فصل ۱۲: مدلسازی تحلیل نیازها

در سطح تکنیکی، مهندسی نرم‌افزار با یک سری از کارهای مدلسازی شروع می‌شود که به مشخصهٔ نیازهای کامل و نمایش طراحی جامعی برای نرم‌افزار درحال ایجاد تبدیل می‌گردند. مدل تحلیل، که در واقع مجموعه‌ای از مدل‌ها است، اولین نمایش تکنیکی از سیستم می‌باشد. در طول سال ها، روش های بسیاری برای مدلسازی تحلیل پیشنهاد شده‌اند. به هر حال، دو روش از آنها در حال حاضر وجود دارند. اولی، **تحلیل ساختیافته**، روش مدلسازی کلاسیک می‌باشد و در این فصل توصیف می‌شود. روش دیگر، **تحلیل شی گرا**، با جزییات در فصول آتی بحث می‌شود.

تحلیل ساختیافته فعالیت ساخت مدل می‌باشد. با به کارگیری اصول تحلیل عملیاتی بحث شده در همین فصل مدل های داده ای، عملکرد، و رفتار، ایجاد شده و مشخص کنندهٔ **آنچه باید** ایجاد شود می‌باشند.

تاریخچه

کارهای اولیه در مدلسازی تحلیل در دههٔ ۱۹۶۰ و اوایل دههٔ ۱۹۷۰ شروع شد، اما اولین ظهور روش تحلیل ساختیافته مرتبط با عنوان مهم دیگری بود: " **طراحی ساختیافته**". محققین نیاز به یک نشان‌گذاری گرافیکی برای نمایش داده‌ها، و فرآیندهایی داشتند که آن داده‌ها را تبدیل می‌نمودند. این فرآیندها به طور تقریبی بر معماری طراحی منطبق می‌شوند.

واژهٔ تحلیل ساختیافته، که ابتدا توسط Douglas Ross استفاده شد، توسط DeMarco مشهور گردید و بعدها توسعه یافت. این توسعه‌ها باعث انجام روش تحلیل قوی‌تری گردیدند که می توانست به طور مؤثری در مسایل مهندسی به کار گرفته شود. تلاش هایی برای توسعهٔ یک نشان‌گذاری یکپارچه پیشنهاد گردید، و روش‌های مدرنی منتشر شدند تا استفاده از ابزارهای CASE را معرفی نمایند.

در این مرحله تحلیلگر اقدام به مدلسازی نیازهای سیستم مورد مطالعه می نماید. روش‌های مدلسازی سیستم ها متنوع و گسترده هستند.

در مجموع ۲ روش متداول برای مدلسازی سیستمها وجود دارد :

- **ساختیافته (Structured)**
- **شئ گرا (Object Oriented)**

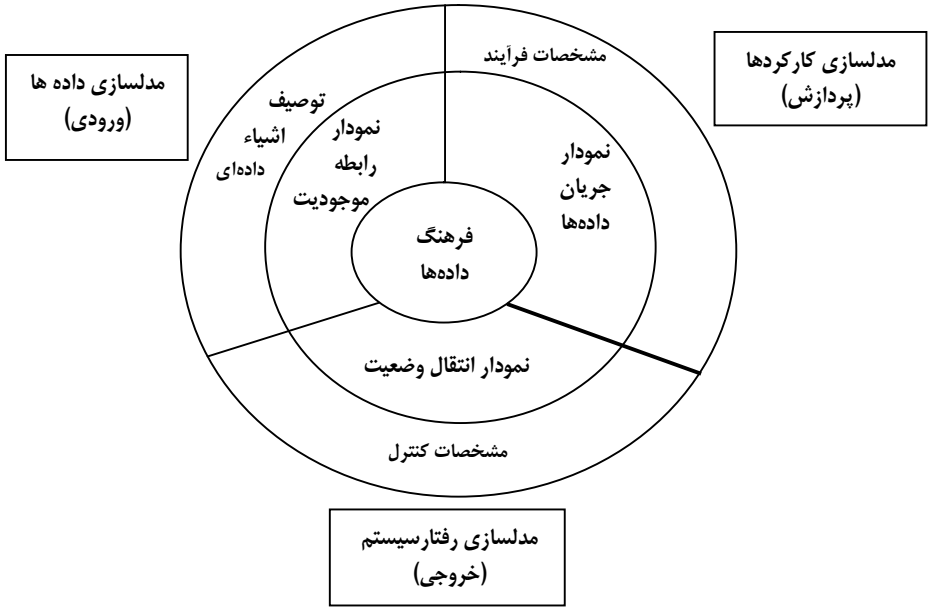
از متداولترین روش های ساختیافته(SSADM-Structured System Analysis & Design Method) است.

نگرش سازماندهی شده و منظم سیستمی از بالا به پایین(Top Down) برای تحلیل و طراحی و نگرش پایین به بالا(Bottom Up) برای پیاده سازی نرم افزار. در این روش:

- **مستندات قوی باعث بالا بردن قابلیت نگهداری می‌گردد.**
- **اندازه مساله با افراز نمودن مناسب تعیین می‌گردد.**
- **از نماد گرافیکی برای نمایش روابط سیستم استفاده می‌گردد.**
- **تمایز بین نمایش فیزیکی و منطقی وجود دارد.**
- **و...**

اهداف مدل تحلیل نیاز

- نیازهای مشتری را تشریح می کند.
- پایه و مبنایی برای مدل طراحی نرم افزار بوجود می آورد.
- مجموعه ایی از نیازها را تعریف می کند که می توان درستی و اعتبار نرم افزار را آزمایش نمود.



شکل ۴: مدل تحلیل ساختیافته

بنابراین اجزای مدل تحلیل در روش ساختیافته عبارتند از :

- **ERD : مدلسازی داده‌ها است.**
- **DFD : مدلسازی کارکردها است.**
- **STD : مدلسازی رفتار سیستم در مقابل وصعیت‌ها و رویدادهای بیرونی است.**

۱- مدلسازی داده ها

مدل داده شامل سه قطعهٔ اطلاعاتی توصیف شده می‌باشد: **شیء داده**، **صفاتی** که توصیف کنندهٔ شیء داده می‌باشند، و **روابطی** که اشیاء داده را به یکدیگر مرتبط می‌کنند.

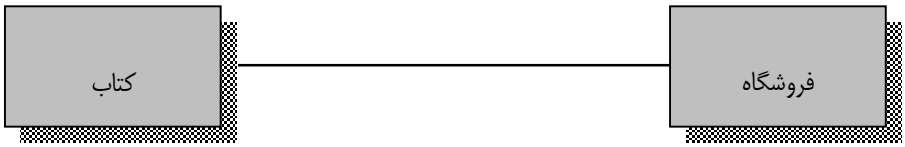
اشیاء داده: شیء داده نمایشی از تقریباً هر اطلاعات مرکبی می‌باشد که باید در توسعه نرم‌افزار تشخیص داده شود. منظور از اطلاعات مرکب، چیزی است که دارای خصوصیت‌ها یا صفات متعددی می‌باشد. بنابراین، طول (یک مقدار واحد)، یک شیء دادهٔ معتبر نمی‌باشد، ولی ابعاد (شامل ارتفاع، طول، و عمق) می‌تواند به عنوان یک شیء تعریف شود. **یک شیء داده** می‌تواند هر یک از این موارد باشد: **موجودیت خارجی**(برای مثال، هر چیزی که اطلاعات را تولید یا مصرف نماید)، هر چیزی(برای مثال، یک گزارش یا یک نمایش)، یک رخداد(برای مثال، مکالمهٔ تلفنی)، یا واقعه(برای مثال، آلارم)، یک نقش(برای مثال، فروشنده)، یک واحد سازمانی(برای مثال، واحد حسابداری)، یک مکان(برای مثال، خانه)، یا یک ساختار(برای مثال، فایل).

صفات: صفات، خصوصیات شیء داده را تعریف می‌کنند و می‌توانند یکی از سه نوع خصوصیت متفاوت باشند. می‌توانند به این منظورها استفاده شوند: ۱) نام‌گذاری نمونه‌ای از شیء داده، ۲) توصیف یک نمونه، یا ۳) ارجاعی به نمونهٔ دیگری در جدول دیگر. علاوه بر این ها، یک یا چند صفت باید به عنوان شناسه تعریف شوند، یعنی، صفت شناسه، به عنوان کلیدی برای یافتن نمونه‌ای از یک شیء داده استفاده می‌گردد.

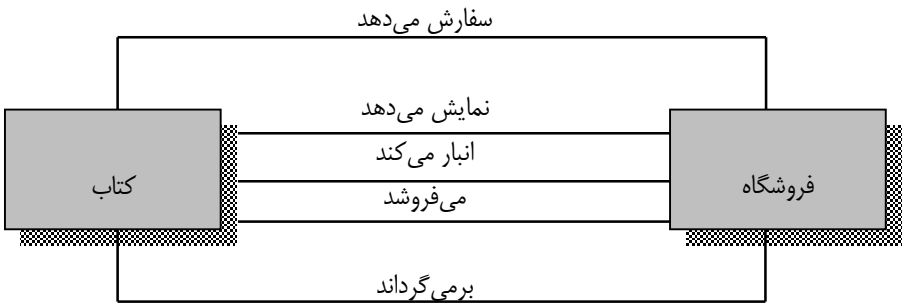
روابط: اشیاء داده به یکدیگر با روش های متفاوتی مرتبط می‌باشند. دو شیء دادهٔ کتاب و کتاب‌فروشی را در نظر بگیرید. این اشیاء می‌توانند با استفاده از نشان‌گذاری ساده‌ای مانند شکل ۴ نشان داده شوند. ارتباطی بین

کتاب و کتاب‌فروشی برقرار می‌شود زیرا این دو شیء با یکدیگر مرتبط می‌باشند. اما این روابط چه هستند؟ به منظور مشخص نمودن این جواب، باید نقش کتاب ها و کتاب‌فروشی در زمینهٔ نرم‌افزاری که ایجاد می‌شود روشن شود. مجموعه‌ای از زوج‌های شیء– رابطه تعریف می‌شوند که روابط را تعریف می‌نمایند. برای مثال،

- کتاب‌فروشی کتاب ها را سفارش می‌دهد.
- کتاب‌فروشی کتاب ها را به نمایش می‌گذارد.
- کتاب‌فروشی کتاب ها را نگهداری می‌کند.
- کتاب‌فروشی کتاب ها را می‌فروشد.
- کتاب‌فروشی کتاب ها را بر می‌گرداند.



شکل ۵: ارتباط اساسی بین اشیاء



شکل ۶: روابط بین اشیاء

چندی و الزام (Cardinality and Modality)

چندی. مدل داده باید قادر به نمایش تعداد اشیاء در یک رابطه باشد. چندی، مشخصهٔ تعداد رخداد یک شیء می‌باشد که می‌تواند با تعداد رخدادی از شیء دیگر مرتبط گردد. چندی معمولاً به صورت " یک " یا " چند " بیان می‌شود. با در نظر گرفتن تمام ترکیبات " یک " و " چند "، دو شیء می‌توانند به این ترتیب مرتبط شوند:

- **یک به یک (۱:۱)**– یک رخداد شیء A می‌تواند به یک و فقط یک رخداد از شیء B مرتبط شود، و یک رخداد از B می‌تواند فقط به یک رخداد A مرتبط گردد.
- **یک به چند (1:N)**– یک رخداد از شیء A می‌تواند با یک یا چند رخداد از شیء B مرتبط شود، اما یک رخداد از B می‌تواند فقط می‌تواند با یک رخداد از A مرتبط گردد. برای مثال، یک مادر می‌تواند چندین فرزند داشته باشد، و یک فرزند می‌تواند یک مادر داشته باشد.
- **چند به چند (M:N)**– یک رخداد از شیء A می‌تواند به یک یا چند رخداد از B مرتبط شود، در حالی که یک رخداد از B می‌تواند با یک یا چند رخداد از A مرتبط گردد. برای مثال، عمو می‌تواند چندین برادرزاده داشته باشد، در حالی که هر برادرزاده می‌تواند چندین عمو داشته باشد.

