

به نام نردان پاک

طراحی سیستم‌های شیء‌گرا

نوشته: پروفسور گریدی بوچ

ترجمه: دکتر محسن صدیقی مشکنانی

فهرست مطالب

فصل ۰: مقدمه.....	۹
فصل ۱: پیچیدگی.....	۳
۱-۱ پنج ویژگی سیستم‌های پیچیده.....	۳
۱-۲ فرم متعارف یک سیستم پیچیده.....	۳
۱-۳ محدودیت‌های انسان در برخورد با پیچیدگی‌ها.....	۵
۱-۴ به نظم آوردن آشفتگی.....	۵
۱-۵ مدل‌های طراحی شیء گرا.....	۶
۱-۶ خلاصه‌ی فصل اول از کتاب طراحی شیء گرا / پیچیدگی.....	۶
فصل ۲: مدل شیء.....	۹
۲-۱ عوامل شکل‌گیری مدل شیء.....	۹
۲-۲ برنامه‌سازی، طراحی و تحلیل شیء گرا.....	۹
۲-۳ ارکان مدل شیء.....	۱۰
۲-۴ خلاصه‌ی فصل دوم از کتاب طراحی شیء گرا / مدل شیء.....	۱۶
فصل ۳: کلاس‌ها و شیء‌ها.....	۱۷
۳-۱ ماهیت شیئی.....	۱۷
۳-۲ انواع روابط شیء‌ها با یکدیگر.....	۱۹
۳-۳ ماهیت کلاسی.....	۱۹
۳-۴ انواع روابط کلاس‌ها با یکدیگر.....	۲۰
۳-۵ نقش متقابل کلاس‌ها و شیء‌ها.....	۲۳
۳-۶ ساخت کلاس‌ها و شیء‌های با کیفیت.....	۲۳
۳-۷ خلاصه‌ی فصل سوم کتاب طراحی شیء گرا / کلاس‌ها و شیء‌ها.....	۲۵
فصل ۴: دسته‌بندی.....	۲۷
۴-۱ دسته‌بندی و طراحی شیء گرا.....	۲۷
۴-۲ شناسایی و تعیین کلاس‌ها و شیء‌ها.....	۳۰
۴-۳ شناسایی و تعیین تجریدهای کلیدی.....	۳۱
۴-۴ شناسایی و تعیین مکانیزم‌ها.....	۳۲
۴-۵ خلاصه‌ی فصل چهارم کتاب طراحی شیء گرا / طبقه‌بندی.....	۳۳
فصل ۵: علامت‌گذاری.....	۳۵
۵-۱ عناصر علامت‌گذاری طراحی شیء گرا.....	۳۸
۵-۲ نمودار کلاس.....	۴۰
۵-۳ نمودارهای تغییر حالت.....	۴۹

۵-۴	نمودارهای شیء	۵۰
۵-۵	نمودارهای زمانی	۵۴
۵-۶	نمودارهای مؤلفه	۵۵
۵-۷	نمودار فرایند	۵۷
۵-۸	محصول طراحی شیء گرا	۵۹
۵-۹	خلاصه‌ی فصل پنجم کتاب طراحی شیء گرا / علامتگذاری	۵۹
فصل ۶: فرایند طراحی شیء گرا		
۶-۱	۱- شناسایی و تعیین کلاس‌ها و شیء‌ها	۶۱
۶-۲	۲- شناسایی و تعیین معانی کلاس‌ها و شیء‌ها	۶۲
۶-۳	۳- شناسایی و تعیین روابط بین کلاس‌ها و شیء‌ها	۶۲
۶-۴	۴- پیاده‌سازی کلاس‌ها و شیء‌ها	۶۳
۶-۵	۵- خلاصه‌ی فصل ششم از کتاب طراحی شیء گرا / فرایند طراحی شیء گرا	۶۴
فصل ۷: نکات علمی		
۷-۱	۱- طراحی شیء گرا در چرخه‌ی عمر نرم‌افزار	۶۵
۷-۲	۲- مدیریت پروژه	۶۷
۷-۳	۳- مهارت‌های تصمیم‌های ایجاد و توسعه	۶۸
۷-۴	۴- مزایا و خطرات طراحی شیء گرا	۶۸
۷-۵	۵- انتقال به طراحی شیء گرا	۶۹
۷-۶	۶- جایگاه موجودیت‌های انسانی از محیط عملیاتی در سیستم کامپیوتری مورد نظر	۶۹
۷-۷	۷- خلاصه‌ی فصل هفتم از کتاب طراحی شیء گرا / نکات علمی در طراحی شیء گرا	۷۰

فهرست شکل‌ها

- شکل ۱-۱: فرم متعارف یک سیستم پیچیده..... ۴
- شکل ۲-۱: مدهای طراحی شیء گرا..... ۶
- شکل ۱-۲: تجرید روی ویژگی‌های اصلی شیء با توجه به دیدگاه ناظر، تکیه می‌کند..... ۱۱
- شکل ۲-۲: دربرگیری جزئیات پیاده‌سازی شیء را پنهان می‌کند..... ۱۱
- شکل ۳-۲: مؤلفه‌ای بودن، تجریدها را در واحدهای مجزا تفکیک می‌کند..... ۱۲
- شکل ۴-۲: تجریدها یک سلسله‌مراتب را تشکیل می‌دهند..... ۱۳
- شکل ۵-۲: نوع‌بندی قوی از تداخل تجریدها جلوگیری می‌کند..... ۱۴
- شکل ۶-۲: توازی اجازه می‌دهد تا شیء‌های مختلف در یک زمان عمل کنند..... ۱۵
- شکل ۷-۲: ماندگاری حالت و کلاس شیء را در رابطه با زمان و مکان حفظ می‌کند..... ۱۵
- شکل ۱-۳: شیء دارای حالت است، رفتارهای کاملاً تعریف شده‌ای را از خود نشان می‌دهد و دارای هویت (شناسه) منحصر به فردی است..... ۱۷
- شکل ۲-۳: کلاس نمایشگر مجموعه‌ای از شیء‌ها با ساختار و رفتار مشترک است..... ۲۰
- شکل ۱-۴: دسته‌بندی، ابزاری است که به وسیله‌ی آن دانش را مرتب می‌کنیم (دسته‌بندی هوشمندانه مشکل است)..... ۲۸
- شکل ۲-۴: ناظران متفاوت، یک شیء خاص را به طرق مختلف دسته‌بندی می‌کنند..... ۲۹
- شکل ۳-۴: کلاس‌ها و شیء‌ها باید در سطح مناسبی از تجرید باشند: نه خیلی بالا و نه خیلی پایین..... ۳۲
- شکل ۴-۴: به وسیله‌ی مکانیزم‌ها شیء‌ها با یکدیگر همکاری می‌کنند تا رفتارهای سطح بالاتری به وجود آورند..... ۳۳
- شکل ۱-۵: مدل‌های مختلف، برای طراحی شیء گرا..... ۳۹
- شکل ۲-۵: نمودار کلاس..... ۴۰
- شکل ۳-۵: شکلک کلاس..... ۴۱
- شکل ۴-۵: یک نمودار کلاس..... ۴۲
- شکل ۵-۵: مثالی از کار دینالیتی..... ۴۲
- شکل ۶-۵: شکلک همه‌بهرهای کلاس..... ۴۳
- شکل ۷-۵: گروه کلاس‌ها..... ۴۳
- شکل ۸-۵: یک نمودار کلاسی بالا، نموداری از گروه‌های کلاس‌ها..... ۴۴
- شکل ۹-۵: نمودار تغییر حالت..... ۴۹
- شکل ۱۰-۵: مثالی از نمودار تغییر حالت..... ۵۰
- شکل ۱۱-۵: کلمات پیام، عمل و متد تقریباً معادل هم به کار می‌روند و معادل تابع عضویت..... ۵۱
- شکل ۱۲-۵: شکلک‌های همگام‌سازی پیام‌ها..... ۵۱
- شکل ۱۳-۵: قابلیت رؤیت یک شیء برای شیء دیگر..... ۵۲
- شکل ۱۴-۵: مثالی برای قابلیت رؤیت یک شیء برای شیء دیگر..... ۵۲
- شکل ۱۵-۵: یک نمودار شیء..... ۵۳
- شکل ۱۶-۵: مثالی برای نمودار شیء..... ۵۳
- شکل ۱۷-۵: مثالی برای نمودار شیء..... ۵۳

- شکل ۱۸-۵: مثالی برای نمودار شیء ۵۳
- شکل ۱۹-۵: نمودار زمانی نمودار زمانی قسمتی از نمودار شیء و بیانگر ترتیب بروز حوادث در بین مجموعه‌ای از شیء‌هاست ۵۵
- شکل ۲۰-۵: نمودارهای مؤلفه ۵۵
- شکل ۲۱-۵: یک نمودار مؤلفه ۵۶
- شکل ۲۲-۵: شکلیک برای زیرسیستم ۵۶
- شکل ۲۳-۵: یک نمودار مؤلفه‌ی سطح بالا (نمودار زیرسیستم‌ها) ۵۷
- شکل ۲۴-۵: شکلیک دستگاه ۵۸
- شکل ۲۵-۵: شکلیک پردازنده ۵۸
- شکل ۲۶-۵: یک نمودار فرایند ۵۸
- شکل ۱-۷: طراحی شیء گرا در چرخه‌ی تولید نرم‌افزار ۶۵
- شکل ۲-۷: منابع انسانی برحسب ماه‌های تلاش مهندسی ۶۷
- شکل ۳-۷: یک انسان در محیط عملیاتی به‌عنوان client ۷۰
- شکل ۴-۷: یک انسان در محیط عملیاتی به‌عنوان server ۷۰
- شکل ۵-۷: جایگاه موجودیت‌های انسانی از محیط عملیاتی در سیستم کامپیوتری ۷۰

فهرست جدول‌ها

۱۶	جدول ۱-۲: کاربردهای مدل شیء.....
۲۱	جدول ۱-۳: روابط شیء‌ها.....
۲۱	جدول ۲-۳: روابط کلاس‌ها.....
۲۲	جدول ۳-۳: روابط کلاس‌ها.....
۲۲	جدول ۴-۳: روابط کلاس‌ها.....
۳۵	جدول ۱-۵: مدلی برای طراحی شیء گرا.....
۳۷	جدول ۲-۵: نمودار کلاس.....
۳۸	جدول ۳-۵: نمودار مؤلفه.....
۴۱	جدول ۴-۵: رابطه‌ی کلاس‌ها.....
۴۲	جدول ۵-۵: کاردینالیتی.....
۴۳	جدول ۶-۵: رابطه‌ی بین گروه‌های کلاس‌ها.....
۴۴	جدول ۷-۵: الگوی نمودار کلاس.....
۴۵	جدول ۸-۵: شرح اجزاء الگوی نمودار کلاس.....
۴۷	جدول ۹-۵: الگوی همه‌بهر کلاس.....
۴۷	جدول ۱۰-۵: الگوی عمل‌ها.....
۴۸	جدول ۱۱-۵: شرح اجزاء الگوی عمل‌ها.....
۴۹	جدول ۱۲-۵: الگوی تغییر حالت.....
۵۴	جدول ۱۳-۵: الگوی نمودار شیء.....
۵۴	جدول ۱۴-۵: الگوی پیام.....
۵۷	جدول ۱۵-۵: الگو برای نمودار مؤلفه.....
۵۸	جدول ۱۶-۵: الگو برای پردازنده.....
۵۹	جدول ۱۷-۵: الگوی پراسس.....
۵۹	جدول ۱۸-۵: الگوی دستگاه.....

فصل

مقدمه

پروفسور گریدی بوچ (Grady Booch) یکی از معروف‌ترین و شناخته شده‌ترین صاحب‌نظران در مورد مدل شیء گرا در جهان است. کتاب طراحی شیء گرا و کاربردها¹ که در سال ۱۹۹۱ میلادی توسط ایشان نوشته شده است، همچنان یکی از مشخص‌ترین متون درسی و یک مرجع عمومی برای تحلیل، طراحی و برنامه‌سازی شیء گرا می‌باشد.

فصل اول این کتاب به موضوع پیچیدگی (complexity) در کل و پیچیدگی نرم‌افزار و نقش تفکیک (decomposition)، تجزید (abstraction) و سلسله‌مراتب (hierarchy) در برخورد با آن، می‌پردازد.

در **فصل دوم** مدل شیء گرا (object model) و ارکان اصلی شامل تجزید (abstraction)، دربرگیری (encapsulation)، مؤلفه‌ای بودن (modularity) و سلسله‌مراتب (hierarchy) و همچنین ارکان فرعی آن شامل نوع بندی (typing)، توازن یا همبودی (concurrency) و ماندگاری (persistence) تبیین شده است.

فصل سوم به شیء‌ها (object)، کلاس‌ها (class) و ویژگی‌های آنها و همچنین به روابط مختلف بین شیء‌ها، بین کلاس‌ها و بین شیء‌ها و کلاس‌ها می‌پردازد.

فصل چهارم حاوی موضوع طبقه بندی یا کلاس بندی (classification)، ارتباط آن با مدل شیء، مشکلات کلاس بندی، روش‌هایی برای شناسایی و تعیین شیء‌ها، کلاس‌ها و همچنین ارتباط و رفتار آنها با یکدیگر (مکانیزم mechanism) می‌باشد.

فصل پنجم قواعد علامت‌گذاری برای چهار نمودار اصلی و دو نمودار کمکی را ارائه می‌کند. نمودارهای اصلی شامل نمودار کلاس (class diagram)، نمودار شیء (object diagram)، نمودار مؤلفه (module diagram) و نمودار فرایند (process diagram) می‌باشد. نمودارهای کمکی شامل دو نمودار تغییر حالت (state transition diagram) و نمودار زمانی (timing diagram) است.

در **فصل ششم** فرایند طراحی شیء گرا به‌عنوان یک فرایند فزاینده (incremental) و تکرار شونده (iterative) مورد بحث قرار گرفته و قدم‌های اصلی در این راه مطرح شده است: شناسایی و تعیین شیء‌ها و کلاس‌ها در سطحی از تجزید، شناسایی و تعیین معنی (semantics) این کلاس‌ها و شیء‌ها، شناسایی و تعیین روابط بین این کلاس‌ها و شیء‌ها، و بالاخره پیاده‌سازی این کلاس‌ها و شیء‌ها.

در **فصل هفتم** نکات عملی (pragmatics) در طراحی شیء گرا، نقش این روش در چرخه‌ی عمر نرم‌افزار و تأثیر آن در مدیریت نرم‌افزار مورد بررسی قرار می‌گیرد.

¹ Booch, G. (1991). "Object Oriented Design with Applications", Benjamin.

بقیه فصول کتاب بوچ (فصول هشتم تا سیزدهم) حاوی پنج کاربرد در زمینه‌های مختلف است، که هر یک طی یک فصل توسط یک زبان شیء‌گرا ارایه شده است.

این نوشتار ترجمه‌ی منتخبی از مطالب فصول اول تا هفتم کتاب بوچ است. منتخبی که سعی شده است ساختار و اهداف اصلی بوچ را دربر داشته باشد. بخش "خلاصه" (summary) از هر فصل به‌طور کامل ترجمه شده است. در عین حال در حاشیه متن عناوینی نیز قید شده است، به این امید که متن را برای خواننده گویاتر نماید. قواعد علامت‌گذاری UML¹ نیز که بعدها توسط بوچ و همکارانش در مؤسسه‌ی غیر انتفاعی OMG² به‌جای علامت‌گذاری فصل پنج ارائه گردید و اکنون عملاً به‌صورت یک استاندارد برای مدل‌سازی شیء‌گرا درآمده است، در قالب ۷ فصل اضافه شده است.

الحمد لله رب العالمين
محسن صدیقی مشکنانی

¹ Unified Modeling Language

² Object Management Group (www.omg.org)

فصل ۱

پیچیدگی (Complexity)

۱-۱ پنج ویژگی سیستم‌های پیچیده (*The Five Attributes of Complex System*)

غالباً پیچیدگی فرم سلسله مراتب (hierarchy) را می‌گیرد، که در آن سیستم پیچیده از زیر سیستم‌های مرتبط به هم تشکیل می‌شود. و هر زیر سیستم نیز به نوبه‌ی خود دارای زیر سیستم‌هایی است و الی آخر. تا اینکه به عناصر اولیه (elementary) برسیم. انتخاب اینکه در یک زیر سیستم چه اجزایی، اولیه (primitive) است، نسبتاً دلخواه بوده و بیشتر به دیدگاه ناظر بستگی دارد. چیزی که برای یک ناظر، اولیه محسوب می‌شود، ممکن است برای ناظری دیگر، از سطح بالاتری از تجرید (abstraction) برخوردار باشد.

پیوندهای درون مؤلفه‌ای (intra component linkages) عموماً قوی‌تر از پیوندهای برون مؤلفه‌ای (inter component linkage) که بین یک مؤلفه با مؤلفه‌های دیگر است، می‌باشند [8]. این تفاوت بین پیوندی‌های درون مؤلفه‌ای و برون مؤلفه‌ای موجب جدایی قسمت‌های مختلف سیستم می‌شود. به گونه‌ای که مطالعه‌ی مجزای هر قسمت را میسر می‌سازد.

سیستم‌های سلسله مراتبی معمولاً ترکیبی تنها از چند نوع مختلف از زیر سیستم‌ها با ترکیب‌ها و ترتیب‌های مختلف هستند [9].

سیستم‌های پیچیده‌ای که کار می‌کنند بایستی از سیستم ساده شروع شده و به تدریج کامل شده باشند.

... سیستم پیچیده‌ای که از هیچی (scratch) طراحی شده باشد، هرگز کار نمی‌کند. باید با

سیستم ساده‌ی در حال کار شروع کنید [11].

در روند تکامل سیستم‌ها، شی‌هایی که زمانی پیچیده بودند، با ساخته شدن سیستم‌های بزرگتر، شی‌های ساده‌تر تلقی می‌شوند.

۱-۲ فرم متعارف یک سیستم پیچیده (*The Canonical Form of a Complex System*)

یک مثال (خلبان) کشف تجرید و مکانیزم‌های عمومی، فهم ما از سیستم‌های پیچیده را ساده کرده است. به عنوان مثال تنها با یک توجیه چند دقیقه‌ای یک خلبان مجرب (با درک خواص مشترک هواپیماهای مشابه و ویژگی‌های منحصر به فرد یک هواپیمای بخصوص) قدم به هواپیمایی می‌گذارد که قبلاً با آن پرواز نکرده است و آن را به سلامت هدایت می‌کند. این مثال بیانگر آن است که ما از واژه‌ی سلسله مراتب (hierarchy) استفاده می‌کرده‌ایم (گرچه سست). اما در اکثر سیستم‌های جالب توجه، سلسله مراتب‌های بسیاری حضور دارند.

ما دریافته‌ایم که سلسله مراتب‌های نوعی از (kind of) و جزئی از (part of) در هر سیستم مهم است. با دلالتی که در فصل دوم مطرح خواهد شد، سلسله مراتب نوعی از، را ساختار کلاسی (class structure) و سلسله مراتب جزئی از، را ساختار شیء (object structure) می‌نامیم.

شکل متعارف
سیستم‌های پیچیده.

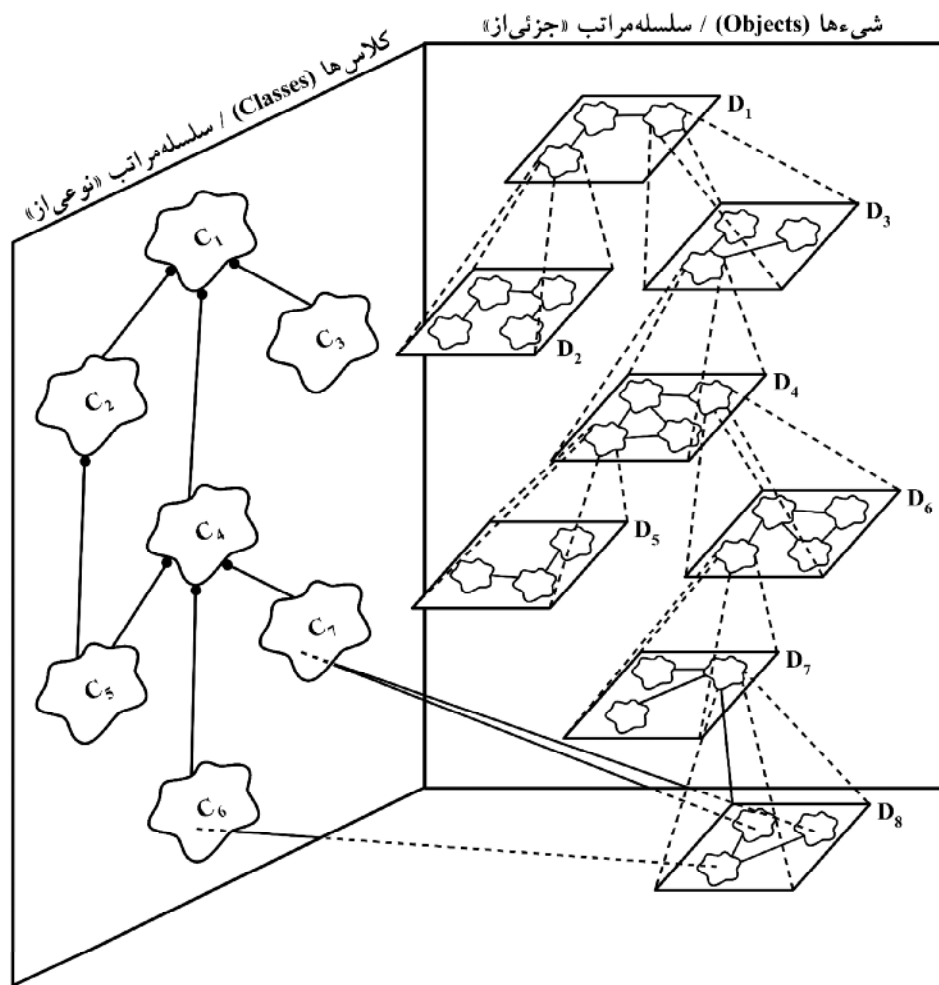
اگر مفاهیم ساختار کلاسی و ساختار شیء و پنج مشخصه‌ی سیستم‌های پیچیده را با هم در نظر بگیریم، درمی‌یابیم که تمام سیستم‌های پیچیده مجازاً به یک شکل متعارف (canonize form) هستند، (شکل ۱-۱).

در این جا ما دو سلسله مراتب متعامد (orthogonal) از یک سیستم را می‌بینیم: ساختار کلاسی و ساختار شیء.

عدم استقلال ساختار
شیء و ساختار
کلاسی از یکدیگر.

هر سلسله مراتب دارای سطوح متعددی است. کلاس‌ها و شیء‌های مجردتر، از موارد ابتدایی‌تر ساخته شده‌اند. به‌ویژه در بین قسمت‌های ساختار شیء روابط کاملاً تعریف شده‌ای بین شیء‌های آن سطح وجود دارد. با نگاه به داخل هر سطح، سطح دیگری از پیچیدگی معین می‌شود. توجه کنید که ساختارهای کلاس و شیء کاملاً مستقل از یکدیگر نیستند. بلکه هر شیء از ساختار شیء بیانگر یک رویداد (instance) خاص از همان کلاس است. همچنان‌که در شکل نمایش داده شده است. در داخل یک سیستم، تعداد شیء‌ها خیلی بیشتر از تعداد کلاس‌هاست. بنابراین با نشان دادن سلسله مراتب‌های «جزئی از» و «نوعی از» ما صراحتاً افزونگی یا زوائد (redundancy) سیستم مورد نظر را نشان می‌دهیم. اگر ساختار کلاسی از سیستم را معلوم نمی‌کردیم، ناچار بودیم دانش خود را در مورد ویژگی‌های هر جزء را به‌طور جداگانه تکرار کنیم. با منظور کردن ساختار کلاسی، تمام این ویژگی‌های مشترک را جمع می‌کنیم.

تعداد شیء‌ها و
کلاس‌ها.
رفع افزونگی.



شکل ۱-۱: فرم متعارف یک سیستم پیچیده

سیستم پیچیده‌ی
موفق

بر اساس تجربه‌ی ما سیستم‌های پیچیده‌ی نرم‌افزاری موفق، آنهایی هستند که طرحشان صراحتاً دارای ساختارهای کلاسی، و شیء خوب (well engineered) بوده و به‌علاوه پنج ویژگی سیستم‌های پیچیده را دربر داشته باشد. به‌عبارت

دیگر به ندرت سیستم نرم‌افزاری را دیده‌ایم که بدون این نکات، به موقع، در محدوده‌ی بودجه‌ی پیش‌بینی شده و مطابق با خواسته‌های تعیین شده، ایجاد شده باشد.

۳-۱ محدودیت‌های انسان در برخورد با پیچیدگی‌ها

(The Limitation of Human Capacity for Dealing with Complexity)

این سؤال مطرح است که اگر می‌دانیم طرح سیستم‌های نرم‌افزاری چگونه باید باشد، پس چرا در ایجاد موفق چنین سیستم‌هایی با مسائل جدی روبرو هستیم؟ همچنان که در فصل دوم بحث خواهیم کرد، این مفهوم پیچیدگی سازمان یافته‌ی (organized complexity) نرم‌افزار، که ما مجموعه‌ی اصول اولیه‌اش را مدل شیء (object model) می‌نامیم، نسبتاً جدید است. با این حال عامل دیگری که حکم‌فرمایی می‌کند محدودیت بنیادی انسان در برخورد با پیچیدگی است.

متأسفانه این غیر ممکن است که یک فرد تنها تمام جزئیات را با هم بتواند در نظر بگیرد [12]، [13]. ما با یک معمای بنیادی روبرو هستیم:

پیچیدگی نرم‌افزارهای مورد تقاضا هر روز بیشتر می‌شود. در حالی که ما در برخورد با این پیچیدگی‌ها بطور بنیادی محدود هستیم. ما چه خواهیم کرد؟

۴-۱ به نظم آوردن آشفتگی (Bringing order to Chaos)

همچنانکه Dijkstra پیشنهاد می‌کند:

”روش چیره شدن بر پیچیدگی از زمان‌های قدیم شناخته شده است: فاصله بینانداز و پیروز شو“ [14]. در موقع طراحی یک سیستم نرم‌افزاری پیچیده، تفکیک آن به قسمت‌های کوچک و کوچکتر بسیار ضروری است. به گونه‌ای که بتوانیم هر یک را به‌طور مستقل مورد توجه قرار دهیم.

تفکیک الگوریتمیک در مقابل تفکیک شیء‌گرا

راه درست تفکیک یک سیستم بزرگ کدامست؟ به وسیله‌ی الگوریتم‌ها یا وسیله‌ی شیء‌ها؟. جواب صحیح اینست که هر دو دیدگاه مهم هستند: دیدگاه الگوریتمی، دیدگاه حوادث را برجسته می‌کند، و دیدگاه شیء‌گرا روی عامل‌ها (agent) تأکید می‌کند. (خواه عامل‌هایی که موجب یک عمل می‌شوند و خواه آنهایی که عملی رویشان انجام می‌شود). با این حال این حقیقت وجود دارد که ما نمی‌توانیم یک سیستم پیچیده را همزمان با هر دو راه بسازیم، بخاطر این که این دو دیدگاه عمود به یکدیگرند. ما باید تفکیک سیستم را با یکی از این دو دیدگاه (الگوریتمی یا شیء‌گرا) شروع کنیم و سپس از ساختار حاصل به‌عنوان چهارچوبی برای بیان دیدگاه دیگر استفاده کنیم.

دیدگاه الگوریتمی
دیدگاه شیء‌گرا.
عدم امکان
استفاده‌ی هم‌زمان
از دو دیدگاه
دلایل مدل شیء.

تجربه (ی نویسنده) ما را به این هدایت می‌کند که ابتدا دیدگاه شیء‌گرا را به کار بریم. به دلیل این که این دیدگاه کمک بیشتری برای سازمان دادن به پیچیدگی ذاتی سیستم‌های نرم‌افزاری می‌کند همچنان که این دو دیدگاه به ما کمک کرده است تا بتوانیم پیچیدگی سازمان یافته‌ی سیستم‌های پیچیده و متنوعی مثل کامپیوترها، گیاهان، کهکشان‌ها و موضوعات بزرگ اجتماعی را تبیین کنیم.

سازمان دادن به
پیچیدگی.
صرفه جویی در بیان.

تفکیک شیء‌گرا، سیستم‌های کوچکتری را از طریق استفاده‌ی مجدد از مکانیزم مشترک ارایه می‌کند، بنابراین موجب صرفه‌جویی مهم (از expression) می‌شود.

سیستم‌های شیء‌گرا انعطاف بیشتری در مقابل تغییرات دارند و لذا بهتر می‌توانند در طول زمان تغییر کنند به دلیل این که طراحی آنها بر اساس فرم‌های پایدار میانی است.

انعطاف در مقابل
تغییر.

تفکیک شیء‌گرا به‌طور قابل ملاحظه ریسک ساختن سیستم‌های نرم‌افزاری بزرگ را کم می‌کند، به دلیل این که طراحی آنها به گونه‌ای است که بتوانند از سیستم‌های کوچکتر که ما در مورد آنها مطمئن هستیم به سیستم‌های بزرگتر رشد یابند. به علاوه تفکیک شیء‌گرا مستقیماً پیچیدگی ذاتی نرم‌افزار را هدف قرار می‌دهد (با کمک به ما برای تصمیم‌گیری‌های هوشمندانه در مورد جدا کردن امور مورد توجه ما در یک فضای بزرگ از حالت‌ها).

تقلیل ریسک.
برخورد با پیچیدگی.

معنی طراحی (The Meaning of Design)

موستوو [37] (Mostove) هدف از طراحی را ساخت سیستمی با خصوصیات زیر می‌داند:

- سیستمی که مشخصات و وظیفه‌ی خاصی را جواب‌گویی می‌کند؛
- جواب‌گویی محدودیت‌های محیط نهایی است؛
- جواب‌گویی خواسته‌های صریح و ضمنی در مورد عملکرد و استفاده از منابع است؛
- معیارهای طراحی صریح یا ضمنی را جواب‌گوست؛
- محدودیت‌های موجود روی خود فرایند طراحی، مثل زمان یا هزینه، یا ابزارهای موجود برای طراحی را ارضاء می‌کند.

اهمیت ساختن مدل (The Importance of Model Building)

ساختن مدل در بین نظام‌های مختلف مهندسی به‌طور وسیعی مورد قبول قرار گرفته است؛ عمدتاً به‌خاطر این که ساخت مدل به اصول تفکیک (decomposition)، تجرید (abstraction)، و سلسله مراتب (hierarchy) متوسل می‌شود [38].

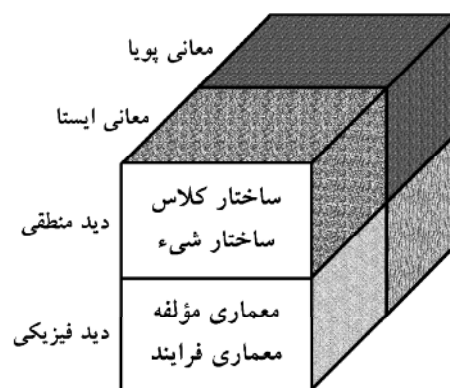
عناصر روش‌های طراحی نرم‌افزار (The Elements of Software Design Methods)

علی‌رغم وجود تفاوت‌ها در روش‌های مختلف طراحی نرم‌افزار (طراحی بالا به پایین، ساخت‌یافته؛ طراحی مبتنی بر داده‌ها - data driven؛ طراحی شیء‌گرا)، همگی عناصر مشترکی دارند:

- علامت‌گذاری (notation): زبان بیان هر مدل؛
- فرایند (process): رهنمودهایی برای ترتیب ساخت مدل‌ها؛
- ابزار (tools): وسیله‌ای که کار یکنواخت ساختن مدل را حذف، و قواعدی را در مورد خود مدل‌ها تحمیل می‌کند، به‌طوری‌که خطاها و ناسازگاری نمایان گردند.

