نشان می‌دهد و دربرگیرنده اطلاعات کلی از سیستم است. نمایش اولیه باید شامل عناصر اصلی و ارتباطات بین دیدها باشد. بدین منظور باید تحت یک مدیریت مناسب قرار بگیرد تا کلیه عناصر و ارتباطات را در برنگیرد. به عنوان مثال، برای یک سیستم می‌توان عناصر و ارتباطاتی که در هنگام اجرا معمولی سیستم وجود دارند را نمایش داد ولی مدیریت خطا و پردازش استثنائات در سیستم را نادیده گرفت.

نمایش اولیه معمولا به صورت گرافیکی نشان داده می‌شود. در حقیقت بیشتر نمادهای گرافیکی به عنوان یک قالب برای نمایش اولیه به شمار می‌روند. اگر نمایش اولیه به صورت گرافیکی باشد باید یک راهنمای کلیدی همراه با آن نیز وجود داشته باشد تا نمایش اولیه و یا نمادهای به کار رفته در آن را تشریح کند. گاهی اوقات نمایش اولیه به صورت جدول است. جداول معمولا یک روش خوب برای نمایش یک مقدار زیادی از اطلاعات به صورت فشرده هستند. همچنین در نمایش اولیه، نمایش متنی وظیفه ارائه یک خلاصه مختصر از مهمترین اطلاعات موجود در هر دید را بر عهده دارد. یک مثال از نمایش اولیه متنی، دید تجزیه ماژول *A-7E* است که در فصل سوم نشان داده شده است.

۲. *کاتالوگ عنصر*[2]

کاتالوگ عنصر، عناصر و ارتباطات بین آنها در نمایش اولیه و یا سایر نمایش‌ها را تشریح می‌کند. نمایش اولیه معمولا چیزهایی هستند که معماران بر روی آن متمرکز هستند در این نمایش هیچ اطلاعاتی در مورد تصاویر به کار رفته وجود ندارد و نیز اینکه اطلاعات ارزشمند کمتری در آن وجود دارد. برای مثال اگر یک نمودار عناصر *A, B* و *C* را نشان دهد همراه با آن یک مستندی وجود خواهد داشت که تشریح می‌کند که *A, B* و *C* چه چیزی هستند و هدف و نقش آنها در سیستم چیست؟

دید تجزیه ماژول، عناصری به نام ماژول دارند، ارتباطات در آن از نوع "بخشی است از" بوده و خصوصیات آنها نشان‌دهنده مسئولیت هر ماژول است. یک دید فرآیند نیز عناصری به نام فرآیند

[1] Primary representation
[2] Element catalog

دارد، ارتباطات در آن نشان‌دهنده همگام‌سازی و سایر ارتباطات ممکن بین فرآیندها است و خصوصیات نیز دربرگیرنده زمانبندی پارامترها است. علاوه بر اینها اگر عناصر یا ارتباط‌هایی وجود داشته باشند که در نمایش اولیه حذف شده‌اند در این بخش باید آن نمایش‌ها را معرفی و هر کدام را تشریح کرد. واسط‌ها و رفتارهای عناصر، دو بعد دیگر از کاتالوگ هستند که در این بخش از مستند باید به صورت مختصر تشریح شوند.

۳. *نمودار متن*[1]

نمودار متن نشان می‌دهد که چگونه سیستم با توجه به دیدهای مرتبط با محیط‌شان و مفاهیم پایه (واژگان) موجود در دیدها به تصویر کشیده می‌شود. برای مثال، در دید مولفه و اتصال نشان داده می‌شود که کدام مولفه و اتصالات با مولفه‌های خارجی و از طریق کدام واسط و پروتکل ارتباط برقرار می‌کند.

۴. *راهنمای تغییرپذیری*[2]

راهنمای تغییرپذیری نشان می‌دهد چگونه نقاط تغییر که به عنوان بخشی از معماری هستند به کار گرفته می‌شوند. در بعضی معماریها، تصمیمات تا مرحله بعدی فرآیند توسعه به تعویق انداخته می‌شوند ولی با این وجود باز هم باید مستندسازی انجام شود. برای مثال، یک تغییرپذیری در معماری خط تولید آن قسمتی است که مناسب برای چندین سیستم خاص است. راهنمای مربوط به تغییرپذیری باید شامل مستنداتی درباره هر نقطه تغییر در معماری به شرح زیر باشد:

– گزینه‌هایی که در یک *نقطه تغییر قابل انتخاب* هستند. در دید ماژول، گزینه‌ها نسخه‌های مختلف یا پارامترهای ماژولها هستند. در دید تخصیص، گزینه‌ها دربرگیرنده شرایطی هستند که در آن حالت یک عنصر نرم‌افزاری باید به یک پردازنده خاص تخصیص داده شود.

– *زمان انقیاد* گزینه‌ها، بعضی انتخاب‌های مربوط به گزینه‌ها در زمان طراحی یا زمان ایجاد و یا زمان اجرا انجام می‌شوند که متناسب با آن نیز زمان انقیاد متفاوت خواهد بود.

[1] Context diagram
[2] Variability guide

۵. *پیش‌زمینه معماری*[1]

پیش‌زمینه معماری تشریح می‌کند که چرا تصمیماتی که در طراحی وجود دارند، اتخاذ شده‌اند و چرا تصمیمات دیگر در نظر گرفته نشده‌اند. در واقع هدف از این بخش آن است که تشریح شود چرا طراحی بدین شکل است و یک توافق بر روی تصمیمات انجام شود. یک پیش‌زمینه معماری دربرگیرنده مواردزیر است:

– *استدلال*[2]، تشریح می‌کند که چرا تصمیمات اعمال شده به دیدها اتخاذ شده‌اند و چرا تصمیمات دیگررد شده‌اند.

– *نتایج تحلیل*، بیان می‌کند که چه تغییراتی باید در هنگام یک تغییر انجام شوند.

– *فرضیات موجود در طراحی*

۶. *فهرست اصطلاحات*[3]

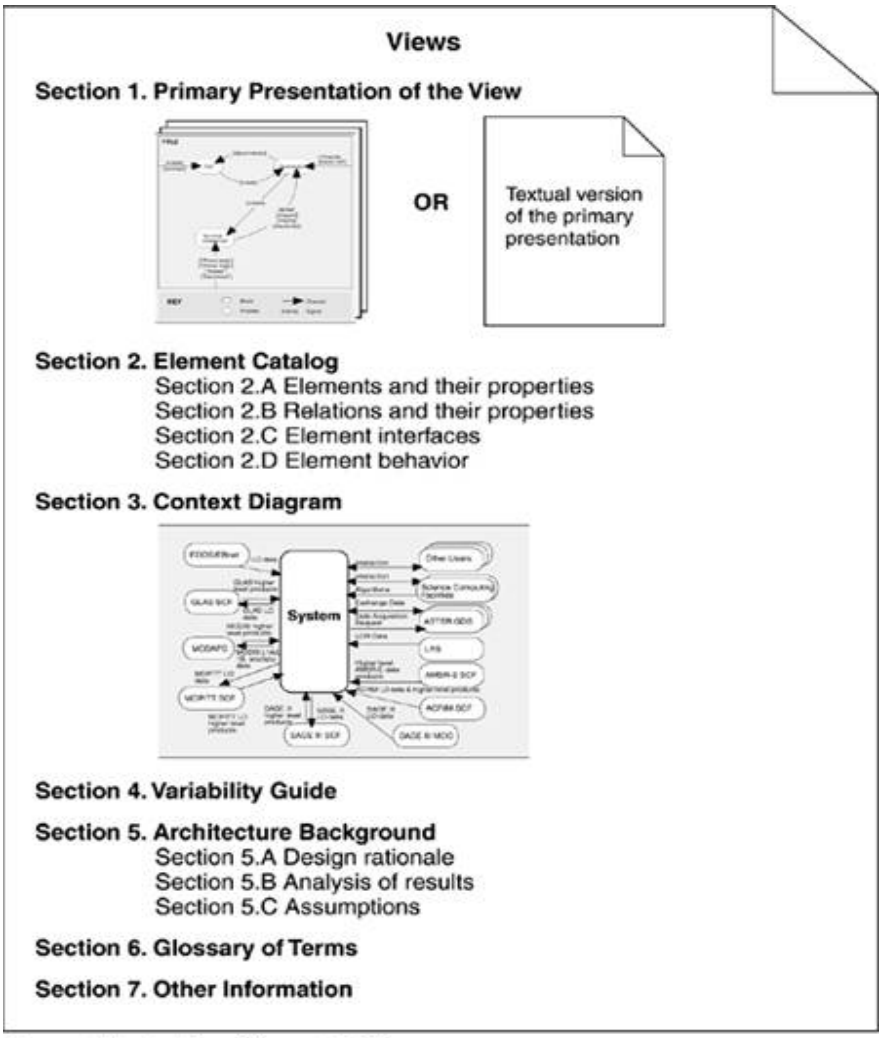
فهرست اصطلاحات در دیدها استفاده می‌شود که دربرگیرنده توضیحات مختصر درباره کلمات استفاده شده در طراحی (دید) است.

۷. *دیگر اطلاعات*[4]

دقت محتویات این بخش به استاندارد استفاده شده در سازمان بستگی دارد. آن می‌تواند در برگیرنده اطلاعاتی از قبیل تاریخچه تغییرات، پیکربندی کنترل داده و ابتکارات موجود باشد. همچنین در این بخش معمار می‌تواند به منظور قابلیت ردیابی ارجاعاتی به بخشهایی از مستند نیازمندیها داشته باشد. در واقع اطلاعات قرار گرفته در این بخش غیرمعمارانه هستند. با این وجود، آن مفید خواهد بود که جنبه‌های غیر معمارانه در این بخش ذخیره شود. در هر صورت نخستین قسمت از این بخش باید جزئیات محتویات آن را مشخص کند.

شکل ۹–۱ بخشهای این قالب استاندارد مستندسازی را به صورت خلاصه نشان می‌دهد.

[1] Architecture background
[2] Rationale
[3] Glossary of terms
[4] Other information



شکل ۹-۱: هفت بخش از قالب استاندارد مستندسازی معماری

۹-۴-۱- مستندسازی رفتار

دیدها اطلاعات ساختاری سیستم را نمایش می‌دهند. با این حال، اطلاعات ساختای برای استدلال پیرامون بعضی خصوصیات سیستمی کافی و مناسب نیستند. برای مثال استدلال درباره بن‌بست به ترتیب تعاملات بین عناصر وابسته است و اطلاعات ساختاری نمی‌تواند این ترتیب اطلاعات را نشان دهد. توصیف رفتار، اطلاعاتی را مشخص می‌کند که منجر به آشکار شده ترتیب تعاملات بین عناصر، فرصتهای همزمانی[1] و وابستگی زمانی تعاملات (در یک زمان خاص یا بعد از یک دوره زمانی) می‌شود.

[1] Opportunities of concurrency

مستندسازی رفتار می‌تواند در ارتباط با یک عنصر یا یک مجموعه از عناصر باشد که به صورت هماهنگ با یکدیگر فعالیتی را انجام می‌دهند. مواردی که در مستند باید مدلسازی شوند کاملا وابسته به سیستمی است که طراحی برای آن انجام می‌شود. برای مثال اگر یک سیستم بلادرنگ وجود داشته باشد باید اطلاعات زیادی پیرامون خصوصیات زمانبندی و زمان وقوع رویدادها بیان شود. در یک سیستم بانک، ترتیب رویدادها خیلی مهمتر از زمان واقعی رخ دادن رویدادها است. همچنین اینکه تکنیک‌های مدلسازی و نمادهای استفاده شده مختلف به نوع تحلیلی که باید انجام شود، بستگی دارد. در *UML* نمودار ترتیب و نمودار حالت مثالهایی از تشریح رفتار هستند.

نمودار حالت در سال ۱۹۸۰ برای تشریح سیستمهای واکنشی[1] توسعه داده شده است. بعد از مدتی انواع گسترشهای مناسب از قبیل "حالت تودرتویی "[2] و "حالت و"[3] به آن اضافه شد تا بتواند از مدلسازی انتزاعی و همزمانی به شکل مناسب حمایت کند. نمودار حالت اجازه استدلال درباره کل سیستم را امکان‌پذیر می‌کند. یک نمودار ترتیب نیز ترتیب تغییرات محرک‌ها را مستند می‌کند. به علاوه یک همکاری[4] بر حسب نمونه‌های مولفه‌ها و ارتباطات بین آنها ارائه می‌کند و ترتیب تعاملات در گذر زمان را نشان می‌دهد.

دید عمودی، زمان و دید افقی مولفه‌های مختلف را نشان می‌دهد. نمودار ترتیب اجازه استدلال مبتنی بر سناریوهای خاص را می‌دهد. همچنین این امکان را فراهم می‌کند که بتوان به سئوالات مهمی پاسخ داد. برای مثال در زمانی که سیستم به محرک‌های خاص تحت شرایط خاص پاسخ می‌دهد، چه فعالیت‌های موازی رخ می‌دهند؟

۹-۴-۲- مستندسازی واسط‌ها

واسط، مرزی در میان دو موجودیت مستقل است که نشان می‌دهد چگونه آنها با یکدیگر تعامل یا ارتباط دارند. تعریف مطرح شده برای معماری در فصل دوم نشان می‌دهد که واسط‌های عناصر که خصوصیات قابل مشاهده از بیرون عناصر را نشان می‌دهند، معمارانه هستند. از آنجایی که تحلیل یا ایجاد سیستم بدون واسط‌ها امکان‌پذیر نیست، بنابراین مستندسازی واسط‌ها یک بخش مهم در مستندسازی معماری است. مستندسازی

[1] Reactive systems
[2] Nesting state
[3] And
[4] Collaboration

واسط دربرگیرنده نام، شناسه، اطلاعات نحوی و لغوی واسط است. دو خصوصیت اول (نام و شناسه) امضا واسط را تعریف می‌کنند. زمانی که منابع واسط برنامه‌های قابل فراخوانی هستند امضا واسط، برنامه‌ها و پارامترهای آنها را تعریف می‌کند. پارامترها به وسیله مشخص کردن ترتیب، نوع داده و اینکه مقادیر آنها به وسیله برنامه تغییر پیدا می‌کند یا نه، تعریف می‌شوند. یک امضا، اطلاعاتی است که باید در مورد یک برنامه وجود داشته باشد تا بتوان آن را فراخوانی کرد. برای نمونه، در زبان C و$C++$ سر فایل[1] یک امضا تلقی می‌شود.

امضاها مفید و کارا هستند (به عنوان مثال می‌توانند بررسی مکانیزه ایجاد[2] را امکان‌پذیر کنند) ولی آنها فقط بخشی از یک توصیف هستند. تطبیق امضا[3] تضمین می‌کند که یک سیستم به صورت موفق لینک یا کامپایل خواهد شد ولی به هیچ عنوان تضمین نمی‌کند که سیستم با موفقیت عمل خواهد کرد.

واسط با استفاده از مشخصات واسط[4] مستندسازی می‌شود. مشخصات واسط شرحی از خصوصیات واسط است که یک معمار برای هر مولفه در نظر می‌گیرد تا با استفاده از آن شرح مولفه را مشخص کند. معمار باید فقط چیزهایی را در مشخصات نمایش دهد که مورد نیاز برای تعامل با واسط هستند. از بعد دیگر معمار باید اطلاعاتی که مجاز برای نمایش بوده و اینکه به احتمال زیاد تغییر پیدا نخواهند کرد را مشخص کند. در مستندسازی واسط باید تعادل برقرار شود یعنی اینکه نه خیلی کم و نه خیلی زیاد در مورد یک واسط مستندسازی صورت بپذیرد. اطلاعات کم پیرامون واسط، توسعه‌دهنده را از برقراری تعامل موفق بین مولفه‌ها منع می‌کند. اطلاعات خیلی زیاد هم باعث می‌شود که اعمال تغییرات در آینده به سختی انجام شده و همچنین باعث می‌شود که درک واسط برای ذینفع مربوطه به سختی انجام شود.

عناصری که به صورت ماژول ظاهر می‌شوند اغلب به طور مستقیم با یک یا چند عنصر در دید مولفه و اتصال در ارتباط هستند. عناصر ماژول و مولفه و اتصال با یکدیگر شباهتهایی دارند. بنابراین مشخصات واسط در دید مولفه و اتصال می‌تواند به مشخصات واسط در دید ماژول ارجاع داشته باشد و خودش فقط اطلاعاتی را در بر گیرد که مختص به خودش است. به همین صورت از آنجایی که یک ماژول می‌تواند در بیش از یک دید ماژول

[1] Header file
[2] Automatic build checking
[3] Signature matching
[4] Interface specification

ظاهر شود (مانند دید تجزیه یا دید استفاده) باید در این صورت یک دید انتخاب شود و در آن مشخصات واسط تعریف شود و بعد در کلیه دیدهایی که این ماژول در آنها به کار رفته است به آن ارجاع کرد.

۹-۴-۲-۱- یک قالب برای مستندسازی واسط‌ها

در این قسمت یک سازماندهی استاندارد برای مستندسازی واسط پیشنهاد خواهد شد. بر حسب نوع کابرد و سازمان استفاده کننده، بخشهایی از آن می‌تواند حذف، اضافه و یا تغییراتی در آنها صورت بپذیرد. مهمتر از اینها آن است که این سازماندهی استاندارد قابل استفاده به صورت عملی است. به عبارت بهتر از تجارب موفق بدست آمده است. بخشهای اصلی این الگو به صورت زیر هستند:

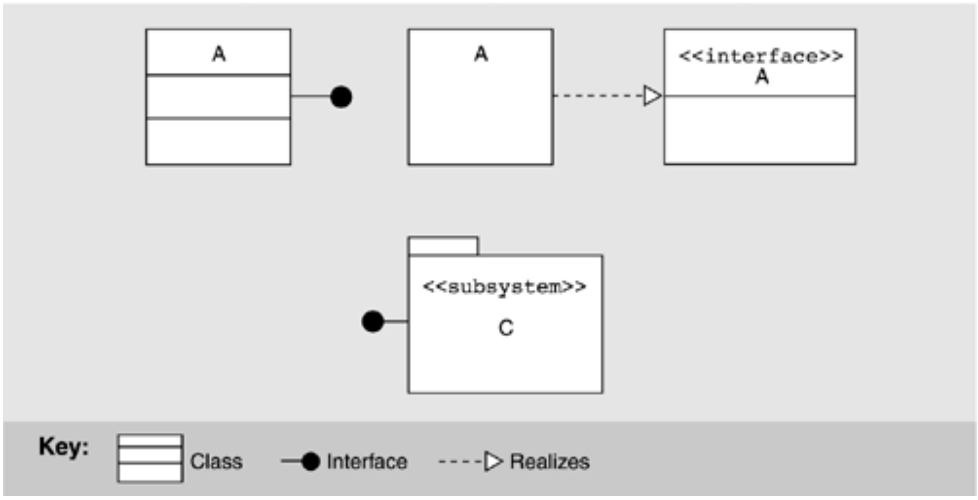
۱. *شناسه واسط*[۱]

زمانی که یک مولفه چندین واسط دارد شناسه واسط باعث جداسازی و مشخص کردن آنها می‌شود. این عمل معمولا به وسیله نامگذاری واسط‌ها انجام می‌شود. البته گاهی اوقات نیز لازم است که علاوه بر مشخص کردن شناسه، برای آنها نسخه نیز در نظر گرفته شود.

۲. *منابع فراهم شده*[۲]

قلب یک مستند واسط، منابعی هستند که یک مولفه فراهم می‌کند. این منابع از طریق تعیین گرامر[۳]، معنی (زمانی که استفاده می‌شوند چه اتفاقی می‌افتد) و هر گونه محدودیت در مورد استفاده از آنها تعریف می‌شوند. چندین نماد برای مستندسازی گرامر واسط وجود دارد. یکی از آنها زبان تعریف واسط *OMG* است که در *CORBA* از آن استفاده شده است. زبان تعریف واسط ساختارهای زبانی برای تعریف نوع داده، عملیات، خصوصیات و استثنائات فراهم می‌کند و تنها زبانی نیز که از اطلاعات معنایی حمایت می‌کند، مکانیزم توضیح نویسی[۴] است. بیشتر زبانهای برنامه‌نویسی روشهای خاص خود را برای تعریف امضا یک مولفه دارند. به عنوان مثال می‌توان سرفایلها در زبان *C* و بسته‌ها را در زبان *Ada* نام برد. نهایتا اینکه استفاده از کلیشه *<<interface>>* در زبان *UML* ابزاری را برای پوشش دادن اطلاعات گرامر در مورد یک واسط فراهم می‌کند (شکل ۹-۲).

[1] Interface identity
[2] Resources provided
[3] Syntax
[4] Comment mechanism



شکل ۹-۲: واسطها در *UML*

– *گرامر منبع*[1]، در واقع امضا منبع است. امضا در برگیرنده هر گونه اطلاعات مورد نیاز برای سایر برنامه‌ها است که از طریق آن بتوانند از منبع استفاده کنند. امضا همچنین شامل نام منبع، نام و نوع داده منطقی، پارامترها و غیره است.

– *مفهوم منبع*[2]، نتایج درخواست منبع را تشریح می‌کند که می‌تواند موارد زیر باشد:

- مقادیر تخصیص داده شده به داده‌های درخواست
- رویدادها و پیامهای خروجی از منبع
- رفتار سایر منابع که از خروجی تولید شده استفاده می‌کنند
- نتایج قابل مشاهده توسط عامل انسانی

علاوه بر اینها، نمایش مفهوم باعث آشکار شدن این موضوع می‌شود که آیا اجرای منابع به صورت اتمیک، معلق یا وقفه‌ای است؟ پرکاربردترین روش برای مشخص کردن اطلاعات مفهومی، زبان طبیعی است. زبان جبر منطقی که معمولا برای نمایش پیش شرطها و پس شرطها استفاده می‌شود یک روش نسبتا ساده و موثر برای بیان مفهوم منبع فراهم می‌آورد. همچنین ردیابها نیز به عنوان روشی برای بیان مفهوم به کار می‌روند. آنها در واقع ترتیبی از فعالیت‌ها و تعاملات بین عناصر را نشان می‌دهند که به نوعی پاسخ عناصر به کاربردهای خاص را تشریح می‌کند.

[1] Resource syntax
[2] Resource semantics

– *محدودیت‌های (اجبارهای) استفاده از منبع*

سئوال مطرح شده در اینجا آن است که منابع تحت چه شرایطی مورد استفاده قرار می‌گیرند. شاید لازم باشد که یک داده قبل از خوانده شدن مقداردهی اولیه شود یا اینکه یک رویه خاص نتواند قبل از فراخوانی رویه دیگر فراخوانی شود. ممکن است یک محدودیت بر روی تعداد مدعیان وجود داشته باشد که می‌توانند در یک لحظه با این منبع ارتباط داشته باشند. شاید یک مدعی، مالک منبع است و فقط او می‌تواند مولفه را در زمان خوانده شدن تغییر دهد. ممکن است منابع خاص یا واسط خاصی برای مدعیان خاص به دلایل امنیتی وجود دارد. علاوه بر اینها اگر منبع برای سرویس دادن نیاز داشته باشد تا سایر منابع نمایش یابند این موضع باید در مستند ذکر شود.

۳. *تعاریف نوع داده*[1]

اگر هر واسط از نوع داده‌ای استفاده کند که توسط نوع داده پایه زبان برنامه‌نویسی حمایت نشود، معمار باید تعریف آن نوع داده را به انواع موجود مرتبط کند. اگر آن نوع داده جدید به وسیله مولفه دیگری تعریف شده باشد در این صورت یک ارجاع به آن مولفه کافی خواهد بود. در هر حالت، برنامه‌نویسان باید بدانند که چگونه باید متغییرها و محدودیت‌های نوع داده را تعریف کنند، چگونه مقادیر حرفی برای نوع داده تعریف کند، چه عملیات و مقایسه‌هایی را باید بر روی عناصر نوع داده انجام دهند و اینکه چگونه یک نوع داده را به نوع داده دیگر تبدیل کند.

۴. *تعاریف استثناها*[2]

این تعاریف انواع استثنائاتی را که می‌توانند از طریق واسط منبع ایجاد شوند، تشریح می‌کنند. از آنجایی که یک استثناء ممکن است توسط چندین واسط ایجاد شود (منجر به یک استثناء شود)، بنابراین یک روش مناسب آن است که استثنائات هر واسط در یک فهرست مشخص شده ولی تعریف آنها را در داخل یک واژه‌نامه مجزا نگهداری شود. همچنین در این بخش می‌توان رفتارهای مدیریت استثنائات مشترک را نیز تعریف کرد.

[1] Data type definitions
[2] Exception definitions

۵. *قابلیت تغییرپذیری فراهم شده به وسیله واسط*[1]

در این بخش سئوال این است که آیا واسطها اجازه می‌دهند که مولفه‌ها از طریق آنها پیکربندی شوند. بنابراین کلیه پارامترهای پیکربندی و اینکه چگونگی آنها مفهوم واسط را تحت تاثیر قرار می‌دهند، باید مستند شوند. به عنوان مثالی از قابلیت تغییر می‌توان قابلیت ساختار داده‌های قابل مشاهده و خصوصیات کارایی الگوریتمهای پایه را نام برد.

۶. *مشخصات خصوصیت کیفی واسط*[2]

معمار باید مستند کند که چه مشخصات خصوصیات کیفی واسط (کارایی یا قابلیت اطمینان) برای کاربران عنصر مشخص باشد. این اطلاعات می‌تواند به صورت محدودیت‌هایی بر روی پیاده‌سازی عناصری باشند که واسط را تحقق می‌بخشند.

۷. *نیازمندیهای واسط*[3]

چیزی که عنصر به آن نیاز دارد ممکن است خاص باشد و توسط منابع فراهم شده به وسیله سایر عناصر تحقق پیدا کند. الزامهای مستندسازی برای منابع فراهم کننده نیازمندیهای یک عنصر با الزامهای مستندسازی سایر منابع یکسان هستند (گرامر، معنا و محدودیت‌های کاربرد). گاهی اوقات مستندسازی اینگونه اطلاعات به عنوان فرضیاتی که طراح عناصر درباره سیستم در نظر گرفته است، پرکاربرد و مناسب است. در این حالت مستندات به وسیله خبرگانی که می‌توانند فرضیات را قبل از پیشرفت طراحی تایید یا رد کنند، مورد بررسی قرار می‌گیرد.

۸. *دلایل و منطق طراحی*[4]

به منظور استدلال در مورد معماری مخصوصا معماریهای بزرگ، معمار باید دلایل طراحی واسطهای یک عنصر را ثبت کند. در مستند منطق طراحی باید انگیزه طراحی، محدودیت‌ها، توافق‌ها، ملاحظات طراحی در نظر گرفته شده و نپذیرفته شده و همچنین ملاحظات مربوط به چگونگی تغییر واسطها در آینده، تشریح شود.

---

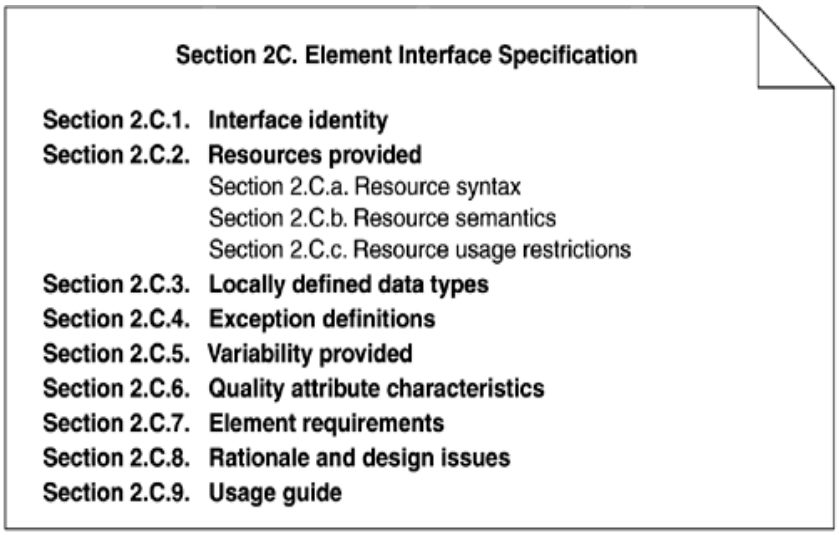
[1] Variability provided by the interface
[2] Quality attribute characteristics of the interface
[33] Interface requirements
[4] Rationale and design issues

۹. *راهنمای استفاده*[1]

موارد ۲ و ۴ مربوط به قالب مستندسازی، اطلاعات مفهومی عنصر را بر روی منابع پایه مستند می‌کنند. این مستندسازی گاها به صورت یک سری اطلاعات مختصر است. در بعضی حالات نیز مفهوم باید از طریق استدلال درباره چگونگی تعاملات بین تعدادی از عناصر مشخص شود.

اساسا یک قرارداد به وسیله ملاحظات مربوط به ترتیب تعاملات مستند می‌شود. قراردادها می‌توانند رفتار کامل تعاملات یا الگوهای کاربرد مورد انتظار طراحان را نمایش دهند. اگر تعاملات با عناصر از طریق واسط‌های پیچیده انجام شود، مستند واسط باید مدل رفتاری ایستا مانند نمودار حالت را در ساختار خود داشته باشد.

شکل ۹-۳ قالب پیشنهادی برای مستندسازی واسط را نشان می‌دهد.



شکل ۹-۳: نه بخش مربوط به مستندسازی واسط

[1] User guide

۹-۵- مستندسازی بین دیدها

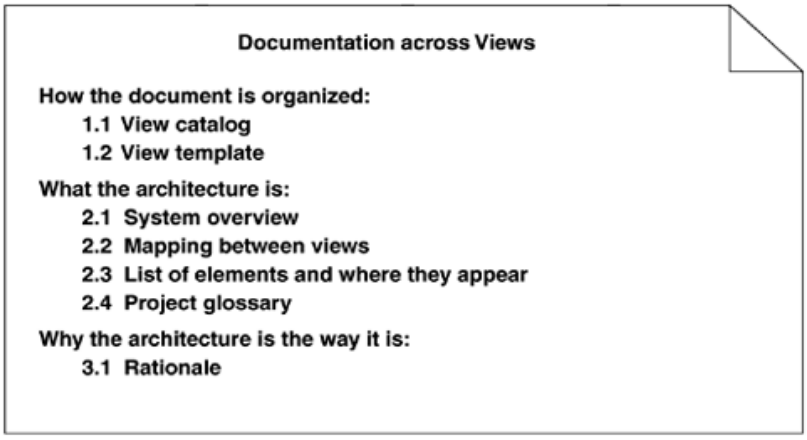
به منظور کامل کردن مستند دید باید اطلاعاتی که به بیش از یک دید یا کل مستند اعمال می‌شوند را مشخص کرد. مستند دید متقاطع[1] متشکل از سه جنبه اصلی چرا، چه چیز و چگونه است که در ادامه هر یک تشریح خواهد شد:

۱. چگونه *باید مستند سازماندهی شود که ذینفع معماری بتواند اطلاعات مورد نیاز خود را به صورت مناسب و با قابلیت اطمینان بالا بدست آورد*. این بخش متشکل از کاتالوگ دید[2] و قالب دید[3] است.

۲. *معماری چه چیزی است*. اطلاعات باقیمانده که در این بخش باید کشف شوند عبارتند از: دید کلی مختصر از سیستم که به نوعی مرتبط کننده ذینفعان با اهداف سیستم هستند، روشی برای مرتبط کردن دیدها به یکدیگر هستند، فهرستی از عناصر و محلی که به کار گرفته شده‌اند و واژه‌نامه‌ای که به کل سیستم اعمال می‌شود.

۳. *چرا معماری به این صورت طراحی می‌شود*. در واقع هدف پاسخ‌دهی به سئوالاتی از این قبیل است: زمینه برای سیستم، محدودیت‌های خارجی که معماری را در یک روش خاص شکل می‌دهند و استدلال برای تصمیمات مقیاس بزرگ دانه درشت .

شکل ۹-۴ خلاصه‌ای از نکات ذکر شده را نشان می‌دهد.



شکل ۹-۴: خلاصه‌ای از مستند دید متقاطع

[1] Cross-view documentation
[2] View catalog
[3] View template

۹-۵-۱- چگونگی سازماندهی مستند برای ارائه به ذینفعان

هر مجموعه از مستندات معماری نیاز به یک بخش مقدمه دارد که سازماندهی آن را برای ذینفعان جدید تشریح کند و همچنین کمک کند که ذینفعان به اطلاعات مورد علاقه خود به سادگی دسترسی داشته باشند. دو نوع از اطلاعات "چگونگی" وجود دارد که کاتالوگ دید و قالب دید نامیده می‌شوند.

۹-۵-۱-۱- کاتالوگ دید

کاتالوگ دید مقدمه‌ای برای خوانندگان دیدها است که معمار برای یک مجموعه از مستندات[1] در نظر گرفته است. زمانی که یک مجموعه مستندات به عنوان مبنایی برای ارتباطات در نظر گرفته می‌شود برای یک خواننده جدید ضروری است که مشخص شود کجا اطلاعات خاص مورد نیاز خود را می‌تواند پیدا کند. کاتالوگ شامل این اطلاعات است. زمانی که مجموعه مستندات به عنوان پایه‌ای برای تحلیل در نظر گرفته می‌شود، ضروری است که مشخص شود هر دید شامل کدام اطلاعات مورد نیاز تحلیل‌گر است. برای مثال، در تحلیل کارایی مصرف منبع قسمت مهمی از اطلاعات است بنابراین کاتالوگ، تحلیل‌گر را قادر می‌سازد که مشخص کند کدام دیدها دربردارنده خصوصیات مناسب برای مصرف منبع هستند. در کاتالوگ دید برای هر دید مشخص شده در مجموعه مستند، یک مدخل وجود دارد که هر مدخل باید شامل اطلاعات زیر باشد:

۱. نام دید و نوع سبک معرفی[2] دید
۲. توصیفی از خصوصیات، نوع ارتباطات و نوع عناصر استفاده شده در دید
۳. توصیفی از اینکه به چه منظور دید ایجاد شده است (استفاده می‌شود)
۴. مدیریت اطلاعات[3] مربوط به مستند دید از قبیل آخرین نسخه، محل و مالک[4] مستند دیدها

۹-۵-۱-۲- قالب دید

قالب دید یک سازماندهی استاندارد برای دید است. شکل ۹-۱ و موضوعات پیرامون آن پایه‌ای برای قالب دید (از طریق تعریف بخشهای استاندارد مستند دید، محتوا و قوانین) فراهم می‌کنند. هدف از قالب دید آن است که کمک شود:

[1] Suite of documentation
[2] Instantiate
[3] Management information
[4] Owner

- خوانندگان به راحتی به بخش مورد نظر خود دسترسی داشته باشند،
- نویسندگان اطلاعات را سازماندهی کنند و
- معیارهای ایجاد شود تا مشخص شود چقدر کار باقی مانده است.

۹-۵-۲- معماری چه چیزی است

این بخش اطلاعاتی در مورد چگونگی مستندسازی معماری، ارتباطات دیدها با یکدیگر و شاخصی برای عناصر معماری، فراهم می‌کند.

۹-۵-۲-۱- دیدکلی سیستم

این بخش توصیفی کوتاه در مورد این موضوعات است که عملکرد سیستم چیست و کاربران سیستم چه کسانی هستند. همچنین دربرگیرنده هر گونه اطلاعات و محدودیت مرتبط با سیستم است. در واقع هدف آن است که یک مدل سازگار فکری از سیستم و اهداف آن برای خواننده ایجاد شود.

۹-۵-۲-۲- نگاشت بین دیدها

از آنجایی که کلیه دیدهای معماری، یک سیستم را تشریح می‌کنند بنابراین قابل استدلال است که هر دو دید متفاوت اشتراکاتی با یکدیگر داشته باشند. کمک کردن به خواننده مستند به منظور درک ارتباطات بین دیدها باعث افزایش قدرت درون‌یابی آن پیرامون چگونگی کار کردن سیستم به صورت یک مفهوم واحد می‌شود. واضح کردن ارتباط بین دیدها ونگاشت بین آنها یک عامل اصلی در افزایش قابلیت درک و کاهش ابهامات است. برای مثال هر ماژول می‌تواند به چندین عنصر در زمان اجرا نگاشت شود (زمانی که کلاس به شیء نگاشت داده می‌شود). زمانی که نگاشت یک به یک نباشد یا عناصر زمان اجرا به صورت یک عنصر کدی[1] وجود نداشته باشد، پیچیدگی افزایش پیدا می‌کند.

۹-۵-۲-۳- فهرست عنصر

فهرست عنصر شاخص ساده‌ای است که عناصر استفاده شده در هر دید را نشان می‌دهد. همچنین ارجاع به محلی دارد که عناصر در آنجا تعریف شده‌اند. فهرست باعث می‌شود که عمل جستجوی دیدها با سرعت بالایی انجام شود.

[1] Code

۹-۵-۲-۴- واژه‌نامه پروژه

فهرست واژه‌گان و تعاریف عبارت‌ها برای هر سیستم منحصر به فرد است و معنا و مفهوم خاص خودش را دارد. وجود یک فهرست از اختصارات و معنی هر کدام علاوه بر فهرست واژگان می‌تواند برای ذینفعان مفید باشد. همچنین اگر از قبل یک واژنامه وجود داشته باشد باید یک ارجاع به آن صورت بپذیرد.

۹-۵-۳- دلایل طراحی معماری به صورت خاص

مشابه با استدلال پیرامون دیدها و طراحی واسطها، استدلال پیرامون دید متقاطع توصیف می‌کند که چگونه معماری یک راه حل برای نیازمندیهای سیستم است. یک استدلال پیرامون این نوع دیدها می‌تواند موارد زیر را تشریح کند:

- اشاره به تصمیمات اخذ شده در کل سیستم برای برآوردن نیازمندیها و محدودیت‌ها (اجبارها)
- تاثیرات معماری در هنگام افزودن نیازمندی از قبل پیش بینی شده یا تغییر یکی از نیازمندیهای فعلی
- اجبارهایی که توسعه‌دهنده در پیاده‌سازی خواهد داشت
- دیگر تصمیماتی که رد شده‌اند

۹-۶- استفاده از *UML*

معماری بیان می‌کند که چه چیزهایی برای یک سیستم نرم‌افزاری ضروری هست و البته ماهیت آن مستقل از زبان و نمادها است. با این وجود امروزه *UML* به عنوان نماد استاندارد برای مستندسازی معماری نرم‌افزار در نظر گرفته می‌شود. بنابراین *UML* به عنوان اصلی‌ترین مفهوم در نمایش اولیه و تشریح رفتار عناصر یا گروهی از عناصر است. استاندارد *UML* به طور مستقیم از مولفه، اتصال، لایه‌بندی، واسط معنایی یا دیگر جنبه‌های سیستم که معمارانه هستند، حمایت نمی‌کند. ولی با این حال می‌توان از ساختارهای ارائه شده به وسیله *UML* برای مدلسازی و تشریح سیستم استفاده کرد.

۹-۶-۱- دیدهای ماژول

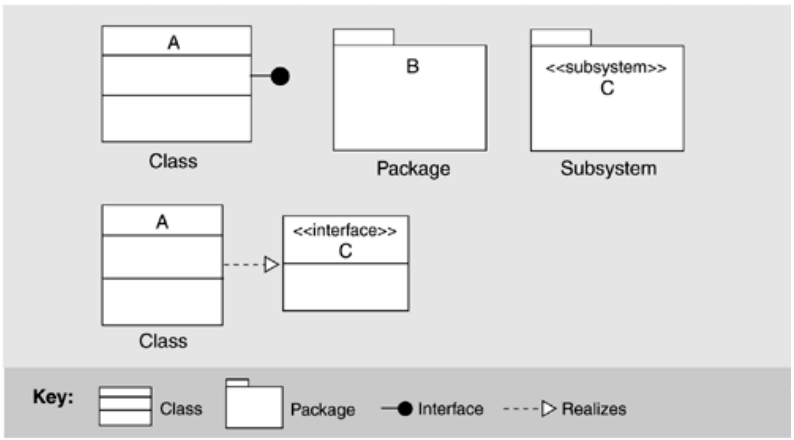
ماژول یک واحد کد یا پیاده‌سازی است و دید ماژول نیز تعدادی از ماژولها با واسطها و دیگر رابطه‌های موجود بین آنها است.

۹-۶-۱-۱- واسطها

شکل ۹-۲ دیدیم که چگونه یک واسط می‌تواند با استفاده از *UML* نمایش داده شود. *UML* از دایره کوچک برای نمایش واسط استفاده می‌کند. واسطها به کلاسها، زیرسیستمها و دیگر عناصر مناسب در سیستم الحاق می‌شوند. *UML* همچنین اجازه می‌دهد که نماد کلاس (مستطیل) به عنوان یک واسط، کلیشه[1] شود. پیکان خط‌چین در *UML* نشان می‌دهد که یک عنصر، واسطی را تحقق می‌بخشد. بخش پائین نماد کلاس می‌تواند برای ذخیره اطلاعات مربوط به امضا واسط، نام رویه‌ها و پارامترها استفاده شود. دایره کوچک معمولا برای نمایش وابستگی عنصر به واسط به کار می رود در حالی که نماد مستطیل اجازه می‌دهد که اطلاعات بیشتری در مورد گرامر واسطها از قبیل عملیات فراهم شده، ذکر شود.

۹-۶-۱-۲- ماژولها

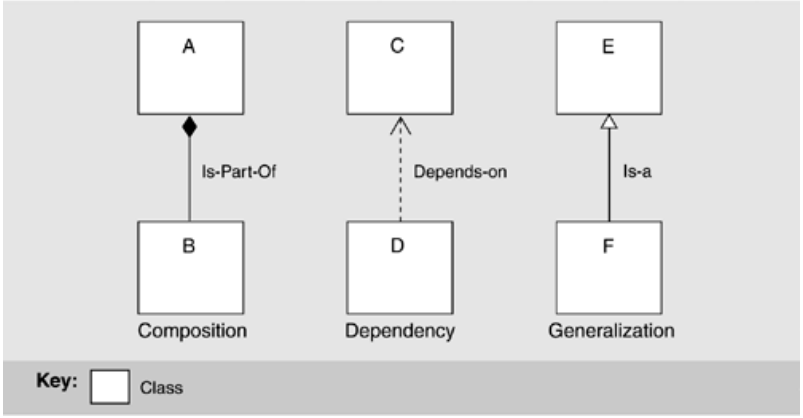
UML ساختارهای متنوعی را برای نمایش انواع مختلف ماژولها فراهم می‌کند. شکل ۹-۵ بعضی مثالها در مورد ساختارهای *UML* را نشان می‌دهد. *UML* یک ساختار کلاس دارد که مشخصات شیءگرایی یک ماژول را نشان می‌دهد. بسته‌ها زمانی که یک گروه از عملکردها مدنظر است مورد استفاده قرار می‌گیرند (مانند نمایش لایه‌ها و کلاسها). ساختار زیرسیستم نیز زمانی استفاده می‌شود که به اطلاعات بیشتر در مورد واسط یا رفتار یک کلاس نیاز است.



شکل ۹-۵: مثالی از نمادهای ماژول در *UML*

[1] Stereotyped

شکل ۹-۶ رابطه‌های موجود در دید ماژول را نمایش می‌دهد. تجزیه ماژول متکی بر رابطه "بخشی از"[1] است. دید استفاده ماژول متکی بر رابطه وابستگی و دید کلاس ماژول نیز متکی بر رابطه تعمیم یا "نوعی است از"[2] است.



شکل ۹-۶: مثالهایی از نمادهای استفاده شده در *UML*

- تجمیع[3]

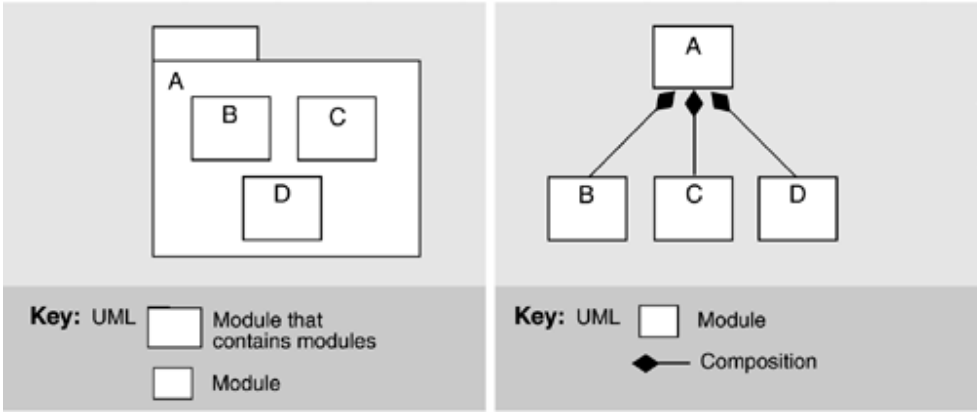
در *UML* ساختار زیرسیستم را می‌توان برای نمایش ماژولی که شامل ماژولهای دیگری است، استفاده کرد. مستطیل کلاس معمولا برای برگ‌های تجزیه به کار می‌رود. زیرسیستمها می‌توانند به عنوان بسته تجزیه شوند و بنابراین برای ماژول تجمیع مناسب هستند. همچنین زیرسیستمها به عنوان دسته‌بندی کننده[4] می‌توانند محتویات خود را محصورسازی کرده و واسطهای صریحی را فراهم کنند. رابطه تجمیع به وسیله سه روش در *UML* نشان داده می‌شود:

۱. *ماژول می‌تواند تودرتو باشد (شکل ۹-۷ سمت چپ)*
۲. *ترکیبی از دو نمودار باشد به طوری که دومی محتویات اولی را نشان دهد*
۳. *یک کمان که رابطه بین فرزندان و والد را نشان می‌دهد (شکل ۹-۷ سمت راست)*

در *UML*، ترکیب شکلی از تجمیع است. یعنی اینکه بخشهای مختلف با یکدیگر ایجاد یا از بین می‌روند. به عنوان مثال اگر ماژول A از ماژولهای B و C تشکیل شده باشد و B و C نتوانند بدون A

[1] Is-part-of
[2] Is-a
[3] Aggregation
[4] Classifier

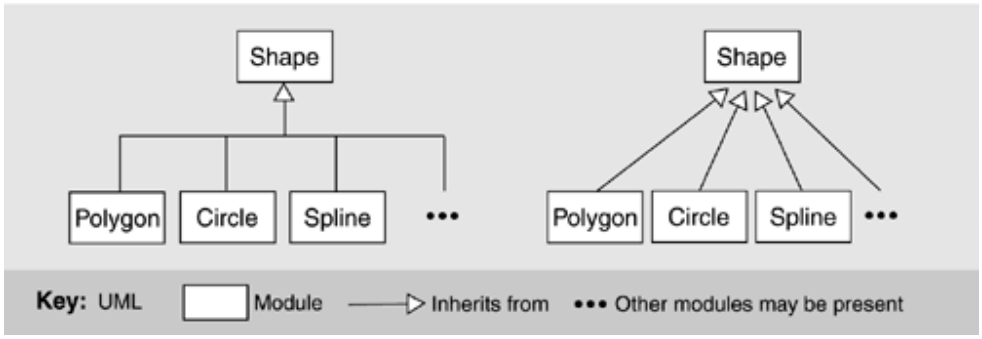
وجود داشته باشند در نتیجه اگر A در زمان اجرا از بین برود B و C نیز از بین خواهند رفت. پس رابطه ترکیب *UML* فراتر از ساختار مربوط به واحدهای پیاده‌سازی است.



شکل ۹-۷: مثالهایی از انواع رابطه‌های تجمیع

- تعمیم[1]

زمانی که ماژولها در *UML* به صورت نمودار کلاس نشان داده می‌شوند رابطه تعمیم نقش بسیار مهمی را ایفا می‌کند. شکل۹-۸ نمادهای پایه موجود در *UML* را برای نشان دادن رابطه تعمیم به تصویر کشیده است.



شکل ۹-۸: مستندسازی رابطه تعمیم در *UML*

دو نمودار نشان داده شده در شکل ۹-۸ از نظر معنایی با یکدیگر برابر هستند. *UML* اجازه می‌دهد که از نماد "..." به جای زیر ماژول استفاده کرد. این نماد نشان می‌دهد که ماژول می‌تواند بیش از تعداد نشان داده شده، زیر ماژول داشته باشد. همان طور که در شکل مشخص است ماژول *Shape*

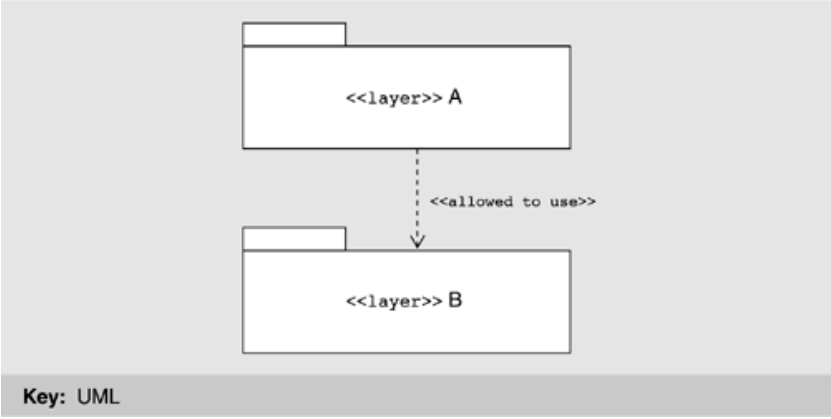
[1] Generalization

پدر *Polygon, Cirle* و *Spline* است و هر کدام از اینها نیز یک زیرکلاس یا فرزند *Shape* هستند. *Shape* حالت عمومی دارد در حالی که سایر ماژولها یک نسخه خاص از آن هستند.

- *وابستگی*

نماد پایه برای وابستگی در شکل ۹-۶ نشان داده شده است. مهمترین محل نمایش رابطه وابستگی در لایه‌ها است. متاسفانه *UML* نماد پایه‌ای خاص داخلی برای نمایش لایه‌ها ندارد. با این حال می‌توان لایه‌بندی ساده را با استفاده از بسته‌ها نشان داد (شکل ۹-۹). در واقع بسته‌ها مکانیزم عمومی برای سازماندهی عناصر در داخل گروه‌ها هستند. به صورت پیش فرض *UML* بسته‌ها را برای سیستم و زیرسیستمها تعریف کرده است ولی می‌توان بسته را برای لایه نیز تعریف کرد (یعنی از کلیشه سازی بسته‌ها استفاده شود).

یک لایه می‌تواند به صورت یک بسته *UML* همراه با محدودیت‌های موجود بین ماژولها و رابطه وابستگی بین آنها نشان داده شود. رابطه وابستگی در این حالت رابطه "اجازه استفاده"[1] است. در واقع لایه به صورت بسته‌ای تعریف می‌شود که در روی آن کلیشه *<<layer>>* قرار می‌گیرد (البته می‌توان نماد جدیدی نیز برای آن تعریف کرد).