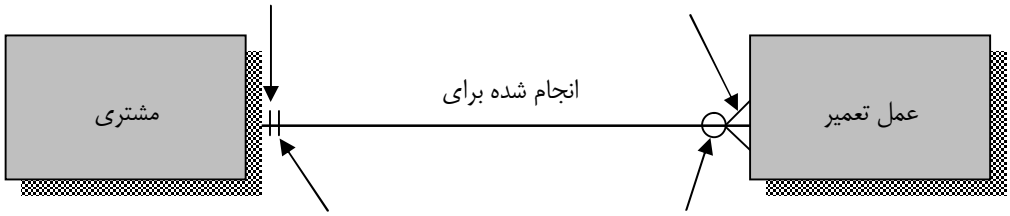
درجه، حداکثر تعداد اشیایی را که می‌توانند در یک رابطه شرکت کنند تعریف می‌نماید. به هر حال، نشان دهندهٔ این مطلب نیست که شیء دادهٔ خاص باید در یک رابطه شرکت نماید یا شرکت ننماید. به منظور مشخص نمودن این اطلاعات، مدل داده، **الزام** را به زوج شیء- رابطه می‌افزاید.

الزام: برای یک رابطه "صفر" است اگر نیاز صریحی برای وجود رابطه نباشد، یا رابطه اختیاری است. برای رابطه ای "یک" است اگر رخدادی از یک رابطه ضروری باشد. به منظور نشان دادن این مطلب، نرم‌افزاری را در نظر بگیرید که توسط شرکت تلفن محلی برای پردازش درخواست‌های سرویس استفاده شده است. یک مشتری نشان می‌دهد که مشکلی وجود دارد. اگر آن مشکل به سادگی تشخیص داده شود، یک عمل تعمیر انجام می گیرد. به هر حال، اگر مشکل پیچیده باشد، چندین تعمیر ممکن است لازم باشد.

اگر الزام داشته باشد با یک و در غیر اینصورت صفر قرار می دهیم.

چندی: دلالت بر انجام
چندین عمل تعمیر ممکن دارد.

چندی: دلالت بر انجام
اعمال تعمیر دارد.



الزام: دلالت بر وضعیتی اختیاری دارد که ممکن است در آن عمل تعمیر لازم نباشد.

الزام: دلالت بر وضعیت قطعی دارد و برای انجام اعمال تعمیر باید یک مشتری وجود داشته باشد.

شکل ۷: نمایش چندی و الزام روابط بین اشیاء

نتیجه این کارها ما را به **نمودار ارتباط موجودیت (Entity Relationship Diagram-ERD)** می‌رساند. نمودار ER به سوالات زیر پاسخ می دهد :

- اشیا، داده ایی را بشناسیم.
- اولویت، علت، صفات خاصه، برای هر **Object** را بشناسیم.
- ارتباط اشیا، با پردازش ها چگونه است؟
- نحوه انتقال داده ها و پردازش‌ها به چه گونه است ؟

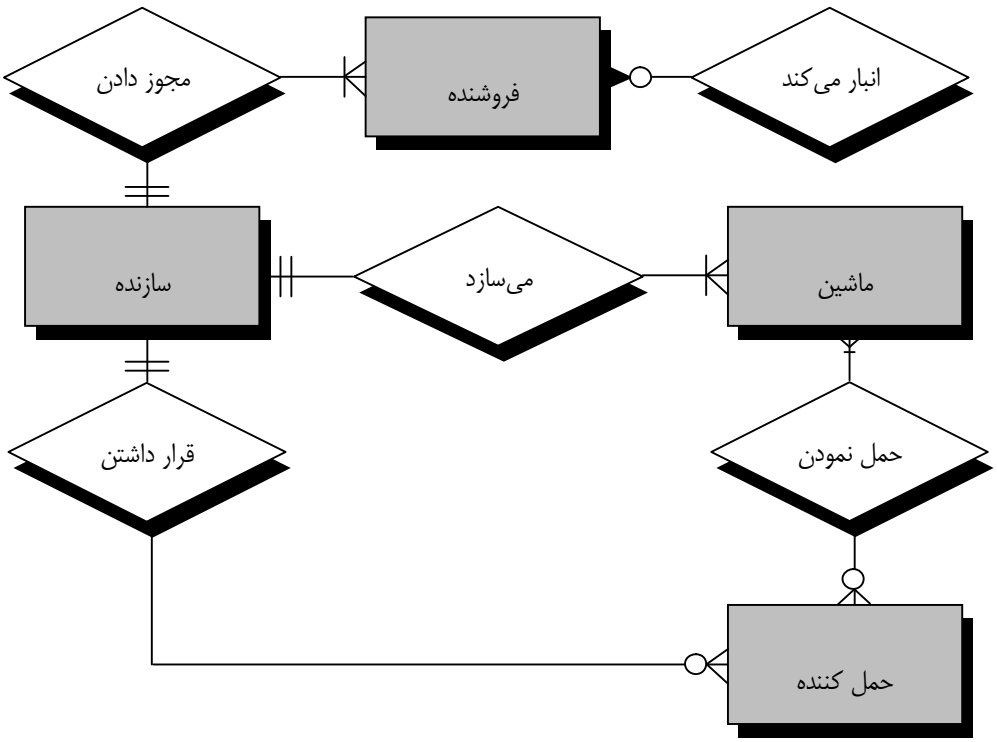
بنابراین

مولفه های پایه‌ای مدلسازی داده ها

Data Object
Attributes
Relationship
Indicators

ERD

نمونه‌ای از نمودار ER در شکل زیر نشان داده شده است.



شکل ۸: نمودار ارتباط موجودیت

توجه : حداقل نتیجه حاصله، شکل جداول پایگاه داده‌ایی می باشد .

۲- مدلسازی کارکردها و جریان اطلاعاتی(Functional Modeling)

اطلاعات در ضمن جریان در سیستم کامپیوتری تبدیل می‌شود. سیستم، ورودی را به شکل‌های مختلف می‌پذیرد، سخت‌افزار، نرم‌افزار، و عناصر انسانی را برای تبدیل آن به کار می‌گیرد، و خروجی را به شکل‌های مختلف تولید می‌نماید. ورودی می‌تواند مثلاً یک سری اعداد تایپ شده باشد. تبدیلات می‌توانند شامل یک مقایسهٔ سادهٔ منطقی، یک الگوریتم عددی پیچیده، یا یک روش استنتاج در سیستم های خبره باشند. خروجی می تواند یک چراغ را روشن کند، یا ۲۰۰ صفحه گزارش تولید نماید. در نتیجه، می‌توان مدل جریان را برای هر سیستم کامپیوتری، علیرغم اندازه و پیچیدگی آن ایجاد نمود.

تمرکز برحرکت و پردازش اطلاعات است که تحت عنوان Information Flow مطرح است.

I ⟶ P ⟶ O

نمودار جریان داده‌ها

اطلاعات، با حرکت در نرم‌افزار، توسط یک سری عملیات اصلاح می‌شود. **نمودار جریان داده**(DFD) نمایشی گرافیکی است که جریان اطلاعات تبدیلاتی را که در ضمن حرکت داده‌ها از ورودی به خروجی انجام می‌شوند نشان می‌دهد. شکل اصلی **نمودار جریان داده، گراف جریان داده، یا چارت حبابی** نامیده می‌شود.

نمودار جریان داده می‌تواند برای نمایش یک سیستم یا نرم‌افزار در هر سطحی از **انتزاع** استفاده گردد. در واقع، DFD ها می‌توانند به سطوحی تقسیم‌بندی شوند که جریان رو به افزایش اطلاعات و جزییات عملکردی را نشان دهند. بنابراین، DFD مکانیزمی را برای مدلسازی تابعی همانند مدلسازی جریان اطلاعات فراهم می‌نماید.

DFD سطح صفر که مدل اساسی سیستم یا مدل زمینه نیز نامیده می‌شود، کل عنصر نرم‌افزار را به صورت یک حباب، همراه با داده‌های ورودی و خروجی نشان داده شده توسط پیکان های وارد شده و خارج شدن نشان می‌دهد. فرآیندهای (حباب های) اضافی و مسیر جریان اطلاعات در ضمن تجزیه DFD سطح صفر برای نشان دادن جزییات بیشتر، نمایش داده می‌شوند. برای مثال، DFD سطح ۱ می‌تواند حاوی پنج یا شش حباب به همراه پیکان های متصل کنندهٔ آن ها باشد. هر یک از این فرآیندهای نشان داده شده در سطح ۱، زیر تابعی از کل سیستم بیان شده در مدل زمینه می‌باشد و الی آخر.

نشان‌گذاری استفاده شده برای توسعهٔ DFD، به تنهایی برای توصیف نیازهای نرم‌افزاری مناسب نمی‌باشد. به کارگیری مؤلفهٔ دیگری از نشان‌گذاری برای تحلیل ساختیافته به نام فرهنگ داده‌ها مکمل DFD است.

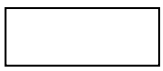
نشان‌گذاری گرافیکی DFD باید با متن توصیفی همراه شود. مشخصهٔ فرآیند (PSPEC) استفاده می‌شود تا جزییات پردازش مشخص شده توسط یک حباب را در DFD مشخص نماید. این مشخصهٔ فرآیند، توصیف کنندهٔ ورودی یک تابع، و مشخص کنندهٔ الگوریتمی است که برای تبدیل ورودی و تولید خروجی به کار گرفته می‌شود، علاوه بر آن، PSPEC محدودیت‌هایی را نشان می‌دهد که بر فرآیند (تابع)، و خصوصیات کارایی مربوط به آن فرآیند، تحمیل شده‌اند. همچنین محدودیت‌های طراحی که بر روش پیاده‌سازی فرآیند تأثیر می‌گذارند توسط PSPEC تعیین می‌گردند.

نکات مربوط به DFD

- **توالی، تاخر و تقدم اطلاعات مشخص نیست و نحوه تبدیل نیز مشخص نمی‌باشد.**
- **هیچ عمل فیزیکی را انجام نداده و فقط گردش اطلاعات را نشان می دهد.**
- **کنترل ها دیده نمی شود.**

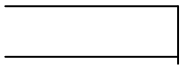
به همین جهت بعد از شکستن سیستم تا سطح جزییات به توصیف پردازش می پردازیم. بعداز رسم DFD باید به شرح پردازش ها (PSPEC) بپردازیم که مشکلات DFD را حل نماید.