۱-۵ مدل‌های طراحی شیء‌گرا (The Models of Object-Oriented Design)

به‌دلیل اهمیت ساختمان مدل در ساختن سیستم‌های پیچیده، طراحی شیء‌گرا مجموعه‌ای غنی از مدل‌ها را ارائه می‌کند. مدل‌های طراحی شیء‌گرا در شکل زیر خلاصه شده است:



شکل ۱-۲: مدهای طراحی شیء‌گرا

۱-۶ خلاصه‌ی فصل اول از کتاب طراحی شیء‌گرا / پیچیدگی (Complexity)

- نرم‌افزار ذاتاً پیچیده است؛ پیچیدگی سیستم‌های نرم‌افزاری غالباً فراتر از ظرفیت عقلانی انسان است؛
- کار گروه ایجاد و توسعه‌ی نرم‌افزار، مهندسیِ ارثه‌ی (illusion) سادگی است؛

- پیچیدگی غالباً فرم سلسله مراتب را می‌گیرد؛ درست کردن مدل‌های سلسله مراتبی "نوعی از" (kind of) و "جزئی از" (part of) برای سیستم‌های پیچیده مفید است؛
- سیستم‌های پیچیده عموماً حاصل فرم‌های پایدار میانی هستند؛
- محدودیت‌های اساسی برای درک انسان وجود دارد؛ با استفاده از تفکیک (decomposition)، تجرید (abstraction)، و سلسله مراتب (hierarchy) می‌توان این محدودیت را نشانه گرفت؛
- به سیستم‌های پیچیده هم با تأکید بر چیزها (things) و هم با تأکید بر فرایندها (processes) می‌توان نگاه کرد؛ دلایل قطعی برای تفکیک شیء گرا (object-orient decomposition) وجود دارد، که در آن ما به جهان به صورت مجموعه‌ای با معنی از شیء‌ها نگاه می‌کنیم، مجموعه‌ای که با یکدیگر همکاری می‌کنند تا رفتارهای برتر (higher level behavior) حاصل شود؛
- طراحی شیء گرا روشی است که ما را به تفکیک شیء گرا هدایت می‌کند؛ طراحی شیء گرا علامت‌گذاری (notation) و همچنین فرایندی (process) را برای ساختن سیستم‌های نرم‌افزاری بزرگ تعریف می‌کند، و مجموعه‌ای از مدل‌های منطقی و فیزیکی غنی را ارائه می‌کند، که توسط آن ما می‌توانیم درباره‌ی جنبه‌های مختلف سیستم مورد نظر، استدلال کنیم.

فصل ۲

مدل شیء (The Object Model)

اصول مدل شیء.

طراحی شیء گرا بر یک پایه‌ی مهندسی محکم (soundly)، که مجموعه‌ی عناصرش را مدل شیء می‌نامیم (object model) بنا شده است. مدل شیء دربرگیرنده‌ی اصول تجرید (abstraction)، دربرگیری (encapsulation)، پیمانه‌ای یا مؤلفه‌ای بودن (modularity)، سلسله‌مراتب (hierarchy)، نوع‌بندی (typing)، هم‌وجودی (concurrency)، و ماندگاری (persistence) می‌باشد. هیچ یک از اصول فوق به خودی خود جدید نیستند. نکته‌ی مهم در مورد مدل شیء اینست که این عناصر به صورت همکاری کننده در کنارهم آورده شده‌اند.

طراحی شیء گرا اصولاً با دیدگاه سنتی طراحی ساخت‌یافته متفاوت است، طراحی شیء گرا نیاز به تفکر دیگری در مورد تفکیک (decomposition) دارد؛ و معماری‌هایی از نرم‌افزار را ارائه می‌کند که از فرهنگ طراحی ساخت‌یافته بسیار فراتر است. این تفاوت ناشی از این حقیقت است که متد طراحی ساخت‌یافته بر اساس برنامه‌سازی ساخت‌یافته درست شده است؛ در حالی که طراحی شیء گرا بر اساس برنامه‌سازی شیء گرا. متأسفانه برنامه‌سازی شیء گرا معنی‌های متفاوتی برای افراد متفاوت دارد. در اینجا نشان می‌دهیم که طراحی شیء گرا چی هست و چی نیست. و چگونه با متدهای دیگر طراحی فرق می‌کند (به‌وسیله‌ی استفاده‌اش از عناصر هفتگانه‌ی مدل شیء).

۲-۱ عوامل شکل‌گیری مدل شیء (Foundations of the Object Model)

- پیشرفت در معماری کامپیوتر، از جمله سیستم‌های قابلیت (capability system) و حمایت سخت‌افزار از مفاهیم سیستم عامل؛
- پیشرفت در زبان‌های برنامه‌سازی، آنچنان که در Ada, CLU, smalltalk, simule ارائه شد؛
- پیشرفت در روش‌های برنامه‌سازی، از جمله عملکرد پیمانه‌ای (مؤلفه‌ای) (modularization) و پنهان کردن اطلاعات (information hiding)؛
- پیشرفت در مدل‌های بانک‌های اطلاعاتی؛
- تحقیق در هوش مصنوعی؛
- پیشرفت در فلسفه و علوم ادراکی (cognitive science).

عواملی که مدل شیء را موجب شدند.

۲-۲ برنامه‌سازی، طراحی و تحلیل شیء‌گرا (OOP, OOD, OOA)

برنامه‌سازی شیء گرا یک روش پیاده‌سازی است که در آن برنامه‌ها به صورت مجموعه‌ای از شیء‌ها که با یکدیگر همکاری می‌کنند، سازمان داده شده‌اند. هر یک از مجموعه‌ی شیء‌ها نمایشگر یک رویداد (instance) از رده (class)

هستند، و رده یا کلاس‌ها اعضاء سلسله مراتبی از کلاس‌ها هستند که توسط روابط وراثت (inheritance relationship) متحد شده‌اند.

طراحی شیء‌گرا (object-oriented design) یک روش طراحی است که فرایند تفکیک شیء‌گرا (object oriented decomposition) و همین یک علامت‌گذاری برای رسم مدل‌های منطقی و فیزیکی و همینطور مدل‌های ایستا و پویا از سیستم مورد نظر را دربر می‌گیرد.

“Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic model of the system under design.”

تحلیل شیء‌گرا (object-oriented analysis) یک روش تحلیل است که خواسته‌ها (requirements) را از نقطه نظر کلاس‌ها و شیء‌هایی که در لغت‌نامه‌ی زمینه‌ی مسأله یافت می‌شود، مورد بررسی قرار می‌دهد. نحوه‌ی ارتباط این‌ها با یکدیگر: اصولاً محصول تحلیل شیء‌گرا به‌عنوان مدلی که از آن می‌توانیم طراحی شیء‌گرا را آغاز کنیم می‌تواند عمل کند؛ سپس محصول طراحی شیء‌گرا می‌تواند برای پیاده‌سازی کامل سیستم، با استفاده از روش‌های برنامه‌سازی شیء‌گرا مورد استفاده قرار گیرد.

۲-۳ ارکان مدل شیء (Elements of Object Model)

مدل شیء، چهارچوب مفهومی (conceptual framework) هر چیز شیء‌گراست. این مدل شیء چهار عنصر یا رکن اصلی دارد:

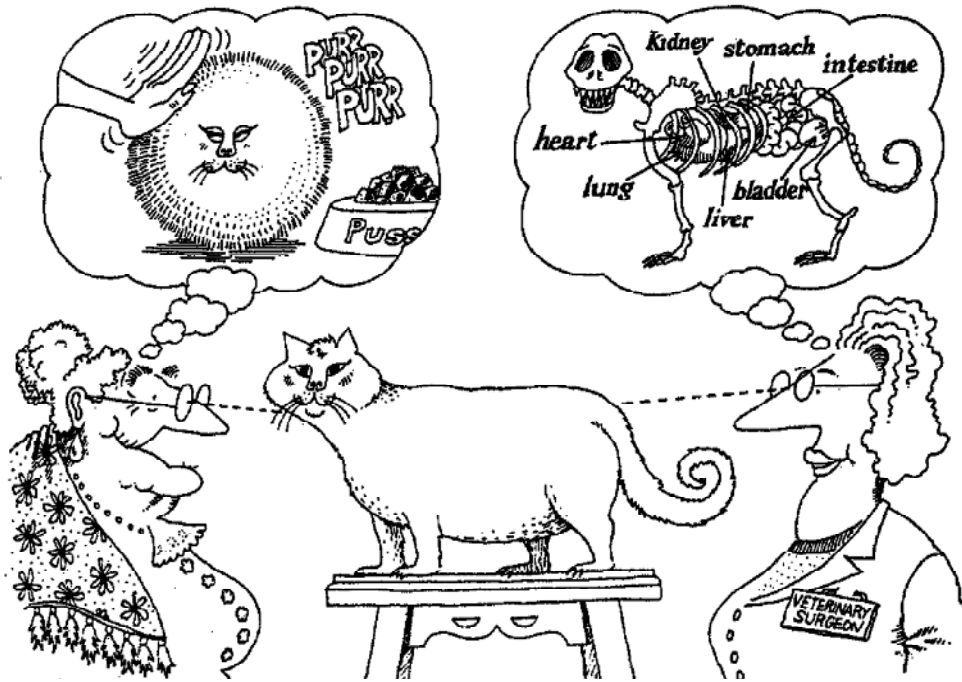
- تجرید (abstraction)؛
 - دربرگیری (encapsulation)؛
 - مؤلفه‌ای یا پیمان‌های بودن (modularity)؛
 - سلسله مراتب (hierarchy).
- منظور از “اصلی” اینست که مدلی که هر یک از ارکان فوق را نداشته باشد شیء‌گرا نیست. مدل شیء دارای سه رکن یا سه عنصر فرعی نیز می‌باشد:
- نوع بندی (typing)؛
 - توازی / هم‌وجودی (concurrency)؛
 - ماندگاری (persistence).

منظور از “فرعی” اینکه هر یک از اینها برای مدل شیء مفید هستند، اما ضروری نیستند.

بدون این چهارچوب مفهومی، شما ممکن است با زبانی مثل Smalltalk, Object Pascal, C++, CLOS یا Ada برنامه‌نویسی کنید، ولی طراحی شما بوی طراحی شبیه FORTRAN, Pascal ویا C را داشته باشد. در اینصورت شما قدرت بیان زبان شیء‌گرا یا زبان مبتنی بر شیء (object-based) را استفاده نکرده‌اید یا بد استفاده کرده‌اید. و احتمالاً بر پیچیدگی مسأله‌ی مورد نظرتان مسلط نشده‌اید.

تجرید (Abstraction)

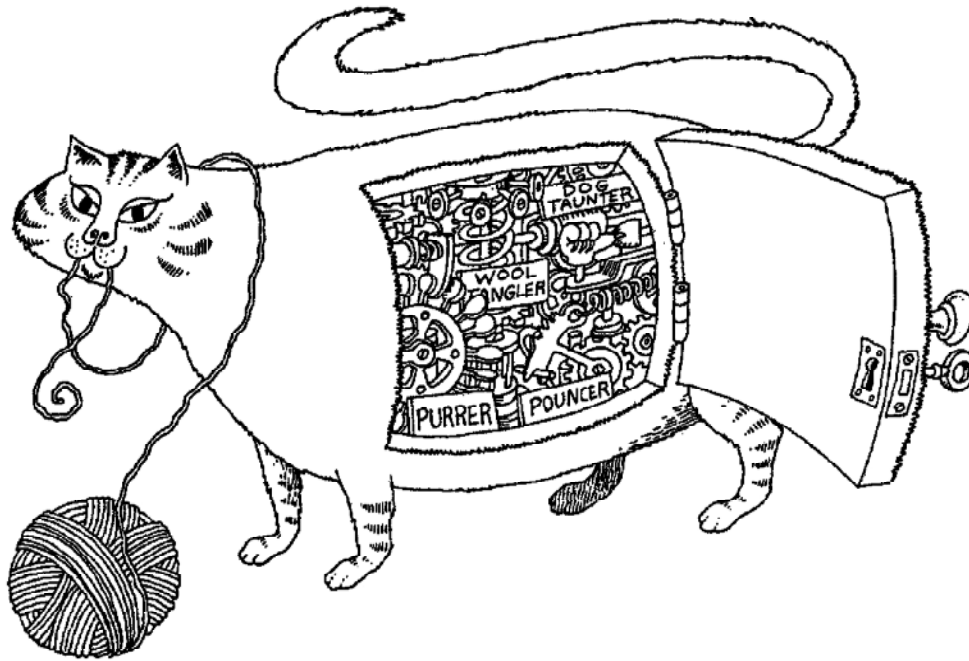
بسته به دیدگاه ناظر، یک تجرید مبین ویژگی‌های اساسی یک شیء است. که آن شیء را از انواع دیگر شیء متمایز می‌کند؛ و لذا حدود مفهومی (conceptual boundaries) تعریف شده‌ای را فراهم می‌کند.



شکل ۱-۲: تجرید روی ویژگی‌های اصلی شیء با توجه به دیدگاه ناظر، تکیه می‌کند

درب‌گیری (Encapsulation)

درب‌گیری، فرایند مخفی نگاه داشتن تمام جزئیات از یک شیء است که مشارکتی در ویژگی‌های اصلی شیء نداشته باشد.



شکل ۲-۲: درب‌گیری جزئیات پیاده‌سازی شیء را پنهان می‌کند

پیمانه‌ای بودن (مؤلفه‌ای) (Modularity)

پیمانه‌ای بودن خصوصیتی است که موجب تفکیک سیستم به مجموعه‌ای از مؤلفه‌های (modules) منسجم (cohesive) و با وابستگی اندک (loosely coupled) می‌گردد.



شکل ۲-۳: مؤلفه‌ای بودن، تجربدها را در واحدهای مجزا تفکیک می‌کند

سلسله مراتب (Hierarchy)

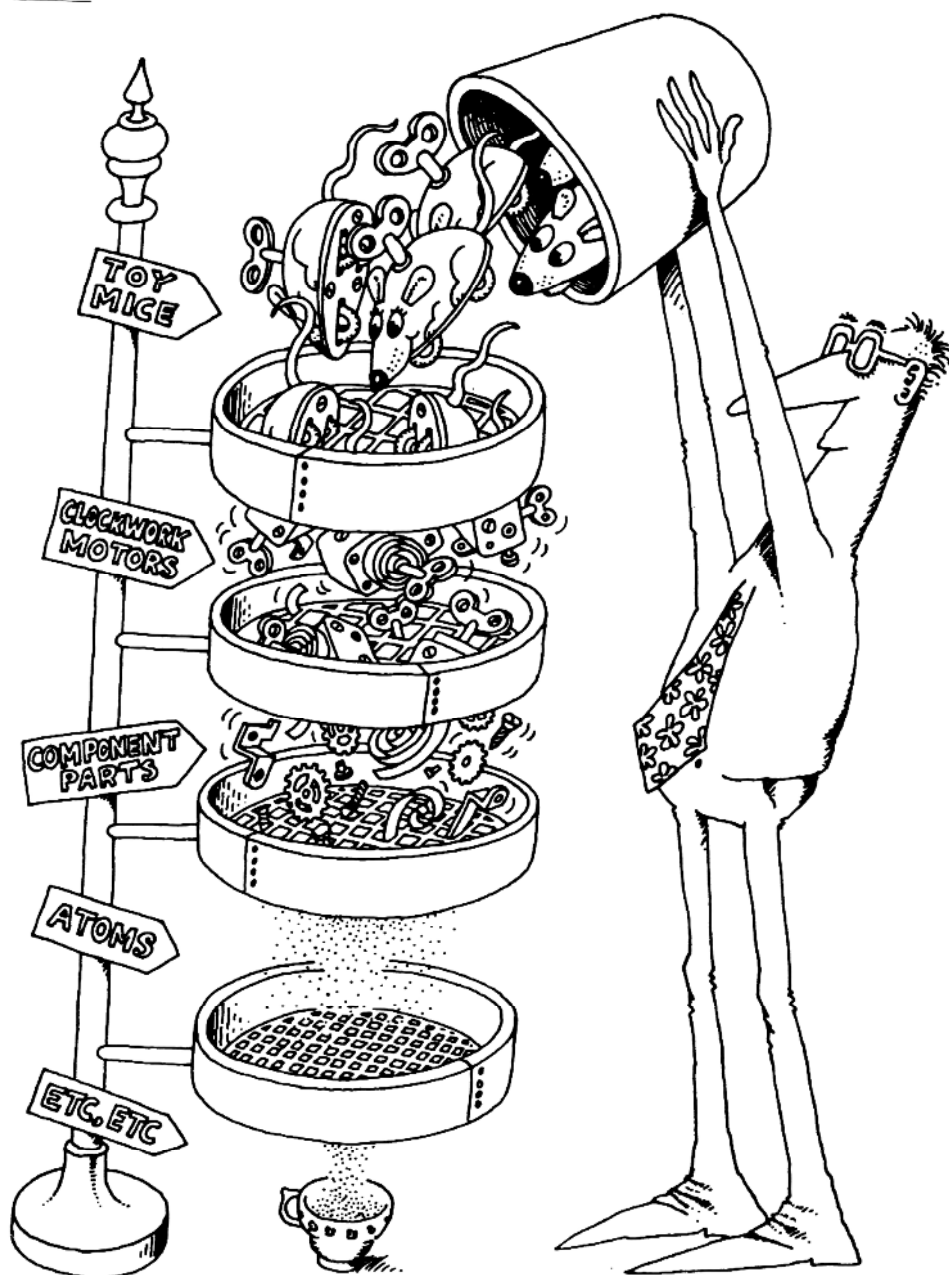
تجربدهای متعددی در یک زمان نتوانیم آنها را مورد شمول قرار دهیم (comprehend). در برگیری کمک می‌کند تا با پنهان کردن دید داخلی تجربدها، بتوانیم با پیچیدگی این تجربدهای متعدد برخورد کنیم. پیمانه‌ای بودن با تفکیک و خوشه‌بندی (clustering) تجربدهایی که منطقاً به یکدیگر مربوط هستند، کمک می‌کند، اما اینها هنوز کافی نیست. مجموعه‌ای از تجربدها معمولاً یک سلسله مراتب را تشکیل می‌دهند، و ما به وسیله‌ی تشخیص این سلسله مراتب‌ها در طراحی خود، به‌طور قابل ملاحظه‌ای فهم خود از مسأله را ساده می‌کنیم.

سلسله مراتب یک رتبه‌بندی و مرتب کردن تجربدها است.

در یک سیستم پیچیده مهمترین سلسله مراتب‌ها عبارتند از: ساختار کلاس آن سیستم (class structure) یا به عبارت دیگر سلسله مراتب "نوعی از" (the "kind of" hierarchy) و ساختار شیء (object structure) یا به عبارت دیگر سلسله مراتب "جزئی از" (the "part of" hierarchy) از آن سیستم.

وراثت مهمترین سلسله مراتب "نوعی از"، و یک عنصر اصلی از سیستم‌های شیء گرا می‌باشد.

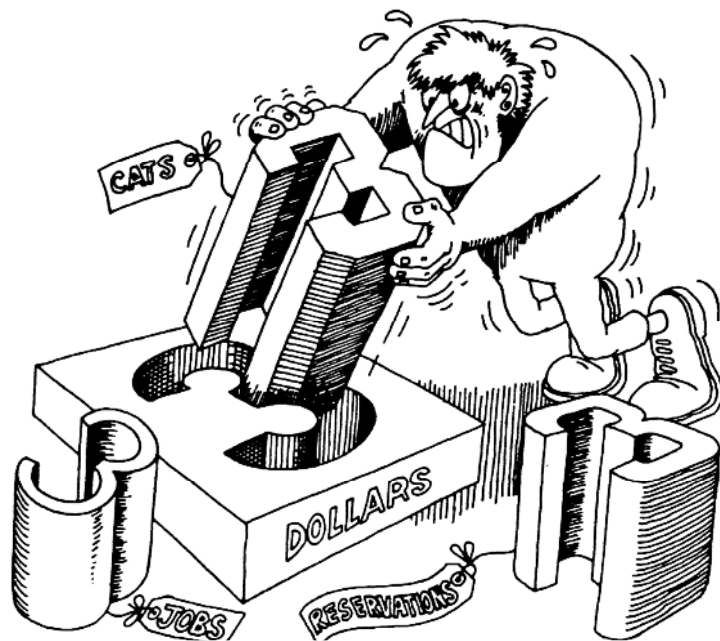
مثال از تسلسل: وراثت یگانه. وراثت یگانه (که سلسله مراتب یگانه single inheritance و سلسله مراتب چندگانه multiple inheritance نامیده می‌شود) شریک می‌شود. بنابراین وراثت نمایش دهنده‌ی سلسله مراتبی از تجربدها است.



شکل ۲-۴: تجزیه‌ها یک سلسله‌مراتب را تشکیل می‌دهند

نوع‌بندی (Typing)

همچنان‌که Deutch پیشنهاد می‌کند، "نوع، توصیفی دقیق از خصوصیات ساختاری یا رفتاری است که در مجموعه‌ای از موجودیت‌ها مشترک است" [65]. در اینجا ما از واژه‌های نوع و کلاس را به صورت معادل هم استفاده می‌کنیم. نوع‌بندی تحمیل (enforcement) کلاسی از شیء‌هاست، به ترتیبی که تعویض شیء‌ها از انواع مختلف با یکدیگر ممکن نباشد، یا این‌که حداکثر به صورت‌های کاملاً محدودی با یکدیگر تعویض گردند.



شکل ۲-۵: نوع‌بندی قوی^۱ از تداخل تجربه‌ها جلوگیری می‌کند

توازی یا هم‌وجودی (Concurrency)

برای بعضی از مسأله‌ها، ممکن است یک سیستم خودکار، حوادث متعددی را به‌طور هم‌زمان دنبال نماید. در بعضی از مسائل حجم محاسبات ممکن است بیشتر از توان اجرایی یک پردازنده باشد. در هر دو حالت استفاده از مجموعه‌ای از کامپیوترهای توزیع شده (distributed) یا ریزپردازنده‌ها، طبیعی به نظر می‌رسد. (در سیستم‌هایی که تنها روی یک پردازنده اجرا می‌شوند، داشتن چندین ریسمان کنترل موازی (concurrent threads of control) به وسیله الگوریتم‌های برهه‌بندی زمان پردازنده (time sliding) نمود می‌یابد.

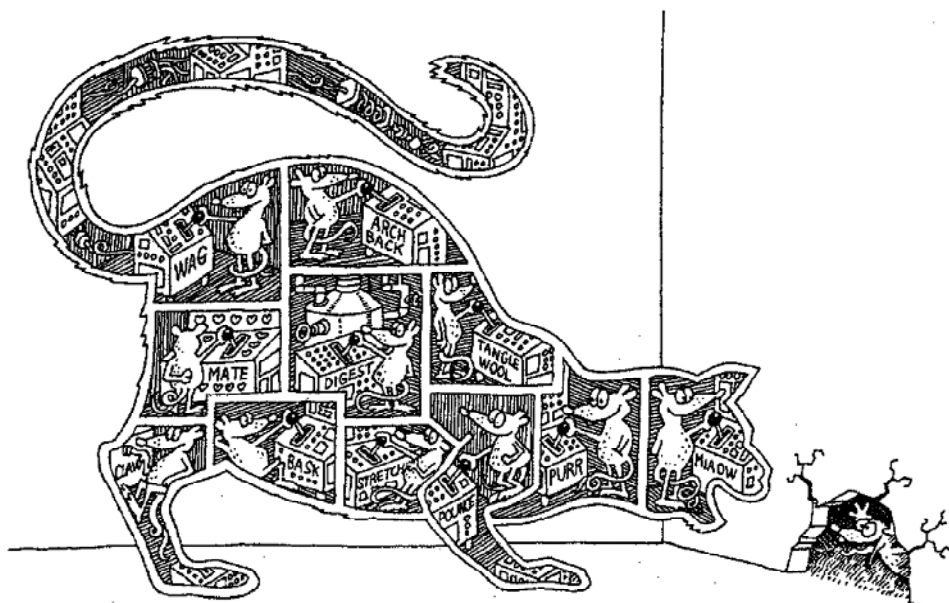
در بالاترین سطح تجزید، برنامه‌سازی شیء‌گرا می‌تواند مسأله توازی (هم‌وجودی) را با پنهان کردن توازی در داخل تجزیدهای با قابلیت استفاده‌ی مجدد (reusable abstraction)، برای اغلب برنامه‌سازان سبکتر کند [72].

در حالی که برنامه‌سازی شیء‌گرا روی تجزید داده‌ها، دربرگیری و وراثت تکیه می‌کند، توازی (هم‌وجودی) روی "تجزید فرارونه‌ها" (process abstraction) و هماهنگی آنها (synchronization) تأکید می‌کند [74].

شیء مفهومی است که این دو دیدگاه مختلف را یکی می‌کند: هر شیء (که از تجزید دنیای واقع به‌دست آمده است) ممکن است یک ریسمان کنترل جداگانه (یک تجزید فرارونه) را نمایش دهد. چنین شیء‌هایی فعال (active) نامیده می‌شوند. در سیستمی که بر اساس طراحی شیء‌گرا می‌باشد، ما می‌توانیم دنیا را به‌صورت مجموعه‌ای از شیء‌ها که با هم همکاری می‌کنند، تصور کنیم؛ شیء‌هایی که برخی از آنها فعال هستند و لذا به‌عنوان مراکزی از فعالیت‌های مستقل عمل کنند. با توجه به این مطالب، توازی را به‌صورت زیر تعریف می‌کنیم:

توازی (هم‌وجودی) خاصیتی است که یک شیء فعال را از یک شیء غیر فعال تمیز می‌دهد.

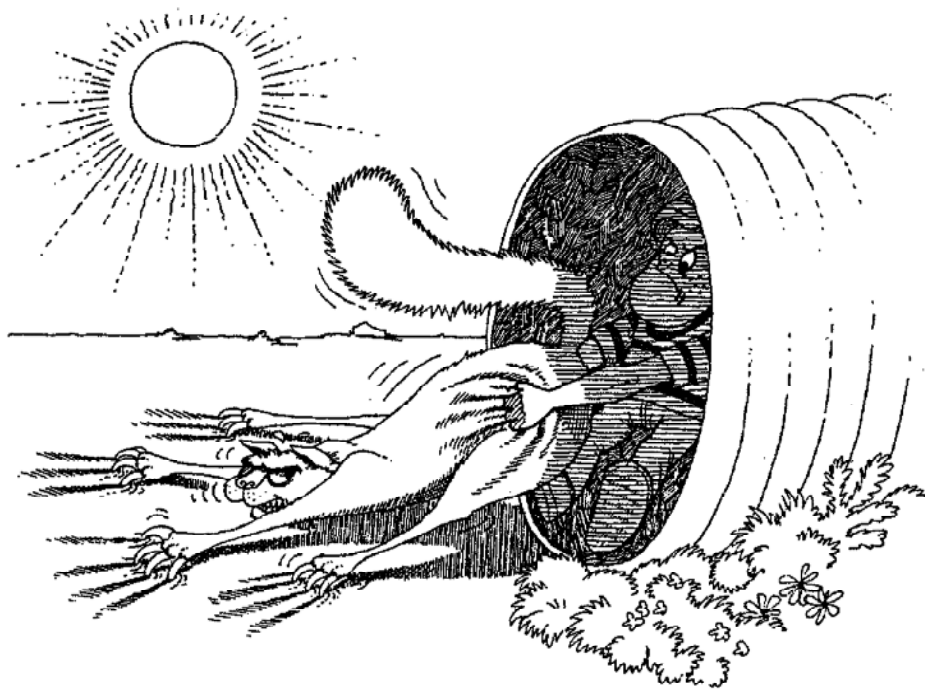
¹ strong typing



شکل ۲-۶: توازی اجازه می‌دهد تا شیء‌های مختلف در یک زمان عمل کنند

ماندگاری (Persistence)

در نرم‌افزار، یک شیء مقداری از فضا را اشغال کرده و برای مقدار مشخصی از زمان وجود دارد. ماندگاری خصوصیتی از شیء است که به واسطه‌ی آن وجود شیء بر زمان غلبه می‌کند (یعنی شیء بعد از توقف حیات به وجود آورنده‌اش به حیات خود ادامه می‌دهد) و/یا وجود شیء بر فضا غلبه می‌کند (یعنی محل شیء از فضای آدرس‌هایی که در ابتدا به وجود آمده بود، حرکت می‌کند).



شکل ۲-۷: ماندگاری حالت^۱ و کلاس شیء را در رابطه با زمان و مکان حفظ می‌کند

^۱ state

کاربردهای مدل شیء (Applications of the Object Model)

مدل شیء ثابت کرده است که در زمینه‌های متعددی قابل به کارگیری است (جدول زیر). طراحی شیء‌گرا شاید تنها روشی باشد که امروز برای مقابله با پیچیدگی سیستم‌های خیلی بزرگ در اختیار ماست. ممکن است طراحی شیء‌گرا برای بعضی زمینه‌ها خیلی قابل توجه نباشد. البته نه به دلایل تکنیکی، بلکه به دلایل غیر تکنیکی مثل عدم وجود افراد آموزش دیده مناسب، یا عدم وجود محیط ایجاد و توسعه‌ی (development) مناسب.

جدول ۲-۱: کاربردهای مدل شیء

Air traffic control	Document preparation	Petroleum engineering
Animation	Expert system	Reusable software components
Avionics	Film and stage storyboarding	Robotics
Banking and insurance software	Hypermedia	Software development environments
Business data processing	Inane recognition	Space station software
Chemical process control	Investment strategies	Spacecraft and aircraft simulation
Command and control systems	Mathematical analysis	Telecommunications
Computer aided design	Medical electronics	Telemetry systems
Computer aided education	Office automation	User interface design
Computer integrated manufacturing	Operating system	VLSI design
Database		

۲-۴ خلاصه‌ی فصل دوم از کتاب طراحی شیء‌گرا / مدل شیء (The Object Model)

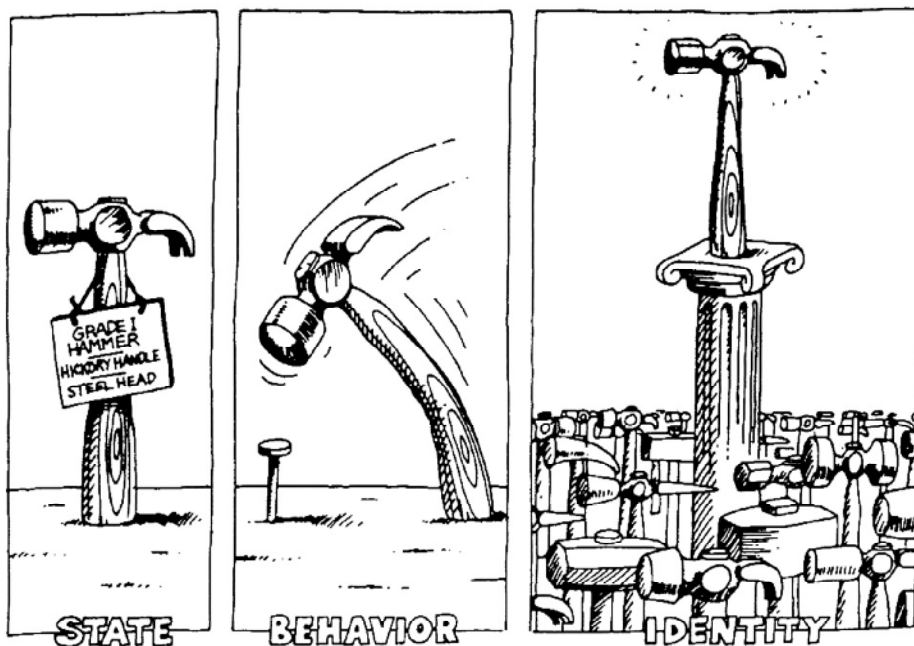
- بلوغ مهندسی نرم‌افزار منجر به روش‌های تحلیل، طراحی و برنامه‌سازی شیء‌گرا شده است که همگی برنامه‌سازی در مقیاس بزرگ را نشانه گرفته‌اند؛
- طُرق مختلف برنامه‌سازی وجود دارد: رویه‌گرا (procedure-oriented)، شیء‌گرا (object-oriented)، منطق‌گرا (logic-oriented) و محدودیت‌گرا (constraint-oriented)؛
- مدل شیء چهارچوب مفهومی برای روش‌های شیء‌گرا را فراهم می‌کند؛ مدل شیء دربرگیرنده‌ی اصول زیر است: تجرید (abstraction)، دربرگیری (encapsulation)، پیمانه‌ای بودن (modularity)، سلسله‌مرتب (hierarchy)، نوع بندی (typing)، توازی / هم‌وجودی (concurrency)، و ماندگاری (persist tance)؛
- تجرید، مبین خصوصیات اصلی یک شیء است که آنرا از تمام انواع شیء‌های دیگر متمایز می‌کند و در نتیجه محدوده‌های مفهومی کاملاً تعریف شده‌ای، را با توجه به دید ناظر، فراهم می‌کند؛
- دربرگیری، فرایند پنهان کردن تمام جزئیات شیء است که در خصوصیات اصلی شیء نقشی ندارند؛
- پیمانه‌ای بودن، خصوصیتی از یک سیستم است که منجر به تفکیک سیستم به مجموعه‌ای از مؤلفه‌های منسجم و با وابستگی کم شده است؛
- سلسله‌مراتب، یک رتبه‌بندی یا مرتب کردن تجریدهاست؛
- نوع‌بندی، تحمیل کلاس شیء‌هاست، به‌ترتیبی که شیء‌های نوع‌های متفاوت با یکدیگر نتوانند عوض شوند، یا حداکثر این که تنها از طُرق بسیار محدود عوض شوند؛
- توازی (هم‌وجودی)، خصوصیتی است که یک شیء فعال را از شیء غیر فعال متمایز می‌کند؛
- ماندگاری خصوصیتی از یک شیء است که از طریق آن، وجود شیء از زمان و / یا مکان فراتر می‌رود؛
- کاربرد مدل شیء منجر به سیستم‌هایی می‌شود که پنج ویژگی سیستم‌های پیچیده‌ی کاملاً ساخت‌یافته را دارد.