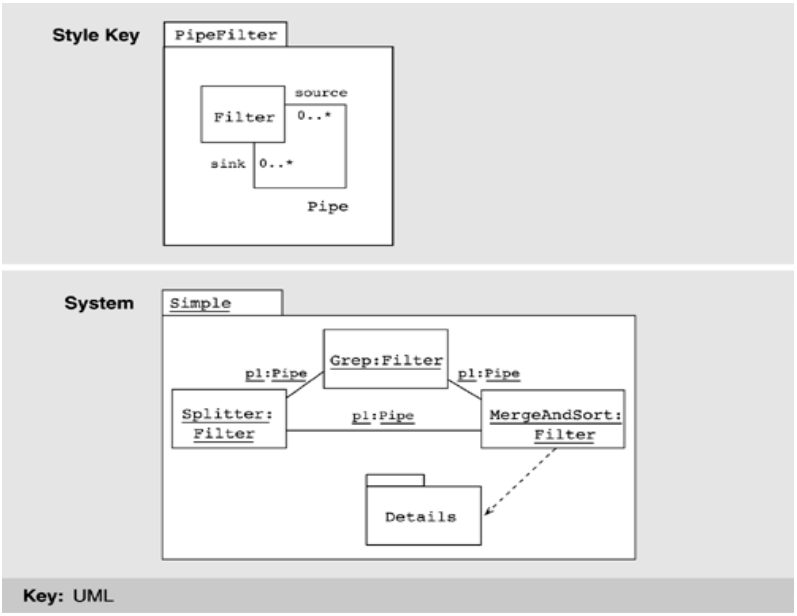


شکل ۹-۹: نمایش لایه در *UML*

[1] Allowed to use

۹-۶-۲- دیدهای مولفه و اتصال

هیچ راهبرد خاص از پیش تعریف شده‌ای برای نمایش دید مولفه و اتصال در *UML* وجود ندارد اما روشهایی برای نمایش آن در *UML* وجود دارد. هر کدام از این روشها مزایا و معایب خاص خودشان را دارند. یک روش طبیعی برای نمایش انواع دید مولفه و اتصال، استفاده از مفهوم کلاسها در *UML* است.شکل ۹-۱۰ ایده عمومی این روش را نشان می‌دهد که از سیستم لوله و فیلتر استفاده می‌شود. در شکل ۹-۱۰ نوع معماری فیلتر به عنوان کلاس *Filter* در نظر گرفته شده است. نمونه‌های فیلترها (از قبیل *Splitter*)، اشیاء موجود در نمودار شیء را نشان می‌دهند. برای مشخص کردن یک فضای نام مرزی[1] توضیحات در درون بسته قرار می‌گیرد. در شکل ۹-۱۰ نمایش *MergeAndSort* که جزئیات را نشان می‌دهد به صورت یک بسته در نظر گرفته شده است.



شکل ۹-۱۰:انواع به عنوان کلاسها و نمونه‌ها به عنوان اشیاء

۹-۶-۲-۱- مولفه‌ها

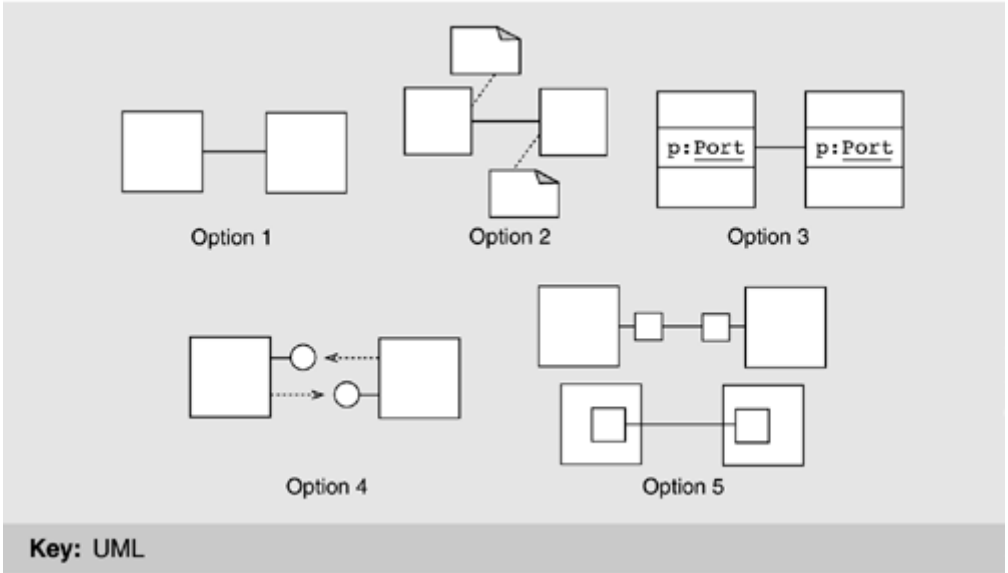
رابطه نوع/نمونه در توصیفات معماری شباهت زیادی به رابطه کلاس/شیء در مدلسازی *UML* دارد. کلاسهای *UML*، مشابه انواع مولفه‌ها در توصیفات معماری، موجودیت‌های رتبه اول[2] هستند و ساختارهای

[1] Namespace boundry
[2] First-class

غنی برای شناسایی تجریدات نرم‌افزار تلقی می‌شوند. مجموعه کاملی از مکانیزمهای توصیف مبتنی بر *UML* که برای توصیف ساختارها، خصوصیات و رفتارهای کلاسها وجود دارند، باعث می‌شود *UML* انتخاب مناسب برای تشریح جزئیات و تحلیل سیستم باشد. خصوصیات مولفه‌های معماری می‌تواند به صورت ویژگیهای کلاس یا با استفاده از رابطه‌های انجمنی[1] نشان داده شود. رفتارها می‌توانند با استفاده از مدلهای رفتاری *UML* نشان داده شوند. تعمیم نیز می‌تواند به منظور مرتبط کردن انواع مولفه‌ها استفاده شود. معنای نوع و نمونه می‌تواند همچنین از طریق به کارگیری کلیشه ایجاد شود. برای مثال، کلیشه *<<process>>* می‌تواند به یک مولفه متصل شود تا نشان دهد که آن مولفه می‌تواند به صورت یک مولفه مجزا اجرا شود.

۹-۶-۲-۲- واسطها

واسطهای مولفه‌ها را که گاها درگاه نیز می‌نامند به پنج روش می‌توان نشان داد. این پنج روش در شکل ۹-۱۱ به ترتیب افزایش قابلیت بیان[2] نشان داده شده‌اند. با این حال زمانی که قابلیت بیان منجر به پیچیدگی زیاد می‌شود باید نخستین راهبردی که اهداف مورد نظر را برآورده می‌کند، انتخاب کرد.



شکل ۹-۱۱: پنج روش برای نمایش واسطهای مولفه‌ها

[1] Associations
[2] Expressiveness

- *گزینه اول: بدون نمایش صریح*

صرف نظر کردن از نمایش صریح واسط‌ها منجر به یک نمودار ساده می‌شود ولی این نوع نمایش از مشکل مربوط به عدم مشخص کردن نام و خصوصیات واسط‌ها در نمایش اولیه رنج می‌برد. با این حال این روش هنوز زمانی که مولفه‌ها یک واسط دارند یا از طریق موقعیت مکانی سیستم قابل تشخیص هستند و یا اینکه نمودار در محل دیگری بهبود خواهد یافت، قابل درک و استدلال است.

- *گزینه دوم: واسط‌ها به عنوان یاداشت*[1]

نمایش واسط‌ها به صورت یاداشت یک محلی برای اطلاعات تکمیلی در مورد واسط‌ها فراهم می‌کند. با توجه به اینکه یاداشت هیچ ارزش معنایی در *UML* ندارد بنابراین نمی‌تواند به عنوان پایه تحلیل در نظر گرفته شود. در این روش اگر جزئیات خصوصیات مدنظر نباشد می‌توان در مورد نمودار ایجاد شده استدلال کرد.

- *گزینه سوم: واسط‌ها به عنوان خصوصیات شیء/کلاس*

در نظر گرفتن واسط‌ها به صورت ویژگیهای کلاس/شیء باعث می‌شوند که آنها بخشی از مدل ساختاری رسمی شوند اما آنها فقط می‌توانند یک نمایش ساده در نمودار کلاس داشته باشند (یعنی یک نام و یک نوع). این محدودیت باعث کاهش قابلیت بیان واسط‌ها می‌شود.

- *گزینه چهارم: واسط‌ها به عنوان واسط‌های UML*

دایره به همراه یک خط که به کلاس وصل می‌شود، تداعی کننده واسط است این نوع نمایش، یک توصیف فشرده از واسط را در نمودار کلاس فراهم می‌کند. در یک نمونه نمودار، نقش انجمنی *UML* یک روش فشرده‌ای را ارائه می‌کند که به وسیله آن می‌توان نشان داد که یک نمونه مولفه از طریق یک نمونه واسط خاص با مولفه‌های دیگر ارتباط برقرار می‌کند.

این روش از تصاویر گرافیکی مجزا برای مولفه و واسط استفاده می‌کند که باعث می‌شود به راحتی بتوان واسط‌ها و مولفه‌ها را درک کرد. با این وجود، این راهبرد روشی برای به تصویر کشاندن سرویسهای مورد نیاز از محیط ارائه نمی‌کند (اغلب یک بخش مهم در واسط به شمار می‌رود). بعلاوه برای حالتی که یک نوع مولفه چندین نمونه از یک نوع واسط دارد قابل فهم است اما برای حالتی که

[1] Annotation

در آن یک کلاس چندین نسخه از یک واسط *UML* را تحقق می‌بخشد قابل درک نیست. برای مثال، هیچ روش ساده‌ای با استفاده از این راهبرد برای تعریف نوع فیلتر *Splitter* که شامل دو درگاه خروجی از یک نوع باشد، وجود ندارد. نهایتا اینکه برخلاف کلاسها واسط‌های *UML* ویژگی یا زیرساختار ندارند.

- *گزینه پنجم: واسط‌ها به عنوان کلاس*

توصیف واسط‌ها به صورت کلاسهایی که شامل نوع مولفه هستند ضعف قابل بیان بودن روش قبلی را از بین می‌برد. بنابراین با چنین روشی می‌توان زیرساختار واسط را تعریف کرد و همچنین مشخص کرد که یک نوع مولفه شامل چندین واسط از نوع یکسان هستند. یک نمونه مولفه به صورت یک شیء دربرگیرنده مجموعه‌ای از اشیاء واسط مدلسازی می‌شود. با این حال به وسیله نمایش واسط‌ها به صورت کلاس علاوه بر پیچیده کردن نمودار، تفکیک‌پذیر بودن تصویری واسط‌ها و مولفه‌ها نیز از بین می رود. همچنین می‌توان از متغییر نمادین برای نشان دادن اینکه واسط‌ها شامل مولفه‌ها هستند، استفاده کرد. این موضوع در بخش پائین گزینه پنجم در شکل ۹-۱۱ نشان داده است.

۹-۶-۲-۳- اتصالات

سه گزینه قابل استدلال برای نمایش اتصالات وجود دارد که عبارتند از:

- *گزینه اول: انواع اتصالات به عنوان انجمنها و نمونه‌های اتصالات به عنوان رابط‌ها*[1]

در نمودار معمارانه خط و جعبه[2] از سیستم، خط‌های بین مولفه‌ها، اتصالات هستند. یک روش برای نمایش اتصالات در *UML* استفاده از انجمنها بین کلاسها و رابط‌ها بین اشیاء است. این روش از نظر گرافیکی ساده و یک تمایز واضح بین مولفه‌ها و اتصالات ایجاد می‌کند و از رابطه آشنا در نمودار کلاس *UML* یعنی رابطه انجمنی استفاده می‌کند. علاوه بر اینها رابطه انجمنی می‌تواند برچسب شود.

متاسفانه اتصالات و انجمن‌ها معنای مختلفی دارند. یک سیستم در توصیف معماری به صورت یک سری مولفه‌ها با رفتارهایی که در واسط‌های آنها ظاهر شده و آنها را به اتصالاتی که مسئول هماهنگی

[1] Link
[2] Box-and-line

رفتارها هستند مرتبط می‌کند، نمایش داده می‌شود. رفتار سیستم نیز به صورت رفتار جمعی از یک مجموعه از مولفه‌ها تعریف می‌شود که تعاملات بین آنها به وسیله ارتباطات تعریف و محدود شده‌اند. در مقابل اگر چه رابط و انجمن در *UML* تعاملات بالقوه بین عناصر مرتبط را نشان می‌دهند ولی مکانیزم انجمنی یک روش اصلی برای توصیف ارتباط مفهومی بین دو عنصر است. علاوه بر این، رابطه انجمنی یک رابطه بین عناصر است، بنابراین نمی‌تواند بر روی خودش در مدل *UML* قرار بگیرد. همچنین این روش اجازه نمی‌دهد که یک نمونه اتصال را مشخص کرد.

- *گزینه دوم: انواع اتصالات به عنوان کلاسهای انجمنی*

یک راه حل برای ضعف قابل بیان بودن آن است که همراه رابطه انجمنی از یک کلاس استفاده شود تا نوع اتصالات را نشان دهد. در این روش نوع اتصال یا ویژگیهای اتصال می‌تواند به عنوان ویژگیهای کلاس یا شیء در نظر گرفته شود. متاسفانه این روش تکنیکی برای نمایش صریح واسط اتصالات فراهم نمی‌کند.

- *گزینه سوم: انواع اتصالات به کلاسها و نمونه‌های اتصالات به عنوان اشیاء*

یک روش برای اینکه به اتصالات در *UML* رتبه اول داده شود آن است که نوع اتصالات را با کلاسها و نمونه اتصالات را با اشیاء نشان داد. استفاده از کلاسها و اشیاء باعث می‌شود که همان چهار گزینه برای نمایش نقش‌ها برای واسطها به وجود آید. مشخص کردن طرحی برای نمایش واسط و یک الصاق بین واسط مولفه و واسط اتصال می‌تواند به صورت رابطه یا انجمن نمایش داده شود.

۹-۶-۲-۴- سیستمها

علاوه بر نمایش جداگانه مولفه‌ها و اتصالات و انواع آنها، نیاز است که گراف مولفه‌ها و اتصالات را محصورسازی کرد. بدین منظور سه روش وجود دارد که به شرح زیر هستند:

- *گزینه اول: سیستمها به عنوان زیرسیستمها*

مکانیزم پایه *UML* برای گروه‌بندی عناصر مرتبط به هم، بسته‌بندی[1] است. در حقیقت، *UML* یک کلیشه بسته‌بندی استاندارد به نام <<*subsystem*>> تعریف می‌کند تا مدلهای *UML* را که بخشهای منطقی سیستم را نمایش می‌دهند، گروه‌بندی کند. یکی از مشکلات استفاده از زیرسیستم، که در

[1] Package

UML 1.4 تعریف شده آن است که معنای آنها کاملا واضح نیست. بعضی‌ها معتقد هستند که یک زیرسیستم را باید به صورت یک موجودیت اتمیک شبیه کلاس در یک مرحله خاص از فرآیند توسعه در نظر گرفت که بعدا امکان بهبود زیرسیستم با جزئیات بیشتر وجود داشته باشد. داشتن یک چنین قابلیتی باعث می‌شود که یک ساختار زیرسیستم ایجاد شود که کاملا مناسب برای مدلسازی مولفه‌های معماری است.

- *گزینه دوم: سیستمها به عنوان اشیاء کامل*[1]

اشیاء کامل می‌توانند برای نمایش سیستمها استفاده شوند. در این روش مولفه‌ها به صورت نمونه‌هایی از کلاسهای کامل نمایش داده می‌شوند و اتصالات با استفاده از گزینه‌های تشریح شده (در قسمت قبل) مدلسازی می‌شوند. اشیاء یک مرز محصورسازی قوی فراهم می‌کنند و با آن این مفهوم را تداعی می‌کنند که هر نمونه از کلاس به زیرساختار مرتبط است. با این حال این روش مشکلاتی دارد. مهمترین آنها در ارتباط با انجمن‌ها است که برای مدلسازی اتصالات به کار می‌رود. انجمن‌های بین کلاسهای کامل به وسیله کلاسها پوشش داده نمی‌شود. یعنی اینکه امکان‌پذیر نیست که گفته شود یک جفت کلاس در یک زمنیه خاص از سیستم از طریق یک اتصال خاص با یکدیگر ارتباط برقرار می‌کنند و یا به صورت یک انجمن مدل می‌شوند.

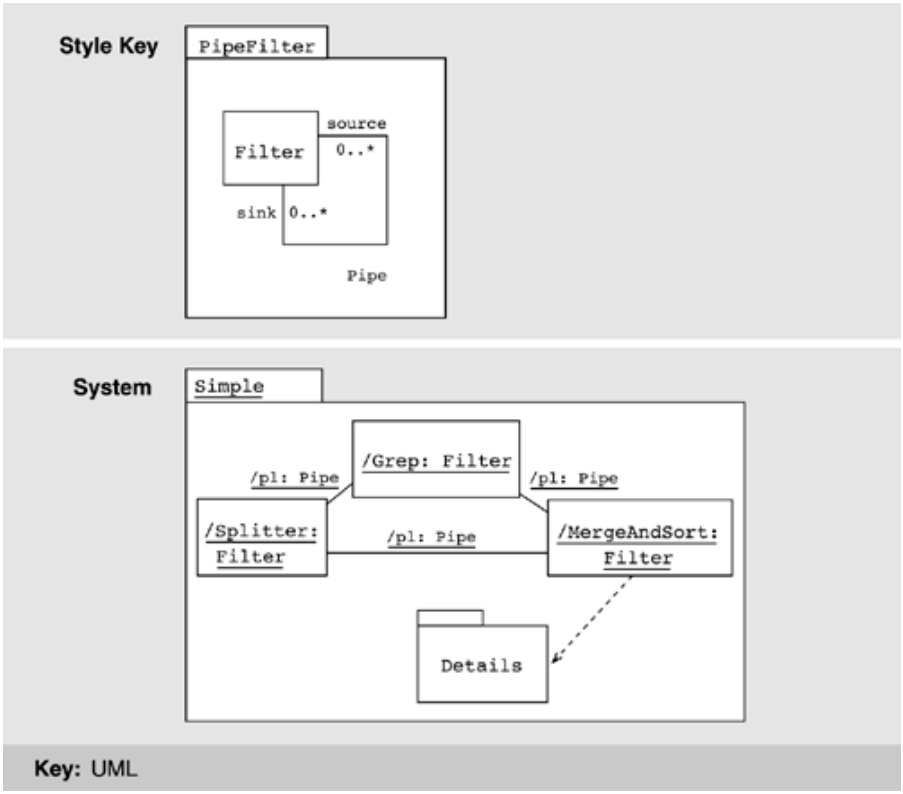
- *گزینه سوم: سیستمها به عنوان همکاران*[2]

یک مجموعه از اشیاء که به وسیله رابط به یکدیگر متصل شده‌اند در *UML* با استفاده همکاری تشریح می‌شود. اگر مولفه‌ها به صورت اشیاء نمایش داده شوند می‌توان از همکاری برای نمایش سیستم استفاده کرد. همکاری مجموعه‌ای از رابط‌ها و شرکت‌کنندگان را تعریف می‌کند که برای یک هدف مشخص معنی‌دار هستند.

در این حالت همکاری در واقع ساختار زمان اجرای سیستم را نشان می‌دهد. شرکت کنندگان، نقشهای دسته‌بندی کننده‌ای که اشیاء در زمان تعامل ایفا می‌کنند را تعریف می‌کنند. به همین صورت رابطه‌ها نقشهای انجمنی که رابط‌ها باید با آنها مطابقت داشته باشند را تعریف می‌کنند.

[1] Contained objects
[2] Collaborations

نمودار همکاری می‌تواند برای نمایش همکاران در سطح نمونه یا مشخصات استفاده شود، نمودار همکاری سطح مشخصات[1] نقشها را تعریف می‌کند تا از طریق آن بتواند زیرساختارهای سیستم را تشریح کند و نهایتا اینکه نمودار همکاری سطح نمونه اشیاء و رابطهایی که با نقشهای سطح مشخصات مطابقت دارند و به منظور دستیابی به اهداف سیستم تعامل می‌کنند را نشان می‌دهد. بنابراین همکاری نمایش داده شده در سطح نمونه بهترین روش برای نمایش ساختار زمان اجرای سیستم است. شکل ۹-۱۲ این روش را نشان می‌دهد.



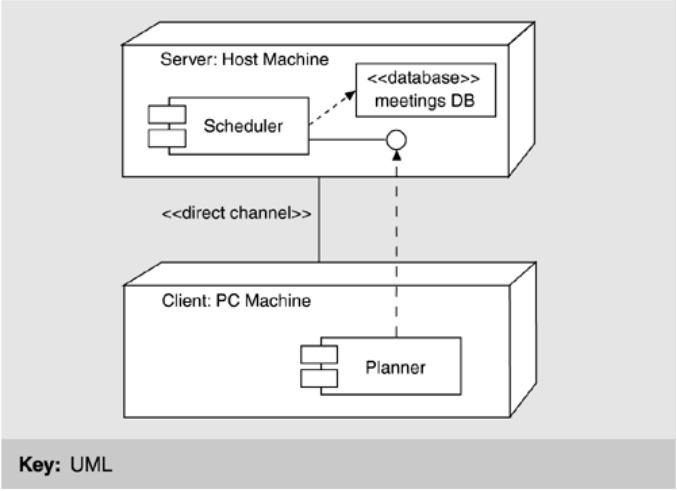
شکل ۹-۱۲: سیستمها به عنوان همکاران

اگرچه این روش یک روش طبیعی برای به تصویر کشیدن ساختار زمان اجرا است ولی هیچ روشی برای نمایش صریح سطح مشخصات فراهم نمی‌کند. همچنین یک اشتباه معنایی نیز وجود دارد و آن زمانی است که همکاری یک نمونه تعامل بین اشیاء را توصیف می‌کند (یعنی آن توصیف جزئی فراهم می‌کند) در حالی که یک پیکربندی معماری باید شامل یک توصیف کامل باشد.

[1] Specification-level

۹-۶-۳- دیدهای تخصیص

در *UML* نمودار استقرار یک گراف از گره‌های مرتبط شده به وسیله رابطه انجمنی هستند. شکل ۹-۱۳ یک مثال در این مورد را نشان می دهد. گره‌ها می‌توانند نمونه مولفه‌ها در نظر گرفته شوند که نشان دهنده آن است که مولفه‌ها بر روی گره‌ها در حال اجرا هستند. مولفه‌ها می‌توانند شامل اشیاء باشند که نشان دهنده آن است که اشیاء بخشی از مولفه‌ها هستند. مولفه‌ها می‌توانند با استفاده از پیکان خط چین که نشان دهنده وابستگی است به یکدیگر متصل شوند. این موضوع نشان می‌دهد که یک مولفه از سرویس‌های مولفه‌های دیگر استفاده می‌کند. همچنین یک کلیشه می‌تواند در این دید استفاده شود که نشان دهنده آن است که یک وابستگی دقیق مورد نیاز است. نمودار نوع استقرار همچنین می‌تواند با استفاده از پیکان خط چین با کلیشه <<*support*>> نشان دهد که کدام مولفه‌ها بر روی کدام گره‌ها اجرا می‌شوند.



شکل ۹-۱۳: دید استقرار در *UML*

گره یک شیء فیزیکی زمان اجرا است که یک منبع پردازشی را نشان می‌دهد که دست کم حافظه و قابلیت پردازشی دارد. گره‌ها دربردارنده دستگاه‌های محاسباتی، منابع پردازشی انسانی و مکانیکی هستند. گره‌ها می‌توانند انواع نمونه‌ها را نشان دهند. نمونه‌های محاسباتی زمان اجرا (مولفه‌ها و اشیاء) می‌توانند در نمونه گره مقیم باشند. گره‌ها می‌توانند با استفاده از رابطه‌های انجمنی به یکدیگر متصل شوند. رابطهای انجمنی اشاره به مسیر ارتباطی بین آنها دارد. رابطه انجمنی می‌تواند شامل کلیشه باشد که این موضوع ماهیت مسیر ارتباطی را نشان می‌دهد (به عنوان مثال نوع کانال یا شبکه).

فهرست مطالب

فصل دهم

بازسازی مجدد معماری

در این فصل به بررسی روش بازسازی مجدد معماری نرم‌افزار پرداخته خواهد شد. بدین منظور ابتدا فعالیت‌های اصلی بازسازی مجدد معرفی خواهند شد و سپس هر فعالیت به صورت کامل همراه با یک مثال تشریح می‌شود. همچنین برای هر فعالیت یک سری راهنماهای کاربردی ارائه خواهد شد که پیروی از آنها منجر به موفقیت فرآیند بازسازی مجدد معماری می‌شود.

۱۰-۱ مقدمه

تا قبل از این فصل به معماری به عنوان یک موجودیت بزرگ که تحت کنترل معمار است پرداخته شد و نشان داده شد که چگونه تصمیمات معماری را در جهت دستیابی به اهداف و نیازمندیهای سیستم در فرآیند توسعه اتخاذ کرد. اما تصویر دیگری از معماری وجود دارد که در این فصل به آن پرداخته خواهد شد. فرض کنید یک سیستم توسعه یافته وجود دارد که هیچ شناختی در مورد معماری آن وجود ندارد. با توجه به این موضوع احتمالا هیچ معماری به وسیله توسعه‌دهندگان برای سیستم ایجاد نشده یا احتمالا معماری ایجاد شده ولی مستند مربوط به معماری گم شده و یا معماری ایجاد شده و مستندی هم برای آن وجود دارد ولی مستند به روز نیست (یعنی تغییراتی در سیستم ایجاد شده ولی در مستند معماری اعمال نشده‌اند). حال سئوال این است که چگونه این سیستم را باید نگهداری کرد؟ چگونه تکامل و توسعه سیستم را مدیریت کرد تا خصوصیات کیفی که سیستم از آن پشتیبانی می‌کند را بتوان حفظ کرد؟

این فصل در ارتباط با چگونگی پاسخ دادن به این سئوالات با استفاده از بازسازی مجدد معماری[1] است. این عمل از طریق تحلیل کامل سیستم با استفاده از ابزار انجام می‌شود. ابزار اطلاعات مربوط به سیستم را استخراج کرده به جمع‌آوری و ایجاد سطوح متوالی از تجریدها[2] کمک می‌کند. اگر ابزار به صورت موفق عمل کند خروجی نهایی یک نمایشی از معماری خواهد بود که به استدلال پیرامون سیستم کمک می‌کند. البته در بعضی موارد امکان‌پذیر نیست که بتوان یک نمایش سودمند را تولید کرد و این زمانی است که سیستم موروثی (عتیقه) بوده و طراحی معماری منسجمی برای آن سیستم وجود ندارد تا بتوان آن را بازیابی کرد.

بازسازی مجدد معماری یک فرآیند تعاملی[3]، تفسیری[4] و تکراری[5] است که دربرگیرنده فعالیت‌های زیادی است (این فرآیندها و فعالیت‌ها به صورت کاملا خودکار نیستند). بلکه نیاز به مهارت و توجه هر دوی معمار و شخص مهندس معکوس دارد زیرا ساختارهای معماری به صورت صریح در کد منبع ارائه نشده‌اند. هیچ ساختار زبان برنامه‌نویسی برای "لایه"، "اتصال" و یا سایر عناصر معماری وجود ندارد که بتوان به راحتی آنها را

[1] Architecture reconstruction
[2] Successive levels of abstraction
[3] Interactive
[4] Interpretive
[5] Iterative

از فایل کد منبع استخراج کرد. اگر داخل سیستم از الگوهای معماری استفاده شده باشد آنها به ندرت برچسب‌گذاری می‌شوند ولی ساختارهای سیستم به وسیله انواع مکانیزمها در یک پیاده‌سازی تحقق پیدا می‌کنند که معمولا هم به صورت یک مجموعه از توابع، کلاسها، فایلها، اشیاء وغیره هستند. هنگامی که سیستم برای اولین بار توسعه می‌یابد عناصر طراحی/معماری سطح بالای آن به عناصر پیاده‌سازی نگاشت می‌شوند بنابراین هنگامی که بازسازی مجدد این عناصر انجام می‌شوند باید عمل نگاشت معکوس انجام شود (به دست آوردن آن عناصر نیازمند درون‌بینی معماری است). به منظور انجام این عملیات آشنایی با تکنیک‌های ساخت کامپایلر و ابزارهایی مانند*grep, sed, awk, perl, python, and lex/yacc* و غیره می‌تواند مفید و سودمند باشد.

نتایج حاصل از بازسازی مجدد معماری می‌تواند در چندین روش مورد استفاده قرار بگیرد. زمانی که مستندی برای معماری وجود نداشته و یا اینکه مستند به روز نباشد، می‌تواند به عنوان پایه‌ای برای مستندسازی مجدد معماری استفاده شود (مانند آنچه که در فصل نهم ذکر شد)، می‌تواند برای بازیابی معماری درونی سیستم به منظور مطابقت آن با طراحی معماری نیز استفاده شود. این عمل تضمین می‌کند که توسعه‌دهندگان از قوانین معماری تعریف شده برای آنها و سیستم تبعیت کرده‌اند (یعنی باعث نقض پنهان‌سازی اطلاعات، انتزاع و غیره نشده‌اند). همچنین این روش می‌تواند به عنوان مبنایی برای تحلیل معماری و یا نقطه شروعی برای مهندسی مجدد سیستم در جهت دستیابی به معماری جدید مطلوب استفاد شود. نهایتا نمایش مجدد معماری می‌تواند برای شناسایی عناصر قابل استفاده مجدد و یا ایجاد یک خط تولید نرم‌افزار مبتنی بر معماری استفاده شود.

بازسازی مجدد معماری در پروژه‌های زیادی از جمله *MRI scanners, public telephone switches, helicopter guidance systems* و *classified NASA systems* استفاده شده است. همچنین برای سیستمهای زیر با هدف خاص نیز به کار رفته است:

- مستندسازی مجدد معماریها برای سیستمهای شبیه‌ساز فیزیکی
- درک وابستگیهای معماری در نرم‌افزارهای کنترل توکار[1] برای استخراج دستگاه‌ها
- ارزیابی مطابقت پیاده‌سازی سیستم ماهواره‌ای زمینی با معماری مرجع آن
- درک سیستمهای مختلف در صنعت خودروسازی

[1] Embedded

۱۰-۱-۱- روش *workbench*

بازسازی مجدد معماری نیاز به پشتیبانی ابزار دارد ولی همیشه یک ابزار منفرد یا یک مجموعه ابزار برای دستیابی به نتیجه موفق کافی نیست. یکی از دلایل این امر آن است که ابزارها مبتنی بر یک زبان خاص هستند در حالی که ممکن است در سیستمی که خواهان بازسازی مجدد معماری آن هستیم زبانهای مختلفی استفاده شده باشند. برای مثال یک اسکنر *MRI* می‌تواند متشکل از نرم‌افزاری باشد که با ۱۵ زبان نوشته شده باشد. دلیل دیگر نیز آن است که ابزارهای استخراج کننده داده کامل نیستند. آنها اغلب نتایج ناقص یا نادرستی را به عنوان خروجی تولید می‌کنند، بنابراین در چنین حالتی از چندین ابزار استفاده می‌شود تا بتوان خروجی آنها را با هم مقایسه کرد. نهایتا اینکه اهداف بازسازی مجدد خیلی متنوع است. در واقع چیزی که خواهان انجام آن با مستند بازسازی شده هستیم مشخص کننده اطلاعاتی خواهد بود که باید استخراج شوند و در این حالت نیز نیاز به ابزارهای مختلف وجود دارد.

در نظر بگیرید که کلیه این موارد منجر به یک فلسفه طراحی خاص به نام میزکار[1] برای مجموعه ابزار پشتیبانی کننده بازسازی مجدد معماری شود. میزکار باید باز[2] بوده و یک چارچوب یکپارچه سبک وزن فراهم کند تا به وسیله آن بتوان ابزارها را به مجموعه ابزار میزکار اضافه کرد بدون اینکه ابزارهای موجود یا داده‌های غیرضروری تحت تاثیر قرار بگیرند. به عنوان یک مثال از میزکار که از آن برای تشریح چندین دیدگاه این فصل استفاده خواهد شد، *Dali* را می‌توان نام برد که توسط *SEI* توسعه داده شده است.

۱۰-۱-۲- فعالیت‌های بازسازی مجدد

بازسازی مجدد معماری نرم‌افزار متشکل از فعالیت‌های زیر است که به صورت تکراری اجرا می‌شوند:

۱. *استخراج اطلاعات*

هدف از این فعالیت آن است که اطلاعات از منابع مختلف استخراج گردد.

۲. *ساخت پایگاه‌داده*

ساخت پایگاه‌داده شامل تبدیل اطلاعات بدست آمده به یک شکل استاندارد مانند شکل استاندارد *Rigi* (یک قالب داده مبتنی بر تاپل) و قالب پایگاه داده مبتنی بر *SQL* برای ایجاد پایگاه‌داده است.

[1] Workbench
[2] Open

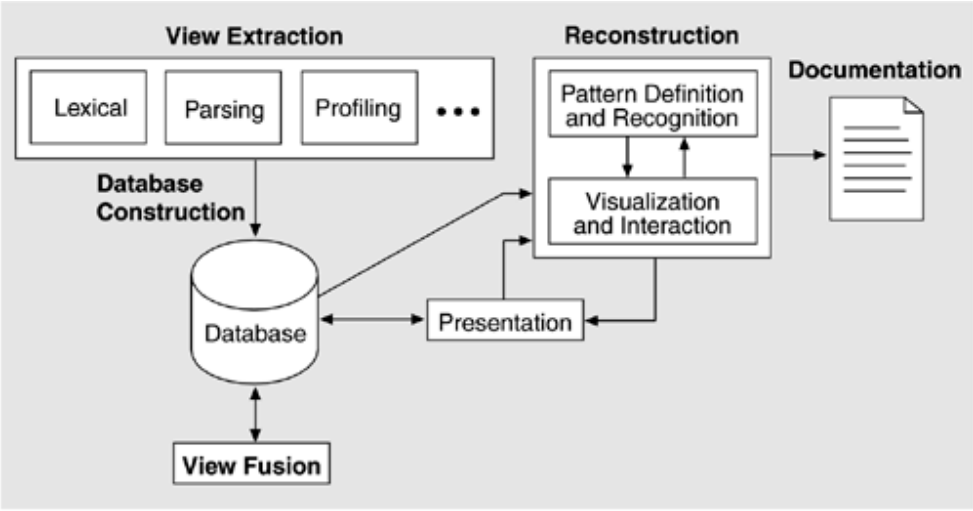
۳. *ترکیب دیدها*[1]

ترکیب دیدها، اطلاعات موجود در پایگاه‌داده را با یکدیگر ترکیب می‌کند تا دید منسجمی از معماری ایجاد کند.

۴. *بازسازی*

فعالیت بازسازی مجدد معماری محلی است که اصلی‌ترین کار مربوط به ساخت تجرید و نمایشهای مختلف از داده در جهت ایجاد یک نمایش معماری قرار دارد.

شکل ۱۰-۱ فعالیت‌های مربوط به بازسازی مجدد معماری و نحوه جریان یافتن داده‌ها بین آنها را نشان می‌دهد.



شکل ۱۰-۱: فعالیت‌های مربوط به بازسازی مجدد معماری

فرآیند بازسازی مجدد معماری نیازمند مشارکت چندین شخص است. تعدادی از این اشخاص عمل بازسازی (بازساز[2]) را انجام می‌دهند و تعدادی نیز کسانی هستند که با معماری سیستم در حال بازسازی آشنا هستند (معمار و مهندس نیازمندیها). بازساز، اطلاعات را از سیستم استخراج می‌کند و به صورت دستی و یا خودکار با استفاده از ابزارها، مدل انتزاعی معماری را بدست می‌آورد. معماری سیستم به وسیله یک سری فرضیات پیرامون سیستم از طریق بازساز ایجاد می‌شود. فرضیات در واقع بازتاب کننده نگاشت‌های معکوس از فرآورده‌های منبع

[1] View fusion
[2] Reconstructor

به معماری هستند. آنها به وسیله ایجاد نگاشت معکوس و اعمال آن به اطلاعات استخراج شده آزمایش می‌شوند و نهایتا نیز یک اعتبارسنجی بر روی آنها انجام می‌شود. به منظور ایجاد فرضیات کارا و انجام اعتبارسنجی مناسب، باید از اشخاصی استفاده شود که با سیستم آشنا هستند. این اشخاص می‌توانند معمار و مهندسینی باشند که بر روی آن سیستم کار کرده‌اند (کسی که برای اولین بار سیستم را توسعه داده و یا کسی که در حال حاضر عمل نگهداری را انجام می‌دهد).

در بخش‌های بعدی فعالیت‌های مختلف بازسازی مجدد معماری به صورت دقیق تشریح خواهند شد و راهنمایی نیز در مورد هر فعالیت ارائه خواهد شد. بسیاری از این راهنماها مخصوص استفاده از یک میزکار خاص نیستند و می‌توانند در بازسازی مجدد به صورت دستی نیز کاربرد داشته باشند.

۱۰-۲- استخراج اطلاعات

استخراج اطلاعات دربرگیرنده تحلیل فرآورده‌های طراحی و پیاده‌سازی سیستم موجود در جهت ایجاد یک مدل از آن است. خروجی، یک سری اطلاعات قرار گرفته در پایگاه‌داده است که در فعالیت ترکیب دیدها برای ایجاد دید منسجم از سیستم استفاده می‌شود. استخراج اطلاعات ترکیبی از اهداف ایده‌ال و عملی است.

منظور از ایده‌ال یعنی چه اطلاعاتی باید درباره معماری استخراج شود تا بتوان بر پایه آن به برآورده‌سازی اهداف بازسازی مجدد کمک کرد و عملی نیز یعنی چه اطلاعاتی را ابزار واقعا می‌تواند استخراج و نمایش دهد. از طریق فرآورده‌های منبع (مانند کد، سرفایلها و فایلهای ایجاد شده) و دیگر فرآورده‌ها (مانند ردیابیهای اجرایی[1]) می‌توان عناصر (مانند فایلها، توابع و متغییرها) و روابط بین آنها شناسایی کرد که این عناصر و روابط منجر به بدست آوردن چندین دید پایه از سیستم می‌شوند. جدول ۱۰-۱ یک فهرست عمومی از عناصر و روابط بین آنها را که می‌توانند استخراج شوند، نشان می‌دهد.

[1] Execution traces

جدول ۱۰-۱: عناصر و روابط استخراج شده عمومی

عنصر مبدا	رابطه	عنصر مقصد	توضیحات
File	*"include"*	*File*	پیش‌پردازنده *C*، دربرداشتن یک فایل توسط فایل دیگر
File	*"contains"*	*Function*	تعریف یک تابع در فایل
File	*"defines_var"*	*Variable*	تعریف یک متغییر درون فایل
Directory	*"contains"*	*Directory*	فهرست شامل زیر فهرست است
Directory	*"contains"*	*File*	فهرست شامل فایل است
Function	*"calls"*	*Function*	فراخوانی ایستای تابع
Function	*"access_read"*	*Variable*	دسترسی خواندن بر روی متغییر
Function	*"access_write"*	*Variable*	دسترسی نوشتن بر روی متغییر

هر ارتباط بین عناصر اطلاعات متفاوتی درباره سیستم می‌دهد. رابطه *"call"* بین توابع به ساخت یک گراف فراخوانی کمک می‌کند. رابطه *"includes"* بین فایلها یک مجموعه از وابستگیها بین فایلها را مشخص می‌کند. رابطه‌های *"access_read"* و *"access_write"* بین توابع و متغیرها نشان می‌دهند که چگونه داده‌ها استفاده می‌شوند. زیرا یک سری از توابع خاص می‌توانند بر روی مجموعه‌ای از داده‌ها عمل نوشتن را انجام دهند و مجموعه‌های دیگر نیز می‌توانند فقط داده‌ها را بخوانند. این اطلاعات به منظور مشخص کردن چگونگی انتقال داده‌ها بین بخشهای مختلف سیستم استفاده می‌شوند. همچنین می‌توان مشخص کرد که آیا از یک مخزن داده عمومی استفاده شده یا نه؟ یا اینکه بیشترین اطلاعات رد و بدل شده بین توابع در هنگام فراخوانی چه اقلامی هستند؟

اگر سیستمی که تحلیل می‌شود بزرگ باشد احتمالا در این صورت چگونگی ذخیره شدن فایلهای منبع در داخل ساختار فهرست‌های سیستم برای فرآیند بازسازی مهم خواهد بود. ممکن است عناصرخاص یا زیرسیستمها در فهرست‌های خاص ذخیره شوند که بدست آوردن رابطه‌های *"dir_contains_file"* و *"dir_contains_dir"* مفید خواهد بود، زیرا بعدا آنها کمک شایانی در جهت شناسایی عناصر خواهند کرد. مجموعه عناصر و ارتباطات استخراج شده به نوع سیستمی که تحلیل می‌شود و همچنین ابزار استخراج کننده قابل دسترس بستگی دارد. به عنوان مثال اگر سیستمی که بازسازی مجدد می‌شود شیءگرا باشد در این صورت

کلاسها و رویه‌ها[1] به عنوان عناصر به فهرست عناصر استخراج شده اضافه می‌شوند و رابطهایی مانند *"class_is_subclass_of_class"* و *"class_contains_method"* نیز استخراج شده و استفاده می‌شوند.

اطلاعات بدست آمده می‌توانند به دو دسته ایستا و پویا دسته‌بندی شوند. اطلاعات ایستا از طریق مشاهده فرآورده‌های سیستم و اطلاعات پویا نیز از طریق مشاهده چگونگی اجرای سیستم بدست می‌آیند. در اصل هدف آن است که هر دو دسته را با یکدیگر ترکیب کرده تا بتوان یک دید دقیق از سیستم ایجاد کرد. هنگامی که معماری سیستم در حال اجرا تغییر کند (به عنوان مثال، یک فایل پیکربندی در بارگذاری اولیه خوانده شده و مبتنی بر آن یک سری عناصر به عنوان نتیجه در حافظه بار می‌شوند) پیکربندی زمان اجرای آن باید تشخیص داده شده و در هنگام بازسازی مجدد سیستم مورد استفاده قرار بگیرد.

به منظور استخراج اطلاعات یک سری از ابزارها به شرح زیر وجود دارند:

- *پارسرها (مانند Imagix, SNiFF+, CIA, rigiparse)*

پارسرها کد را تحلیل کرده و یک نمایش داخلی از سیستم ایجاد می‌کنند (با هدف تولید کد ماشین) و معمولا هم امکان‌پذیر است که نمایش داخلی را ذخیره کرده تا بتوان یک دید را از آن بدست آورد.

- *تحلیلگر درخت نحوی تجرید[2] (مانند Gen++, Refine)*

درخت نحوی تجرید، عملکردی مشابه با پارسر دارد ولی آن درخت صریح از اطلاعات پارس شده را نمایش می‌دهد. بنابراین امکان‌پذیر است که ابزار تحلیلی را فراهم کرد تا بتوان درخت نحوی تجرید را پیمایش کند و قسمت‌های مناسب از نظر معماری را در قالب خاص به عنوان خروجی تولید کند.

- *تحلیلگر لغوی (مانند LSME)*

تحلیلگرهای لغوی منحصرا فرآورده‌های منبع را به صورت رشته‌هایی از عناصر لغوی یا نشانه‌ها[3] بررسی می‌کنند. همچنین کاربر تحلیلگر لغوی می‌تواند مجموعه‌ای از الگوهای کد را برای گرفتن خروجی مورد انتظار یا تطبیق مشخص کند.

[1] Methods
[2] Abstract syntax tree (AST) analysers
[3] token

- *نمایش‌دهندها*[1] *(مانند gprof)*

نمایش‌دهنده‌ها می‌توانند به منظور گرفتن اطلاعاتی درباره کدی که اجرا می‌شوند، به کار روند و معمولا هم هیچ کدی به سیستم وارد نمی‌کنند.

- *ابزارهای مرتب کننده کد*[2]

ابزارهای مرتب کننده، کاربردهای گسترده‌ای در آزمایش و وارد کردن کدهای خاص به سیستم در جهت دریافت اطلاعت خاص در هنگام اجرای سیستم دارند.

- *کاربردهای موردی*[3] *(مانند grep, perl)*

کاربردهای موردی نیز مشابه با تحلیلگر لغوی می‌توانند الگوهای تطابق خاصی را از کاربر دریافت کرده و آن را در داخل کد منبع جستجو کنند تا از طریق آن بتوانند به یک سری اطلاعات دست پیدا کنند.

پارسرهای تولید کننده کد، تحلیلگرهای مبتنی بر درخت تحلیل نحوی، تحلیلگر لغوی و کاربردهای موردی تطابق الگو، همگی به عنوان ابزارهایی هستند که به برای بدست آوردن اطلاعات ایستا به کار گرفته می‌شوند. ابزارهای نمایش‌دهنده و مرتب کننده کد نیز به منظور دستیابی به اطلاعات پویا استفاده می‌شوند.

ابزارها برای تحلیل مدلهای طراحی، فایلهای ساخته شده[4]، فایلهای ایجاد شده[5] و فایلهای اجرایی نیز می‌توانند برای استخراج اطلاعات بیشتر مورد نیاز برای بازسازی معماری مورد استفاده قرار بگیرند. برای نمونه، فایلهای ایجاد و ساخته شده، شامل اطلاعاتی در مورد ماژولها و وابستگیهای فایلها هستند که در سیستم وجود دارند ولی نمی‌شود آنها را از طریق کد منبع بدست آورد. به عبارت بهتر اینگونه اطلاعات در فایلهای منبع وجود ندارند. اطلاعات بیشتر مرتبط با معماری می‌تواند به صورت ایستا از طریق کد منبع، فرآورده‌های زمان کامپایل و طراحی استخراج شوند، با این حال بعضی اطلاعات مرتبط با معماری در فرآورده‌های منبع به دلیل انقیاد دیرهنگام[6] در دسترس نیستند. نمونه‌هایی از انقیاد دیرهنگام به شرح زیر هستند:

[1] Profilers
[2] Code instrumentation tools
[3] Ad hoc
[4] Build files
[5] Makefiles
[6] Late binding

- چند ریختی
- اشاره‌گرهای توابع
- پارامتریک نمودن زمان اجرا[1]

گاها ممکن است توپولوژی دقیقی از سیستم تا اجرای آن نتوان تشخیص داد. برای مثال، سیستمهای چند پردازنده و چند فرآیندی (از میان‌افزارهایی مانند *J2EE, Jini, .NET* استفاده می‌کنند) به صورت متناوب توپولوژی خود را به صورت پویا ایجاد می‌کنند که این بستگی به منابع قابل دسترس سیستم دارد. توپولوژی چنین سیستمهایی در فرآورده‌های منبع آنها وجود ندارد و لذا نمی‌توان با استفاده از ابزارهای استخراج کننده ایستا، مهندسی معکوس را برای آنها انجام داد. به همین دلیل باید از ابزارهایی استفاده شود تا بتوان از طریق آنها به اطلاعات پویای سیستم دست پیدا کرد. البته این نیازمند آن است که ابزارها در سکویی که سیستم مورد نظر اجرا می‌شوند قابلیت اجرا شدن را داشته باشند. همچنین گاهی اوقات نمی‌توان خروجی مرتب کننده کد را جمع‌آوری کرد. به عنوان مثال سیستمهای توکار به هیچ عنوان اجازه خروجی گرفتن از این نوع داده‌ها را نمی‌دهند.

۱۰-۲-۱- راهنمایی‌ها

به منظور استخراج اطلاعات یک سری از ملاحظات کاربردی وجود دارند که اعمال آنها موجب بازسازی مجدد موفق برای یک پروژه می‌شود. این ملاحظات به شرح زیر هستند:

- *استفاده از "کمترین تلاش"[2] برای استخراج*

قبل از اینکه اقدام به استخراج اطلاعات از منبع شود باید در نظر گرفته شود که چه اطلاعاتی مورد نیاز است؟ آیا نیاز است که ساختارهای نحوی[3] پیچیده درک شوند؟ آیا نیاز به تحلیل معنایی است؟ در هر حالت ابزارهای متفاوتی می‌تواند در جهت موفقیت به کار گرفته شود. عموما روشهای لغوی برای استفاده ارزان هستند و زمانی باید از آنها استفاده کرد که اهداف بازسازی ساده هستند.

[1] Runtime parameterization
[2] Least effort
[3] Syntactic

- *اعتبارسنجی اطلاعات استخراج شده*

قبل از اینکه اقدام به دستکاری یا ترکیب دیدهای بدست آمده شود باید مطمئن گشت که دیدهای صحیحی کشف شده‌اند. به عبارت بهتر، یعنی اینکه اطلاعات دیدها صحیح هستند. بسیار مهم است که ابزار استفاده شده برای استخراج اطلاعات، کار خود را به درستی انجام دهند. بدین منظور نخست باید بررسی جزئیات انجام شد و سپس اعتبارسنجی زیر مجموعه‌های عناصر و ارتباطات بین آنها نسبت به کد منبع انجام شود تا اطمینان حاصل شود که اطلاعات صحیح بدست آمده است. دقت اطلاعاتی که مورد نیاز است تا بر روی آن اعتبارسنجی انجام شود با توجه به نوع سیستم و نظر بازساز متفاوت است. فرض کنید که یک نمونه‌گیری آماری وجود دارد بنابراین بر روی درجه اطمینان تصمیم‌گیری می‌شود و سپس راهبرد نمونه‌گیری مناسب برای دستیابی با آن انتخاب می‌شود.

- *استخراج اطلاعات پویا در صورت نیاز*

ممکن است مکانهایی در سیستم وجود داشته باشد که انقیادهای زمان اجرا زیادی در آن وجود داشته باشند یا اینکه معماری به صورت پویا قابل پیکربندی باشد بنابراین باید محل مناسب را مشخص کرده و اطلاعات مورد نیاز را استخراج کرد.

۱۰-۳- ساخت پایگاه‌داده

در طول فعالیت ساخت پایگاه‌داده اطلاعات استخراج شده به یک قالب استاندارد جهت ذخیره‌سازی در پایگاه‌داده تبدیل می‌شوند. بنابراین ضروری است که یک مدل پایگاه‌داده انتخاب شود. هنگام انتخاب مدل پایگاه‌داده باید ملاحظات زیر را مد نظر قرار داد:

- مدل باید شناخته شده باشد تا عمل جایگزینی پیاده‌سازی یک پایگاه‌داده با نوع دیگر به راحتی انجام شود.
- مدل باید اجازه پرس‌وجوهای کارا را بدهد. این موضوع بسیار مهم است از آنجایی که مدلهای منبع می‌توانند کاملا بزرگ باشند.
- مدل باید از دسترسی راه دور به پایگاه‌داده آن هم از طریق یک یا چند واسط کاربری توزیع شده از نظر جغرافیایی حمایت کند.

- مدل باید از ترکیب دیدها به وسیله ترکیب کردن اطلاعات از جداول مختلف حمایت کند.
- مدل باید از زبانهای پرس‌وجو که می‌توانند الگوهای معماری را بیان کنند، حمایت کند.
- نقطه وارسی باید توسط پیاده‌سازی حمایت شود یعنی اینکه نتایج میانی بتوانند ذخیره شوند. این موضوع در فرآیندهای تعاملی بسیار مهم است زیرا به کاربران آزادی عمل می‌دهد که به راحتی تغییرات را اعمال کنند و بدانند که برگشتن به حالتی قبلی به سادگی امکان‌پذیر است.

برای مثال، میزکار *Dali* از پایگاه‌داده رابطه‌ای استفاده می‌کند. برای این کار دیدهای استخراج شده را به شکل استاندارد *Rigi* تبدیل می‌کند. سپس این قالب به وسیله نوشته *Perl* خوانده شده و خروجی تولید می‌شود که دارای کدهای *SQL* مورد نیاز برای تولید جداول رابطه‌ای است. شکل ۱۰-۲ طرح کلی این فرآیند را نشان می‌دهد و شکل ۱۰-۳ نیز مثالی از کد *SQL* تولید شده از طریق این فرآیند را برای ایجاد جداول رابطه‌ای نمایش می‌دهد.