نمادهای روش ساختیافته (yordoun)


موجودیت خارجی External Entity

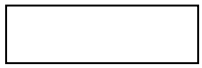

سیستم، فعالیت، پردازش Process
سطح صفر، سطح یک، پایین ترین سطح


اطلاعات،شیء داده‌ایی Data Object

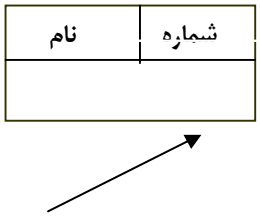

ذخیره داده Data Store

نمادهای روش SSADM

External Entity

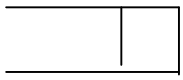


شماره : سطح صفر، یک، دو، سه و الی آخر
نام: سیستم، زیر سیستم، فعالیت، پردازش



جریان اطلاعات Information

ذخیره اطلاعات Data Storde

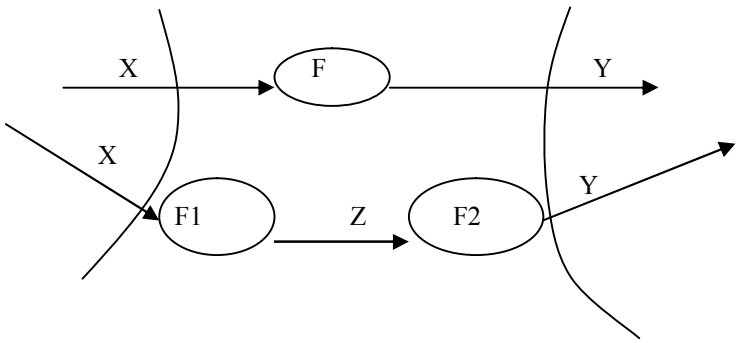


(C-Computer) کامپیوتری

(M-Manual) دستی

توجه: مجموعه‌ایی از پردازش ها، فعالیت و مجموعه‌ایی از فعالیت ها، زیر سیستم و مجموعه ایی از زیر سیستم ها، سیستم نامیده می‌شود .

مهمترین کار در DFD شکستن به اجزا، کوچکتر از خودش است (به طور منطقی و سلسله مراتبی).



شکل ۹ : نحوه شکست نمودار جریان داده

توسعه‌هایی برای سیستم‌های بلادرنگ

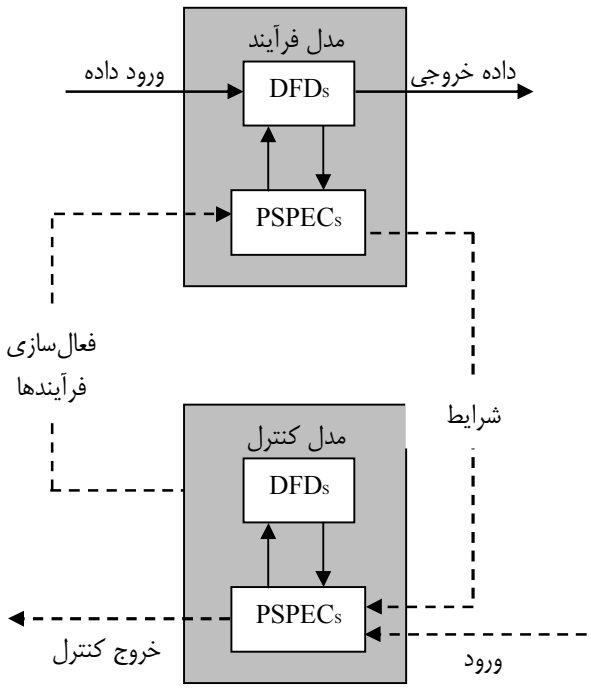
بسیاری از کاربردهای نرم‌افزاری، وابسته به زمان هستند و اطلاعات کنترلی را مانند داده‌ها پردازش می‌کنند. یک سیستم بلادرنگ باید با دنیای واقعی در یک دوره زمانی کوتاه مشخص شده توسط آن ارتباط برقرار نماید. ناوبری هواپیما، کنترل فرآیند تولید، محصولات مشتری، و تجهیزات صنعتی چند مورد از صدها کاربرد نرم‌افزاری بلادرنگ می‌باشند. به منظور سازماندهی تحلیل نرم‌افزار بلادرنگ، توسعه‌هایی از نشان‌گذاری، برای تحلیل ساختیافته تعریف شده است. یکی از این توسعه‌ها، که توسط Hatley و Pirbhai انجام شده‌اند، در این روش تحلیل‌گر جریان کنترل و پردازش کنترل را مانند جریان داده و پردازش داده، نمایش می دهد.

توسعه‌های Hatley و Pribhai

توسعه‌های Hatley و Pirbhai برای نشان‌گذاری پایه‌ای تحلیل ساختیافته تمرکز کمتری بر ایجاد نمادهای گرافیکی اضافی دارد و بیشتر بر نمایش و مشخصهٔ جنبه‌های کنترلی نرم‌افزار توجه دارد. پیکان خط‌چین برای

نمایش جریان کنترل یا واقعه استفاده می‌شود. بنابراین، نمودار جریان کنترل تعریف می‌شود. CFD(Control Flow Diagram) شامل همان فرآیندهای DFD است، اما، به جای نشان دادن جریان داده، جریان کنترل را نشان می‌دهد. به جای نشان دادن فرآیندهای کنترل به طور مستقیم در این مدل جریان، یک نشان‌گذاری ارجاعی (خط توپر) به یک مشخصهٔ کنترل (CSPES) می‌باشد که فرآیندهایی (توابع) را کنترل می‌نماید که در DFD بر مبنای واقعهٔ عبورکننده از این پنجره نشان داده شده‌اند.

نمودارهای جریان داده برای نشان دادن داده‌ها و فرآیندهایی که آنها را دستکاری می‌کنند استفاده می‌شوند. نمودارهای جریان کنترل نشان می‌دهند که چگونه وقایع در میان فرآیندها جریان دارند و وقایع خارجی را که باعث فعال شدن فرآیندهای گوناگون می‌شوند مشخص می‌نمایند. رابطهٔ بین مدل های فرآیند و کنترل به صورت شماتیک در شکل نشان داده شده‌اند. مدل فرآیند به مدل کنترل از طریق شرایط داده‌ها مرتبط می‌شود. این مدل کنترل، به مدل فرآیند از طریق اطلاعات فعال‌سازی فرآیند موجود در CSPEC متصل می‌شود.



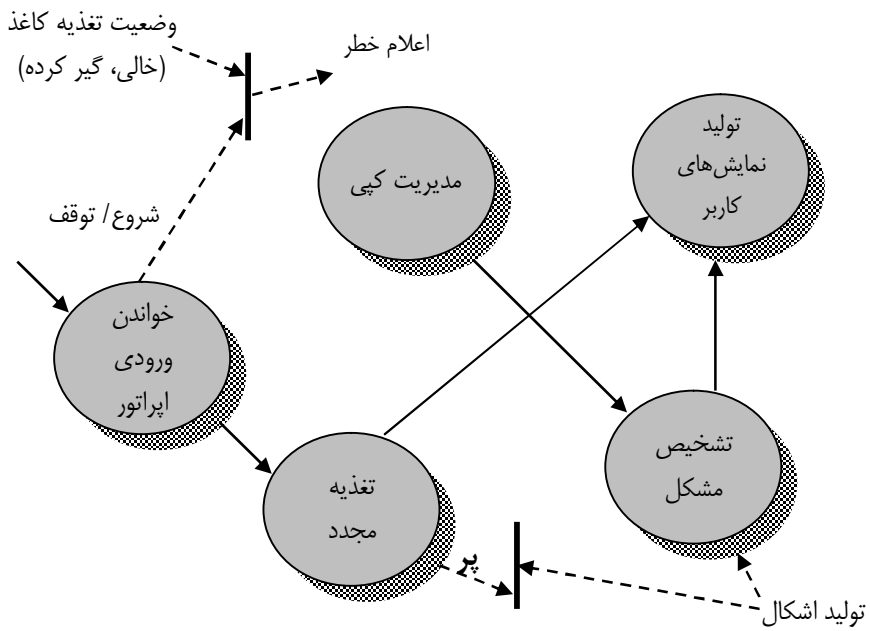
شکل ۱۰ : رابطه بین مدل های داده و کنترل

۳ – مدلسازی رفتارسیستم (نمودار انتقال حالت) (STD-State Transition Diagram)

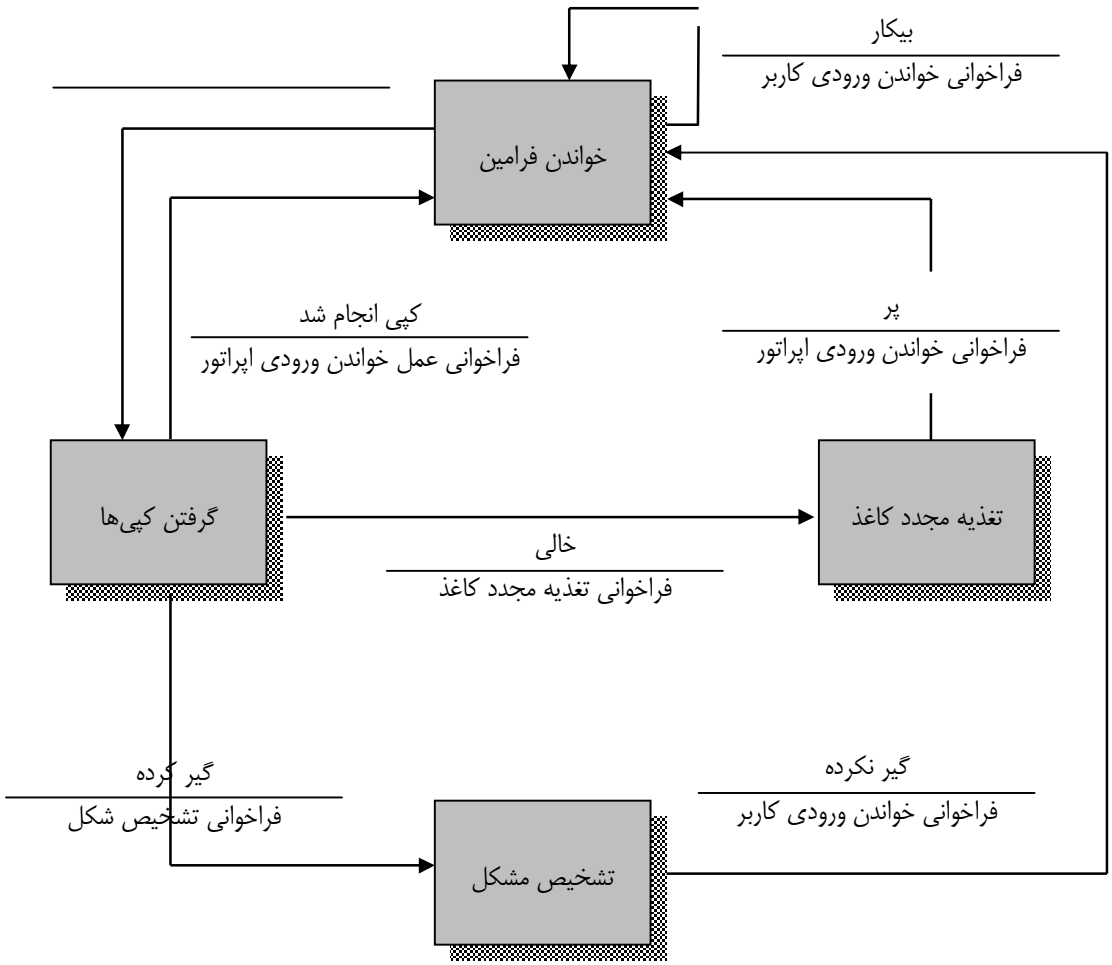
مدلسازی رفتاری مبنای عملیاتی تمام روش های تحلیل نیازها می‌باشد. نمودار انتقال و تغییر حالت، نشان دهندهٔ رفتار سیستم، با استفاده از مشخص نمودن حالت ها و وقایعی است، که باعث تغییر حالت می‌شوند. علاوه بر آن، STD نشان می‌دهد که چه عکس‌العمل‌هایی (برای مثال، فعالسازی فرآیند) در نتیجهٔ واقعهٔ خاصی باید انجام شود. در شکل‌های زیر نمودارهای CFD و STD مربوط به یک دستگاه کپی نشان داده شده است.