فصل ۳

کلاس‌ها و شیء‌ها (Classes & Objects)

۱-۳ ماهیت شیئی (The Nature of an Object)

شیء دارای حالت (state)، رفتار (behavior) و هویت (identity) است؛ ساختار (structure) و رفتار شیء‌های مثل هم در کلاس (class) مشترک آنها تعریف می‌شود؛ واژه‌های رویداد (instance) و شیء، قابل تعویض با یکدیگرند.



شکل ۳-۱: شیء دارای حالت است، رفتارهای کاملاً تعریف شده‌ای را از خود نشان می‌دهد و دارای هویت (شناسه) منحصر به فردی است

حالت شیء (State)

حالت یک شیء دربرگیرنده‌ی تمام خاصیت‌های (معمولاً ایستا) آن شیء و همین‌طور مقادیر (معمولاً پویا) جاری هر یک از این خاصیت‌ها است.

رفتار شیء (Behavior)

هیچ شیء به صورت منفرد و منزوی وجود ندارد. شیء‌ها مورد تأثیر شیء‌های دیگر قرار می‌گیرند و یا شیء‌های دیگر را مورد تأثیر قرار می‌دهند. لذا می‌توان گفت:

رفتار یک شیء عبارت از اینست که چگونه عمل می‌کند و یا عکس‌العمل نشان می‌دهد، بر حسب تغییرات حالتش و ارسال پیام (message passing). به عبارت دیگر، رفتار شیء کاملاً به وسیله‌ی عمل‌هایش (its action) تعریف می‌شود [7].

در زبان‌هایی مثل Smalltalk، صحبت از این می‌کنیم که یک شیء به شیء دیگر پیام می‌فرستد. عموماً یک پیام، عملی است که یک شیء روی شیء دیگر انجام می‌دهد، گرچه مکانیزم به کار رفته متفاوت است. برای منظور ما، واژه‌ی عمل (operation) و پیام (message) می‌توانند به جای یکدیگر بکار روند. در زبان‌های برنامه‌سازی شیء‌گرا (object-oriented) و مبتنی بر شیء (object-based) عملیاتی که مشتری‌ها (clients) روی شیء‌ها ممکن است انجام دهند، نوعاً به صورت روش‌ها (methods) که قسمتی از کلاس آن شیء است تعریف می‌شوند. ++C از واژه‌ی تابع عضویت (member function) برای بیان همین مفهوم استفاده می‌کند، لذا ما از واژه‌های روش (method) و تابع عضویت (member function) به صورت قابل تعویض با یکدیگر استفاده می‌کنیم.

اعمال متداول روی شیء‌ها (معنی عملیات)

در عمل نوعاً یک مشتری (client) پنج عمل زیر را روی یک شیء انجام می‌دهد:

- تغییر دهنده (modifier): عملی که حالت (state) یک شیء را تغییر می‌دهد. (عمل نویسنده)؛
- انتخاب‌کننده (selector): عملی که به حالتی از شیء دسترسی پیدا می‌کند ولی آن حالت را تغییر نمی‌دهد. (عمل خواننده)؛
- تکرارگر (integrator): عملی که اجازه می‌دهد تمام قسمت‌های شیء با یک ترکیب کاملاً مشخصی، مورد دسترسی قرار گیرند.

در بعضی زبان‌های برنامه‌سازی مثل Smalltalk, ++C, CLOS معرفی دو نوع دیگر از عملیات ممکن است:

- سازنده (constructor): عملی که یک شیء را به وجود می‌آورد / یا به حالتش مقدار اولیه می‌دهد؛
- تخریب‌کننده (destructor): عملی که حالت یک شیء را رها (free) می‌کند یا خود شیء را نابود می‌کند.

شیء‌ها به عنوان ماشین

وجود حالت در شیء این معنی را می‌دهد که ترتیب عملیات مهم است. این نکته، این مطلب را به ذهن می‌آورد که هر شیء مثل یک ماشین مستقل کوچک است. در واقع برای بعضی شیء‌ها، این ترتیب زمانی عملیات آنقدر فزاینده و نافذ است که می‌توانیم رفتار شیء را به وسیله‌ی ماشین با حالت‌های محدود (finite state machine) مشخص کنیم.

شیء‌های فعال و منفعل (Active and passive object)

شیء‌ها ممکن است فعال (active) و یا منفعل (passive) باشند. شیء فعال آن است که ریسمان کنترلی (thread of control) مال خود را داشته باشد، در حالی که شیء منفعل این‌طور نیست. شیء‌های فعال عموماً خودمختار (autonomous) هستند. بدین معنی که بدون این که شیء دیگری روی آنها عملی انجام دهد، خودش می‌تواند رفتاری را نشان دهد. از سوی دیگر شیء‌های منفعل (passive) تنها وقتی که صراحتاً شیء دیگری روی آنها عمل می‌کند، دست‌خوش یک تغییر حالت شوند. بدین ترتیب شیء‌های فعال به عنوان بنیان کنترل در سیستم‌ها عمل می‌کنند. اگر سیستم ما شامل چند ریسمان کنترل باشد، در این صورت ما معمولاً چند شیء فعال داریم. سیستم‌های ردیفی دقیقاً تنها یک شیء فعال در هر لحظه دارند (یعنی دقیقاً یک ریسمان کنترل).

هویت / شناسه (Identity)

هویت، خاصیتی از شیء است که آن شیء را از تمام شیء‌های دیگر متمایز می‌کند [10].
عدم تمایز اسم یک شیء و خود آن شیء، عامل بسیاری از خطاها در زبان‌های برنامه‌سازی شیء‌گرا است.

عامل بسیاری از خطاها.

۲-۳ انواع روابط شیء‌ها با یکدیگر (Kinds of Relationships)

رابطه‌ی بین دو شیء دربرگیرنده‌ی تمام فرضیاتی است که یک شیء از شیء دیگر دارد، از جمله این که، چه عمل‌هایی می‌توان انجام داد و این که چه رفتارهایی در مقابل انجام خواهد شد. دو نوع سلسله‌مراتب شیء‌ها در طراحی شیء‌گرا مورد توجه است:

- رابطه‌ی به‌کارگیری (using relationship) یا ارشدیت (seniority) [14]، که یک شیء، شیء دیگر را به‌کار می‌برد؛ رابطه‌ی به‌کارگیری.
- رابطه‌ی دربرداشتن (containing relationship) یا رابطه‌ی پدر، فرزند (parent / child) که یک شیء در داخل شیء دیگر قرار می‌گیرد. رابطه‌ی دربرداشتن.

نقش شیء (The Role of object)

در هر رابطه‌ی استفاده در بین شیء‌ها، هر شیء ممکن است یکی از سه نقش زیر را بازی کند:

- فعال یا خواهان (actor): شیئی که می‌تواند روی شیء‌های دیگر عمل کند، ولی هیچ‌گاه به‌وسیله‌ی شیء‌های دیگر مورد عمل قرار نگیرد؛ واژه‌های actor, active, object معادل یکدیگر گرفته می‌شوند؛
- خدمتگذار (server): شیئی که هرگز روی شیء‌های دیگر عمل نمی‌کند؛ تنها به‌وسیله‌ی شیء‌های دیگر مورد عمل قرار می‌گیرد؛
- واسطه (agent): شیئی که هم می‌تواند روی شیء‌های دیگر عمل کند و هم توسط شیء‌های دیگر مورد عمل قرار گیرد؛ معمولاً یک شیء واسطه برای انجام کاری از طرف یک شیء فعال (actor) یا شیء واسطه‌ی (agent) دیگر به‌وجود می‌آید.

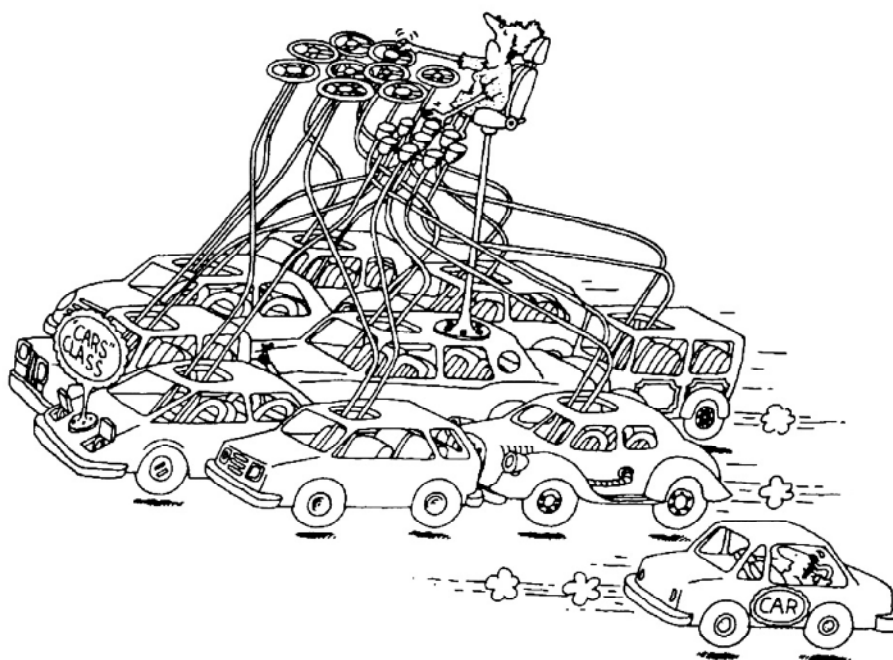
معنی همگام‌سازی شیء‌ها (The Meaning of Synchronization)

هرگاه یک شیء به شیء دیگری که با آن رابطه‌ی استفاده (using relationship) دارد پیامی می‌فرستد، این دو شیء باید به‌گونه‌ای همگام شوند. برای شیء‌هایی که در یک کاربرد کاملاً ردیفی هستند، این همگام‌سازی معمولاً به‌وسیله‌ی فراخوانی زیربرنامه‌ها صورت می‌گیرد. با این‌حال برای شیء‌هایی که در چند ریسمان کنترل (multiple thread of control) درگیر هستند، همگام‌سازی پیچیده‌تری برای برخورد با مسأله ممانعت دوجانبه (mutual exclusion) که در سیستم‌های هم‌وجود (concurrent) پیش می‌آید، لازم است. این مطلب ما را به نوع دیگری از طبقه‌بندی شیء‌ها هدایت می‌کند:

- شیء ردیفی (sequential object): یک شیء منفعل (passive) که معانی آن تنها با وجود یک ریسمان کنترل ضمانت می‌شود؛
- شیء مسدود (Blocking object): یک شیء منفعل که معانی آن با وجود چند ریسمان کنترل ضمانت می‌شود؛
- شیء هم‌وجود (concurrent object): یک شیء فعال (active) که معانی آن با وجود چند ریسمان کنترل ضمانت می‌شود.

۳-۳ ماهیت کلاسی (The Nature of a Class)

در حالی که شیء یک موجودیت مشخص و موجود در زمان و مکان است، کلاس تنها نمایشگر یک تجرید است. کلاس مجموعه‌ای از شیء‌هاست، شیء‌هایی که داری ساختار و رفتار مشترک هستند.



شکل ۳-۲: کلاس نمایشگر مجموعه‌ای از شیء‌ها با ساختار و رفتار مشترک است

دیدگاه بیرونی و دیدگاه داخلی یک کلاس (The Outside and Inside Views of a Class)

- واسطه (interface) یک کلاس بیانگر دید خارجی است. بنابراین تأکید بر تجرید (abstraction) دارد. این در حالی است که ساختار و اسرار رفتار را پنهان می‌کند. این واسطه به‌طور عمده شامل تعاریف (declaration) تمام اعمالی است که روی تمام رویدادهای (شیء‌های) این کلاس قابل اعمال باشد. در عین حال این واسطه ممکن است حاوی تعریف کلاس‌ها، ثابت‌ها، متغیرها و موارد استثنای دیگر برای تکمیل تجرید باشد؛
- پیاده‌سازی (implementation) یک کلاس بر خلاف واسطه (interface)، دیدگاه درونی یک کلاس است. دیدگاه درونی در برگیرنده‌ی اسرار و رفتار شیء می‌باشد. پیاده‌سازی کلاس به‌طور عمده مشتمل بر پیاده‌سازی تمام عمل‌های (operation) تعریف شده در واسطه آن کلاس است.

تقسیم بندی واسطه کلاس

- می‌توانیم دیدگاه بیرونی یا واسطه (interface) از کلاس را به سه قسمت زیر تقسیم کنیم:
- همگانی (public): تعریفی که بخش قابل مشاهده از واسطه کلاس را برای تمام مشتری‌های (clients) این کلاس معرفی می‌کند. البته مشتری‌هایی که برایش قابل رؤیت هستند؛
 - حفاظت شده (protected): تعریفی که بخشی از واسطه که تنها برای زیر کلاس‌ها (sub classes) قابل مشاهده هستند را معرفی می‌کند؛
 - خصوصی (private): تعریفی که بخشی از واسطه که برای هیچ کلاس دیگر قابل مشاهده نیست را معرفی می‌کند. از زبان‌هایی که استفاده می‌کنیم، ++C بهترین کار را می‌کند که به تولیدکننده‌ی نرم‌افزار اجازه می‌دهد، صراحتاً این قسمت‌های متفاوت از واسطه کلاس را متمایز سازد.

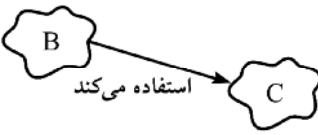
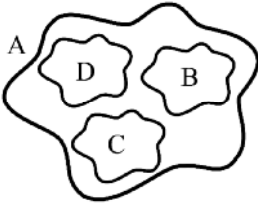
۳-۴ انواع روابط کلاس‌ها با یکدیگر (Kinds of relationship)

اکثر زبان‌های برنامه‌سازی شیء‌گرا یا مبتنی بر شیء، ترکیبی از روابط زیر بین کلاس‌ها را حمایت می‌کنند:

- رابطه‌ی وراثت (inheritance relationships):
یگانه؛
چندگانه.
- رابطه‌ی استفاده (using relationships):
واسطه یک کلاس از کلاس دیگر استفاده می‌کند؛
پایه‌سازی یک کلاس، از کلاس دیگر استفاده می‌کند. کلاس استفاده شده در کلاس استفاده‌کننده پنهان خواهد بود.
- رابطه‌ی رویداد (instantiation relationship)
- رابطه‌ی کلاس برتر / آبر کلاس (meta class relationship).

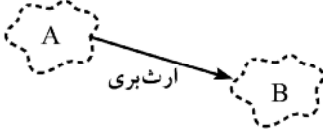
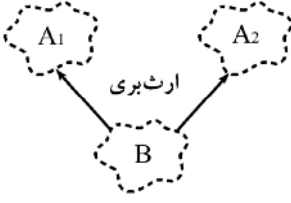
خلاصه در مورد روابط شیء‌ها

جدول ۱-۳: روابط شیء‌ها

استفاده:	دربرداشتن:
	
شیء A از شیء B استفاده می‌کند.	شیء A شیء‌های B، C، و D را دربردارد.

خلاصه در مورد روابط کلاس‌ها

جدول ۲-۳: روابط کلاس‌ها

ارث بری یگانه:	ارث بری چندگانه:
	
کلاس B از کلاس A ارث می‌برد (ارث بری یگانه).	کلاس B از کلاس‌های A ₁ و A ₂ ارث می‌برد (ارث بری چندگانه).

جدول ۳-۳: روابط کلاس‌ها

استفاده (در پیاده سازی):	استفاده (در واسط):
<p>کلاس مورد استفاده برای مشتری‌های کلاس استفاده‌کننده نباید قابل رویت باشد. کلاس مورد استفاده بخشی از اسرار کلاس استفاده‌کننده است.</p> <p>کلاس انسان، هیئت علمی (در پیاده سازی عملکرد خود) از کلاس قلب‌ها استفاده می‌کند (یک عضو هیئت علمی از یک کتاب).</p> <p>کلاس عضو هیئت علمی (در پیاده‌سازی عملکرد خود) از کلاس کتاب استفاده می‌کند (یک عضو هیئت علمی از $n \geq 0$ کتاب).</p>	<p>کلاس مورد استفاده، برای مشتری‌های کلاس استفاده‌کننده نیز باید قابل رویت باشند.</p> <p>کلاس دانشجو از کلاس هیئت علمی (در بخش واسط) استفاده می‌کند (m دانشجو از n هیئت علمی).</p>

جدول ۴-۳: روابط کلاس‌ها

رویداد (Instantiation):	آبر کلاس (Meta class):
<p>کلاس انباره‌های مربوط به template رویدادی از کلاس عمومی stack است.</p>	<p>B با A رابطه‌ی آبر کلاسی دارد، به این معنی که کلاس B، به‌عنوان یک شیء عضو کلاس A است.</p>

تذکر: گرچه C++ صراحتاً از آبر کلاس حمایت نمی‌کند ولی برای متغیرها و روش‌های کلاس امکاناتی را فراهم کرده‌است. شخص می‌تواند یک شیء عضو (member object) یا تابع عضو (member function) را به‌صورت static معرفی کند، به این معنی که چنین عضوهایی بین تمام رویدادهای آن کلاس مشترک است (مثل این که بخواهیم تعدادی ساعت مختلف داشته باشیم ولی همگی آنها دقیقه و ثانیه‌ی یکسانی را نمایش دهند).

در Java از رابطه‌ی آبر کلاس عملاً زیاد استفاده شده است.

چند شکلی (Polymorphism)

اصولاً چند شکلی مفهومی از تئوری نوع (type theory) می‌باشد. در مفهوم چند شکلی، یک اسم ممکن است به معنی شیء‌هایی از کلاس‌های مختلف باشد؛ کلاس‌هایی که به‌وسیله‌ی یک سوپر کلاس (super class) به یکدیگر مرتبط هستند. بنابراین هر شیء که با این اسم مشخص شده است قادر به پاسخ‌گویی مجموعه‌ای از عمل‌ها به‌صورت‌های مختلف باشد.

زبان‌هایی مثل Pascal، بر اساس این ایده هستند که، توابع، رویه‌ها و بنابراین عملگرها دارای نوع منحصر به فرد هستند. چنین زبان‌هایی یک شکلی (mono morphic) نامیده می‌شوند.

بارگذاری اضافی (Over loading)

در زبان‌های برنامه‌نویسی جدید، این که بتوان برای علامتی مثل "+" بیش از یک معنی تعریف کرد را بارگذاری اضافی (over loading) می‌گویند.

۳-۵ نقش متقابل کلاس‌ها و شیء‌ها (The Interplay of Classes and Objects)

ویژگی‌های کلاس‌ها
ویژگی‌های شیء‌ها

کلاس‌ها و شیء‌ها مفاهیمی مجزا و در عین حال مرتبط به هم هستند. به خصوص هر شیء رویدادی از یک یا چند کلاس است، و هر کلاس دارای صفر یا بیشتر از صفر رویداد (شیء) می‌باشد. عملاً در کاربردها، کلاس‌ها ایستا هستند؛ بنابراین، وجود، معنا و روابط آنها قبل از اجرای برنامه تثبیت شده است. همین‌طور، کلاس‌های غالب شیء‌ها ایستا هستند و در نتیجه وقتی که شیئی ایجاد شد، کلاسش ثابت است. در مقابل، شیء‌ها نوعاً با یک نرخ شدید در طول زندگی یک کاربرد، ایجاد شده و از بین می‌روند.

دو وظیفه‌ی اصلی تولیدکننده

در طی تحلیل و مراحل اولیه‌ی طراحی، تولیدکننده (developer) دو وظیفه‌ی اصلی دارد:

- شناسایی و تعیین کلاس‌ها و شیء‌ها که لغت‌نامه‌ی (vocabulary) زمینه‌ی مسأله را تشکیل می‌دهند؛
- ابداع ساختارهایی که به وسیله‌ی آنها مجموعه‌ای از شیء‌ها با یکدیگر کار کرده و رفتارهایی را ایجاد کنند که جواب‌گوی خواسته‌های مسأله باشند.

تجربیده‌های کلیدی مکانیزم‌ها

در مجموع چنین کلاس‌ها و شیء‌هایی را تجربیده‌های کلیدی (key abstractions) مسأله می‌نامیم، و این ساختارهای همکاری را مکانیزم‌های (mechanisms) پیاده‌سازی می‌نامیم.

دیدگاه خارجی دیدگاه داخلی

در طی این مرحله (در طی مرحله‌ی تحلیل و مراحل اولیه‌ی طراحی) از ایجاد و توسعه‌ی نرم‌افزار، تمرکز تولیدکننده باید روی دیدگاه خارجی (outside view)، تجربیده‌های کلیدی و مکانیزم‌ها باشد. این دیدگاه (دیدگاه خارجی) نمایشگر چهارچوب منطقی سیستم است، و بنابراین دربرگیرنده‌ی ساختار کلاس (class structure) و ساختار شیء (object structure) از سیستم است. در مراحل بعدی طراحی و گذر به مرحله‌ی پیاده‌سازی وظیفه‌ی تولیدکننده تغییر می‌کند:

معماری مؤلفه‌ها و فرایندها

تمرکز در آن موقع روی دیدگاه داخلی (inside view) از تجربیده‌های کلیدی و مکانیزم‌ها، و روی نمایش فیزیکی آنها خواهد بود. این تصمیم‌های طراحی (در مورد تجربیده‌های کلیدی و مکانیزم‌ها) را ممکن است به‌عنوان قسمتی از معماری مؤلفه‌های سیستم (systems modal architecture) و معماری فرایند سیستم (process architecture) بیان کنیم.

۳-۶ ساخت کلاس‌ها و شیء‌های باکیفیت (On Bridling Ouality Classes and Objects)

فرایند افزایشی و تکراری

بر اساس تجربه‌ی ما، طراحی کلاس‌ها و شیء‌ها، یک فرایند افزایشی (incremental) و تکرار شونده (iterative) است. صادقانه می‌گوییم، که به‌جز تجربیده‌های بسیار ساده، هیچ‌گاه قادر نبوده‌ایم که یک کلاس را در ابتدا به‌طور دقیق تعریف کنیم. البته پالایش تجربیده‌های اولیه با هزینه است و لذا می‌خواهیم هرچه سریعتر به شیء‌ها و کلاس‌های باکیفیت برسیم. پنج معیار زیر را برای ارزیابی کیفیت کلاس‌ها و شیء‌ها پیشنهاد می‌کنیم:

- عدم وابستگی (coupling)، قبلاً در کلاس بحث شده؛
- انسجام (cohesion)، قبلاً در کلاس بحث شده؛
- کافی بودن (sufficiency)، موردی اضافه نباشد؛
- کامل بودن (completeness)، موردی جا نمانده باشد؛
- سادگی (primitiveness).

منظور ما از کافی بودن اینست که کلاس یا مؤلفه، خصوصیات کافی از تجرید را دربر بگیرد تا تراکنشی بامعنی و کارآمد را میسر سازد.

کافی بودن در مقابل کامل بودن.

منظور ما از کامل بودن اینست که واسط کلاس یا مؤلفه، دربردارنده‌ی تمام خصوصیات بامعنی تجرید باشد. درحالی که کافی بودن میین کمترین واسط (minimal interface) است. کامل بودن موردی است که تمام جنبه‌های تجرید را دربر می‌گیرد. بنابراین واسط یک کلاس یا مؤلفه کامل آنقدر عمومی است که برای هر مشتری (client) قابل استفاده است.

اعمال ساده (primitive operations) آنهایی هستند که بتوانند تنها با دسترسی به نمایش تجرید، به‌طور کارآمد پیاده‌سازی شوند.

رهیافتهایی برای انتخاب عمل‌ها یا متدها (Heuristics for Choosing Operation)

۱. از جنبه‌ی عملکرد (trade offs of functional semantics)؛

سادگی عمل‌ها. جدایی عمل‌های بی‌ارتباط.

در داخل یک کلاس شیوه‌ی ما اینست که تمام عملیات را ساده نگه داریم، به‌طوریکه هر نمایشگر یک رفتار کوچک (small) و کاملاً تعریف شده (well defined) باشد. این روش‌ها را (fine-grained) می‌نامیم. مایل هستم عمل‌هایی که با یکدیگر ارتباط ندارند را نیز جدا از هم ترتیب دهیم. در عین حال تفکیک عمل‌ها از یکدیگر و یا ادغام آنها مسلماً در سهولت استفاده از آنها و یا پیاده‌سازی و موارد این چنین تأثیر دارد.

معیارهایی برای تفکیک متدها.

Herbert و O'Brine معیارهای زیر را برای این تفکیک (method) مطرح می‌کنند:

- قابلیت استفاده‌ی مجدد (reusability): آیا این رفتار (behavior) می‌تواند در موارد بیشتری مورد استفاده قرار گیرد؟
 - پیچیدگی (complexity): پیاده‌سازی این رفتار چقدر مشکل است؟
 - به‌کارگیری (applicability): چقدر این رفتار، مرتبط با آن نوع (type) است؟
 - آگاهی پیاده‌سازی (implementation knowledge): آیا پیاده‌سازی این رفتار به اطلاع از جزئیات یک type بستگی دارد؟
- در هر حال ما تعریف عمل‌های بامعنی را انتخاب می‌کنیم.

عمل‌های بامعنی. زیربرنامه‌های آزاد.

در عین حال در زبان‌هایی مثل Ada, CLOS, Object Pascal, C++ تعریف زیربرنامه‌های آزاد (free subprograms) امکان پذیر است. این زیربرنامه را ممکن است در کلاس همه‌بهرها (class utilities) قرار دهیم. در C++ یک زیربرنامه‌ی آزاد، یک تابع غیر عضو (non-member function) می‌باشد.

۲. از جنبه‌ی زمان و حافظه (trade-offs of time and soace semantics)؛

ما دریافته‌ایم که بیان معانی هم‌وجودی (concurrency) برای هر یک از عمل‌ها (operation) مفید است. همچنان که بیان هم‌وجودی برای کلیت شیء مفید است. به‌دلیل این که ممکن است عملیات مختلف نیاز به انواع مختلف همگام‌سازی (synchronization) داشته باشند. بنابراین انتقال پیام ممکن است به یکی از صورت‌های زیر باشد:

- همگام (synchronous): یک عمل (operator) تنها موقعی شروع می‌شود که فرستنده، عمل (action) را شروع کرده است و دریافت‌کننده، آماده‌ی دریافت پیام است؛ فرستنده و دریافت‌کننده هر دو به‌طور نامحدود صبر می‌کنند تا هر دو طرف برای ادامه آماده باشند؛
- ترک‌کننده (balking): مثل همگام، با این تفاوت که در صورتی که دریافت‌کننده بلافاصله آماده نباشد، فرستنده عمل را ترک می‌کند؛
- انقضاء (timeout): مثل همگام، با این تفاوت که فرستنده تنها برای مدت مشخصی برای آماده شدن دریافت‌کننده، صبر می‌کند؛
- ناهمگام (asynchronous): فرستنده بدون توجه به این که دریافت‌کننده انتظار پیامی را دارد یا نه، ممکن است یک action را آغاز نماید.

برای هر یک از عمل‌ها چگونگی همگام‌سازی می‌تواند انتخاب گردد، اما تنها بعد از این که در مورد معانی عملکردی (functional semantics) تصمیم‌گیری شد.

رہیافت‌هایی برای انتخاب رابطه‌ها

انتخاب رابطه بین کلاس‌ها و رابطه‌ی بین شیء‌ها به انتخاب عمل‌ها بستگی دارد. یک رهنمود مفید در انتخاب رابطه‌ی شیء‌ها، قانون Demeter است. این قانون می‌گوید:

قانون Demeter

”متدها یا عمل‌های یک کلاس به هر حال نباید به ساختار هیچ کلاسی به جز ساختار کلاسِ بلافاصله سطح بالاتر، بستگی داشته باشد. به علاوه هر متدی باید تنها به شیء‌های مجموعه‌ی محدودی از کلاس‌ها پیام بفرستد.

یک سطحی.
محدود.

Meyer می‌گوید:

”بین دو کلاس A و B رابطه‌ی وراثت موقعی مناسب است که هر رویداد از B را بتوان رویدادی از A دانست. رابطه‌ی استفاده موقعی مناسب است که هر رویداد از B نمایشگر یک یا چند ویژگی (attribute) از A باشد.“

وراثت
رویداد

نقش مکانیزم‌ها و قابلیت رؤیت (Role of Mechanisms & Visibility) برای انتخاب رابطه‌ها قابل توجه است.

در طول فرایند طراحی، گاهی بیان صریح چگونگی قابلیت رؤیت یک شیء برای شیء دیگر مفید است، اساساً سه راه برای این که شیء x را برای شیء y قابل رؤیت کند وجود دارد:

- قلمرو لغوی یکسان (same lexical scope): y در قلمرو x است؛ بنابراین x صراحتاً می‌تواند y را نام ببرد؛
- پارامتر (parameter): y به عنوان پارامتر برای بعضی عملیات که قابل به کارگیری روی x است ارسال (pass) می‌شود؛
- فیلد (field): y عنصری از x است.

یک اختلاف (variation) روی هر یک از این‌ها، ایده‌ی قابلیت رؤیت مشترک (shared visibility) می‌باشد. به عنوان مثال، ممکن است y فیلدی از x باشد. اما y ممکن است به طرق مختلف برای شیء‌های دیگر قابل رؤیت باشد. در Smalltalk این نوع قابلیت رؤیت معمولاً نشان دهنده‌ی وابستگی بین دو شیء است. قابلیت رؤیت مشترک مستلزم اشتراک ساختاری (structural sharing) است. به این معنی که یک شیء دسترسی انحصاری به دیگری را ندارد. یعنی حالت شیء ممکن است به بیش از یک طریق تغییر داده شود. چنین رابطه‌های قابلیت رؤیت غالباً غیر قابل اجتناب هستند، و لذا بیان صریح این حقیقت در طی فرایند طراحی مفید است.

رہیافت‌هایی برای پیاده‌سازی

انتخاب چگونگی نمایش کلاس یا شیء. از این جهت که کپسوله (encapsulated) باشد. انتخاب محل قرار دادن یک کلاس یا شیء در یک مؤلفه. برای زبان‌هایی مثل C++, CLOS که از ساختار مؤلفه حمایت می‌کنند مهم است. قابلیت رؤیت و پنهان‌سازی اطلاعات رهنمود ما برای انتخاب است. عوامل متعددی در این تصمیم دخالت دارند. حتی موارد غیر تکنیکی مثل مستند سازی.