شکل ۱۰-۲: تبدیل اطلاعات استخراج شده به قالب *SQL*

```
create table calls( caller text, callee text );
create table access( func text, variable text );
create table defines_var( file text, variable text );
...
insert into calls values( 'main', 'control' );
insert into calls values( 'main', 'clock' );
...
insert into accesses values( 'main', 'stat 1' );
```

شکل ۱۰-۳: مثالی از کد *SQL* تولید شده در *Dali*

زمانی که داده به درون پایگاه‌داده وارد می‌شود دو جدول اضافی نیز به نامهای عناصر و رابطه ایجاد می‌شود. این لیست‌ها شامل عناصر و رابطه‌های استخراج شده هستند. روش میزکار، تطبیق با ابزارها و تکنیک‌های جدید را امکان‌پذیر می‌کند تا از این طریق بتواند از انواع تبدیلات برای استفاده در ابزارهای مختلف پشتیبانی کند. برای مثال اگر ابزاری نیاز است تا زبان جدیدی مدیریت شود آن مدیریت (تحلیل) می‌تواند انجام شود و در نهایت خروجی می‌تواند به قالب میزکار تبدیل شود.

۱۰-۳-۱- راهنمایی‌ها

هنگام ساخت پایگاه‌داده ملاحظات زیر باید مد نظر قرار داده شوند:

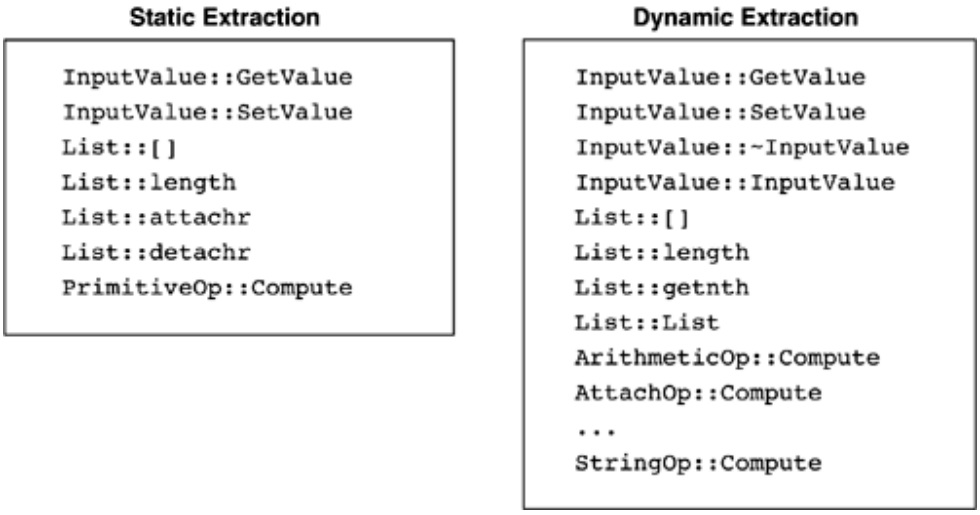
- جداول پایگاه‌داده باید از رابطه‌های استخراج شده ایجاد شود تا اینکه پردازش دیدهای داده در روند فرآیند ترکیب دیدها به آسانی انجام شود. برای مثال، جدولی برای ذخیره نتایج پرس‌وجوی خاص استفاده شود تا اینکه دیگر نیاز به اجرای مجدد پرس‌وجو وجود نداشته باشد. یعنی اگر نتایج مورد نیاز باشد می‌تواند به آسانی آن را از طریق جدول مورد نظر به دست آورد.
- قبل از ایجاد پایگاه‌داده، طراحی پایگاه‌داده باید به دقت مورد بررسی قرار بگیرد. در واقع باید بتوان به این سئوالات پاسخ داد. کلیدهای اصلی چه اقلامی می‌توانند باشند؟ آیا پیوندهای خاصی وجود دارند که پرهزینه بوده و چندین جدول را تحت تاثیر قرار می‌دهند؟ در بازسازی مجدد، جداول بسیار ساده هستند و کلید اصلی تابعی از کل سطر است.
- از ابزارهای ساده تحلیل لغوی شبیه *perl* و *awk* برای تغییر قالب داده‌ها به قالبی که بتواند توسط میزکار به کار گرفته شود، استفاده کنید.

۱۰-۴- ترکیب دیدها

ترکیب دیدها شامل تعریف و دستکاری اطلاعات استخراج شده (در این مرحله اطلاعات در پایگاه‌داده ذخیره هستند) در جهت تطبیق، تکمیل و برقراری ارتباطات بین عناصر است. شکلهای مختلف استخراج باعث فراهم نمودن اطلاعات کامل می‌شوند. ترکیب دیدها در این بخش با استفاده از یک مثال تشریح خواهد شد تا درک و فهم آن آسانتر گردد.

۱۰-۴-۱- اصلاح یا بهبود دید[1]

دو بخش نشان داده شده در شکل ۱۰-۴ را در نظر بگیرید که مجموعه‌ای از رویه‌های استخراج شده از یک سیستم پیاده‌سازی شده با *C++* است. این جداول دربرگیرنده اطلاعات پویا و ایستا مربوط به قطعه کد شیءگرا است. به عنوان مثال، از اطلاعات پویا قابل مشاهده است که *List::getnth* فراخوانی می‌شود. با این حال، این متد در بخش اطلاعات ایستا قرار نگرفته است زیرا ابزار استخراج کننده اطلاعات ایستا نتوانسته است آن را تشخیص دهد. همچنین فراخوانی سازنده و مخرب رویه‌های *InputValue* و *List* در اطلاعات ایستا قرار ندارد و نیاز است که به جدول کلاس/رویه در جهت تطبیق اضافه شوند.



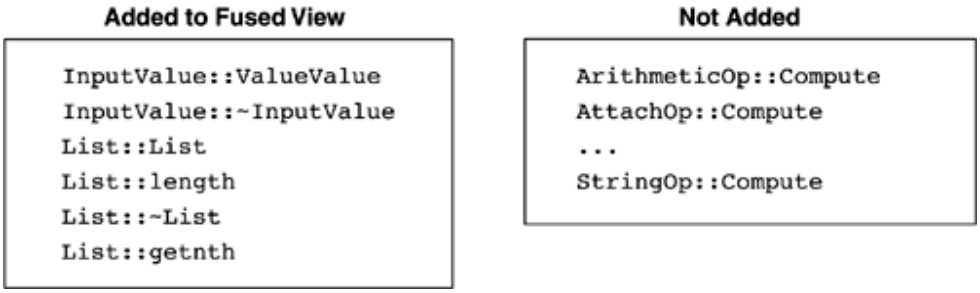
شکل ۱۰-۴: اطلاعات در مورد داده‌های ایستا و پویا موجو در رابطه *class_contains_method*

علاوه بر اینها، استخراج ایستا در این مثال نشان می‌دهد که *PrimitiveOp* یک رویه به نام *Compute* دارد. نتایج حاصل از استخراج پویا چنین کلاسی را نشان نمی‌دهد اما آن کلاسهایی مانند *ArithmeticOp, AttachOp* و *StringOp* را نشان می‌دهد. هر کدام از این کلاسها رویه *Compute* را دارند. در حقیقت آن یک زیرکلاس از *PrimitiveOp* است . کلاس *PrimitiveOp* کاملا یک زیرکلاس است و هرگز در روند اجرای برنامه به صورت واقعی فراخوانی نمی‌شود.

برای دستیابی به دید دقیق از معماری نیاز است که اطلاعات پویا و ایستا *PrimitiveOp* با هم تطبیق پیدا کنند. برای انجام آن می‌توان از یک ترکیب با استفاده از پرس‌وجوهای *SQL* بر روی رابطه‌های استخراج شده *calls,*

[1] Improving view

actually_calls و *has_subclass* استفاده کرد. در این روش کاملا مشخص است که فراخوانی *PrimitiveOp::Compute* (از اطلاعات ایستا بدست آمده) و انواع زیرکلاسهای آن (از اطلاعات پویا بدست آمده) واقعا یکسان هستند. فهرست نشان داده شده در شکل ۱۰-۵ موارد اضافه و حذف شده به دید ترکیب شده را نشان می‌دهد.



شکل ۱۰-۵: موارد اضافه و حذف شده از دید کلی

۱۰-۴-۲- رفع ابهام فراخوانی توابع[1]

در کابردهای چند فرآیندی احتمال رخ دادن تداخل نام وجود دارد. برای مثال، چندین فرآیند ممکن است یک رویه به نام *Call* داشته باشند. بنابراین کاملا مهم و ضروری است که تداخل‌ها در دیدهای استخراج شده، کشف و رفع ابهام شوند. با استفاده از ترکیب اطلاعاتی که می‌توانند به راحتی استخراج شوند می‌توان این نوع تداخلات بالقوه را از بین برد. در این حالت نیاز است که جدول ایستای *Call* با جدول *"file/function containment"* (برای تشخیص اینکه کدام توابع در فایلهای منبع تعریف شده‌اند) و جدول *"build dependency"* (برای تشخیص اینکه کدام فایلها کامپایل می‌شود تا فایلهای اجرای تولید شوند) ترکیب شود. ترکیب این اطلاعات اجازه می‌دهد که رویه‌ها و توابعی که دارای تداخل هستند نامشان منحصر به فرد شود و لذا بدون ابهام به آنها در فرآیند بازسازی مجدد معماری ارجاع شود. بدون ترکیب دیدها این ابهامات و تداخلات در معماری بازسازی شده باقی خواهند ماند.

۱۰-۴-۳- راهنمایی‌ها

در ادامه یک سری ملاحظات کاربردی پیرامون نحوه اعمال کردن این فعالیت وجود دارد که عبارتند از:

[1] Disambiguated function calls

- ترکیب جداول باید زمانی انجام شود که هیچ جدول منفردی اطلاعات مورد نیاز را فراهم نمی‌کند.
- ترکیب جداول باید زمانی انجام شود که ابهامی در یکی از جداول وجود دارد و یا اینکه غیر ممکن است تا تداخل را با استفاده از یک جدول منفرد حل کرد.
- برای استخراج اطلاعات مختلف باید از تکنیک‌های متفاوت استخراج استفاده شود. برای مثال، می‌توان از استخراج پویا و ایستا استفاده کرد یا اینکه ممکن است از نمونه‌های مختلف یک تکنیک استفاده شود و این عمل زمانی انجام می‌شود که یک نمونه ممکن است اطلاعات نادرست و ناقصی را فراهم کند. مانند پارسرهای مختلف برای یک زبان.

۱۰-۵- بازسازی

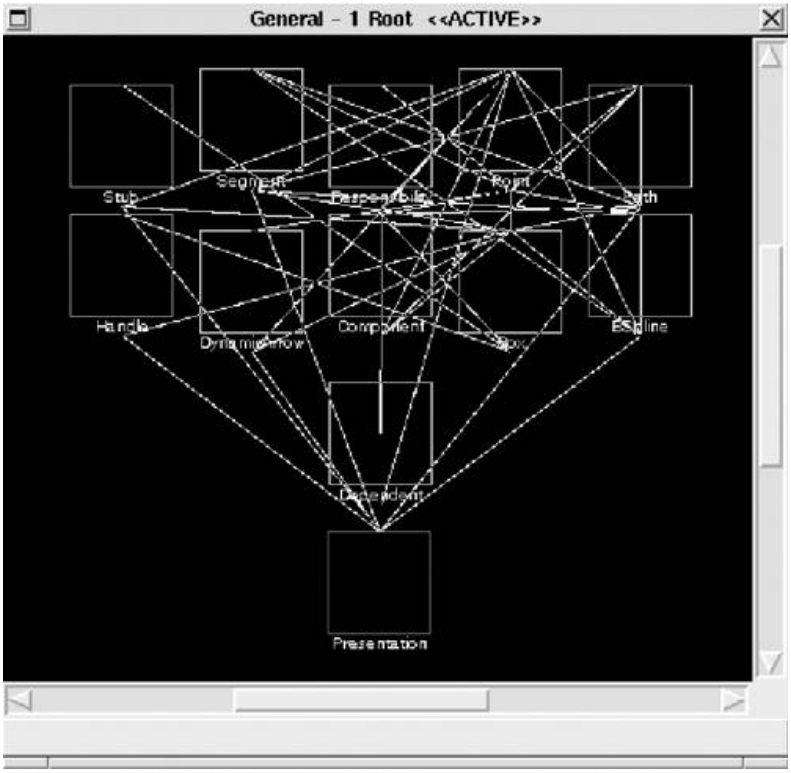
تا این بخش از روش بازسازی مجدد معماری، اطلاعات دیدها استخراج، ذخیره، اصلاح و تکامل‌یافته تا اینکه کیفیت آن بهبود پیدا کند. بازسازی مجدد بر روی دیدها عمل می‌کند تا درون‌بینی گسترده و دانه درشتی از معماری را آشکار کند. بازسازی متشکل از دو فعالیت اصلی است. اولی مصورسازی[۱] و تعامل[۲] است دومی نیز تشخیص و تعریف الگو[۳] است که هر کدام در ادامه تشریح خواهد شد.

مصورسازی و تعامل مکانیزمی را فراهم می‌کند که به وسیله آن کاربر می‌تواند به صورت تعاملی دیدها را دستکاری، بررسی و به تصویر بکشاند. در *Dali* دیدها به وسیله ابزار *Rigi* به صورت گراف تجزیه سلسله مراتبی از عناصر و ارتباطات نشان داده می‌شوند. یک مثال از دید معماری در شکل ۱۰-۶ به تصویر کشیده شده است.

تشخیص و تعریف الگو امکاناتی را برای بازسازی معماری فراهم می‌کند. یکی از آنها تشخیص و تعریف کد است. تشخیص و تعریف کد باعث آشکار شدن الگوهای معماری می‌شود. به عنوان مثال، امکانات بازسازی *Dali* به کاربر اجازه ساخت دیدهای انتزاعی سیستم نرم‌افزاری از دیدهایی با جزئیات بیشتر به وسیله شناسایی عناصر تجمیع را می‌دهد. الگو در *Dali* با استفاده از ترکیب *SQL* و *Perl* تعریف می‌شوند و قطعات کد[۴] نامیده

[1] Visualization
[2] Interaction
[3] Pattern definition and recognition
[4] Code segments

می‌شود. پرس‌وجوی *SQL* برای شناسائی عناصر از مخزن *Dali* که در یک تجمیع جدید شرکت می‌کنند، استفاده می‌شود و عبارات *Perl* نیز برای تبدیل نام‌ها و دستگاری نتایج پرس‌وجو مورد استفاده قرار می‌گیرد.



شکل ۱۰-۶: دید معماری تولید شده توسط *Dali*

مبتنی بر الگوهای معماری که معمار انتظار دارد در سیستم آنها را پیدا کند بازساز می‌تواند انواع پرس‌وجوها را ایجاد کند. نتایج این پرس‌وجوها منجر به تجمیعهای جدید می‌شود که انواع تجریدها یا خوشه‌بندیهای[1] عناصر سطح پایین (می‌تواند فرآورده منبع یا تجریدها باشد) را نشان می‌دهد. به وسیله تفسیر و تحلیل فعال این دیدها می‌توان پرس‌وجوها و تجمیعها را بهبود بخشید تا چندین دید معماری پیش فرض تولید کرد که این دیدها می‌توانند تفسیر، اصلاح یا رد شوند.

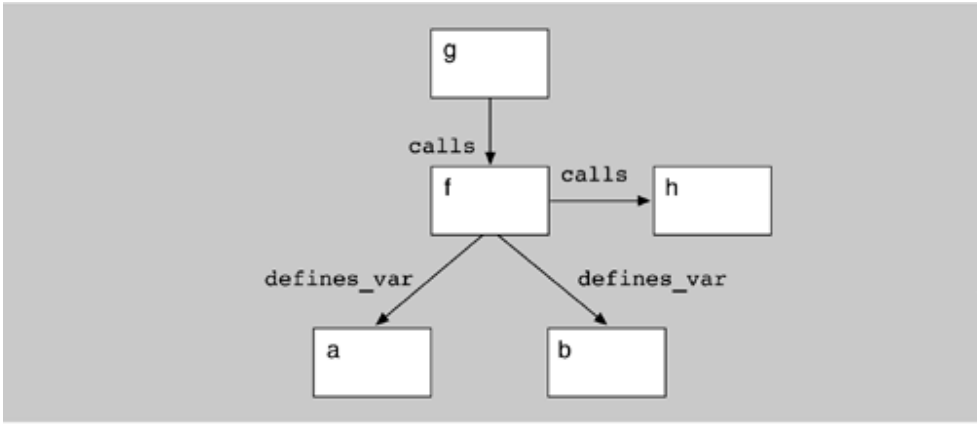
در ضمن هیچ معیار کامل جهانی برای این فرآیند وجود ندارد. این فرآیند زمانی کامل می‌شود که نمایش‌های معماری برای تحلیل و مستندسازی کافی باشند.

[1] Clusterings

فرض کنید یک پایگاه‌داده فرضی متشکل از زیرمجموعه‌ای از عناصر و ارتباطات نشان داده شده در شکل ۱۰–۷ است. در این مثال متغییرهای a و b در تابع f تعریف شده‌اند. یعنی اینکه آنها متغییرهای محلی برای f هستند. بنابراین می‌تواند این اطلاعات را به صورت شکل ۱۰–۸ نشان داد.

Element	Relationship	Element
f	defines_var	a
f	defines_var	b
g	calls	f
f	calls	h

شکل ۱۰–۷: زیر مجموعه‌ای از عناصر و ارتباطات



شکل ۱۰۸: نمایش گرافیکی از عناصر و ارتباطات

بازسازی معماری توجه زیادی به متغییرهای محلی ندارد زیرا آنها نقش خیلی کمی در درون‌بینی معماری سیستم دارند. بنابراین می‌توان نمونه‌های متغییرهای محلی را در داخل توابع که آنها رخ می‌دهد، تجمیع کرد. یک مثال از کد *SQL* و *Perl* که این عمل را انجام می‌دهند در شکل ۱۰–۹ نشان داده شده است.

```
#Local Variable aggregation

SELECT tName
        FROM Elements
        WHERE tType='Function';
print "$fields[0]+ $fields[0] Function\n";

SELECT d1.func, d1.local_variable
        FROM defines_var d1;
print "$fields[0] $fields[1] Function\n";


<function>+ <function> Function
```

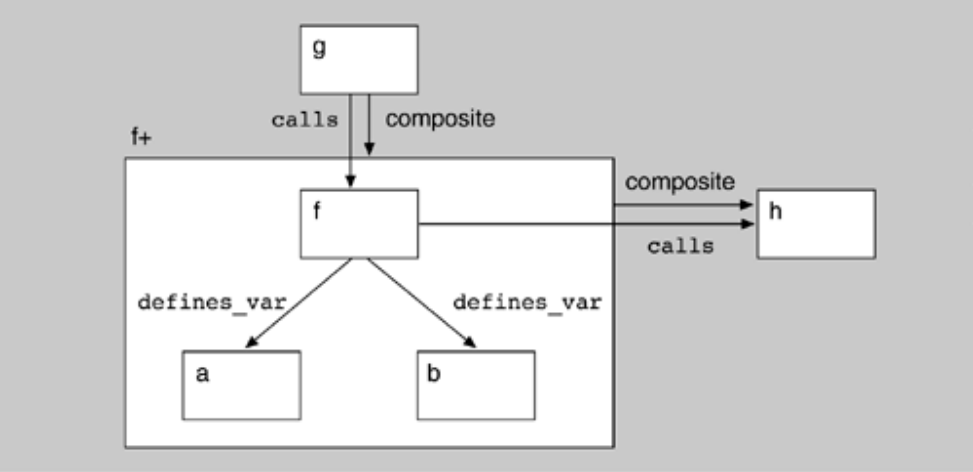
شکل ۱۰-۹: تجمیع متغییرهای محلی در توابع به کار رفته با استفاده از *SQL* و *Perl*

بخش نخست از کد نشان داده شده در شکل ۱۰-۹، ارائه بصری را به وسیله اضافه کردن "+" بعد از نام هر تابع به روز می‌کند. حال تابع با متغییرهای محلی درون خودش تجمیع شده است. پرس‌وجوی *SQL* توابع را از جدول عناصر انتخاب می‌کند و عبارت *Perl* نیز به ازای هر خط خروجی پرس‌وجو، اجرا می‌شود. آرایه *$fields* به صورت خودکار توسط نتیجه پرس‌وجو مقدار می‌گیرد. در این حالت، فقط یک فیلد از جدول به نام *tName* انتخاب شده است بنابراین *$fields[0]* مقدار آن فیلد را به ازای هر تاپل انتخاب شده ذخیره می‌کند. عبارت نوشته شده توسط *perl* نیز قالب خطوط را تولید می‌کند و مشخص می‌کند که عنصر <*function*> باید با +<*function*> تجمیع شود.

بخش دوم از کد، متغییرهای محلی را از مصورسازی مخفی می‌کند. پرس‌وجوی *SQL* متغییرهای محلی برای هر تابع تعریف شده را به وسیله انتخاب هر تاپل در جدول *defines_var* شناسایی می‌کند. بنابراین در عبارت *perl*، *$fields[0]* مربوط به فیلد *func* و *$fields[1]* نیز مربوط به فیلد *local_variable* است. لذا خروجی به صورت زیر خواهد بود

<*function*>+ <*variable*> *Function*

خروجی نشان دهنده این است که هر متغییر محلی تابع در تجمیع تابع +<*function*> اضافه می‌شود. ترتیب اجرای این دو قطعه کد مهم نیست زیرا نتایج نهایی از اعمال هر دوی اینها مرتب می‌شوند. نتیجه اعمال قطعات کد به صورت گرافیکی در شکل ۱۰-۱۰ نمایش داده شده است.



شکل ۱۰-۱۰: نتیجه اعمال قطعه کد

مکانیزم اصلی برای دستکاری اطلاعات استخراج شده نگاشت معکوس است. مثالهایی از آن عبارتند از:

- شناسایی انواع
- تجمیع متغییرهای محلی در توابع
- تجمیع اعضا در کلاسها
- ترکیب عناصر سطح معماری

یک مثال از پرس‌وجویی که عناصر معماری را شناسایی می‌کند در شکل ۱۰-۱۱ نشان داده شده است. این پرس‌وجو، عنصر *Logical_Interaction* را شناسایی می‌کند و بیان می‌کند که اگر نام کلاس *Presentation, BSpline* است یا اگر کلاس یک زیرکلاس از *Presentation* باشد آن متعلق به عنصر *Logical_Interaction* است. قطعات کد به طریقی نوشته می‌شوند تا با تجرید از اطلاعات سطح پائین بتوانند دیدهای سطح معماری را تولید کنند. بازساز این قطعات را برای آزمایش فرضیات سیستم ایجاد می‌کند. اگر یک قطعه کد خروجی مطلوب و مفیدی نداشته باشد می‌توان آن را نادیده گرفت. بازساز این روند را به صورت تکراری آنقدر اجرا می‌کند تا اینکه دیدهای معماری مناسب حاصل شوند.

```
SELECT tSubclass
       FROM has_subclass
      WHERE tSuperclass='Presentation';
print "Logical_Interaction $fields[0]";

SELECT tName
       FROM element
      WHERE tName='Presentation'
     OR tName='BSpline'
     OR tName='Color';
print "Logical_Interaction $fields[0]";
```

شکل ۱۰-۱۱: پرس‌وجو برای شناسایی عنصر *Logical_Interaction*

۱۰-۵-۱- راهنمایی‌ها

یک سری ملاحظات کاربردی در مورد چگونگی به کارگیری فعالیت بازسازی وجود دارند که عبارتند از:

- آماده همکاری نزدیک با معمار باشید و در نظر داشته باشید که چندین بار عملیات مربوطه باید بر روی تجریدهای معماری ایجاد شده تکرار شود. این موضوع مخصوصا زمانی که سیستم خیلی صریح و دارای مستند معماری نیست، برجسته می‌شود. در یک چنین حالتی می‌توان تجریدهای معماری را به عنوان فرضیات سیستم ایجاد کرده و بعد آن فرضیات را به وسیله ایجاد دیدها و نمایش آنها به ذینفعان و معمار آزمایش کرد. با توجه به بازخوردهای مثبت و منفی، بازساز می‌تواند تصمیمات مناسب را اتخاذ کند.
- در زمان توسعه قطعات کد باید سعی شود که آنها مختصر بوده و هر عنصر منبع را در فهرست خود نداشته باشند. در شکلهای ۱۰-۱۱ و ۱۰-۱۲ به ترتیب قطعه کدهای مناسب و نامناسب نمایش داده شده‌اند. در شکل ۱۰-۱۲ عناصر منبع شامل عناصر معماری هستند که به راحتی لیست شده‌اند این عمل باعث می‌شود که استفاده مجدد، درک و به کارگیری آنها دشوار شود.

```
SELECT tName
       FROM element
       WHERE tName='vanish-xforms.cc'
       OR tName='PrimativeOp'
       OR tName='Mapping'
       OR tName='MappingEditor'

       OR tName='InputValue'
       OR tName='Point'
       OR tName='VEC'
       OR tName='MAT'
      OR ((tName ~ 'Dbg$' OR tName ~ 'Event$')
          AND tType='Class');
print "Dialogue $fields[0]";
```

شکل ۱۰-۱۲: مثالی از یک قطعه کد نامناسب

- قطعات کد می‌توانند مبتنی بر یک قرارداد نامگذاری باشند به شرط آنکه این قرارداد در سرتاسر سیستم رعایت شود. به عنوان مثال کلیه توابع، داده‌ها و فایلهای متعلق به واسط باید با *i_* شروع شوند.
- قطعات کد می‌توانند مبتنی بر ساختار فهرست باشند که در آن توابع و فایل قرار دارند. همچنین تجمیع عناصر نیز می‌تواند مبتنی بر همین فهرست‌ها باشد.
- بازسازی معماری تلاشی برای تشخیص مجدد تصمیمات معماری و نمایش نتایج این تصمیمات در فرآورده‌های واقعی است. به محض اینکه بازسازی شروع شد باید اطلاعات مربوط به تصمیمات معماری دوباره تعریف شده، به آنها اضافه شوند که این موضوع باعث می‌شود که نیازی به دانش فرد خاص دیگری وجود نداشته باشد.

فهرست مطالب

فصل یازدهم

روش جامع برای ارزیابی معماری

در این فصل، *ATAM* به عنوان یک روش جامع برای ارزیابی معماری مورد بررسی قرار خواهد گرفت و ذکر خواهد شد که هدف از هر مرحله از این روش چیست. نهایتا نیز برای درک بهتر *ATAM*، یک مطالعه موردی مطرح خواهد شد.

۱۱-۱- مقدمه

در این فصل *ATAM*[1] به عنوان یک روش جامع و کامل برای ارزیابی معماری نرم‌افزار معرفی خواهد شد. دلیل نامگذاری این روش بر این اساس است که آن آشکار می‌کند به چه میزان معماری اهداف کیفی را برآورده می‌کند و همچنین از آنجا که تصمیمات معماری چندین خصوصیت کیفی را تحت تاثیر قرار می‌دهند، این روش درون‌نگری[2] لازم را در مورد چگونگی تعامل اهداف کیفی فراهم می‌کند (یعنی اینکه چگونه با یکدیگر مصالحه[3] می‌کنند).

ارزیابی معماری برای سیستمهای بزرگ فرآیند پیچیده‌ای به حساب می‌آید. دلیلش هم این است که اولا، سیستمهای بزرگ معماری بزرگی دارند که درک آن در یک زمان محدود عمل بسیار دشواری است. دوما، با توجه به اینکه چرخه حرفه کاری سیستمهای کامپیوتری تمایل به حمایت از اهداف تجاری دارد بنابراین ارزیابی نیازمند برقراری ارتباط بین اهداف و تصمیمات معماری است. نهایتا اینکه، سیستمهای بزرگ ذینفعان زیادی دارند و بدست آوردن کلیه دیدگاه‌های آنها در یک زمان محدود نیازمند مدیریت دقیق فرآیند ارزیابی است. بنابراین مدیریت زمان برای ارزیابی معماری یک مشکل اصلی و مرکزی به شمار می‌رود.

روش *ATAM* به منظور استخراج[4] اهداف حرفه برای سیستم و معماری طراحی شده است. این روش از اهداف و مشارکت ذینفعان استفاده می‌کند تا توجه ارزیابان را به بخشهایی متمرکز کند که برای دستیابی به اهداف کیفی بسیار مهم هستند.

۱۱-۲- مشارکت کنندگان در *ATAM*

ATAM به مشارکت و همکاری سه گروه نیازمند است که عبارتند از:

۱. *تیم ارزیابی*

این گروه جزو اعضای معرفی شده پروژه معماری نیستند و معمولا متشکل از سه یا پنج نفر است. هر عضو از تیم دارای تعدادی از نقشهایی است که در روند ارزیابی باید مسئولیت‌هایی مرتبط با آن نقشها

[1] Architecture Tradeoff Analysis Method
[2] Insight
[3] Trade off
[4] Elicit

را انجام دهد. جدول ۱۱-۱ فهرست نقشهای موجود در تیم ارزیابی و خصوصیات مهم هر کدام را نشان می‌دهد. تیم ارزیابی یک واحد همیشه فعال است که ارزیابی معماری به صورت مداوم در آن در حال انجام شدن است. اعضا این تیم می‌تواند از معماران سیستم که در زمینه مورد نظر مهارت دارند، انتخاب شوند. نیروهای تیم می‌توانند متعلق به تیم توسعه‌ی سازمانی باشند که معماری آن در حال ارزیابی است و یا اینکه به عنوان مشاور از خارج سازمان حضور داشته باشند.

۲. *تصمیم‌گیرندگان پروژه*[1]

این افراد در مذاکره به منظور توسعه و تکمیل پروژه توانایی زیادی دارند و یا قدرت و اجازه اعمال تغییرات در معماری را دارند. آنها معمولا شامل مدیر پروژه و نمایندگانی از مشتری هستند که از توسعه سیستم به صورت مالی پشتیبانی می‌کنند. برای معمار معمولا نقش خاصی در ارزیابی معماری وجود دارد که باید آن را به صورت مناسب انجام دهد. نهایتا، شخصی که ماموریت ارزیابی را دارد معمولا توانایی و قدرت مذاکره برای توسعه و تکمیل پروژه دارد (حتی اگر آن قدرت را نداشته باشد) و باید جزو این گروه قرار بگیرد.

۳. *ذینفعان معماری*

ذینفعان در مورد این که معماری را به عنوان مبنایی برای تبلیغ کار خود قرار دهند، سود زیادی را بدست می‌آوردند. آنها اشخاصی هستند که توانایی انجام کارهایشان وابسته به ترویج قابلیت تغییرپذیری، امنیت، قابلیت دسترسی بالا و سایر خصوصیات معماری است. ذینفعان دربرگیرنده توسعه‌دهندگان، آزمایش کنندگان، یکپارچه‌سازها، نگهدارنده‌ها، مهندسین کارایی[2]، کاربران، سازندگان سیستم و سایر افراد درگیر در توسعه سیستم هستند.

نقش ذینفعان در روند ارزیابی آن است که اهداف خصوصیات کیفی خاص که لازم است در معماری برای موفقیت سیستم وجود داشته باشد، را بیان کنند. به منظور ارزیابی مناسب معماری سیستم، باید بین دوازده تا پانزده سرویس مورد انتظار ذینفعان مشخص شود.

[1] Project decision makers
[2] Performance engineers

جدول ۱۱-۱: نقشهای تیم ارزیابی *ATAM*

نقش	مسئولیت‌ها	مشخصات مطلوب
رهبر تیم	تنظیم ارزیابی، هماهنگی با مشتریان، تضمین اینکه نیاز مشتریان برآورده شده است، بستن قرارداد ارزیابی، تشکیل تیم پروژه و رویت گزارش نهایی قابل تحویل	سازمان‌دهنده خوب با مهارتهای مدیریتی، در ارتباط برقرار کردن با مشتری ماهر باشد و به فرجه‌های زمانی نیز توجه داشته باشد
رهبر ارزیابی	اجرای ارزیابی، تسهیل استنباط (استخراج) سناریوها، مدیریت انتخاب سناریوها/فرآیند اولویت‌بندی، تسهیل ارزیابی سناریوها در برابر معماری و تسهیل تحلیل در محل[1]	راحت در مقابل شنوندگان، مهارت عالی در تسهیل کردن، درک خوب از پیامدهای معماری، دارای تجربه در زمینه ارزیابی معماری، قادر به مشخص کردن این موضوع باشد که آیا نیاز است یک بحثی به صورت طولانی انجام شود یا اینکه به خاطر بیفایده بودن باید قطع شود.
نویسنده سناریو[2]	نوشتن سناریوها روی *flipchart* یا *whiteboard* در زمان استخراج سناریوها، دستیابی به توافق بر روی لغات استفاده شده در هر سناریو و متوقف کردن مذاکره تا زمان استخراج شدن لغات	دست خط خوب، سخت گیری در مورد حرکت از طریق شناخت سناریوها، توانایی در درک ماهیت بحث‌های تکنیکی
نویسنده شرح مذاکرات[3]	نوشتن شرح مذاکرات (اقدامات) به شکل الکترونیکی، سناریوهای خام[4]، علت و جزئیات هر سناریو و چاپ فهرستی از سناریوهای پذیرفته شده برای ارائه به مشارکت کنندگان	تایپ کننده خوب و سریع، سازمان‌دهنده خوب برای بازخوانی سریع اطلاعات، قابلیت درک خوب پیامدهای معماری، قابلیت تطبیق سریع پیامدهای تکنیکی، نترس در زمینه ایجاد وقفه در بحث به منظور درک بهتر پیامد و تولید مستندات مفید
وقت نگهدار[5]	به رهبر ارزیابی کمک می‌کند تا مبتنی بر زمانبندی حرکت کند و زمانی را که به هر سناریو در فاز ارزیابی تخصیص داده می‌شود را کنترل می‌کند	حاضر باشد در میان جلسه وقفه ایجاد کند تا زمان را اعلام کند.
ناظر فرآیند[6]	نکاتی را در مورد اینکه فرآیند ارزیابی چگونه می‌تواند بهبود و یا منحرف شود مطرح می‌کند، دادن پیشنهادهایی به رهبر ارزیابی در بخشهای مختلف فرآیند ارزیابی، بعد از اتمام ارزیابی مسئول ارائه گزارش به تیم ارزیابی معمار در زمینه روند فرآیند ارزیابی، نکات مثبت و تجارب موفق به دست آمده است.	ناظری باملاحظه، دارای دانش در زمینه فرآیند ارزیابی، داشتن تجارب در زمینه روش ارزیابی معماری
اجرا کننده فرآیند[7]	به رهبر ارزیابی کمک می‌کند تا مراحل روش ارزیابی را به خاطر آورده و انجام دهد	تسلط کامل بر مراحل روش، بتواند و بخواهد راهنمایی‌های خردمندانه به رهبر ارزیابی بدهد.
پرسشگر	مسائل معمارانه که ذینفعان به آن توجه نداشته‌اند را مطرح می‌کند	بینش معمارانه خوب، بینش خوب در مورد

[1] Onsite analysis
[2] Scenario scribe
[3] Proceedings scribe
[4] Raw scenarios
[5] Timekeeper
[6] Process observer
[7] Process enforcer

		نیازهای ذینفعان، داشتن تجربه در مورد حوزه سیستم مورد ارزیابی، نترس در بیان کردن پیامدهای جنجال برانگیز و دنبال کردن آنها، آشنا با خصوصیات دغدغه‌ها

۱۱-۳- خروجی‌های *ATAM*

ارزیابی مبتنی بر *ATAM* دست کم خروجی‌های زیر را تولید خواهد کرد:

- *نمایش مختصر از معماری*

مستند معماری اغلب به عنوان مجموعه‌ای متشکل از مدل شیء، فهرستی از واسط‌ها و امضاهای آنها و سایر عناصر در نظر گرفته می‌شود. اما یکی از نیازمندیهای *ATAM* آن است که معماری در یک ساعت قابل ارائه باشد که منجر به این می‌شود که نمایش معماری مختصر و قابل فهم باشد.

- *بیان اهداف حرفه*

اهداف حرفه نمایش داده شده در *ATAM* دائما توسط برخی تیمهای توسعه برای اولین بار مشاهده می‌شوند.

- *نیازهای کیفی بر حسب مجموعه‌ای از سناریوها*

اهداف حرفه منجر به نیازهای کیفی می‌شوند. برخی نیازهای کیفی مهم در قالب سناریوها بدست می‌آیند.

- *نگاشت تصمیمات معماری به نیازهای کیفی*

تصمیمات معماری می‌توانند بر حسب خصوصیاتی کیفی که آنها پشتیبانی یا جلوگیری می‌کند، تفسیر شوند. برای هر سناریوی کیفی مورد بررسی قرار گرفته در روند اعمال *ATAM*، تصمیمات معماری که منجر به دستیای به آن می‌شود، مشخص می‌شوند.

- *مجموعه‌ای از نقاط مصالحه و حساس شناسایی شده*

تصمیمات معماری وجود دارند که تاثیرات مشخصی بر روی خصوصیات کیفی دارند. برای مثال، کاملا واضح است که در نظر گرفتن پایگاه داده پشتیبان یک تصمیم معمارانه است به طوریکه این تصمیم قابلیت اطمینان را به صورت مثبت تحت تاثیر قرار می دهد بنابراین چنین تصمیمی یک نقطه

حساس با توجه به قابلیت اطمینان تلقی می‌شود. با این وجود پشتیبان‌گیری باعث مصرف منابع سیستم شده و بنابراین کارایی را به صورت منفی تحت تاثیر قرار می‌دهد. لذا آن را می‌توان یک نقطه مصالحه بین کارایی و قابلیت اطمینان دانست. اینکه این تصمیم خطر است یا نه؟ بستگی به این دارد که آیا هزینه کارایی در زمینه نیازمندیهای خصوصیات کیفی معماری زیاد است یا نه؟

- *مجموعه‌ای از خطرات و غیرخطرات*[1]

خطر در *ATAM* به عنوان تصمیم معماری تعریف می‌شود که منجر به نتایج نامطلوب در وضعیت نیازمندیهای خصوصیات کیفی می‌شود. به همین صورت غیرخطر نیز تصمیم معماری است که امن در نظر گرفته می‌شود. خطرات شناسایی شده می‌توانند مبنایی را برای برنامه کاهش خطر معمارانه شکل دهند.

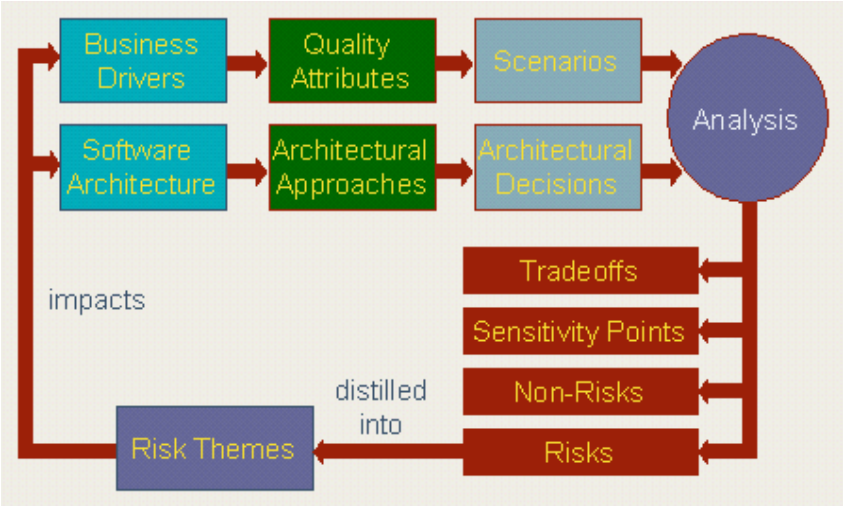
- *مجموعه‌ای از موضوعات خطرات*

زمانی که تحلیل کامل شد تیم ارزیابی کلیه خطرات کشف شده را بررسی می‌کنند تا عناوینی را به دست آورد که ضعف‌های سیستماتیک در معماری و یا حتی در فرآیند و تیم معماری را مشخص می‌کند. اگر با این خطرات برخورد نشود آنها اهداف حرفه پروژه را تهدید خواهند کرد.

از خروجی‌های *ATAM* در تدوین گزارش نهایی به گونه‌ای استفاده می‌شود که در آن روشها بازنگری شده، شرح مذاکرات خلاصه شده، سناریوها و تحلیل‌های مربوط به آنها بدست آورده شده و یافته‌های سیستم نیز فهرست گردند. *ATAM* علاوه بر خروجی‌های ذکر شده، خروجی‌های ثانویه‌ای نیز دارد. اغلب اوقات، نمایش‌های معماری صریحا برای ارزیابی ایجاد می‌شوند و احتمالا نسبت به آنچه که قبلا وجود داشت، بهتر خواهد بود. این مستندات اضافی باعث نجات ارزیابی می‌شود و می‌تواند به عنوان بخشی از میراث (خروجی) سیستم در نظر گرفته شود. همچنین سناریوهای ایجاد شده به وسیله مشارکت کنندگان که اهداف حرفه و نیازمندیها را برای معماری بیان می‌کردند، می‌توانند به عنوان راهنمایی برای توسعه سیستم استفاده شوند. نهایتا، تحلیل انجام شده در گزارش نهایی می‌تواند به عنوان یک شرحی از استدلالها برای تصمیمات معماری اتخاذ شده در نظر گرفته شود. خروجی‌های ثانویه قابل لمس و شمارش‌پذیر هستند.

[1] Nonrisks

ارزیابی مبتنی بر *ATAM* خروجی‌های غیرقابل لمس نیز دارد که دربرگیرنده حس قوی ارتباط بین ذینفعان، کانال ارتباطی باز بین معمار و ذینفعان و بهتر از همه درک مشارکت کنندگان معماری، ضعف‌ها و قدرت آن است. اگرچه اندازه‌گیری این نتایج بسیار مشکل است ولی کم اهمیت‌تر از سایر نتایج نیستند و غالبا تاثیر طولانی مدتی نیز دارند. شکل ۱۱-۱ یک دید کلی از خروجی‌های *ATAM* را نشان می‌دهد.



شکل ۱۱-۱: خروجی‌های *ATAM*

۱۱-۴- فازهای *ATAM*

فعالیت‌های مربوط به ارزیابی مبتنی بر *ATAM* در چهار فاز گسترده شده است. در فاز صفر که "مشارکت و آماده‌سازی"[1] نامیده می‌شود رهبر تیم ارزیابی و تصمیم گیرنده اصلی پروژه به صورت غیررسمی در جهت کامل کردن جزئیات کار با یکدیگر ملاقات می‌کنند. نمایندگان پروژه به طور مختصر پروژه را برای ارزیابان تشریح کرده تا اینکه تیم بتواند از طریق اشخاصی که خبرگی لازم را دارند، پشتیبانی شود. هر دو تیم بر روی مباحثی از قبیل زمان و مکان ملاقات، چه کسی *flipchart* را آماده کند و چه کسی شیرینی و چائی را فراهم کند، توافق می‌کنند. همچنین توافقی بر روی فهرست اولیه ذینفعان (برحسب نام نه اینکه نقش آنها) و زمان تحویل گزارش نهایی و شخص تحویل گیرنده، صورت می‌پذیرد. هر دو تیم تشریفاتی مانند دستور شروع کار و عدم افشا کردن توافقنامه‌ها را نیز رعایت می‌کنند. تیم تصمیم گیرندگان هر گونه مستند معماری که موجود است و یا می‌تواند مفید واقع شود را به تیم ارزیابی تحویل می‌دهند. نهایتا، رهبر تیم ارزیابی تشریح می‌کند که چه

[1] Partnership and preparation

اطلاعاتی را مدیر و معمار باید در روند فاز اولیه ارائه کنند و در صورت نیاز کمک می‌کند که آنها ارائه‌های خود را ایجاد کنند.

فازهای یک و دو فازهای ارزیابی هستند که در آن تمرکز هر شخص بر روی تحلیل است. در این فاز تیم ارزیابی مستندات معماری را مطالعه می‌کند و ایده‌های خوبی درباره اینکه سیستم چیست؟ روشهای به کار رفته در معماری چیست؟ و خصوصیات کیفی خیلی مهم در سیستم چه مواردی هستند؟ بدست می‌آورند. در روند فاز یک، تیم ارزیابی با تصمیم گیرندگان پروژه ملاقات می‌کنند (معمولا در حدود یک روز) تا شروع به جمع آوری اطلاعات و تحلیل کنند. در فاز دوم، به مدت دو روز ذینفعان معماری به مذاکرات و تحلیل ملحق می‌شوند (جزئیات کامل فاز یک و دو در بخش های بعدی تشریح خواهد شد).

فاز سوم در مورد ایجاد و تحویل گزارش نهایی توسط تیم ارزیابی است. در ملاقات مربوط به بررسی، تیم بحث می‌کند که چه مواردی خوب پیش رفته است و چه مواردی پیشرفت چندانی نداشته است. آنها دست نوشته‌های مشارکت کنندگان در فازهای یک و دو را مطالعه می‌کنند و ناظر فرآیند، گزارشهای مربوطه را تولید می‌کند. اعضای تیم به دنبال بهبودهایی در مورد چگونگی انجام عملکردهای خود هستند تا اینکه ارزیابی بعدی خیلی موثر و ساده‌تر انجام شود. تیم همچنین میزان تلاش لازم برای ارزیابی را در حوزه هر سه دسته از مشارکت کنندگان فهرست می‌کند. بعد از چند ماه (بستگی به نوع پروژه دارد)، رهبر تیم با مشتری ارزیابی ارتباط برقرار می‌کند تا تاثیرات بلند مدت در جهت اینکه هزینه و سود قابل قیاس با یکدیگر باشند را مشخص کند. جدول ۱۱-۲ چهار فاز مربوط به *ATAM* را نشان می‌دهد که در آن برای هر فاز مشارکت کنندگان و زمان معمول نیز مشخص شده است.

جدول ۱۱-۲: فازهای *ATAM* و خصوصیات هر یک

فاز	فعالیت	مشارکت کنندگان	زمان معمول
صفر	مشارکت و آماده‌سازی	رهبر تیم ارزیابی و تصمیم گیرندگان کلیدی پروژه	در صورت نیاز اجرا می‌شود (به صورت غیررسمی) و معمولاً چند هفته است
یک	ارزیابی	تیم ارزیابی و تصمیم گیرندگان پروژه	یک روز که با وقفه ۲ یا ۳ هفته‌ای است
دو	ارزیابی (ادامه)	تیم ارزیابی، تصمیم گیرندگان پروژه و ذینفعان	دو روز
سه	پیگیری	تیم ارزیابی و مشتریان ارزیابی	یک هفته

۱۱-۴-۱- مراحل ارزیابی

شکل ۱۱-۲ مراحل *ATAM* را به صورت کلی نشان می‌دهد.

شکل ۱۱-۲: دید کلی از مراحل *ATAM*

فازهای تحلیل *ATAM* (فازهای یک و دو) متشکل از نه مرحله هستند. مرحله اول تا شش در فاز اول انجام می‌شوند. در فاز دوم، همراه با نمایش‌های ذینفعان، مراحل یک تا شش خلاصه شده و همچنین مراحل هفت تا نه انجام می‌شوند. مراحل تحلیل اسماً به صورت ترتیبی و آن هم مبتنی بر یک مجموعه از دستورات انجام می‌شود اما بعضی اوقات باید اصلاحات پویایی در زمانبندی ایجاد شود تا بتوان با در دسترس بودن پرسنل و اطلاعات مربوط به معماری تطبیق ایجاد کرد. هر ارزیابی منحصر به فرد بوده و تیم ارزیابی ممکن است در صورت نیاز به مراحل اولیه برگردد یا یک مرحله را لغو کرده و مرحله بعد از آن را انجام دهد و یا چندین مرحله را به صورت تکراری انجام دهد.

۱۱-۴-۱-۱- مرحله اول: نمایش *ATAM*

در مرحله اول، رهبر ارزیابی *ATAM* را برای نمایندگان پروژه ارائه می‌کند. در واقع از این مرحله به منظور تشریح فرآیندی که هر شخص آن را انجام خواهد داد و پاسخگویی به سئوالات و ایجاد زمینه و انتظاراتی برای باقیمانده فعالیت‌ها، استفاده می‌شود. بنابراین رهبر با استفاده از ارائه استانداردی، مراحل *ATAM* و خروجی‌های آن را تشریح خواهد کرد.

۱۱-۴-۴-۲- مرحله دوم: نمایش پیشرانهای حرفه

هر شخص درگیر در ارزیابی (نماینده پروژه و اعضای تیم پروژه) نیاز به درک زمینه سیستم و پیشرانهای منجر به توسعه سیستم را دارد. در این مرحله تصمیم گیرنده پروژه (مدیر پروژه یا مشتری سیستم)، سیستم را از بعد حرفه تشریح می‌کند. ارائه‌ای که تصمیم گیرنده پروژه انجام می‌دهد باید مطالب زیر را تشریح کند:

- عملکردهای بسیار مهم سیستم
- هر گونه محدودیت‌ها سیاسی، اجتماعی، مدیریتی و تکنیکی مربوط به سیستم
- اهداف حرفه و زمینه‌های مرتبط به سیستم
- ذینفعان اصلی
- پیشرانهای معماری (یعنی اهداف خصوصیات کیفی اصلی که معماری را شکل می‌دهند)

۱۱-۴-۱-۳- مرحله سوم: نمایش معماری

در این مرحله سرپرست معماری (یا تیم معماری)، معماری سیستم را با سطح جزئیات مناسب تشریح می‌کند. میزان یا سطح جزئیات به چندین فاکتور بستگی دارد: چقدر از معماری طراحی و مستندسازی شده است؟ چقدر زمان لازم است؟ ماهیت رفتار و خصوصیات کیفی چیست؟

معمار در ارائه‌ای که انجام می‌دهد محدودیت‌های تکنیکی از قبیل سیستم عامل، سخت‌افزار، میان‌افزار تعیین شده برای کاربر و دیگر سیستمهایی که باید با آنها ارتباط برقرار شود را پوشش می‌دهد. همچنین خیلی مهم است که معمار حتما روشهای معمارانه یا الگوهای استفاده شده برای برآورده کردن نیازمندیها را تشریح کند (بهتر است زمانی الگوها تشریح شوند که واژگان استفاده شده در معماری ساده باشند). به منظور استفاده از زمان محدود، ارائه معماری باید حاوی اطلاعات مهم و اصلی باشد. یعنی اینکه باید ماهیت معماری نشان داده

شود و از پرداختن به جزئیات عمیق یک جنبه خاص و یا بیان کردن اطلاعات غیر مرتبط خودداری شود. بنابراین تشریح اطلاعات مورد نیاز برای تیم ارزیابی آن هم از جانب معمار بسیار سودمند است. در شکل ۱۱-۳ قالبی نشان داده شده است که کمک می‌کند معمار بتواند ارائه خود را آماده کند. همچنین با توجه به نظر معمار، آخرین ارائه انجام شده می‌تواند به عنوان بخشی از فعالیت فاز اول در نظر گرفته شود.