STD رفتار سیستم را در برخورد با رویداد های مختلف که منجر به تغییر وضعیت سیستم می‌شود به تصویر می‌کشد .

- **در STD: حرکت سیستم از یک وضعیت به وضعیت دیگر نشان داده می شود.**
- **در اینجا نقاط کنترل نیز مشخص می شود.**



شکل ۱۱ : نمودار جریان کنترل دستگاه کپی



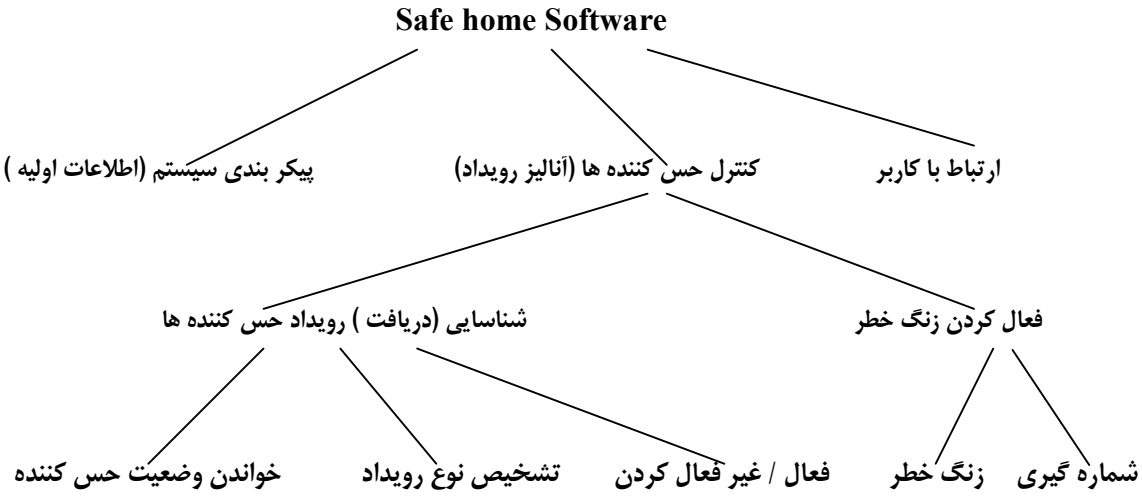
شکل ۱۲ : نمودار تغییر حالت برای نرم‌افزار دستگاه کپی

مراحل تحلیل و مدلسازی ساختیافته برای سیستم خانه امن (دزدگیر)

تعدادی سنسور پالس‌هایی را تولید می‌کنند و برنامه این پالس‌ها را دریافت و پس از تحلیل عکس‌العمل نشان می دهد.

کار اول: نسبت به اطلاعات، قلمرو، عملکرد و رفتار سیستم عمل تجزیه(افراز) را انجام می‌دهیم.

Pratitioning : افراز کردن اجزاء سیستم در سه جوزه اطلاعات ، رفتار، کارکرد صورت می‌گیرد.



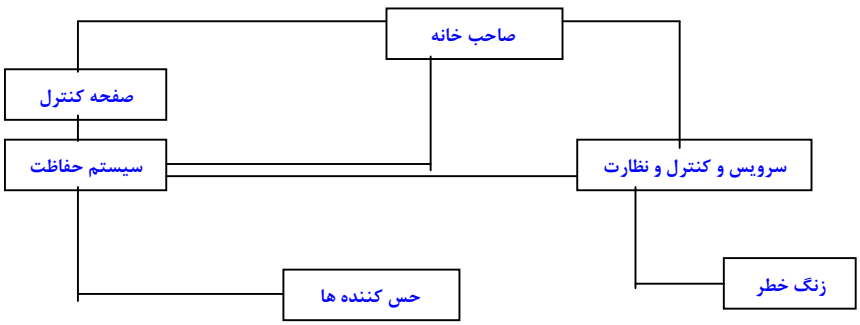
شکل ۱۳ : افراز مولفه‌های سیستم خانه امن

کار دوم: ایجاد نمودار ارتباط موجودیت ERD

با مشتری چیزهایی که در این سیستم وجود دارند به شرح زیر فهرست می‌کنیم:

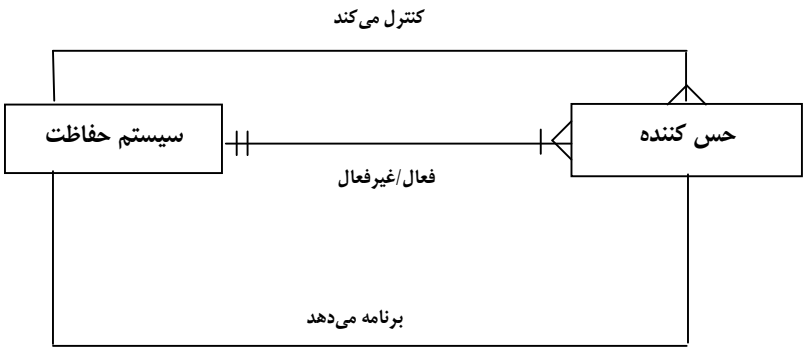
- **صاحبخانه**
- **حس کننده‌ها**
- **صفحه کنترل**
- **سیستم حفاظت**
- **کنترل و نظارت**

بعد ارتباط این اشیا را می کشیم(درختER)



شکل ۱۴ : رابطه بین مدل های داده و کنترل

کار سوم : توصیف صفات اشیا،



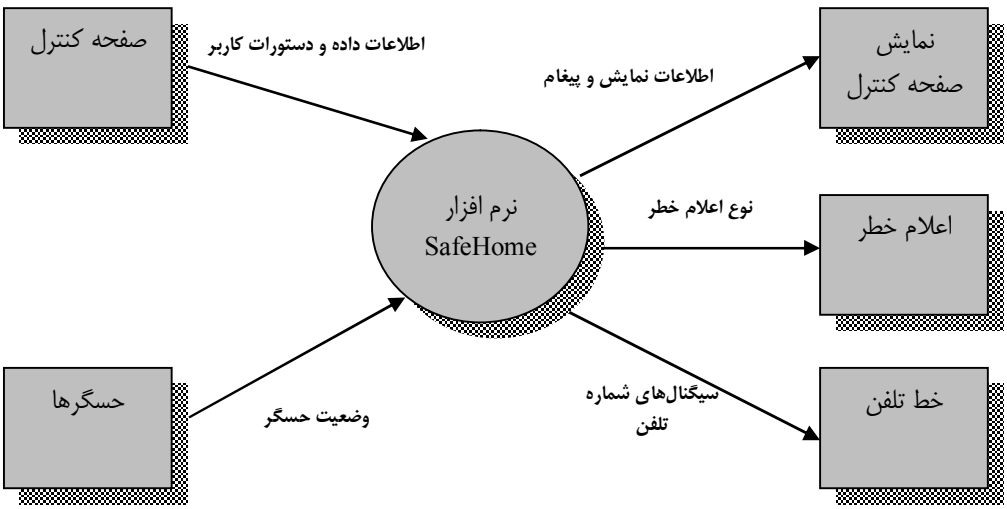
شکل ۱۵ : رابطه بین اشیا سیستم خانه امن

رهنمودی های رسم ERD

مهندس نرم افزار از نمودار ER ورودی و خروجی های سیستم را شناسایی می‌کند لذا:

۱. در حین جمع آوری نیازها با مشتری لیست اشیا، مورد نظر را بنویسید.
۲. یک شی را انتخاب و با مشتری ارتباط آنرا با سایر اشیا، تعیین کنید.
۳. هرزمان ارتباطی موجود بود مشترکا با مشتری این ارتباط را تعریف و ایجاد کنید.
۴. برای هر جفت ارتباط اشیا، چندی و الزامی بودن آنرا تعیین کنید.
۵. مراحل ۲ تا ۴ را مرتب تکرار کنید تا تمام ارتباطات تعریف شوند و در صورت لزوم اشیا، را به اجزا، کوچکتر خرد کنید.
۶. ویژگی هر موجودیت را تعیین کنید.
۷. ERD حاصل را مورد باز نگری قرار دهید.
۸. مراحل ۱ تا ۷ را تکرار تا ERD کامل شود.

کار چهارم: ایجاد مدل جریان داده و شکست آن تا سطح تفصیلی (DFD)



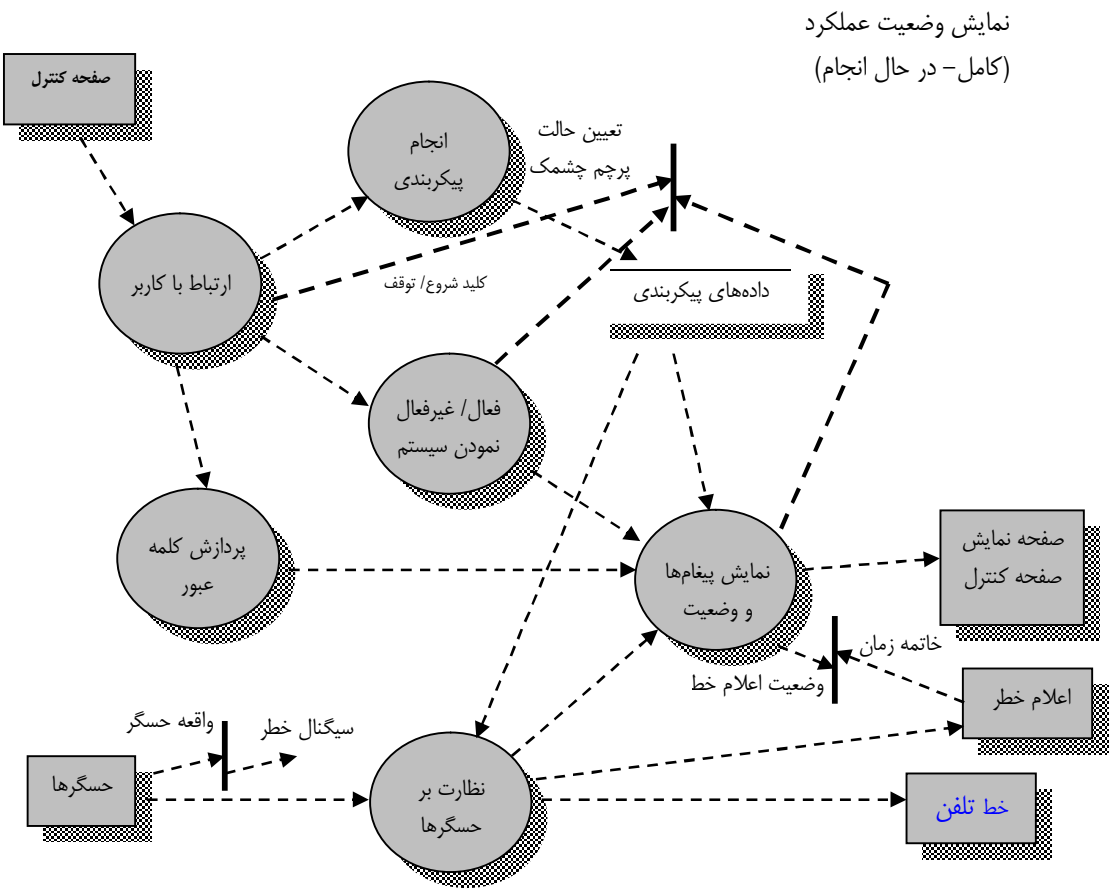
شکل ۱۶ : DFD سطح زمینه سیستم برای خانه امن

رهنمودهای رسم DFD

۱- DFD سطح صفر باید نرم افزار یا سیستم را به تصویر بکشد.
۲- ورودی و خروجی‌های اولیه را به دقت تعریف کنید.
۳- با عمل ایزوله کردن پردازش‌ها، موجودیت‌ها، ذخایر داده یرای تجزیه در سطوح بعدی، بایستی پالایش گردد.
۴- تمام خطوط اطلاعاتی بایستی با اسامی با معنی مشخص گردد.
۵- پیوستگی جریان اطلاعات باید در تمام سطوح حفظ گردد.
۶- در هر لحظه فقط یک بخش تجزیه و پالایش می گردد.