۷-۳ خلاصه‌ی فصل سوم کتاب طراحی شیء‌گرا / کلاس‌ها و شیء‌ها (Classes & Objects)

- شیء دارای حالت، رفتار و هویت است؛
- ساختار و رفتار شیء‌های مشابه در کلاس مشترک آنها تعریف می‌شود؛
- حالت شیء دربرگیرنده‌ی تمام خواص (معمولاً ایستا) از شیء و همچنین مقادیر (معمولاً پویا) جاری از هر یک از این خواص است؛

- رفتار عبارت از این است که چگونه یک شیء عمل و عکس‌العمل می‌کند، بر حسب تغییرات در حالتش و بر حسب ارسال پیام؛
- هویت خصوصی از شیء است که آن را از تمام شیء‌های دیگر متمایز می‌کند؛
- دو نوع سلسله مراتب شیء‌ها عبارتند از روابط "استفاده" و "شمول" (در برداشتن)؛
- کلاس مجموعه‌ای از شیء‌هاست که دارای ساختار و رفتار مشترک هستند؛
- چهار نوع سلسله مراتب کلاس‌ها عبارتند از: وراثت، استفاده، رویداد (instantiation) و آبر کلاس (meta class)؛
- تجربدهای کلیدی، کلاس‌ها و شیء‌هایی هستند که لغت‌نامه‌ی (vocabulary) زمینه‌ی مسأله را تشکیل می‌دهند؛
- مکانیزم (mechanism) یک ساختار است که به وسیله‌ی آن مجموعه‌ای از شیء‌ها با هم همکاری می‌کنند تا رفتاری را به وجود آورند که جواب‌گوی (بعضی از) خواسته‌های مسأله باشد؛
- کیفیت یک تجرید ممکن است به وسیله‌ی وابستگی کم (coupling)، انسجام (cohesion)، کفایت (sufficiency)، کامل بودن (completeness) و سادگی (primitiveness) اندازه‌گیری شود.

فصل ۴

دسته‌بندی (Classification)

دسته‌بندی ابزاری است که به وسیله‌ی آن دانش را مرتب می‌کنیم. در طراحی شیء گرا درک شباهت بین چیزها به ما اجازه می‌دهد که مشترکات بین تجربه‌های کلیدی (key abstractions) و مکانیزم‌ها (mechanisms) را معلوم کرده، و در نهایت ما را به طرح‌های ساده‌تر هدایت می‌کند.

۴-۱ دسته‌بندی و طراحی شیء گرا (Classification and Object Oriented Design)

شناسایی و تعیین کلاس‌ها و شیء‌ها مشکل‌ترین قسمت طراحی شیء گراست. تجربه‌ی ما نشان می‌دهد که شناسایی و تعیین کلاس‌ها و شیء‌ها متضمن هم کشف، (discovery) و هم ابداع (invention) است. در ضمن کشف، ما به درک "تجربدهای کلیدی" و "مکانیزم"‌های زمینه‌ی مسأله‌ی خود می‌رسیم. و در ضمن ابداع، ما تجربه‌های تعمیم داده شده و همین‌طور مکانیزم‌های جدید برای تنظیم نحوه‌ی همکاری شیء‌ها را ابداع می‌کنیم. در نهایت، کشف و ابداع هر دو، مسائل دسته‌بندی هستند. دسته‌بندی مسأله‌ی پیدا کردن شباهت‌هاست. ما وقتی دسته‌بندی می‌کنیم، دنبال گروه‌بندی چیزها هستیم که دارای ساختار مشترک و یا رفتار مشترک هستند.

دسته‌بندی به تمام جوانب طراحی شیء گرا مربوط است. دسته‌بندی به ما کمک می‌کند تا سلسله مراتب‌های تعمیم (generalization)، خصوصی شدن (specialization)، و اجتماع (aggregation) در بین کلاس‌ها را شناسایی و تعیین کنیم. با درک الگوهای مشترک تراکنش شیء‌ها، ما به ابداع "مکانیزم‌ها" که روح پیاده‌سازی، هستند می‌رسیم.

دسته‌بندی همچنین ما را به تصمیم در مورد مؤلفه‌ها (modularization) هدایت می‌کند. ممکن است ما در مورد گذاشتن بعضی کلاس‌ها و شیء‌ها در مؤلفه‌ی مشترک یا مؤلفه‌های متفاوت، بسته به شباهت‌ها تصمیم بگیریم. انسجام (cohesion) و عدم وابستگی (coupling) معیارهای این شباهت‌ها هستند.

دسته‌بندی در تخصیص فرایندها به پردازنده‌ها نیز ربط دارد. بسته به این که فرایندها از نظر عملکردی چقدر به یکدیگر مرتبط هستند، ممکن است آنها را به یک پردازنده و یا پردازنده‌های متفاوت نسبت دهیم.

دشواری دسته‌بندی (The Difficulty of Classification)

حتی در نظام‌های کاملاً مشخص نیز، دسته‌بندی به میزان زیادی به دلیل دسته‌بندی بستگی دارد. ... این حقیقت که "دسته‌بندی هوشمندانه مشکل است" یک اطلاع جدید است! ناظران مختلف، یک چیز واحد را به طرق مختلفی دسته‌بندی می‌کنند.

دسته‌بندی ماهیت افزایش یابنده (incremental) و تکرار شونده (تکراری) (iterative) دارد.

مشکل‌ترین قسمت طراحی شیء گرا.

کشف و ابداع.

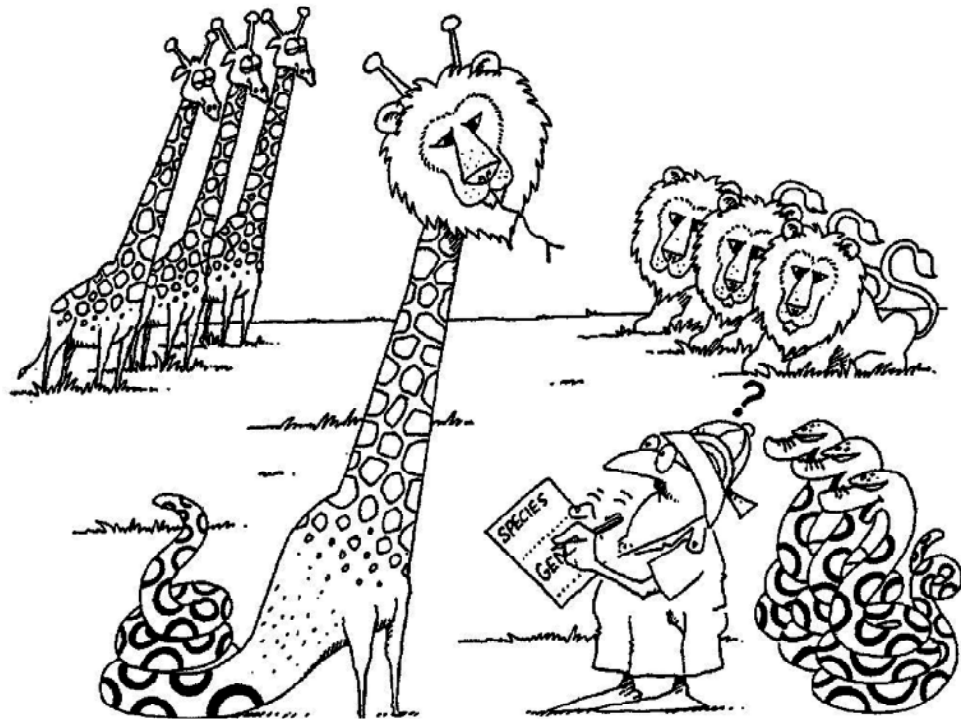
دسته‌بندی و تمام جوانب طراحی شیء گرا در تجربید کلیدی.

در ابداع مکانیزم.

در تصمیم‌گیری فرایندها.

در تصمیم‌گیری مؤلفه‌ها.

ماهیت افزایشی و تکراری دسته‌بندی.



شکل ۴-۱: دسته‌بندی، ابزاری است که به وسیله‌ی آن دانش را مرتب می‌کنیم (دسته‌بندی هوشمندانه مشکل است)

تولید هر تجربه واحد معمولاً روند مشترکی را دنبال می‌کند. ابتدا، مسائل به‌طور پراکنده حل می‌شوند. با افزایش تجربه، بعضی راه‌حل‌ها بهتر از راه‌حل‌های دیگر عمل می‌کنند، و نوعی فرهنگ عامه (folklore) به‌طور غیر رسمی از فردی به فرد دیگر منتقل می‌شود. بالاخره، راه‌حل‌های مفید به‌طور سیستماتیک تری درک شده و مورد تحلیل قرار می‌گیرند (تبلور شاخه‌ای مختلف در نرم‌افزار یک مثال است).

ماهیت فزاینده و تکراری دسته‌بندی، مستقیماً در ساخت سلسله مراتب شیء‌ها و کلاس‌ها در طراحی سیستم‌های نرم‌افزاری پیچیده، تأثیر می‌گذارد. در عمل، در نظر گرفتن یک "ساختار کلاس" در بدو طراحی و تجدید نظر در مورد آن در طول زمان یک امر متداول است. تنها در مراحل ابتدایی، وقتی که مشتریان (clients) ساخته شده‌اند که از این ساختار استفاده می‌کنند، در آن موقع در مورد کیفیت دسته‌بندی می‌توانیم ارزیابی کنیم. بر اساس تجربه، ممکن است از کلاس‌های موجود بخواهیم زیر کلاس‌هایی را به‌وجود آوریم (اشتقاق یا انشعاب derivation). ممکن است بخواهیم یک کلاس بزرگ را به چند کلاس کوچک تقسیم کنیم (تفکیک factorization)، یا با اتحاد چند کلاس، یک کلاس بزرگتر به‌وجود آوریم (ترکیب composition). گاهی حتی ممکن است مشرکاتی را درک کنیم که قبلاً متوجه نشده بودیم و به کلاس جدیدی برسیم (تجريد abstraction) [10].

ممکن است از کلاس‌های موجود به جمع‌بندی‌های جدیدی برسیم:

- اشتقاق یا انشعاب (derivation): به‌وجود آوردن زیر کلاس‌هایی از کلاس‌های موجود؛
مثلاً: به‌وجود آوردن زیر کلاس شیرهای آفریقایی یا زیر کلاس شیرهای باغ وحش از کلاس شیرها؛
مثلاً: به‌وجود آوردن زیر کلاس چند وجهی‌ها از کلاس شکل‌های هندسی؛
مثلاً: به‌وجود آوردن یک زیرعنوان جدید در نگارش.
- تفکیک (decomposition factorization): تقسیم یک کلاس بزرگ به چند کلاس کوچکتر؛
مثلاً: به‌وجود آوردن زیر کلاس‌های شیرهای آفریقایی و شیرهای غیر آفریقایی از کلاس شیرها؛
مثلاً: تقسیم کل شکل‌ها به شکل‌های هندسی و غیرهندسی؛
تفاوت با مورد قبل در اینست که مجموع زیر کلاس‌های اخیر کل کلاس اولیه را به‌دست می‌دهد؛

مثلاً: تفکیک یک عنوان به چند عنوان.

• ترکیب (composition): اتحاد چند کلاس برای به وجود آوردن کلاس بزرگتر.

مثلاً: به وجود آوردن کلاس برتر راه‌روندگان از کلاس‌های شیرها؛ زرافه‌ها؛

مثلاً: ترکیب کلاس‌های مربع‌ها، مثلث‌ها و پنج ضلعی‌ها و به وجود آوردن کلاس چند وجهی‌ها؛

مثلاً: ترکیب دو زیرعنوان در یک عنوان بزرگتر.

ممکن است با بررسی‌های مجدد به کلاس‌های جدیدی برسیم که قبلاً در نظر نداشته‌ایم (abstraction).

مثلاً:

قبلاً به کلاس شیرها و زرافه‌ها رسیده بودیم و حالا متوجه شدیم که کلاس جدید مورچه‌ها را بهتر است در نظر

بگیریم؛

قبلاً به کلاس اجسام توجه کرده بودیم. و حالا به کلاس "سایه‌ها" رسیدیم؛

مثلاً نوشتن یک فصل کاملاً جدید.

دو دلیل مهم برای دشواری دسته‌بندی:

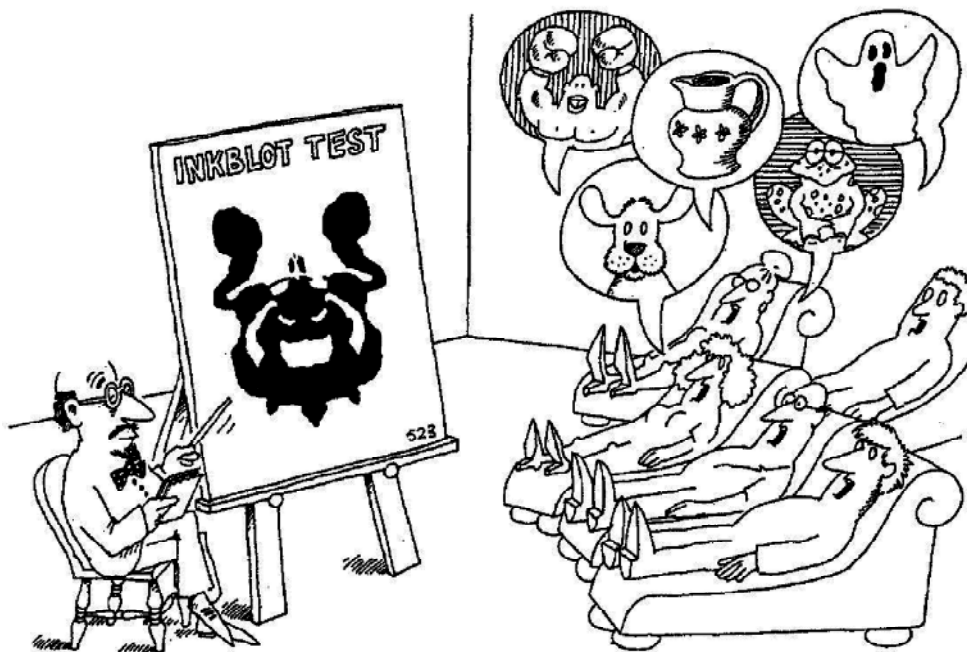
دلیل اول:

چیزی به عنوان دسته‌بندی کامل (perfect) وجود ندارد، گرچه مطمئناً بعضی از دسته‌بندی‌ها از دسته‌بندی‌های دیگر

بهتر هستند. حداقل به اندازه‌ی تعداد کسانی که متصدی امری می‌شوند، می‌توان دنیا را به سیستم‌هایی از شیء‌ها

تقسیم کرد. هر دسته‌بندی به دیدگاه ناظری که دسته‌بندی می‌کند، بستگی دارد.

وابستگی به ناظر.



شکل ۴-۲: ناظران متفاوت، یک شیء خاص را به طرق مختلف دسته‌بندی می‌کنند

دوم اینکه:

دسته‌بندی هوشمندانه به میزان قابل توجهی نیاز به بینش خلاق (creative insight) دارد. گاهی پاسخ روشن است،

نیاز به بینش خلاق.

گاهی سلیقه است، و گاهی انتخاب عناصر مناسب، یک نکته‌ی کلیدی و حساس از تحلیل است [13].

۴-۲ شناسایی و تعیین کلاس‌ها و شیء‌ها (Identifying Classes & Objects)

سابقه‌ی طولانی
مشکل دسته‌بندی.
سه دیدگاه کلی به
دسته‌بندی.

مسئله‌ی دسته‌بندی مورد توجه تعداد بیشماری از فلاسفه، زبان‌شناسان، دانشمندان علوم اداری، راضی‌دانان، و حتی قبل از افلاطون بوده است. مطالعه‌ی تجارب آنها و به‌کارگیری آنها در طراحی شیء‌گرا مغتنم است.

از نظر تاریخی سه دیدگاه کلی به دسته‌بندی بوده است:

- دسته‌بندی کلاسیک (classical categorization)؛
- خوشه‌بندی مفهومی (conceptual clustering)؛
- تئوری خواص (property theory) [15].

تجربه‌ای برای
شناسایی شیء‌ها و
کلاس‌ها.

بر اساس تجربه، ما کلاس‌ها و شیء‌ها را ابتدا بر اساس خواص زمینه‌ی مورد نظر شناسایی و تعیین می‌کنیم. معمولاً این تجربیها (برای انتخاب) وجود دارد، به‌دلیل این‌که مستقیماً بخشی از لغت‌نامه‌ی فضای مسئله را تشکیل می‌دهد [27]. اگر این دیدگاه به ساختار کلاس (class structure) رضایت‌بخشی منجر نشود سپس به خوشه‌بندی (clustering) شیء، به‌وسیله‌ی مفاهیم می‌پردازیم. اگر این هم شکست بخورد، بالاخره دسته‌بندی به‌وسیله‌ی association را مورد توجه قرار می‌دهیم، که از طریق آن خوشه‌های (clusters) شیء‌ها بر اساس این‌که چقدر هر یک از آنها با بعضی شیء‌های نمونه، شباهت دارند تعریف می‌شوند.

به‌علاوه سه دیدگاه فوق (در انتهای پاراگراف قبل) اساس تئوریک تحلیل شیء‌گرا، تحلیل زمینه، و متدهای دیگر (که ما از آنها برای شناسایی و تعیین کلاس‌ها و شیء‌ها در طراحی سیستم‌های بزرگ نرم‌افزاری بزرگ استفاده می‌کنیم) را فراهم می‌کند.

شناسایی شیء‌ها و
کلاس‌ها با تحلیل
شیء‌گرا.

شناسایی و تعیین کلاس‌ها و شیء‌ها به‌وسیله‌ی تحلیل شیء‌گرا

حدس تحلیل و
طراحی شیء‌گرا
روش نیست.
تحلیل شیء‌گرا.
طراحی شیء‌گرا.

گرچه تمرکز روی تحلیل یا روی طراحی کاملاً از هم جدا نیستند، ولی حد بین تحلیل و طراحی تا حدودی مشکوک (fuzzy) است. در تحلیل شیء‌گرا ما دنبال مدل کردن جهان (مسئله) با شناسایی و تعیین شیء‌ها و کلاس‌هایی هستیم که فرهنگ لغت‌نامه‌ی زمینه‌ی مسئله را تشکیل می‌دهد؛ و در طراحی شیء‌گرا ما تجربیها و مکانیزم‌ها را ابداع می‌کنیم به گونه‌ای که رفتار مورد نیاز این مدل را فراهم آورد. لذا تحلیل شیء‌گرا جبهه‌ی ایده‌آل برای طراحی شیء‌گراست.

Mellor و Shalor موارد زیر را به‌عنوان نامزدهایی برای استخراج کلاس‌ها و شیء‌ها پیشنهاد می‌کنند:

نامزدهایی برای
شیء‌ها و کلاس‌ها.

- چیزهای ملموس (tangible things): ماشین‌ها، داده‌های سنجش از راه دور (Telemetry data)، حس‌کننده‌های فشار؛
- نقش‌ها (roles): مادر، معلم، سیاست‌مدار؛
- حوادث (events): فرود، وقفه، درخواست؛
- تراکنش‌ها (interactions): قرض، ملاقات، تقاطع.

از دیدگاه مدل کردن داده‌ها (data modeling)، Ross لیست مشابهی ارائه می‌کند [29]:

- مردم (people)؛
- محل‌ها (places)؛
- چیزها (things)؛
- سازمان‌ها (organization)؛
- مفاهیم (concepts)؛
- حوادث (events).

Coad و Yourdon مجموعه‌ی دیگری از شیء‌های بالقوه را پیشنهاد می‌کنند [30]:

- ساختار (structure): روابط "نوعی از" (kind of) و "جزئی از" (part of)؛
- سیستم‌های دیگر (other systems): سیستم‌های خارجی که با کاربرد مورد نظر ما در تراکنش هستند؛
- دستگاه‌ها (devices): دستگاه‌هایی که با کاربرد مورد نظر ما در تراکنش هستند؛

- حوادث؛
- نقش‌ها؛
- محل‌ها؛
- واحدهای سازمانی.

شناسایی و تحلیل شیء‌ها به‌وسیله‌ی تحلیل زمینه (Domain Analysis)

شناسایی و تحلیل شیء‌ها و کلاس‌ها با تحلیل زمینه.

ما تحلیل زمینه را به‌صورت زیر تعریف می‌کنیم:

”تلاشی برای شناسایی و تعیین شیء‌ها، عمل‌ها (operations)، و روابط که خبرگان زمینه،

اهمیت آنها را دریافته‌اند [31].“

تعریف تحلیل زمینه

Moore و Bailain گام‌های زیر را برای تحلیل زمینه پیشنهاد کرده‌اند [31]:

- ساخت یک مدل عمومی (خبریک) از زمینه با مشورت خبرگان زمینه؛
- بررسی سیستم‌های موجود در زمینه و نمایش این فهم، با یک قالب عمومی؛
- شناسایی و تعیین شباهت‌ها و تفاوت‌های بین سیستم‌ها با مشورت خبرگان زمینه؛
- پالایش مدل خبریک برای جواب‌گویی به سیستم‌های موجود.

دیدگاه‌های دیگر برای شناسایی و تعیین شیء‌ها و کلاس‌ها (Alternate Approaches)

شناسایی شیء‌ها و کلاس‌ها از شرح غیر رسمی مسأله.

تحلیل شیء‌گرا و تحلیل زمینه دو روشی هستند که ما برای طراحی شیء‌گرا ترجیح می‌دهیم. با این حال دو دیدگاه دیگر نیز مطرح شده است:

- شرح غیر رسمی (informal description): این روش ابتدا به‌وسیله‌ی Abbott [33] پیشنهاد شد، پیشنهاد این است که شرح غیر رسمی در مورد مسأله (یا قسمتی از مسأله) نوشته شود. و سپس اسم‌ها و فعل‌ها مشخص شوند. اسم‌ها نامزدهایی برای شیء‌ها هستند و فعل‌ها نامزدهایی برای عملیات روی آنها هستند. و این تکنیک خود منجر به روش‌های خودکار شده است [34]. این روش از این جهت که ساده است و طراح را موظف به کار با فرهنگ زمینه می‌کند مفید است. ولی مسئله پیچیده را جواب‌گو نیست. پیچیدگی‌های زبان طبیعی و وابسته شدن شیء‌ها به شیوه‌ی نگارش از مشکلات دیگر است؛

اسم ~ شیء
فعل ~ عمل

- تحلیل ساخت‌یافته (structured analysis): این روش استفاده از محصول تحلیل ساخت‌یافته را به‌عنوان جبهه‌ی (front end) طراحی شیء‌گرا معرفی می‌کند. این روش قابل توجه است، از این جهت که تعدادی از تحلیل‌گران، با تحلیل ساخت‌یافته آشنا هستند و ابزار (CASE tools) متعددی برای مکانیزه کردن آن وجود دارد. تجربه‌ی ما می‌گوید که تحلیل ساخت‌یافته می‌تواند به‌عنوان یک جبهه‌ی مناسب برای طراحی شیء‌گرا باشد، اما به شرطی که طراح از افتادن در دام طراحی ساخت‌یافته مقاومت کند. خطر جدی دیگر، این حقیقت است که بسیاری از تحلیل‌گران گرایش به این دارند که نمودار جریان داده‌ها را طوری بنویسند که منعکس‌کننده‌ی یک طرح باشد، تا این که مسأله مورد نظر را مدل کند.

شناسایی شیء‌ها و کلاس‌ها از تحلیل ساخت‌یافته.

دو خطر.

۳-۴ شناسایی و تعیین تجربدهای کلیدی (Identifying key Abstraction)

یک تجرید کلیدی، یک کلاس یا شیء است که قسمتی از لغت‌نامه‌ی (فرهنگ) زمینه‌ی مسأله را تشکیل می‌دهد. ارزش اصلی شناسایی چنین تجربدهایی، اینست که حدود مسأله‌ی ما را مشخص می‌کند، آنها چیزهایی که در سیستم ما هستند و در نتیجه در طراحی ما تأثیر می‌گذارند را جلوه می‌دهند. و چیزهایی که خارج سیستم هستند را کنار می‌گذارند. تعیین تجربدهای کلیدی بسیار به زمینه بستگی دارد.

شناسایی و تعیین تجربدهای کلیدی متضمن دو فرایند است: کشف (discovery) و ابداع (intention)، در طی کشف، ما تجربدهایی که به‌وسیله‌ی خبرگان زمینه بکار رفته است را درک می‌کنیم، وقتی خبرگان زمینه در مورد چیزی صحبت

کشف و ابداع.

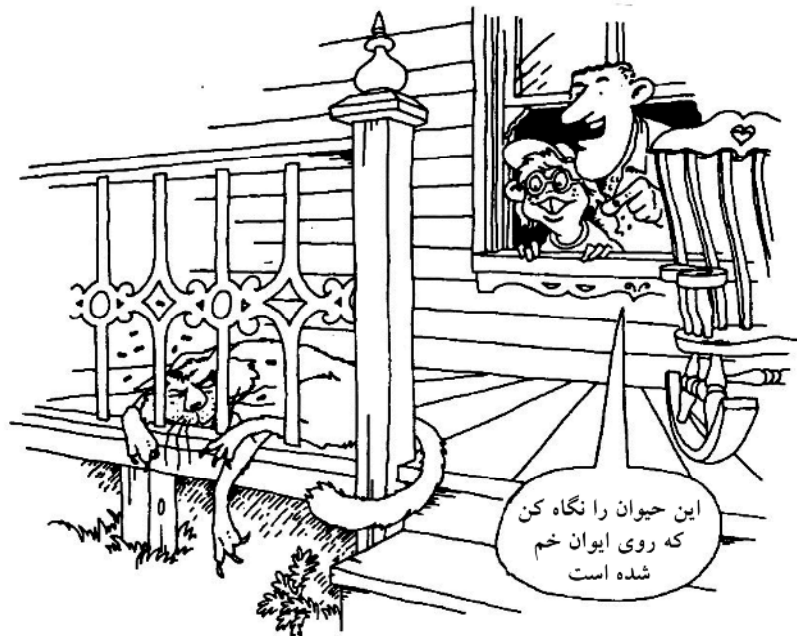
می‌کنند، در این صورت آن تجرید معمولاً مهم است [40]. در طی ابداع، ما کلاس‌ها و شیء‌های جدیدی را که الزاماً جزئی از زمینه‌ی مسأله نیستند، ولی برای طراحی یا پیاده‌سازی مفید هستند را ایجاد می‌کنیم. به‌عنوان مثال، مشتری عابر بانک از واژه‌های شماره حساب، دریافت و پرداخت صحبت می‌کند؛ این لغات بخشی از لغت‌نامه‌ی زمینه هستند. تولیدکننده‌ی چنین سیستمی، از همین تجربدها استفاده می‌کند، در عین حال باید تجربدهای جدیدی مثل بانک‌های اطلاعاتی، مدیر صفحه‌ی نمایش، را معرفی کند. این تجربدها عواملی از آن طراحی خاص هستند و نه از زمینه‌ی مسأله. شاید مهمترین راه‌شناسایی تجربدهای کلیدی، نگاه کردن به مسأله یا طرح و دیدن این که آیا تجربدهایی که شبیه کلاس‌ها و شیء‌های موجود باشد وجود دارند یا نه. چون این یک مسأله دست‌بندی است، ما می‌توانیم از هر یک از روش‌های کلاسیک یا مدرن دست‌بندی استفاده کنیم (روش‌هایی که در این فصل اشاره شد). این دیدگاه تأکید بر استفاده‌ی مجدد از این تجربدها (که ذاتی طراحی شیء‌گراست) دارد.

مهمترین راه
شناسایی تجربدهای
کلیدی.

پالایش تجربدهای کلیدی

هنگامی که یک تجرید کلیدی را به‌عنوان کاندید در نظر گرفتیم، باید آن را بر اساس معیارهایی که در فصل قبل گفته شد ارزیابی کنیم و در صورت لزوم تجدید نظر کنیم. قرار دادن کلاس‌ها و شیء‌ها در سطح درستی از تجرید مشکل است.

پالایش تجربدهای
کلیدی.
سطح تجرید.



شکل ۴-۳: کلاس‌ها و شیء‌ها باید در سطح مناسبی از تجرید باشند: نه خیلی بالا و نه خیلی پایین

نام‌گذاری مناسب چیزها، به ترتیبی که منعکس‌کننده‌ی معانی آنها باشد و گویای تجرید مورد نظر ما باشد، مهم است. ما موارد زیر را پیشنهاد می‌کنیم:

نام‌گذاری مناسب.

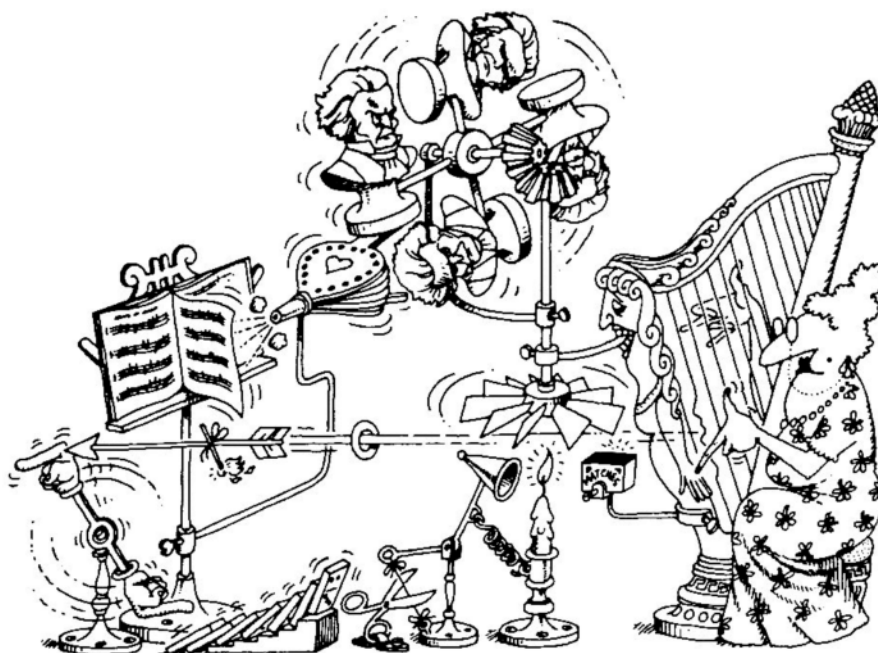
چند معیار نام‌گذاری

- شیء‌ها باید با اسامی خاص نام‌گذاری شوند. (مثل پودر لباسشویی الف)؛
- کلاس‌ها باید با اسامی عام نام‌گذاری شوند. (مثل مواد پاک‌کننده)؛
- عمل‌های تغییر (modifier operations) با فعل‌های معلوم نام‌گذاری شوند. (مثل شستن)؛
- عمل‌های انتخابی (selector operations) باید مبین یک سؤال باشند یا باید با فعل‌های بودن (tobe) نام‌گذاری شوند. (مثل تمیز است).

۴-۴ شناسایی و تعیین مکانیزم‌ها (Identifying Mechanisms)

در طراحی شیء‌گرا ما ابتدا با تعیین تجربدهای کلیدی شروع می‌کنیم تا مدلی از واقعیت را شکل دهیم؛ تنها بعد از آنست که ما به این تجربدها رفتار (behavior) اضافه می‌کنیم تا به رفتار ملموس سیستم برسیم [46].

از واژه‌ی مکانیزم (mechanisms) برای بیان هر ساختاری که به وسیله‌ی آن شیء‌ها با یکدیگر کار می‌کنند تا رفتار لازم برای جواب‌گویی خواسته‌ها را فراهم آورند، استفاده می‌کنیم.