- این قالب ۲۰ اسلاید و مدت زمان ۶۰ دقیقه را برای ارائه معماری در نظر می‌گیرد.
- در ۲ تا ۳ اسلاید، نیازمندیهای اصلی معماری، خصوصیات قابل اندازه‌گیری مرتبط با نیازمندیهای اصلی و هرگونه روشها، مدلها و استانداردهای به کار گرفته شده برای تحقق آنها ارائه می‌شوند.
- اطلاعات مهم معماری در ۴ تا ۸ اسلاید ذکر می‌شوند.
  - *نمودار متن*: این نمودار، زمینه یا متنی را که سیستم در آینده در آن قرار می‌گیرد و همچنین اشخاص و سیستمهایی که با آنها تعامل خواهد داشت را مشخص می‌کند.
  - *دید لایه‌بندی یا ماژول*: در این قسمت باید مواردی از قبیل: ماژولهایی (که می‌تواند لایه یا زیرسیستم در نظر گرفته شوند) که تجزیه عملکردی سیستم را نشان می دهند همراه با اشیاء، رویه‌ها، توابع و ارتباطات بین آنها (به عنوان مثال، فراخوانی رویه، درخواست متد و غیره)، نمایش داده شوند
  - *دید مولفه و اتصال*: در این قسمت باید فرآیندها و نخها همراه با همگام‌سازی، جریان داده و رویدادها که آنها را به یکدیگر متصل می‌کند، نشان داده شود.
  - *دید استقرار*: پردازشگرها، رسانه‌های ذخیره‌سازی و سنسورها و رسانه‌های داخلی همراه با رسانه‌های ارتباطی و شبکه‌هایی که آنها را به یکدیگر متصل می‌کنند باید در این قسمت نمایش داده شوند. همچنین باید مشخص شود که فرآیندها بر روی کدام پردازشگرها اجرا خواهند شد.
- در۳ تا ۶ اسلاید، روشهای معمارانه، الگوها و تاکتیک‌های به کار گرفته شده، انواع خصوصیات کیفی و چگونگی برآورده شدن خصوصیات کیفی تشریح می‌شوند.
  - در ۱ تا ۲ اسلاید، مولفه‌های آماده برای استفاده به کار رفته و همچنین چگونگی انتخاب و یکپارچگی آنها تشریح شود.
  - در ۱ تا ۳ اسلاید، یک الی سه سناروی موارد کاربری بسیار مهم نمایش داده شود و اگر امکان‌پذیر باشد منابع استفاده در زمان اجرا هر یک از آنها نیز نمایش داده شود.
  - در ۱ تا ۳ اسلاید، یک الی سه سناریوی مربوط به تغییرات نمایش داده شود و اگر امکان‌پذیر باشد تاثیر تغییرات بر ماژولهای دیگر و واسطها نیز تشریح شود.
  - در ۲-۳ اسلاید خطرات و پیامدهای معماری با توجه به نیازمندیهای اصلی معماری تشریح شود.
  - در ۱ اسلاید واژه‌نامه مربوط به سیستم نمایش داده شود.

شکل ۱۱-۳: قالبی برای نمایش معماری

همان طور که از قالب معرفی شده قابل استنباط است دیدهای معماری اصلی‌ترین ابزار برای معمار هستند تا بتواند از طریق آنها معماری را توصیف کند. نمودار متن، دیدهای مولفه و اتصال، دیدهای لایه‌بندی و تجزیه ماژول و دید استقرار تقریبا برای هر نوع روش ارزیابی سودمند هستند و معمار باید آنها را در ارائه خود لحاظ کند. دیدهای دیگر نیز می‌توانند سودمند باشند به شرط آنکه شامل اطلاعات مناسب در مورد معماری باشند، مخصوصا اگر آنها شامل اطلاعاتی در مورد دستیابی به اهداف خصوصیات کیفی مهم سیستم باشند. بنابراین به عنوان قانون کلی، معمار باید دیدهایی را ارائه کند که در روند ایجاد معماری به این نتیجه رسیده است که آنها اهمیت زیادی دارند.

در روند ارائه معماری، تیم ارزیابی به منظور آشکارسازی اطلاعات بدست آمده از مستندات معماری و پیشرانهای معماری، سئوالاتی از معمار خواهد پرسید. آنان همچنین به گفته‌های معمار گوش داده و الگوها و تاکتیک‌های به کار گرفته شده در سیستم را یاداشت می‌کنند.

۱۱-۴-۱-۴- مرحله چهارم: تعیین روشهای معمارانه

تمرکز *ATAM* بر روی تحلیل معماری از طریق درک روشهای معمارانه آن است. همان طور که در فصل پنجم بیان شد الگوهای معماری در زمینه شناخت روشهایی که هر یک خصوصیات کیفی خاصی را تحت تاثیر قرار می‌دهند، بسیار کارآمد هستند. به عنوان مثال، الگوی لایه‌بندی قابلیت حمل را از طریق تحت تاثیر قراردادن کارایی به ارمغان می‌آورد (باعث کاهش کارایی می‌شود) و الگوی مخزن داده نیز معمولا قابلیت گستردگی برای مصرف کنندگان و تولیدکنندگان داده به ارمغان می آورد.

در این وضعیت، تیم ارزیابی دید خوبی نسبت به الگوها و روشهای به کار گرفته شده توسط معمار در طراحی سیستم دارد. آنها مستند معماری را مطالعه کرده و همچنین ارائه معمار را نیز خواهند شنید. در طول ارائه معماری، از معمار خواسته می‌شود که به طور صریح نام الگوها و روشهای استفاده شده را ذکر کند با این وجود اعضای تیم باید در بدست آوردن نکات مهم مربوط به معماری که به صورت صریح ذکر نشده‌اند، ماهر باشند. در این مرحله کوتاه، تیم ارزیابی به سادگی الگوها و روشهای به کار گرفته را فهرست می‌کند. در واقع فهرست توسط یک منشی نوشته می‌شود و در اختیار همه قرار داده می‌شود تا آن را مشاهده کرده و پایه‌ای برای کارهای بعدی قرار دهند.

۱۱-۴-۱-۵- مرحله پنجم: ایجاد درخت سودمندی خصوصیات کیفی

معماری با توجه به قابلیتش در مورد برآورده کردن خصوصیات کیفی خاص در سیستمی که از روی آن ایجاد می‌شود، ممکن است مناسب یا نامناسب باشد. معماری با کارایی بالا احتمالا به صورت کلی برای سیستمی که کارایی در آن از اهمیت زیادی برخوردار نیست، نامناسب است. اهداف خصوصیات کیفی مهم برای سیستمی که ارزیابی بر روی آن اجرا می‌شود در مرحله دوم معرفی می‌شوند، اما هیچ درجه‌ای یا میزانی از کیفیت برای آنها مشخص نشده است. اهداف گسترده مانند قابلیت تغییرپذیری یا توان عملیاتی بالا[1] و یا توانایی اتصال[2] به تعدادی از ماشین‌ها، زمینه و راهنمایی مهمی را ایجاد می‌کند و پیش زمینه‌ای درباره اینکه چه اطلاعات بعدی باید ارائه شود، فراهم می‌کند. اما مشخصه خاصی وجود ندارد که در آن ذکر شده باشد که آیا معماری کفایت می‌کند؟ یا قابلیت تغییرپذیری در چه روشهایی دارد؟ یا توان عملیاتی چگونه بالا رفته است؟ و یا به چه ماشین‌هایی متصل می‌شود؟

در این مرحله، اهداف خصوصیات کیفی از طریق یک سازوکار، مشهور به "درخت سودمندی" به صورت کامل بیان می‌شوند. در این مرحله، تیم ارزیابی با تصمیم گیرندگان پروژه همکاری می‌کنند تا اهداف خصوصیات کیفی مهم سیستم را شناسایی، اولویت‌بندی و تصحیح کرده و سپس آنها را با استفاده از سناریوها نشان دهند. درخت سودمندی به این دلیل به کار گرفته شده است تا نیازمندیها را عینی[3] کرده و نماینده مشتری و معمار مجبور شوند تا نیازمندیهای کیفی مناسب را که در حال تلاش برای فراهم کردن آنها هستند را دقیق‌تر تعریف کنند.

درخت سودمندی با سودمندی به عنوان ریشه آغاز می‌شود. سودمندی یک عبارت از خوب بودن کلی سیستم است. خصوصیات کیفی، سطح دوم را شکل می‌دهند زیرا آنها مولفه‌های سودمندی هستند. خصوصیات کیفی نامگذاری شده در نمایش پیشرانهای حرفه در مرحله دوم، مجموعه عناصر اولیه سطح دوم را ایجاد می‌کنند. معمولا، کارایی، قابلیت تغییرپذیری، امنیت، قابلیت استفاده و قابلیت دسترسی فرزندان ریشه (سودمندی) هستند اما مشارکت کنندگان آزاد هستند که آنها را به میل خود نامگذاری کنند. گاهی اوقات گروه‌های ذینفعان مختلف از نامهای متفاوت برای بیان ایده‌های یکسان استفاده می‌کنند (برای مثال، بعضی ذینفعان ترجیح می‌دهند از کلمه

[1] High throughput
[2] Ported
[3] Conceret

قابلیت نگهداری[1] به جای قابلیت تغییرپذیری استفاده کنند). گاهی اوقات هم ذینفعان نامهایی را برای خصوصیات کیفی استفاده می‌کنند که برای فرهنگ خودشان معنی‌دار است و زیاد در جاهایی دیگر استفاده نمی‌شود (مانند *flextensibility*). نام‌هایی که ذینفعان معرفی می‌کنند به شرطی مناسب خواهد بود که ذینفعان بتوانند تشریح کنند که معنی آن در سطح بعدی فرآیند پالایش چیست؟

برای هر کدام از خصوصیت کیفی که به عنوان فرزند ریشه هستند خصوصیات کیفی پالایش شده خاص وجود دارند. برای مثال، کارایی می‌تواند به "تاخیر داده"[2] و "توان عملیاتی تراکنش"[3] تجزیه شود. در این مرحله هدف، پالایش اهداف کیفی به سناریوهای خصوصیات کیفی است که به اندازه کافی مشخص و واضح برای اولویت‌بندی و تحلیل هستند. "تاخیر داده" می‌تواند بیشتر پالایش شده و به "تاخیر ذخیره‌سازی نهایی بر روی پایگاه‌داده مشتری تا ۲۰ میلی ثانیه" و "نمایش ۲۰ قاب/ثانیه ویدیو در زمان واقعی) تجزیه شود. دلیل این امر نیز آن است که هر دو نوع از "تاخیر داده" به سیستم مرتبط هستند.

سناریوها مکانیزمی هستند که به وسیله آن توصیف مبهم و گسترده خصوصیات کیفی مطلوب سیستم، خاص و قابل آزمایش می‌شوند. آنها برگ‌های درخت سودمندی را شکل می‌دهند که از طریق خصوصیات کیفی بیان کننده آنها، احاطه شده‌اند. سناریوهای *ATAM* از سه بخش تشکیل شده است:

- *محرک:* چه شرایطی به سیستم وارد می‌شود، چه کسی آن را تولید می‌کند و چه فرآورده سیستمی را تحریک می‌کند
- *محیط:* چه اتفاقی در طول زمان می‌افتد
- *پاسخ:* پاسخ سیستم به محرک در یک روش قابل اندازه‌گیری بیان می‌شود

حال موارد قابل لمسی در برابر ارزیابی معماری وجود دارد. درحقیقت، مراحل تحلیل *ATAM* متشکل از انتخاب یک سناریو در هر لحظه و بدست آوردن میزان پاسخگویی معماری به آن است (میزان دستیابی به سناریو). موارد بیشتر بر روس سناریوها در مراحل بعدی انجام خواهد شد.

بعضی سناریوها می‌توانند بیش از یک خصوصیت کیفی را بیان کنند و بنابراین می‌توانند در بیش از یک مکان در درخت سودمندی ظاهر شوند. این موضوع لزوما یک مشکل نیست اما رهبر ارزیابی باید از سناریوهایی که

[1] Maintainability
[2] Data latency
[3] Transaction throughput

سعی در پوشش چندین حوزه دارند، جلوگیری کند زیرا تحلیل آنها خیلی دشوار است. بدین منظور می‌توان چنین سناریوهایی را به چندین بخش تقسیم کرد تا هر کدام دغدغه‌های کوچکتری را تحت تاثیر قرار دهند.

تیم ارزیابی علاوه بر اینکه نیاز دارد تا اهداف دقیق موجود در معماری را درک کند، باید اهمیت نسبی آنها را نیز درک کند. درخت سودمندی به راحتی می‌تواند شامل پنجاه سناریو در برگ‌ها باشد و زمانی نیز برای تحلیل آنها در ارزیابی معماری وجود نخواهد داشت. لذا درخت سودمندی شامل مرحله اولویت‌بندی است. تصمیم گیرندگان با توافق یکدیگر می‌توانند به هر سناریو اولویت دهند. اولویت‌بندی می‌تواند به صورت مقیاس عددی بین ۰ تا ۱۰ در نظر گرفته شود و یا اینکه از کلمات "بالا"، "پائین" و "متوسط" استفاده شود (استفاده از حالت دوم بسیار مناسب است زیرا به خوبی با توجه به تیمهای مختلف عمل کرده و زمان کمتری نیز برای تخصیص به خود اختصاص می‌دهد). بعد از اولویت‌بندی اولیه، اولویت‌بندی ثانویه دیگری نیز این بار با استفاده از معیارهای دیگر بر روی سناریوها انجام می‌شود. به عنوان مثال، یکی از معیارها می‌تواند میزان سختی دستیابی آن سناریو از طریق معماری باشد. برای اولویت‌بندی ثانویه نیز می‌توان باز از دو روش پیشنهادی استفاده کرد.

حال هر سناریو یک جفت مرتب به صورت *(H, H), (H, M), (H, L)* و غیره دارد. سناریوهایی که بسیار مهم و خیلی دشوار هستند مواردی خواهند بود که وقت تحلیل بسیاری را به خود اختصاص می‌دهند و بقیه سناریوها به عنوان بخشی از رکود نگهداری می‌شوند. یک سناریویی که به صورت نه چندان مهم مورد ارجاع قرار می‌گیرد (به صورت *(L, *)*) و سناریوی که خیلی قابل دستیابی است (به صورت *(*, L)*)، توجه زیادی را به خود جلب نمی‌کنند.

خروجی درخت سودمندی تولید شده فهرستی از اولویت‌ها است که به عنوان برنامه یا طرح برای مراحل بعدی ارزیابی *ATAM* در نظر گرفته می‌شود. چنین خروجی مشخص می‌کند که کجاها تیم ارزیابی باید وقت خود را صرف کنند، مخصوصا اینکه در کجا خطرات و روشهای معمارانه را جستجو کنند. درخت سودمندی، ارزیابان را به سمت روشهای معمارانه برای برآورده کردن سناریوهای اولویت بالا در برگ‌های خودش هدایت می‌کند.

در این نقطه از ارزیابی، کلیه اطلاعات مورد نیاز برای تحلیل در یک جدول قرار دارد. جدول شامل خصوصیات مورد انتظار مهم معماری است که از مرحله دوم (پیشرانهای حرفه)، مرحله پنجم (درخت سودمندی) و مرحله سوم (نمایش معماری) و مرحله چهارم (دسته‌بندی روشها استفاده شده) حاصل شده‌اند.

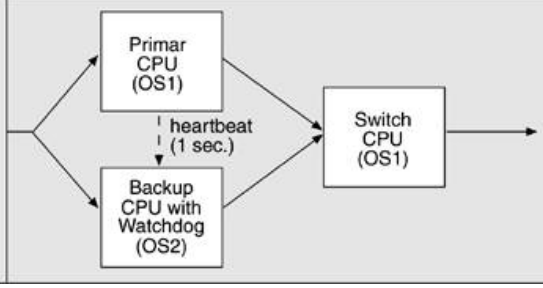
۱۱-۴-۱-۶- مرحله ششم: تحلیل روشهای معمارانه

در این مرحله تیم ارزیابی سناریوها با رتبه بالا را یک به یک بررسی می‌کند. در این روند از معمار خواسته می‌شود تا بیان کند که چگونه معماری هر کدام از سناریوها را پشتیبانی می‌کند. اعضای تیم (مخصوصا پرسشگرها) بررسی خود را بر روی شناخت روشهای معمارانه‌ای متمرکز می‌کنند که معمار برای تحقق سناریوها از آنها استفاده کرده است. همراه با این عملیات، تیم ارزیابی تصمیمات معماری مربوطه را مستندسازی و خطرات، غیرخطرات، نقاط حساس و مصالحه‌ها را شناسایی و دسته‌بندی می‌کند. برای روشهای مشهور و مشخص تیم ارزیابی از معمار سئوال می‌کند که چگونه بر ضعف‌های آن روش چیره گشته و یا اینکه چگونه به این تضمین دست پیدا کرده که روش مورد نظر نیاز مربوطه را برآورده می‌کند. در واقع هدف تیم ارزیابی آن است که متقاعد شود گونه‌ای از روش بکار گرفته شده برای برآورده ساختن نیازهای خاص خصوصیات کیفی مناسب است.

برای مثال تعدادی از سرویس‌گیرندگان همزمان پایگاه‌داده می‌توانند تعداد تراکنش‌هایی که پایگاه‌داده در هر ثانیه پردازش می‌کند را تحت تاثیر قرار بدهد. بنابراین، تخصیص سرویس‌گیرندگان به سرویس‌دهنده با توجه به اینکه پاسخ برحسب تراکنش بر ثانیه اندازه‌گیری می‌شود، یک نقطه حساس به شمار می‌رود. بعضی از تخصیص‌ها منجر به یک پاسخ غیرقابل قبول می‌شوند که اینها مشخص کننده خطرات هستند. زمانی که آشکار شد یک تصمیم معماری نقطه حساس برای بیش از یک خصوصیت کیفی است آن نقطه به عنوان نقطه مصالحه در نظر گرفته می‌شود. بررسی سناریوها منجر به بحث در مورد خطرات ممکن، غیرخطرات، نقاط حساس و نقاط مصالحه می‌شود. این موضوعات نیز به نوبه خود می‌تواند به تحلیل‌های عمیق‌تر منجر شوند که آن نیز بستگی به چگونگی پاسخ دادن معمار به سئوالات دارد. برای مثال، اگر معمار نتواند تعداد سرویس‌گیرندگان را مشخص کند یا نتواند مشخص کند که چگونه تعادل بار از تخصیص پردازشگرها به سخت‌افزار حاصل می‌شود، نقطه کوچکی در صف‌بندی پیچیده[1] یا تحلیل کارایی با نرخ یکنواخت[2] وجود دارد. اگر یک چنین سئوالاتی بتوانند پاسخ داده شوند تیم ارزیابی می‌تواند دست کم یک تحلیل اولیه انجام دهد تا مشخص شود آیا این تصمیمات معماری مشکلاتی در مقابل نیازمندیهای خصوصیات کیفی که آنها برآورده می‌کنند، دارند یانه؟

[1] Sophisticated queuing
[2] Rate-monotonic performance analysis

تحلیل نیاز ندارد که به صورت جامع انجام شود بلکه هدف اصلی از آن این است که اطلاعات معماری کافی برای ایجاد بعضی ارتباطات بین تصمیمات معماری که اعمال شده‌اند و نیازمندیهای خصوصیات کیفی که نیاز به برآورده کردن دارند، استخراج شود.

Scenario #: A12	Scenario: Detect and recover from HW failure of main switch.			
Attribute(s)	Availability			
Environment	Normal operations			
Stimulus	One of the CPUs fails			
Response	0.999999 availability of switch			
Architectural decisions	Sensitivity	Tradeoff	Risk	Nonrisk
Backup CPU(s)	S2		R8	
No backup data channel	S3	T3	R9	
Watchdog	S4			N12
Heartbeat	S5			N13
Failover routing	S6			N14
Reasoning	Ensures no common mode failure by using different hardware and operating system (see Risk 8) Worst-case rollover is accomplished in 4 seconds as computing state takes that long at worst Guaranteed to detect failure within 2 seconds based on rates of heartbeat and watchdog Watchdog is simple and has proved reliable Availability requirement might be at risk due to lack of backup data channel ... (see Risk 9)			
Architecture diagram				

شکل ۱۱-۳: مثالی از تحلیل روش معمارانه

به عنوان مثال شکل ۱۱-۳ یک قالبی را برای ثبت تحلیل روش معمارانه برای یک سناریو نشان می‌دهد. همان طور که نشان داده شده است مبتنی بر نتایج این مرحله، تیم ارزیابی می‌تواند مجموعه‌ای از نقاط حساس، مصالحه، خطرات و غیرخطرات را شناسایی و ثبت کند. کلیه نقاط حساس و مصالحه، نامزد خطرات هستند. در پایان در *ATAM* کلیه این خطرات باید دسته‌بندی شوند. خطرات، غیرخطرات، نقاط حساس و نقاط مصالحه هر کدام در یک فهرست جداگانه قرار می‌گیرند. شماره‌های *R8, T3, S4, N12* و الی آخر در شکل ۱۱-۳ به

سادگی به این نوع فهرست اشاره دارند. در پایان این مرحله، تیم ارزیابی یک تصویر واضح از جنبه‌های مهم کل معماری، استدلالهایی درباره هر تصمیم کلیدی طراحی و فهرستی از خطرات، غیرخطرات، نقاط حساس و نقاط مصالحه بدست خواهد آورد.

بعد از اینکه فاز اول به پایان رسید تیم ارزیابی خلاصه‌ای از آنچه که تا این فاز بدست آورده، بررسی مجدد می‌کند و به صورت غیررسمی (معمولا با تلفن) با معمار در طول وقفه یک یا دو هفته‌ای تعامل می‌کند. بسیاری از سناریوها می‌تواند در این بازه زمانی تحلیل شوند و اگر مطلوب باشد سئوالات مربوط به آنها می‌تواند حل شوند. زمانی که تصمیم گیرندگان پروژه آماده ادامه هستند و ذینفعان نیز گرد هم آمده‌اند فاز دوم می‌تواند شروع شود. این فاز از طریق مشخص شدن فهرست گسترده‌ای از مشارکت کنندگان و ذینفعان جدید اضافه شده، رسمیت پیدا می‌کند. ابتدا فاز اول تکرار می‌شود تا ذینفعان روشها و نقشی که آنها در معماری ایفا می‌کنند را درک کنند. سپس رهبر ارزیابی مروری بر نتایج مراحل دو تا شش انجام می‌دهد و فهرست مربوط به خطرات، غیرخطرات، نقاط حساس و نقاط مصالحه را در اختیار سایرین قرار می‌دهد. حال ذینفعان با نتایج بدست آمده از کار تیم ارزیابی آشنا شده و بنابراین سه مرحله باقیمانده می‌تواند شروع شد.

۱۱-۴-۱-۷- مرحله هفتم: طوفان ذهنی و اولویت‌بندی سناریوها

همان گونه که از درخت سودمندی، اصولا برای درک چگونگی کنترل و مدیریت پیشرانهای خصوصیات کیفی معماری استفاده می‌شد، هدف از طوفان ذهنی آن است که نکات راه‌اندازی را از مجموعه بزرگ ذینفعان بدست آورد. طوفان ذهنی سناریوها در گروه‌های بزرگ خیلی خوب عمل می‌کند. زیرا باعث ایجاد فضایی می‌شود که ایده و نظر یک شخص، ایده‌های دیگری را تحریک می‌کند. این فرآیند خلاقیت و ارتباط را پرورش می‌دهد و باعث بیان ذهن جمعی[1] مشارکت کنندگان می‌شود. فهرست اولویت‌بندی شده سناریوهای طوفان ذهنی مشابه با سناریوهای اولویت‌بندی شده درخت سودمندی هستند. اگر آنها مطابقت داشته باشند این موضوع اشاره به تطبیق خوب بین چیزی که معمار در ذهن دارد و چیزی که ذینفعان واقعا می‌خواهند، دارد. اگر سناریوهای پیشران اضافی کشف شود این موضوع می‌تواند یک خطر باشد که نشان می‌دهد ناسازگاری در اهداف بین معمار و ذینفعان وجود دارد.

[1] Collective mind

در این مرحله، تیم ارزیابی از ذینفعان می‌خواهد تا با توجه به نقش خود در سیستم سناریوهایی که نشات گرفته از آشفتگی ذهنی بوده و به صورت عملیاتی قابل درک هستند را بیان کنند. برای مثال یک نگهدارنده احتمالا یک سناریوی نگهداری را پیشنهاد داده و کاربر نیز سناریوهایی پیرامون عملکرد سیستم و راحتی انجام عملیات پیشنهاد می‌دهد. سناریوهای درخت سودمندی که زیاد مورد تحلیل قرار نگرفته‌اند در این مرحله می‌توانند مورد بررسی قرار بگیرند. ذینفعان آزاد هستند که این نوع سناریوها را در جلسه طوفان ذهنی مطرح کنند تا فرصتی حاصل شود که آنها را دوباره بازبینی کرد.

وقتی سناریوها جمع آوری شدند باید اولویت‌بندی شوند. دلیل اولویت‌بندی در این مرحله مشابه دلیلی است که برای اولویت‌بندی در درخت سودمندی مطرح شد (تیم ارزیابی نیاز دارد بداند که کجاها باید وقت محدود خود را صرف کند). بدین منظور نخست از ذینفعان خواسته می‌شود سناریوهایی را که احساس می‌کنند دارای رفتار یکسان یا مرتبط با یک دغدغه یکسانی هستند را با یکدیگر ادغام کنند سپس آنها به سناریوهایی که احساس می‌کنند خیلی مهم هستند رای می‌دهند. هر ذینفع معادل با سی درصد تعداد سناریو، رای می‌دهد. بنابراین اگر چهل سناریو بدست آمده باشد هر ذینفع شش رای می‌تواند بدهد. آراء می‌توانند در هر روشی تخصیص یابد. به عنوان مثال یک ذینفع می‌تواند هر شش رای خود را به یک سناریو دهد و یا اینکه به هر سناریو یک رای دهد.

۱۱-۴-۱-۸- مرحله هشتم: تحلیل روشهای معمارانه

بعد از اینکه سناریوها جمع آوری و اولویت‌بندی شدند تیم ارزیابی معمار را در فرآیند اجرای سناریوهایی با رتبه بالا که از مرحله هفت بدست آمده‌اند، راهنمایی می‌کند. معمار تشریح می‌کند که چگونه تصمیمات مربوط به معماری برای تحقق هر کدام از سناریوها به کار گرفته شده‌اند. در این مرحله تیم ارزیابی فعالیت‌های مشابه با مرحله شش را که نگاشت سناریوهای جدید و دارای بالاترین رتبه به فرآورده‌های معماری تا به حال پوشش داده نشده است، تکرار می‌کند.

۱۱-۴-۱-۹- مرحله نهم: نمایش نتایج

نهایتا، ضروری است که اطلاعات جمع آوری شده از *ATAM* خلاصه شده و دوباره برای ذینفعان ارائه شود. معمولا این ارائه به صورت شفاهی همراه با اسلاید است اما می‌تواند همراه با یک سری گزارشات جامع و

قابل درک درباره ارزیابی *ATAM* باشد. در این ارائه، رهبر ارزیابی مراحل *ATAM* و همچنین کلیه اطلاعات بدست آمده از این مراحل شامل زمینه حرفه، نیازمندیهای پیشران، محدودیت‌ها و معماری را مرور می‌کند. رهبر ارزیابی خروجی‌های زیر را ارائه می‌دهد:

- روشهای معمارانه مستند شده
- مجموعه‌ای از سناریوها و اولویت‌بندی آنها که از طوفان ذهنی بدست آمده‌اند
- درخت سودمندی
- خطرات کشف شده
- غیرخطرات مستند شده
- نقاط حساس و نقاط مصالحه یافت شده

کلیه این خروجی‌ها در طول ارزیابی مشخص، دسته‌بندی و برای عموم آشکار شده‌اند. اما در مرحله نهم، تیم ارزیابی برای خطرات شناسایی شده سیستم ارزش‌گذاری می‌کند و این کار مبتنی بر بعضی دغدغه‌های پایه‌ای مشترک یا کمبودهای سیستمی انجام می‌شود (ابتدا خطرات در یک عنوان خطر[1] طبقه‌بندی می‌شوند و بعد به هر عنوان خطر یک ارزش یا اولویت داده می‌شود). برای مثال، یک گروه از خطرات درباره عدم کافی بودن یا عدم به روز بودن مستندات می‌تواند در داخل یک عنوان خطر فهرست شود که بیان می‌کند مستندات معرفی شده ملاحظات کافی را ندارد و یا گروهی از خطرات درباره عدم توانایی سیستم در برخورد با انواع مختلف شکست‌های سخت‌افزاری یا نرم‌افزاری می‌تواند منجر به یک عنوان خطر درباره عدم توجه به قابلیت پشتیبانی یا فراهم کردن قابلیت دسترسی بالا شود. برای هر عنوان خطر، تیم ارزیابی شناسایی می‌کند که کدام یک از پیشرانهایی حرفه فهرست شده در مرحله دوم، تحت تاثیر قرار می‌گیرند. شناسایی عناوین خطرات و مرتبط کردن آنها به پیشرانهای خاص، حلقه کامل ارزیابی را از طریق مرتبط کردن نتایج نهایی به نمایش‌های اولیه به ارمغان می‌آورد. بنابراین خاتمه رضایت بخشی را برای عملیات به همراه می‌آورد. جدول ۱۱-۳ نه مرحله مربوط به *ATAM* را خلاصه کرده و نشان می‌دهد که چگونه هر مرحله در خروجی *ATAM* نقش ایفا می‌کند. "××" در شکل به این معنی است که مرحله مربوطه مشارکت کننده اصلی برای خروجی است. "×" نیز مشخص می‌کند که مرحله مربوطه نقش ثانویه را ایفا می‌کند.

[1] Risk theme

جدول ۱۱-۳: مراحل و خروجی‌های *ATAM*

مراحل	دستورات اولیت‌بندی شده نیازمندیهای خصوصیات کیفی	فهرست روشهای معمارانه استفاده شده	پرسشهای خاص تحلیل روشها و خصوصیات کیفی	نگاشت روشهای معمارانه به خصوصیات کیفی	خطرات و غیرخطرات	نقاط حساس و نقاط مصالحه
۱.نمایش *ATAM*						
۲.نمایش پیشرانهای حرفه	× [آ]					× [ب]
۳.نمایش معماری		××			× [پ]	× [ت]
۴.تعیین روشهای معمارانه		××	××		× [ث]	× [ج]
۵.ایجاد درخت سودمندی خصوصیات کیفی	××					
۶.تحلیل روشهای معمارانه		× [چ]	××	××	××	××
۷.طوفان ذهنی و اولویت‌بندی سناریوها	××					
۸.تحلیل روشهای معمارانه		×	××	××	××	××
۹.نمایش نتایج						

[آ] : پیشرانهای معماری دربرگیرنده نخستین و سطح بالاترین توصیف از خصوصیات کیفی هستند.

[ب]: نمایش پیشرانهای حرفه می‌توانند خطر از قبل شناسایی شده یا قدیمی که باید ثبت می‌شدند را آشکار کند.

[پ]: معمار می‌تواند یک خطر را در ارائه خود شناسایی کند.

[ت]: معمار می‌توان یک نقطه حساس در ارائه خود شناسایی کند.

[ث]: بسیاری ار روشهای معمارانه خطرات استاندارد دارند.

[ج]: بسیاری از روشهای معمارانه حساسیت‌ها و مصالحه‌های بین خصوصیات کیفی استاندارد دارند.

[چ]: مراحل تحلیل می‌تواند روشهای معمارنه‌ای را شناسایی کند که در مرحله چهار م شناسایی نشده‌اند.

۱۱-۵- مطالعه موردی (سیستم *Nightingle*)

در این بخش *ATAM* به صورت عملی با استفاده از مطالعه موردی مبتنی بر ارزیابی واقعی تشریح خواهد شد. اطلاعات مشخص شده برای این مطالعه موردی به منظور حفظ محرمانگی اطلاعات مشتری تغییر پیدا کرده‌اند.

۱۱-۵-۱- فاز صفر. مشارکت و آماده‌سازی

مشتری یا سازمان برای ارزیابی، یک تولید کننده نرم‌افزار سیستمهای مراقب سلامتی[1] برای بیمارستان‌ها، کلینیک‌ها[2] و آزمایشگاه‌های پزشکی[3] است. سیستم مورد بحث *Nightingle* نامیده می‌شود و جزو سیستمهای بزرگ بوده و دربرگیرنده چندین میلیون خط کد منبع است که باید پیاده‌سازی آن با دقت خوبی انجام شود. سیستم *Nightingle* مشتری اولیه دارد که یک بیمارستان زنجیره‌ای با حدود چهل بیمارستان در سرتاسر جنوب شرقی ایالات متحده است.

چیزی که خواهان دانستن آن هستیم اینکه چرا شرکت تولید کننده (مشتری) علاقه‌مند به ارزیابی معماری سیستم خود است در حالی که قبلا در مورد کار خود به موفقیت دست پیدا کرده است؟ دو دلیل وجود دارد. نخست اینکه اگر به هر دلیلی معماری آن دچار مشکل باشد خیلی بهتر است که مشکل را زودتر کشف کرد. ثانیا، سازمان خواهان آن است که سیستم خود را به چندین مشتری بفروشد ولی این را می‌داند که باید خود را با نیازمندیها، کاربردها و محیط‌های هر کدام از مشتریان وفق دهد. با این وجود، در حالی که معماری می‌تواند برای این موضوع مناسب باشد ولی مشتریان خواهان آن هستند که مطمئن شوند سیستم، قابلیت تغییرپذیری و استحکام لازم را دارد که بتواند به عنوان پایه‌ای برای کل مجموعه محصولات سیستمهای مدیریت مراقبت سلامت در نظر گرفته شود.

سیستم باید به عنوان پایه اطلاعات اصلی برای موسسه‌های مراقب سلامتی باشد که سیستم در آنها نصب می‌شود. لذا باید داده‌هایی در مود تاریخچه مراقبت بیماران، ردگیری بیمه آنها و دیگر پرداختی آنان فراهم کند. همچنین باید قابلیت نگهداری اطلاعات به صورت یک مخزن را داشته باشد تا بتوان از طریق آن یک سری

[1] Health care systems software
[2] Clinic
[3] Health maintenance organization (HMO)

پیش‌بینی‌ها را انجام داد (مانند پیش‌گویی در مورد بروز یک بیماری خاص). سیستم باید تعداد زیادی از گزارشات را به صورت دوره‌ای یا بر حسب تقاضا تولید کند که هر کدام از این گزارشات مورد نیاز بخش خاصی از موسسه است. برای بیمارانی که خودشان باید هزینه را تقبل کنند سیستم باید جریان کاری مرتبط با میزان پرداخت اولیه و میزان وام پرداختی به بیمار را کنترل کند. علاوه بر این، از آنجایی که سیستم در کلیه بخشهای تسهیلاتی موسسه مراقبت سلامت اجرا خواهد شد باید قادر باشد که به نیازهای پیکربندی مربوط به هر بخش پاسخ دهد. به عنوان نمونه، بخشهای مختلف می‌توانند پیکربندیهای سخت‌افزاری متفاوتی داشته و یا اینکه نیاز به گزارشات مختلفی داشته باشند. همچنین یک کاربر ممکن است از یک محل به محل دیگر منتقل شود بنابراین سیستم باید قادر باشد نام کاربری و مشخصات موردنیاز مرتبط با آن را تشخیص دهد، بدون توجه به این که آن کاربر به کدام محل یا بخش منتقل شده است.

مذاکرات پیرامون امضأ قرارداد شروع به کار این سیستم در حدود یک ماه به طول انجامید و بعد از کامل شده آن، یک گروه شش نفره برای ارزیابی معماری تشکیل شد و نقشها به هرکدام از اعضای تیم به صورت جدول ۱۱-۴ تخصیص داد شد.

جدول ۱۱-۴: تخصیص نقشها به تیم ارزیابی

عضو	نقش
۱	رهبر تیم، رهبر ارزیابی، پرسشگر
۲	رهبر ارزیابی، پرسشگر
۳	وقت نگهدار، پرسشگر
۴	نویسنده سناریو، پرسشگر، گرد آوردنده داده
۵	پرسشگر، اجرا کننده فرآیند
۶	نویسنده شرح مذاکرات، ناظر فرآیند

به منظور انجام عملیات، دو رهبر ارزیابی در نظر گرفته شده است تا در شرح مذاکرات بتوانند تسهیلات متفاوتی را فراهم کنند. این طرح در مورد کاهش استرس و خستگی بسیار کارآمد است و منجر به نتایج بهتر می‌شود. با توجه به آشنایی تیم ارزیابی با مفاهیم کارایی و قابلیت نگهداری، ارزیابی با پرسش سئوالاتی پیرامون این دو خصوصیت کیفی آغاز شد. همچنین تعدادی نیروی جدید که آشنایی خوبی در مورد مولفه‌های آماده

برای استفاده داشتند، به کار گرفته شدند. زیرا مشتری اعلام کرده بود که در سازمان یک سری از بسته‌های نرم‌افزاری تجاری وجود دارد که از آنها برای توسعه سیستم *Nightingle* استفاده شده است. خوشبختانه در تیم ارزیابی شخصی وجود داشت که قبلا در زمینه سیستمهای مراقبت سلامتی تجربه داشته است.

یک جلسه متشکل از تیم ارزیابی، مدیر پروژه، معمار ارشد و مدیر پروژه مشتری اولیه *Nightingle* تشکیل شد. سه شرکت‌کننده تصمیم گیرندگان نهایی *Nightingle* هستند. در جلسه، اطلاعات بیشتری در مورد قابلیت‌های *Nightingle*، نیازمندیها و فهرستی از مستندات معماری قابل دسترس بدست آورده شد همچنین یک فهرستی از ذینفعان برای فاز دوم تهیه و بر روی محتوا آن توافق شد. به علاوه توافقی بر روی زمانبندی برای جلسات مربوط به فازهای اول و دوم و نیز تحویل گزارش نهایی صورت پذیرفت. نهایتا، ارائه‌های مربوط به معمار و مدیر پروژه که باید در مراحل اول و دوم فاز یک ارائه شوند مورد بررسی قرار گرفت و اطمینان حاصل شد که آنها دربرگیرنده اطلاعاتی خواهند بود که مورد نیاز تیم ارزیابی است.

قبل از فاز یک، تیم ارزیابی جلسه‌ای دو ساعته با یکدیگر داشتند. رهبر تیم دوباره نقشهای تخصیص داده شده را بررسی کرده و مطمئن شد که هر کس وظیفه خود را می‌داند. همچنین مروری بر مستندات معمای انجام شد و بخشهای مهم آن درک شده و در مورد الگوها و تاکتیک‌های اشاره شده در آن یاداشت برداریهایی صورت پذیرفت. این پیش جلسه تیم را در رسیدن به یک سری اطلاعات کلی در مورد معماری راهنمایی می‌کند و زمینه‌ای را برای مرحله چهارم فراهم می‌کند که در آن الگوها و روشها باید دسته‌بندی شوند.

در ارزیابی *Nightingle*، برگزاری جلسه منجر به تشخیص این موضوع شد که مستندات معماری ناقص و مبهم هستند. کلیه بخشها هنوز مستدسازی نشده و معماری بزرگ سیستم توسط یک مجموعه‌ی ناقص از نمودارهای جعبه–خط نمایش داده شده است. نتیجه بررسی این شد که اگر در این لحظه فاز اول آغاز شود تیم ارزیابی دید کافی در مورد معماری نخواهد داشت، بنابراین به صورت تلفنی با معمار ارتباط برقرار شد و از او خواسته شد که به صورت شفاهی بعضی ابهامات و کاستی را پوشش دهد. سپس با وجود اینکه می‌دانستیم کمبودهای زیادی در معماری وجود دارد ولی دست کم احساس راحتی در مورد ارزیابی وجود داشت (چون به بعضی کمبودها پاسخ داده شده بود). در آن لحظه کافی نبودن مستندات به عنوان یک خطر که باید فهرست شود، در نظر گرفته شد.

۱۱-۵-۲- فاز اول: ارزیابی

همان طور که مشخص شده بود در فاز اول تیم ارزیابی با تصمیم گیرندگان پروژه ملاقات می‌کند. در این جلسه علاوه بر اشخاصی که پیش‌بینی شده بود که در جلسه شرکت کنند دو طراح ارشد نیز در جلسه شرکت کرده بودند.

۱۱-۵-۲-۱- مرحله اول: نمایش *ATAM*

رهبر ارزیابی برای تشریح روش از بسته‌ی گراف‌های تصویری استاندارد سازمان تیم ارزیابی استفاده کرد. ارائه رهبر ارزیابی در حدود یک ساعت به طول انجامید که در آن فازها، مراحل و مفاهیم پایه *ATAM* تشریح شدند. همچنین فهرستی از خروجی‌هایی که از طریق به کارگیری *ATAM* تولید خواهند شد، نمایش داده شد. از آنجایی که تصمیم گیرندگان کاملا با *ATAM* آشنایی داشتند و توضیحات آنها در فاز صفر شنیده شده بود بدون هیچ مشکلی این مرحله به پایان رسید.

۱۱-۵-۲-۲- مرحله دوم: نمایش پیشرانهای حرفه

در ارزیابی، مدیر پروژه (مربوط به سازمان مشتری) اهداف حرفه برای سیستم *Nightingle* را از دید سازمان توسعه‌دهنده و همچنین سازمانهایی که آنها امیدوارند مشتری آنها شوند، ارائه می‌کند. برای سازمان توسعه‌دهنده، سیستم *Nightingle* نیازمندیهای حرفه زیر را برآورده می‌کند:

- حمایت از کاربردهای مختلف مشتریان (به عنوان مثال، ردگیری معاینات، تاریخچه پرداختی‌ها، روند تشخیص)
- ایجاد یک نسخه جدید از سیستم (به عنوان مثال مدیریت مطب دکتر) که سازمان توسعه‌دهنده بتواند آن را علاوه بر سیستم اصلی به مشتری بفروشد.

پیشران حرفه دوم نشان‌دهنده آن است که این معماری گرایش به این دارد که برای یک خط تولید نرم‌افزار بکار رود. برای شروع، *Nightingle* جایگزین سیستمهای موروثی می‌شود که دارای مشخصات زیر هستند:

- قدیمی بوده (یکی از سیستم‌ها دارای طول عمر ۲۵ سال بود)
- مبتنی بر زبانها و تکنولوژیهای قدیمی هستند (مانند *COBOL* و *IBM Assembler*)
- نسبت به نیازهای حرفه جاری سیستم و پروژه در بخشهای مختلف مراقب سلامت بی توجه هستند

نیازمندیهای حرفه مشتری اولیه عبارتند از:

- توانایی برخورد با انواع مختلف تفاوت‌های منطقه‌ای و فرهنگی
- توانایی برخورد با چند زبانی (مخصوصا انگلیسی و فرانسه) و ارز (مخصوصا دلار امریکا و پسو مکزیک)
- سیستم جدید باید دست کم دارای میزان کارایی باشد که سیستم موروثی دارد
- سیستم منفرد جدید باید برنامه‌های مدیریت مالی موروثی را ترکیب نماید

محدودیت‌های حرفه برای سیستم به شرح زیر هستند:

- تعهد به عدم از دست دادن شغل کارکنان از طریق آموزش مجدد کارکنان
- پذیرش رویکرد خرید به جای ساخت نرم‌افزار
- تشخیص اینکه مشتریان تجاری کم شده‌اند (بازار مشتریان کوچک شده است)

محدودیت‌های تکنیکی برای سیستم به شرح زیر هستند:

- استفاده از مولفه‌های آماده برای استفاده هر موقع که امکان آن وجود داشته باشد
- یک مدت زمان دو ساله برای اجرای سیستم با تعویض فیزیکی سخت‌افزار که در هر ۲۶ هفته اتفاق می‌افتد.

خصوصیات کیفی زیر به عنوان خصوصیات کیفی با اولویت بالا شناسایی شده‌اند:

- *کارایی:* سیستمهای مراقب سلامت نیاز به زمان پاسخ سریعی دارند تا اینکه بتوانند مفید واقع شوند (در سیستمهای موروثی پنج ثانیه زمان پاسخ تراکنش برای سئوالات و گزارشات بر روی خط خیلی کم بود). توان عملیاتی سیستم به عنوان یک دغدغه کارایی در نظر گرفته می‌شود.
- *قابلیت استفاده:* در سیستم مراقب سلامت کاربران زیادی وجود دارد که مدام در حال تغییر هستند، بنابراین آموزش، یک پیامد مهم مشتری است. سیستم جدید باید خیلی راحت قابل یادگیری و استفاده باشد.
- *قابلیت نگهداری:* سیستم مجبور است قابلیت نگهداری، قابلیت پیکربندی و قابلیت گسترش را برای حمایت از بازارهای جدید، نیازمندیهای جدید مشتریان، تغییرات در قوانین ایالت‌ها و نیازهای مختلفی منطقه‌ای و فرهنگی، داشته باشد.

همچنین مدیر، خصوصیات کیفی زیر را به عنوان خصوصیات مهم شناسایی کرده است ولی تا حدودی اولویت آنها نسبت به قبلی‌ها پائین است:

- *امنیت:* سیستم مجبور است که سطح عادی امنیت (به عنوان مثال، یکپارچگی و اطمینان داده) مورد نیاز برنامه‌های مالی را فراهم کند.
- *قابلیت دسترسی:* سیستم مجبور است که در ساعات عادی حرفه (کسب و کار) قابل دسترس باشد.
- *مقیاس‌پذیری:* سیستم مجبور است که در جهت برآورده کردن نیازهای مشتریان دارای بیمارستانهای بزرگ قابلیت افزایش مقیاس را داشته و همچنین برای برآورده کردن اهداف کلینیک‌های کوچک قابلیت کاهش مقیاس را داشته باشد.
- *پیمانه‌بندی:* سازمان توسعه‌دهنده تمایل دارد نه تنها نسخه‌های جدید از *Nightingle* را بفروشد بلکه خواهان فروش مولفه‌های آن به صورت منفرد نیز است. فراهم آوردن این توانایی نیازمند خصوصیات مرتبط به قابلیت نگهداری و مقیاس‌پذیری است.
- قابلیت آزمایش و قابلیت پشتیبانی[1]: سیستم مجبور است قابلیت درک به وسیله کارکنان فنی مشتری را داشته باشد زیرا آموزش کارمندان از جمله پیامدهای سیستم است.

۱۱-۵-۲-۳- مرحله سوم: نمایش معماری

در روند تعامل تیم ارزیابی با معمار (قبل از روند عملیات ارزیابی) چندین دید از معماری و روشهای معمارانه به وجود آمده‌اند. مهمترین درون‌بینی‌ها شامل موارد زیر هستند:

- *Nightingle* متشکل از دو زیرسیستم اصلی است. اولی مدیر تراکنش برخط *(OLTM)*[2] و دومی مدیر تولید گزارش و حمایت تصمیمات *(DSRGM)*[3] نامیده می‌شود. *OLTM* نیازمندیهای تعاملاتی کارایی را اجرا می‌کند در حالی که *DSRGM* فراتر از یک سیستم پردازش دسته‌ای است که وظایف آن به صورت دوره‌ای مشخص می‌شود.
- *Nightingle* برای قابلیت اطمینان بالا ساخته شده است.
- *OLTM* به شدت به صورت لایه‌بندی است.

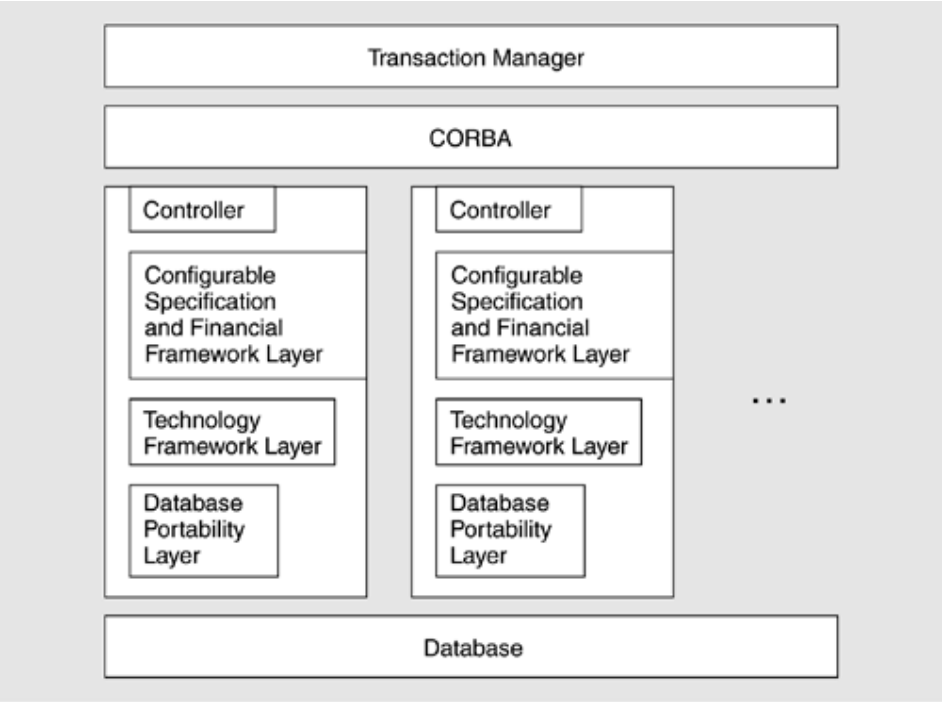
[1] Supportability
[2] OnLine Transaction Manager
[3] Decision Support and Report generation Manager (DSRGM)

- *Nightingle* یک سیستم مبتنی بر مخزن است. پایگاه‌داده‌های تجاری بزرگ در قلب آن قرار گرفته‌اند.
- *Nightingle* به شدت متکی بر نرم‌افزارهای آماده برای استفاده است. که عبارتند از: پایگاه‌داده‌های مرکزی، موتور قوانین[1]، موتور جریان کار[2]، *CORBA*، موتور وب[3]، ابزار توزیع نرم‌افزار[4] و غیره.
- *Nightingle* به شدت شیءگرا است و آن برای بدست آوردن قابلیت اطمینان متکی بر چارچوب‌های شیءگرایی است.

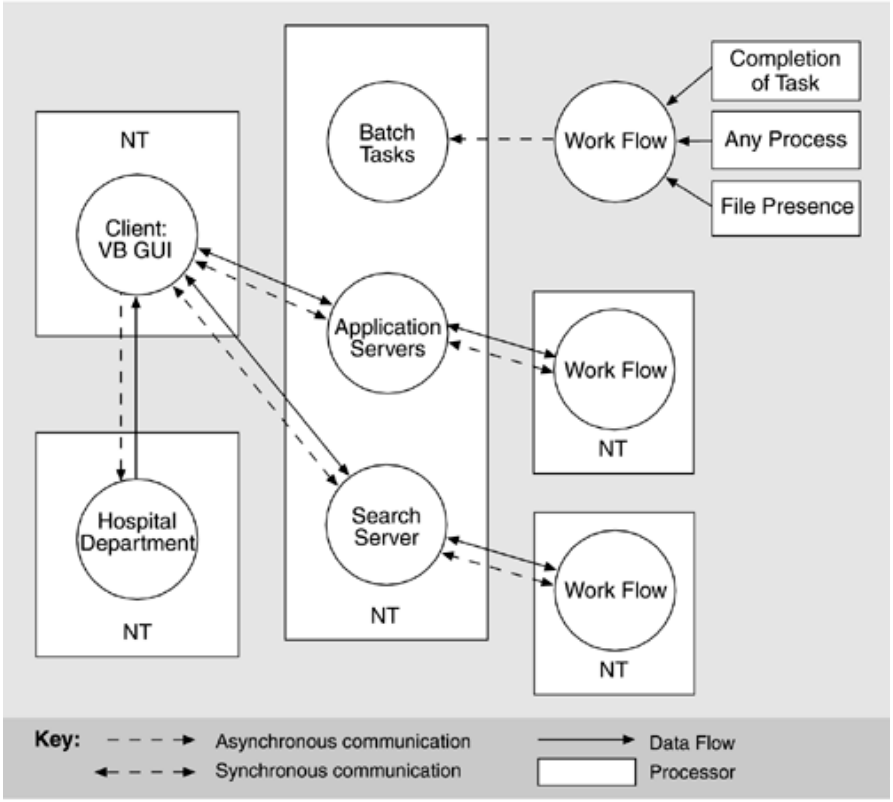
شکل ۱۱-۴ با نمادهای غیررسمی[5] استفاده شده به وسیله معمار، دید لایه‌بندی *OLTM* را نمایش می‌دهد. شکل ۱۱-۵ نشان می‌دهد که چگونه *OLTM* در زمان اجرا عمل می‌کند. در این شکل ارتباطات اصلی و مسیر جریان داده بین بخشهای سیستم استقرار یافته بر روی پردازشگرهای سخت‌افزاری مختلف، نشان داده شده است. این نمودارها به این دلیل ارائه شده‌اند تا بتوان دید بهتر از ماهیت واقعی ارزیابی *ATAM* را نشان داد. البته توجه داشته باشید که نگاشت واضح و کاملی ارائه نشده است. یعنی اینکه در شکل ۱۱-۴ مدیر تراکنش[6] و *CORBA* وجود دارند اما آنها در شکل ۱۱-۵ مطرح نشده‌اند. این نوع از قلم افتادگی‌ها[7] در بسیاری از ارزیابی‌های *ATAM* متداول است و یکی از فعالیت‌هایی که در مرحله سوم انجام می‌شود آن است که ارزیابان سئوالاتی در مورد ناسازگاری در نمودارها می‌پرسند تا اینکه به یک سطح دانش از سیستم دست پیدا کنند.