کار پنجم: ایجاد مدل کنترل(CFD)

مدلسازی رفتار سیستم نشان دهنده تغییر وضعیت سیستم و وقـوع رویـدادها در آن اسـت. بـرای اینکـار از نمـودار جریان کنترل (Control Flow Diagram – CFD) بایستی استفاده نمود و سپس نسبت بـه تـدوین و تشریح کنترل‌ها اقدام کرد. در این راستا رسم**STD** برای نمایش تغییر وضعیت سیستم نیز لازم است.
برای تهیه CFD تمام اطلاعات موجود روی پیکان‌های DFD را پاک می‌کنیم و رویدادها و کنتـرل هـا را روی یک خط افقی می‌نویسیم.



شکل ۱۷ : نمودار کنترل سیستم خانه امن (CFD)

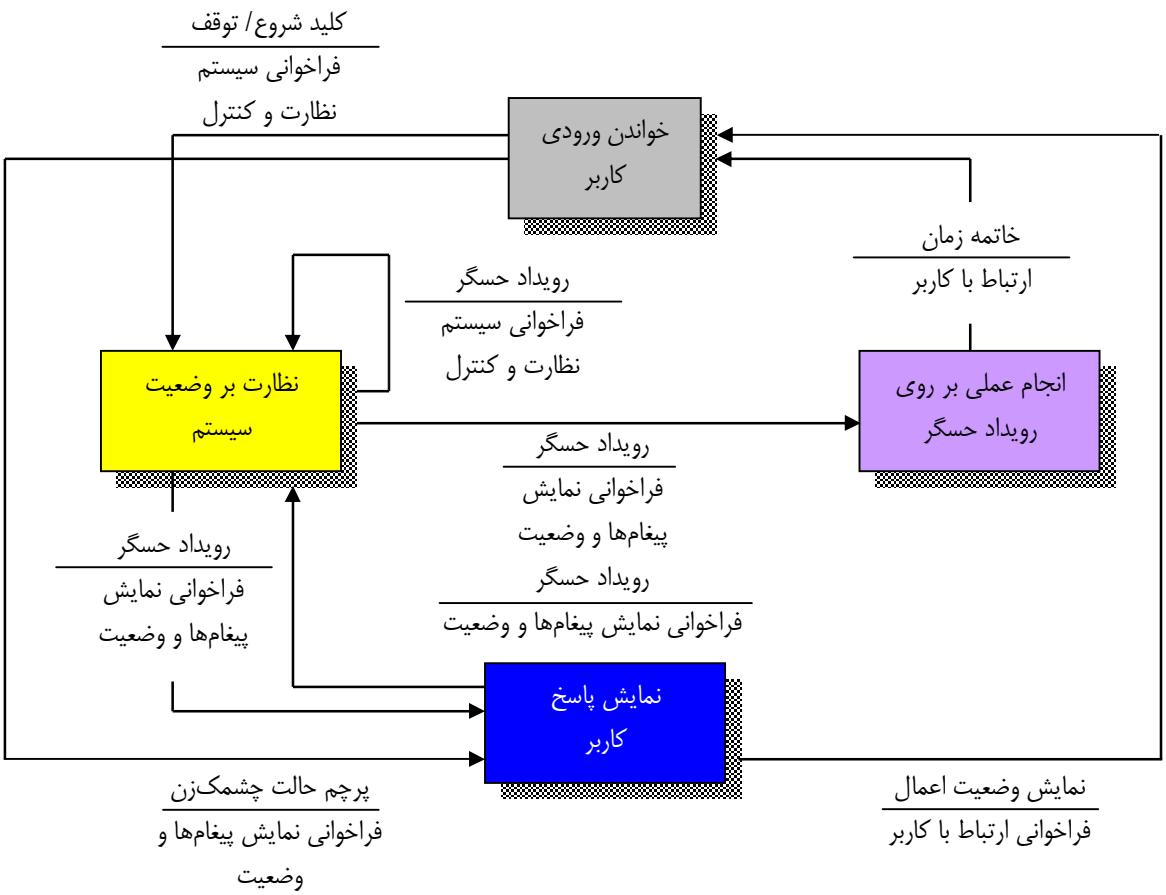
کار ششم: ایجاد مدل تغییر حالت (STD)

رویدادهای مهم در سیستم

۱-رویدادحس کننده ها

۲-اعلام خطر توسط نمایشگر (LCD BLINKING)

۳-کلیدهای شروع/ خاتمه START /STOP SWITCHES (برای کاربر است)



شکل ۱۸ : نمودار انتقال وضعیت سیستم خانه امن

تدوین مشخصه فرایندPSPEC

فرهنگ داده‌ها DD

فرهنگ داده‌ها لیستی سازماندهی شده از تمام عناصر داده‌هایی است که مربوط به سیستم می‌باشند، همراه با تعاریف دقیق و قاطع به گونه‌ای که کاربر و تحلیل‌گر سیستم فهم مشترکی از ورودی‌ها، خروجی‌ها، مؤلفه‌های حافظه‌ها و حتی محاسبات میانی داشته باشند.

امروزه، فرهنگ داده‌ها همیشه به عنوان بخشی از یک CASE (ابزاری برای تحلیل و طراحی ساختیافته) پیاده‌سازی می‌شود. اگرچه این قالب فرهنگ‌ها از یک به ابزار به ابزار دیگر متفاوت است، اکثر آنها حاوی اطلاعات زیر می‌باشند:

- نام- نام اولیهٔ شیء داده یا کنترل، حافظهٔ داده، یا موجودیت خارجی.
- نام مستعار- نام‌های دیگر استفاده شده برای اولین وارده.

- محل استفاده و نحوهٔ استفاده- لیستی از فرآیندهایی که این شیء داده یا کنترل را استفاده می‌کنند همراه با نحوهٔ استفاده از آنها (برای مثال، ورودی فرآیند، خروجی از فرآیند، به عنوان حافظه، به عنوان موجودیت خارجی).
- توصیف محتویات- توضیحی برای نمایش محتویات.
- اطلاعات تکمیلی- اطلاعات دیگری در مورد انواع داده‌ها، مقادیر اولیه (در صورت وجود)، محدودیت‌ها، و مانند آن.

تست‌های فصل دوازده: مدلسازی تحلیل نیازها

۱- Use Case ها سناریوهای کاربردی و تعامل با سیستم را تشریح می‌کنند.

الف) درست ب) نادرست

۲- نمودار گذار حالت در مدل تحلیل نشاندهنده نحوه برخورد سیستم در نتیجه وقوع رویدادهای خارجی است.

الف) درست ب) نادرست

۳- تحلیل ساختیافته ابزاری بسیار مناسب برای مدلسازی کار سیستم‌های اطلاعاتی است و نه برای مسایل مهندسی بلادرنگ.

الف) درست ب) نادرست

٤- کدامیک از موارد زیر از اهداف ساخت یک مدل تحلیل نیست.

الف) تعریف مجموعه‌ای از نیازهای نرم‌افزار ب) تشریح نیازها مشتری

ج) توسعه یک راه‌حل خلاصه برای مسأله د) استقرار پایه‌ای برای طراحی نرم‌افزار

۵- نمودار جریان داده‌ها

الف) ارتباط بین اشیاء داده‌ای را نشان می‌دهد.

ب) توابعی را نشان می‌دهد که جریان داده را تبدیل می‌کند.

ج) تصمیمات منطقی اصلی را به محض و قوع مشخص می‌سازند.

د) عکس‌العمل سیستم را در مقابل رویدادهای خارجی نشان می‌دهند.

٦- نمودار ارتباط- موجودیت

الف) روابط بین اشیاء داده‌ای را نشان می‌دهد.

ب) توابعی را نشان می‌دهد که جریان داده را تبدیل می‌کند.

ج) نشان می‌دهد که چگونه داده توسط سیستم تبدیل می‌شود.

د) عکس‌العمل سیستم را نسبت به رویدادهای خارجی نشان می‌دهد.

۷- نمودار گذار انتقال

الف) روابط بین اشیاء داده‌ای را نشان می‌دهد.

ب) توابعی را نشان می‌دهد که جریان داده را تبدیل می‌کند.

ج) نشان می‌دهد که چگونه داده توسط سیستم تبدیل می‌شود.

د) عکس‌العمل سیستم را نسبت به رویدادهای خارجی نشان می‌دهد.

۸- مدل داده‌ای شامل سه بخش اطلاعات به هم وابسته است:

الف) صفات ب) اشیاء داده‌ای ج) روابط د) همه موارد فوق

۹- روابط نشان داده شده در مدل داده‌ای بایستی نشان دهندهٔ

الف) طول و عرض داده (عمق و ارتفاع داده) ب) جهت ج) چندی و الزام د) احتمال و ریسک

۱۰- تصور اولیه نمودار ارتباط- موجودیت در مدل داده‌ای آن است که امکان نرمال‌سازی روابط بین جداول را بدهد.

الف) درست ب) نادرست

۱۱- **با هدف مدلسازی یک وضعیت رفتاری یک حالت عبارتست از**

الف) مصرف کننده و تولید کننده داده ب) سلسله مراتب داده‌ای

ج) نمایشی قابل مشاهده از رفتار د) فرآیند خوش تعریف

۱۲- نمودارهای جریان کنترل عبارتند از:

الف) برای سیستم‌های مبتنی بر رویداد نیاز به مدلسازی آنها است.

ب) برای تمام سیستم مورد نیاز هستند.

ج) به جای نمودارهای جریان داده‌ها استفاده می‌شوند.

د) برای مدلسازی واسطهای کاربران مفید هستند.

۱۳- **مشخصات فرآیندها برای توصیف تمام جریان فرآیندهایی که در نمودار جریان داده نهایی وجود دارند و باید طوری نوشته شوند که از یک زبان طراحی برنامه (PDL) استفاده گردد.**

الف) درست ب) نادرست

۱٤- فرهنگ داده (Data Dictionary) شامل توصیفات کدامیک از نرم‌افزارهای زیر است.

الف) مؤلفه‌های پیکربندی ب) اشیاء داده‌ای

ج) نمودار د) نمادها

۱۵- جدول فعال‌سازی فرآیند (PAT) شامل نمایی از فرآیند اطلاعاتی است که در نمودار انتقال حالت (STD) وجود دارد.