شکل ۴-۴: به وسیله‌ی مکانیزم‌ها شیء‌ها با یکدیگر همکاری می‌کنند تا رفتارهای سطح بالاتری به وجود آورند

در حالی که تجربه‌های کلیدی منعکس‌کننده‌ی لغت‌نامه‌ی زمینه‌ی مسأله هستند، مکانیزم‌ها روح طراحی هستند. در طی فرایند طراحی تولیدکننده‌ی نرم‌افزار، نه تنها طراحی هر یک از کلاس‌ها را باید مورد توجه قرار دهد، بلکه باید متوجه چگونگی عملکرد متقابل رویدادهای کلاس‌ها (شیء‌ها) نیز باشد. وقتی که یک تولیدکننده‌ی نرم‌افزار در مورد یک مکانیزم بخصوص تصمیم بگیرد، کار در بین شیء‌ها به وسیله‌ی تعریف متدهای مناسب در کلاس‌های مناسب تقسیم شده است. ... مکانیزم‌ها نمایشگر سطح دیگری از استفاده‌ی مجدد طراحی (Design reuse) می‌باشند، سطحی بالاتر از استفاده‌ی مجدد از هر یک از کلاس‌ها.

مکانیزم‌ها، روح طراحی.

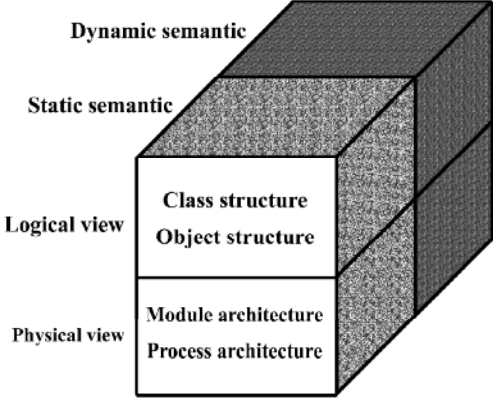
۴-۵ خلاصه‌ی فصل چهارم کتاب طراحی شیء‌گرا / طبقه‌بندی (Classification)

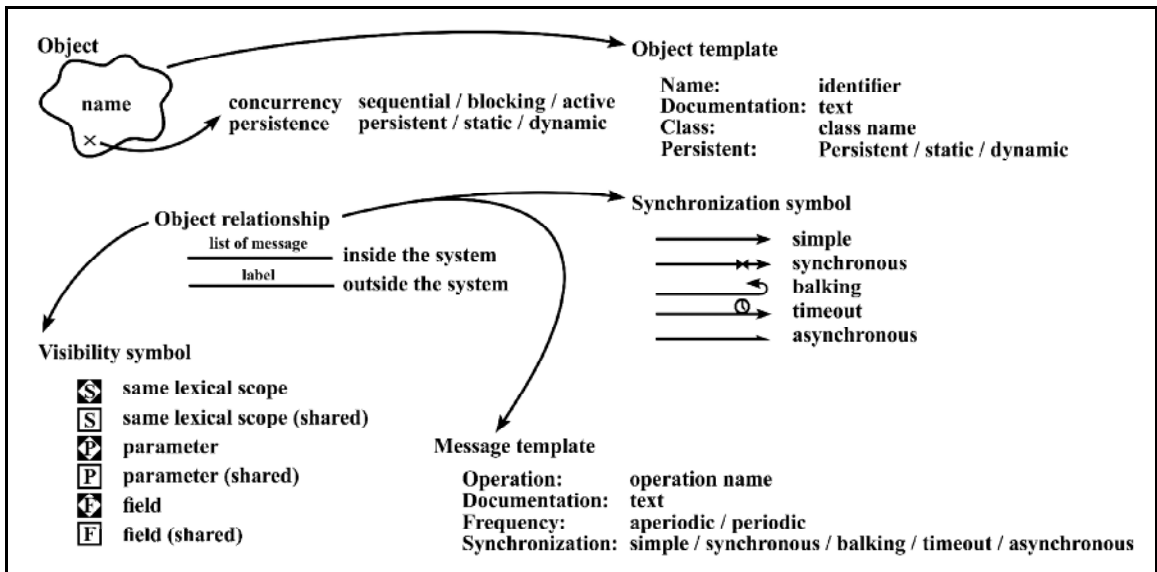
- شناسایی و تعیین کلاس‌ها نکته‌ی اصلی در طراحی شیء‌گراست؛ شناسایی و تعیین، هم اکتشاف و هم ابداع را در بر دارد؛
- کلاس‌بندی (classification) اصولاً یک مسأله‌ی خوشه‌بندی (clustering) است؛
- طبقه‌بندی یک فراروند اضافه‌شونده (incremental) و تکرار شونده (iterative) می‌باشد، و از این جهت مشکل اینست که مجموعه‌ی خاصی از شیء‌ها را ممکن است به طرق مختلف طبقه‌بندی کرد؛
- سه دیدگاه به طبقه‌بندی عبارتند از: طبقه‌بندی شیئی (طبقه‌بندی به وسیله‌ی خصوصیات)، خوشه‌بندی مفهومی (conceptual clustering) یا طبقه‌بندی به وسیله‌ی مفاهیم، و تئوری خواص (property theory) یا طبقه‌بندی به وسیله‌ی انتساب به یک خصوصیت؛
- طراحی شیء‌گرا پیشنهاد می‌کند که چیزهای ملموس، مثل نقش‌ها (roles)، حوادث (events)، و تراکنش‌ها (interactions) کاندیدهایی برای کلاس‌ها و شیء‌ها می‌باشند؛
- تحلیل زمینه‌ی دنبال‌شناسایی و تعیین کلاس‌ها و شیء‌هایی است که در کاربردهای مشابه در یک زمینه‌ی مشترک هستند؛
- تجربه‌های کلیدی منعکس‌کننده‌ی لغت‌نامه‌ی زمینه‌ی مسأله کشف است و ممکن است از قلمرو مسأله کشف شود، یا به‌عنوان قسمتی از طرح، ابداع گردد؛
- مکانیزم‌ها روح طرح‌ها و نمایشگر تصمیم‌های استراتژیک طراحی در مورد همکاری انواع مختلف و متعدد شیء‌ها هستند.

فصل ۵

علامت گذاری (The Notation)

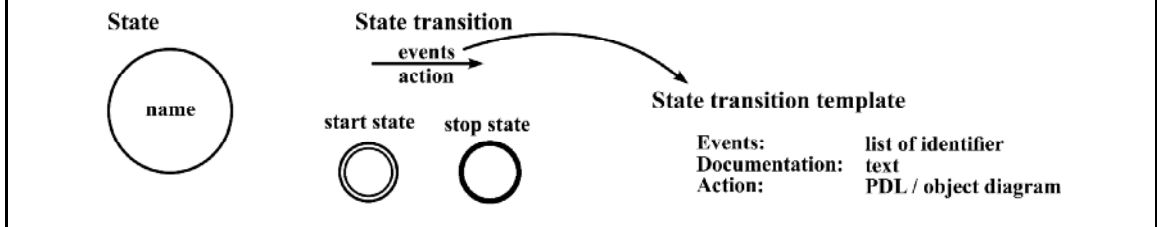
جدول ۵-۱: امدهلی برای طراحی شیء گرا

The Models of Object-Oriented Design	
	Supporting multiple, interrelated views
	
The Process of Object-Oriented Design	
Supporting the incremental and iterative process of round-trip gest off design.	
Identify the classes and object at a given level of abstraction.	
Identify the semantics of these classes and objects;	
Identify the relationship among these classes and objects;	
Implement these classes and objects.	
Object Design	
Illustrates the object structure, including the specification of individual object and their relationship.	



State Transition Diagram

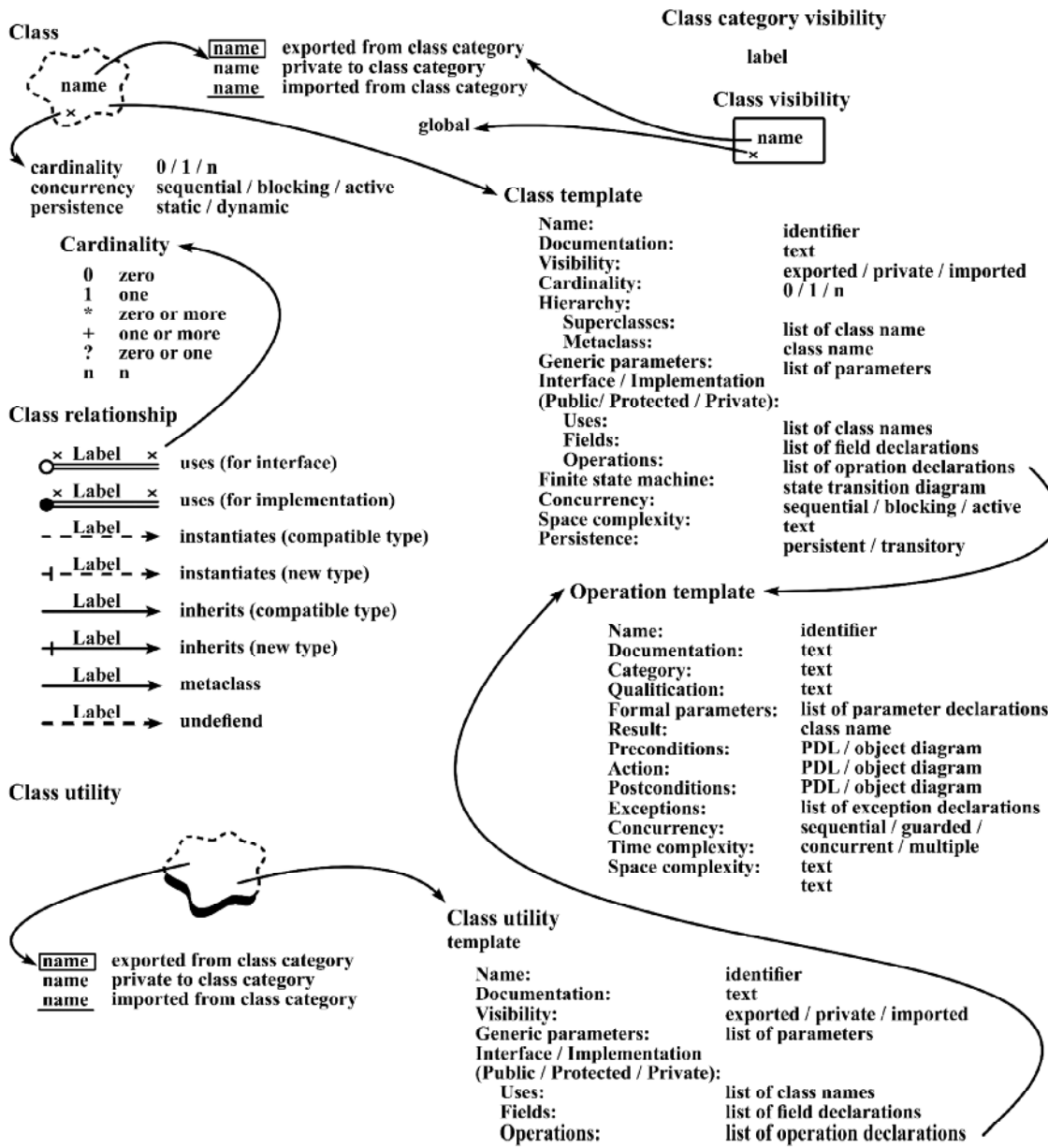
Part of an object diagram, illustrates the order of events among a set of objects.



جدول ۵-۲: نمودار کلاس

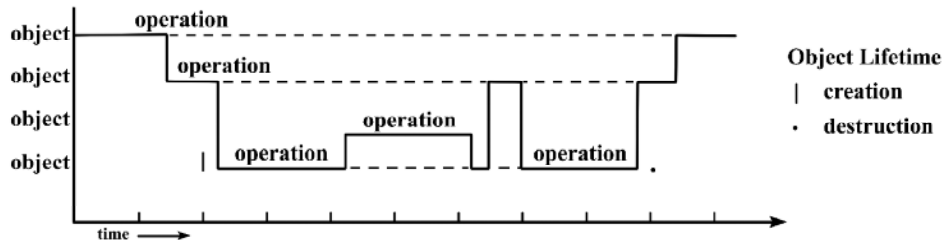
Class Diagram

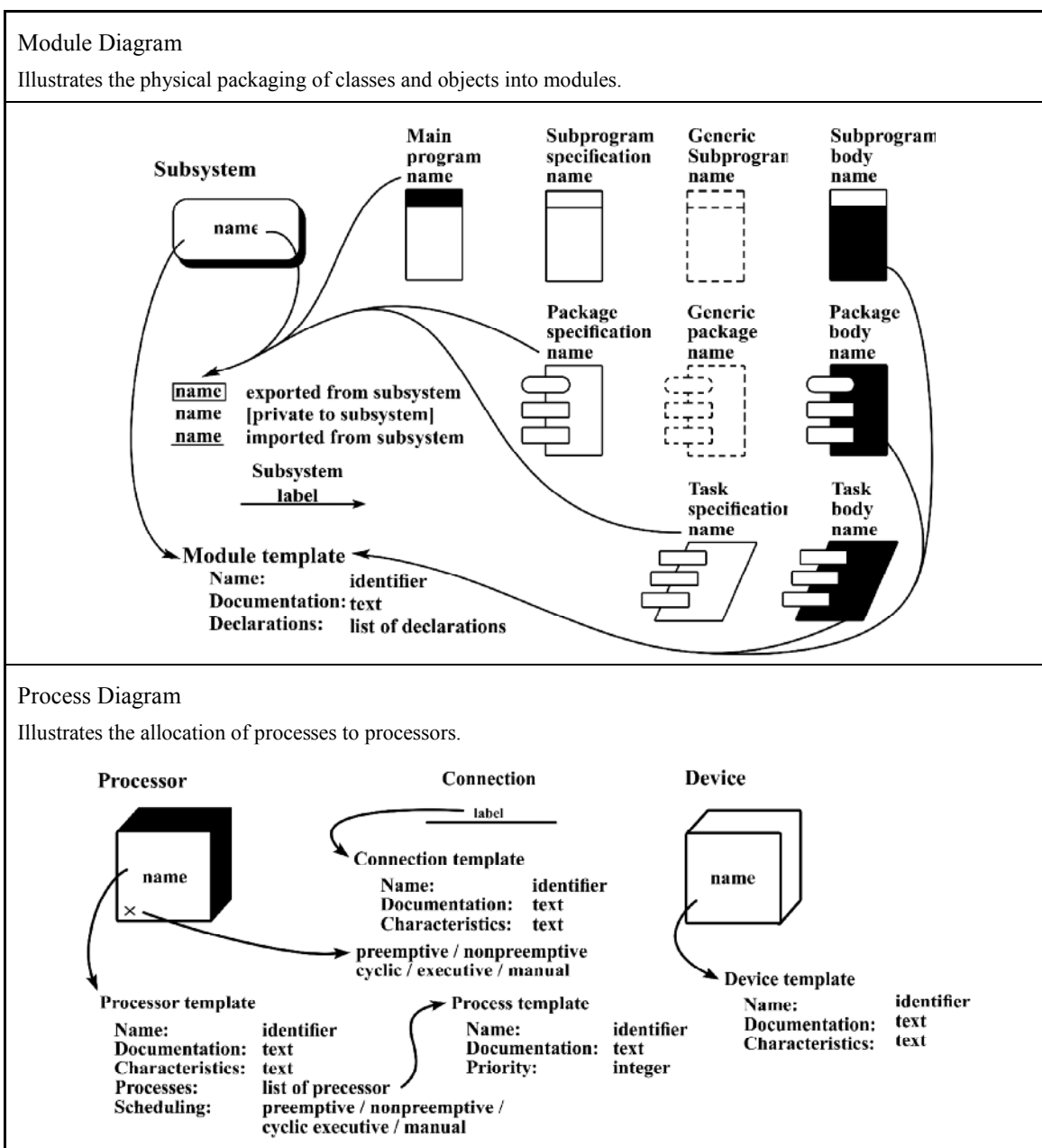
Illustrates the class structure, including the specification of individual classes and their relationships.



Timing Diagram

Part of object diagram, illustrates the order of events among a set of objects.





دلایل علامت‌گذاری

داشتن علامت‌گذاری گویا و کاملاً تعریف شده مهم است چرا که:

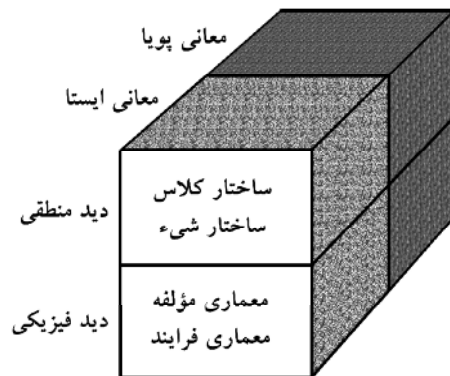
۱. علامت‌گذاری استاندارد، فرموله کردن یک طرح و انتقال آن به دیگران را ممکن می‌سازد؛
۲. یک علامت‌گذاری خوب، تمرکز روی مسایل پیشرفته را میسر می‌سازد؛
۳. استفاده از علامت‌گذاری گویا (expressive)، این امکان را به وجود می‌آورد که اکثر کار یکنواخت و چک‌های سازگاری (consistency) و صحت (correctness) طرح با استفاده از ابزار خودکار حذف گردد.

۵-۱ عناصر علامت‌گذاری طراحی شیء‌گرا (Elements of the Notation)

به کارگیری این علامت‌گذاری در سیستم‌های کوچک و بزرگ.

یک نوع نمودار برای دریافت مناسب تمام جزئیات یک سیستم نرم‌افزاری پیچیده کافی نیست، دیدگاه‌های متعدد لازم است. شکل زیر (که قبلاً هم آمده است) مدل‌های مختلفی را، که برای طراحی شیء‌گرا مهم می‌دانیم، نمایش می‌دهد. ما دریافت‌ایم که این علامت‌گذاری قابل به کارگیری در سیستم‌های کوچک (چند صد خط دستوری) و همین‌طور در

سیستم‌های بزرگ (چندین میلیون خط دستوری) می‌باشد. البته ضرورتی ندارد که همه‌ی جوانب آن همیشه مورد استفاده قرار گیرد.



شکل ۵-۱: مدل‌های مختلف، برای طراحی شیء گرا

این فصل برای بیان نحوه‌ی نگارش (syntax) و معانی (semantics) علامت گذاری، برای طراحی شیء گراست.

مدل‌های منطقی و فیزیکی (Logical Versve Physical Models)

در عمل، ما دریافته‌ایم که جدا کردن انواع مختلف تصمیم‌گیری‌های طراحی، یک نکته‌ی اساسی است. از جمله، یک تولیدکننده‌ی نرم‌افزار، باید موارد بنیانی زیر را در طراحی شیء گرا مد نظر قرار دهد:

- چه کلاس‌هایی وجود دارد و آنها چگونه با یکدیگر مرتبط هستند؟
- چه مکانیزم‌هایی برای تنظیم چگونگی همکاری شیء‌ها به کار رفته است؟
- هر یک از کلاس‌ها و شیء‌ها باید در کجا معرفی شوند؟
- یک فرایند (process) باید به کدام پردازنده نسبت داده شود، و برای یک پردازنده‌ی بخصوص، فرایندهای متعدّدش چگونه زمان‌بندی می‌شوند؟

پاسخ چهار سؤال فوق را به ترتیب در نمودارهای زیر می‌توان تبیین کرد:

- نمودارهای کلاسی (class diagrams)؛
- نمودارهای شیئی (object diagrams)؛
- نمودارهای مؤلفه (module diagrams)؛
- نمودارهای فرایند (process diagrams).

نمودارهای چهارگانه‌ی فوق علامت‌گذاری اصلی طراحی شیء گرا را تشکیل می‌دهند. نمودارهای کلاس و شیء قسمتی از دید منطقی یک سیستم هستند، چرا که در خدمت تبیین وجود و معنی تجربی‌های کلیدی تشکیل دهنده‌ی طرح می‌باشند. نمودارهای مؤلفه و فرایند قسمتی از ساختار فیزیکی سیستم هستند، به دلیل این که برای تبیین دقیق، عناصر سخت‌افزاری و نرم‌افزاری یک پیاده‌سازی هستند.

معانی ایستا و پویا (Static Versve Dynamic Semantics)

نمودارهای چهارگانه‌ی فوق عموماً ایستا هستند. با این حال در تمام سیستم‌های شدیداً نرم‌افزاری، وقایع به‌طور پویا اتفاق می‌افتد: شیء‌ها به‌وجود می‌آیند و از بین می‌روند، شیء‌ها با ترتیب‌های گوناگون به یکدیگر پیام (Message) می‌فرستند، و در بعضی سیستم‌ها، پیام‌ها به‌طور هم‌زمان ارسال می‌شوند.

در طراحی شیء گرا، معانی پویای طرح را توسط دو نمودار دیگر بیان می‌کنیم:

- نمودارهای تغییر حالت (state transition diagrams)؛
- نمودارهای زمانی (timing diagrams).

موارد بنیانی در طراحی شیء گرا.

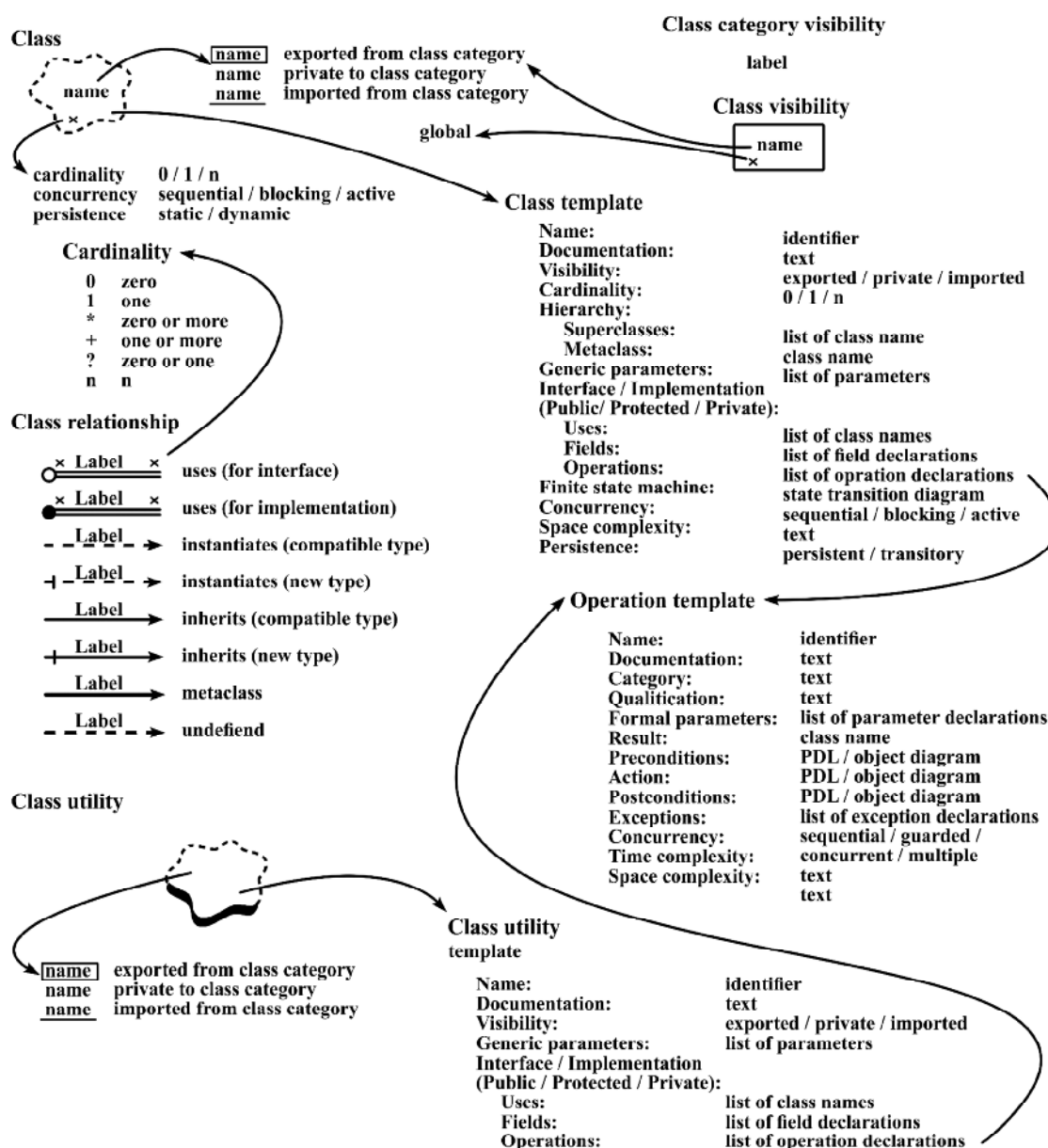
چهار نمودار برای پاسخ‌گویی به موارد بنیانی.

دید منطقی با نمودار کلاس شیء دید فیزیکی با نمودارهای مؤلفه و فرایند.

هر کلاس ممکن است یک نمودار تغییر حالت داشته باشد، که به وسیله آن ترتیب زمانی حوادثی که می‌تواند هر یک از رویدادهای این کلاس را تحت تأثیر قرار دهد، بیان می‌شود. یک نمودار شیء واحد، نمایشگر یک لحظه از یک شیء است (مگر این که آن شیء گذرا transitory باشد)؛ بنابراین، ممکن است از یک نمودار زمانی همراه هر نمودار شیء استفاده کنیم تا ترتیب زمانی ارسال و ارزیابی پیام‌ها را نشان دهیم. در ادامه مجموعه علامت گذاری طراحی شیء گرا به‌طور فشرده ارائه می‌گردد:

۲-۵ نمودار کلاس (Class Diagram)

بیانگر ساختار کلاس، شامل مشخصات هر کلاس و روابط کلاس‌هاست.



شکل ۲-۵: نمودار کلاس

- کاردینالیته (cardinality): تعداد رویدادهایی که یک کلاس ممکن است داشته باشد؛ تعداد رویدادهایی که در یک "رابطه‌ی استفاده" (using class relationship) شرکت می‌کند؛
- همه‌بهر کلاس (class utility): مجموعه‌ای از زیربرنامه‌های آزاد؛

- قابلیت رؤیت (visibility): این توان که یک تجرید، تجرید دیگری را (و در نتیجه منابع مورد مراجعه در دید خارجی را) ببینید. تجریدها تنها در جایی برای دیگری قابل رؤیت هستند که در قلمروشان هم پوشانی (overlap) داشته باشند. در ادامه هر یک از اجزاء مطرح شده در شکل فوق تشریح می شوند.

نمودارهای کلاس (Class diagram)

از نمودار کلاس برای نمایش وجود کلاس ها و رابطه‌ی بین آنها در طرح منطقی (logical design) سیستم استفاده می شود. برای یک سیستم کوچک، معمولاً یک نمودار کلاس کافیهست، اما طراحی اکثر سیستم‌ها نیاز به مجموعه‌ای از این نمودارهای کلاسی برای نمایش ساختار کلاس (class structure) دارد.

عناصر ساختار کلاس.

- سه جزء اصلی یک ساختار کلاس عبارتند از:
 - کلاس‌ها (classes)؛
 - رابطه‌ی کلاس‌ها (class relationships)؛
 - همه‌بهرهای کلاس (class utilities) (برای زبان‌هایی که تعریف زیربرنامه‌های آزاد free subprograms را حمایت می کنند).

کلاس‌ها (Classes)

از شکلک (Icon) زیر برای بیان کلاس استفاده می شود.



شکل ۵-۳: شکلک کلاس

بعضی این شکل را ابر (cloud) می نامند. مشتریان کلاس تنها بر رویدادهای کلاس می توانند عمل کنند، نه خود کلاس. نام کلاس لازم است و در داخل ابر قید می شود. اسم کلاس باید در گروه کلاس‌ها (class category) منحصر به فرد باشد.

رابطه‌ی کلاس‌ها

قبلاً انواع رابطه بین کلاس‌ها را (در فصل سوم) دیدیم. شامل وراثت، استفاده، رویداد و کلاس برتر. شکلک‌های زیر بیانگر این روابط هستند.

رابطه‌ی کلاس‌ها

جدول ۵-۴: رابطه‌ی کلاس‌ها

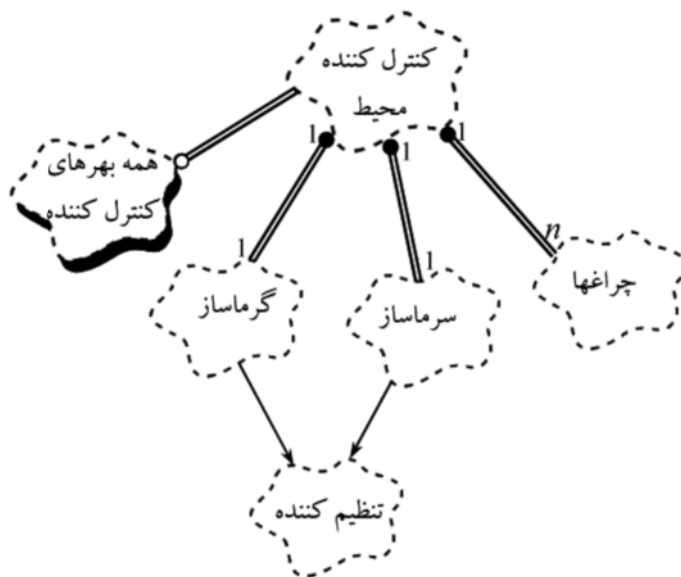
استفاده می کند (برای واسط)	
استفاده می کند (برای پیاده‌سازی)	
روی می دهد (instantiates) (نوع قابل تطبیق compatible)	
روی می دهد (نوع جدید) / کلاس باید مجازی باشد.	
ارث می برد (نوع قابل تطبیق)	
ارث می برد (نوع جدید) / پدر مجازی بوده است.	
کلاس برتر (meta class)	
تعریف نشده	

هر رابطه ممکن است دارای یک برچسب برای مستندسازی اسم یا نقش آن رابطه، باشد.

در بعضی موارد، به‌خصوص در مدل کردن کلاس‌هایی که جزئی از یک بانک اطلاعاتی را شکل می‌دهند، نمایش کاردینالیته (cardinality) در بین کلاس‌هایی که از یکدیگر استفاده می‌کنند بسیار ارزشمند است. در شکل قبل \times می‌تواند یکی از موارد زیر باشد:

جدول ۵-۵: کاردینالیته

0	صفر
1	یک
*	صفر یا بیشتر
+	یک یا بیشتر
?	صفر یا یک
n	n



شکل ۵-۴: یک نمودار کلاس

به‌عنوان مثال، اگر در بین کلاس‌های A و B از علامت "1" نزدیک A و از علامت "+" نزدیک B استفاده کنیم، گویای این است که برای هر رویداد از A ممکن است یک، یا بیشتر از یک رویداد از B داشته باشیم و برای هر رویداد از B، دقیقاً یک رویداد از A وجود دارد.



شکل ۵-۵: مثالی از کاردینالیته

تذکر: رابطه‌ی رویداد تنها برای زبان‌هایی که به نوعی از parameterized class یا generic mechanism حمایت می‌کنند، معنی دارد.

رابطه‌ی تعریف نشده که با خط‌چین و بدون پیکان نمایش داده شده است، گویای این نکته است که طراح رابطه‌ای را بین دو کلاس بیان می‌کند ولی نوع دقیق این رابطه بعداً مشخص خواهد شد.

همه‌بهرهای کلاس (Class Utilities)

برای نمایش همه‌بهرهای کلاس از شکلک زیر استفاده می‌شود.



شکل ۵-۶: شکلک همه‌بهرهای کلاس

(مثل شکلک کلاس با این تفاوت که سایه‌دار شده است). این شکلک بیانگر یک زیربرنامه‌ی آزاد (free subprogram) یا مجموعه‌ای از زیربرنامه‌های آزاد است. اگر زبان مورد استفاده از این ساختار حمایت نکند، می‌تواند حذف شود.

گروه‌های کلاس‌ها و قابلیت رؤیت آنها (Class Categories and Class Category Visibility)

نوعاً نمودار کلاس می‌تواند شامل یک یا چند ده کلاس باشد. با این حال برای یک سیستم بزرگ ممکن است ساختار کلاس شامل چند هزار کلاس باشد. در چنین مواردی قطعاً نمایش تمام کلاس‌ها در یک نمودار کلاس میسر نیست. لذا سازمان دادن کلاس‌ها در "گروه کلاس" (class category) مطرح می‌شود.