شکل ۱۱-۶ یک دید زمان اجرا مشابه از *OLTM* را نشان می‌دهد که در آن مشخص می‌شود آیا تراکنش‌ها می‌توانند در سرتاسر سیستم بدون تشریح معنادار پیکان‌ها ردیابی شوند. در این حالت نیز باز هم مشکل ناسازگاری وجود دارد. همچنین پیکانها نشان دهنده جریان داده نیز هستند.

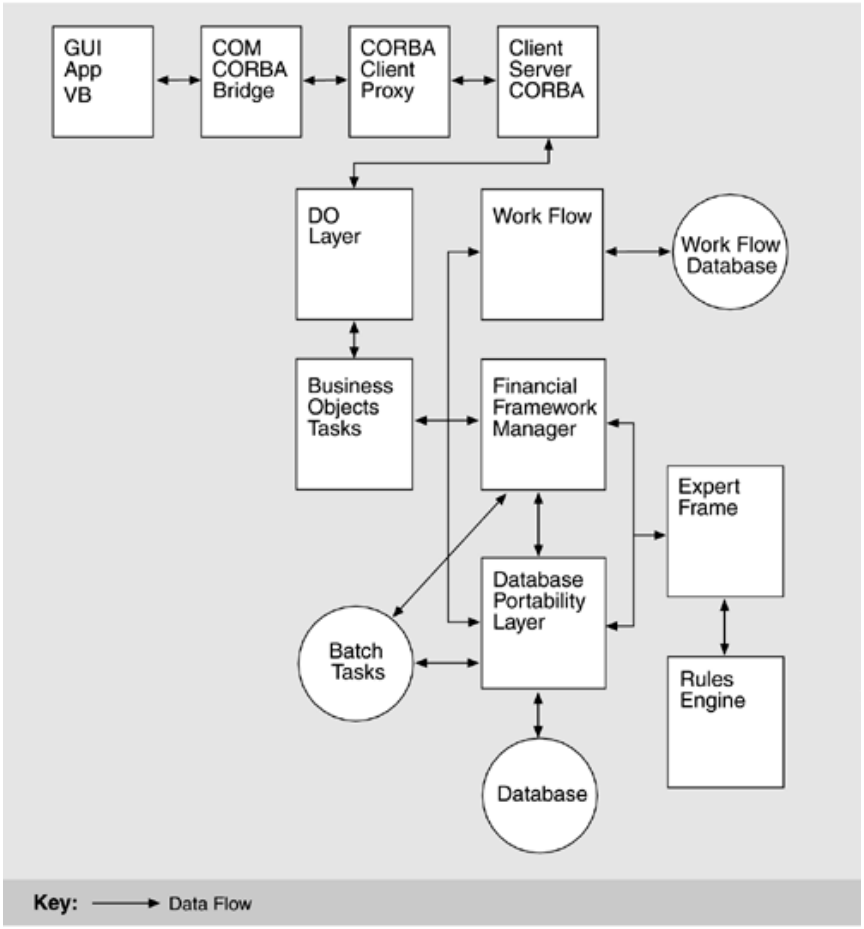
[1] Rule engine
[2] Work flow engine
[3] Web engine
[4] Software distribution tool
[5] Informal
[6] Transaction manager
[7] Omission



شکل ۱۱-۴: دید لایه‌بندی شده *OLTM* در نمادهای غیررسیمی معمار



شکل ۱۱-۵: دید نمایش دهنده ارتباطات، جریان داده‌ها و پردازشگرهای *OLTM*



شکل ۱۱-۶: دید معماری جریان داده *OLTM*

کلیه این دیدهای *Nightingle* صحیح و دارای اطلاعات مهمی هستند. هر کدام از دیدها یک جنبه مربوط به دغدغه‌های مختلف را نشان می‌دهد و کلا برای اجرا و تحلیل مراحل *ATAM* مورد استفاده قرار می‌گیرند.

۱۱-۵-۲-۴- مرحله چهارم: تعیین روشهای معمارانه

بعد از نمایش معماری، تیم ارزیابی روشهایی را که شنیده‌اند به علاوه مواردی که در روند دید پیش از ارزیابی از مستند معماری درک کرده‌اند را فهرست می‌کنند. موارد اصلی آنها عبارتند از:

- لایه‌بندی مخصوصا در *OLTM*
- شیءگرایی
- استفاده از فایل پیکرندی برای بدست آوردن قابلیت تغییرپذیری بدون ثبت یا کامپایل مجدد
- پردازش تراکنش سرویس‌گیرنده سرویس‌دهنده

- الگوی معماری متمرکز داده با پایگاه‌داده‌های تجاری در مرکز آن

این نکات و سایر روشها برای تیم ارزیابی یک پایه مفهومی را فراهم می‌کنند تا زمانی که تحلیل سناریوها آغاز می‌شود از آنها برای طرح سئوالات استفاده کند.

۱۱-۵-۲-۵- مرحله پنجم: ایجاد درخت سودمندی خصوصیات کیفی

جدول ۱۱-۵ درخت سودمندی ایجاد شده در روند اجرای *ATAM* را نشان می‌دهد. توجه شود که کلیه خصوصیات کیفی شناسایی شده در مرحله دوم در جدول نمایش داده شده است و هر کدام از آنها به یک یا چند معنی خاص پالایش شده‌اند. برای بعضی از خصوصیات کیفی پالایش شده سناریو خاصی وجود ندارد. این مورد اغلب اوقات اتفاق می‌افتد و هیچ مشکلی در ارتباط با آن وجود ندارد. اغلب اوقات افراد قابلیت پالایش خصوصیات کیفی، بدون استدلال در مورد حیطه آن را دارند اما زمانی که اقدام به ایجاد آن در زمینه سیستم خودشان می‌کنند در می‌یابند که واقعا قابل اعمال نیست.

برای ثبت درخت سودمندی به شکلی که قابل فهم برای همه باشد، نویسنده شرح مذاکرات از صفحه *flipchart* برای هر خصوصیت کیفی استفاده می‌کند و آن را به دیوار می‌چسباند. سپس بعد از آنکه خصوصیات کیفی با استفاده از سناریوها پالایش شدند آن اطلاعات را بر روی *flipchart* ثبت می‌کند. در واقع اطلاعات جدید را به ادامه *flipchart* اضافه می‌کند.

سناریوها در جدول ۱۱-۵ همراه با اولویت‌بندی هستند که به وسیله تصمیم گیرندگان تعیین شده است. اولین عنصر از هر جفت مرتب شده اشاره به اهمیت قابلیت (یا توانایی) دارد. دومین عنصر نیز اشاره به تخمین معمار از سختی دستیابی به آن دارد.

در نظر داشته باشید که بعضی از سناریوها مطابق با بحث‌های اولیه خوش فرم هستند در حالی که بعضی‌ها پاسخ یا محرک ندارند. در این مرحله، عدم اطمینان در مشخصات سناریوها تا زمانی که ذینفعان معنی آن را درک کنند، مجاز است. همچنین اگر سناریوها برای تحلیل انتخاب شدند در آن صورت حتما باید به صورت صریح پاسخ و محرک برای سناریو مشخص شود.

جدول ۱۱-۵: فرم جدولی درخت سودمندی برای سیستم *Nightingle*

خصوصیت کیفی	پالایش خصوصیت	سناریوها
کارایی	زمان پاسخ تراکنش	کابر در پاسخ به ابلاغ تغییر آدرس بیمار در شرایط حداکثر بارکاری حساب بیمار را به روز می‌کند و تراکنش در کمتر از ۰.۷۵ ثانیه پایان می‌یابد *(H, M)*. کاربر در پاسخ به ابلاغ تغییر آدرس بیمار در شرایط حداکثر بار کاری دو برابر، حساب بیمار را به روز می‌کند و تراکنش در کمتر از ۴ ثانیه پایان می‌یابد *(L, M)*.
	توان عملیاتی	در حداکثر بار کاری، سیستم می‌تواند در هر ثانیه ۱۵۰ تراکنش عادی را کامل کند *(M,M)*
	تائید گزارش	هیچ سناریو پیشنهاد نشده است
قابلیت استفاده	آموزش متخصص‌ها	کارمندان جدید با دو یا چند سال سابقه در تجارت در انجام کارهای اصلی شرکت طی مدت کمتر از یک هفته مهارت می‌یابند *(M,L)*. کاربر در یک زمینه خاص نیاز به راهنمایی دارد و سیستم این راهنمایی را مهیا می‌کند *(H, L)*.
	عملیات عادی	مسئول پرداخت در بیمارستان یک برنامه پرداخت را بدون هیچگونه تاخیر برای بیماران در طول مدت تعامل آنها و سپس پایان فرآیند عملیات آغاز می‌نماید
قابلیت پیکربندی		بیمارستان در ازاء سرویس‌های ویژه، قیمت را افزایش می‌دهد. تیم پیکربندی تغییراتی را به ازاء یک روز کاری ایجاد می‌کنند. برای تغییرات نیاز به کد اصلی نیست *(H, L)*.
قابلیت نگهداری		نگهدارنده با کاهش زمان جستجو و پاسخ، حل مشکلات و توزیع مشکلات حل شده رو در رو است *(H,M)* نیازمندی گزارش، نیاز به تغییراتی در فوق داده تولید کننده گزارش دارد *(M, L)*. فروشنده پایگاه‌داده یک نسخه جدید که باید در حداقل زمان نصب شود را منتشر می‌کند *(H, M)* .
قابلیت گسترش	اضافه کردن محصولات جدید	یک محصول که اهدا کنندگان خون را ردگیری کند، ایجاد شود.
امنیت	قابلیت اطمینان	یک دکتر فقط اجازه دارد تا به اطلاعات مربوط به بیمار خود دسترسی داشته باشد و نباید قادر باشد اطلاعات مالی بیمار خود را

		مشاهده کند *(H, M)*
	یکپارچگی	سیستم در مقابل دسترسی‌های غیر مجاز ایستادگی کند.
قابلیت دسترسی		فروشنده پایگاه‌داده یک نسخه جدید منتشر کند که به سرعت جایگزین می‌شود *(H, L)* . سیستم از حساب ۲۴/۷ تحت وب که توسط بیمار قابل دسترسی است حمایت می‌کند *(L, L)*.
مقیاس‌پذیری	رشد سیستم	مشتری یک شرکت مراقب سلامتی خریده است که سه برابر اندازه خودش است و تنها به قسمتی از پایگاه داده نیاز دارد *(L, H)*. مشتری جدید از داشتن واحد تجاری بی بهره است *(L, M)*. مشتری جدید دو واحد تجاری را یکی کرده است *(L, M)*. سازمان توسعه‌دهنده خواهان فروش اجزاء *Nightingle* است *(M, L)*.
پیمانه‌بندی	زیرمجموعه‌های عملکردی	ساخت سیستم خودگردان[1] با عملکردی اصلی *(M, L)*.
	قابلیت انعطاف برای جایگزین نمودن محصولات آماده برای استفاده	جایگزین نمودن پایگاه‌داده‌های تجاری فروشنده یکی پس از دیگری *(H, M)*. جایگزین نمودن سیستم عامل *(H, M)*. جایگزین نمودن لایه قابل حمل *(H, M)*. جایگزین نمودن مدیر تراکنش *(H, M)*. جایگزین نمودن موتور جریان کار *(H, M)*. جایگزین نمودن بسته‌های پیش‌نویس تجاری حسابداری *(H, M)*. جایگزین نمودن *solaris* با سکوهای *sun* که پایگاه‌داده بر روی آنها است *(H,M)*. حایگزین نمودن موتور قوانین *(H, M)*.
تعامل‌پذیری		ایجاد سیستمی که با پایگاه‌داده امراض مسری در مرکز بین المللی کنترل بیماری در ارتباط است *(M,M)*.
قابلیت آزمایش		
قابلیت پشتیبانی		

[1] Autonomously

۱۱-۵-۲-۶- مرحله ششم: تحلیل روشهای معمارانه

درخت سودمندی هیچ سناریویی با رتبه *(H,H)* را تولید نکرد. این نشان‌دهنده آن است که هیچ سناریوی با اهمیت بالا و سختی زیاد برای دستیابی، وجود ندارد که ارزش تحلیل اولویت بالا را داشته باشد. بنابراین باید به دنبال سناریوهایی با رتبه *(H, M)* بود. در این بررسی سناریوی جایگزاری مولفه‌های آماده برای استفاده مختلف در سیستم که جزو خصوصیت کیفی پیمانه‌بندی است، کشف شد. اگر چه استفاده از مولفه‌های آماده برای استفاده راهبرد هدفمند برای کاهش خطرات توسعه است اما آن نگرانیهایی را برای معمار ایجاد می‌کند زیرا معمار احساس می‌کند که سیستم به تعداد زیادی از مولفه‌های آماده برای استفاده مربوط به فروشندگان مختلف وابسته است. بنابراین دستیابی به انعطاف‌پذیری معمارانه برای خارج شدن از وابستگی محصولات آماده برای استفاده، بسیار حساس خواهد بود.

در این مرحله تیم ارزیابی هر یک از سناریوها را با معمار بررسی می‌کند. هر بررسی به طور متوسط زمانی در حدود نیم ساعت را به خود اختصاص می‌دهد. از آنجایی که سناریوهایی در مورد تغییرات وجود داشت تیم ارزیابی سئوالاتی در مورد دامنه و تاثیرات تغییرات مطرح کرد که منجر به درک موضوعات زیر شد:

- جایگزین نمودن پایگاه‌داده‌های تجاری با پایگاه‌داده‌هایی که توسط فروشندگان دیگر حمایت می‌شود فرآیند دشواری خواهد بود. یک نسخه از *SQL* در *Nightingle* استفاده می‌شود که مخصوص یک فروشنده پایگاه‌داده خاص است. همچنین در آن از ابزار و مولفه‌های فروشندگان دیگر نیز استفاده شده است. بنابراین اولین خطر معمارانه مبتنی بر تحلیل ثبت می‌شود. این خطر بدین صورت بیان می‌شو."بدلیل اینکه *Nightengile* از ابزار، مولفه‌ها و زبان *SQL* خاص که متعلق به یک سری شرکت‌های خاص هستند و با پایگاه‌داده‌های دیگر نیز سازگار نیستند، استفاده می‌کند جایگزینی پایگاه‌داده بسیار هزینه بر و مشکل خواهد بود و نیازمند چندین سال تلاش است". تصمیم معمارانه مبتنی بر تخصیص پایگاه‌داده به معماری نیز به عنوان یک نقطه حساس ثبت شد که به صورت منفی قابلیت تغییر را تحت تاثیر قرار می‌دهد.
- جایگزین کردن سیستم عامل باسیستمهای دیگر منطقا کار آسانی است. در سمت سرویس‌دهنده، سیستم عامل با یک لایه پوشیده شده که فقط تغییرات ضروری و آن هم در یک محدوده کوچک را ممکن می‌سازد. اما *OLTM* مستقیما به امکانات *NT* تکیه دارد و جایگزین کردن سیستم عامل مشابه

با ایجاد یک تغییر ساده خواهد بود. در *DSRGM* تمام وابستگی‌های سیستم عامل به کد از بین رفته است. *DSRGM* در سکوی ویندور *NT* توسعه یافته است اما بر روی *UNIX* مستقر شده است که خود این مدرک روشنی است که آن از قبل مستقل از سیستم عامل بوده است. بنابراین در این مرحله اولین غیرخطر ثبت می شود. "به دلیل اینکه وابستگی سیستم عامل محلی شده یا از *OLTM* یا *DSRGM* حذف شده است. جایگزینی سیستم عامل با سیستم عامل دیگر فقط نیازمند یک سری تغییرات کوچک است". محصورسازی وابستگیهای سیستم عامل به عنوان یک نقطه حساس ثبت گردید که به صورت مثبت قابلیت تغییرپذیری را تحت تاثیر قرار می‌دهد.

- تغییر موتور قوانین باعث به وجود آمدن چندین دغدغه خواهد شد. وقوع این سناریو غیر ممکن نیست زیرا درک شده است که دغدغه‌های کارایی و قابلیت نگهداری در ارتباط با استفاده از موتور قوانین وجود دارد. احتمالا سناریویی خواهد بود که موتور قوانین را حذف می‌کند و سپس قوانین را مستقیما در *C++* پیاده‌سازی می‌کند. از آنجا که زنجیره پیشرو[1] در میان قوانین غیر مجاز شده است قوانین به صورت رویه‌ای خواهد بود و باید کامپایل شود. یک چنین تغییراتی چندین تاثیر جدی به شرح زیر دارند:

– احتمالا کارایی را بهبود می‌بخشد

– نیاز کارکنان به آموزش زبان قوانین و درک آنها در ارتباط با موتور قوانین حذف می‌شود.

– تیم توسعه را از توسعه قوانین کارا و شبیه‌سازی محیط بی نیاز می‌کند.

– این موضوع می‌تواند باعث شود که قواعد در بقیه کد *C++* استفاده نشده و درک این نکته را برای آنها ساده‌تر کند که از کد تابعی که الزاما به قواعد ربطی ندارند استفاده کنند. لذا تشخیص و نگهداری آن مشخص خواهد بود.

– این قابلیت را که قواعد به برخی اشیاء که در حقیقت وجود ندارند ارجاء شوند، از بین می‌برد. امکاناتی که امروزه وجود دارد و باعث می‌شود خطا بتواند بعد از آزمایش باقی بماند و وارد سیستم تولید شده شود. نوشتن قواعد در *C++* این گونه خطاها را در زمان کامپایل از بین می‌برد.

[1] Forward chaining

برای تسهیل این تغییرات، یک تولید کننده کد قوانین به $C++$ باید نوشته شود و یک تلاش گسترده نیز برای فضاهای مهم و ناشناخته شده باید صورت بپذیرد. برای این سناریو، تلاشهای مهم مورد نیاز برای کنار گذاشتن قواعد ماشین به عنوان خطر در نظر گرفته می‌شود. همچنین استفاده از قواعد ماشین (در مقابل کد $C++$) به عنوان نقطه مصالحه در معماری در نظر گرفته می‌شود. این سودمندی در ازای کاهش کارایی و دشوار کردن قابلیت آزمایش بدست می‌آید. اگر بررسی این سناریوها ادامه پیدا کنند نقاط دیگری از قبیل بسته حسابداری تجاری، موتور جریان کار و الی آخر نیز دارای نقاط حساس، نقاطه مصالحه، خطر و غیر خطر خواهند بود. در این قسمت مرحله یک به پایان می‌رسد و در کل برای این سیستم شش نقطه حساس، یک نقطه مصالحه، چهار خطر و پنج غیرخطر شناسایی می‌شود.

۱۱-۵-۳- فاز دوم: ارزیابی (ادامه)

جلسات مربوط به فاز دوم بعد از دو هفته وقفه شروع می‌شود. در روند این وقفه زمانی، تیم ارزیابی بخشهایی از گزارش نهایی را می‌نویسند که نتوانسته‌اند کامل کنند (از قبیل: پیشرانهای حرفه، نمایش معماری، فهرست روش‌ها، درخت سودمندی و تحلیل فاز اول).

برای فاز دوم، علاوه بر تصمیم گیرندگان پروژه، نه ذینفع دیگر نیز مشخص شده‌اند. این افراد شامل توسعه‌دهندگان، نگهداران، نمایندهگان مشتری و دو کاربر هستند. نخستین فعالیت از فاز دوم تکرار مرحله اول برای شرکت‌کنندگان جدید بوده و سپس نتایج فاز اولیه برای همگان آشکار می‌شود. بعد از آن نیز مراحل هفت، هشت و نه آغاز می‌شود.

۱۱-۵-۳-۱- مرحله هفتم: طوفان ذهنی و اولویت‌بندی سناریوها

ذینفعان، گروه فعالی بودند به طوری در کل ۷۲ سناریو در این فاز معرفی کردند. بسیاری از این سناریوها از برگ‌های درخت سودمندی حاصل شده‌اند که در فاز اول تحلیل نشده‌اند. جدول ۱۱-۶ شامل خلاصه‌ای از انواع سناریوهای به وجود آمده در مرحله هفت است. باید توجه داشت که بسیاری از آنها خیلی ساختار یافته نیستند و بعضی از آنها ماهیت پیچیده‌ای دارند. این موضوع ماهیت خود برانگیزندگی[1] روش طوفان ذهنی را نشان می‌دهد.

[1] Spontaneous

جدول ۱۱-۶: سناریوهای طوفان ذهنی

شماره	سناریو
۱	قبلا داده‌های عمومی، اختصاصی می‌شدند و دسترسی مبتنی بر آن تنظیم می‌شد.
۲	داده موجود در مرکز اطلاعات در شعبه کلینیک‌ها تکرار شده و باعث کاهش کارایی می‌شود
۳	تصمیم برای پشتیبانی از زبان آلمانی
۴	...

بعد از ترکیب سناریوهایی که مشابه هستند ذینفعان باید رای خود را در مورد آنها اعلام کنند. به هر نفر ۲۲ رای اختصاص داده می‌شود (۳۰٪ از ۷۲ سناریو که گرد شده است). بعد از محاسبه آراء، در مورد سناریوهای با اولویت بالا که در درخت سودمندی مشخص شده‌اند، بحث می‌شود. برای این مطالعه موردی، کلیه سناریوهای اولویت بالا به عنوان برگ‌هایی جدید شاخه موجود در درخت سودمندی قرار داده می‌شوند. بعد از انطباق سناریوی جدید با درخت سودمندی، تحلیل سناریوهایی که بیشترین رای را به خود اختصاص داده‌اند آغاز می‌شود.

۱۱-۵-۳-۲- مرحله هشتم: تحلیل روشهای معمارانه

در مرحله هشت، سناریوهای کشف شده مورد تحلیل قرار می‌گیرد. در روند این تحلیل به هفت سناریوی اضافی نیز دست پیدا شد و آنها نیز مورد تحلیل قرار گرفتند. بنابراین کاملا واضح است که تعداد سناریوهای بررسی شده کمی بالاتر از میانگین *ATAM* است.

۱۱-۵-۳-۳- مرحله نهم: نمایش نتایج

مرحله نهم یک تا دو ساعت ارائه خلاصه‌ای از نتایج و یافته‌های عملیات *ATAM* است. این جلسه با یکسری اسلایدها که دربرگیرنده توصیف روش، پیشرانهای حرفه، خلاصه معماری، فهرست روشها، درخت سودمندی، تحلیل نیازمندیها و فهرست خروجی تحلیل است، آغاز می‌شود. در پایان فاز دوم تیم ارزیابی جلسه‌ای را تشکیل می‌دهند و به بررسی کلیه نتایج بدست آمده می‌پردازند. همچنین مرحله نه شامل یک بازه زمانی است که اجازه می‌دهد تیم ارزیابی بتواند بسته مستندات خود را کامل کند.

تیم ارزیابی علاوه بر خطرات، غیرخطرات، نقاط حساس، نقاط مصالحه، عنوان خطراتی که به نظر می‌رسد به صورت سیستماتیک حوزه‌های مشکل‌ساز معماری را مشخص می‌کنند، را نیز نمایش می‌دهد. برای *Nightingle* سه عنوان خطر شناسایی شده است که عبارتند از:

۱. *اعتماد بیش از حد به محصولات آماده برای استفاده خاص*
۲. *فرآیندهای بازیابی خطا کاملا تعریف نشده‌اند.*
۳. *پیامدهای مرتبط با مستندسازی*

۱۱-۵-۴- فاز سوم: پیگیری

خروجی‌های قابل لمس *ATAM* گزارش نهایی است که دربرگیرنده فهرست خطرات، نقاط حساس، نقاط مصالحه است. همچنین شامل فهرستی از روشهای معمارانه استفاده شده، درخت سودمندی، سناریوهای طوفان ذهنی و تحلیل هر سناریوی انتخاب شده است. نهایتا گزارش نهایی شامل مجموعه‌ای از عناوین خطر شناسایی شده از طریق تیم ارزیابی است که مشخص می‌کند کدام پیشرانهای حرفه تحت تاثیر آنها قرار می‌گیرد. مشابه ارائه نتایج، گزارش نهایی نیز از یک قالب استاندارد برای نمایش بخشهای خود استفاده می‌کند.

فهرست مطالب

فصل دوازدهم

روش کمی جهت تصمیم‌گیری در مورد طراحی معماری

در این فصل یک روش کمی برای تصمیم‌گیری در مورد طراحی معماری به نام روش تحلیل هزینه فایده تشریح خواهد شد. بدین منظور ابتدا ساختار پایه روش تشرح خواهد شد سپس بیان می‌شود که چگونه این ساختار به یک سری مراحل عملی تبدیل می‌شود. نهایتا یک مطالعه موردی مطرح خواهد شد که نحوه اعمال روش تحلیل هزینه فایده را بر روی یک سیستم دنیای واقعی نشان می‌دهد.

۱۲-۱- مقدمه

همان طور که در فصل یازده عنوان شد، *ATAM* برای معماران نرم‌افزار روشی را برای ارزیابی مصالحه‌های فنی رو در روی طراحی و نگهداری سیستمهای نرم‌افزاری فراهم می‌آورد. در *ATAM* اساسا به بررسی این موضوع پرداخته می‌شود که به چه کیفیتی معماری با توجه به خصوصیات کیفی که برای ذینفعان آن مهم هستند، طراحی شده است.

با این وجود، *ATAM* یک ملاحظه مهم را نادیده گرفته است و آن این است که بزرگ‌ترین مصالحه‌ها در سیستمهای بزرگ و پیچیده معمولا در ارتباط با مسائل اقتصادی هستند. یعنی اینکه چگونه سازمان باید در یک روشی خاص منابع خود را به کار ببرد که کمترین خطر و بیشترین بهره را از آنها بدست آورد. در گذشته این سئوال بر روی هزینه متمرکز بود مخصوصا هزینه‌هایی که مربوط به ساخت اولیه سیستم می‌شدند و هزینه‌های موجود در سرتاسر چرخه نگهداری و به روز رسانی در نظر گرفته نمی‌شدند. با این حال مهمتر از هزینه، سودی است که یک تصمیم معماری می‌تواند برای یک سازمان به ارمغان بیاورد.

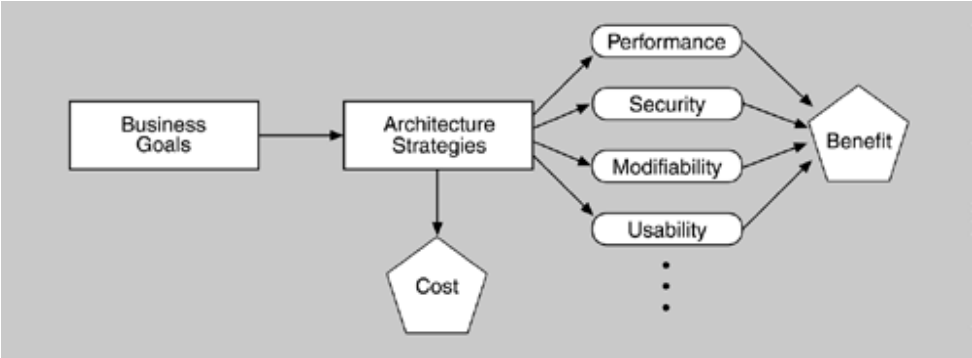
در نظر بگیرید که منابع برای ساخت و نگهداری یک سیستم محدود هستند، بنابراین باید فرآیند منطقی وجود داشته باشد که به انتخاب گزینه‌های معماری در طول فاز آغازین طراحی و دوره به روز رسانی بعدی آن کمک کند. این گزینه‌ها هزینه‌های متفاوتی خواهند داشت، مقدار منابع متفاوتی را مصرف خواهند کرد، ویژگیهای متفاوتی را پیاده‌سازی خواهند کرد و بعضی عدم قطعیت‌ها و خطرات ذاتی را خواهد داشت. برای بدست آوردن این جنبه‌ها به یک مدل اقتصادی از نرم‌افزار نیاز است که هزینه‌ها، فایده‌ها، خطرات و الزامات[1] زمانبندی را در نظر بگیرد.

برای برآورده کردن نیاز به اتخاذ تصمیمات اقتصادی، یک روش مدلسازی اقتصادی سیستمهای نرم‌افزاری به نام "روش تحلیل هزینه فایده"[2] توسعه داده شده است. این روش مبتنی بر *ATAM*، برای مدلسازی هزینه و فایده تصمیمات طراحی معماری ایجاد شده و یک روش بهینه‌سازی چنین تصمیماتی نیز محسوب می‌شود. همچنین *CBAM* یک ارزیابی از پیامدهای فنی و اقتصادی و نیز تصمیمات معمارانه فراهم می‌آورد.

[1] Implications
[2] Cost Benefit Analysis Metod (CBAM)

۱۲-۲- زمینه تصمیم‌گیری

معمار یا تصمیم گیرنده نرم‌افزار خواهان آن است که تفاوت فاحشی بین سود بدست آمده از سیستم و هزینه حاصل از پیاده‌سازی طراحی سیستم ایجاد کنند. *CBAM* از محلی شروع می‌شود که *ATAM* به پایان رسیده است در حقیقت، آن وابسته به فراورده‌های خروجی *ATAM* است. شکل ۱۲-۱ زمینه *CBAM* را نشان می‌دهد.



شکل ۱۲-۱: زمینه برای *CBAM*

اهداف حرفه یک سیستم نرم‌افزاری باید راهبردهای استفاده شده به وسیله معماران و طراحان را تحت تاثیر قرار دهد زیرا راهبردهای معماری دلالت‌های اقتصادی و فنی دارند. الزامات اقتصادی مستقیم، هزینه پیاده‌سازی سیستم است. دلالت‌های فنی نیز خصوصیات سیستم یا همان خصوصیات کیفی هستند. همچنین به دلیل فواید حاصل از خصوصیات کیفی، در خصوصیات کیفی دلالت‌های اقتصادی نیز وجود دارند.

زمانی که *ATAM* به یک سیستم نرم‌افزاری اعمال می‌شود نتایج به صورت یک مجموعه از فراورده‌های مستندسازی شده تولید می‌شود که عبارتند از:

- توصیفی از اهداف حرفه که برای موفقیت سیستم حیاتی هستند.
- مجموعه‌ای از دیدهای معماری که معماری موجود و پیشنهادی را مستند می‌کند.
- درخت سودمندی که تجزیه اهداف ذینفعان برای معماری را نمایش می‌دهد که با توصیف سطح بالای خصوصیات کیفی آغاز شده و با سناریوهای خاص متعلق به آنها خاتمه می‌یابد.
- مجموعه‌ای از خطرات شناسایی شده

- مجموعه‌ای از نقاط حساس
- مجموعه‌ای از نقاط مصالحه

ATAM مجموعه‌ای از تصمیمات معماری مرتبط با سناریوهای خصوصیات کیفی استخراج شده برای ذینفعان را شناسایی می‌کند. این تصمیمات منجر به تعدادی واکنشهای خاص خصوصیات کیفی می‌شوند (یعنی سطح خاصی از قابلیت دسترسی، کارایی، امنیت، قابلیت استفاده، قابلیت تغییرپذیری و غیره). اما هر تصمیم معماری هزینه خاص خود را دارد. برای مثال، استفاده از سخت‌افزار اضافی برای دستیابی به سطح خاصی از قابلیت دسترسی، هزینه خاص خودش را دارد و همچنین نقطه وارسی برای فایلهای روی دیسک نیز هزینه متفاوت و خاص خود را دارد. به علاوه، هر دوی این تصمیمات معماری منجر به سطح قابل اندازه‌گیری از قابلیت دسترسی می‌شود که میزانی از ارزش ارزشی برای سازمان توسعه دهنده سیستم خواهد داشت. احتمالا دلیل استفاده از تصمیم اول آن است که سازمان توسعه‌دهنده متعقد است ذینفعان آن توجه زیادی به قابلیت دسترسی بالا خواهند داشت (مانند سوئیچ‌های تلفن و یا نرم‌افزارهای نظارت کننده پزشکی) و تصمیم دوم نیز برای زمانهایی استفاده می‌شود که سیستم دچار شکست شده است (مانند نرم‌افزار کنترل ترمزهای ضد قفل در خودرو).

ATAM تصمیمات معماری اتخاذ شده در سیستم را مشخص کرده و آنها را به اهداف تجاری و معیارهای پاسخ خصوصیات کیفی مرتبط می‌کنند. *CBAM* مبتنی بر این اصل و از طریق استخراج هزینه و فایده مرتبط با این تصمیمات ایجاد شده است. بنابراین با توجه به این اطلاعات، ذینفعان می‌توانند تصمیم بگیرند که آیا از سخت‌افزار افزونه، نقطه وارسی، یا دیگر تاکتیک‌ها برای دستیابی به قابلیت دسترسی مطلوب سیستم استفاده کنند، یا اینکه می‌توانند این تصمیم را بگیرند که منابع محدود خود را بر روی دیگر خصوصیات کیفی سرمایه‌گذاری کنند (مثلا، کارایی بالا یک نسبت سود به هزینه بهتری را نسبت به قابلیت دسترسی به ارمغان می‌آورد). هر سیستم همیشه یک بودجه محدود برای تولید و به روز رسانی خود دارد لذا در نهایت هر تصمیم معماری در حال رقابت با تصمیمات دیگر در جهت قرار گرفتن در ساختار معماری است (البته این رقابت نیز در بعضی حالات رخ می‌دهد).

همانند یک مشاور مالی که هرگز نمی‌گوید چگونه پول خود را سرمایه گذاری کنید، *CBAM* نیز برای ذینفعان تصمیم‌گیری نمی‌کند. بلکه به سادگی به آنها در استخراج و مستندسازی هزینه‌ها، فایده‌ها و عدم قطعیت‌ها در

مورد سهام سرمایه‌گذاری معمارانه کمک می‌کند و برای آنها یک چارچوب فراهم می‌کند که در داخل آن مشخص شده است که چگونه می‌تواند یک فرآیند تصمیم‌گیری منطقی به کار ببرند که برای نیازها و خطرات ناسازگار آنها مناسب باشد.

ایده پشت *CBAM* آن است که راهبردهای معماری، خصوصیات کیفی سیستم را تحت تاثیر قرار می‌دهند و اینها نیز به نوبه خود برای ذینفعان فایده‌ای به همراه می‌آورند. به این فایده‌ها[1] سودمندی[2] نیز گفته می‌شود. هر راهبرد معماری یک سطح خاصی از سودمندی را برای ذینفعان به ارمغان می‌آورد. همچنین زمان و هزینه خاص خود را برای پیاده‌سازی به همراه دارد. با توجه به این اطلاعات، *CBAM* می‌تواند به ذینفعان در انتخاب راهبردهای معمارانه خود مبتنی بر بازگشت سرمایه[3] (نسبت هزینه به سرمایه) کمک کند.

۱۲-۳- اساس روش تحلیل هزینه فایده

در این بخش ایده‌های کلیدیی تشریح خواهد شد که پایه‌های *CBAM* را تشکیل می‌دهند. تحقق عملی[4] این ایده‌ها به عنوان یک سری از مراحل، در بخش بعدی تشریح خواهد شد. در واقع هدف از این بخش توسعه تئوری پشتیبان معیار بازگشت سرمایه برای راهبردهای مختلف معماری با استفاده از سناریوهای انتخاب شده به وسیله ذینفعان است.

نقطه شروع *CBAM* مجموعه‌ای از سناریوها است. سناریوها می‌توانند خروجی *ATAM* بوده یا طبق فرآیندی خاص فقط برای ارزیابی *CBAM* فراهم شوند سپس بررسی می‌شود که چقدر این سناریوها با ارزش پاسخهای طرح‌ریزی شده[5] تفاوت دارند و به هر کدام از آنها یک سودمندی تخصیص داده می‌شود. سودمندی مبتنی بر اهمیت هر سناریو و ارزش پاسخ پیش‌بینی شده برای آن است. در این مرحله، راهبردهای معماری که منجر به انواع مختلف پاسخهای طرح‌ریزی شده می‌شوند، مورد ملاحظه قرار می‌گیرند. هر راهبرد یک هزینه دارد و هر کدام چندین خصوصیت کیفی را تحت تاثیر قرار می‌دهد. یعنی اینکه، یک راهبرد می‌تواند برای دستیابی به چندین پاسخ طرح‌ریزی شده پیاده‌سازی شود اما در حالی که آن راهبرد منجر به دستیابی به پاسخ می‌شود،

[1] Benefits
[2] Utility
[3] Return of Investment (ROI)
[4] Practical realization
[5] Projected responses

تعدادی از خصوصیات کیفی دیگر را نیز تحت تاثیر قرار می‌دهد. سودمندی این "تاثیرات جانبی"[1] هنگامی که سودمندی کل راهبرد بررسی می‌شود باید مورد توجه قرار بگیرد. این سودمندی همان است که با هزینه پروژه یک راهبرد معماری ترکیب می‌شود تا میزان بازگشت سرمایه نهایی محاسبه شود.

۱۲-۳-۱- سودمندی

سودمندی با ملاحظه قرار دادن موضوعات زیر تعیین می‌شود:

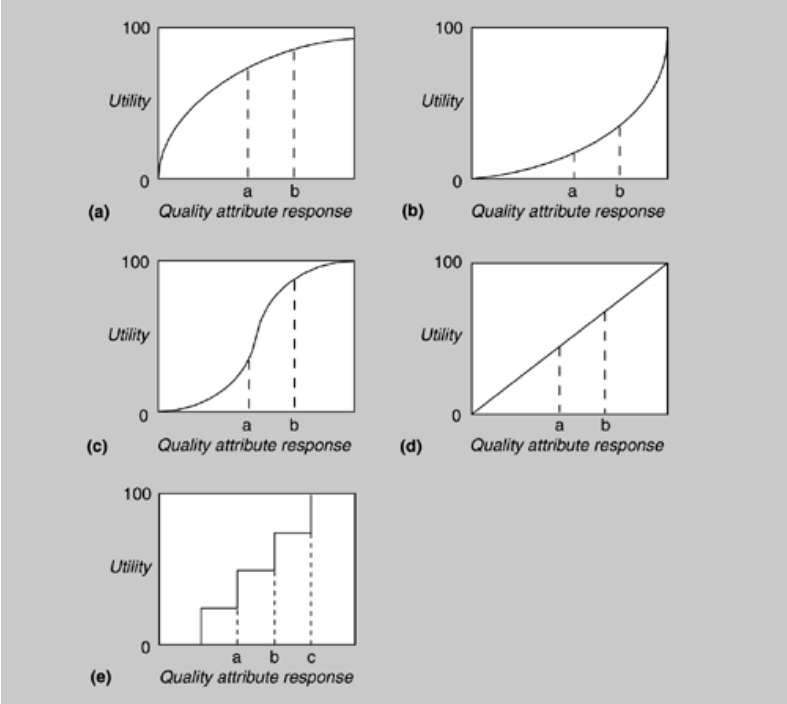
- *گوناگونی سناریوها*[2]

CBAM از سناریوها به عنوان روشی برای بیان و نمایش منسجم خصوصیات کیفی خاص استفاده می‌کند (مانند *ATAM*). در *ATAM* سناریوها در سه بخش محرک، محیط و پاسخ سازماندهی می‌شدند اما بین *ATAM* و *CBAM* تفاوتی وجود دارد. *CBAM* اساسا از مجموعه‌ای از سناریوها (به وسیله تغییر مقادیر پاسخها تولید می‌شوند) به جای سناریوهای منفرد استفاده می‌کند که این امر منجر به مفهوم منحنی سودمندی-پاسخ می‌شود.

- *منحنی‌های سودمندی-پاسخ*[3]

هر جفت مقدار پاسخ-محرک در یک سناریو، تعدادی سودمندی برای ذینفعان فراهم می‌کند و سودمندی مقادیر مختلف ممکن برای پاسخ می‌تواند خیلی متفاوت باشد. برای مثال، قابلیت دسترسی خیلی بالا در پاسخ به شکست می‌تواند از طریق ذینفعان فقط کمی بیشتر از حالت متوسط قابلیت دسترسی ارزش‌گذاری شود یا "تاخیر کم" می‌تواند خیلی بیشتر از "تاخیر متوسط" ارزش گذاری شود. هر رابطه بین مجموعه‌ای از معیارهای سودمندی و مجموعه مرتبط با معیارهای پاسخ را می‌توان به صورت یک نمودار (منحنی سودمندی-پاسخ) ترسیم کرد. بعضی مثالها از منحنی سودمندی-پاسخ در شکل ۱۲-۲ نشان داده شده است. در شکل ۱۲-۲ هر کدام از برچسب‌های *a, b, c* مقادیر پاسخ متفاوتی را نشان می‌دهند. بنابراین منحنی سودمندی-پاسخ، سودمندی را به صورت تابعی از مقادیر پاسخ نشان می‌دهد.

[1] Side effect
[2] Variations of scenarios
[3] Utility-response curves



شکل ۱۲-۲: تعدای نمونه از منحنی سودمندی-پاسخ

منحنی سودمندی-پاسخ نشان می‌دهد که چگونه سودمندی از تغییر پاسخ خاص مشتق می‌شود. چنانچه در شکل ۱۲-۲ قابل مشاهده است سودمندی می‌تواند خطی، غیرخطی و یا به صورت تابع پله‌ای باشد. برای مثال، نمودار (c) سودمندی با شیب بالا رونده بر روی تغییرات جزئی در سطح پاسخ یک خصوصیت کیفی مانند کارایی را نشان می‌دهد. قابلیت دسترسی می‌تواند خیلی بهتر از طریق نمودار (a) مشخص شود که در آن تغییرات متوسط در سطح پاسخ منجر به تغییرات کوچک در سودمندی برای کاربر می‌شود.

استخراج مشخصات سودمندی از ذینفعان می‌تواند فرآیند طولانی و خسته کننده باشد. برای عملی کردن آن می‌توان فقط تقریبی از این منحنی‌ها (تقریبا مشابه) را از طریق ذینفعان استخراج کرد که آن نیز با استفاده از پنج مقدار پاسخ خصوصیات کیفی برای سناریوها قابل دستیابی است. چهار مقدار از اینها می‌تواند بدون در نظر گرفتن راهبرد معماری مشخص شوند اما مقدار پنجم وابسته به راهبرد استفاده شده است.

برای ساختن منحنی سودمندی-پاسخ، ابتدا باید بهترین حالت و بدترین حالت سطح خصوصیات کیفی مشخص شوند. بهترین حالت سطح خصوصیت کیفی از مقدار پیش‌بینی ذینفعان بیشتر بوده اما از سودمندی بیشتر نیست. برای مثال، پاسخ سیستم برای یک کاربر ۰.۱ ثانیه در نظر گرفته شده است بنابراین بهبود بیشتر از آن در جهت اینکه در ۰.۰۳ ثانیه پاسخ داده شود یک سودمندی نیست. به صورت مشابه، بدترین حالت سطح خصوصیات کیفی حداقل آستانه بالای آن چیزی است که سیستم باید انجام دهد. در غیر این صورت نتیجه آن برای ذینفعان قابل استفاده نیست. برای سطوح بدترین حالت و بهترین حالت، مقدار سودمندی ۰ و ۱۰۰ تخصیص داده می‌شود.

حال باید سطح سودمندی فعلی و مطلوب برای سناریوها مشخص شوند. مقادیر سودمندی (بین ۰ تا ۱۰۰) برای حالات فعلی و مطلوب، از ذینفعان با استفاده از مقادیر بهترین حالت و بدترین حالت به عنوان نقطه ارجاع استخراج می‌شوند. به عنوان مثال در یک سیستم، در حال حاضر به خوبی نصف حالتی هستیم که می‌خواهیم باشیم ولی اگر به سطح خصوصیات کیفی مطلوب رسیده شود، ۹۰ درصد سودمندی نهایی را خواهیم داشت لذا سطح سودمندی فعلی، مقدار ۵۰ و سطح سودمندی مطلوب مقدار ۹۰ را می‌گیرند. در این روش منحنی‌ها برای کلیه سناریوها تولید می‌شوند.

- *اولویت‌های سناریوها*

سناریوهای مختلف در یک سیستم، سطوح مختلفی از اهمیت را برای ذینفعان دارند و لذا سودمندی آنها نیز متفاوت است. برای مشخص کردن اهمیت نسبی هر سناریو، در یک فرآیند دو مرحله‌ای به هر سناریو وزنی تخصیص داده می‌شود. در مرحله اول ذینفعان به سناریوها رای می‌دهند تا اینکه یک ترتیب در میان آنها شکل بگیرد. رای‌گیری بر پایه مقدار پاسخ مورد انتظار هر سناریو است. سپس ذینفعان وزن ۱ را به سناریو بالاترین رتبه می‌دهند و یک مقدار کسری نیز به دیگر سناریوها مبتنی بر اهمیت نسبی‌شان می‌دهند. اگر در آینده سناریوهای دیگری اضافه شوند آنها نیز می‌توانند وزن بگیرند. ذینفعان در هنگام توافق می‌توانند مطمئن شوند که وزن سناریوها منطبق با برداشت[1] آنهاست.

[1] Intuition

- *راهبردهای معماری*

برای اینکه بتوان از سطح پاسخ فعلی خصوصیات کیفی به سطح مطلوب یا بهترین حالت حرکت کرد، معمار یا معماران باید راهبردهای معماری را مشخص کنند. در *CBAM* بخشی به این موضوع اختصاص داده شده است. برای هر راهبرد، موارد زیر را می‌توان بدست آورد:

- *مقدار مورد انتظار پاسخ، در هر سناریو* (سودمندی مقدار مورد انتظار با استفاده از درون‌یابی چهار مقدار استخراج شده از ذینفعان محاسبه می‌شود)

- *تاثیر راهبرد معماری بر روی دیگر خصوصیات کیفی مورد علاقه*

- *تخمین هزینه برای پیاده‌سازی راهبرد معماری*

- *تاثیرات جانبی*

هر راهبرد معماری فقط خصوصیت کیفی که در وضعیت فعلی مورد بررسی قرار می‌گیرد را تحت تاثیر قرار نخواهد داد، بلکه معمولا دیگر خصوصیات کیفی را نیز تحت تاثیر قرار خواهد داد (این موضوع نشان دهنده این است که چرا نیاز به مصالحه وجود دارد). بنابراین مهم است که سودمندی این تاثیرات جانبی اضافی پاسخهای خصوصیت که به عنوان نتیجه بکارگیری راهبرد هستند، تعیین شود. در بدترین حالت، باید یک نسخه جدید از سناریو برای خصوصیت تاثیر جانبی ایجاد شده و منحنی سودمندی-پاسخ آن تعیین شود. اما در عمل اگر خصوصیت کیفی برای ذینفعان مهم باشد در دیگر خصوصیات کیفی اتفاق افتاده است و منحنی سودمندی- پاسخ قبلا برای آن پاسخ ترسیم شده است. در این حالت، تنها چیزی که باقیمانده آن است که سودمندی مورد انتظار خصوصیت کیفی مرتبط با راهبرد معماری تعیین شود. توجه شود که سودمندی مورد انتظار یک خصوصیت خاص می‌تواند منفی باشد. البته به شرط آنکه راهبرد معماری طوری طراحی شده باشد که بر روی خصوصیتی تاکید کند که با خصوصیتی که در حال محاسبه سودمندی آن هستیم دارای تداخل باشد.

نهایتا اینکه وقتی اطلاعات اضافی استخراج شدند می‌توان فایده به کارگیری یک راهبرد معماری را به وسیله جمع کردن کلیه فواید خصوصیات کیفی مرتبط با آن بدست آورد.

- *مشخص کردن فایده و نرمال‌سازی*[1]

سودمندی کل یک راهبرد معماری در میان کلیه سناریوها از طریق جمع کردن سودمندی مرتبط با هر یک از آنها محاسبه می‌شود (هر سناریو با توجه به اهمیت خودش وزن می‌گیرد). برای هر راهبرد معماری i فایده آن یعنی B_i به صورت زیر محاسبه می‌شود.

$$B_i = \sum_j (b_{i,j} * W_j).$$

$b_{i,j}$ سود بدست آمده برای راهبرد i به دلیل تاثیر آن بر روی سناریو j است و W_j نیز وزن سناریوی jاست. همان طور که در شکل ۱۲-۲ قابل مشاهده است هر $b_{i,j}$ به صورت تغییر در سودمندی به وجود آمده از طریق راهبرد معماری محاسبه می‌شود : $b_{i,j} = U_{expected} - U_{current}$

این فرمول بدین معنی است که مقدار سودمندی مورد انتظار راهبرد معماری منهای سودمندی فعلی سیستم مرتبط به این سناریو است. تاثیر ضرب وزن W_j نیز آن است که از طریق اهمیت نسبی سناریوهای مختلف، مقدار این سودمندی نرمال شود.

۱۲-۳-۲- محاسبه بازگشت سرمایه

مقدار بازگشت سرمایه برای هر راهبرد معماری، نسبت فایده کل (B_i) به هزینه پیاده سازی آن (C_i) است. هزینه با استفاده از یک مدل مناسب برای سیستم و محیط توسعه محاسبه می‌شود.

راهبردهای معماری می‌توانند از طریق امتیاز بازگشت سرمایه رتبه‌بندی ترتیبی[2] شوند. این رتبه‌بندی ترتیبی می‌تواند برای تعیین ترتیب بهینه پیاده‌سازی راهبردهای مختلف مورد استفاده قرار بگیرد. منحنی a, b نشان داده شده در شکل ۱۲-۲ را در نظر بگیرید. منحنی a به عنوان یک پاسخ خصوصیت کیفی بهبود پیدا می‌کند. در این حالت، احتمالا به یک نقطه‌ای می‌رسد که بازگشت سرمایه کاهش می‌آید (دیگر ثابت می‌ماند). به عبارت دیگر ، مصرف بیشتر هزینه منجر به افزایش بیشتر سودمندی نخواهد شد. اما در منحنی b یک بهبود کوچک در پاسخ خصوصیت کیفی می‌تواند منجر به افزایش چشم‌گیری در سودمندی شود.

[1] Determining benefit and normalization
[2] Rank-ordered

۱۲-۴- پیاده‌سازی *CBAM*

تبدیل کردن مفاهیم و روشهای تشریح شده برای *CBAM* در بخش قبل به مجموعه‌ای از مراحل عملی، دربرگیرنده یک سری اصول است که نحوه تحقق آنها باید به صورتی باشد که میزان فعالیت‌های مورد نیاز حداقل شود. عملی کردن مبانی این تشریح در واقع به نوعی محدود کردن اندازه فضای تصمیم‌گیری است.