الف) درست ب) نادرست

فصل ۱۳: اصول و مفاهیم و طراحی نرم‌افزار

طراحی چیست؟ طراحی بازنمایی مهندسی و هدفمند چیزی است که قرار است ساخته شود. طراحی را می‌توان مطابق با خواسته‌های مشتری پیش برد و در عین حال براساس مجموعه معیارهای از پیش تعریف شدهٔ طراحی "خوب"، کیفیت آن را ارزیابی کرد. در زمینهٔ مهندسی نرم‌افزار، طراحی بر چهار کانون اصلی متمرکز است: **داده‌ها، معماری، واسطها، پیمانه‌ها.**

روش‌های طراحی نرم‌افزار از عمق، انعطاف‌پذیری و ماهیت، از ارتباط اندکی با شیوه‌های کلاسیک طراحی مهندسی برخوردارند. با این وجود شیوه‌هایی برای طراحی نرم‌افزار وجود دارند؛ معیارهایی برای کیفیت طراحی در دسترس می‌باشند و می توان از نشان‌گذاری طراحی استفاده کرد.

تعریفی دیگر از طراحی

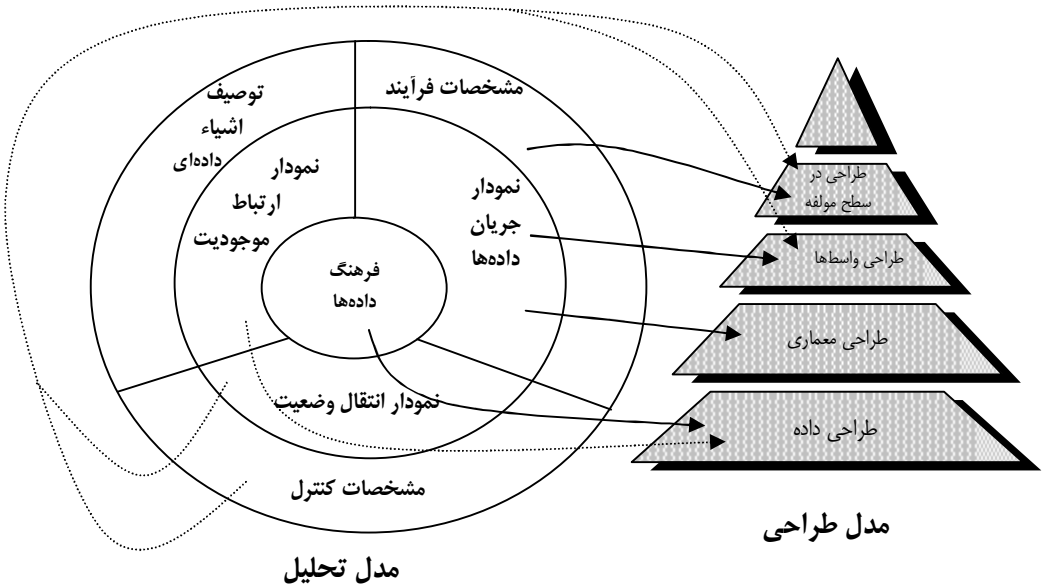
فرایند بکار گیری تکنیک‌ها اصول و قواعد مورد نیاز برای تعریف یک وسیله، یک فرایند، یک پردازش یا یک سیستم تا حد تفصیل است که با واقعیت فیزیکی آن تطبیق داشته باشد. مدل طراحی مبنایی برای پیاده سازی خواهد بود.

هر تغییری در زمان پیاده سازی بایستی روی مدل طراحی و متعاقب آن روی مدل تحلیل نیاز اعمال گردد.

طراحی نرم‌افزار و مهندسی نرم‌افزار

طراحی نرم‌افزار جزء بخش اصلی و فنی مهندسی نرم‌افزار بوده و بدون توجه به مدل به کار رفتهٔ فرآیند نرم‌افزار، اعمال می‌گردد. در آغاز و پس از تحلیل و تعیین نیازمندی‌های نرم‌افزار، طراحی نرم‌افزار با انجام سه فعالیت فنی یعنی طراحی، تولید برنامه، و آزمون، که در ساخت و تأیید نرم‌افزار ضرورت دارند انجام می‌شود. هر یک از این فعالیت‌ها، اطلاعات را به گونه‌ای تغییر می‌دهد که در نهایت به نرم‌افزار معتبر کامپیوتری منجر می‌گردد.

هر یک از عناصر مدل تحلیلی (فصل ۱۱)، اطلاعات لازم را در ایجاد چهار مدل طراحی به منظور تعیین کامل طراحی فراهم می‌آورند.



شکل ۱: برگردان مدل تحلیلی به طراحی نرم‌افزار

طراحی داده‌ها ، مدل اطلاعاتی تولید شده در طول تحلیل را بـه سـاختارهای داده‌ای لازم در اجـرای نـرم‌افـزار تبدیل می‌کند. اشیاء و روابط تعیین شدهٔ داده‌ها در نمودار ارتباط موجودیت‌ها و محتوای مشروح داده‌ای در فرهنـگ داده‌ها، مبنای فعالیت طراحی داده‌ها را فراهم می‌کنند. بخشی از طراحی داده‌ها ممکن اسـت بـا طراحـی معمـاری نرم‌افزار همراه شود. طراحی جزیی‌تر داده‌ها همراه با طراحی هر یک از مولفه‌های نرم‌افزار صورت می‌گیرد.

طراحی معماری ، رابطهٔ بین عناصر اصلی ساختاری نرم‌افزاری را تعیین می‌کند، یعنی رابطهٔ " الگوهای طراحـی" به کار رفته در تحقق نیازمندی‌های تعیین شدهٔ سیستم و محدودیت‌های مؤثر بر شـیوهٔ اجـرای الگوهـای طراحـی معماری نمایش طراحی معماری یا – چارچوب یک سیستم کامپیوتری – حاصل تعیین سیسـتم، مـدل تحلیلـی و ارتباط سیستم‌های فرعی تعیین شده در مدل تحلیلی است.

طراحی واسط ، توصیف کنندهٔ نحوهٔ ارتباط نرم‌افزار در محدودهٔ خود، با سیستم‌هایی کـه بـا آن تعامـل دارنـد و افرادی است که آن را به کار می‌برند. یک واسط بر گردش اطلاعات (مثل داده‌ها و یا کنتـرل) و نـوعی خاصـی از رفتار دلالت دارد. بنابراین نمودارهای جریان داده‌ها و کنتـرل، اکثـر اطلاعـات لازم بـرای طراحـی واسـط را ارایـه می‌کنند.

طراحی مؤلفه ، عناصر ساختاری معماری نرم‌افزار را بـه توصـیف رویـه‌ای مولفـه نـرم‌افـزاری تبـدیل مـی‌کنـد. اطلاعات به دست آمده از STD, CSPEC, PSPEC پایه و اساس طراحی مؤلفه به شمار می‌رونـد. اهمیـت طراحی نرم‌افزار را تنها با یک کلمه یعنی " کیفیت " می‌توان بیان کرد. طراحی روندی است که طـی آن کیفیـت در مهندسی نرم‌افزار، بهبود می‌یابد. طراحی، نمونه‌هایی از نرم‌افزار را که از لحاظ کیفی قابل ارزیابی هستند، در اختیـار ما قرار می‌دهد. طراحی تنها راهی است که به واسطهٔ آن می‌تـوانیم بـه طـور صـحیح نیازمنـدیها و خواسـته‌هـای مشتری را به یک محصول نرم‌افزاری یا سیستم تکمیل شده تبدیل کنیم.

طراحی نرم‌افزار یک فرآیند تکراری است که به موجب آن نیازمندی‌ها و ضرورت‌ها برای ساخت نـرم‌افـزار، تبـدیل به یک " طرح یا نقشه " می‌شوند. در ابتدا، طرح یک دید کلی از نرم‌افزار را نشان می‌دهـد. یعنـی آنکـه طراحـی در سطح بالایی از انتزاع ارائه می‌شود. با انجام تکرار در طراحی، پالایش بعدی، به نمـایش طراحـی در سـطوح بسـیار پایین‌تر انتزاع منجر می‌گردد. این سطوح نیز برای دستیابی به نیازمندی‌ها قابل ردیابی است، امـا ایـن نـوع ارتبـاط ظریف‌تر می‌باشد.

راهنمای ارزیابی یک طراحی خوب به شرح زیر ارایه شده‌اند:

- طراحی باید همهٔ ضرورت‌های آشکار موجود در مدل تحلیل را تحقق بخشیده و با تمامی خواسـته‌هـا و نیازهای ضمنی و مطلوب مشتری سازگار باشد.
- طراحی باید برای کسانی که برنامه‌نویسی می‌کنند و نیز اشخاصی که نرم‌افزار را آزمون نموده و بعـداً آن را پشتیبانی می‌کنند، راهنمایی خوانا و قابل درک باشد.
- طراحی باید تصویر کاملی از نرم‌افزار را ارائه کرده و حوزه‌های داده‌ای، کـاربردی و رفتـاری را از دیـد پیاده‌سازی مورد توجه قرار دهد.

برای ارزیابی کیفیت یک طراحی خوب باید:

- طراحی باید یک ساختار معماری ارایه کند که (۱) با استفاده از الگوهای قابل تشخیص طراحـی ایجـاد شده باشد. (۲) متشکل از اجزاء و عناصری باشد که خصوصیات طراحی خوب را نشان می‌دهنـد (بعـداً در این فصل مورد بحث قرار می‌گیرند) و (۳) به شیوه‌ای تکـاملی اجـرا شـده و بـدین ترتیـب اجـرا و آزمون را تهسیل کند.
- طراحی باید پیمانه‌ای (ماژولار) باشد یعنی نرم‌افزار باید به طور منطقـی بـه اجزایـی تقسـیم شـود کـه اعمال اصلی و فرعی خاصی را انجام دهند.
- طراحی بایستی نمایش‌های مجزایی از داده‌ها، معماری، واسطها و اجزاء (پیمانه‌ها) را در برداشته باشد.

- طراحی باید به ساختارهای داده‌ای منجر شود که برای پیاده‌سازی اشیاء مناسب بوده و از الگوهای قابل تشخیص داده‌ها ناشی می‌شوند.
- طراحی باید به اجزایی منتهی گردد که خصوصیات مستقل کاربردی را نمایش دهند.
- طراحی باید به واسطهایی ختم شود که از پیچیدگی روابط بین پیمانه‌ها و محیط خارجی می‌کاهند.
- طراحی باید حاصل کاربرد یک شیوهٔ قابل تکرار با استفاده اطلاعات به دست آمده در طول تحلیل نیازمندیهای نرم‌افزاری باشد.