در تعداد معدودی از زبان‌های برنامه‌سازی شیء گرا، می‌توان کلاس‌های تودرتو (کلاس در داخل کلاس) داشت. در چنین مواردی، تصمیم‌گیری در مورد کلاس‌ها، تصمیم مهمی است. نکته‌ی مهمتر در طراحی، تعیین گروه کلاس‌ها، نقش زمینه (که به‌عنوان subject areas نیز نامیده می‌شود) است. این نکته برای برنامه‌سازان بسیار اساسی است ولی اکثر زبان‌های برنامه‌سازی شیء گرا از ساختار گروه کلاس‌ها حمایت نمی‌کنند. در عمل هر سیستم بزرگ دارای یک نمودار کلاس سطح بالا (top level class diagram)، شامل گروه‌های کلاس‌ها در بالاترین سطح تجزیه است. چنین نموداری فهم عمومی از معماری سیستم را برای تولیدکننده میسر می‌سازد.

نمودار کلاس سطح بالا.

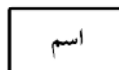
از شکلک جعبه برای نمایش گروه کلاس‌ها استفاده می‌شود. هر گروه کلاس به یک نمودار کلاس گسترش می‌یابد. هر نمودار کلاس ممکن است شامل هم گروه‌های کلاس‌ها، و هم کلاس‌های منفرد باشد.

شکلک برای گروه کلاس‌ها.

قابلیت رؤیت گروه کلاس (Class Category Visibility)

نظر به این که هر گروه کلاس‌ها بیانگر یک دربرگیری (encapsulation) است. ممکن است بعضی از عناصر آن برای خارج از آن گروه کلاس‌ها قابل رؤیت (visible) باشد. بعضی ممکن است خصوصی خود گروه کلاس‌ها و بعضی از گروه‌های دیگر وارد (import) شده باشند. از شکلک زیر برای بیان این مفاهیم استفاده شده است.

قابلیت رؤیت گروه کلاس‌ها.



شکل ۵-۷: گروه کلاس‌ها

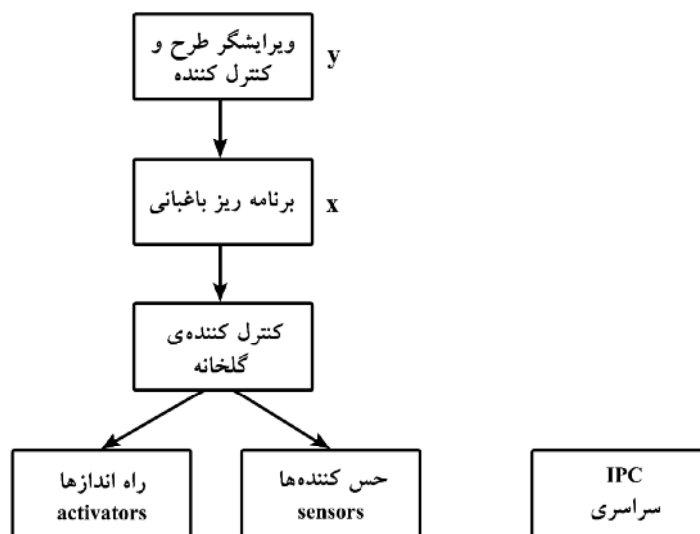
جدول ۵-۶: رابطه‌ی بین گروه‌های کلاس‌ها

اسم	قابل رویت برای دیگران
اسم	خصوصی برای همین گروه کلاس
اسم	وارد شده از گروه کلاس دیگر

رابطه‌ی بین گروه‌های کلاس‌ها به‌وسیله‌ی پیکان نمایش داده می‌شود. برای نمایش این که گروه کلاس‌های y ، کلاس‌هایی را که در گروه x تعریف شده است را وارد می‌کند (یعنی x برای y قابل رؤیت است)، می‌توانیم یک کمان

جهت‌دار از y به x رسم کنیم. به این کمان که بیانگر رؤیت است می‌توان یک برچسب نسبت داد تا این رابطه یا نقش این رابطه را مستند سازی نماید.

گروه کلاس‌های
سراسری.



شکل ۵-۸: یک نمودار کلاسی بالا، نموداری از گروه‌های کلاس‌ها

گروه کلاس‌های y ، کلاس‌هایی را که در گروه x تعریف شده‌اند، را وارد می‌کند. یعنی x برای y قابل رویت است. برای نمایش گروه کلاس‌هایی که قبلاً تعریف شده‌اند و می‌خواهیم در تمام گروه‌های کلاس‌ها قابل رویت باشد، از درج واژه‌ی "سراسری" (global) در داخل جعبه‌ی مربوط به چنین گروه‌هایی استفاده می‌کنیم. در مثال بالا گروه IPC برای تمام گروه‌های کلاس‌های شکل فوق قابل رویت است. این بدین معنی است که تمام مواردی که توسط این گروه کلاس‌های سراسری صادر می‌شود، به وسیله‌ی هر یک از گروه‌های کلاس‌های دیگر از همان نمودار وارد (import) شده است.

کلاس *meta class* که رویدادهایش کلاس هستند. کلاس کلاس‌ها *super class* کلاسی که از آن کلاس دیگر (که زیر کلاس بلافاصل نامیده می‌شود) ارث می‌برد.

الگوهای نمودار کلاس (Class Diagram Templates)

گرچه نمودارهای فوق بسیار کمک کننده است و مرور ساختار را تسهیل می‌نماید ولی کافی نیست. بخصوص باید به گونه‌ای مفهوم هر کلاس (آتر کلاسش، فیدهایش و اعمالش) را بیان کنیم.

جدول ۵-۷: الگوی نمودار کلاس

متغیر	اسم :	Name :	identifier :	
متن	مستندات :	Documentation :	text :	
وارد شده، خصوصی، صادر شده	قابل رویت نسبت به گروه کلاس‌های دربرگیرنده :	Visibility :	exported / private / imported :	در مورد کلیت یک کلاس
n / ۱ / ۰	کاردینالیته :	Cardinality :	0 / 1 / n :	
سلسله مراتب		Hierarchy		
لیست اسامی کلاس‌هایی که این مستقیماً از آنها ارث می‌برد	سوپر کلاس‌ها	Super classes	list of class names :	قابل رویت درمورد اجزاء این کلاس

اسم کلاس (کلاسی که این کلاس یک رویداد از آن است)	متا کلاس ها	Meta classes	: class name
جنریک پارامترها	پارامترهای جنریک	Generic parameters	: list of parameters
واسط یا پیاده سازی		Interface / Implementations	
(عمومی / حفاظت شده / خصوصی):		(Public / Protected / Private) :	
لیست اسامی کلاس ها	استفاده می کند	Users	: list of class names
فهرست تعریف فیلدها	فیلدها	Fields	: list of field declarations
لیست تعریف عمل ها	عمل ها	Operations	: list of operation declarations
نمودار تغییر حالت	ماشین با حالت های محدود	Finite state machine	: state transition diagram
ردیفی / مسدودی / فعال	هم وجودی	Concurrency	: sequential / blocking / active
متن	پیچیدگی فضا	Space complexity	: text
ایستا / پویا	ماندگاری	Persistence	: static / dynamic

ما برای هر یک از کلاس های نمودار کلاسی، یک الگوی کلاسی (class template) نیز داریم.

تذکره: این الگو نباید با الگوی مطرح شده در زبان ++C برای کلاس های پارامتری (parameterized class) اشتباه گرفته شود. الگوی کلاس تمام جنبه های مهم هر کلاس را که در فصل سوم مطرح شد، دربر می گیرد. این الگو تقریباً وارد جزئیات شده است. اما در عمل انتظار نمی رود که تولیدکننده نرم افزار تمام این اجزاء را پر کند، مگر این که نمایش چنین جزئیاتی لازم باشد. در حقیقت، در مراحل اولیه طراحی این چنین الگوهایی به طور پراکنده پر می شوند؛ با پیشرفت طراحی، اجزاء بیشتری مشخص شده و اضافه می گردند. در صورتی که زبان پیاده سازی به اندازه ی کافی گویا باشد، رها کردن این الگوها و بیان مستقیم طرح به وسیله ی زبان مربوطه معقول است. در ادامه به شرح اجزاء این الگو (در صورت لزوم) می پردازیم.

جدول ۵-۸: شرح اجزاء الگوی نمودار کلاس

اسم (name):	بدون شرح
مستند (document):	شرحی در مورد این کلاس
قابلیت رویت:	بیانگر این نکته است که نسبت به گروه کلاس های دربرگیرنده، این کلاس صادر شده (قابل رویت توسط دیگران)، وارد شده (import) و یا خصوصی همین گروه کلاس هاست.
کاردینالیتی:	بیانگر این است که چه تعداد رویداد (instance) را برای این کلاس اجازه می دهیم. نوعاً این مقدار، 0، 1 و n است.
سلسله مراتب	این فیلد بیانگر نقش این کلاس در سلسله مراتب کلاس هاست. بسته به زمان پیاده سازی مورد استفاده ی، این کلاس ممکن است، صفر، یک یا بیشتر، super class و نیز class meta داشته باشد. در هر حال کلاس هایی که در اینجا ذکر شده است، از روابط بین کلاس ها، که در نمودار کلاس نمایش داده شد، استخراج شده است؛ دلیل مطرح شدن آنها در الگو، تنها به خاطر کامل و قابل فهم بودن الگو است (البته

طراحی شیء‌گرا و کاربردها

	در صورت اجازه دادن زبان).	
زبان‌های مختلف.	در صورتی که زبان مورد استفاده اجازه دهد، این جزء از الگوی کلاس، فراهم کننده پارامترهایی به این کلاس است (به‌عنوان مثال، پارامترهای جنریک برای Ada، یا ماکروها و کلاس‌های پارامتریک (parameterized classes) برای C++).	generic parameters:
در مورد هر فیلد.	سه فیلد بعدی تا چهار بار تکرار شده است: سه بار برای واسط (interface) و یک بار برای پیاده‌سازی (implementation) این کلاس. در صورتی که زبان پیاده‌سازی اجازه دهد، واسط کلاس را می‌توان به سه قسمت عمومی (public)، حفاظت شده (protected) و خصوصی (private) تقسیم کرد. بنابراین C++ ممکن است از تمام این سه قسمت استفاده کند. Ada تنها از قسمت‌های عمومی و خصوصی استفاده می‌کند و Object Pascal تنها از قسمت‌های public استفاده می‌کند. در هر صورت این بخش از الگوی کلاسی مهمترین قسمت آن است؛ چرا که در این قسمت است که دیدگاه خارجی کلاس بیان می‌شود. این دیدگاه خارجی در بردارنده متغیرهای رویداد (instance variables) و متغیرهای کلاس (class variables) - که در طی field ها مستند شده است - و همچنین تمام عمل‌ها (operations) می‌باشد. برای هر فیلد ممکن است نام آن را مستند کنیم، این که ثابت یا متغیر است؛ کلاسش، هرگونه محدودیت آن (مثلاً: فیلد ما عدد صحیح است و مقدارش در محدوده‌ی ۱ تا ۱۰۰ است) و این که این فیلد چگونه مقدار اولیه می‌گیرد. باز هم مجبور نیستیم که تمام این چیزها را مستند کنیم، تنها مواردی که برای ما مهم است را مستند می‌کنیم. با این حال الگو به اندازه‌ی کافی کامل هست تا تمام جنبه‌های مهم فیلد را بیان کند. در ادامه، جزء عمل‌ها (operations) مبین لیستی از عمل‌ها (با الگوی خاص خود) است که به اختصار شرح می‌دهیم. جزء uses شبیه meta class و super class است که در اینجا به‌خاطر کامل و قابل فهم بودن قرار داده‌ایم، در واقع تمام روابط استفاده (using relationship) در نمودار شیء به‌صورت گرافیکی برقرار شده است. چهار عنصر بعدی از الگوی کلاس، مستندکننده معانی پویا و رفتار زمان (time) و حافظه‌ی (space) رویدادهای این کلاس است.	public / protected / private uses :fields operations
اجزاء دیگر الگوی نمودار کلاس.	این جزء از الگوی کلاس نمایش دهنده‌ی یک نمودار تغییر حالت (state transition) است که به‌زودی به تفصیل مطرح خواهیم کرد.	ماشین با حالت‌های محدود:
	این جزء بیانگر این است که آیا رویدادهای این کلاس ردیفی (sequential)، مسدودی (blocking)، یا فعال (active) است. طبیعتاً این معانی با معانی هم‌وجودی مربوط به عمل‌ها باید تطابق داشته باشد. به عنوان مثال اگر کلاسی به‌صورت ردیفی مشخص شده است، در این صورت هیچ یک از عمل‌هایش نمی‌تواند معانی guarded، concurrent یا چندگانه (multiple) داشته باشد.	هم‌وجودی (concurrency):
	این جزء از الگوی کلاس مستند کننده‌ی زمان و فضای مورد مصرف رویدادهای این کلاس است. این اندازه می‌تواند بر حسب واحدهای واقعی حافظه، یا بر حسب مفاهیم نسبی (نوعاً بر حسب علامت‌گذاری big O) بیان گردد.	پیچیدگی (space & time complexity)
	آخرین جزء الگوی کلاسی مستند کننده‌ی ماندگاری شیء‌های این کلاس است. یک شیء ماندگار (persistence object) موردی است که حالت و کلاسش بعد از طول زندگی برنامه‌ای که آن را به‌وجود آورده است، باقی بماند. یک شیء انتقالی (transitory object) موردی است که بعد از مرگ برنامه‌ی به‌وجود آورنده‌اش، باقی نماند.	ماندگاری (persistence):

باز هم تأکید می‌کنیم که ذکر تمام موارد لازم نیست. در هر صورت موارد مهم و ضروری را طراح می‌تواند در این قالب برای هر کلاسی مستند نماید. ضمن این که بعضی مفاهیم در عین حال در نمودارهای مختلف به صورت گرافیکی بیان می‌شوند.

الگوی همه‌بهر کلاس (Class Utility Templates)

در ادامه الگوی همه‌بهر کلاس آمده است. نظر به این که همه‌بهر کلاس بیانگر مجموعه‌ای از زیر برنامه‌های آزاد (free sub programs) (و گاهی بیانگر ثابت‌ها و متغیرهای سراسری است) می‌باشد، الگوی آن زیرمجموعه‌ای از الگوی کلاس است.

جدول ۵-۹: الگوی همه‌بهر کلاس

اسم	Name	: identifier
مستندات	Documentation	: text
قابلیت رویت	Visibility	: exported/private/imported
پارامترهای جنریک	Generic parameters	: list of parameters
واسط / پیاده‌سازی :	Interface / Implementation :	
لیست اسامی کلاس‌ها	Uses	: list of class name
لیست تعریف فیلدها	Field	: list of field declaration
لیست تعریف عمل‌ها	Operations	:list of operation declaration

الگوی عمل‌ها (Operation Templates)

هم الگوهای کلاس‌ها و هم الگوی همه‌بهری کلاس ممکن است شامل لیستی از عمل‌ها باشند. اغلب بیان اسم هر عمل، پارامترها و معانی آنها (با یک متن آزاد) کافیت. اگر نیاز به مستندسازی دقیقتر باشد از الگوی زیر می‌توان استفاده کرد.

جدول ۵-۱۰: الگوی عمل‌ها

Name	: Identifier
Documentation	: text
Category	: text
Qualification	: text
Formal parameters	: list of parameter declarations
Result	: class name
Preconditions	: PDL / object diagram
Action	: PDL / object diagram
Post conditions	: PDL / object diagram
Exceptions	: list of exception declaration
Conevrrency	: sequential / guarded / concurrent / multiple
Time Complexity	: text
Space	: text

جدول ۵-۱۱: شرح اجزاء الگوی عمل‌ها

نام	.name
مستندات	:document
ممکن است عمل‌ها را با معیارهایی گروه‌بندی کنیم، مثلاً گروه modifier ها که حالت‌ها را تغییر می‌دهند، یا selector ها که موجب تغییر حالت نمی‌شوند. این جزء برای مواقعی که کلاس دارای عملیات زیادی است، مفید می‌باشد.	گروه (category):
در زبان CLOS از این فیلد بیشتر استفاده شده است. این که این عمل (operation) نمایشگر یک متد اصلی است یا نه، این که این عمل یک متدِ after, before, around است یا نه (و یا تلفیقی از تعریف نوع متدهای کلاس) هست یا نه، می‌تواند در اینجا مستند گردد. به مطالب صفحه‌ی ۱۰۴، پاراگراف آخر مراجعه شود. (۱۰۴ از کتاب Booch). در مورد Invoke شدن این متد است.	توصیف (qualification)
	پارامترهای رسمی :
تنها در صورتی که متد یک تابع باشد، نوع مقدار برگشتی از تابع، در اینجا قید می‌گردد. فیلدهای قبلی از عملیات، جنبه‌های ایستای عمل (operation) را مشخص می‌کرد. در چهار فیلد بعدی معانی پویای عمل مورد توجه قرار می‌گیرد.	نتیجه (result):
بیان معنی operation به‌وسیله‌ی یک متن آزاد و یا به‌صورت رسمی با بیان سه فیلد زیر: پیش شرط (pre condition): شرط لازم برای انجام عمل؛ پس شرط (post condition): شرایط بعد از انجام عمل؛ استثنا (exceptions): شرایط استثنایی. معنای یک operation ممکن است گاهی به نمودار شیء (که نمایشگر رابطه‌ی شیء‌های شرکت‌کننده در عمل است) و معنای پویای آن عمل (که به‌وسیله‌ی نمودار زمانی و یا Program Description Language \equiv PDL بیان شده است) اشاره داشته باشد.	عملکرد (action):
یک operation ممکن است ردیفی باشد. به این معنی که تنها با وجود یک ریسمان کنترل معنای آن عمل تضمین می‌شود. ممکن است چند ریسمان کنترلی، در معنا پیدا کردن یک عمل مطرح باشد. اما با نگرش‌های مختلف به همگام سازی: Guarded Concurrent Multiple	هم‌وجودی (concurrency):
متن در مورد زمان مورد مصرف.	پیچیدگی زمانی (time complexity):
متن در مورد حافظه‌ی مورد مصرف.	حافظه (space complexity):

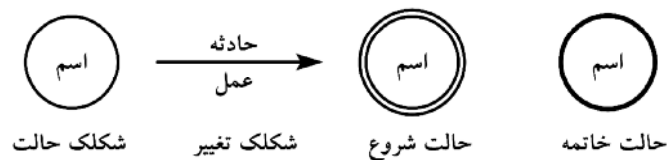
۳-۵ نمودارهای تغییر حالت (State Transition Diagrams)

تنها با نگاه کردن به نمودار کلاس، رفتار پویای رویدادهای هر یک از کلاس‌ها مشهود نیست. رفتار پویای کلاس‌ها را توسط نمودارهای تغییر حالت (state transition diagrams) بهتر می‌توان مستند کرد. این نمودارها بیانگر فضای حالت‌های (state space) یک کلاس، حوادثی (events) که موجب انتقال از یک حالت به حالت دیگر می‌شود، و همچنین بیانگر عمل (action) حاصل از تغییر حالت است. بنابراین نمودارهای تغییر حالت با بخش‌های دیگر علامت گذاری مرتبط است: الگوی کلاس ممکن است شامل یک نمودار تغییر حالت باشد و عمل‌هایی که در یک نمودار تغییر حالت به‌خصوص مطرح شده است ممکن است به نمودارهای شیء دیگر اشاره کند.

حالت‌ها توسط دایره و نام آن حالت نمایش داده می‌شود. نام حالت لازم است و باید در آن نمودار تغییر حالت منحصراً به‌فرد باشد.

نوفاً نمودارهای تغییر حالت مربوط به یک کلاس، دارای حالت شروع و یا حالت خاتمه نیستند: هنگامی که یک شیء ایجاد می‌شود، بسته به شرایط محیطش وارد یک حالت می‌شود، و وقتی آن شیء از بین می‌رود، تمام حالت‌های مربوط به آن شیء از بین می‌رود (moot). در مواردی که نمایش صریح حالت‌های شروع و خاتمه لازم باشد به ترتیب از دایره‌ی دوخطی و دوخطی پر شده (دایره‌ای با محیط ضخیم) استفاده می‌شود.

تغییر حالت: تنها رابطه‌ای که در بین حالت معنی دارد، تغییر حالت است، تغییر از یک حالت به حالت دیگر و همچنین تغییر یا انتقال از یک حالت به همان حالت. تغییر حالت توسط کمان جهت‌دار نمایش داده می‌شود. این کمان باید دارای برچسب (label) باشد. این برچسب گویای نام حداقل یک حادثه‌ای است که موجب این انتقال شده است، و یا مربوط به عمل حاصل است.



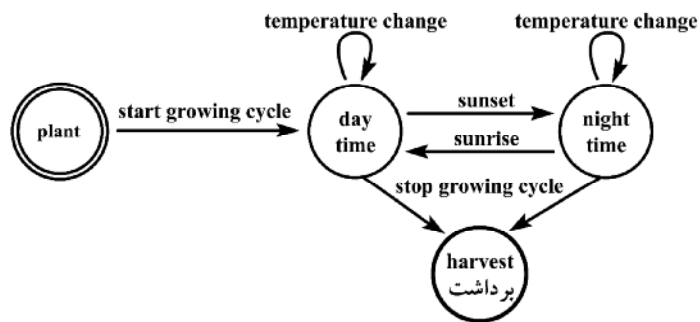
شکل ۵-۹: نمودار تغییر حالت

الگوی تغییر حالت (State Transition Template)

مثل کلاس‌ها، همه‌بهرهای کلاس و عملیات، برای تغییر حالت هم الگو وجود دارد. نظر به این که هر تغییر حالت را با یک عمل (action) مربوط و جمع می‌کنیم، این نوع ماشین با حالات محدود (finite state machine) را مدل "روی لبه‌ها" (mealy model) می‌نامند بر خلاف مدل "روی گره‌ها" (more model) که عمل‌ها به حالت‌ها نسبت داده می‌شوند.

جدول ۵-۱۲: الگوی تغییر حالت

سیستمی از معرف‌ها	حادثه‌ها	Event	: list of identifier
متن	مستند	Documentation	: text
PDL / نمودار شیء	عمل	Action	: PDL / object diagram



شکل ۵-۱: مثالی از نمودار تغییر حالت

۴-۵ نمودارهای شیء (Object Diagrams)

از نمودار شیء برای نمایش وجود شیء‌ها و رابطه‌ی بین آنها در طرح منطقی سیستم استفاده می‌شود. یک نمودار شیء واحد، نمایشگر تمام یا قسمتی از شیء (object structure) از سیستم است. نوعاً برای طرح یک سیستم، مجموعه‌ای از نمودارهای شیء لازم است.

هدف از هر نمودار شیء، بیان معانی مکانیزم‌های کلیدی (key mechanisms) در طرح منطقی است. در طرح یک سیستم، کلاس‌ها عمدتاً ایستا هستند، در حالی که شیء‌ها بیشتر انتقالی (transitory) هستند. به طوری که بسیاری از آنها در حین یک اجرا ممکن است ایجاد و نابود شوند، بنابراین از نمودار شیء برای گرفتن معانی پویای (dynamic semantics) عملیات و ماشین‌های با حالت محدود (state machines finite) استفاده می‌کنیم. بنابراین یک نمودار شیء، خاص یک لحظه از یک حادثه‌ی گذرا را نشان می‌دهد. از این جهت نمودارهای شیء، نمونه‌ای (prototypical) هستند: هر یک نمایشگر تراکنش‌هایی است که می‌تواند در بین مجموعه‌ای از شیء‌ها بروز کند.

روابط مهمی بین نمودارهای کلاس و نمودارهای شیء برای به موازات هم کردن نقش متقابل کلاس‌ها و شیء‌ها وجود دارد. به ویژه، هر شیء از نمودار شیء بیانگر یک رویداد (رویداد مشخص یا دلخواه) از یک کلاس است. به علاوه عملیاتی که در یک نمودار شیء به کار رفته است باید با عملیاتی که در کلاس‌های مربوطه تعریف شده است، سازگار (consistent) باشد. به عنوان مثال، اگر شیء S پیام یا عمل M را برای شیء R می‌فرستد، در این صورت عمل (operation) M باید برای کلاس مربوطه به S تعریف شده باشد. البته این سازگاری دوطرفه است. اگر در نمودارهای کلاس تغییراتی اعمال شد، تغییرات نظیر در نمودارهای شیء نیز باید داده شود.

برای مستند کردن طرح منطقی سیستم، هم به نمودارهای کلاس و هم به نمودارهای شیء نیاز داریم. چراکه این نمودارها نمایشگر تصمیم‌های طراحی کاملاً متفاوت هستند. نمودارهای کلاس تجربدهای کلیدی از سیستم را مستند می‌کنند و نمودارهای شیء نمایشگر مکانیزم‌های مهم پردازش‌کننده‌ی این تجربدها هستند. دو عنصر مهم نمودار شیء، شیء‌ها و رابطه‌ی شیء‌ها می‌باشند.






شیء‌ها. از شکلک مشابه شکلک مربوط به کلاس‌ها ولی به صورت خط پر استفاده می‌شود. احتیاجی نیست که اسم شیء منحصر به فرد باشد. در حقیقت بسیاری از شیء‌های یک برنامه هیچ اسمی ندارند که صراحتاً در سطح کد مبدأ (source) شناخته شده باشند. به این دلیل ضرورتی ندارد که اسمی شیء دقیق باشد ولی می‌تواند گویای رویدادی نامعین (indefinite) که نماینده‌ی تجرید است، باشد. مثلاً "یک سردکننده"، "یک کتاب". در صورت لزوم می‌توان رویدادهای خاصی را نام‌گذاری کرد. مثلاً گل‌خانه‌ی شماره‌ی ۷.

روابط شیء‌ها. رابطه‌ی بین دو شیء بیانگر این است که این شیء‌ها می‌توانند به یکدیگر پیام بفرستند. نظر به این که پیام‌ها نوعاً دوطرفه (bidirectional) هستند، از خطوط بدون پیکان برای نمایش رابطه‌ها استفاده می‌کنیم: خط پر (solid) برای رابطه‌هایی از شیء‌ها که در نرم‌افزار داخل سیستم بتوانند مدل شوند. خط نازک (gray) برای مواردی که خارج از سیستم هستند (مثلاً برای مستند کردن حلقه‌ی بازخورد، در سیستم‌های گرمایی و سرمایی).

شکلک برای نمایش شیء	اسم
داخل سیستم	<u>لیست پیامها</u>
خارج سیستم	<u>برچسب</u>

شکل ۵-۱۱: کلمات پیام، عمل و مدت تقریباً معادل هم به کار می‌روند و معادل تابع عضویت





برای سیستم‌هایی که صرفاً ردیفی هستند، رسم یک خط ساده جهت‌دار برای بیان تراکنش بین دو شیء کافی است. اما در مواردی که چند ریسمان کنترل (multiple threads of control) وجود داشته باشد کمی مشکل‌تر است. به عنوان مثال، اگر دو شیء فعال و خواهان (active) هستند، در این صورت پیامی از S به R ممکن است تأخیر انداخته شود (چراکه S نمی‌تواند منتظر پاسخ بماند). ممکن است پیام‌هایی داشته باشیم که به یک شیء مشغول، وقفه (interrupt) بدهد. در مجموع پنج نوع همگام‌سازی پیام‌ها وجود دارند، که در فصل ۳ شرح داده شده‌اند: ساده (simple)، همگام (synchronous)، (balking)، (timeout) و ناهمگام (asynchronous). شکل زیر این شکلک‌ها را نمایش می‌دهد، هر کدام از شکلک‌ها ممکن است برچسبی (label) از فهرست اسامی پیام‌ها داشته باشد.



	simple
	synchronous
	balking
	timeout
	asynchronous

شکل ۵-۱۲: شکلک‌های همگام‌سازی پیام‌ها

قابلیت رؤیت شیء‌ها و هماهنگی (Object Visibility and Synchronization)

می‌توانیم به دو طریق جزئیات بیشتری از نمودار شیء را مشخص کنیم. قابلیت رؤیت شیء، اولاً: می‌توانیم این نکته که چگونه دو شیء یکدیگر را می‌بینند، را مستند کنیم (این مطلب را تنها در مواردی که لازم هست، اضافه می‌کنیم). به شش صورت یک شیء می‌تواند برای شیء دیگر قابل رؤیت باشد. در شکل زیر این شش راه به وسیله‌ی شکلک‌ها نشان داده شده است.

	Same lexical scope
	Same lexical scope (shared)
	parameter
	Parameter (shared)

	field
	Field (shared)

شکل ۵-۱۳: قابلیت رؤیت یک شیء برای شیء دیگر

به‌عنوان مثال، اگر شیء R یک فیلد مشترک (shared field) از شیء S است، در این صورت این قابلیت رؤیت را با قرار دادن شکلک "F" همراه رابطه‌ی R و S، و نزدیک شیء R قرار می‌دهیم.



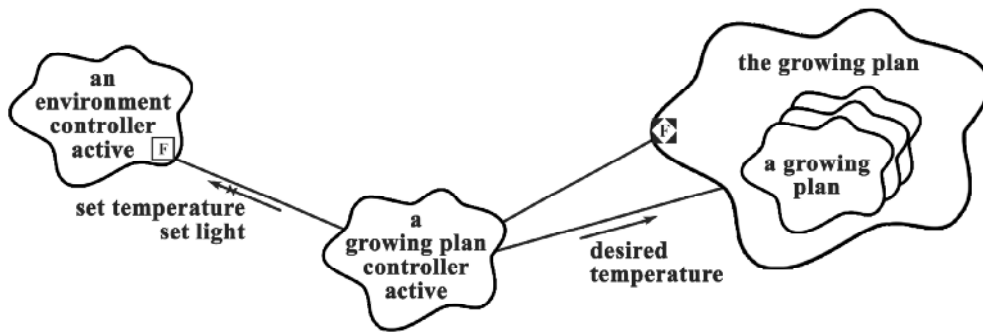
شکل ۵-۱۴: مثالی برای قابلیت رؤیت یک شیء برای شیء دیگر

با قرار دادن مناسب شیء‌ها می‌توانیم به درک خود از قابلیت رؤیت در بین شیء‌ها کمک کنیم. مثلاً، نوعاً شیء‌های خواهان و فعال (actor) را در بالای این نمودارها و شیء‌های خدمتگذار (server) را در پایین قرار می‌دهیم. همین‌طور برای نمایش تجمع و انبوهی (aggregation)، ممکن است شکلک شیء را درون دیگری جای دهیم.

هماهنگی پیام‌ها (Message Synchronization)

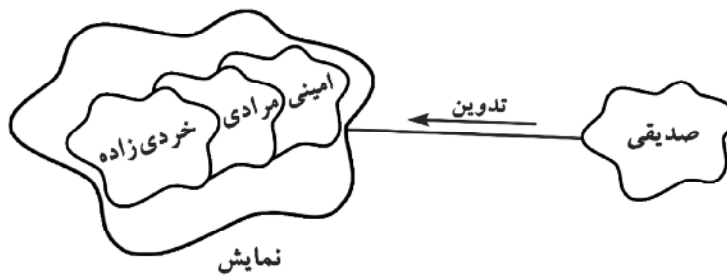
ثانیاً: نمایش چگونگی تراکنش شیء‌ها نیز مهم است. علامت‌گذاری ما اجازه می‌دهد تا پیام‌های ارسالی از شیء S به شیء R را (با قرار دادن جهت روی رابطه و با برچسب‌گذاری اسم پیام) نمایش دهیم. جهت این خط، شیئی را نشان می‌دهد که روی آن عمل شده است. گرچه داده‌ها ممکن است در همان جهت یا جهت مخالف، جریان یابند (هم جهت در صورتی که یک پارامتر به عمل باشد، و در خلاف جهت اگر این داده حاصل فراخوان تابع باشد).