نمودار جریان فرآیند برای *CBAM* در شکل ۱۲-۳ نشان داده شده است. چهار مرحله اول با تعداد نسبی از سناریوهایی که آنها باید بررسی کنند، حاشیه گذاری شده است. تعداد سناریوها به صورت پیوسته کاهش پیدا می‌کند و تضمین می‌کند که این روش زمان ذینفعان را بر روی سناریوهایی که میزان بازگشت سرمایه بیشتری دارند، متمرکز می‌کند.

- *مرحله ۱: مرتب سازی سناریوها*[1]

سناریوهای استخراج شده در روش *ATAM* باید مرتب شده و این اجازه به ذینفعان داده شود تا اینکه سناریوهای جدیدی را معرفی کنند. سناریوها مبتنی بر توانایی خود در برآورده کردن اهداف حرفه سیستم اولویت‌بندی شده و یک سوم سناریوهای دارای بالاترین اولویت برای مطالعه بیشتر انتخاب شوند.

- *مرحله ۲: اصلاح (بهبود، پالایش) سناریوها*[2]

سناریوهای خروجی مرحله ۱ اصلاح شده و تمرکز بر روی معیار پاسخ محرک‌ها صورت می‌گیرد و بهترین، بدترین وضع، فعلی و مطلوب سطح پاسخ خصوصیات کیفی برای هر سناریو استخراج می‌شوند.

- *مرحله ۳: اولویت‌بندی سناریوها*[3]

به هر ذینفع صد رای داده می‌شود و از آنها خواسته می‌شود که این رای‌ها را بین سناریوها توزیع کنند. توزیع رای‌ها مبتنی بر مقدار پاسخ مطلوب برای هر سناریو است. رای‌های داده شده برای هر سناریو جمع شده و پنجاه درصد سناریوهای با بیشترین رای برای مرحله بعدی انتخاب می‌شوند. به

[1] Collate scenarios
[2] Refine scenarios
[3] Prioritize scenarios

سناریوی با بالاترین رای وزن یک تخصیص داده می‌شود و برای دیگر سناریوها نیز وزنی متناسب با وزن بالاترین سناریوها انتساب داده می‌شود. این وزن‌گذاری در محاسبه فایده کل هر راهبرد به کار گرفته می‌شود. نهایتا فهرستی از خصوصیات کیفی مرتبط به ذینفعان تهیه می‌شود.

- *مرحله ۴: انتساب سودمندی*

سودمندی برای سطح پاسخ خصوصیات کیفی (بهترین، بدترین، فعلی، مطلوب) کلیه سناریوهای مشخص شده در مرحله سوم تعیین می‌شود.

- *مرحله ۵: توسعه راهبردهای معماری برای سناریوها و تعیین سطح پاسخ مورد انتظار خصوصیات کیفی*

راهبردهای معماری که سناریوهای انتخاب شده را برآورده کرده، توسعه داده می‌شوند (یا از توسعه‌های پیشین استفاده می‌شود) و سطوح پاسخ خصوصیت کیفی مورد انتظار که از آنها حاصل خواهد شد، تعیین می‌شود. این موضوع باید در نظر گرفته شود که راهبرد معماری می‌تواند بر روی چندین سناریو تاثیر داشته باشد بنابراین باید این محاسبه برای هر تاثیر سناریو انجام شود.

- *مرحله ۶: تعیین سودمندی سطح پاسخ خصوصیات کیفی مورد انتظار از طریق درون‌یابی*

سودمندی سطح پاسخ خصوصیات کیفی مورد انتظار برای راهبرد معماری با استفاده از مقادیر سودمندی استخراج شده (از منحنی سودمندی) تعیین می‌شود. این عمل برای هر خصوصیت کیفی بدست آمده از مرحله ۳ انجام می‌شود.

- *مرحله ۷: محاسبه فایده کلی بدست آمده از راهبرد معماری*

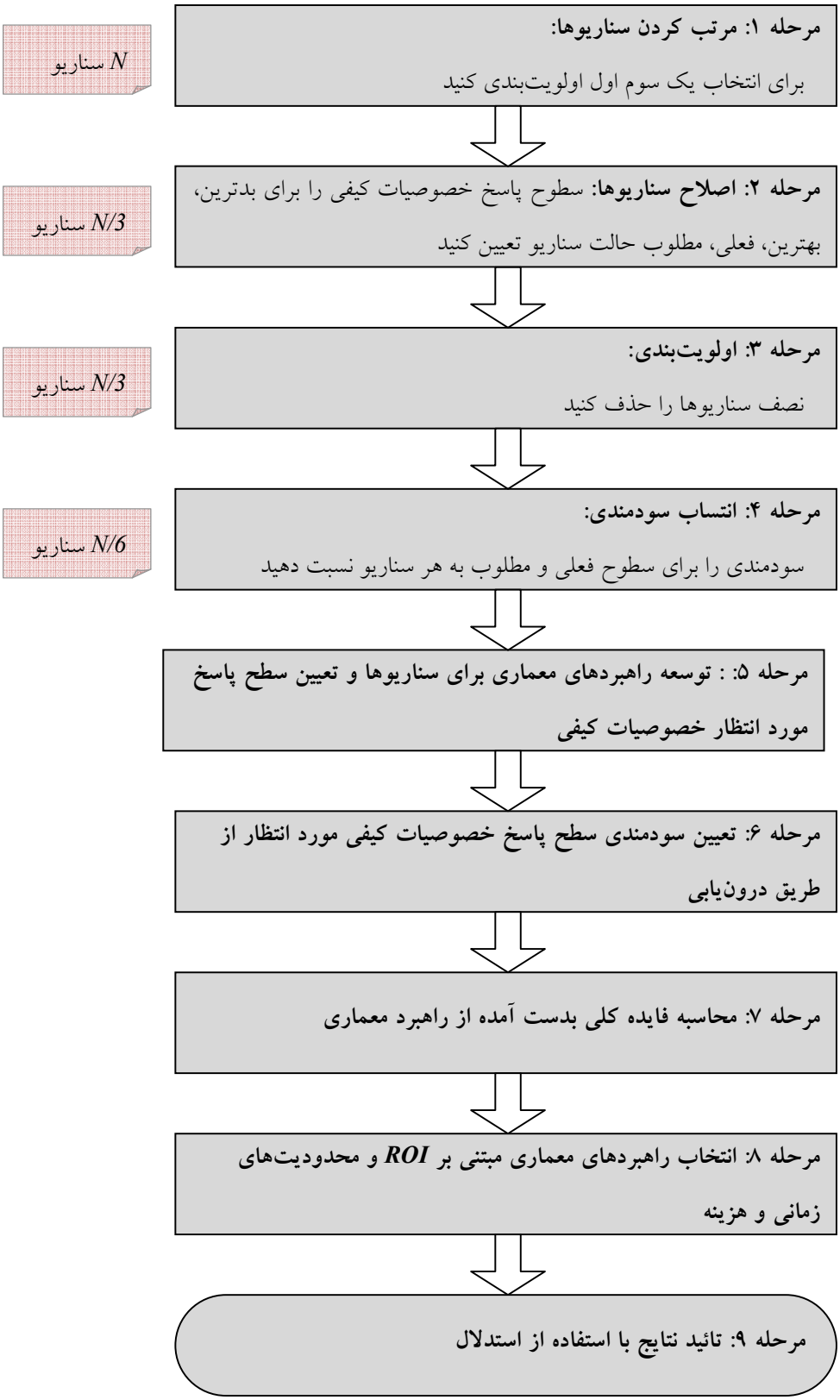
مقدار سودمندی فعلی از مقدار سطح مورد انتظار کم شده و با استفاده از رای‌های اخذ شده در مرحله ۳ عمل نرمال‌سازی انجام می‌شود. فایده یک سناریو خاص در سرتاسر سناریوها و کلیه خصوصیات کیفی مرتبط جمع می‌شوند.

- *مرحله ۸: انتخاب راهبردهای معماری مبتنی بر بازگشت سرمایه و محدودیت‌های زمانی و هزینه*

الزامات زمانبندی و هزینه هر راهبرد معماری تعیین می‌شود و مقدار بازگشت سرمایه برای هر یک از آنها محاسبه می‌شود. راهبردهای معماری را مبتنی بر میزان بازگشت سرمایه مرتب کرده و تا جایی که بودجه و زمان اجازه می‌دهد راهبردهای رتبه بالا را انتخاب کنید.

- *مرحله ۹: تائید نتایج با استفاده از استدلال*

برای راهبردهای انتخاب شده، بررسی صورت می‌پذیرد که آیا آنها همسو با اهداف حرفه سازمان هستند یا نه؟ اگر همسو نباشند مواردی که ممکن است در هنگام تحلیل نادیده گرفته شوند بررسی می‌شوند. اگر پیامد خاصی وجود داشته باشد تکرار دیگری از این مراحل انجام می‌شود.



شکل ۱۲-۳: نمودار جریان فرآیند *CBAM*

۱۲-۵- مطالعه موردی: *NASA ECS Project*

در این بخش نحوه اعمال *CBAM* بر روی یک سیستم دنیای واقعی به عنوان مثال عملی نشان داده خواهد شد. سیستم مشاهده زمین[1] یک مجموعه‌ای از ماهواره‌های *NASA* است که داده‌ها را برای سایر برنامه‌های تحقیقاتی ایالت متحده آمریکا[2] و دیگر انجمن‌ها در سرتاسر دنیا جمع‌آوری می‌کند. *ECS* و *EOSDIS*[3] داده‌ها را از ماهواره‌های مختلف برای پردازش بیشتر جمع‌آوری می‌کنند. ماموریت *ECS* آن است که داده‌ها را پردازش کرده و آنها را به اطلاعات سطح بالا تبدیل کند تا برای دانشمندان به یک شکل قابل تحقیق در بیایند. بنابراین در کل هدف آن است که روشی برای ذخیره کردن داده‌ها (و لذا پردازش آنها) و نیز سازوکاری برای معرفی قالب داده‌های جدید و الگوریتم‌های پردازشی فراهم کنند. این امر باعث می‌شود اطلاعات به صورت وسیع قابل استفاده باشند.

سیستم *ECS* روزانه رشته‌ای از صدها گیگابایت داده خام وابسته به محیط را به عنوان ورودی پردازش می‌کند. محاسبه دویست و پنجاه محصول استاندارد منجر به هزاران گیگا بایت اطلاعات می‌شود که در هشت مرکز داده در ایالات متحده بایگانی می‌شوند. نیازمندیهای مهم سیستم کارایی و قابلیت دسترس‌پذیری است. همچنین ماهیت طولانی مدت پروژه، قابلیت تغییرپذیری را نیز در این سیستم برجسته می‌کند.

مدیر سیستم *ECS* بودجه بسیار محدودی برای نگهداری و گسترش سیستم جاری دارد. از تحلیل قبلی انجام شده برای این سیستم (با استفاده از *ATAM*)، از طریق ذینفعان مجموعه بزرگی از تغییرات مطلوب برای سیستم استخراج شده که در کل منجر به مجموعه گسترده‌ای از راهبردهای معماری شده است. مشکل پیرامون این سیستم آن است که مجموعه کوچکی باید برای پیاده‌سازی انتخاب شود (به دلیل مشکلات مالی) یعنی چیزی در حدود ۱۰ تا ۲۰ درصد مواردی که به عنوان خصوصیات و عملکردهای سیستم در نظر گرفته شده‌اند. بنابراین مدیر از *CBAM* استفاده می‌کند تا یک تصمیم منطقی مبتنی بر معیارهای اقتصادی بازگشت سرمایه اتخاذ کند. در اجرای *CBAM*، تمرکز بر روی بخش *DAWG*[4] سیستم *ECS* خواهد بود.

[1] Earth observing system
[2] U.S. Global Change Reseach Program
[3] Earth Observing System Data Information System (EOSDIS) Core System (ECS)
[4] Data Access Working Group

۱۲-۵-۱- مرحله ا: مرتب کردن سناریوها

سناریوهای بدست آمده از *ATAM* به علاوه یک مجموعه از سناریوهای جدید استخراج شده از ذینفعان گرد هم آمده[1] از بخشهای سیستم *ECS*، مرتب‌سازی شده‌اند. از آنجا که سهامداران در جریان اجرای *ATAM* بودند، این مرحله نسبتاً ساده بود. یک مجموعه از سناریوهای خام توسط تیم *DAWG* مطرح شدند که در جدول ۱۲-۱ نشان داده شده است. نکته قابل توجه این است که بسیاری از این سناریوها هنوز خوش فرم نیستند و برای بعضی از آنها هنوز پاسخی تعریف نشده است. این پیامدها در مرحله دوم زمانی که تعداد سناریوها کاهش خواهد یافت، حل خواهند شد.

جدول ۱۲-۱: سناریوهای مرتب شده در یک ترتیب اولویت‌دار

سناریو	شرح سناریو
۱	خرابی‌های توزیع داده که منجر به هنگ کردن توزیع درخواست‌ها می‌شود و نیاز به مداخله دستی دارد را کاهش دهید
۲	خرابی‌های توزیع داده که منجر به از بین رفتن توزیع درخواست‌ها می‌شود را کاهش دهید
۳	تعداد سفارشاتی که در فرآیند تحویل سفارش خطا می‌دهند را کاهش دهید
۴	خرابی‌های سفارش که منجر به هنگ کردن سفارش می‌شود که نیاز به مداخله دستی دارد را کاهش دهید
۵	خرابی‌های سفارش که منجر به از بین رفتن سفارش می‌شود را کاهش دهید
۶	هیچ روش خوبی برای پیگیری سفارش رد شده/لغو شده مهمان *ECS* بدون مداخله دستی وجود ندارد (مانند *spreadsheet*)
۷	کاربران به اطلاعات بیشتری نیاز دارند که چرا سفارش آنها برای داده دچار خطا شده است
۸	به دلیل محدودیت‌ها، این نیاز وجود دارد که اندازه و تعداد سفارشات به طور غیرواقعی محدود شود
۹	سفارشات کوچک منجر به اخطارهای زیاد به کاربر می‌شود
۱۰	سیستم باید درخواست ۵۰ گیگا بایت کاربر را در یک روز و درخواست یک ترا بایت کاربر را در یک هفته پردازش کند.

[1] Assembled

۱۲-۵-۲- مرحله ۲: اصلاح سناریوها

سناریوها در این مرحله اصلاح شدند و توجه خاصی به مشخص کردن معیارهای پاسخ-محرک آنها شد. اهداف پاسخ بهترین حالت، بدترین حالت، حالت فعلی و حالت مطلوب برای هر سناریو استخراج و ثبت شد که این موضوعات در جدول ۱۲-۲ نشان داده شده است.

جدول ۱۲-۲: اهداف پاسخ برای سناریوهای اصلاح شده

اهداف پاسخ				
سناریو	بدترین	فعلی	مطلوب	بهترین
۱	*10% hung*	*5% hung*	*1% hung*	*0% hung*
۲	*> 5% lost*	*< 1% lost*	*0% lost*	*0% lost*
۳	*10% fail*	*5% fail*	*1% fail*	*0% fail*
۴	*10% hung*	*5% hung*	*1% hung*	*0% hung*
۵	*10% lost*	*< 1% lost*	*0% lost*	*0% lost*
۶	*50% need help*	*25% need help*	*0% need help*	*0% need help*
۷	*10% get information*	*50% get information*	*100% get information*	*100% get information*
۸	*50% limited*	*30% limited*	*0% limited*	*0% limited*
۹	*1/granule*	*1/granule*	*1/100 granules*	*1/1,000 granules*
۱۰	*< 50% meet goal*	*60% meet goal*	*80% meet goal*	*> 90% meet goal*

۱۲-۵-۳- مرحله ۳: اولویت‌بندی سناریوها

در رای‌گیری نمایش‌های اصلاح شده سناریوها، تیم قدری از روش منحرف شد. به جای رای‌گیری انفرادی، آنها بر سر هر سناریو بحث کردند و وزن‌ها را با اتفاق نظر تعیین کردند. آراء اختصاص یافته به مجموعه کامل سناریوها به ۱۰۰ محدود شد که در جدول ۱۲-۳ نشان داده شده است. هر چند که از سهامداران خواسته نشده بود که رای‌ها مضرب پنج باشد، آنها فکر کردند که این یک تصمیم منطقی است و نیاز و توجیهی برای دقت بیشتر وجود نداشت.

جدول ۱۲-۳: سناریوهای اصلاح شده با رای‌گیری

اهداف پاسخ					
سناریو	آراء	بدترین	فعلی	مطلوب	بهترین
۱	۱۰	*10% hung*	*5% hung*	*1% hung*	*0% hung*
۲	۱۵	*> 5% lost*	*< 1% lost*	*0% lost*	*0% lost*
۳	۱۵	*10% fail*	*5% fail*	*1% fail*	*0% fail*
۴	۱۰	*10% hung*	*5% hung*	*1% hung*	*0% hung*
۵	۱۵	*10% lost*	*< 1% lost*	*0% lost*	*0% lost*
۶	۱۰	*50% need help*	*25% need help*	*0% need help*	*0% need help*
۷	۵	*10% get information*	*50% get information*	*100% get information*	*100% get information*
۸	۵	*50% limited*	*30% limited*	*0% limited*	*0% limited*
۹	۱۰	*1/granule*	*1/granule*	*1/100 granules*	*1/1,000 granules*
۱۰	۵	*< 50% meet goal*	*60% meet goal*	*80% meet goal*	*> 90% meet goal*

۱۲-۵-۴- مرحله ۴: انتساب سودمندی

در این مرحله سودمندی برای هر یک از سناریوها به وسیله ذینفعان تعیین می‌شود. این بار نیز مشخص کردن سودمندی به صورت مشورتی انجام شد. یک امتیاز سودمندی صفر عدم وجود سودمندی را نشان می‌دهد و امتیاز صد نیز بیشترین سودمندی ممکن را نشان می دهد. نتایج این فرآیند در جدول ۱۲-۴ نشان داده شده است.

جدول ۱۲-۴: سناریو با آراء و امیتاز سودمندی

امتیازهای کارایی					
سناریو	آراء	بدترین	فعلی	مطلوب	بهترین
۱	۱۰	۱۰	۸۰	۹۵	۱۰۰
۲	۱۵	۰	۷۰	۱۰۰	۱۰۰
۳	۱۵	۲۵	۷۰	۱۰۰	۱۰۰
۴	۱۰	۱۰	۸۰	۹۵	۱۰۰

۵	۱۵	۰	۷۰	۱۰۰	۱۰۰
۶	۱۰	۰	۸۰	۱۰۰	۱۰۰
۷	۵	۱۰	۷۰	۱۰۰	۱۰۰
۸	۵	۰	۲۰	۱۰۰	۱۰۰
۹	۱۰	۵۰	۵۰	۸۰	۹۰
۱۰	۵	۰	۷۰	۹۰	۱۰۰

۱۲-۵-۵- مرحله ۵: توسعه راهبردهای معماری برای سناریوها و تعیین سطح پاسخ مورد انتظار خصوصیات کیفی

بر اساس نیازمندیهای اعمال شده توسط سناریوهای پیشین، یک مجموعه از ۱۰ راهبرد معماری توسط معماران *ECS* توسعه یافت. به خاطر داشته باشید که یک راهبرد معماری ممکن است بر بیش از یک سناریو تاثیر بگذارد. برای تشکیل دادن این روابط پیچیده، سطح پاسخ خصوصیات کیفی مورد انتظاری که هر راهبرد پیش‌بینی می‌کند به آن دست پیدا کند باید با توجه به هر سناریوی مرتبط تعیین شوند.

مجموعه راهبردهای معماری به علاوه تعیین سناریوهایی که به آنها اشاره دارد در جدول ۱۲-۵ نشان داده شده است. برای هر زوج راهبرد معماری/سناریو، با توجه به سناریوهای نشان داده شده انتظار می‌رود به سطوح پاسخ مورد انتظار دست پیدا کرد (همراه با پاسخ فعلی به منظور مقایسه).

جدول ۱۲-۵: راهبردهای معماری و سناریوهای برآورده شده

راهبرد	نام	شرح	سناریوهای تحت تاثیر	پاسخ فعلی	پاسخ مورد انتظار
۱	استمرار دستور در زمان ارسال	دستور را به محض اینکه به سیستم رسید ذخیره کنید	*3* *5* *6*	*5% fail* *<1% lost* *25% need help*	*2% Fail* *0% lost* *0% need help*
۲	تکه کردن دستور	به اپراتورها اجازه دهید دستورات بزرگ را به چندین دستور کوچک تقسیم کند	*8*	*30% limited*	*15% limited*
۳	دسته کردن دستور	چندین دستور کوچک را در یک دستور بزرگ ترکیب	*9* *10*	*1 per granule* *60% meet*	*1 per 100* *55% meet goal*

		کنید		*goal*	
۴	بخش بندی دستور	به اپراتور اجازه دهید تا از مواردی که به خاطر مسائل دسترس‌پذیری یا کیفیت داده نمی‌تواند بازیابی شود ، رد شود	*4*	*5% hung*	*2% hung*
۵	انتساب دوباره دستور	به اپراتور اجازه دوباره تخصیص دادن نوع رسانه برای آیتم‌ها در یک دستور بدهید	*1*	*5% hung*	*2% hung*
۶	اجرای مجدد دستور	به اپراتور اجازه دهید تا بتواند دستور یا آیتم‌های یک دستور را که به خاطر مشکلات موقتی سیستم یا داده انجام نشده را مجداً اجرا کند	*4*	*5% hung*	*3% hung*
۷	اجبار در اتمام دستور	به اپراتور اجازه دهید تا از آیتم‌هایی که به دلیل محدودیت‌های کیفی داده در دسترس نیستند، بگذرند	*1*	*5% hung*	*3% hung*
۸	اخطار برای انجام نشدن دستور	تضمین کنید فقط زمانی که واقعاً قسمتی از دستور آنها انجام نشده به کاربران اخطار داده شود و مشروح وضع هر آیتم را در اختیار بگذارید. اخطار دادن به کاربر فقط اگر اپراتور اخطار را تائید کند اتفاق می‌افتد. کاربر ممکن است اخطار را ویرایش کند.	*6*	*25% need help*	*20% need help*
۹	دانه‌بندی در سطح پیگیری	اپراتور و کاربر وضع هر آیتم در دستورشان را می‌توانند تعیین کنند	*6*	*25% need help*	*10% need help*

۱۰	ارتباط با اطلاعات کاربر	یک اپراتور بتواند به سرعت اطلاعات تماس کاربر را پیدا کند. سرویس‌دهنده به اطلاعات *SDSRV* دسترسی خواهد داشت تا محدودیت‌های هر داده‌ای را تعیین کند که ممکن است بخش‌هایی از دستور/دستورات را به کار ببرد و بفرستد. قابلیت‌های توزیع شدگی شامل *DDIST*، *PDS*، *external subsetters* و ابزارهای پردازش داده و غیره را به خود اختصاص دهد.	*7*	*50% get information*	*60% get information*

۱۲-۵-۶- مرحله ۶: تعیین سودمندی سطح پاسخ خصوصیات کیفی مورد انتظار از طریق درون‌یابی

قبلاً سطح پاسخ مورد انتظار هر راهبرد معماری با توجه به مجموعه سناریوها مشخص شده است. با مد نظر داشتن امتیازهای سودمندی هر پاسخ فعلی و مطلوب برای تمام خصوصیات تحت تاثیر قرار گرفته، می‌توان سودمندی آنها را محاسبه کرد. با استفاده از این امتیازها می‌توان از طریق درونیابی سودمندی سطوح پاسخ خصوصیت کیفی مورد انتظار برای زوج راهبرد معماری/سناریو به کار برده شده توسط *DAWG* مربوط به *ECS* را محاسبه کرد. نتایج این محاسبات در جدول ۱۲-۶ نشان داده شده است که برای زوج‌های راهبرد معماری/ سناریو ارائه شده در جدول ۱۲-۵ است.

جدول ۱۲-۶: راهبردهای معماری و سودمندی مورد انتظار آنها

راهبرد	راهبرد	سناریوهای تحت تاثیر	کارایی فعلی	کارایی مورد انتظار
۱	استمرار دستور در زمان ارسال	۳	۷۰	۹۰
		۵	۷۰	۱۰۰
		۶	۸۰	۱۰۰
۲	تکه کردن دستور	۸	۲۰	۶۰

۳	دسته کردن دستور	۹ ۱۰	۵۰ ۷۰	۸۰ ۶۵
۴	بخش بندی دستور	۴	۸۰	۹۰
۵	انتساب دوباره دستور	۱	۸۰	۹۲
۶	اجرای مجدد دستور	۴	۸۰	۸۵
۷	اجبار در اتمام دستور	۱	۸۰	۸۷
۸	اخطار برای انجام نشدن دستور	۶ ۷	۸۰ ۷۰	۸۵ ۹۰
۹	دانه‌بندی در سطح پیگیری	۶ ۷	۸۰ ۷۰	۹۰ ۹۵
۱۰	ارتباط با اطلاعات کاربر	7	۷۰	۷۵

۱۲-۵-۷- مرحله ۷: محاسبه فایده کلی بدست آمده از راهبرد معماری

بر اساس اطلاعات جمع‌آوری شده که در جدول ۱۲-۶ نشان داده شده است اکنون فایده کلی هر راهبرد معماری را بر اساس فرمول ذکر شده می‌توان محاسبه کرد. این فرمول فایده کلی را به شکل جمع فایده‌های متعلق به هر سناریو که توسط وزن نسبی سناریو نرمال شده، محاسبه می‌کند. امتیازهای فایده کلی برای هر راهبرد معماری در جدول ۱۲-۷ ارائه شده است.

جدول ۱۲-۷: فایده کلی راهبردهای معماری

راهبرد	سناریوی تحت تاثیر	وزن سناریو	سود خام راهبرد معماری	سود نرمال شده راهبرد معماری	سود کلی راهبرد معماری
۱	۳	۱۵	۲۰	۳۰۰	۹۵۰
۱	۵	۱۵	۳۰	۴۵۰	
۱	۶	۱۰	۲۰	۲۰۰	
۲	۸	۵	۴۰	۲۰۰	۲۰۰

۳	۹	۱۰	۳۰	۳۰۰	۲۷۵
۳	۱۰	۵	-۵	-۲۵	
۴	۴	۱۰	۱۰	۱۰۰	۱۰۰
۵	۱	۱۰	۱۲	۱۲۰	۱۲۰
۶	۴	۱۰	۵	۵۰	۵۰
۷	۱	۱۰	۷	۷۰	۷۰
۸	۶	۱۰	۵	۵۰	۱۵۰
۸	۷	۵	۲۰	۱۰۰	
۹	۶	۱۰	۱۰	۱۰۰	۲۲۵
۹	۷	۵	۲۵	۱۲۵	
۱۰	۷	۵	۵	۲۵	۲۵

۱۲-۵-۸- مرحله ۸: انتخاب راهبردهای معماری مبتنی بر بازگشت سرمایه و محدودیت‌های زمانی و هزینه

برای اتمام تحلیل، تیم هزینه هر راهبرد معماری را تخمین می‌زند. تخمین بر اساس تجربه درباره سیستم و مقدار بازگشت سرمایه برای هر راهبرد معماری محاسبه می‌شود. با استفاده از بازگشت سرمایه امکان اولویت‌بندی هر راهبرد وجود دارد. اولویت‌بندی مربوط به راهبردها در جدول ۱۲-۸ نشان داده است. تعجب آور نیست که اولویت‌ها تقریباً همان ترتیبی را دارند که راهبردها پیشنهاد شده‌اند: راهبرد ۱ بالاترین اولویت را دارد. راهبرد ۳ دومین اولویت را دارد. راهبرد ۹ کمترین اولویت را دارد و راهبرد ۸ دومین کمترین اولویت را.

جدول ۱۲-۸: بازگشت سرمایه راهبردهای معماری

راهبرد	هزینه	سود کلی راهبرد	*ROI* راهبرد	اولویت راهبرد
۱	۱۲۰۰	۹۵۰	۰.۷۹	۱
۲	۴۰۰	۲۰۰	۰.۵	۳
۳	۴۰۰	۲۷۵	۰.۶۹	۲
۴	۲۰۰	۱۰	۰.۵	۳
۵	۴۰۰	۱۲۰	۰.۳	۷
۶	۲۰۰	۵۰	۰.۲۵	۸
۷	۲۰۰	۷۰	۰.۳۵	۶

۸	۳۰۰	۱۵۰	۰.۵	۳
۹	۱۰۰۰	۲۲۵	۰.۲۲	۱۰
۱۰	۱۰۰	۲۵	۰.۲۵	۸

۱۲-۵-۹- مرحله ۹: تائید نتایج با استفاده از استدلال

آشکارترین نتایج *CBAM* در جدول ۱۲-۸ نشان داده شده است در این شکل راهبردهای معماری بر اساس بازگشت سرمایه پیش‌بینی شده برای آنها مرتب شده‌اند. خروجی حاصل از *CBAM* فراتر از خروجی کیفی است. این روش همچنین مزایایی فرهنگی و اجتماعی دارد.

درست همان قدر که اولویت‌بندی راهبردها در *CBAM* مهم است، مباحثی که فرآیند جمع‌آوری اطلاعات و تصمیم‌گیری را با هم توام می‌کند نیز مهم است. فرآیند *CBAM* تعداد زیادی ساختار برای بحث‌های سازمان نیافته فراهم می‌کند. جاهایی که نیازمندیها و راهبردهای معماری مخلوط شده و جاهایی که اهداف محرک و پاسخ به وضوح بیان نشده است. فرآیند *CBAM* سهامداران را وادار می‌کند تا سناریوهایشان را برای تعیین نتیجه سودمندی از قبل واضح کنند. در پایان، این فرآیند منتجر به شفاف کردن سناریوها و نیازمندی‌ها می‌شود که خودش مزیت مهمی است.

فهرست مطالب

فصل سیزدهم

مطالعه موردی در زمینه قابلیت تعامل‌پذیری

در این فصل، موضوع وب از نقطه نظر چرخه حرفه معماری تفسیر شده و نشان داده خواهد شد که چگونه تغییرات در معماری وب، تغییرات در اهداف و نیازهای حرفه ذینفعان مختلف آن را بازتاب می‌دهد. بدین منظور ابتدا نگاهی به ماهیت اصلی وب، نیازمندیها و ذینفعان اصلی آن انداخته و سپس به بررسی این موضوع پرداخته خواهد شد که چگونه معماری سمت سرویس‌دهنده به عنوان نتیجه‌ای از چرخه حرفه معماری تغییر پیدا کرده است.

۱۳-۱- ارتباط با چرخه حرفه معماری

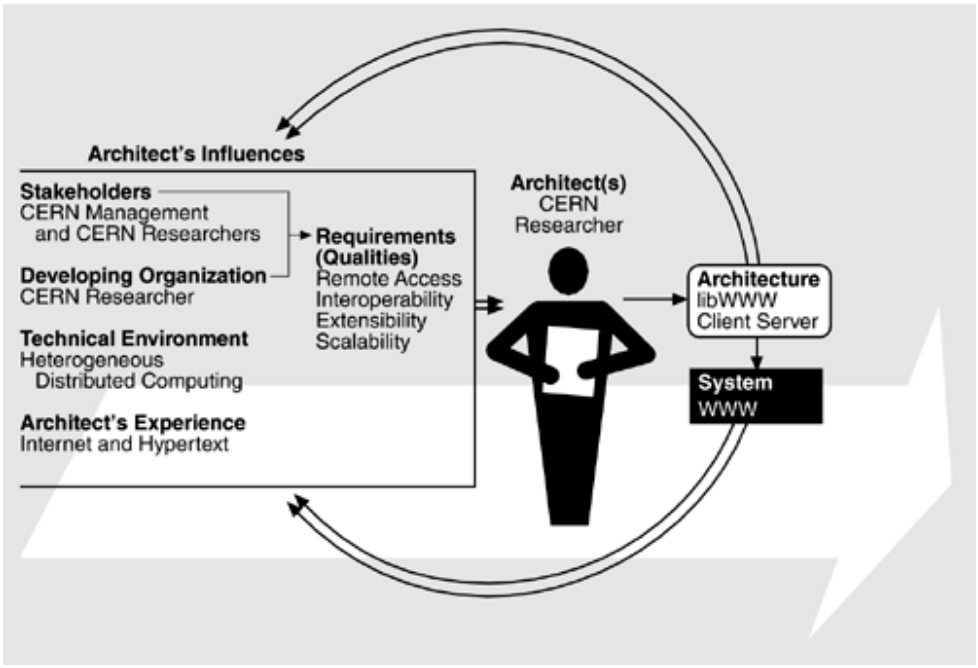
یکی از جالب‌ترین نمونه‌ها از نحوه عملکرد چرخه حرفه معماری, وب[1] است که در آن اهداف، مدل حرفه و معماری از زمان معرفی (سال ۱۹۹۰)، مدام در حال تغییر است. پیشنهاد اصلی وب از جانب آقای ***Tim Berners-Lee*** بود (محققی از آزمایشگاه فیزیک ذره‌ای اروپا[2]) که مشاهده کرد چندین هزار محقق در ***CERN*** شکل یک وب (تار عنکبوت) را به خود گرفته‌اند. در ***CERN*** محققان زیاد و مختلفی رفت و آمد می‌کردند، کارهای تحقیقاتی جدیدی را با یکدیگر انجام می‌دادند، مقاله‌های علمی خودشان را با یکدیگر به اشتراک می‌گذاشتند و در راهروهای آزمایشگاه با یکدیگر صحبت می‌کردند. بنابراین ***Berners-Lee*** تصمیم گرفت که از این وب غیررسمی از طریق وب الکترونیکی حمایت کند. در سال ۱۹۸۹ او یک مستند تحت عنوان مدیریت اطلاعات ایجاد و در ***CERN*** منتشر کرد (در واقع یک طرح پیشنهادی بود). تا سال ۱۹۹۰ یک نسخه اصلاح شده از طرح پیشنهادی به وسیله مدیریت تصویب شد و نام آن وب گذاشته شد و بعد از آن فرآیند توسعه آغاز گردید.

شکل ۱۳-۱ عناصر چرخه حرفه معماری برای طرح پیشنهادی اولیه پذیرفته شده به وسیله مدیریت ***CERN*** را نشان می‌دهد. هدف از سیستم، گسترش تعاملات بین محققان ***CERN*** (کاربران نهایی) در داخل محدودیت‌های محیط محاسباتی غیرهمگن بود. مشتری، مدیریت ***CERN*** بود. سازمان توسعه‌دهنده یک محقق مدیریت ***CERN*** بود. مورد حرفه که به وسیله ***Berners-Lee*** ایجاد شده بود سیستمی بود که ارتباطات بین کارکنان ***CERN*** را افزایش می‌داد. این سیستم یک طرح پیشنهادی با اهداف خیلی محدود بود و هیچ راهی وجود نداشت که مشخص شود آیا یک چنین سیستمی ارتباطات را افزایش می‌دهد یا نه؟ از طرف دیگر هم بودجه در اختیار ***CERN*** برای تولید و توسعه این سیستم نیز بسیار محدود بود. آنها فقط قادر بودند هزینه یک محقق را آن هم فقط برای چند ماه تامین کنند.

محیط فنی برای موسسه آشنا بود زیرا اینترنت از زمان معرفی آن در اوایل سال ۱۹۷۰ نقش بسیاری مهمی را در آنجا ایفا کرده بود. شبکه (اینترنت) مفهوم ضعیفی در مورد کنترل مرکزی (انجمن‌های مسئول مشخص کردن قراردادهای ارتباطی بین گره‌های مختلف در اینترنت) و سبک غیرمعمول از نحوه تعاملات (مخصوصا در میان

[1] World Wild Web (WWW)
[2] European Laboratory Particle Physics (CERN)

گروه‌های خبری خاص) داشت. سیستمهای ابرمتن تاریخچه طولانی از رویدادها را از زمان شروع به کار با استفاده از دید آقای ***Vannervar Bush*** به دنبال داشتند. دید ***Bush*** در دهه‌های ۱۹۶۰،۱۹۷۰ و همچنین ۱۹۸۰ مورد تحقیق و بررسی قرار گرفت. در این سالها به صورت مرتب کنفرانس‌هایی در مورد ابرمتن‌ها تشکیل می‌شد و محققان مختلف در این حوزه را دور هم جمع می‌کرد. با این حال، دید ***Bush*** تا دهه ۱۹۸۰ برای مقیاس بزرگ اثبات شده نبود. در آن زمان استفاده از ابرمتن به صورت خاص برای سیستمهای مستندسازی مقیاس کوچک، محدود شده بود.



شکل ۱۳-۱: چرخه حرفه معماری برای وب

مدیریت ***CERN*** طرح پیشنهادی ***Berners-Lee*** را در سال ۱۹۹۰ تصویب کرد. تا نوامبر او نخستین برنامه وب خود را بر روی سکوی ***NeXT*** توسعه داد که به وضوح نشان‌دهنده آن است که او کار بر روی پیاده‌سازی را قبل از اینکه تائیدیه رسمی را از مدیریت دریافت کند، شروع کرده بود. این نوع ارتباط ضعیف بین موافقت مدیریت و فعالیت محقق در سازمانهای تحقیقاتی که نیازمند سرمایه اندک اولیه است، کاملا معمول است. سازمانهای تحقیقاتی اغلب تمایل دارند تا پروژه‌ها را از پائین با بالا توسعه دهند در حالی که در سازمانهای تجاری به ندرت یک چنین نگرشی وجود دارد. دلیل این امر ناشی از آن است که سازمانهای تحقیقاتی به ابتکار

و خلاقیت محققین وابسته هستند و اجازه آزادی زیادی را نسبت به چیزی که در سازمانهای تجاری معمول است به محققین خود می‌دهند.

پیاده‌سازی اولیه سیستم وب، ویژگی‌های زیادی داشت که امروزه از بسیاری مرورگرهای وب برداشته شده‌اند. برای مثال به کاربران اجازه داده می‌شد که از داخل مرورگر یک پیوند[1] ایجاد کند یا اینکه خوانندگان یا نویسندگان اطلاعاتی را از طریق مرورگر تشریح کنند. آقای ***Berners-Lee*** در آغاز فکر می‌کرد که کاربران مشتاق نخواهند بود که ***HTML*** را یاد گرفته یا آن را بنویسند یا اینکه با ***URL*** سروکار داشته باشند. اما او در اشتباه بود زیرا کاربران خواهان آن بودند که با این سختی‌ها کار کنند تا اینکه بتوانند قدرت انتشار بر روی وب را به دست آورند.

۱۳-۲- نیازمندیها و خصوصیات کیفی

وب از زمانی که توسط موسسه ***CERN*** درک و به صورت اولیه پیاده‌سازی شد چندین خصوصیت کیفی مطلوب داشت. وب قابلیت حمل و تعامل با انواع کامپیوترهای اجرا کننده نرم‌افزار یکسان را داشت و همچنین مقیاس‌پذیر و قابل گسترش نیز بود. اهداف تجاری از جمله تعاملات گسترش یافته و اجازه محاسبات غیرهمگن منجر به اهداف کیفی همچون دسترسی از راه‌دور، قابلیت تعامل‌پذیری[2]، قابلیت گسترش و مقیاس‌پذیری شد و این اهداف کیفی نیز منجر به ***libWWW*** شد. ***libWWW*** کتابخانه اصلی نرم‌افزار است و از توسعه مبتنی بر وب و معماری توزیع‌شده سرویس‌دهنده و سرویس‌گیرنده حمایت می‌کند.

تحقق این خصوصیات در معماری اصلی نرم‌افزار یک زیربنایی را ایجاد کرد که به صورت موثر از رشد شگفت‌انگیز وب حمایت کرد (جدول ۱۳-۱ رشد وب را نشان می دهد). ***libWWW*** دربرگیرنده جداسازی دغدغه‌ها است و بنابراین به صورت مجازی بر روی هر سخت‌افزاری کار می‌کند و به سادگی قراردادهها، قالب داده‌ها و کاربردهای جدید را می‌پذیرد. همچنین به دلیل اینکه وب کنترل مرکزی ندارد می‌تواند بدون هیچ گونه محدودیتی رشد پیدا کند.

[1] Link
[2] Interoperability

جدول ۱۳-۱: رشد آماری وب

تاریخ سال/ماه	تعداد وب سایت‌ها	درصد سایت‌های .com	نسبت میزبان‌ها به سرویس‌دهنده وب[1]
۶/۹۳	۱۳۰	۱.۵	۱۳۰۰۰
۱۲/۹۳	۶۲۳	۴.۶	۳۴۷۵
۶/۹۴	۲۷۳۸	۱۳.۵	۱۰۹۵
۱۲/۹۴	۱۰۰۲۲	۱۸.۳	۴۵۱
۶/۹۵	۲۳۵۰۰	۳۱.۳	۲۷۰
۱/۹۶	۱۰۰۰۰۰	۵۰.۰	۹۴
۶/۹۶	۲۵۲۰۰۰	۶۸.۰	۴۱
۱/۹۷	۶۴۶۱۶۲	۶۲.۶	۴۰
۱/۹۸	۱۸۳۴۷۱۰	-	۱۶.۲
۱/۹۹	۴۰۶۲۲۸۰	-	۱۰.۶
۱/۰۰	۹۹۵۰۴۹۱	-	۷.۳
۱/۰۱	۲۷۵۸۵۷۱۹	۵۴.۶۸	۴.۰

۱۳-۲-۱- نیازمندیهای اولیه (اصلی)

مجموعه نیازمندیهای اولیه برای وب که در طرح پیشنهادی اولیه مطرح شده بودند به شرح زیر هستند:

- *دسترسی راه دور از طریق شبکه*

اطلاعات باید قابلیت دسترسی از طریق هر کامپیوتر متعلق به شبکه ***CERN*** را داشته باشند.

- *ناهمگنی*

سیستم نباید محدود به اجرا شدن بر روی سخت‌افزار یا نرم‌افزار خاص شود.

- *عدم تمرکز*

با وجود وب انسانی[2] و اینترنتی، یک منبع داده یا سرویس منفرد نمی‌تواند وجود داشته باشد. این نیازمندی بیان کننده این موضوع است که وب می‌تواند رشد کند. در واقع عملیات پیوند به یک مستند به نوبه خود منجر به عدم تمرکز می‌شود.

[1] Host per web server
[2] Human web

§ *دسترسی به داده‌های موجود*

پایگاه‌داده‌های موجود مجبور هستند که قابل دسترس باشند.

§ *توانایی افزایش داده توسط کاربر*

کاربر باید قادر باشد داده‌های متعلق به خود را منتشر کند. این عمل از طریق واسطی که کاربر برای خواندن داده‌های دیگر استفاده می‌کند، صورت می‌پذیرد.

§ *پیوندهای اختصاصی*

پیوندها و گره‌ها باید قابلیت این را داشته باشند که به صورت اختصاصی معرفی شوند.

§ *ویژگیها یا دستگاه‌های جانبی* [1]

تنها قالب نمایش داده که در ابتدا مدنظر بود نمایش بر روی پایانه‌های اسکی ۸۰*۲۴ حرفی بود.

§ *تحلیل داده‌ها*

کاربر باید قادر باشد در پایگاه‌داده‌های مختلف جستجو کرده و همچنین داده‌های نامتعارف، قانونی و غیرقانونی را نیز جستجو کند. به عنوان مثال ***Berners-Lee*** این توانایی را فراهم کرد که بتوان نرم‌افزارهای مستند نشده و سازمانهای بدون کارمند را جستجو کرد.

§ *پیوندهای زنده*

در نظر بگیرید که اطلاعات پیوسته در حال تغییر هستند بنابراین باید راهکارهایی برای به‌روزرسانی دید کاربر نسبت به تغییرات وجود داشته باشد. این عمل می‌تواند به سادگی با استفاده از تکنیک‌هایی انجام شود. به عنوان مثال هر موقع پیوند مورد ارجاع قرار می‌گیرد تغییرات نمایش داده شود و یا اینکه هر موقع تغییرات انجام شد پیوند جدید برای کاربر ارسال شود.

علاوه بر این نیازمندیها، تعدادی غیرنیازمندی[2] شناسایی شده نیز وجود داشتند. برای مثال، اعمال حق کپی و امنیت داده از جمله نیازمندیهایی بودند که به صورت صریح مورد ارجاع قرار گرفته بودند ولی در پروژه اولیه هیچ توجهی به آنها نشده بود.

[1] Bell and whistles
[2] Nonrequirements

وب یک رسانه عمومی بود (درک اولیه از وب منجر به این تفکر شده بود) همچنین طرح پیشنهادی اصلی نیز به طور صریح بیان کرده بود که کاربران نباید مجبور باشند که از یک قالب نماد خاص[1] استفاده کنند. معیارها و ویژگیهای دیگری که در طرح پیشنهادی برای سیستمهای ابرمتن مشترک هستند ولی در طرح پیشنهادی برای وب نادیده گرفته شده‌اند عبارتند از :

- کنترل توپولوژی (مکان‌شناسی)
- تعریف فنون ناوبری و نیازمندیهای واسط کاربری از جمله نگاه داشتن یک تاریخچه بصری
- داشتن انواع مختلف پیوند برای بیان رابطه‌های متفاوت در میان گره‌ها

اگرچه بسیاری از نیازمندیهای اولیه ماهیت چیزی که امروزه وب نامیده می‌شود را شکل داده‌اند ولی تعدادی از این نیازمندیها تحقق پیدا نکرده و یا اینکه تاثیر تحقق آنها بسیار ناچیر بوده است. برای مثال امروزه قابلیت‌هایی همچون تحلیل داده، پیوندهای زنده و پیوندهای اختصاصی هنوز به طور کامل پشتیبانی نمی‌شوند. تاثیر تحقق بعضی نیازمندیها که در ابتدا خیلی ناچیز در نظر گرفته شده بودند می‌تواند تغییر پیدا کند. به عنوان مثال "ویژگیها یا دستگاه‌های جانبی" تصویری، حجم بالایی از ترافیک وب امروزی را به وجود می‌آوردند. امروزه تصاویر بخش عمده‌ای از توجه همگان را به خود اختصاص داده‌اند و حجم بزرگی از ترافیک شبکه را مصرف می‌کنند. در زمان پیشنهاد طرح اولیه وب، مدیریت ***CERN*** و آقای ***Berners-Lee*** این موضوع را در نظر نگرفته بودند، به طوری که مرورگرهای اولیه مبتنی بر متن بودند. به شکل مشابه طرح اصلی وب خصوصیاتی پیرامون جستجوی چندرسانه‌ای برای پشتیبانی از صدا و تصویر نداشت.

بعضی غیرنیازمندیها ممکن است به نیازمندی تبدیل شوند. برای مثال، اثبات شده است که امنیت باید به عنوان یک پیامد ذاتی در نظر گرفته شود که این افزایش اهمیت امنیت در وب ناشی از ترافیک تجاری است (در طرح اولیه توجه زیادی به این نیازمندی نشده بود). پیامد امنیت بزرگ و پیچیده است. همچنین رسیدن به امنیت زمانی که دسترسی محافظت شده به داده‌های اختصاصی، تضمین شده نیست بسیار دشوار است. این موضوع کاملا در سالهای اخیر آشکار شد از آنجایی که تجارت الکترونیکی شروع به تغییر دادن ساختار و تعدادی زیادی از مکانیزمها کرد که برای تسهیل عملکرد ایجاد شده بودند. آشکارترین این موضوعات رمزگذاری

[1] Particular markup format

داده‌های حساس است که معمولا از طریق ***SSL***[1] انجام می‌شود (در مرورگر به صورت ***HTTPS***[2] نمایش داده می‌شود). این قرارداد فقط امکان گوش دادن و خواندن داده‌های اختصاصی توسط افراد غیرمجاز را در زمانی که آنها در یک شبکه عمومی، انتقال داده می‌شوند را کاهش می‌دهد. راه حلهای دیگر مانند ***Microsoft Passport*** نیز برای اثبات این موضوع به کار می‌روند تا مشخص شود که فرد استفاده کننده همان کسی است که ادعا می‌کند.

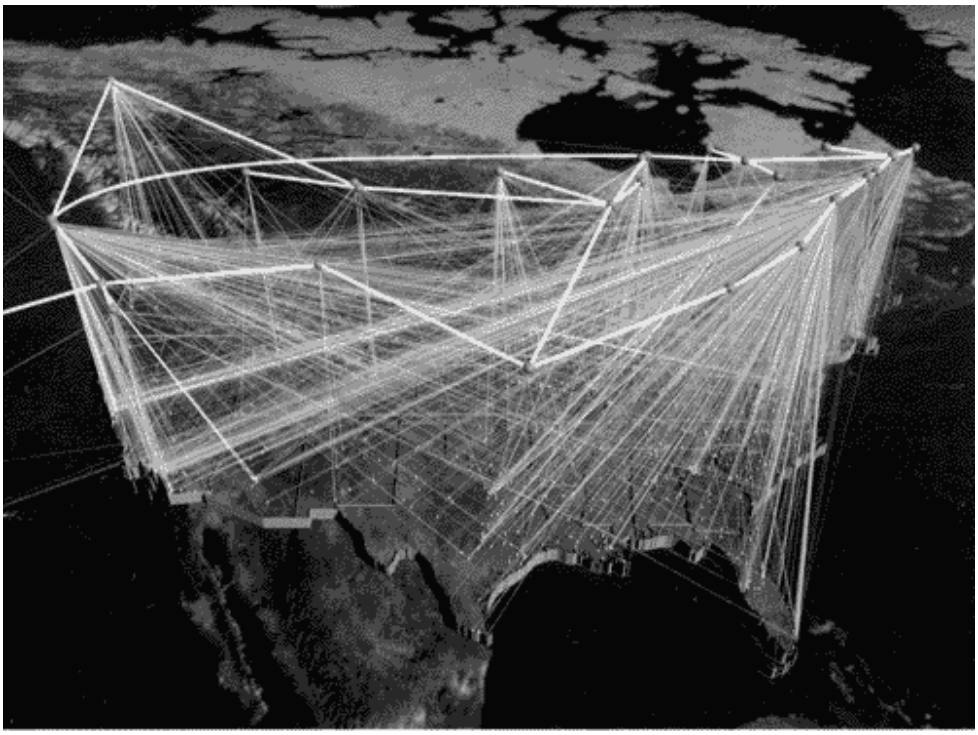
۱۳-۲-۱- نیازمندیهای جدید و منسوخ شده[3]

در سالهای گذشته هیچ کسی نتوانسته است که رشد سریع و فوق‌العاده اینترنت یا وب را پیش‌بینی کند. مبتنی بر آمار اخیر که در جدول ۱۳-۱ نشان داده شده است وب از نظر اندازه هر سه یا شش ماه دو برابر می‌شود (از ۱۳۰ سایت در میانه سال ۱۹۹۳ تا بیش از ۲۳۰۰۰۰ سایت در میانه سال ۱۹۹۶ و همچنین ۲ میلیون سایت در ابتدای سال ۲۰۰۱). شکل ۱۳-۲ مسیرهای ارتباط اصلی برای پوشش اینترنت در ایالات متحده را نشان می‌دهد. همچنین تعداد میزبانهای اینترنی از ۱.۳ میلیون در سال ۱۹۹۳ به ۹.۵ میلیون در اواسط ۱۹۹۶ رسیده است (این آمار از طریق ***IP*** های ثبت شده بدست آمده است).