اصول طراحی

طراحی نرم‌افزار، هم یک فرآیند است و هم یک مدل. داویس [DAV95] مجموعه اصولی را برای طراحی نرم‌افزار پیشنهاد می‌کند که این اصول به شرح زیر ارایه می‌شوند:

- **باریک بینی نباید در فرآیند طراحی وجود داشته باشد.** یک طراح خوب بایستی رهیافت‌های دیگر را در نظر داشته باشد و هر یک را براساس ضرورت‌های مساله، منابع موجود برای انجام کار و مفاهیم طراحی مورد قضاوت قرار دهد.
- **طراحی باید قابل ردیابی به مدل تحلیل باشد.** از آن جا که هر یک از عناصر مدل طراحی اغلب قابل ردیابی به ضروریات چندگانه است، بنابراین وجود روشی برای پیگیری چگونگی رفع نیازمندی‌ها در مدل طراحی لازم است.
- **طراحی نباید دوباره‌کاری باشد.** سیستم‌ها با استفاده از مجموعه الگوهای طراحی ساخته می‌شوند که احتمالاً با خیلی از آنها قبلاً هم برخورد داشته‌اند. این الگوها باید همواره به عنوان گزینه دیگر دوباره‌کاری انتخاب گردند. منابع محدودند! لذا زمان طراحی باید صرف ارایه نظرات واقعاً جدید و یکپارچه کردن الگوهای از قبل موجود شود.
- **طراحی باید فاصلهٔ منطقی بین نرم‌افزار و مساله موجود در جهان واقعی را به حداقل برساند.** یعنی، ساختار طراحی نرم‌افزار باید (در صورت امکان) ساختار میدان مساله را تقلید نماید.
- **طراحی باید از یکنواختی و یکپارچگی برخوردار باشد.** اگر این طور به نظر برسد که توسعهٔ کل کار برعهدهٔ یک شخص بوده، در این صورت طراحی یکسان و یکنواخت است. قبل از شروع کار طراحی، قوانین، سبک و قالب‌ها باید برای تیم طراحی تعریف و تعیین گردد. توجه و دقت در تعیین واسطهای بین اجزای طراحی، یکپارچگی طراحی را به دنبال خواهد داشت.
- **ساختار طراحی باید پذیرای تغییر باشد.** مفاهیم مورد بحث طراحی در بخش بعدی، باعث تطابق آن با این اصل می‌باشند.
- **ساختار طراحی حتی در صورت مواجهه با داده‌های غیرعادی، رویدادها یا شرایط کاری، باید به آرامی از کار بایستد.** یک نرم‌افزار خوب طراحی شده نباید هرگز ناگهان متوقف گردد. بلکه باید طراحی آن به گونه‌ای باشد که با شرایط غیرعادی سازگار بوده و اگر قرار شد پردازشی را پایان دهد این کار به طور ملایم انجام دهد.
- **طراحی به معنای برنامه‌نویسی نیست و برنامه‌نویسی نیز معادل طراحی نمی‌باشد.** حتی پس از ایجاد طراحی‌های جزیی رویه‌ای برای اجزای برنامه، سطح انتزاع مدل طراحی بالاتر از برنامهٔ منبع است. تنها تصمیمات طراحی اتخاذ شده در سطح برنامه‌نویسی، جزییات ظریف پیاده‌سازی را که موجب برنامه‌نویسی طراحی رویه‌ای می‌شوند را مطرح می‌کند.
- **طراحی باید ضمن شکل‌گیری، از نظر کیفی مورد ارزیابی قرار گیرد نه بعد از اتمام.** مجموعه‌ای از مفاهیم طراحی (فصل قبل) و اقدامات طراحی برای کمک به طراح به منظور ارزیابی کیفی، در دسترس می‌باشند.

❖ **طراحی باید به منظور به حداقل رساندن خطاهای مفهومی (معنایی) مرور و بررسی شود.**
گاه هنگام بررسی و مرور طراحی، تمایل به تأکید روی جزییات وجود دارد. یعنی در جنگل فقط به دنبال درخت بودن. تیم طراحی بایستی قبل از نگرانی دربارهٔ مدل طراحی، تضمین کند که عناصر اصلی مفهومی در طراحی (حذف و از قلم افتادگی، ابهام و ناسازگاری) مورد توجه قرارگرفته و رفع و رجوع گشته‌اند.

قواعد طراحی

قواعد طراحی که در ادامه بحث ارایه شده است، در پاسخ سوالات زیر کمک می نماید:

- چه معیاری برای نرم افزار به مولفه های مجزا چیست ؟
- چگونه کار کردها یا ساختار داده ها از نمایش مفهومی نرم افزار مجزا می شود ؟
- آیا معیار یکنواختی وجود دارد که کیفیت طراحی نرم افزار را تعریف کند ؟
- هدف از نوشتن برنامه کار کردن آن و یا درست کار کردن آن است؟

GETTING A PROGRAM TO WORK OR GETTING IT RIGHT?

- هدف از نوشتن برنامه درست کار کردن است و برای اینکار :

خوب طراحی شود ⟵ تحلیل نیاز درست ⟵ تحلیل سیستم درست

انتزاع

وقتی برای هر مساله به دنبال راه‌حل پیمانه‌ای باشیم، بسیاری از سطوح انتزاع نیز مطرح می‌گردند. در بالاترین سطح انتزاع، راه‌حل با بکارگیری زبان محیط مسئله و به صورت کلی بیان می‌شود. در سطوح پایین‌تر انتزاع، جهت‌گیری و گرایش بیشتر رویه‌ای است. اصطلاحات مساله‌گرا در تلاش برای بیان یک راه‌حل با اصطلاحات اجرایی تلفیق می‌شوند. و نهایت این که در پایین‌ترین سطح انتزاع، راه‌حل به گونه‌ای بیان می‌شود که مستقیماً قابل اجرا باشد.

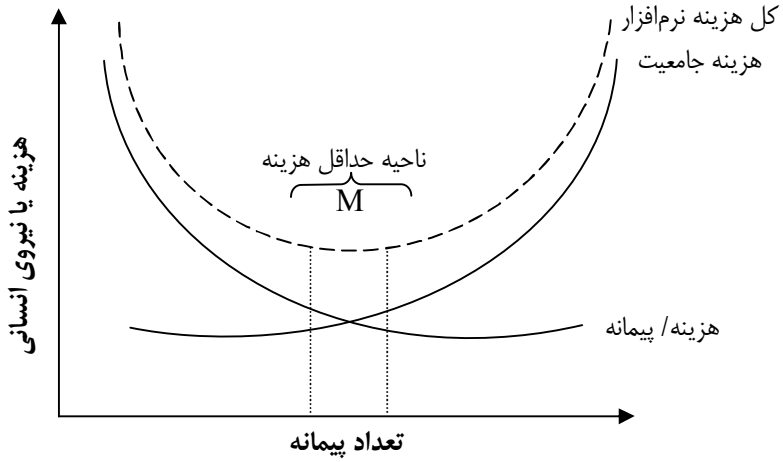
حین پیشروی در سطوح مختلف انتزاع، تلاش می‌کنیم تا انتزاع‌های رویه‌ای و داده‌ها ایجاد گردد. انتزاع رویه‌ای توالی مشخصی از دستورالعمل‌ها است که عملکرد خاص و محدودی دارد. نمونه‌ای از یک انتزاع رویه‌ای، وجود کلمهٔ " Open " برای یک " Door " می‌باشد. Open بر زنجیرهٔ طولانی مراحل رویه‌ای دلالت دارد: (مثلاً رفتن به سمت در، یا بردن دست و گرفتن دستگیره، چرخاندن دستگیره و کشیدن در، فاصله گرفتن از در و غیره)

انتزاع کل سومین شکل انتزاعی است که در طراحی نرم‌افزار به کار می‌رود. همانند انتزاع رویه‌ای و داده‌ای، انتزاع کل بر مکانیزم کنترل برنامه بدون مشخص کردن جزییات داخلی اشاره دارد. نمونهٔ انتزاع کنترل "سمافور همزمانی" است که برای هماهنگ کردن فعالیت‌ها در سیستم عامل به کار می‌رود.

پالایش

پالایش گام به گام یک راهبرد طراحی بالا به پایین است. برنامه با سطوح پالایشی متوالی جزییات رویه‌ای، توسعه می‌یابد. تا زمان دستیابی به دستورات زبان برنامه‌نویسی، ایجاد توسعهٔ سلسله مراتب با تجزیهٔ دستور ماکروسکوپی عمل (انتزاع رویه‌ای) بر شیوه‌ای گام به گام صورت می‌گیرد. دید کلی این مفهوم توسط Wirth ارایه می‌شود:

مفهوم پیمانه‌ای در نرم‌افزار کامپیوتر تقریباً پنج دهه، مورد حمایت واقع شده است. معماری نرم‌افزاری پیمانه‌ای است؛ نرم‌افزار بر اجزای نشانی پذیر با اسامی جداگانه به نام " پیمانه‌ها " تقسیم می‌شود که برای رفع نیازهای مساله، یکپارچه و مجتمع می‌باشند.



شکل۲: پیمانه شدن و هزینه نرم‌افزار

Meyers پنج معیار را در ارزیابی یک شیوهٔ طراحی و براساس توانایی آن در تعریف یک سیستم مؤثر پیمانه‌ای معرفی می‌کند:

- **تجزیه‌پذیری پیمانه‌ای:** اگر شیوهٔ طراحی مکانیزم منظمی را برای تجزیهٔ مساله به مسایل فرعی ارایه دهد، در آن صورت پیچیدگی مساله کلی کاهش یافته و بدین ترتیب تحقق یک راه‌حل مؤثر پیمانه‌ای میسر می‌گردد.
- **قابلیت ترکیب پیمانه‌ای:** اگر یک شیوهٔ طراحی امکان هم‌گذاری اجزای موجود (قابل استفاده مجدد) طراحی را در یک سیستم جدید به وجود بیاورد، آن‌گاه یک راه‌حل پیمانه‌ای به دست خواهد آمد که دوباره‌کاری نخواهد داشت.
- **قابلیت درک پیمانه‌ای:** اگر پیمانه‌ای به عنوان یک پیمانه مستقل قابل درک باشد (بدون ارجاع به پیمانه‌های دیگر) ساختن و تغییر آن آسان‌تر خواهد بود.
- **استمرار پیمانه‌ای:** اگر تغییرات کوچک در نیازمندی‌های سیستم، بیش از تغییر سیستم، منجر به تغییرات در پیمانه‌های جداگانه شود، تأثیر اثرات جانبی حاصل از تغییر به حداقل خواهد رسید.
- **محافظت پیمانه‌ای:** اگر در یک پیمانه شرایط غیرعادی پیش بیاید و تأثیرات آن به همان پیمانه محدود شود، تأثیر اثرات جانبی ناشی از خطا، به حداقل خواهد رسید.

معماری نرم‌افزار

آیا معماری نرم افزار بایستی از مدل نیاز استخراج گردد؟

معماری نرم‌افزار به " ساختار کلی نرم‌افزار و راه‌های ایجاد یکپارچگی ذهنی سیستم از طریق این ساختار " اشاره می‌کند. **معماری در ساده‌ترین شکل خود عبارت است از ساختار سلسله مراتبی اجزاء برنامه (پیمانه‌ها)، شیوهٔ ارتباط این اجزاء و ساختار داده‌هایی که توسط اجزاء مورد استفاده قرار می‌گیرند.**

یکی از اهداف طراحی نرم‌افزار، نقشهٔ معماری سیستم است. این نقشه چارچوب و مبنایی است که فعالیت‌های جزیی‌تر طراحی براساس آن انجام می‌گیرند. مجموعه الگوهای معماری، مهندس نرم‌افزار را قادر می‌سازد تا مفاهیم سطح طراحی را مجدداً قابل استفاده نماید.