مثالی از یک نمودار شیء. شکل زیر مثالی از نمودار شیء است. شیئی که به نام a growing plan controller می‌باشد دو پیام set temperature و set lights را برای شیئی به نام an environment controller می‌فرستد. ارسال پیام بین این شیء فعال به صورت همگام (synchronous) می‌باشد. به‌علاوه نشان داده شده است که an environment controller فیلدی از a growing plan، می‌باشد، اما مشترک است، به این معنی که برای شیئی که توسط این فیلد مشخص شده است (aliases) وجود دارد. مجموعه‌ی the growing plan هم فیلدی از a growing plan controller می‌باشد، ولی فیلدی غیرمشترک است، به این معنی که شناسه‌ی (identity) این شیء هیچ‌جا دیگر شناخته شده نیست. شیئی که دارای نام the growing plan است، نمایشگر یک مجموعه، شامل شیء‌های متمایز growing-plan می‌باشد، و برای این که نمایش داده شود که آنها nested هستند، آنها در شکلک the growing plan گذاشته شده‌اند. می‌توان از همین روش برای نمایش تودرتویی کلاس‌ها، همه‌بهرهای کلاس‌ها و مؤلفه‌ها استفاده کرد. در این طرح شیء a growing plan controller ممکن است پیام (انتخاب a selector) desired temperature را به هر یک از growing plan‌ها بفرستد. در اینجا معانی انتقال پیام‌ها ساده است (simple)، به‌دلیل این که شیء‌های growing-plan ردیفی هستند.

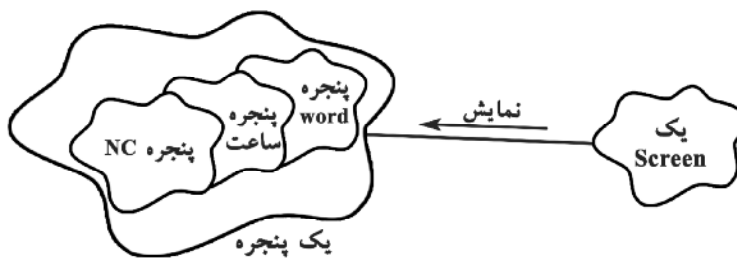


شکل ۵-۱۵: یک نمودار شیء

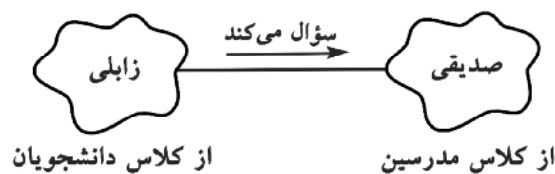
مفهوم ضرورت همگام سازی. در مواردی که بیش از یک خط کنترلی وجود دارد.



شکل ۵-۱۶: مثالی برای نمودار شیء



شکل ۵-۱۷: مثالی برای نمودار شیء



شکل ۵-۱۸: مثالی برای نمودار شیء

الگوی نمودار شیء (Object Diagram Templates)

شیء ها و پیام های نمودار شیء هر یک الگویی برای ارائه ی اطلاعات بیشتر دارند. الگوی شیء (object template). جدول زیر الگو برای شیء است. این الگوی کلاس شیء، که گویای عملیاتی است که مشتری ها (client) ممکن است روی آن شیء، در نمودار شیء انجام دهند را مستند می کند. این الگو در عین حال بیان ماندگاری (persistence) شیء را میسر می سازد. خصوصیتی از شیء که باید با ویژگی ماندگاری که در کلاس مربوطه تعریف شده است، مطابقت داشته باشد. به ویژه اگر در الگوی کلاس مربوطه ذکر شده است که، تمام رویدادها گذرا (transitory) هستند، در این صورت ماندگاری این شیء فقط می تواند ایستا (static)، (یعنی شیء در تمام طول اجرای شیء ها و پیام های نمودار شیء هر یک الگویی برای ارائه ی اطلاعات بیشتر دارند.

برنامه وجود دارد)، یا پویا، (یعنی شیء در حین اجرای برنامه به صورت پویا ایجاد و نابود می‌گردد) باشد. اگر الگوی کلاس مبین این باشد که رویدادها ممکن است ماندگار باشند، در این صورت شیء می‌تواند ایستا، پویا، و یا ماندگار باشد (یعنی بعد از اتمام برنامه‌ای که طی آن، ایجاد شده بود، وجود داشته باشد).

جدول ۵-۱۳: الگوی نمودار شیء

: identifier	Name	اسم	: معرف
: text	Documentation	مستند	: متن
: class - name	Class	کلاس	: نام کلاس
: persistence / static / dynamic	Persistence	ماندگاری	: ماندگار / ایستا / پویا

الگوی پیام (message template). این الگو اطلاعات بیشتری از عمل تعریف شده برای کلاس یک شیء که به نوبه‌ی خود جزئیات بیشتری از معنای عمل است، را ارائه می‌نماید. می‌توان اطلاعات زمانی را نیز بیان کرد. مثل این که آیا این پیام به صورت دوره‌ای (periodically) ارسال می‌شود و یا نه، و اگر می‌شود با چه نرخی (how often). این جزئیات بخصوص در طراحی کاربردهایی که زمان در آنها حیاتی است، بسیار مفید است.

جدول ۵-۱۴: الگوی پیام

operation name	Operation :	عمل	: نام عمل
text	Document :	مستند	: متن
periodic / periodic	Frequency :	تکرار	: ادواری / غیر ادواری
simple / synchronous / balking / timeout / asynchronous	Synchronization :	همگام سازی	: ساده / همگام / ترک کننده / انقضا / غیرهمگام

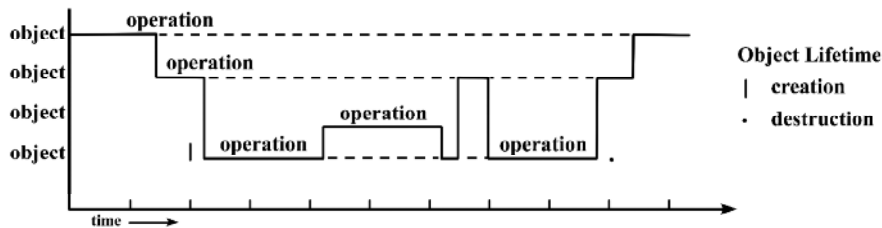
اضافه کردن اطلاعات زمانی در اینجا، ساخت ابزارهای تحلیلی برای تعیین زمان لازم برای تکمیل عملیات و برای ارزیابی مکانیزم‌ها را میسر می‌سازد.

۵-۵ نمودارهای زمانی (Timing Diagrams)

نمودارهای شیء به خودی خود ایستا هستند: مجموعه‌ای از شیء‌ها را که با یکدیگر همکاری می‌کنند و به یکدیگر پیام می‌فرستند، را نشان می‌دهند. اما نمودارهای شیء، جریان کنترل (flow of control) و یا ترتیب حوادث (events) را نشان نمی‌دهند. نمودارهای تغییر حالت کلاس‌ها هم کمکی نمی‌کنند؛ چرا که تنها چگونگی انجام تغییرات در یک شیء واحد را نشان می‌دهد و نه تغییرات مجموعه‌ای از شیء‌هایی که با یکدیگر همکاری می‌کنند. برای مستندسازی غالب مکانیزم‌ها، اطلاعات دیگری لازم است، لذا در علامت‌گذاری خود سه راه ممکن برای مستندسازی پویایی انتقال پیام‌ها در نمودار شیء را منظور می‌کنیم:

- دیدگاه اول خیلی ساده است. هر یک از پیام‌ها در نمودار شیء را شماره‌گذاری می‌کنیم به گونه‌ای که مبین ترتیب نسبی آنها برای فعال شدن (invoke) باشد. ابتدا پیام ۱ ارسال می‌شود؛ سپس پیام ۲ و الی آخر. این روش برای یک گردش کنترل کاملاً ردیفی کار می‌کند، اما اگر بخواهیم گردش کنترل شرطی را بیان کنیم کافی نیست (مثلاً اگر شرط C برقرار بود پیام M ارسال شود، در غیر این صورت پیام N ارسال شود)؛
- در دیدگاه دوم، باهر نمودار شیء PDL ≡ Program Description Language را هم برای بیان ترتیب حادثه‌ها در نظر می‌گیریم. برای اغلب طرح‌ها، PDL انتخاب می‌شود؛ چرا که گویا (expressive)، قابل فهم و مساعد برای خودکارسازی است.

۳. دیدگاه سومی را که مفید یافته‌ایم، ریشه در نمودارهای زمانی (که توسط مهندسين سخت‌افزار به کار می‌رود) دارد. نمودار زمانی (timing diagram) گرافیکی است که زمان (واقعی یا نسبی) را روی محور افقی و شیء‌ها را روی محور عمودی در نظر می‌گیرد. مثل شیء‌ها، تنها مواردی را در نظر می‌گیریم که تراکنش متقابل آنها مستندسازی یک مکانیزم باشد. با حرکت از چپ به راست ممکن است عملی فعال `invoke` شود. به‌عنوان مثال ممکن است با عمل ۱ برای شیء `R` شروع کنیم، با گذشت زمان، این عمل، عمل ۲ را برای شیء `S` فعال می‌کند؛ که به نوبه‌ی خود پیام ۳ را برای شیء `T` می‌فرستد. خط‌چین‌ها مبین تودرتویی (nesting) پیام‌هاست. به‌عنوان مثال، با کامل شدن عمل ۳، کنترل به عمل ۲ بازمی‌گردد، که در این صورت آزاد است که چیزهای دیگر را انجام دهد.

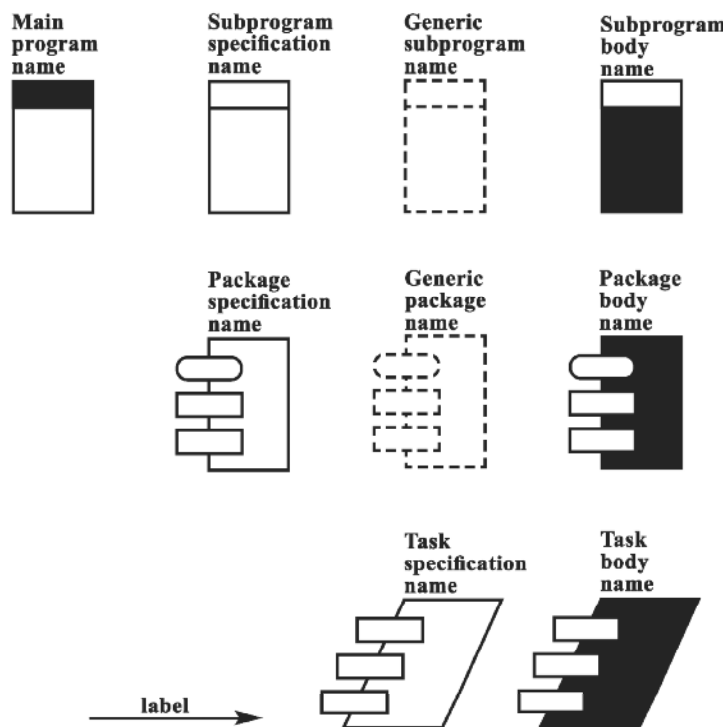


شکل ۵-۱۹: نمودار زمانی نمودار زمانی قسمتی از نمودار شیء و بیانگر ترتیب بروز حوادث در بین مجموعه‌ای از شیء‌هاست

۵-۶ نمودارهای مؤلفه (Module diagrams)

از نمودارهای مؤلفه برای نمایش انتساب کلاس‌ها و شیء‌ها به مؤلفه‌ها، در طرح فیزیکی سیستم، استفاده می‌شود. در بعضی زبان‌ها مثل `C++` از ساختار مؤلفه حمایت می‌شود.

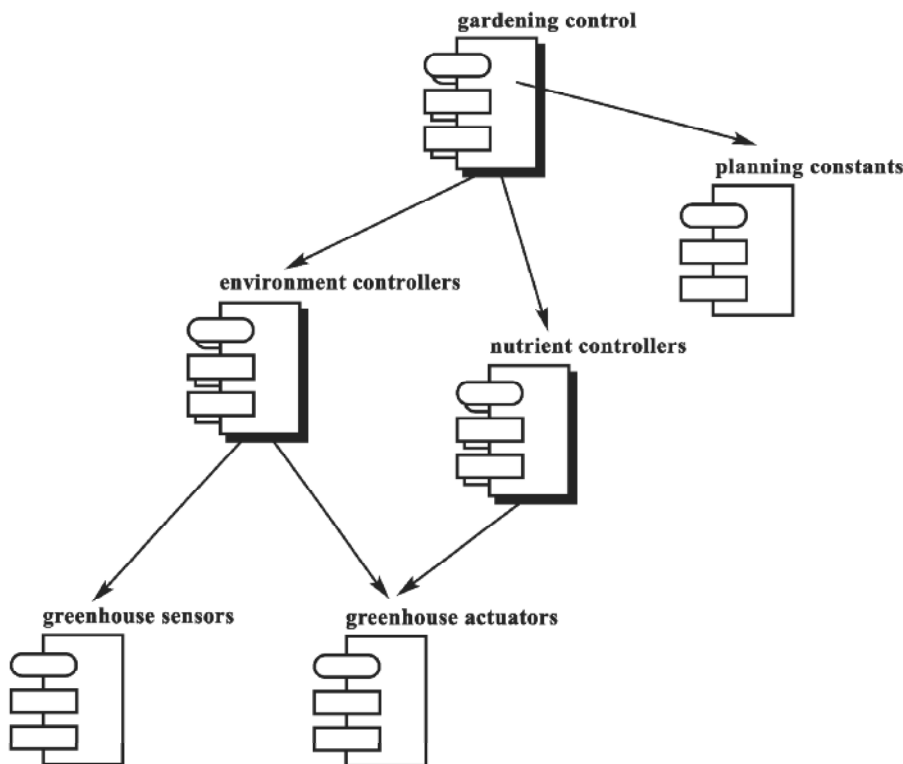
دو عنصر مهم معماری مؤلفه‌ها (module architecture)، مؤلفه‌ها و قابلیت رؤیت مؤلفه (module visibility) هستند. مؤلفه‌ها، شکل زیر بیانگر شکلک‌های انواع مؤلفه‌هایی است که در نمودار مؤلفه از آنها استفاده می‌کنیم.



شکل ۵-۲۰: نمودارهای مؤلفه

قابلیت رؤیت مؤلفه‌ها (Module visibility)

تنها رابطه‌ی ممکن بین دو مؤلفه وابستگی کامپایل (compilation dependency) است که توسط کمائی نمایش داده می‌شود. ممکن است با اضافه کردن یک برچسب (label) معنای این رابطه را مستند کنیم. به‌عنوان مثال برای نمایش این که مؤلفه‌ی G به مؤلفه‌ی H وابسته است، یک کمان از G به H رسم می‌کنیم، به این معنی که H برای G قابل رؤیت است.



شکل ۵-۲۱: یک نمودار مؤلفه^۱

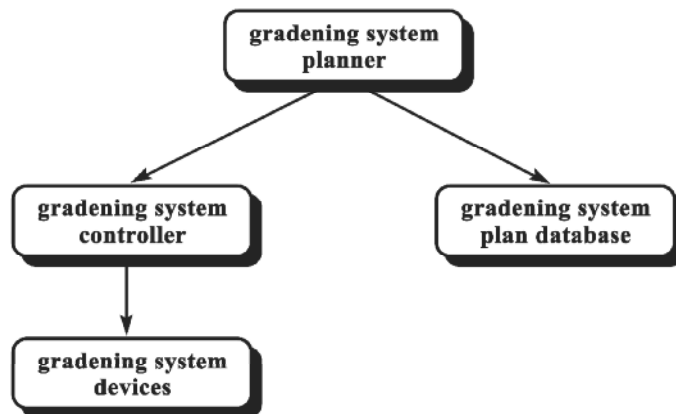
زیرسیستم‌ها (Subsystems)

یک سیستم بزرگ ممکن است صدها مؤلفه داشته باشد. در چنین مواردی برای درک بهتر سیستم، از زیرسیستم‌ها استفاده می‌شود. زیرسیستم‌ها نمایشگر خوشه‌هایی (clusters) از مؤلفه‌های مرتبط به یکدیگرند و برای برنامه‌سازی در سطح کلان (programming-in-the-large) بسیار کمک‌کننده هستند. متأسفانه زبان‌های برنامه‌سازی موجود از این ساختار حمایت نمی‌کنند. گرچه محیط‌های تولید نرم‌افزار (software development environment) از این ساختار حمایت می‌کنند. در عمل، یک سیستم بزرگ در بالاترین سطح تجرید دارای (مؤلفه‌هایی از) زیرسیستم است. معماری فیزیکی بزرگتر برای کل سیستم است. در اینجا هر زیرسیستم نمایشگر یک نمودار مؤلفه است (مشابه گروه کلاس‌ها که قبلاً در همین فصل مطرح شد).



شکل ۵-۲۲: شکلک برای زیرسیستم

^۱ module diagram



شکل ۵-۲۳: یک نمودار مؤلفه‌ی سطح بالا (نمودار زیرسیستم‌ها)^۱

الگو برای نمودار مؤلفه (Module Diagram Templates)

شامل اجزاء زیر:

جدول ۵-۱۵: الگو برای نمودار مؤلفه

اسم	Name :	identifier
مستندات	Documentation :	text
تعاریف : فهرست تعاریف	Declarations :	list of declaration

معانی پویای الگوی مؤلفه (Module Template Dynamic Semantics)

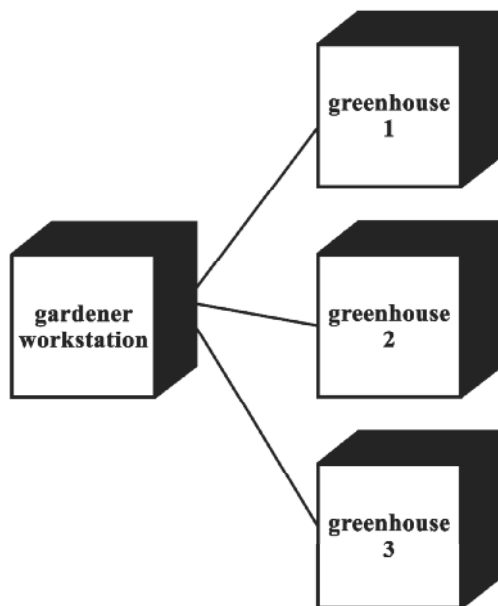
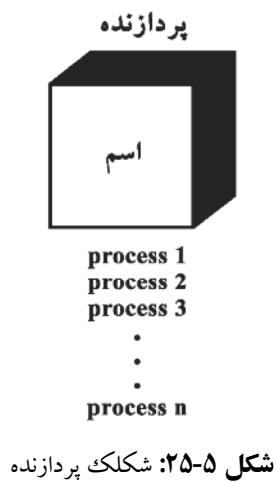
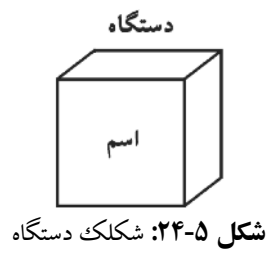
تمام نمودارهایی که تا به حال بررسی کرده‌ایم، هم دارای معانی ایستا و هم دارای معانی پویا هستند؛ گرچه نمودارهای مؤلفه نوعاً ایستا هستند. با این حال در بعضی کاربردها، ممکن است مؤلفه‌ها دارای معانی پویا هم باشند. به عنوان مثال برای کاربردی که حافظه‌ی مورد نیاز را در اختیار ندارد، ممکن است برای اجرا از swap کردن کد مؤلفه‌های مختلف به حافظه و overlay استفاده کنیم. این چنین معانی پویا را می‌توان توسط نمودارهای زمانی (timing diagram) و یا Program Description Language \equiv PDL تشریح کنیم.

۵-۷ نمودار فرایند (Process Diagram)

معماری فرایند (process architecture). در سیستم‌های بزرگ ممکن است از بیش از یک کامپیوتر استفاده کنیم. از نمودار فرایند برای نمایش فرایندها (processes) به پردازنده‌ها استفاده می‌شود. و این نمودار هم بیانگر طرح فیزیکی (physical design) سیستم است. حتی برای سیستم‌هایی که روی تنها یک پردازنده اجرا می‌شوند نیز نمودارهای فرایند مفید هستند، اگر پیاده‌سازی ما متضمن دستگاه یا شیء‌های فعال (active objects) باشد. به این معنی که چند پردازش (یا چند خط کنترل) در یک زمان وجود دارد.

شکل زیر شکلهای نمایشگر پردازنده‌ها و دستگاه‌ها و مثالی از نمودار فرایند را نشان می‌دهد.

^۱ top-level module diagram



شکل ۵-۲۶: یک نمودار فرایند

الگو برای پردازنده

جدول ۵-۱۶: الگو برای پردازنده

Processor		
Name :	identifier	اسم :
Documentation :	text	مستندات :

Characteristics :	text	مشخصات پردازنده :
Processor :	list of processes	لیست فرایندهای منتسب به این پردازنده :
Scheduling :	preemptive / non preemptive / cycle ¹ / executive ² / manual ³	چگونگی زمان بندی :

الگوی پراسس

جدول ۵-۱۷: الگوی پراسس

Processor		
Name :	identifier	اسم :
Documentation :	text	مستندات :
Priority :	integer	اولویت این پراسس :

الگوی دستگاه

جدول ۵-۱۸: الگوی دستگاه

Device		
Name :	identifier	اسم :
Documentation :	text	مستندات :
Characteristics :	text	مشخصات این دستگاه :

۵-۸ محصول طراحی شیء گرا

محصول اساسی طراحی شیء گرا، نوعاً مجموعه‌ای از نمودارهای کلاسی، نمودارهای شیء، نمودارهای مؤلفه و نمودارهای فرایند است. در رأس طراحی ما، بالاترین نمودار کلاس (بیانگر تجربدهای اصلی در طرح منطقی سیستم)، بالاترین نمودار مؤلفه (بیانگر اجزاء کلیدی از ایجاد فیزیکی سیستم) و نمودار اصلی فرایند (نمایشگر اجزاء کلیدی طرح فیزیکی سخت افزار) می باشد. مکانیزم‌های به کار رفته در پیاده‌سازی ما، در نمودارهای شیء مطرح شده‌اند. مسلماً استفاده از ابزار برای تهیه‌ی این نمودار کمک کننده است.

تغییر مقیاس (Scaling Up and Sealing Down)

ما دریافته‌ایم که این علامت گذاری هم برای سیستم‌های کوچک (چند صد خطی) و هم سیستم‌های بسیار بزرگ (چند میلیون خطی) قابل به کار گیری است. نکته‌ی دیگر این که این علامت گذاری مستقل از زبان برنامه‌سازی مورد استفاده در پیاده‌سازی بوده و برای زبان‌های مختلف شیء گرا قابل به کار گیری است.

۵-۹ خلاصه‌ی فصل پنجم کتاب طراحی شیء گرا / علامت‌گذاری (Notation)

- طراحی عبارت از رسم نمودار نیست؛ نمودار یک طرح را دربر دارد؛

^۱ هر پراسس محدودی مشخصی از زمان دارد

^۲ زمان بندی با الگوریتمی خاص

^۳ زمان بندی توسط کاربر از خارج سیستم

- در طراحی یک سیستم پیچیده، این مهم است که از چند نقطه نظر، مورد توجه قرار گیرد: از نظر ساختار منطقی، از نظر ساختار فیزیکی، از نظر معانی ایستا (static semantics) و از نظر معانی پویا (dynamic semantic)؛
- علامت‌گذاری برای طراحی شیء‌گرا شامل چهار نمودار اصلی (نمودارهای کلاس، نمودارهای شیء، نمودارهای مؤلفه و نمودارهای فرایند) و همین‌طور دو نمودار کمکی (نمودار تغییر حالت و نمودار زمان‌بندی) می‌باشد؛
- نمودار کلاس (class diagram) برای نشان دادن وجود کلاس‌ها و روابط بین آنها در طرح منطقی سیستم به کار می‌رود؛ یک نمودار کلاس، نمایشگر تمام یا بخشی از ساختار کلاس (class structure) از یک سیستم است؛
- نمودار شیء (object diagram) برای نشان دادن وجود شیء‌ها و روابط بین آنها در طرح منطقی سیستم به کار می‌رود؛ یک نمودار شیء، نمایشگر تمام یا بخشی از ساختار شیء (object structure) از یک سیستم بوده و بیانگر معانی تجزیه‌های کلیدی در طرح منطقی است. یک نمودار شیء واحد، نمایشگر یک لحظه‌ی (snapshot) زمانی از شیء و در غیراین صورت نمایشگر یک حادثه‌ی گذرا (transitory event) یا پیکربندی (configuration) از شیء است؛
- نمودار مؤلفه (module diagram) برای نشان دادن تخصیص کلاس‌ها و شیء‌ها به مؤلفه‌ها در طرح فیزیکی (Physical design) یک سیستم است؛ یک نمودار مؤلفه نمایشگر تمام یا بخشی از معماری مؤلفه‌ای (architecture of a system) یک سیستم است؛
- نمودار فرایند (process diagram) برای نشان دادن تخصیص فرایندها به پردازنده‌ها در طرح فیزیکی یک سیستم به کار می‌رود؛ نمودار فرایند نمایشگر تمام یا بخشی از معماری فرایند (process architecture) از یک سیستم است؛
- نمودار تغییر حالت (state transition diagram) برای نشان دادن فضای حالت از یک رویداد، از یک کلاس، داده شده، و برای نمایش حادثه‌ای که موجب انتقال از یک حالت به حالت دیگر شده، و عمل‌هایی که از این تغییر حالت نتیجه می‌شوند، به کار می‌رود؛
- نمودار زمانی (timing diagram) برای نشان دادن تراکنش‌های پویا (dynamic transaction) بین شیء‌های مختلف در نمودار شیء به کار می‌رود؛
- **تذکر مهم:** یک دلیل پرداختن به این همه جزئیات در این متن فشرده، توجه دادن به مجموعه مواردی است که در طراحی شیء‌گرا می‌تواند (و بهتر است) مورد توجه قرار گیرد.

فصل ۶

فرایند طراحی شیء گرا (The Process)

طراحی شیء گرا فرایندی نیست که با یک مشخصات خواسته‌ها شروع، و با گزارشی برای پیاده‌سازی تمام شود. با تجربه‌ی ما فرایند طراحی شیء گرا عموماً ترتیب زیر از وقایع را دنبال می‌کند:

- شناسایی و تعیین (identify) کلاس‌ها و شیء‌ها در یک سطح مشخصی از تجزیه؛
- شناسایی و تعیین معانی این کلاس‌ها و شیء‌ها؛
- شناسایی و تعیین روابط بین این کلاس‌ها و شیء‌ها؛
- پیاده‌سازی این کلاس‌ها و شیء‌ها.

طراحی شیء گرا، یک فرایند افزایش یابنده است: شناسایی و تعیین کلاس‌ها و شیء‌های جدید معمولاً موجب می‌شود که ما معانی و روابط کلاس‌ها و شیء‌های موجود را پالایش کرده و ارتقاء دهیم. طراحی شیء گرا در عین حال تکرار شونده (iterative) است: پیاده‌سازی کلاس‌ها و شیء‌ها غالباً ما را به کشف یا ابداع کلاس‌ها و شیء‌های جدید هدایت می‌کند، کلاس‌ها و شیء‌هایی که حضورشان موجب ساده شدن و عمومی تر شدن طرح ما می‌شود. در ادامه هر یک از گام‌های فوق مورد بررسی قرار می‌گیرد.

۶-۱ شناسایی و تعیین کلاس‌ها و شیء‌ها (Identifying Classes and Objects)

اولین گام شامل دو فعالیت است:

- کشف تجربدهای کلیدی در فضای مسأله (کلاس‌ها و شیء‌های مهم)؛
 - ابداع مکانیزم‌های مهم (ساختارهایی که به وسیله‌ی آنها مجموعه‌ای از شیء‌ها با یکدیگر همکاری کرده و رفتارهایی را ایجاد کنند که جواب گوی (بخشی) از خواسته‌های مسأله باشند).
- این کار با استفاده از تکنیک‌هایی که در فصل ۴ مطرح شده انجام می‌شود. اصولاً تولیدکننده باید به‌عنوان یک تجزیدگر (abstractionist) عمل کند. باید لغت‌نامه‌ی زمینه‌ی مسأله را (با مطالعه‌ی خواسته‌های مسأله و با ارتباط با خبرگان محیط مسأله) یاد بگیرد.

محصولات اولین گام

محصولات این گام می‌تواند خیلی رسمی تا خیلی غیر رسمی باشد. به‌صورت غیر رسمی می‌تواند تنها فهرستی از اسامی با معنی شیء‌ها و کلاس‌های مسأله باشد. از سوی دیگر، می‌تواند به‌طور رسمی مبین معانی این تجربدها و مکانیزم‌ها با پر کردن الگوهای مربوطه باشد (الگوهای علامت‌گذاری در فصل پنجم آمده است).

گام اول.

در اکثر موارد این گام وقت چندانی نمی‌گیرد. غالباً یک طراح ارشد، فهرستی از کلاس‌ها و شیء‌های کاندید را پیشنهاد می‌کند. اگر تفکیک فیزیکی (physical decomposition) سیستم مهم باشد (همچنان‌که در موارد موجود در زیرقراردادها (sub contracts) مهم است)، در این صورت، پیش‌نویس‌های طرح‌های مؤلفه‌ها (module design) نیز در این زمان تهیه می‌شود. و بالاخره اگر عوامل سخت‌افزاری برای سیستم وجود داشته باشد، در این صورت طرح اولیه‌ی نمودارهای فرایند (process diagrams) نیز مفید است.

۶-۲ شناسایی و تعیین معانی کلاس‌ها و شیء‌ها

(Identifying the Semantics of Classes and Objects)

گام دوم شامل یک فعالیت اصلی است:
تعیین معنی‌های کلاس‌ها و شیء‌های مشخص شده در گام اول.
در اینجا تولیدکننده‌ی نرم‌افزار به صورت یک ناظر خارجی و مستقل عمل می‌کند و سعی می‌کند به هر کلاس از نقطه نظر واسط آن (interface) نگاه کند؛ تا چیزهایی که ما می‌توانیم با هر رویداد (instance) از هر کلاس انجام دهیم و همچنین چیزهایی را که هر شیء می‌تواند با شیء دیگر انجام دهد را شناسایی و تعیین نماییم.
این گام از گام قبلی بسیار مشکل‌تر و وقت‌گیرتر است. پیدا کردن کلاس‌ها و شیء‌ها ساده است؛ تصمیم‌گیری در مورد پروتکل هر شیء مشکل است. به همین دلیل فرایند طراحی شیء‌گرا در این نقطه تکرار شونده می‌گردد. تصمیم‌گیری در مورد پروتکل یک شیء، ممکن است به تغییر شیء دیگر منجر گردد. عملاً وجود تجربدهای کلیدی ما تغییر نمی‌کند، بلکه ما تنها حدود آنها را تغییر می‌دهیم.
یک روش خوب برای هدایت این فعالیت‌ها، نوشتن متنی در مورد هر شیء است. متنی که چرخه‌ی زندگی هر شیء از ایجاد تا مرگ و رفتار آن تعریف کند.

محصولات این مرحله

محصولات این مرحله ماهیت افزایش‌یابنده‌ی طراحی شیء‌گرا را منعکس می‌کنند. برای مستندسازی تصمیم‌ها در مورد معنی هر شیء و کلاس، ما عموماً الگوهایی که در مرحله‌ی قبل (مطابق قواعد علامت‌گذاری) رسم کرده‌ایم را پالایش (refine) می‌کنیم. ما باید تمام معانی ایستا و پویای هر تجرید کلیدی و مکانیزم را به بهترین وجهی که می‌توانیم به موقع مستند کنیم. ممکن است نمودارهای شیء (object diagrams) جدید را هم برای مستند کردن مکانیزم‌هایی که احتمالاً در این مرحله ابداع کرده‌ایم، رسم کنیم. در اینجا ممکن است در مورد نمونه‌سازی بعضی از قسمت‌های طرح تصمیم بگیریم.