هر دوی وب و اینترنت رشد کرده‌اند اما وب در کل سریعترین رشد را داشته است. این موضوع به راحتی از ستون آخر جدول ۱۳-۱ قابل مشاهده است. در این ستون مشاهده می‌شود که نسبت میزبان به سرویس‌دهنده وب در حال کاهش است. این موضوع نشان دهنده این است که بخش بزرگی از میزبانها در حال تبدیل شدن به سرویس‌دهندگان وب هستند. علاوه بر رشد سریع و باورنکردنی وب، ماهیت آن نیز تغییر پیدا کرده است (در ستون سوم جدول ۱۳-۱ قابل مشاهده است). اگرچه شروع آن در موسسه تحقیقاتی بود ولی به سرعت تحت کنترل ترافیک تجاری قرار گرفت (سایت‌هایی با پسوند ***.com***). درصد سایت‌های ***.com*** درحدود ۵۵ درصد است اما این موضوع منجر به افزایش دیگر دامنه‌ها مانند ***.net*** و ***.biz*** شد البته بدون اینکه به فعالیت‌های تجاری لطمه‌ای وارد کند (یعنی میزان سایت‌های ***.com*** کم شود).

[1] Secure Sockets Layer
[2] HyperText Transfer Protocol Secure)
[3] Requirements come and go

ظهور دسترسی گسترده و آسان به وب یک تاثیر جانبی ساده داشت. دسترسی آسان به تصاویر در یک روش توزیع شده و به صورت غیرمتمرکز صنعت "***cyberporn***" را تولید کرد که منجر به نیازمندیهای جدید شد. یعنی محتوا برچسب‌گذاری شده و دسترسی به آن کنترل شود. نتیجه این موضوع یک سکو برای مشخصات ***PICS***[1] است. ***PICS*** یک مجموعه گسترده صنعتی از اصول و پیاده‌سازی تجاری از آنها است که اجازه می‌دهد محتوا را برچسب‌گذاری کرده و به صورت انعطاف‌پذیر قابلیت انتخاب معیارها را ممکن می‌سازد. در این روش تولید کنندگان محتوا محدود به تولید محتوای خاص نمی‌شوند. برای مثال، والدین می‌توانند فرزندان خود را از دیدن فیلم‌ها یا عکس‌های غیرمناسب منع کنند و یا مدیر می‌تواند از دسترسی کارمند خود را به سایت‌های غیرمرتبط با حرفه سازمان در زمان کار جلوگیری کند.

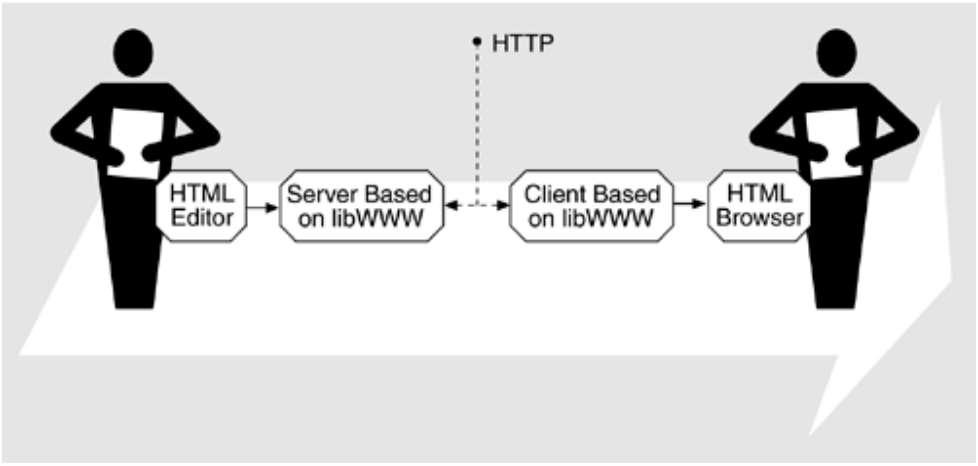


شکل ۱۳-۲: ساختار اصلی اینترنت در ایالات متحده

[1] Platform for Internet Context Selection

۱۳-۳- راه‌حل معمارانه

نخستین معماری استفاده شده در موسسه ***CERN*** و سپس در کنسرسیوم جهانی وب مبتکی بر سرویس‌دهنده‌ها، سرویس‌گیرنده‌ها و کتابخانه ***libWWW*** است که تمام وابستگی‌های سخت‌افزاری، نرم‌افزاری و قراردادها را مخفی می‌کند. شکل ۱۳-۳ نشان می‌دهد که چگونه تولید کننده و مصرف کننده محتوا از طریق سرویس‌دهنده‌ها و سرویس‌گیرنده‌ها با یکدیگر ارتباط برقرار می‌کنند. تولید کننده، محتوایی را که از طریق ***HTML*** تشریح شده است در ماشین سرویس‌دهنده قرار می‌دهد. نرم‌افزار در هر دو سمت یعنی سرویس‌دهنده و سرویس‌گیرنده مبتنی بر ***libWWW*** است بنابراین جزئیات قرارداد و وابستگی بین سکوها از دید نرم‌افزار پنهان است. یکی از عناصر سمت سرویس‌گیرنده مرورگر است که می‌داند چگونه ***HTML*** را نمایش دهد تا اینکه مصرف کننده بتواند درک درستی از محتوای ارائه شده داشته باشد.

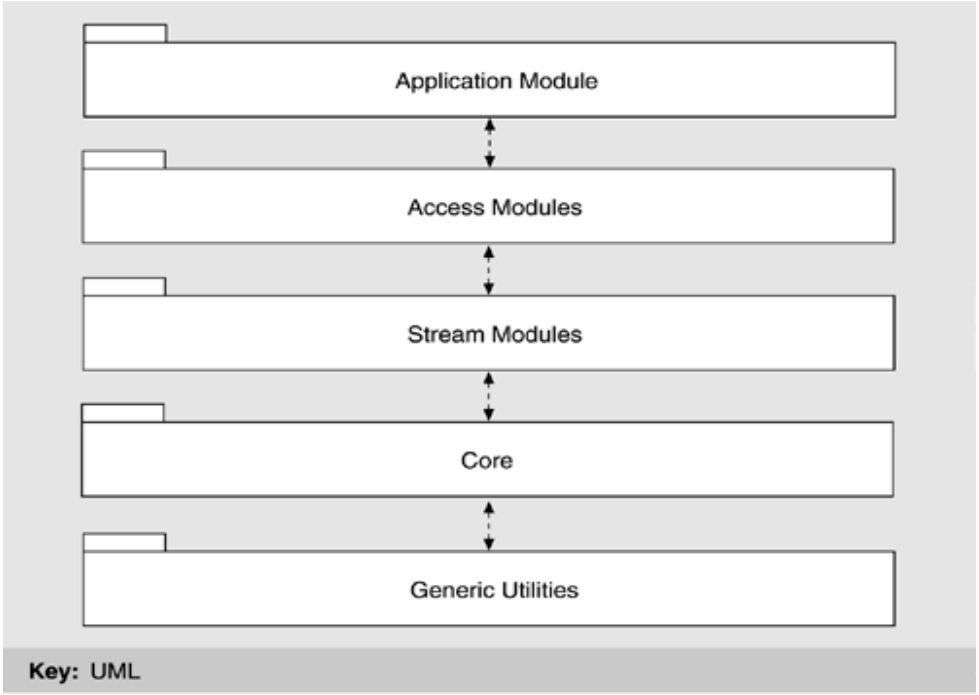


شکل ۱۳-۳: نحوه ارتباط تولید کننده و مصرف کننده محتوا از طریق سرویس‌گیرنده و سرویس‌دهنده

در ادامه به برررسی جزئیات مربوط به ***libWWW*** و معماری سرویس‌گیرنده و سرویس‌دهنده پرداخته خواهد شد که به عنوان پایه‌ای برای وب اصلی است و امروزه نیز بخش بزرگی از معماری سیستمهای مبتنی بر وب را تشکیل می‌دهد.

۱۳-۳-۱- برآورده کردن نیازمندیهای اصلی: *libWWW*

همان طور که قبلا بیان شد ***libWWW*** یک کتابخانه نرم‌افزار برای ایجاد برنامه‌های کاربردی است که بر روی سرویس‌دهنده یا سرویس‌گیرنده اجرا می‌شوند. آن عملکردهای عمومی را که در میان برنامه‌های کاربردی مختلف مشترک است را به اشتراک می‌گذارد. از این نوع عملکردها می‌توان توانایی اتصال به میزبان راه‌دور یا توانایی درک جریان داده‌های مبتنی بر ***HTML*** را نام برد. ***libWWW*** یک کتابخانه فشرده و قابل حمل است که می‌تواند برای ایجاد برنامه‌های کاربردی مبتنی بر وب از قبیل سرویس‌دهنده‌ها، سرویس‌گیرنده‌ها، پایگاه‌داده‌ها و



تارهای وب[1] مورد استفاده قرار بگیرد. ***libWWW*** به پنج لایه تقسیم شده است که این لایه‌ها در شکل ۱۳-۴ نشان داده شده‌اند.

شکل ۱۳-۴: دید لایه‌ای از *libWWW*

لایه سودمندیهای عمومی[2]، یک لایه قابل حمل فراهم می‌کنند که سایر قسمت‌های سیستم بر روی آن قرار می‌گیرند. این لایه شامل بلوک‌های سازنده اصلی برای سیستم از جمله مدیریت شبکه، انواع داده و برنامه‌های کاربردی مربوط به دستکاری رشته است. از طریق سرویس فراهم شده توسط این لایه، کلیه لایه‌های بالایی

[1] Web spiders
[2] Generic utilities

می‌توانند مستقل از سکو شوند و همچنین وظیفه وصل شدن به سخت‌افزار و یا سکوی نرم‌افزار جدید می‌تواند محدود به وصل شدن به واسطهای لایه سودمندیها شود.

لایه هسته[1] دربرگیرنده اسکلت عملکردی برنامه کاربردی وب مانند دسترسی به شبکه، مدیریت داده و تجزیه، واقعه‌نگاری و غیره است. این لایه به تنهایی هیچ کاربردی ندارد. بلکه یک واسط استاندارد برای ایجاد شدن برنامه‌ها کاربردی وب بر روی آن فراهم می‌کند. این لایه، عملکردهای واقعی را به وسیله ماژولها ***plug-in*** و ***call-out*** فراهم می‌کند که آنها نیز به وسیله برنامه‌های کاربردی ثبت می‌شوند. ماژولهای ***Plug-in*** در زمان اجرا در سیستم ثبت می‌شوند و کارهای اصلی لایه هسته (ارسال و نگهداری داده) را انجام می‌دهند. آنها معمولا از قراردادها، مدیریت انتقال سطح پائین و درک قالب داده‌ها حمایت می‌کنند. این ماژولها می‌توانند به صورت پویا تغییر پیدا کنند و این موضوع باعث می‌شود که به آسانی عملکردها یا رویدادهای جدید را برای تغییر ماهیت برنامه‌های کاربردی وب اضافه کرد. عملکردهای ***Call-out*** نیز روش دیگری برای گسترش عملکردهای فراهم شده در لایه هسته فراهم می‌کند. آنها عملکردهای خاص اختیاری متعلق به برنامه‌های کاربردی هستند که می‌توانند قبل یا بعد از درخواست برای ماژول قرارداد، فراخونی شوند.

حال سئوال این است که چه رابطه‌ای بین لایه سودمندی و لایه هسته وجود دارد؟ لایه سودمندیهای عمومی، عملکردهای مستقل از سکو را فراهم می‌کنند اما آنها می‌توانند برای ایجاد هر برنامه کاربردی متعلق به شبکه به کار روند. در طرف دیگر لایه هسته، انتزاع‌های خاصی را برای ساخت برنامه‌های کابردی تحت وب فراهم می‌کند.

لایه جریان[2]، انتزاعی از جریان داده استفاده شده برای انتقال کلیه داده‌های بین برنامه کاربری و شبکه فراهم می‌کند. لایه دسترسی[3] مجموعه‌ای از ماژولهای آگاه به قراردادهای شبکه فراهم می‌کند. مجموعه استانداردهایی که ***libWWW*** به صورت پایه‌ای از آنها حمایت می‌کنند ***HTTP, NNTP***[4]***, WAIS***[5]***, FTP***[6]***, TELNET, relog, Gopher, local file system*** و ***TN3270*** هستند. بسیاری از اینها در حال خارج شدن از چرخه هستند در

[1] Core layer
[2] Stream layer
[3] Access layer
[4] Network News Transport Protocol
[5] Wide Area Information Server
[6] File Transfer Protocol

حالی که انواع دیگری مانند ***HTTPS*** به آنها اضافه شده است. اضافه کردن یک ماژول قرارداده جدید کار بسیار ساده‌ای است زیرا بر روی انتزاع لایه‌های پائین ساخته می‌شود. لایه ماژول متشکل از ماژولهای برنامه کاربردی وب است. این یک برنامه کاربردی واقعی نیست بلکه مجموعه‌ای از عملکردهای مفید برای نوشتن (ایجاد) برنامه‌های کاربردی است. آن شامل ماژولهایی برای عملکردهای عمومی مانند نهان‌سازی، واقعه‌نگاری، ثبت سرویس‌دهنده پراکسی[1] و دروازه[2] و نگهداری است.

۱۳-۳-۲- درس‌های آموخته شده از *libWWW*

از ایجاد برنامه‌های کاربردی مبتنی بر ***libWWW*** چندین نکته آموخته شده است. این نکات از تجارب توسعه‌دهندگان در هنگام تلاش برای برآورده کردن نیازمندیهای فهرست شده در بخش ۱۳-۲ بدست آمده‌اند (غیرهمگنی، کنترل از راه‌دور در سرتاسر شبکه، عدم تمرکز و غیره). با این حال نیازمندی که مستلزم تلاشهای زیادی برای برآورده شدن بود رخدادهای غیر قابل پیش‌بینی را معرفی می‌کرد. یعنی اینکه اجازه دادن به رشد خصوصیاتی از برنامه‌های کابردی منجر به تصمیمات زیادی در ***libWWW*** شد و تصمیمات نیز باعث دستیابی به نکات زیر شدند:

- *واسط‌های برنامه‌نویسی کاربردی رسمی مورد نیاز هستند.*

  اینها واسطهایی هستند که عملکردهای موجود در ***libWWW*** را برای برنامه‌های ساخته شده مبتنی بر آن ارائه می‌کنند. واسط‌های برنامه‌نویسی کاربردی باید در یک روش مستقل از زبان مشخص شوند زیرا ***libWWW*** خواهان حمایت از توسعه برنامه‌های کاربردی در دامنه گسترده‌ای از زبانها و سکوها است.

- *عملکردها و واسط‌های برنامه‌نویسی کاربردی باید لایه‌بندی شوند.*

  برنامه‌های کاربردی مختلف نیاز خواهند داشت که به سطوح متفاوت انتزاع سرویس که بیشتر توسط لایه‌ها فراهم می‌شوند، دسترسی داشته باشند.

- *کتابخانه باید از یک مجموعه خصوصیات پویا و باز (نامحدود)*[3] *حمایت کند.*

[1] Proxy servers
[2] Gateway
[3] Open-ended

کلیه خصوصیات باید قابل جایگذاری باشند و همچنین این قابلیت نیز باید وجود داشته باشد که تعویض آنها در زمان اجرا امکان‌پذیر باشد.

§ *فرآیندهای ایجاد شده در نرم‌افزار باید از نظر نخ‌کشی امن باشند.*

برنامه‌های کاربردی مبتنی بر وب باید از قابلیت اجرای چندین عملکرد به صورت همزمان حمایت کنند، مخصوصا عملیات‌هایی همچون بارگیری فایل حجیم از طریق پیوند ارتباطی کند که می‌تواند زمان قابل توجهی را به خود اختصاص دهد. بنابراین عملکردهای استفاده شده به وسیله واسط‌های برنامه‌نویسی کاربردی باید صحیح و امن باشند تا در محیط نخ بتوانند اجرا شوند.

کاملا واضح است که ***libWWW*** از کلیه این اهداف حمایت نمی‌کند. برای مثال هسته ***libWWW*** چندین فرض در مورد سرویس‌های لازم و ضروری در نظر می‌گیرد بنابراین کلیه خصوصیات نمی‌تواند به صورت پویا جایگذاری شوند. بعلاوه، ***libWWW*** گرایش به اجرا شدن بر روی سکوهای مختلف دارد و آن نمی‌تواند وابسته به یک مدل نخ منفرد[1] باشد. بنابراین به صورت شبه نخ پیاده‌سازی شده تا تعدادی از عملکردهای مورد نیاز را فراهم کند. نهایتا بیشتر برنامه‌های کاربردی وب از پیکربندی خصوصیات به صورت پویا حمایت نمی‌کنند. یعنی آنها نیاز دارند قبل از اینکه سرویس جدید بتواند ثبت شود، راه‌اندازی مجدد شوند.

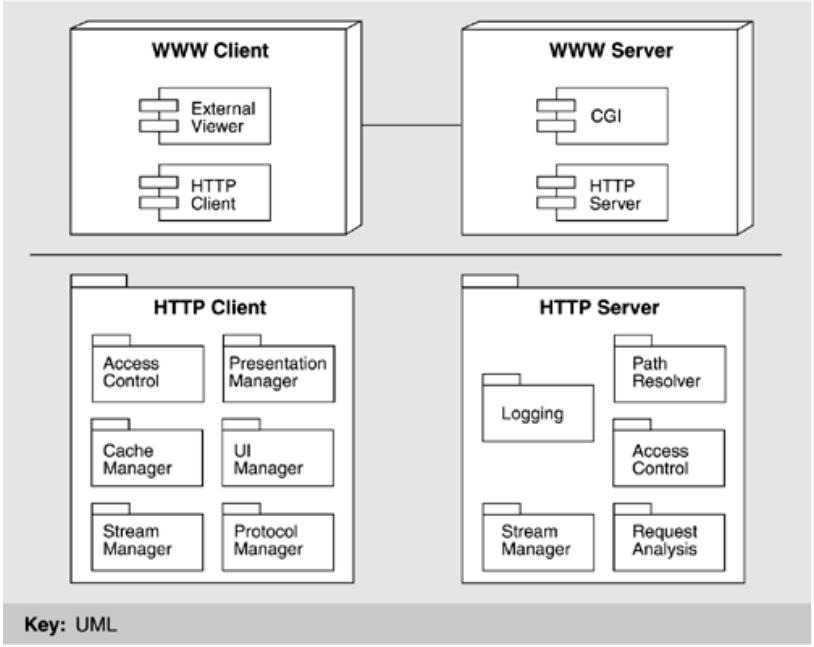
۱۳-۳-۳- یک معماری سرویس‌گیرنده و سرویس‌دهنده اولیه با استفاده از *libWWW*

در شکل ۱۳-۵ یک دید استقرار از سرویس‌گیرنده و سرویس‌دهنده وب عمومی ساخته شده با استفاده از سرویس‌های ***libWWW*** نشان داده شده است. همچنین یک دید تجزیه ماژول برای مولفه‌های سرویس‌گیرنده و سرویس‌دهنده ***HTTP*** در دید استقرار نشان داده شده است. شکل ۱۳-۵ چندین نکته در مورد ***libWWW*** را نشان می‌دهد. اولین نکته این است که کلیه بخش‌های سرویس‌گیرنده و سرویس‌دهنده مبتنی بر آن ساخته نشده‌اند. برای مثال واسط کاربری مستقل است. دومین نکته این است که نام مدیران با نام لایه‌ها به صورت مستقیم تطابق ندارند. اگرچه مدیر دسترسی، مدیر قرارداد و مدیر جریان به صورت واضح به لایه دسترسی و جریان مرتبط هستند، مدیر نهان‌سازی از سرویس‌های لایه کاربرد استفاده می‌کند. مدیران جریان در هر دوی

[1] Single-thread model

سرویس‌گیرنده و سرویس‌دهنده ارتباط سطح پائین را مدیریت می‌کنند بنابراین ارتباط شفاف در سرتاسر شبکه برای دیگر بخش‌های سیستم را تضمین می‌کند.

مدیر واسط کاربری شکل و ظاهر واسط کاربری سرویس‌گیرنده را مدیریت می‌کند. با این حال، مجموعه‌ای نامحدود از منابع را در نظر بگیرید که سیستم ***WWW*** می‌تواند آن را مدیریت کند. مدیر نمایش[1] می‌تواند اطلاعات نمایش داده شده برای برنامه‌های خارجی را مدیریت کند (در واقع یک نماینده برای این کار است). برای مثال، بیشتر نمایش‌دهنده‌های وب از برنامه خارجی برای نمایش ***PostScript*** یا فایلهای ***pdf*** استفاده می‌کنند. این نوع نمایندگی یا واسطه‌گری توافقی بین یکپارچگی واسط کاربری و قابلیت گسترش است.



شکل ۱۳-۵: دید استقرار سرویس‌گیرنده و سرویس‌دهنده وب همراه با دید تجزیه ماژول مولفه *HTTP*

مدیر واسط کاربری درخواست کاربر برای دریافت اطلاعات را به صورت یک ***URL*** دریافت کرده و آن را به مدیر دسترسی منتقل می‌کند. مدیر دسترسی تعیین می‌کند که آیا ***URL*** درخواستی در حافظه نهان است یا نه؟ و همچنین راهبری مبتنی بر تاریخچه را نیز تفسیر می‌کند (یعنی این امکان را فراهم می‌کند که بتوان به صفحه قبلی برگشت). اگر فایل در حافظه نهان وجود داشته باشد از مدیر نهان‌سازی دریافت شده و برای مدیر نمایش

[1] Presentation manager

برای نمایش در واسط کاربری یا دیگر نمایش‌دهنده‌های خروجی ارسال می‌شود. اگر آن فایل در حافظه نهان نباشد مدیر قرارداد نوع درخواست را تعیین می‌کند و قرارداد مناسب برای آن سرویس را فرخوانی می‌کند. مدیر جریان سرویس‌گیرنده از این قرارداد برای برقراری ارتباط با سرویس‌دهنده و انجام درخواست استفاده می‌کند. زمانی که مدیر جریان یک درخواست از سرویس‌دهنده در قالب مستندی دریافت کرد این اطلاعات به مدیر نمایش به منظور نمایش در محل مناسب انتقال داده می‌شود. مدیر نمایش با یک فایل پیکربندی کنترل نمایش ایستا[1] ارتباط برقرار می‌کند تا بتواند به صورت مناسب عمل نگاشت انواع مستندات به نمایش‌دهنده‌های خارجی را انجام دهد (در واقع از آن فایل کمک و راهنمایی می‌گیرد).

سرویس‌دهنده ***HTTP*** دسترسی شفاف به فایل‌های سیستم را تضمین می‌کند. این عمل از طریق کنترل دسترسی مستقیم یا از طریق یک پراکسی که به واسط دروازه عمومی ***(CGI)***[2] مشهور است انجام می‌گردد. ***CGI*** انواع منابع که یک سرور محلی نمی‌تواند مدیریت کند را مدیریت کرده و همچنین گسترش‌های دیگر از عملکردهای سرویس‌دهنده را نیز مدیریت می‌کند. سرویس‌دهنده‌های قابل دسترس ***WWW*** یک مجموعه از درخواست‌های تعریف شده ***HTTP*** را پیاده‌سازی کرده‌اند که اجازه بازیابی مستندات، فوق داده مستندات و برنامه‌های اجرا شونده در سمت سرویس‌دهنده از طریق ***CGI*** را می‌دهد.

زمانی که یک درخواست توسط مدیر جریان سرویس‌دهنده دریافت می‌شود نوع آن تعیین شده و مسیر ***URL*** از طریق مشخص کننده مسیر[3]، مشخص می‌شود. سرویس‌دهنده ***HTTP*** با یک فهرست دسترسی همفکری می‌کند تا تعیین کند که آیا درخواست سرویس‌گیرنده مجاز به این دسترسی است یا نه؟ ممکن است یک جلسه[4] تشخیص تصدیق کلمه عبور با سرویس‌گیرنده را آغاز کند تا دسترسی به داده‌ها را به آن اجازه دهد. در هر صورت ***CGI*** یکی از اصلی‌ترین روشها برای فراهم آوردن گستردگی برای سرویس‌دهنده است. همچنین مهمترین نیازمندی در به اجرا در آوردن تکامل نرم‌افزارهای وب است (***CGI*** یک جنبه مهم برنامه‌های کاربردی تحت وب است).

۱۳-۳-۴- دستیابی اولیه به اهداف کیفی

[1] Static view control configuration file
[2] Common gateway interface (CGI)
[3] Path resolver
[4] Session

جدول ۱۳-۲ نشان می‌دهد که وب چگونه به اهداف کیفی اولیه خود شامل دسترسی از راه دور، تعامل‌پذیری، قابلیت گسترش و مقیاس‌پذیری دست پیدا کرده است.

جدول ۱۳-۲: چگونگی دستیابی *WWW* به هاداف کیفی اولیه خود

هدف	چگونه انجام می‌شود	تاکتیک‌های مورد استفاده
دسترسی از راه دور	وب بر روی اینترنت ساخته می‌شود	- الصاق به قراردادهای تعریف شده
تعامل‌پذیری	استفاده از ***libwww*** برای پوشش جزئیات سکو	- تجرید سرویس‌های مشترک - پنهان‌سازی اطلاعات
قابلیت گسترش نرم‌افزار	جداسازی پروتکل و گسترش‌های نوع داده از ***libwww***	- تجرید سرویس‌های مشترک - پنهان‌سازی اطلاعات - مولفه‌های قابل جایگذاری - فایل‌های پیکربندی
قابلیت گسترش داده	هر قلم داده‌ای به جز برای ارجاع به کنترلش مستقل است	- گزینه‌های ممکن محدود
مقیاس‌پذیری	استفاده از معماری سرویس‌گیرنده وسرویس‌دهنده و نگهداری ارجاعات به دیگر داده‌های محلی برای اشاره به محل داده	- معرفی همزمانی - کاهش سرباز محاسباتی

۱۳-۴- چرخه حرفه معماری جدید

موفقیت باور نکردنی وب منجر به توجه غیرقابل پیش‌بینی از جانب حرفه شد و لذا منجر به اجبارها و فشارهای زیاد بر روی معماری از طریق چرخه حرفه معماری شد. بنابراین نیازمندیهای حرفه بر معماری وب تسلط پیدا کردند (یعنی اولویت بسیار بالایی را در ساختار معماری به دست آورند). همچنین وب سایت‌های ***B2B***[1] و ***B2C***[2] بسیاری از ابتکارات جدید در نرم‌افزارهای مبتنی بر وب را تحریک کردند.

مفهوم اصلی وب به عنوان یک مجموعه‌ای (تارهایی) از مستندات بود که در ارتباط با ابرمتن‌های اصلی خود نگهداری می‌شدند. با این حال، تجارت الکترونیکی وب را به صورت وب داده‌ها نمایش داد و این دیدهای

[1] Business-To-Business
[2] Business-To-Customer

مختلف منجر به بعضی تنش‌ها[1] شد. برای مثال، اعمال کردن داده برای کاربر حالت‌های مختلفی دارد. عمومی‌ترین روش برای به‌روزرسانی داده بارگذاری آن در دوره‌های زمانی خاص به جای تغییر داده با اعمال یک صفحه خاص است که در آن کاربر را وادار می‌کند که عمل به‌روزرسانی را انجام دهد. یا می‌توان دکمه "عقب" در مرورگر را در نظر گرفت که در حالت‌های خاص می‌تواند منجر به این شود که داده‌های قبلی قابل دسترس باشند. نیازمندیهای جدید تجارت الکترونیکی دقیق و کاملا متفاوت از نیازمندیهای اصلی ارائه شده برای وب در بخش ۱۳-۲ است. این نیازمندیها عبارتند از:

§ *کارایی بالا*

یک وب سایت محبوب معمولا ده‌ها میلیون بیننده در هر روز خواهد داشت و کاربران کمترین تاخیر را از جانب سایت خواهان هستند. همچنین مشتریان این موضوع را قبول نمی‌کنند که وب سایت به سادگی درخواست‌های آنها را قبول نکرده و یا رد کند (به دلیل بارکاری زیاد).

§ *قابلیت دسترسی بالا*

از وب سایت‌های تجارت الکترونیکی انتظار می‌رود که هفت روز هفته و آن هم بیست و چهار ساعت قابل دسترس باشند. آنها نباید هرگز غیرفعال باشند بنابراین باید حداقل مدت زمان غیرقابل دسترس بودن را داشته باشند (احتمالا چند دقیقه در سال).

§ *مقیاس‌پذیری*

با توجه به اینکه یک وب سایت در مورد محبوبیت خود در حال رشد است بنابراین قابلیت یا ظرفیت پردازش آن نیز باید همراه با رشد محبوبیت آن افزایش یابد تا اینکه اولا بتواند مقدار داده قابل مدیریت در دسترس سیستم را گسترش داده و همچنین سطح قابل قبول از پاسخ‌دهی سرویس به مشتریان را تامین کند.

§ *امنیت*

کاربران باید مطمئن شوند که هر گونه داده‌های مهم و حساس که از طریق وب ارسال می‌کنند از دسترس هکرها در امان است (مخصوصا آنهایی که جاسوسی[2] را انجام می‌دهند). گردانندگان[3] وب

[1] Tension
[2] Snooping
[3] Operators

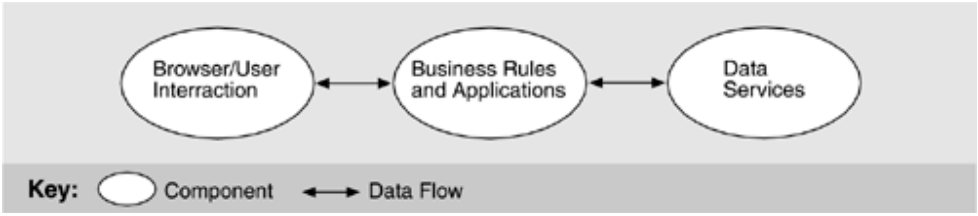
سایت‌ها باید مطمئن شوند که سیستم آنها از نظر حمله هکرها ایمن است (تغییر داده‌ها یا بارگیری غیرمجاز آنها، وارد کردن داده‌های غیر معتبر از طریق ارسال انبوه آنها از طریق درخواست‌ها، متوقف کردن سرور و غیره).

§ *قابلیت تغییرپذیری*

وب سایت‌های تجارت الکترونیکی مدام در حال تغییر هستند بنابراین محتوای آنها باید بتوانند به راحتی تغییر پیدا کنند.

راه‌حل معمارانه برای این نیازمندیها به جای معماری نرم‌افزار بیشتر در ارتباط با معمای سیستم است. زیرا مولفه‌هایی که سیستم را محبوب می‌کنند از بنگاه‌های تجاری وارد می‌شوند (سرویس‌دهنده‌ها و سرویس‌گیرنده‌های وب، پایگاه‌داده‌ها، سرویس‌دهنده‌های امنیتی[1]، سرویس‌دهنده‌های برنامه‌های کاربردی[2]، سرویس‌دهنده‌های پراکسی ، سرویس‌دهنده‌های تراکنش‌ها[3] و غیره).

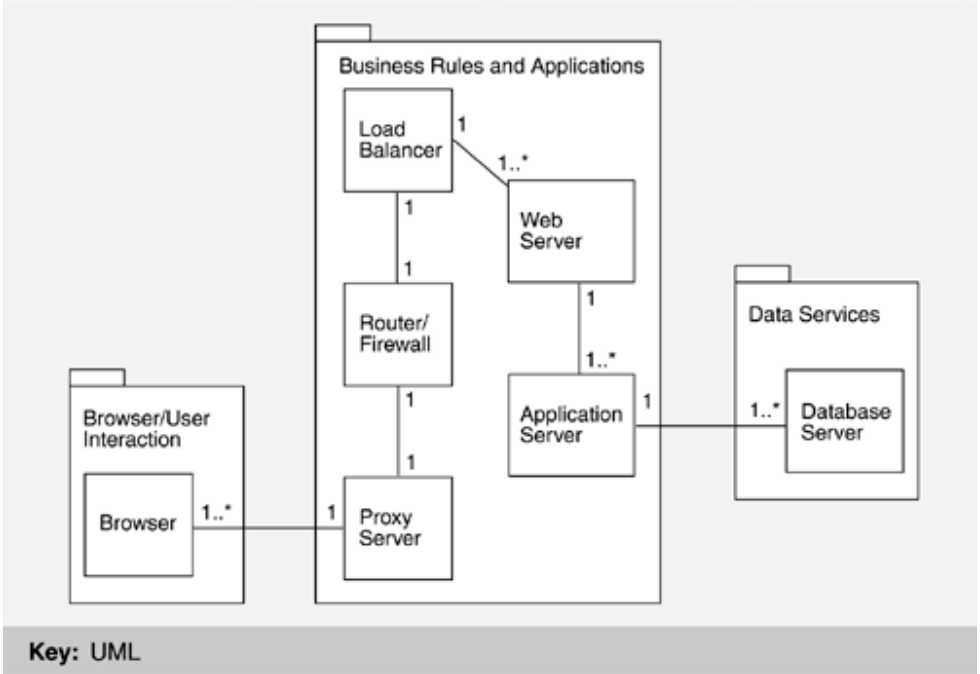
یک معماری مرجع برای سیستمهای امروزی تجارت الکترونیکی در شکل ۱۳-۶ نشان داده شده است. عملکرد تعامل کاربر معمولا به وسیله یک مرورگر برآورده می‌شود. قوانین حرفه و عملکردهای کاربردی معمولا از طریق سرویس‌دهندگان تراکنش و برنامه‌های کاربردی برآورده می‌شوند. لایه سرویس‌دهنده داده معمولا از طریق پایگاه‌داده‌های نوین امروزی برآورده می‌شوند. با این حال ارتباط با سیستمها و پایگاه‌داده‌های موروثی هنوز هم به صورت معمول انجام می‌شود. این نوع طرح معمولا به عنوان معماری ***n*** لایه ارجاع می‌شود که در اینجا ***n***=۳ است (لایه بخشی از عملکرد است که می‌تواند به یک ماشین فیزیکی مجزا تخصیص داده شود).



شکل ۱۳-۶: یک معماری مرجع برای تجارت الکترونیکی

[1] Security servers
[2] Application servers
[3] Transaction servers

پیاده‌سازی عمومی از معماری سیستم تجارت الکترونیکی شامل تعدادی لایه است که هر کدام متشکل از گروهی منسجم از نرم‌افزار و سخت‌افزار است (معمولا مولفه‌های تجاری خاص). یک چنین پیکربندی در شکل ۱۳-۷ به تصویر کشیده شده است و در آن چگونگی تخصیص نرم‌افزار به سخت افزار نشان داده شده است.



شکل ۱۳-۷: یک سیستم تجارت الکترونیکی عمومی

عنوان بسته‌های اصلی در شکل ۱۳-۷ با عناوین مشخص شده در شکل ۱۳-۶ یکسان است که این موضوع بیان کننده این است که یک عملکرد منفرد در مدل مرجع می‌تواند به چندین لایه در معماری تجارت الکترونکی نگاشت داده شود. دو بخش از شکل ۱۳-۵ در شکل ۱۳-۷ به عنوان مولفه‌های اصلی در نظر گرفته شده‌اند (یعنی مرورگرها به عنوان سرویس گیرنده‌ها و سرویس‌دهنده‌های وب هم به عنوان سرویس‌دهنده‌ها در نظر گرفته شده است). مرورگرهای وب و سرویس‌دهندگان وب بازتاب‌دهنده تکامل درجهت سیستم مبتنی بر مولفه هستند که در آن ساختار مولفه‌های داخلی خیلی مهم نیستند.

حال در ادامه هر یک از مولفه‌های مطرح شده در شکل ۱۳-۷ را به همراه خصوصیات کیفی که هر کدام از آنها کمک می‌کنند به آنها دست پیدا شود، تشریح خواهد شد.

۱۳-۴-۱- مرورگرهای وب برای قابلیت تغییرپذیری

هر کاربر نهایی از طریق تعامل با یک مرورگر درخواست اطلاعات را انجام می دهد. مرورگرهای امروزی از قابلیت تغییر واسط کاربری در روشهای مختلف حمایت می‌کنند. واسط کاربری که مرورگر از آن حمایت می‌کند ثابت نیست اما آن از طریق ***HTML*** مشخص می‌شود. امروزه فناوری‌های زیاد دیگری برای ایجاد واسط‌های کاربری پیچیده وجود دارد. ***XML, Flash, ActiveX*** و ***Java applets*** که فقط یک تعداد از روشهایی هستند که به وسیله آنها الگوهای استاندارد وب (گرافیک و غیره) گسترش می‌یابند تا واسط‌های تعاملی کاملا قابل برنامه‌نویسی از طریق مرورگرها فراهم شود.

۱۳-۴-۲- ***HTTPS*** برای امنیت

وقتی کاربر درخواستی را صادر کرد آن درخواست باید به وب سایت مقصد منتقل شود. این انتقال می‌تواند از طریق ***HTTP*** انجام شود و یا برای اطلاعات حساس مثل شماره حساب یا شماره شناسایی می‌توان از ***HTTPS*** استفاده کرد. ***HTTPS*** از ***SSL***[1] به عنوان یک زیرقراردادی[2] استفاده می‌کند که ***HTTP*** بر روی آن عمل می‌کند. برای این کار از گذرگاه دیگری برای درخواست سرویس‌های ***TCP/IP*** در یک قالب رمزگذاری شده استفاده می‌شود (گذرگاه ۴۴۳ به جای گذرگاه استاندارد ۸۰ که توسط ***HTTP*** استفاده می‌شود). ***SSL*** از جفت کلید عمومی/خصوصی ۱۲۸ بیت برای رمزگذاررری داده‌ها استفاده می‌کند. این سطح از رمزگذاری برای تبادل مقدار کم اطلات تجاری در تراکنش‌های کوتاه کافی است.

۱۳-۴-۳- سرویس‌دهنده‌های پراکسی برای کارایی

درخواست‌ها از مرورگرها می‌توانند ابتدا به سرویس‌دهنده پراکسی منتقل شود که دلیل وجود آنها بهبود کارایی سیستمهای مبتنی بر وب است. این سرویس‌دهنده‌ها صفحه‌های وبی را که به صورت مکرر درخواست شده‌اند را در حافظه داخلی خود نهان‌سازی می‌کند تا کاربران بتوانند آنها را بدون دسترسی به وب سایت اصلی دریافت کنند (نهان‌سازی از تاکتیک "چند نسخه کردن"[3] استفاده می‌کند). آنها معمولا در نزدیکی کاربر هستند و اغلب هم در شبکه‌ای که کاربر متعلق به آن است، قرار می‌گیرند. بنابراین مقدار زیادی از منابع محاسباتی و

[1] Secure Sockets Layer
[2] Subprotocol
[3] Multi copies

ارتباطی را حفظ می‌کنند. سرویس‌دهندگان پراکسی همچنین توسط شرکت‌هایی استفاده می‌شود که می‌خواهند کارمندان خود را محدود کنند که به یک سری از سایت‌ها دسترسی نداشته باشند. در این حالت سرویس‌دهنده پراکسی به نوعی کاری شبیه دیوار آتش را انجام می‌دهد.

۱۳-۴-۴- مسیریاب‌ها و دیوارهای آتشی برای امنیت

درخواست بعد از اینکه از مرورگر (یا سرویس‌دهنده پراکسی) صادر شد به مسیریاب می‌رسد که معمولا بر روی شبکه فراهم کننده تجارت الکترونیکی[1] مستقر است و آن نیز می‌تواند شامل دیوار آتشی برای امنیت باشد. مسیریاب می‌تواند یک ***NAT***[2] را پیاده‌سازی کند که یک آدرس ***IP*** قابل مشاهده خارجی را به آدرس ***IP*** داخلی تبدیل می‌کند.

هدف از دیوار آتشی آن است که از دسترسی‌های غیر مجاز به اطلاعات از طریق هر عامل مستقر در دنیای خارج جلوگیری کرد. برای مثال "دسترسی محدود" یکی از تاکتیک‌های استفاده شده بدین منظور است. انواع مختلفی از دیوارهای آتشی وجود دارد ولی بیشترین آنها مبتنی بر فیلتر کردن بسته‌ها و پراکسی‌های کاربردی[3] هستند. روش فیلتر کردن، بسته‌ها سرآیند[4] کلیه بسته‌های ***TCP/IP*** رسیده را بررسی می‌کند و اگر رفتار غیرمناسب در آنها مشاهده کند بسته مورد نظر را نادیده می‌کند. دیوارهای آتشی مبتنی بر فیلتر کردن بسته‌ها، مناسب برای ارتباطات مبتنی بر وب هستند زیرا آنها هر بسته را به صورت مجزا بررسی می‌کند بنابراین نیازی نیست که یک تاریخچه‌ای از ارتباطات قبلی نگهداری شود. دیوارهای آتشی مبتنی بر پراکسی کاربردی، همان طور که از نامشان مشخص است برای برنامه‌های کاربردی خاص است. آنها معمولا قرارداد کاربری را درک می‌کنند و لذا می‌توانند ترافیک را مبتنی بر الگوهای شناخته شده رفتار فیلتر کنند. برای مثال یک پراکسی کاربردی یک پاسخ ***HTTP*** را قبول نمی‌کند مگر اینکه قبلا یک درخواست ***HTTP*** به یک سایت از طریق آن صادر شده باشد. این نوع دیوارهای آتشی از دیوارهای آتشی مبتنی بر فیلتر کردن بسته‌ها کندتر هستند زیرا آنها مبتنی بر نگهداری مقدار مشخص از اطلاعات تاریخچه‌ای هستند و لذا پردازش این اطلاعات زمان‌بر خواهد بود (در واقع الگوریتم بررسی پیچیده است).

---

[1] E-commerce provider's network
[2] Network Address Translation
[3] packet filters and application proxies
[4] Header

۱۳-۴-۵- تعادل بار برای کارایی، مقیاس‌پذیری و قابلیت دسترسی

مولفه متعادل‌کننده بار یکی از بخش‌های اصلی از هر وب سایت مهم تجارت الکترونیکی است زیرا از کارایی، مقیاس‌پذیری و قابلیت دسترسی حمایت می‌کند. وظیفه متعادل‌کننده بار توزیع درخواست‌های رسیده (درخواست‌های ***HTTP*** و ***HTTPS***) در میان کامپیوترهایی است که سرویس‌دهنده وب را اجرا می‌کنند. متعادل‌کننده بار به راحتی می‌تواند درخواست رسیده را به یک کامپیوتر ارجاع دهد یا می‌تواند به سرویس‌گیرنده پاسخی ارسال کند و راهنمایی‌های لازم در مورد ارجاع درخواست به سرویس‌دهنده دیگر را به آن بدهد. این نوع ارجاع مجدد به سرویس‌دهنده دیگر کاملا از دید کاربر نهایی پنهان است و باعث به وجود آمدن مجموعه ارتباطات اضافی می‌شود. در انتخاب اینکه کدام کامپیوتر باید مورد ارجاع قرار بگیرد متعادل‌کننده بار می‌تواند از یک روش نوبت گردشی استفاده کند یا اینکه انتخاب می‌تواند مبتنی بر قدرت پردازش‌های هر یک از کامپیوترها باشد. زیرا متعادل‌کننده بار به عنوان پراکسی برای مجموعه‌ای از کامپیوترها عمل می‌کند و می‌توان به راحتی کامپیوتر جدیدی را به این مجموعه بدون تغییر واسط خارجی انجام داد. در این روش متعادل‌کننده بار از کارایی و مقیاس‌پذیری که به توسعه افقی معروف است، حمایت می‌کند. همچنین تعادل‌کننده بار می‌تواند بر فعال بودن هر یک از کامپیوترها نظارت کند و اگر یکی از آنها غیرفعال شود درخواست‌ها را به کامپیوترهای دیگر ارجاع دهد. در این حالت متعادل‌کننده بار از قابلیت دسترسی حمایت می‌کند.

۱۳-۴-۶- سرویس‌دهنده وب برای کارایی

بعد از متعادل‌کننده بار درخواست ***HTTP*** و ***HTTPS*** به سرویس‌دهنده وب می‌رسد. این سرویس‌دهنده‌های اولیه مانند آنچه که در شکل ۱۳-۵ نشان داده شده است فقط ار یک نخ[1] برای پردازش استفاده می‌کنند. نسخه‌های امروزی این نوع سرویس‌دهنده‌ها، چندنخی هستند و از مجموعه‌ای از نخ‌ها استفاده می‌کنند که هر کدام می‌تواند یک درخواست را مدیریت کرد و به آن پاسخ دهد. یک سرویس‌دهنده چند نخی خیلی کم در معرض آسیب قرار گرفتن به صورت گلوگاه است آن هم زمانی که تعداد زیادی از درخواست‌های ***HTTP*** و

[1] Single thread

HTTPS به سرویس‌دهنده می‌رسد. دلیل این امر آن است که نخ‌های دیگری هنوز قابل دسترس هستند که می‌توانند به درخواست‌های رسیده جواب دهند (این یک نوع تاکتیک کارایی "معرفی همزمانی" است).

توسعه عمودی می‌تواند با جایگزین کردن سرویس‌دهنده وب موجود با ماشین‌های قدرتمند که از همزمانی بیشتر نخ‌ها حمایت می‌کنند، انجام می‌شود. به محض تحلیل درخواست، سرویس‌دهنده وب آن را به یک سرویس‌دهنده برنامه کاربردی ارسال می‌کند تا به آن پاسخ داده شود و معمول نیز این سرویس‌ها از پایگاه‌داده استفاده می کنند.

۱۳-۴-۷- سرویس‌دهنده‌های برنامه کاربردی برای قابلیت تغییرپذیری، کارایی و مقیاس‌پذیری

سرویس‌دهنده برنامه کاربردی یک عبارت گسترده برای کلاسی از کاربردها است که در لایه میانی معماری ***n*** لایه اجرا می‌شوند. این سرویس‌دهنده‌ها اهداف حرفه و اتصال را پیاده‌سازی می‌کنند که مشخص می‌کند چگونه سرویس‌دهنده‌ها و سرویس‌گیرنده‌ها با یکدیگر ارتباط برقرار کنند. گرایش به سمت سرویس‌دهنده‌های برنامه کاربردی اجازه داده است که حجم زیادی از عملکردها که در سبک قدیمی در لایه سرویس‌گیرنده بود به لایه میانی منتقل شوند. همچنین آنها این اجازه را داده‌اند که پایگاه‌داده‌ها بدون نگرانی در مورد دقت اینکه چگونه داده‌ها استفاده می‌شوند آنها را در حافظه ذخیره، بازیابی و تحلیل کنند.

در سطح پائین، سرویس‌دهنده‌های برنامه کاربری معمولا سرویس‌دهنده زمان اجرا و محیط توسعه یکپارچه[1] ارائه می‌کنند. محیط‌های توسعه یکپارچه از مدلهای برنامه‌نویسی مانند ***COM, CORBA*** یا ***J2EE*** حمایت می‌کنند. بسیاری از سرویس‌دهنده‌های برنامه کاربردی از یک مجموعه از سرویس‌های مشترک برای توسعه سریع برنامه‌های کابردی تجارت الکترونیکی و حرفه استفاده می‌کند. در سطح بالایی نیز بر حسب هزینه، پیچیدگی و عملکرد ناظر و پردازشگر تراکنش‌ها قرار دارند. ناظرها و پردازنده‌های تراکنش با پایگاه‌داده ارتباط برقرار می‌کنند و وظایفی مانند تراکنش‌های توزیع شده، صف‌بندی، یکپارچگی تراکنش و تعادل بارکاری را مدیریت می‌کنند.

[1] Integrated Development Environment (IDE)

۱۳-۴-۸- پایگاه‌داده‌ها برای کارایی، مقیاس‌پذیری و قابلیت دسترسی

نهایتا، درخواست برای سرویس به پایگاه‌داده می‌رسد و آن نیز درخواست را به دستورات اضافه، تغییر یا بازیابی اطلاعات تبدیل می‌کند. معماری پایگاه‌داده‌های امروزی بسیاری از خصوصیات موجودیت سیستم تجارت الکترونیکی نشان داده شده در شکل ۱۳-۷ را به اشتراک می‌گذارند. آنها به صورت مداوم از تکرار داخلی برای کارایی، مقیاس‌پذیری و قابلیت دسترسی بالا استفاده می‌کنند و همچنین از نهان‌سازی برای کارایی سریع‌تر کمک می‌گیرند.

۱۳-۵- دستیابی به خصوصیات کیفی

عناصر تشریح شده در بخش قبلی به سیستم تجارت الکترونیکی مبتنی بر وب اجازه می‌دهند که به اهداف خصوصیات کیفی مهم خود از قبیل امنیت، قابلیت دسترسی بالا، قابلیت تغییرپذیری، مقیاس‌پذیری و کارایی بالا دست پیدا کنند. نحوه چگونگی دستیابی به آنها در جدول ۱۳-۳ نشان داده شده است.

جدول ۱۳-۳: چونگی دستیابی معماری تجارت الکترونیکی مبتنی بر وب به اهداف کیفی خودش

هدف	چگونه انجام می‌شود	تاکتیک‌های مورد استفاده
کارایی بالا	تعادل بار، ترجمه آدرس شبکه، سرویس‌دهنده‌های پراکسی	معرفی همزمانی، افزایش منابع، چند نسخه‌گی
قابلیت دسترسی بالا	- افزونگی در پردازنده، شبکه پایگاه‌داده و نرم‌افزار - تعادل بار	افزونگی فعال، تراکنش‌ها، معرفی همزمانی
مقیاس‌پذیری	- اجازه توسعه عمودی و افقی - تعادل بار	تجرید سرویس‌های مشترک، الصاق به قراردادهای تعریف شده، معرفی همزمانی
امنیت	- دیوار آتش - رمزگذاری کلید عمومی/خصوصی در سرتاسر شبکه عمومی	دسترسی محدود، یکپارچگی، نمایش محدود
قابلیت تغییرپذیری	جداسازی عملکردهای مرورگر، پایگاه‌داده و منطق حرفه و قرار دادن آنها در لایه‌های مختلف	تجرید سرویس‌های مشترک، انسجام معنایی، واسطه، ماندگاری واسط

۱۳-۶- چرخه حرفه معماری امروزی

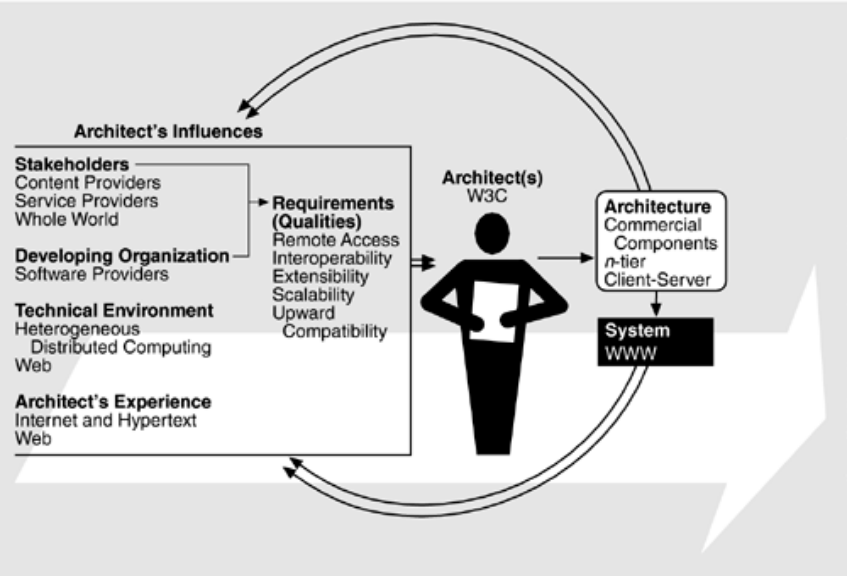
اگر نگاهی به وضعیت جاری وب بعد از اینکه چندین چرخه در چرخه حرفه معماری انجام شد، انداخته شود تعدادی پدیده[1] مشاهده می‌شود که عبارتند از:

- انواع سازمانها محیط‌های فنی فراهم می‌کنند. این سازمانها می‌توانند به فراهم کننده سرویس و فراهم کننده محتوا تقسیم شوند. فراهم کنندگان سرویس نرم‌افزارهایی را تولید می‌کنند که وب را تولید می‌کند (مانند مرورگرها، سرویس‌دهنده‌ها، پایگاه‌داده‌ها، سرویس‌دهندگان برنامه کاربردی، فناوریهای امنیتی از قبیل دیوار آتش، سرویس‌دهندگان تراکنش، شبکه‌ها و مسیریاب‌ها). فراهم کنندگان محتوا نیز داده‌های مورد نیاز وب را تولید می کنند. در هر دوی این سازمانها رقابتی شدیدی وجود دارد.
- تعدادی پروژه‌های متن باز علاوه بر ***W3C*** به وجود آمده‌اند که خواهان نشان دادن برتری خود در توسعه وب هستند از جمله آنها می‌تواند پروژه ***Apache*** را ذکر کرد.
- موسسه ***CERN*** هیچ نقش خاصی در توسعه وب نداشته است.
- زبانهای مربوط به وب (مخصوصا جاوا) در حال تغیر روش توسعه و تحویل عملکردها بر روی وب هستند.
- ضرورت وب به عنوان یک محیط توسعه توزیع‌شده موجب به وجود آمدن چند سازمان و تولید کننده جدید شده است. برای مثال، ***UDDI***[2] ثبت مبتنی بر وب توزیع شده برای سرویس‌های وب را فراهم می‌کند. این سرویس‌ها می تواند به عنوان بلوک‌های سازنده برنامه‌های کاربردی مبتنی بر وب استفاده شود.

شکل ۱۳-۸ چرخه حرفه معماری امروزی را نشان می‌دهد. مشتریان، فراهم کنندگان نرم‌افزارهای مرورگر و سرویس‌دهنده و همچنین فراهم کنندگان سرویس و محتوا هستند. کاربران نهایی مردمان دنیا هستند. نقش معمار به وسیله ***W3C*** و دیگر سازمانها مانند ***UDDI*** ، پروژه ***Apache*** و چندین سازمان دیگر (***Sun, Microsoft, AOL/Netscape***) تامین شده است. مابقی بخش‌های چرخه حرفه معمای هیچ تغییری پیدا نکرده

[1] Phenomena
[2] Universal Description, Discovery, and Integration

است به استثناء اینکه هم اکنون وب نیز به عنوان محیط فنی در نظر گرفته شده است و نیازمندیهای جدیدی را به خصوصیات کیفی اضافه می‌کند



شکل ۱۳-۸: چرخه حرفه معماری امروزی برای وب

فهرست مطالب

فصل چهاردهم

خط تولید نرم‌افزار: استفاده مجدد از دارایی‌های معماری

در این فصل در ارتباط با استفاده مجدد صریح و برنامه‌ریزی شده از معماری نرم‌افزار (همچنین سایر دارایی‌های) در میان خانواده‌ای از سیستم مرتبط به هم بحث خواهد شد. بدین منظور ابتدا نحوه عملکرد خط تولید بررسی شده بعد نحوه حیطه‌بندی و انواع معماری‌های خط تولید تشریح خواهد شد و نهایتا نیز مشکلات خط تولید بیان خواهد شد.

۱۴-۱- مقدمه

معماری نرم‌افزار سرمایه‌گذاری قابل توجهی در زمان و هزینه ارائه می‌کند لذا کاملا طبیعی است که خواهان بیشترین بازگشت این سرمایه‌گذاری از طریق استفاده مجدد معماری در سرتاسر سیستم‌های چندگانه (مختلف) باشیم. سازمانهایی که به بلوغ معماری رسیده‌اند تمایل دارند تا معماری خودشان را به عنوان یک دارایی ارزشمند و زائیده خلاقیت خود در نظر بگیرند و به دنبال راهکارهایی هستند که در آن دارایی مورد نظرشان نقش کلیدی در تولید محصولات پردرآمد و کم هزینه داشته باشد (هر دو این خصوصیات از طریق استفاده مجدد معماری قابل دستیابی هستند).

زمانی که سازمانی در حال تولید چندین سیستم مشابه است و از معماری یکسانی استفاده می‌کند (عناصر مرتبط با آن معماری نیز یکسان هستند) آن شرکت از سود و منفعت چشم‌گیری بهره می‌برد که شامل کاهش هزینه ایجاد و زمان عرضه به بازار است. این نوع عملکرد خط تولید نرم‌افزار نامیده می‌شود که به صورت زیر تعریف می‌شود:

مجموعه‌ای از سیستم‌های شدیدا نرم‌افزاری[1] (مقصود سیستمهایی است که نرم‌افزار در آنها نقش کلیدی را ایفاء می‌کند و کاملا وابسته به نرم‌افزار هستند مانند سیستم بانکداری الکترونیکی) که دارای یک سری ویژگیهای عمومی و مدیریت شده مشترک بوده، نیازهای مشخصی از بازار یا ماموریت[2] خاصی را برآورده می‌نمایند و براساس مجموعه مشترکی از دارایی‌های اصلی[3] به صورت تجویزی توسعه داده می‌شوند.

در واقع دید خط تولید نرم‌افزار، مجموعه‌ای از دارایی‌های قابل استفاده مجدد است که دربرگیرنده معماری پایه و عناصر مشترک (و احتمالا قابل تنظیم[4]) است. خط تولید نرم‌افزار همچنین شامل طراحی‌ها و مستندات، راهنمای کاربران[5]، فرآورده‌های مدیریت پروژه از قبیل بودجه، زمانبندی، طرح‌های آزمایش نرم‌افزار و داده‌های آزمایشی است. موفقیت در دستیابی به این دید شدیدا به ایجاد حیطه[6] صحیح برای محصول خط تولید وابسته است. وقتی خط تولیدی با موفقیت ایجاد شده باشد هر دارایی‌های قابل استفاده آن به عنوان یک پایه دارایی

[1] Software-Intensive
[2] Mission
[3] Core assets
[4] Tailorable
[5] Users manual
[6] Scope

اصلی[1] ذخیره می‌شود دلیل آن نیز این است که پایه دارایی‌های اصلی می‌توانند در بیش از یک سیستم مورد استفاده واقع شوند و همچنین به دلیل قابلیت استفاده مجدد، هزینه استفاده از آنها ارزانتر از ایجاد یا ابداع است. به صورت ایده‌ال دارایی‌های اصلی همراه با نقاط تغییر[2]، طراحی می‌شوند. یعنی محلهای از پیش تعیین شده‌ای که آنها می‌توانند سریعا در آنجا به کار گرفته شوند (تنظیم شوند).

در خط تولید موفق، فرآیند ساخت سیستم تبدیل می‌شود به دسترسی به دارایی‌ها و تجمیع آنها در روشی که برای سیستم در دست تولید مورد نیاز است. همچنین در خط تولید یکپارچه‌سازی و آزمایش به جای طراحی و کدنویسی مورد استفاده قرار می‌گیرد که در روش‌های غیر مبتنی بر خط تولید، فعالیت‌های حاکم بر تولید نرم‌افزار بودند. البته خط تولید چیز جدیدی در صنعت[3] نیست. بسیاری از مورخین این مفهوم را به آقای ***Eli Whitney*** نسبت می‌دهند که در اوایل دهه ۱۸۰۰ از قطعات قابل تعویض برای ساخت اسلحه استفاده می‌کرد. با این وجود مثالهای دیگری نیز از شرکت‌هایی وجود دارند که در حال حاضر با این روند کار می‌کنند. شرکت‌های بوئینگ، فورد، دل، مک‌دانالد نمونه‌هایی از این نوع شرکت‌ها هستند. هر شرکت از دارایی‌های خود به شیوه متفاوت استفاده می‌کند. به عنوان مثال شرکت بوئینگ هواپیماهای ۷۵۷ و ۷۵۶ را به گونه‌ای با یکدیگر توسعه داده است که ۶۰ درصد قطعات استفاده شده در این دو هواپیما با یکدیگر هم پوشانی دارند.

خط تولید نرم‌افزار مبتنی بر خصوصیات مشترک محصولات داخلی[4] است که مفهومی جدید و قابل رشد را در مهندسی نرم‌افزار ارائه می‌کند. هر مشتری نیازمندیهای خاص خود را دارد که این امر منجر می‌شود که در قطعات تولید شده انعطاف‌پذیری لازم وجود داشته باشد. همچنین خط تولید نرم‌افزار فرآیند ایجاد سیستمهایی که به صورت خاص برای مشتریان ویژه یا مشتریان گروهی[5] تولید می‌شود را آسان می‌کند. بنابراین ایجاد خط تولید موفق متکی بر راهبرد هماهنگی است که دربرگیرنده مهندسی نرم‌افزار، مدیریت فنی[6] و مدیریت سازمانی است. بهبود در هزینه، زمان عرضه به بازار و بهره‌وری که با موفقیت خط تولید نرم‌افزار حاصل می‌شود می‌تواند خیلی ارزشمند و مهیج باشد. برای مثال:

---

[1] Core asset base
[2] Variation points
[3] Manufacturing
[4] Inter-product commonality
[5] Customer groups
[6] Technical management

- شرکت ***Nokia*** به دلیل استفاده از خط تولید این توانایی را کسب کرده است که در یک سال ۲۵ تا ۳۰ نوع گوشی متفاوت را روانه بازار کند (قبلا فقط چهار نوع گوشی در سال به بازار عرضه می‌کرد).
- شرکت ***Cummins*** زمان مورد نیاز برای تولید نرم‌افزار برای موتور دیزل را از یک سال به یک هفته کاهش داد.
- شرکت ***Motorola*** بهره‌وری ۴۰۰ درصد را در خانواده پیجرهای یک طرفه کسب کرد.
- شرکت ***Hewlett-Packard*** کاهش هفت برابری در زمان عرضه محصولات خود به بازار و افزایش بهره‌وری شش برابری در یک خانواده از چاپگرها بدست آورده است.
- اداره ***U.S. National Reconnaissance Office*** اعلام کرده است که از طریق بکارگیری خط تولید در یک خانواده از سیستمهای کنترل ماهواره به کاهش ۱۰ درصدی در زمان تولید و کاهش ۹۰ درصدی در خطاهای حاصله دست یافته است.

۱۴-۲- نحوه عملکرد خط تولید نرم‌افزار

ماهیت خط تولید نرم‌افزار استفاده مجدد راهبردی[1] و نظام‌مند[2] از دارایی‌ها در جهت تولید خانواده‌ای از محصولات است. عاملی که باعث موفقیت خطوط تولید از منظر فروشندگان و توسعه‌دهندگان می‌شود آن است که خصوصیات مشترک (دارایی‌ها) به اشتراک گذاشته شده از طریق محصولات مختلف می‌توانند در روند استفاده مجدد در جهت دستیابی به محصول باصرفه از نظر اقتصادی، استفاده شوند. پتانسیل برای استفاده مجدد بسیار گسترده و زیاد است که بعضی از آنها به شرح زیر هستند:

- *نیازمندیها*

بسیاری از نیازمندیها با نیازمندیهای سیستمهای اولیه مشترک هستند و بنابراین می‌توانند مجددا استفاده شوند. در واقع تحلیل نیازمندها به نوعی کم شده یا از بین می‌رود.

- *طراحی معماری*

معماری برای سیستم نرم‌افزاری یک سرمایه‌گذاری عظیم زمانی از جانب مهندسین خبره سازمان است. همان طور که از قبل بیان شد اهداف کیفی برای سیستم (کارایی، قابلیت اطمینان، قابلیت تغییرپذیری و

[1] Strategic
[2] Disciplined

غیره) در معماری یا لحاظ شده یا در آن نادیده گرفته می‌شوند و اگر معماری اشتباه باشد سیستم نمی‌تواند درست تولید شود. بنابراین برای یک محصول جدید این مرحله بسیار مهم قبلا انجام شده است و نیازی نیست که دوباره تکرار شود.

- *عناصر*

عناصر نرم‌افزاری در سرتاسر محصولات منفرد قابل بکارگیری هستند. فرآیند ایجاد مولفه قابل استفاده مجدد دربرگیرنده طراحی واسط عناصر، مستندات، طرحهای آزمایش و رویه‌ها و هر مدل استفاده شده برای پیش‌بینی یا اندازه‌گیری رفتارهای عنصر است. یک مجموعه از عناصر قابل استفاده مجدد، واسط کاربری سیستم است که مجموعه بزرگ و حیاتی از تصمیمات طراحی را نشان می‌دهد.

- *تحلیل و مدلسازی*

مدلهای کارایی، تحلیل قابلیت زمانبندی، پیامدهای سیستمهای توزیع شده، تخصیص فرآیندها به پردازنده‌ها، طرح تحمل‌پذیری خطا و سیاست‌های محلی شبکه همگی از یک محصول به محصول دیگر منتقل می‌شوند. بنابراین به نوعی مشکلات مربوط به این نوع دغدغه‌ها از بین می‌رود.

- *آزمایش*

طرح‌های آزمایش، فرآیندهای آزمایش، موارد آزمایش، داده‌های آزمایش، مشکلات آزمایش و مسیرهای ارتباطی موردنیاز برای گزارش و اصلاح خطا قبلا مشخص شده‌اند بنابراین می‌توانند مجددا استفاده شوند.

- *برنامه‌ریزی پروژه*[1]

زمانبندی و بودجه از مواردی هستند که می‌توانند از قبل پیش‌بینی شوند (در این زمینه تجربه یکی از شاخص‌ها تاثیرگذار است). ساختارهای تقسیم کار دیگر نیاز نیست که هر بار انجام شوند و همچنین تیمها، اندازه تیم و ترکیب تیم به آسانی مشخص می‌شوند.

- *ابزارها، روش‌ها و فرآیندها*

رویه‌های کنترل پیکربندی، تسهیلات، برنامه‌ریزی‌های مستندات و فرآیندهای مشخص شده، محیط ابزارها، رویه‌های توزیع و ایجاد سیستم، استاندارد کدنویسی و بسیاری از فعالیت‌های مهندسی روزانه

[1] Project planning

می‌توانند از محصولی به محصول دیگر منتقل شوند. همچنین فرآیند کلی توسعه نرم‌افزار از قبل موجود بوده و استفاده نیز شده است.

- *افراد*

به دلیل خصوصیات مشترک برنامه‌های کاربردی پرسنل می‌تواند به راحتی در میان پروژه (در صورت نیاز) منتقل شود. همچنین اینکه تجارب پرسنل در کل خط تولید قابل استفاده هستند.

- *سیستمهای نظیر*[1]

محصولات مستقر شده به عنوان نمونه‌های نمایشی کیفیت بالا و مدلهای مهندسی کیفیت بالا از منظر کارایی، امنیت[2]، ایمنی[3] و قابلیت اطمینان[4] ارائه می‌شوند.

- *حذف خطا*

خط تولید به دلیل اینکه در آن هر سیستم از مزایای حذف خطاها در سیستم قبلی بهره می‌برد باعث افزایش کیفیت می‌شود. اطمینان توسعه‌دهنده و مشتری نیز در هر نمونه جدید افزایش پیدا می‌کند. در سیستمهای بسیار پیچیده مسائلی که باید مورد توجه قرار گرفته و حل شوند فقط یکبار برای کل مجموعه محصولات خانواده حل شده و بارها از آن استفاده می‌شود (کارایی، توزیع، قابلیت اطمینان و دیگر پیامدهای مهندسی، نمونه‌های از این نوع مسائل هستند).

خط تولید نرم‌افزار متکی بر قابلیت استفاده مجدد است اما قابلیت استفاده مجدد سابقه طولانی ولی نه خیلی درخشان در مهندسی نرم‌افزار دارد زیرا همواره این نوید را داده که نتیجه (خروجی)، همیشه عالی خواهد بود. یک دلیل برای این شکست آن است که تا به امروز قابلیت استفاده مجدد بر روی این ایده پیش‌بینی شده است که "اگر شما آن را بسازید آنها به وجود خواهد آمد". در واقع کتابخانه قابل استفاده مجدد از پروژه‌های قبلی ایجاد شده است و انتظار می‌رود که توسعه‌دهندگان قبل از کدنویسی، عنصر جدید را در کتابخانه بررسی کنند تا دریابند که آیا چنین عنصری وجود دارد یا نه؟ تقریبا کلیه شرایط بر علیه این مدل هستند. اگر کتابخانه زیاد پراکنده باشد توسعه‌دهنده نخواهد توانست عنصر مورد نظر خود را پیدا کند و در نتیجه جستجو را متوقف

[1] Examplar systems
[2] Security
[3] Safety
[4] Reliability

خواهد کرد (زیرا زمان زیادی باید صرف آن شود). اگر کتابخانه خیلی غنی باشد جستجوی آن بسیار دشوار خواهد شد. اگر عناصر بسیار کوچک باشند بنابراین ایجاد آنها سریعتر و سهل‌تر از آن خواهد بود که مولفه مورد نظر را جستجو کرده و تغییرات مورد نیاز را بر آن اعمال کرد. اگر عناصر زیاد بزرگ باشند آن بسیار دشوار خواهد بود که به صورت دقیق مشخص کرد که آنها چگونه عمل می‌کنند. در هر صورت توسعه‌دهنده نخواهد توانست به صورت مطمئن تشخیص دهد که هر مولفه چه عملکردی دارد، قابلیت اطمینان آن چقدر است یا اینکه تحت چه شرایطی آن آزمایش شده است. بنابراین تقریبا هیچ تطبیقی بین خصوصیات کیفی مورد نیاز برای برنامه کاربردی جدید و خصوصیاتی که به وسیله عناصر موجود در کتابخانه فراهم می‌شوند وجود نخواهد داشت. همچنین این امکان وجود دارد که عناصر برای یک مدل معماری متفاوتی نوشته شده باشند و توسعه‌دهنده از مدل معماری دیگر استفاده کند. به عبارت بهتر اگر بتوان مولفه‌ای یافت که درست عمل کرده و از خصوصیات کیفی مورد نیاز نیز به خوبی حمایت کند آن مولفه ممکن است نوع صحیحی از مولفه معماری مورد نظر نباشد (به عنوان مثال از قرارداد تعاملی صحیحی استفاده نکند یا اینکه با مدیریت کنترل خطای برنامه کاربردی جدید سازگاری نداشته باشد).

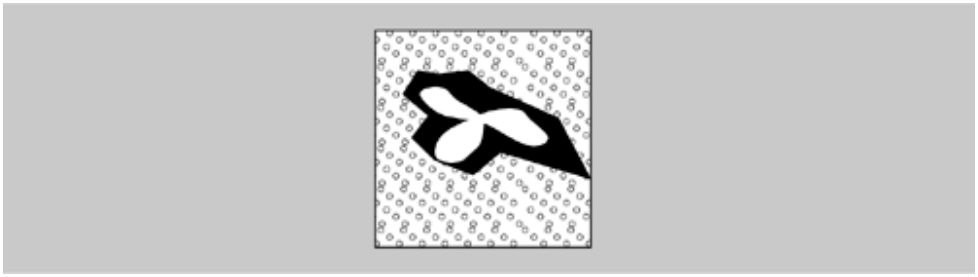
خط تولید نرم‌افزار قابلیت استفاده مجدد را برای یک حیطه کوچک و خاص فراهم می‌آورد. یعنی در خط تولید نرم‌افزار معماری تعریف شده است، عملکردها تعیین شده‌اند و خصوصیات کیفی نیز کاملا واضح مشخص شده‌اند. هیچ چیزی در کتابخانه قابل استفاده یا پایه دارایی اصلی قرار داده نمی‌شود (مگر اینکه در آن خط تولید مورد استفاده قرار بگیرد). در واقع خط تولید به جای وابستگی به استفاده مجدد خوش‌بینانه متکی بر راهبرد یا برنامه‌ریزی است.

۱۴-۳- حیطه‌بندی

حیطه خط تولید نرم‌افزار تعیین می‌کند چه سیستمهایی درون خط تولید و چه سیستمهایی بیرون آن قرار دارند. در واقع حیطه خط تولید مشخص کننده سیستمهایی هستند که سازمان خواهان آن است که آنها را به عنوان بخشی از خط تولید ایجاد کند و همچنین سیستمهایی که سازمان تمایلی به تولید آنها ندارد. تعریف حیطه خط تولید شبیه ترسیم یک شیء چنبری[1] مدل در فضای کلیه سیستمهای ممکن است. این موضوع در

[1] Doughnut

شکل ۱۴-۱ نشان داده شده است. بخش مرکز چنبر نشان دهنده سیستمهایی است که سازمان می‌تواند تولید کند زیرا آن سیستمها در حوزه قابلیت خط تولید قرار گرفته‌اند. سیستمهای خارج از چنبر، خارج از حیطه قرار دارند یعنی سیستمهایی که خط تولید به خوبی قادر به مدیریت آنها نیست. سیستمهای درون چنبر قابل مدیریت و کنترل توسط خط تولید هستند ولی بدین منظور باید کمی فعالیت اضافی انجام شود. برای درک بهتر در نظر بگیرید که در خط تولید برای سیستمهای مکانیره اداری زمانبند اتاق کنفرانس جزو سیستمهای خط تولید است اما شبیه‌ساز پرواز خارج از حیطه خط تولید است. یک موتور جستجوی داخلی خاص می‌تواند در حیطه آن قرار بگیرد به شرطی که در زمان معقول قابل تولید و همچنین از نظر راهبردی قابل استدلال باشد.



شکل ۱۴-۱: فضای کلیه سیستمهای ممکن به مناطقی در داخل حیطه (سفید) تقسیم شده است مناطق خارج از حیطه به صورت خالدار و مناطق مورد نیاز در حالات موردی با رنگ سیاه نشان داده شده‌اند.

حیطه بهترین پیش‌بینی را برای سازمان درباره محصولاتی که خواهان ایجاد آن در آینده است، ارائه می‌کند. ورودی برای فرآیند حیطه‌بندی از طریق برنامه‌ریزان راهبردی سازمان، کارکنان فروش یا بازاریابی[1]، تحلیل‌گران حوزه و خبرگان فناوری فراهم می‌شود. حیطه خط تولید فاکتور حیاتی در موفقیت خط تولید است. اگر حیطه خیلی محدود باشد تعداد اندکی از محصولات ضعیف در توجیه سرمایه‌گذاری تولید خواهد شد. اگر حیطه خیلی بزرگ و گسترده باشد تلاشهای لازم برای توسعه محصولات منفرد از دارایی‌های اصلی زیاد خواهد بود. حیطه می‌تواند در طول پایه‌گذاری خط تولید بهبود داده شود یا به صورت سریع با راهبرد خط تولید تطبیق پیدا کند.

مشکل تعریف حیطه پیدا کردن خصوصیات مشترک نیست (یک معمار خلاق می‌تواند به راحتی نقاط مشترک بین دو سیستم را پیدا کند) بلکه پیدا کردن خصوصیات مشترکی است که می‌تواند مورد استفاده واقع شود تا

[1] Marketing

هزینه تولید سیستمهایی که سازمان تمایل به تولید آنها دارد، کاهش یابد. در زمان تعریف حوزه، ملاحظات باید فراتر از سیستمهایی که خواهان تولید آن هستیم در نظر گرفته شود. وضعیت بازار و انواع تعاملات مشتریان می‌توانند در تعیین حیطه خط تولید کمک شایانی انجام دهند. برای مثال شرکت فیلیپس برای سیستمهای الکترونیکی تصویری خانگی یک خط تولید و برای ارتباطات تصویری دیجیتال یک خط تولید مجزا دارد. زیرا هر کدام از این محصولات بازار و مشتریان خاص خود را دارد. سیستمهای ویدیویی بازار خوبی دارد و مشتریان آن تمایل دارند که سیستمهای ویدئویی خیلی ساده را خریداری کند در حالی که مشتریان محصول دوم کسانی هستند که دانش زیادی در این حوزه دارند و بازار مصرف آنها نیز خیلی محدود است.

خط تولید با حیطه محدود فرصت‌هایی را برای ساخت ابزارهای خاص برای حمایت از مشخصات محصولات جدید فراهم می‌کند در حالی که خط تولید با حیطه گسترده تمایل به توسعه چارچوب یا مجموعه‌ای از سرویس‌ها دارد.

۱۴-۴- معماری‌های خط تولید

از کلیه دارایی‌های موجود در مخزن دارایی‌های اصلی، معماری نرم‌افزار مهمترین نقش را ایفا می‌کند. ماهیت ایجاد خط تولید نرم‌افزار موفق تمایز قائل شدن بین چیزهایی است که انتظار می‌رود در کلیه محصولات یک خانواده بدون تغییر باقی بماند و چیزهایی که انتظار می‌رود تغییر پیدا کنند. از آنجایی که کلیه معماریها تجریدهایی هستند که تعدادی نمونه‌ها[1] را پذیرش می‌کنند لذا معماری نرم‌افزار این قابلیت را دارد که دوگانگی در بین عناصر را مدیریت کند.

یکی از ارزشهای مفهومی[2] معماریها آن است که اجازه می‌دهند بر روی ضروریات طراحی در داخل پیاده‌سازیهای مختلف متمرکز شد. ماهیت طبیعی معماری بیان کننده این موضوع است که چه چیزهایی انتظار می‌رود بدون تغییر باقی بمانند و چه چیزهایی به عنوان موارد قابل تغییر در نظر گرفته می‌شوند (در یک خط تولید نرم‌افزار، معماری بیان کننده جنبه‌های غیرمتغیر است). خط تولید فراتر از جداسازی عناصر ثابت و متغیر عمل می‌کند در حالی که با معماری‌های مرسوم هر نمونه با دستیابی به رفتارهای سیستم و خصوصیات کیفی انجام می‌شود. بنابراین شناسایی نقاط تغییر مجاز بخشی از وظایف معماری است. محصولات در خط تولید به

[1] Instance
[2] Conceptual value

صورت همزمان موجود هستند و می‌توانند بر حسب رفتارها، خصوصیات کیفی، سکو، شبکه، پیکربندی فیزیکی، میان افزار، فاکتورهای مقیاس‌پذیری و غیره تغییر پیدا کنند. خط تولید نیاز دارد که سه موضوع مهم را مد نظر قرار دهد که عبارتند از:

- تعیین نقاط تغییر
- حمایت از نقاط تغییر
- ارزیابی معماری در جهت متناسب بودن برای خط تولید

۱۴-۴-۱- تعیین نقاط تغییر

تعیین نقاط تغییر یک فعالیت مداوم است. به دلیل روشهای مختلفی که محصول می‌تواند تغییر پیدا کند نقاط تغییرات (گونه‌ای مختلف) می‌تواند به صورت مجازی در هر زمان از فرآیند توسعه تعیین شوند. بعضی از نقاط تغییر در زمان استخراج نیازمندیهای خط تولید تعیین می‌شوند، بعضی‌ها در زمان طراحی معماری شناسایی می‌شوند و تعدادی نیز در زمان پیاده‌سازی مشخص می‌شوند. حتی ممکن است تعدادی نقاط تغییر در زمان پیاده‌سازی دومین محصول (یا محصولات بعدی) تعیین شوند.

تغییرات شناسایی شده در فرآیند نیازمندیها می‌تواند دربرگیرنده خصوصیات، سکوها، واسط‌های کاربری، خصوصیات کیفی و بازار مقصد باشد. بعضی از این تغییرات وابستگی درونی به یکدیگر دارند. برای مثال، واسط کاربری می‌تواند به سکوی مورد استفاده یا بازار مقصد خاص وابسته باشد. نقاط تغییر مشخص شده در فرآیند طراحی معماری گزینه‌های برای پیاده‌سازی تغییرات شناسایی شده خواهد بود یا اینکه تغییرات عادی در روند طراحی خواهند شد زیرا تصمیمات خاص تا زمانی که اطلاعات بیشتر قابل دسترس باشند به تعویق انداخته می‌شود.

۱۴-۴-۲- حمایت از نقاط تغییر

در معماریهای مرسوم تقریبا همیشه مکانیزم مربوطه برای دستیابی به نمونه‌ها با تغییرات کد همراه است. اما در خط تولید نرم‌افزار معماری حمایت کننده نقاط تغییر می‌تواند حالتهای مختلف را به خود بگیرد:

§ *افزودن یا حذف عناصر*[1]

این تصمیم در ایجاد رویه‌ها برای محصولات مختلف بازتاب دارد یا اینکه پیاده‌سازی یک عنصر می‌تواند مبتنی بر بعضی پارامترهای مشخص کننده وجود یا عدم وجود آن کامپایل شود.

§ *افزودن تعداد مختلف از عناصر تکرار شده*[2]

برای یک نمونه، تغییرات خیلی زیاد می‌توانند از طریق اضافه کردن سرویس‌های جدید به عنوان نقطه تغییر تولید شوند.

§ *انتخاب نسخه‌هایی از عناصر که دارای واسط‌های یکسان ولی رفتارها یا خصوصیات کیفی متفاوت هستند.*

انتخاب می‌تواند در زمان کامپایل، زمان ایجاد و حتی زمان اجرا نیز اتفاق بیفتد. دو مکانیزم انتخاب، کتابخانه‌های ایستا[3] و کتابخانه‌های اتصال پویا[4] نامیده می‌شوند. کتابخانه‌های ایستا دربرگیرنده عملکردهای خارجی هستند که بعد از زمان کامپایل اتصال می‌یابند و کتابخانه‌های اتصال پویا انعطاف‌پذیری حالت ایستا را دارد ولی تصمیمات را مبتنی بر زمینه و شرایط اجرایی به زمان اجرا منتقل می‌کنند. با تغییر کتابخانه‌ها می‌توان پیاده‌سازی عملکردهایی که نام و امضاء آنها مشخص است، تغییر داد.

مکانیزمهای تشریح شده تغییرات کلی در سطح معماری را ایجاد می‌کنند. مکانیزمهای دیگر نیز می‌توانند برای تغییر جنبه‌های عنصر خاص معرفی شوند که تغییر کد منبع در این دسته قرار می‌گیرد. تکنیک‌های پیچیده دیگر عبارتند از :

§ در سیستمهای شیءگرا عمومی‌سازی یا خصوصی‌سازی[5] کلاس‌های خاص می‌تواند تغییرات را نشان دهد. کلاسها می‌توانند طوری نوشته شوند که انواع مشخصاتی را بپذیرند که می‌تواند برای محصولات مختلف در صورت لزوم نوشته شود.

[1] Inclusion or omission of elements
[2] Inclusion of a different number of replicated elements
[3] Static libraries
[4] Dynamic link libraries
[5] Specializing

- ساخت نقاط گسترش در عناصری که پیاده‌سازی می‌شوند. این نقاط محلهایی هستند که رفتارهای یا عملکردهای اضافی می‌توانند با اطمینان افزوده شوند.
- تغییرات می‌تواند به معرفی پارامترهای زمان ایجاد در یک عنصر، زیرسیستم یا مجموعه‌ای از زیرسیستمها اعمال شود که به موجب آن، عنصر از طریق تنظیم مجموعه‌ای از مقادیر پیکربندی می‌شود.
- بازتاب[1] قابلیتی از برنامه برای دستکاری داده در خود، محیط اجرا و یا حالت خاص است. برنامه‌های بازتاب‌کننده می‌توانند رفتار خود را مبتنی بر زمینه تنظیم کنند.
- سربارگذاری[2] روشی از استفاده مجدد عملکردها است تا بر روی انواع مختلف عمل کند. سربازگذاری استفاده مجدد کد را افزایش داده ولی این امر همراه با هزینه درک و پیچیدگی کد است.

برای معماری خط تولید، پایه دارایی اصلی و معماری هر یک از محصولات باید مستندسازی انجام شود (یعنی مستند وجود داشته باشد). مستند برای معماری خط تولید باید به وضوح نقاط تغییر و استدلالهای[3] لازم برای هر نقطه تغییر را نشان دهد. همچنین باید فرآیند ایجاد نمونه‌های مختلف توسط معماری را تشریح کند. یعنی اینکه چگونه نقاط تغییر به کار گرفته می‌شوند. به صورت نظری، هر نقطه تغییر باید به صورت مستقل تشریح شود اما در عمل چنین حالتی برای تمام نقاط تغییر ممکن نیست. بعضی از ترکیبات[4] ممکن است اشتباه بوده یا منجر به خطا شود بنابراین مستندی لازم است که شرح دهد کدام انقیاد نقاط تغییر درست و کدام نادرست است.

برای معمار یک محصول منفرد مستندسازی می‌تواند بر حسب تفاوت‌های بین انقیاد نقاط تغییر حاصل شود. برای مثال، معماری برای محصول #۱۶ می‌تواند نیازمند سه سرویس‌دهنده، شصت و چهار ایستگاه سرویس‌گیرنده، دو پایگاه‌داده، عنصر تصویری سرعت بالا اما قدرت تفکیک‌پذیری[5] پائین و تولیدکننده[6] پیام بدون هیچ گونه رمزگذاری باشد.

[1] Reflection
[2] Overloading
[3] Rationale
[4] Combinations
[5] Resolution
[6] Generator

۱۴-۴-۳- ارزیابی معماری در جهت متناسب بودن برای خط تولید

مشابه با هر نوع معماری، معماری خط تولید نیز باید ارزیابی شود تا اینکه مشخص شود آیا اهداف مورد نظر را برآورده می‌کند یا نه؟ در حقیقت در نظر گرفتن اینکه تعدادی سیستم متکی بر آن خواهند شد منجر به این می‌شود که فرآیند ارزیابی نقش خیلی مهم در معماری خط تولید داشته باشد. روشهای ارزیابی که قبلا تشریح شده‌اند (***ATAM, CBAM***) قابل بکارگیری برای خط تولید نیز هستند. معماری باید برای استحکام[1] و عمومیت[2] خود ارزیابی شود تا اطمینان حاصل شود که می‌تواند پایه‌ای برای محصولات مشخص شده در حیطه خط تولید در نظر گرفته شود. معماری خط تولید همچنین باید به منظور تضمین اینکه رفتارهای خاص و نیازمندیهای کیفی مربوط به محصولات را برآورده می‌کند مورد ارزیابی قرار بگیرد.

این نوع ارزیابی می‌بایستی که بر روی نقاط تغییر متمرکز شود تا تضمین کند که آنها مناسب هستند، انعطاف‌پذیری کافی برای پوشش حیطه خط تولید را دارند، اجازه ایجاد سریع محصولات را می دهند و اینکه آنها هزینه‌های کارایی زمان اجرای غیرقابل قبول را تحمیل نمی‌کنند. اگر روش ارزیابی سناریو محور باشد باید سناریوهایی استخراج شوند که دربرگیرنده نمونه‌سازی معماری برای حمایت از محصولات مختلف در خط تولید باشد. همچنین به دلیل اینکه محصولات مختلف در خط تولید می‌توانند نیازمندیهای خصوصیات کیفی متفاوتی داشته باشند بنابراین معماری باید ارزیابی شوند تا مشخص گردد که توانایی لازم را برای فراهم کردن کلیه ترکیبات مورد نظر دارد که در این حالت نیز نیاز به استخراج سناریوهایی است که خصوصیات کیفی مورد نیاز برای سیستمهای عضو خانواده را پوشش دهد.

معمولا بعضی از سخت‌افزارها و فاکتورهای تاثیر گذار بر کارایی در معماری خط تولید در زمان شروع مشخص نیستند. در این حالت ارزیابی می‌تواند مبتنی بر کارایی که معماری می‌تواند با فرضیاتی در مورد سخت‌افزار و نقاط تغییر در نظر گرفته می‌شود، انجام شود. ارزیابی می‌تواند محلهایی که پتانسیل رقابتی[3] دارند را شناسایی کرده به صورتی که بتوان سیاست‌ها و راهبردهای لازم برای حل آنها را در نظر گرفت.

ارزیابی باید بر روی نمونه یا گونه‌ای از معماری که برای تولید یک یا بیش از یک محصول در خط تولید استفاده می‌شود، انجام بگیرد. ارزیابی معماری خط تولید به حوزه‌های مختلفی که در داخل خط تولید وجود

[1] Robustness
[2] Generality
[3] Potential contention

دارد، بستگی دارد. اگر تفاوت‌ها در این حوزه‌ها وجود نداشته باشد ارزیابی معماری خط تولید می‌تواند مختصر شود زیرا پیامدهای عادی که درون ارزیابی یک محصول منفرد وجود دارد در ارزیابی معماری خط تولید مورد بررسی قرار گرفته است. در حقیقت با توجه به این که معماری محصول گونه‌ای از معماری خط تولید است ارزیابی معماری محصول نیز گونه‌ای از ارزیابی معماری خط تولید است. بنابراین با توجه به روشی که برای ارزیابی استفاده می‌شود محصولات ارزیابی (چک لیست‌ها، سناریوها و غیره) پتانسیل قابلیت استفاده مجدد را خواهند داشت. نتایج ارزیابی معماری محصول اغلب بازخوردهای بسیار مفیدی برای معمار خط تولید فراهم می‌آورد و باعث بهبود معماری می‌شود.

زمانی که محصولی پیشنهاد می‌شود که خارج از حیطه اصلی خط تولید است معماری خط تولید می‌تواند مجددا مورد ارزیابی قرار بگیرد تا مشخص شود که آیا معماری برای آن مناسب است یا نه؟ اگر مناسب بود حیطه خط تولید می‌تواند گسترش یابد تا آن محصول را پوشش دهد یا منجر به یک خط تولید جدید شود. اگر مناسب نباشد ارزیابی نشان خواهد داد که چگونه معماری می‌تواند تغییر پیدا کند تا بتواند آن محصول را تحت پوشش قرار دهد.

۱۴-۵- مشکلات ایجاد خط تولید نرم‌افزار

موفقیت در حوزه خط تولید کاملا به بلوغ سازمان توسعه‌دهنده مرتبط است و در این زمینه فناوری تنها مشکل بر سر راه نیست. سازمان، فرآیند و پیامدهای حرفه به صورت یکسان نقش حیاتی در بدست آوردن سودمندی از خود تولید نرم‌افزار دارند. موسسه مهندسی نرم‌افزار[1] بیست و نه پیامد یا "حوزه‌های عملی"[2] را شناسایی کرده است که موفقیت یک سازمان در حوزه خط تولید را تحت تاثیر قرار می‌دهد. بیشتر این حوزه‌های عملیاتی در جریان توسعه یک سیستم منفرد اعمال می‌شوند اما در زمینه خط تولید به یک بعد جدید تبدیل می‌شوند. دو مثال در این مورد تعریف معماری و مدیریت پیکربندی است.

تعریف معماری مهمترین فعالیت برای هر پروژه است و همان طور که در بخش قبلی ذکر شد معماری برای بهتر مشخص کردن نقاط تغییر در خط تولید نرم‌افزار مورد نیاز است. مدیریت پیکربندی نیز فعالیت مهم برای هر پروژه‌ای است و برای خط تولید نرم‌افزار نیز بسیار پیچیده است زیرا هر محصول نتیجه انقیاد تعداد

[1] Software engineering institute
[2] Practice areas

بیشماری از گونه‌ها یا نقاط تغییر است. مشکل مدیریت پیکربندی برای خطوط تولید نیز یعنی تولید مجدد هر نسخه از محصول قابل تحویل به ازای هر مشتری است که منظور از محصول یعنی کد منبع و فرآورده‌های حمایت کننده که می‌توانند متغیر بین فضای نیازمندی و موارد آزمایش تا راهنمای کاربران و راهنمایی نصب باشد. این موضوع دربرگیرنده این نکات است که کدام نسخه از هر دارایی هسته در ساختار محصول استفاده شده است، چگونه هر دارایی در پیکره سیستم به کار گرفته شده است و یا کدام کدهای خاص یا مستنداتی اضافه شده است.

۱۴-۵-۱- راهبردهای سازگاری[1]

مشابه مشکلات ناشی از دیگر فناوریها، بدست آوردن یک رهیاف خط تولید برای سازمان نیازمند در نظر گرفتن ملاحظات زیادی است. چگونگی حل کردن این موضوع به فرهنگ و زمینه سازمان بستگی دارد. سازگاری (تطبیق) بالا به پائین[2] زمانی ایجاد می شود که مدیر دستور دهد که سازمان از این روش استفاده کند. مشکلی که در اینجا وجود دارد آن است که بتوان کارکنانی را بدست آورد که روشی کار خود را تغییر دهند. سازگاری پائین به بالا[3] زمانی اتفاق می‌افتد که طراحان و توسعه‌دهندگانی که در سطح یک محصول کار می‌کنند تشخیص دهند که آنها بدون هیچ الزامی کار هر یک را تکرار می‌کنند و لذا شروع به اشتراک‌گذاری منابع و توسعه دارایی‌های اصلی عمومی می‌کنند. مشکل در اینجا نیز پیدا کردن مدیری است که از کار حمایت کرده و این نوع تکنیک را در دیگر بخش‌های سازمان اعمال کند. هر دوی این روشها به طور صحیح می‌کنند و هر دو از وجود شخصی که کلیه چشم‌انداز خط تولید بومی را ساخته و می‌تواند این چشم‌انداز را به دیگران اجبار کند، کمک می‌گیرند. مشکل دیگری که متعامد بر مشکل قبلی است چگونگی رشد[4] خط تولید است.

در خط تولید پیشگیرانه[5] سازمان خانواده محصولات خود را با استفاده از تعریف جامع حیطه مشخص می‌کند. آنها این فرآیند را به صورت شانسی انجام نمی‌دهند بلکه از مزایای تجربه خود در حوزه کاربرد، دانش خود درباره فناوری و گرایش بازار و وضعیت حرفه بهره می‌برند. مدل پیشگیرانه از قدرتمندترین مدلهای خط

[1] Adoption strategies
[2] Top-down adoption
[3] Botton-up adaption
[4] Grow
[5] Proactive

تولید در حال رشد است زیرا به سازمان اجازه می‌دهد که تصمیمات راهبردی بلند مدت را اتخاذ کند. حیطه‌بندی صریح خط تولید اجازه می‌دهد که محصولات قبلی موجود در بازار مورد بررسی قرار دارد و گسترشهایی را برای خط تولید ایجاد کرد تا بتوان سریعا شکاف موجود در بازار را پر کرد. در واقع حیطه خط تولید پیشگیرانه به سازمان اجازه می‌دهد که سرنوشت خود را در دست بگیرد.

گاهی سازمانی توانایی آن را ندارد که نیازهای بازار را با قطعیتی که مدل پیشگیرانه پیشنهاد می‌دهد، پیش‌بینی کند. دلیل این امر می‌تواند ناشی از این موضوع باشد که احتمالا حوزه کاری یک حوزه جدیدی است، وضع بازار ثابت نیست و یا سازمان نمی‌تواند برای ایجاد پایه دارایی سازمان که کل حیطه را یک مرتبه پوشش دهد، تلاشهای زیادی انجام دهد. در این حالت می‌توان از مدل واکنشی[1] استفاده کرد. در این مدل، سازمان اعضا خانواده محصولات را از آخرین محصولات انتخاب می‌کند. با هر محصول جدید معماری و طراحی‌ها در صورت نیاز گسترش می‌یابند و پایه دارایی هسته نیز به جای اینکه از مشترکاتی که قبلا طرح ریزی شده‌اند ساخته شود از مشترکاتی که جدیدا کشف شده‌اند ساخته می‌شوند. مدل واکنشی تمرکز کمتری بر روی برنامه‌ریزی آینده و تنظیمات راهبردی دارد و به جای آن سازمان به خود اجازه می‌دهد که به سمتی حرکت کند که بازار دیکته می‌کند.

آگاهی از انواع مدلهای سازگاری به سازمان کمک می‌کند که مدل مناسب را انتخاب کند. مدل پیشگیرانه نیازمند سرمایه‌گذاری اولیه اما کار مجدد کمتری نسبت به مدل واکنشی است. مدل واکنشی به صورت انحصاری به کار مجدد همراه با سرمایه‌گذاری اولیه خیلی کم متکی است. انتخاب اینکه کدام مدل مناسب سازمان است بیشتر به موقعیت حرفه وابسته است.

۱۴-۵-۲- ایجاد محصولات و تکامل خط تولید

سازمانی که دارای خط تولید است یک معماری و مجموعه‌ای از عناصر مرتبط به آن را خواهد داشت. گاها سازمان عضو (محصول) جدیدی از خط تولید ایجاد می‌کند که بعضی ویژگیهای آن مشترک و بعضی دیگر متفاوت از ویژگیهای سایر اعضا خط تولید است. مشکل مرتبط با خط تولید مدیریت تکامل آن است. با گذر

[1] Reactive

زمان خط تولید یا در حالت خاص مجموعه دارایی‌های هسته که از آنها محصولات ساخته می‌شوند باید تکامل یابد. این تکامل از طریق منابع داخلی و خارجی به پیش رانده می‌شود.

۱. *منابع خارجی*

- نسخه‌های جدید عناصر در خط تولید از طریق فروشندگان آنها عرضه خواهد شد و محصولات آینده نیاز خواهند داشت که بر پایه آنها ساخته شوند.
- عناصر ایجاد شده خارجی می‌توانند به خط تولید اضافه شوند. بنابراین عملکردهایی که قبلا به وسیله عناصر توسعه داده شده داخلی انجام می‌شدند حال می‌توانند از طریق عناصر بدست آمده از خارج خط تولید انجام شوند (این عملکرد می‌تواند حالت عکس نیز داشته باشد) یا محصولات آینده نیاز خواهند داشت که از مزایایی فناوریهای جدید که در عناصر توسعه داده شده خارجی قرار دارند، استفاده کنند.
- ویژگیهای جدید می‌تواند در جهت پاسخ به نیازهای کاربران یا فشارهای رقابتی وارد خط تولید شوند.

۲. *منابع داخلی*

- این موضوع باید مشخص شود که عملکرد جدیدی که به خط تولید اضافه می‌شود در حیطه آن است یا نه؟ اگر در داخل حیطه باشد آنها به راحتی می‌تواند از طریق دارایی‌های پایه ساخته شود و اگر در داخل حیطه نباشد تصمیم‌گیری باید اتخاذ شود. این تصمیم‌گیری می‌تواند دو حالت داشته باشد یکی اینکه محصول مورد نظر از طریق طی کردن مسیر تکاملی معماری تولید می‌شود یا اینکه پایه دارایی‌ها باید گسترش یابند تا بتوان آن را پوشش دهد. در صورتی که عملکرد جدید به احتمال زیاد در محصولات آتی به کار گرفته شود به روز رسانی خط تولید می‌تواند انتخاب مناسبی باشد اما این نوع عملکرد در بردارنده هزینه و زمان است.
- نگهداری محصولات سازگار با خط تولید نیازمند زمان و تلاش است. اما عدم انجام آن می‌تواند به روزرسانی آتی را خیلی زمانبر و پرهزینه کند زیرا یا محصول نیاز خواهد داشت تا سازگار با آخرین عناصر خط تولید شود یا قادر نخواهد بود که از مزایایی عملکردهای جدید اضافه شده به خط تولید استفاده کند.

۱۴-۵-۳- ساختار سازمانی

پایه دارایی که یک محصول به آن وابسته است ولی مسیر تکاملی خاص خودش را دارد نیازمند سازمانی است تا تصمیم بگیرد چگونه پایه دارایی و توسعه محصول را مدیریت کند. آقای ***Jan Bosch*** مدلهای سازمانی خط تولید را مورد مطالعه قرار داده است و چهار نوع را شناسایی کرده است که به شرح زیر هستند:

۱. *بخش توسعه*[1]

کل توسعه نرم‌افزار در یک واحد متمرکز شده است. از هر عضو واحد انتظار می‌رود که هر کاری را در خط تولید انجام دهد. به عنوان مثال مهندسی حوزه یا مهندسی کاربرد را در زمان مورد نیاز انجام دهد. این مدل در سازمانهای کوچک و سازمانهایی که سرویس‌های مشاوره‌ای می‌دهند انجام می‌شود. اگر چه مسیر تعاملی در آن کوتاه و ساده است ولی داشتن یک واحد تعدادی زیادی مشکلات دارد. آقای ***Bosch*** بیان کرده است که این مدل احتمالا برای واحدهای حداکثر ۳۰ نفر کار می‌کند اما در سازمانهای خیلی کوچک که خط تولید آنها به تناسب کوچک است این مدل می‌تواند روش شروع خوبی باشد.

۲. *واحدهای حرفه*[2]

هر واحد حرفه مسئول زیرمجموعه‌ای از سیستمها در یک خانواده از محصولات است که به صورت مشابه تقسیم‌بندی شده‌اند. دارایی‌های به اشتراک گذاشته شده از طریق واحدهایی که به آنها نیاز دارند توسعه داده شده و در دسترس تیم قرار می‌گیرند (تعامل در میان واحدهای حرفه برای توسعه دارایی‌های جدید امکان‌پذیر است). این مدل وابستگی‌های مختلفی بر میزان انعطاف‌پذیری واحد حرفه دارد. بدون هیچ محدودیتی محصولات تمایل دارند که از مسیر توسعه خود خارج شده و رهیافت خط تولید را ناکارآمد کنند. این مسئولیت برای دارایی‌های خاص به واحدهای حرفه خاص که مسئولیت نگهداری آنها برای استفاده در کل خط تولید دارند، تخصیص داده می‌شود. واحدهای حرفه دیگر نیز برای استفاده از این دارایی‌ها مورد نیاز هستند. آقای ***Bosch*** تخمین زده که این مدل می‌تواند به سازمانهایی با کارکنان بین ۳۰ تا ۱۰۰ نفر اعمال شود. این مدل از یک ریسک رنج می‌برد و آن این

[1] Development department
[2] Business units

است که هر واحد حرفه نخست بر روی محصول خود متمرکز می‌شود و موفقیت خط تولید در جایگاه بعدی قرار می‌گیرد.

۳. *واحد مهندسی حوزه*[1]

واحد خاصی مسئول توسعه و نگهداری پایه دارایی اصلی است که واحد حرفه با استفاده از آنها محصول مورد نظر را تولید می‌کنند. آقای ***Bosch*** معتقد است زمانی که سازمان بیش از ۱۰۰ نفر کارمند دارد کانالهای ارتباطی بین واحدهای حرفه مختلف اجتناب ناپذیر خواهد شد و بنابراین یک کانال ارتباطی متمرکز به واحد مرکزی دارایی‌های به اشتراک‌گذاشته شده ضروری خواهد بود. در این مدل، وجود فرآیند نظام‌مند و مستحکم برای مدیریت ارتباطات و تضمین اینکه سلامت کلی خط تولید هدف تمام بخش‌ها است، بسیار مهم است.

۴. *واحدهای مهندسی حوزه سلسله مراتبی*[2]

زمانی که خط تولید بسیار بزرگ و خیلی پیچیده است می‌توان خط تولید را به صورت سلسله مراتبی در نظر گرفت. یعنی اینکه، خط تولید متشکل از زیرگروهایی هست که اشتراکات بیشتری با یکدیگر در مقایسه با اشتراکات دیگر محصولات خط تولید دارند. در این حالت واحد مهندسی حوزه دارایی‌های به اشترک گذاشته شده دانه درشتی را برای خط تولید، ایجاد می‌کند و واحد مهندسی حوزه دیگر نیز می‌تواند دارایی‌های اصلی خاص هر زیر گروه را ایجاد کند. این مثال ذکر شده دارای دو سطح است ولی مدل می‌تواند در صورت اینکه زیرگروها دربرگیرنده زیرگروه‌های دیگر نیز باشد، گسترش پیدا کند. واحد حوزه سلسله مراتبی برای هر خط تولید بزرگ که توسط سازمانهای بزرگ ایجاد شده است کار می‌کند. مزیت اصلی آنها گرایش به کاهش یا افزایش پاسخ‌دهی سازمان به نیازهای جدید است.

[1] Domain engineering unit
[2] Hierarchical domain engineering units