۶-۳ شناسایی و تعیین روابط بین کلاس‌ها و شیء‌ها

(Identifying the Relationship Among Classes and Objects)

فعالیت‌ها: گام سوم عمدتاً بسط فعالیت‌های گام قبلی است. در اینجا بطور دقیق تعیین می‌کنم که چگونه چیزها در داخل سیستم با یکدیگر در تراکنش هستند. در رابطه با تجربدهای کلیدی، این بدین معنی است که ما باید در مورد روابط استفاده (using)، وراثت (inheritance) و انواع دیگر رابطه بین کلاس‌ها اظهار نظر قطعی کنیم. در رابطه با شیء‌ها در پیاده‌سازی ما، این بدین معنی است که ما باید معانی ایستا و پویای هر مکانیزم را مشخص کنیم.
در اینجا دو فعالیت مربوط وجود دارد که ممکن است موجب تجدید نظر در محصولات قبلی ما گردد:
۱. اول این که ما باید الگوها (patterns) را کشف کنیم: الگوها در میان کلاس‌ها، که موجب می‌شود ما ساختار کلاس سیستم (systems class structure) را درک و ساده کنیم؛ و الگوها در بین مجموعه‌های شیء‌های همکاری‌کننده، که ما را به تعمیم مکانیزم‌هایی که هم‌اکنون در طرح ما وجود دارند هدایت می‌کند. این قسمت از طراحی تمام مهارت‌های خلاق طراح را می‌طلبد. توفیق در این قسمت است که یک طرح خوب را از یک طرح عالی متمایز می‌کند؛

۲. تصمیم در مورد قابلیت رؤیت: دوم این که ما باید در مورد قابلیت رؤیت (visibility) تصمیم بگیریم: این که کلاس‌ها چگونه یکدیگر را می‌بینند، چگونه شیء‌ها یکدیگر را می‌بینند، و مطلب دیگری که به همان اندازه مهم است این که چه کلاس‌هایی و چه شیء‌هایی یکدیگر را نباید ببینند. این تصمیم‌ها به ما کمک می‌کند تا در مورد دسته‌بندی (packaging) در طراحی معماری مؤلفه‌های (modal architecture) سیستم تصمیم بگیریم.

کشف الگوها و تصمیم‌ها در مورد قابلیت رؤیت ممکن است موجب تجدید نظر در مورد پروتکل‌های کلاس‌های مراحل مختلف قبل گردد.

استفاده از کارت‌های (کلاس-مسئولیت-همکاری) \equiv CRC \equiv (Class, Responsibility, Collaboration) برای هدایت این فعالیت‌ها مفید است. هر کارت "کلاس-مسئولیت-همکاری"، یک کارت 5×3 اینچ فیش برداری است که برای هر کلاس از شیء‌ها تهیه می‌شود. در این کارت نام کلاس، مسئولیت‌های آن کلاس (رفتار آن کلاس)، و کلاس‌های دیگری که این کلاس با آنها همکاری می‌کند قید می‌گردد. طراح می‌تواند این کارت‌ها را به صورت فضایی سازمان داده و محتوای آنها را بر اساس یادداشت‌های چرخه‌ی عمرش (که در مرحله‌ی قبل ذکر شد) تغییر دهد.

تأکید می‌کنیم که در این مرحله از طراحی نیز هنوز تمرکز ما روی دید خارجی از تجربه‌های کلیدی و مکانیزم‌های موجود می‌باشد.

محصولات مرحله‌ی سوم

شامل تکمیل مدل‌های منطقی طراحی است. در اینجا نمودارهای کلاس از مراحل قبل را با توجه به الگوی کشف شده و تصمیم‌های قابلیت رؤیت، پالایش می‌کنیم. همچنین جزئیات نمودارهای شیء (که مکانیزم‌های اصلی را مستند می‌کنند) را تکمیل می‌کنیم. برای ایجاد این نمودارها از علامت‌گذاری فصل پنجم استفاده می‌کنیم. ابتدا شیء‌هایی را که می‌دانیم با هم همکاری می‌کنند، را رسم می‌کنیم. سپس هر زوج از شیء‌ها را در نظر گرفته و در مورد رابطه‌ی بین آن‌دو سؤال می‌کنیم. اگر جواب مثبت باشد دو سؤال بعدی را مطرح می‌کنیم که: این دو شیء چگونه به هم مربوط می‌شوند و هر یک چه پیامی برای دیگری می‌فرستد؟ این کار معمولاً به تجدید نظر در مورد پروتکل کلاس‌های مربوطه منجر می‌شود. و این تجدید نظر، به نوبه‌ی خود موجب کشف الگوی دیگر می‌گردد.

در اینجا، ممکن است ایجاد نمودارهای اصلی مؤلفه را شروع کنیم. در این گام ممکن است به ایجاد و توسعه‌ی نمونه‌ها (prototypes) نیز ادامه دهیم. ممکن است نمونه‌های جدیدی را ایجاد کنیم و یا نمونه‌های قبل را پالایش کنیم.

۴-۶ پیاده‌سازی کلاس‌ها و شیء‌ها (Implementing Classes and Objects) گام چهارم.

فعالیت‌ها: چهارمین گام الزاماً آخرین گام نیست. این گام شامل دو فعالیت است:

- تصمیم‌گیری‌های طراحی با توجه به نمایش کلاس‌ها و شیء‌هایی که ابداع کرده‌ایم؛
- تخصیص کلاس‌ها و شیء‌ها به مؤلفه‌ها، و تخصیص برنامه‌ها به پردازنده‌ها.

اینجا اولین جا در طراحی است که به دید داخلی هر کلاس و مؤلفه می‌پردازیم، تا در مورد چگونگی پیاده‌سازی رفتار آن، تصمیم بگیریم.

به جز در مورد تجربه‌ها و مکانیزم‌های بسیار ابتدایی، معمولاً در این مرحله ناچار به بازگشت به مرحله‌ی اول می‌شویم، و این فرایند را دوباره در دید داخلی کلاس‌ها و مؤلفه‌های موجود اعمال می‌کنیم.

محصولات: در این گام از طراحی، ما باید واسطه‌ی دقیق هر کلاس یا شیء مهم را (در سطحی از تجرید) مشخص کرده باشیم. لذا محصولات ما شامل پالایش ساختار کلاس، و بخصوص تکمیل قسمت پیاده‌سازی از الگوی کلاس می‌باشد. تکمیل قسمت‌های مشابه از نمودار مؤلفه، و در صورت لزوم، در نمودار فرایندهای سیستم.

۵-۶ خلاصه‌ی فصل ششم از کتاب طراحی شیء‌گرا / فرایند طراحی شیء‌گرا

(The Process)

- فرایند طراحی شیء‌گرا نه بالا به پایین و نه پایین به بالا است؛ بلکه بهترین بیان برای طراحی شیء‌گرا، طراحی مرکب دوره‌ای (round-trip gestalt design) می‌باشد، که ایجاد و توسعه‌ی سیستم به‌صورت اضافه شونده/فزاینده (incremented) و تکرار شونده (iterative)، از طریق پالایش دیدگاه‌های متفاوت و در عین حال سازگار و هماهنگ (consistent) به تمامیت سیستم، صورت می‌گیرد؛
- فرایند طراحی شیء‌گرا با کشف کلاس‌ها و شیء‌های زمینه‌ی مسأله شروع می‌شود؛ و هنگامی پایان می‌یابد که ما دریابیم هیچ تجرید ابتدایی (primitive) و مکانیزم جدیدی وجود ندارد، یا هنگامی که کلاس‌ها و شیء‌هایی که تا حالا کشف کرده‌ایم بتوانند به‌وسیله‌ی ترکیب مؤلفه‌های نرم‌افزاری موجود پیاده‌سازی شوند؛
- اولین گام در فرایند طراحی شیء‌گرا شامل شناسایی و تعیین کلاس‌ها و شیء‌ها در یک سطح داده شده از تجرید است؛ در اینجا مهمترین فعالیت‌ها عبارتند از کشف تجریدهای کلیدی و ابداع مکانیزم‌های مهم؛
- دومین گام شامل شناسایی و تعیین معانی (semantics) این کلاس‌ها و شیء‌هاست؛ کار مهم تولیدکننده در اینجا اینست که به‌عنوان یک بیگانه‌ی مستقل و مجزا، به هر یک از کلاس‌ها از نقطه نظر واسط (interface) نگاه کند؛
- سومین گام شامل شناسایی و تعیین رابطه‌ی بین این کلاس‌ها و شیء‌هاست؛ در اینجا با توجه به معانی ایستا و پویای تجریدهای کلیدی و مکانیزم‌های مهم، چگونگی تعامل چیزها با سیستم را برقرار و مشخص می‌کنیم؛
- چهارمین گام شامل پیاده‌سازی این کلاس‌ها و شیء‌هاست؛ کار مهم در اینجا شامل انتخاب یک نمایش (representation) برای هر یک از کلاس‌ها و شیء‌ها، و تخصیص کلاس‌ها و شیء‌ها به مؤلفه‌ها، و تخصیص برنامه‌ها به فرایندهاست؛ این گام الزاماً گام آخر نیست، برای تکمیل کردن، معمولاً لازم است که، تمام فرایند را، برای سطح پایین‌تری از تجرید، تکرار کنیم.

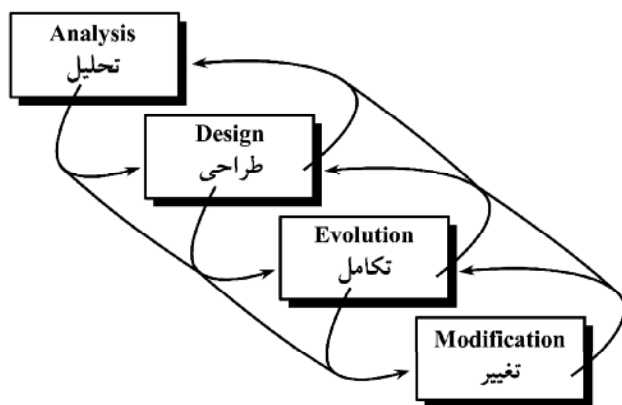
فصل ۷

نکات علمی (Pragmatics)

در این فصل نکات علمی در طراحی شیء گرا، نقش این روش در چرخه‌ی عمر نرم‌افزار و تأثیر آن در مدیریت، مورد بررسی قرار می‌گیرد.

۷-۱ طراحی شیء گرا در چرخه‌ی عمر نرم‌افزار

(Object-Oriented Design in the Life Cycle)



شکل ۷-۱: طراحی شیء گرا در چرخه‌ی تولید نرم‌افزار

تحلیل (Analysis)

فعالیت‌ها (the activates of analysis): حادّ بین تحلیل و طراحی می‌تواند گاهی مشکک (Fuzzy) باشد. به‌عنوان مثال تجزیه‌های کلیدی از زمینه‌ی مسأله را می‌توان هم به‌عنوان قسمتی از تحلیل و یا قسمتی از طراحی دانست. ولی هدف‌های تحلیل و طراحی کاملاً متفاوت هستند. در تحلیل، ما (با شناسایی کلاس‌ها و شیء‌هایی که فرهنگ زمینه‌ی مسأله را تشکیل می‌دهند) دنبال مدلی از دنیای مسأله هستیم؛ در طراحی شیء گرا، ما تجزیه‌ها و مکانیزم‌هایی را ابداع می‌کنیم که رفتار مورد نیاز این مدل را فراهم کنند. به عبارت دیگر تحلیل، رفتار مطلوب سیستمی که باید بسازیم را به ما می‌گوید، در حالی که طراحی، منشور (blueprints) پیاده‌سازی آن سیستم را به وجود می‌آورد (تعریف تحلیل در فصل دوم آمده است). روش‌های تحلیل (analysis methods): انواع مختلفی از روش‌های تحلیل وجود دارد، که بعضی از آنها برای این که مقدم (front end) طراحی شیء گرا قرار گیرند، مناسب هستند. تحلیل ساخت یافته مقدم جاذبی برای طراحی شیء گراست؛

در تحلیل.

در طراحی.

به عبارت دیگر...

جاذبه‌ی تحلیل ساخت یافته

عمدتاً به خاطر این که متداول ترین روش، طراحی جاری است (1991) و افراد زیادی در این رابطه آموزش دیده اند، با این حال تحلیل ساخت یافته، مقدم بهینه برای طراحی شیء گرا نیست. دانستن این که چه موقع یک نمودار جریان داده ها نمایش مدل مهمی از مسأله را متوقف و در عوض یک تصمیم طراحی را مطرح می کند، مشکل است. به علاوه گاهی تحلیل ساخت یافته می تواند دید الگوریتم گرا (algorithm oriented) به مسأله را برای همیشه باقی گذارد، که این طراحی شیء گرا را بسیار مشکل تر خواهد کرد.

فرایند تحلیل شیء گرا، به طراحی شیء گرا کاملاً نزدیک است. به این معنی که محصول تحلیل شیء گرا تقریباً مستقیماً در ابتدای فرایند طراحی شیء گرا می تواند مورد استفاده قرار گیرد.

روش های تحلیل دیگری مثل تکنیک های تحلیل و طراحی ساخت یافته

(Structured Analysis and Design Techniques ≡ SADT) [16]، نیز به عنوان مقدم طراحی شیء گرا مورد استفاده

قرار گرفته اند.

طراحی (Design)

شروع فرایند طراحی: هرگاه یک مدل رسمی یا غیر رسمی (و احتمالاً ناقص) از مسأله وجود داشته باشد، طراحی شیء گرا می تواند شروع شود. ایده ی مقداری تحلیل، مقداری طراحی (analysis a little design a little) می تواند مورد توجه باشد (فرایند طراحی در فصل ششم مفصل تر بحث شده است).

خاتمه ی فرایند طراحی: تعیین این نکته هم مشکل است. قاعده ی ره یافتی که ما استفاده می کنیم این است که موقعی فرد می تواند طراحی را متوقف کند که تجربه های کلیدی آنقدر ساده هستند که به تفکیک بیشتر نیاز ندارند، ولی می توانند از مؤلفه های نرم افزار قابل استفاده ی مجدد ترکیب گردند.

تکامل (Evolution)

فعالیت های تکامل (the activities of evolution): تکامل یک سیستم در چرخه ی ایجاد توسعه ی نرم افزار به صورت شیء گرا، ترکیبی از جنبه های سنتی برنامه نویسی (coding)، آزمایش (testing) و مجتمع سازی (integration) می باشد. لذا مجتمع سازی یک باره (big-bang) در رابطه با طراحی شیء گرا وجود ندارد. در عوض فرایند ایجاد و توسعه ی سیستم، نتیجه ی تولید اضافه شونده ی دنباله ای از نمونه ها است، که در نهایت به صورت پیاده سازی نهایی ظاهر می شود. این روند افزایش یابنده ی ایجاد و توسعه امتیازاتی دربر دارد:

• بازخورد مهم برای کاربران موقعی فراهم می شود که بیشترین نیاز به آن هست، بیشترین استفاده را دارد، و بیشترین معنی را دارد؛

• کاربران می توانند گونه های مختلفی از اسکلت سیستم را ببینند، و لذا به آرامی از سیستم قدیمی خود به سیستم جدیدشان انتقال یابند؛

• تأخیر در زمان بندی پروژه، بعید است موجب قطع پروژه گردد؛

• واسطه های اصلی سیستم زودتر و بیشتر آزمایش می شوند؛

• آزمایش منابع، عادلانه تر توزیع می گردد؛

• پیاده کنندگان می توانند نتایج زودرس از سیستم در حال کار را ببینند، که موجب بالا رفتن روحیه ی آنها می شود؛

• در صورت کوتاهی زمان، برنامه نویسی و آزمایش می توانند قبل از اتمام طراحی، شروع شوند [19]؛

• طرح های متفاوت و رسیدن به تصمیم های بهتر می تواند مطرح باشد.

ایجاد و توسعه ی سیستم عمدتاً تلاشی برای جواب گویی به تعدادی از محدودیت های رقیب (از جمله عملکرد (functionality)، زمان و فضا) است. شخص همیشه محدود به بزرگترین تنگناهاست. با پیاده سازی تکاملی، ما می توانیم میزان اهمیت هر یک از محدودیت ها را درک کرده و لذا بهتر می توانیم سیستم را در طول زمان تنظیم کنیم.

تحلیل ساخت یافته
بهینه نیست

تناسب تحلیل
شیء گرا

برنامه سازی، آزمایش
و مجتمع سازی سنتی.

مزایای روند افزایش
یابنده و تکرار شونده
مزایای روند افزایش
یابنده و تکرار ونده.

انتقال آرام به سیستم
جدید.

به موقع بودن.

روند مناسب آزمایش.

روحیه ی کاری.

کوتاه کردن زمان.

طرح های متفاوت.

انواع تغییرات تکاملی (Kinds of Evolutionary Charles):

در عمل انواع تغییرات زیر در طرح کلی تکامل تدریجی سیستم مطرح می‌شود:

- اضافه کردن یک کلاس جدید؛
- تغییر پیاده‌سازی یک کلاس؛
- تغییر نمایش یک کلاس؛
- سازماندهی مجدد ساختار کلاس؛
- تغییر واسط یک کلاس.

هر یک از انواع تغییرات فوق دارای دلایل و همچنین هزینه‌های خاص خود است.

تغییر (Modification)

با بلوغ یک سیستم نرم‌افزاری پیاده شده، نکاتی مشاهده می‌شود:

- برنامه‌ای که در محیط واقعی به کار رفته است الزاماً یا باید تغییر کند و یا این که از مفید بودن آن در محیطش کاسته شود قانون تداوم تغییر (the law of continuing elang)؛
- وقتی یک برنامه‌ی متعامل تغییر می‌کند، ساختارش پیچیده می‌شود، مگر اینکه تلاش‌های خاصی برای جلوگیری از این پدیده صورت گیرد قانون ازدیاد پیچیدگی (the law of increasing complexity) [22].
- ما دریافته‌ایم که فعالیت‌های مرحله‌ی تغییر سیستم با فعالیت‌های زمان تکامل سیستم تفاوت‌های اندکی دارد. به‌ویژه اگر در طراحی شیء گرا، خوب عمل کرده باشیم، اضافه کردن قابلیت‌های جدید (new functionality) و یا تغییر بعضی از رفتارهای موجود بطور عادی انجام می‌شود. و بخصوص این راحتی را در مورد سیستم‌های بزرگ نیز دریافته‌ام.

بعد از پیاده‌سازی.

قانون تداوم تغییر.

قانون ازدیاد پیچیدگی.

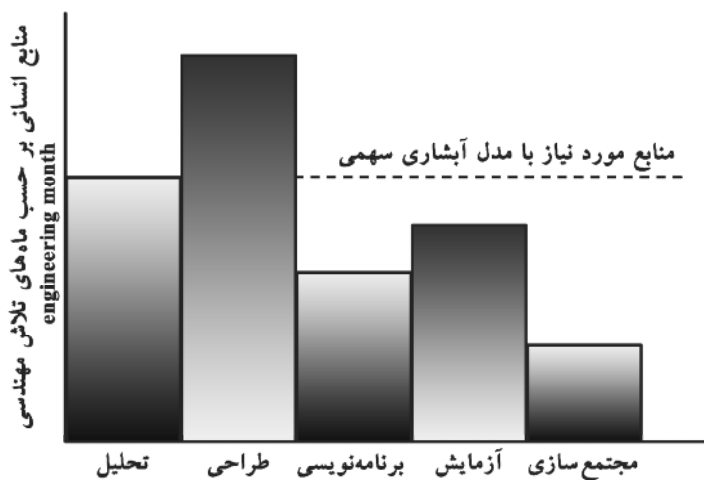
تفاوت اندک مراحل تغییر و تکامل.

۲-۷ مدیریت پروژه (Managing a Project)

تخصیص منابع (resource allocation): ما دریافته‌ایم که به کارگیری طراحی شیء گرا موجب کوتاه شدن زمان ایجاد و توسعه‌ی development سیستم، موجب ارتقاء کیفی نرم‌افزار، یعنی نزدیکی بیشتر آن با نیازهای واقعی می‌گردد. یکی از جنبه‌های اعجاب‌آور مدیریت پروژه‌ها با استفاده از طراحی شیء گرا اینست که کاهش عمومی منابع مورد نیاز و تغییر در زمان کنار رفتن نیروها محسوس است. شکل ۲-۷ بیانگر این پدیده است.

زمان کمتر.

منابع کمتر.



شکل ۲-۷: منابع انسانی بر حسب ماه‌های تلاش مهندسی

با طراحی شیء گرا، منابع باید به گونه‌ی دیگری تخصیص یابد. منابع در بخش تحلیل تفاوت چندانی در مقایسه با مدل سنتی آبشاری ندارند. اما برای طراحی، منابع بیشتری لازم است چرا که کار بیشتری انجام می‌گیرد. این ازدیاد به معنی بیشتر

کردن افراد نیست؛ معمولاً به معنی کار بیشتر چند نیروی خوب برای مدت طولانی‌تر است. برای برنامه‌نویسی نیروی کمتری لازم است به‌خاطر این که کار باقیمانده محدود است. آزمایش هم منابع کمتری احتیاج دارد. بیشتر به‌دلیل این که اضافه کردن توانایی جدید یک کلاس، عمدتاً با تغییر کلاسی که رفتار صمیمی دارد انجام می‌شود. بلاخره منابع مورد نیاز در قسمت مجتمع‌سازی خیلی کمتر است چراکه مجتمع‌سازی به تدریج و به‌صورت افزایش یابنده در طول چرخه‌ی عمر ایجاد و توسعه انجام می‌شود. در مجموع منابع مورد نیاز برای روش شیء‌گرا معمولاً کمتر یا مساوی منابع مورد نیاز با دیدگاه‌های سنتی، و کیفیت محصول خیلی بهتر است.

۳-۷ مهارت‌های تصمیم‌های ایجاد و توسعه (Development Team Skills)

با دیدگاه شیء‌گرا در نقش تصمیم‌های تولیدکننده‌ی نرم‌افزار نیز تغییراتی می‌بینیم. چهار نوع تولیدکننده مورد نیاز است، به ترتیب ارشدیت:

- معماران سیستم (system architects)؛
 - طراحان کلاس (class designer)؛
 - پیاده‌کنندگان کلاس (class Implementers)؛
 - برنامه‌نویسان کاربردی (application programmers).
- دید شیء‌گرا امکان استفاده از تیم‌های کوچکتر را میسر می‌سازد. ایجاد بیش از یک میلیون خط برنامه‌ی کیفی توسط تیم‌های کوچک. ۳۰ تا ۴۰ نفر در طول یک‌سال غیر عادی نیست. با توجه به استفاده از ابزار در دیدگاه شیء‌گرا؛ وجود ابزارمند tool smith و کتابدار (librarian) نیز در تیم‌ها کمک‌کننده / ضروری است.

حمایت ابزار (Tool Support)

- ما حداقل شش نوع ابزار مختلف که در طراحی شیء‌گرا مفید هستند را شناسایی و تعیین کرده‌ایم:
- سیستم متکی بر گرافیک برای نشان دادن علامت‌گذاری‌های تعریف شده (فصل ۵) در طراحی شیء‌گرا. این ابزار در تمام چرخه‌ی عمر روند ایجاد و توسعه‌ی نرم‌افزار با دید شیء‌گرا می‌تواند کمک‌کننده باشد؛
 - مرورکننده (browser) که ساختار کلاس و معماری سیستم را می‌داند و در مواردی مثل سلسله‌مراتب و تعریف کلاس‌ها، پیدا کردن یک کلاس می‌تواند کمک کند. مرورکننده‌ها ابزار مهمی در طراحی شیء‌گرا هستند؛
 - کامپایلر افزایش یابنده (incremented compiler)؛
 - اشکال‌زدا (debuggers)؛
 - ابزار مدیریت پیکربندی و کنترل گونه‌ها (configuration management & version control tools) بخصوص برای پروژه‌های بزرگ؛
 - کتابدار کلاس‌ها (class librarian).
- با توجه به ابزار فوق‌الذکر، یک ابزارمند (tool smith) و کتابدار (librarian) در تیم‌های تولید و ایجاد نرم‌افزار با دید شیء‌گرا مفید / ضروری خواهد بود.

۴-۷ مزایا و خطرات طراحی شیء‌گرا

(The Benefits & Risks of Object-Oriented Design)

مزایا:

- برخورد با مشکل ایجاد و توسعه‌ی سیستم‌های بزرگ؛
- ساخت سیستم‌های پیچیده خوش‌ساخت (well structure) (اصولاً مزایای مدل شیء‌گرا)؛
- استفاده از امکانات زبان‌های برنامه‌سازی شیء‌گرا یا مبتنی بر شیء؛
- تشویق استفاده‌ی مجدد از نرم‌افزار (reuse)؛

- محصول منعطف تر در مقابل تغییرات؛
- کاهش خطرات تولید؛
- استفاده از توان ادراکی انسان؛
- امکان تقلیل زمان تولید و اندازه‌ی کد (بر اساس تجربه‌ها).

خطرات

دو زمینه‌ی خطر باید مورد توجه قرار گیرد:

۱. کارایی (performance)؛
۲. هزینه‌های شروع (start-up cost).

خطرات کارایی:

- ارسال پیام در مقابل فراخوانی عادی کندتر است؛
 - به‌خاطر تجربه‌ها؛
 - به‌خاطر بار کلاس‌ها (یک کلاس ممکن است زیر کلاس‌های بسیار زیادی داشته باشد)؛
 - به‌خاطر رفتار صفحه‌بندی (paging behavior) کاربردهای در حال اجرا؛
 - به‌خاطر تخصیص پویا و تخریب شیء‌ها.
- بعضی موارد مورد تردید است. مثل این که نشان داده شده برنامه‌های ++C از برنامه معادل C سریع تر است [31].

خطرات هزینه‌های شروع:

- نیاز به ابزار؛
- عدم وجود نرم‌افزارهای قبلی که بتوانند مورد استفاده‌ی مجدد قرار گیرند (در شروع)؛
- نیاز به آموزش نیروها (با یادگیری زبان شیء گرا مسأله حل نمی‌شود. بلکه تغییر در تفکر لازم است).

۷-۵ انتقال به طراحی شیء‌گرا (The Transition to Object-Oriented Design)

تغییر تفکر لازم است. اقدامات زیر را برای رسیدن به این تفکر پیشنهاد می‌کنیم:

- برگزاری آموزش‌های رسمی هم برای تولیدکنندگان و هم برای مدیران روی عناصر مدل شیء؛
- استفاده از طراحی شیء‌گرا ابتدا در پروژه‌هایی که ریسک خطرات کمی دارند؛ اجازه دادن، به موارد خطای اعضای تیم؛ استفاده از این اعضای تیم‌ها در پروژه‌های دیگر برای هدایت پروژه با دید شیء‌گرا؛
- قرار دادن تولیدکنندگان و مدیران در معرض سیستم‌های شیء‌گرای خوش ساخت (well structured).

۷-۶ جایگاه موجودیت‌های انسانی از محیط عملیاتی در سیستم کامپیوتری مورد نظر

استفاده از مدل شیء به‌عنوان یک مدل مهندسی نرم‌افزار (برای ایجاد، توسعه و نگهداری نرم‌افزار) همواره با این حقیقت روبرو است که (بخصوص در مراحل تحلیل و طراحی) عوامل انسانی متعددی در کل سیستم نقش دارند. معمولاً نیروهای انسانی که هر یک، یک شیء از محیط عملیاتی محسوب می‌شوند را می‌توان در کلاس‌هایی طبقه‌بندی کرد.

در مثال کتابخانه. کلاس‌های مراجعین کتاب‌داران و پردازش‌کنندگان کتاب، مطرح می‌شوند.

- سؤال این جاست که این شیء‌ها و کلاس‌ها چه جایگاهی در تحلیل، طراحی و بخصوص در پیاده‌سازی سیستم دارند؟
- در تحلیل قطعاً نقش دارند. (غیر از کمکی که به ما برای فهم زمینه‌ی عملیاتی می‌کنند) آنها مهمترین پردازشگران محیط اجرایی هستند. یک سیستم کامپیوتری قرار است زیر مجموعه‌ای از فعالیت‌های آنها را انجام دهد. در حالی که جایگاه تشکیلاتی آن شخص در کل محیط عملیاتی معمولاً همچنان پابرجاست، و احتمالاً با تغییرات کمی روبروست.

پس در تحلیل‌ها قطعاً باید مورد توجه باشند؛

از نظر وظایف و مسئولیت‌ها؛

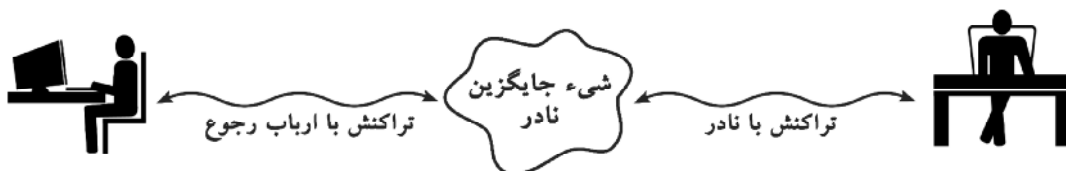
- از نظر روش انجام وظایف؛
- از نظر واحدها یا شیء‌های دیگر که با آنها در تراکنش هستند.
- در طراحی نیز انسان‌ها و کلاس‌هایی که تشکیل می‌دهند، جایگاه ویژه‌ای دارند:
از نظر منظور کردن شیء‌هایی که بتوانند وظایف مورد نظر از نیروی انسانی را به عهده بگیرند؛
از نظر ارتباط با شیء‌ها و کلاس‌هایی از برنامه که جایگاه عملکرد نیروهای انسانی دیگر را در کل سیستم می‌گیرند؛
از نظر ایجاد ابزار (نرم‌افزار) لازم برای تراکنش با انسان.
چه انسانی که کامپیوتر زیرمجموعه‌ای از وظایف او را به عهده گرفته است.
چه انسان (هایی) که خود، مراجعه‌کننده‌ی انسان وظیفه‌مند اول بوده‌اند.



شکل ۳-۷: یک انسان در محیط عملیاتی به عنوان client



شکل ۴-۷: یک انسان در محیط عملیاتی به عنوان server



- شکل ۵-۷: جایگاه موجودیت‌های انسانی از محیط عملیاتی در سیستم کامپیوتری
- به‌جای هر انسانی که نقش اجرایی دارد، مؤلفه‌ای از سیستم باید وظایف مربوطه را به عهده گیرد.
- برای هر انسانی که نقش مراجعه‌کننده یا مشتری را دارد، مؤلفه‌ای (هایی) از سیستم باید واسط لازم با او را به وجود آورد. به عبارت دیگر واسط‌هایی بین کاربر و واحد(های) مسؤل انجام وظایف باشند.

۷-۷ خلاصه‌ی فصل هفتم از کتاب طراحی شیء‌گرا / نکات علمی در طراحی شیء‌گرا

(Pragmatism)

- چرخه‌ی زندگی ایجاد و توسعه‌ی نرم‌افزار با استفاده از طراحی شیء‌گرا تکیه بر ایجاد سیستم به صورت اضافه شونده و تکرار شونده دارد؛
- تحلیل ساخت‌یافته، آغاز جالبی برای طراحی شیء‌گراست؛ تحلیل شیء‌گرا مقدم (front end) مؤثرتری است؛
- هرگاه که یک مدل رسمی یا غیر رسمی (احتمالاً ناقص) از مسأله وجود داشته باشد، طراحی می‌تواند شروع شود؛ و زمانی طراحی متوقف می‌شود که ترکیب، به‌جای تفکیک حاکم باشد؛
- در طراحی شیء‌گرا هرگز یک حادثه‌ی بزرگ (big-bang) یک‌باره برای مجتمع‌سازی (integration) سیستم وجود ندارد؛

- در طی تکامل سیستمی که با استفاده از طراحی شیء گرا به دست آمده است، چندین نوع از تغییرات روی طرح قابل پیش‌بینی است؛ هر یک از این تغییرات متضمن هزینه‌های متفاوتی هستند؛
- استفاده از طراحی شیء گرا روی مدیریت نیرو، معیارهای ارزیابی (milestones) و محصول، مدیریت گونه‌های مختلف محصول (release management)، اطمینان از کیفیت و ابزار حمایتی تأثیر می‌گذارد؛
- به کارگیری طراحی شیء گرا دارای مزایای بسیار زیاد و در عین حال با ریسک‌هایی همراه است؛ تجربه نشان داده است که این مزایا بسیار فراتر از ریسک‌ها می‌باشند؛
- در یک سازمان، تغییر وضع به استفاده از مدل شیء گرا، محتاج تغییر در اندیشه است؛ یادگیری یک زبان شیء گرا یا مبتنی بر شیء کافی نیست.