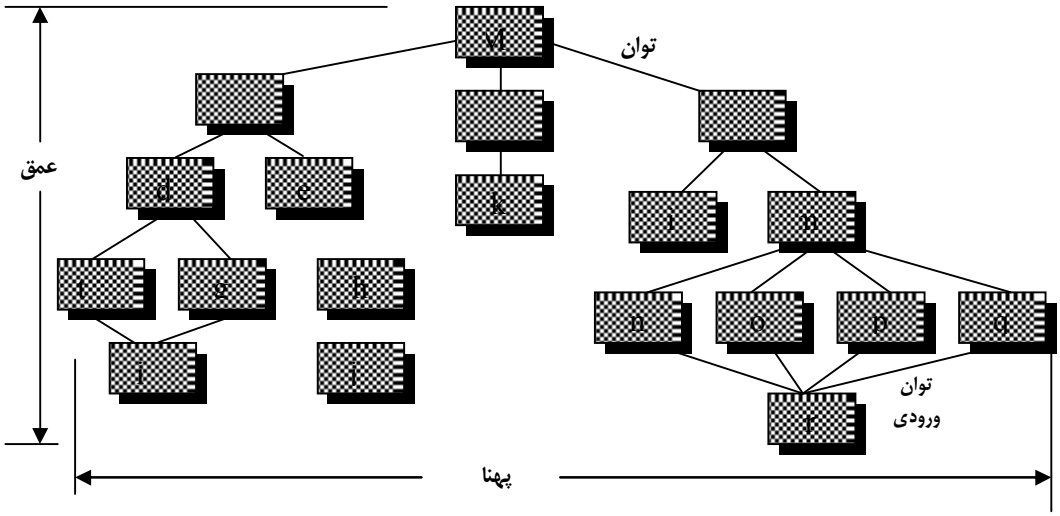
سلسله مراتب کنترل در برنامه ها

"**سلسله مراتب کنترل**" که "**ساختار برنامه**" نیز نام دارد، بیانگر سازمان‌دهی اجزاء برنامه (پیمانه‌ها) بوده و بر سلسله مراتب کنترل دلالت دارد. سلسله مراتب کنترل نشانهٔ جنبه‌های رویه‌ای نرم‌افزار مثل توالی فرآیندها، وقوع/ ترتیب تصمیمات یا تکرار عملکردها نبوده و لزوماً در تمامی سبک‌های معماری قابل کاربرد نمی‌باشد.

در آن دسته از سبک‌های معماری که قابل نمایش هستند، نشان‌گذاری‌های مختلفی برای نمایش سلسله مراتب کنترل به کار می‌روند.

توان خروجی مقیاس تعداد پیمانه‌هایی است که مستقیماً توسط پیمانه‌ای دیگر کنترل می‌شوند. **توان ورودی** نیز بیانگر تعداد پیمانه‌هایی است که یک پیمانه خاص را به طور مستقیم کنترل می‌کنند.

رابطهٔ کنترلی میان پیمانه‌ها به شیوهٔ زیر بیان می‌شود. پیمانه‌ای که پیمانه دیگری را کنترل می‌کند، **پیمانه حاکم** نام دارد و برعکس پیمانه تحت کنترل پیمانه دیگر، تابع **پیمانه کنترل کننده** می‌باشد.



شکل۳: ساختار سلسله مراتبی پیمانه‌ها در برنامه

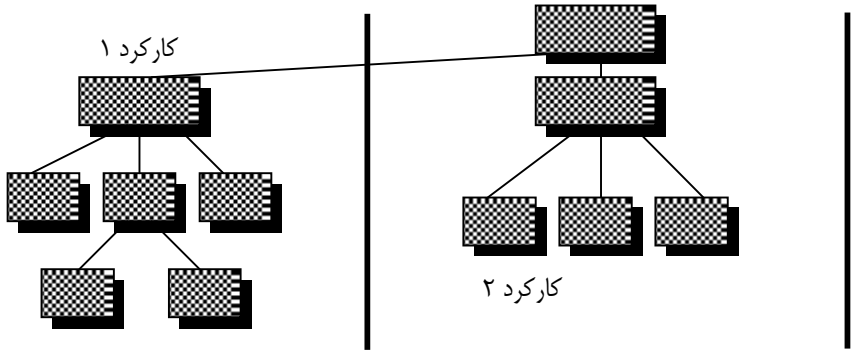
تجزیه ساختاری

اگر سبک معماری یک سیستم، سلسله مراتبی باشد، می‌توان ساختار برنامه را هم به صورت افقی و هم به طور عمودی تقسیم‌بندی کرد. با توجه به شکل۴ الف، تقسیم‌بندی افقی، شاخه‌های جداگانهٔ سلسله مراتب پیمانه‌ای را برای هر یک از وظایف اصلی برنامه تعیین می‌کند. پیمانه‌های کنترلی که با سایهٔ تیره‌تر نشان داده شده‌اند، برای هماهنگی ارتباط بین وظایف و اجرای آنها به کار می‌رود. ساده‌ترین شیوهٔ تقسیم‌بندی افقی، سه بخش را تعیین می‌کند که عبارتند از: ورودی، تغییر و تبدیل داده‌ها (که اغلب فرآیند نام دارد) و خروجی.

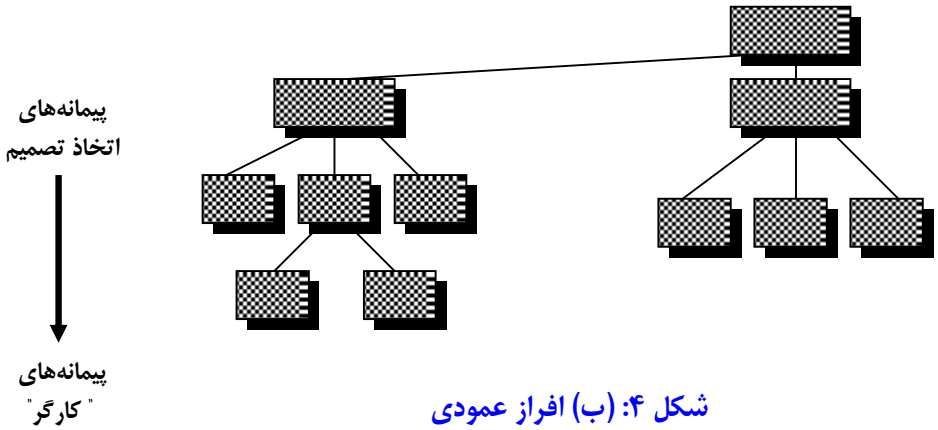
تقسیم‌بندی معماری به صورت افقی، مزایای روشنی را به همراه دارد.

- منجر به نرم‌افزاری می‌شود که آزمون آن آسان‌تر است.
- منجر به نرم‌افزاری می‌شود که نگهداری و حفظ آن آسان‌تر است.
- منجر به نرم‌افزاری می‌شود که انتشار آن از پیامدهای جنبی کمتری برخوردار است.
- نرم‌افزاری را ایجاد می‌کند که بسط و توسعهٔ آن ساده‌تر است.

از آن جا که کارکرد اصلی از یکدیگر جدا می‌گردند، تغییرات پیچیدگی کمتری دارند و بسط و توسعهٔ سیستم (که امری رایج است) بدون تأثیرات جانبی، راحت‌تر انجام می‌شود. از جنبهٔ منفی، تقسیم‌بندی افقی باعث می‌شود داده‌های بیشتری از واسطهای پیمانه عبور کرده و کنترل کلی گردش برنامه را دشوار و پیچیده می‌سازد(در صورتی که فرآیند مستلزم جابه‌جایی سریع از یک کارکرد به کارکردی دیگر باشد).



شکل ۴ : (الف) افراز عمودی



شکل ۴: (ب) افراز عمودی

تعریف ساختمان داده ها

ساختار داده‌ها، نمایش رابطهٔ منطقی میان عناصر جداگانهٔ داده‌ها است. از آن جا که ساختار اطلاعات بر طراحی نهایی رویه‌ای تأثیر خواهد داشت، ساختار داده‌ها به اندازهٔ ساختار برنامه در نمایش معماری نرم‌افزار اهمیت دارد.
ساختار داده‌ها تعیین کنندهٔ سازمان‌دهی، روش‌های دستیابی، میزان شرکت‌پذیری و جایگزین‌های فرآیند اطلاعات می‌باشد.

تعریف پردازش‌های نرم‌افزار

ساختار برنامه، سلسله مراتب کنترل را بدون توجه به توالی فرآیند و تصمیمات تعیین می‌کند. رویهٔ نرم‌افزار بر جزییات پردازشی هر پیمانه به طور جداگانه تأکید دارد. رویه باید تعیین دقیق پردازش از جمله توالی رویدادها، نقاط دقیق تصمیم‌گیری، اعمال تکراری و حتی سازمان‌دهی/ ساختار داده‌ای را ارایه دهد.
البته، بین ساختار و رویه ارتباطی وجود دارد. فرآیند تعیین شده برای هر پیمانه، باید ارجاع به تمامی پیمانه‌های تابع آن را در بر داشته باشد.

عدم نمایش اطلاعات (پنهان کردن کردن)(Information Hiding)

مفهوم پیمانه‌ای بودن یک سؤال اساسی را برای هر طراح نرم‌افزاری مطرح می‌کند. «برای دستیابی به بهترین مجموعهٔ پیمانه‌ها، چگونه یک راه‌حل نرم‌افزاری را تجزیه کنیم؟» اصل پنهان‌سازی اطلاعات(Information Hiding) بیانگر آن است که "وجه مشخصهٔ پیمانه‌ها، تصمیمات طراحی است که هر پیمانه از پیمانه‌های دیگر مخفی می‌سازد". به عبارتی دیگر، پیمانه‌ها باید طوری طراحی و مشخص شوند که اطلاعات (رویه و داده‌ها) موجود در هر پیمانه برای پیمانه‌های دیگری که به چنین اطلاعاتی نیاز ندارند، غیر قابل دسترسی باشد. بنابراین:

همه اطلاعات را به همه کاربران نباید نشان داد.

طراحی پیمانه های کارآمد

- **ویژگی پیمانه ای(modular) یک روش پذیرفته شده در کلیه رشته های مهندسی است.**
- **تاثیرات طراحی پیمانه‌ای:**
  - کاهش پیچیدگی
  - تسهیل تغییرات
  - امکان توسعه موازی مولفه های مختلف و در نتیجه آن تسهیل پیاده سازی
- **سه مفهوم قابل توجه در طراحی پیمانه‌ها:**
  - استقلال عملیاتی (Functional Independence)
  - انسجام (Cohesion)
  - پیوستگی (Coupling)

استقلال عملیاتی

مفهوم " استقلال عملیاتی " پیامد مستقیم پیمانه ساختن و مفاهیم انتزاع و پنهان‌سازی اطلاعات است. استقلال عملیاتی از طریق ایجاد پیمانه‌هایی با عملکرد یک منظوره و عدم ارتباط بیش از حد با پیمانه‌های دیگر، تحقق می‌یابد. به بیان دیگر، ما قصد داریم نرم‌افزاری را طراحی کنیم که هر پیمانه یک وظیفهٔ خاص فرعی از ضرورت‌ها را انجام داده و از دید سایر بخش‌های ساختار برنامه، واسط ساده‌ای داشته باشد.

توسعهٔ نرم‌افزاری با پیمانه‌ای شدن کارآمد یعنی پیمانه‌های مستقل، آسان‌تر است زیرا کار تقسیم‌بندی شده و واسطها ساده شده‌اند. (انشعابات را هنگام انجام توسعه توسط یک تیم، در نظر بگیرید.) نگهداری و آزمون پیمانه‌های مستقل ساده‌تر است زیرا تأثیرات ثانویه ایجاد شده به واسطهٔ تغییر طراحی/ برنامه، محدود می‌شوند. انتشار خطا کاهش می‌یابد و پیمانه‌هایی با قابلیت استفادهٔ مجدد امکان‌پذیر می‌گردند. به طور خلاصه، استقلال عملیاتی رمز طراحی خوب و طراحی، رمز کیفیت نرم‌افزار است. استقلال با دو معیار کیفی ارزیابی می‌گردد: **انسجام و اتصال**

انسجام" قدرت عملیاتی نسبی یک پیمانه است و اتصال مقیاس وابستگی نسبی پیمانه‌ها به هم می‌باشد.

چرا استقلال دارای اهمیت است؟

- **توسعه آسانتر**
- **تسهیل درنگهداری و آزمایش**
- **کاهش انتشار خطا**
- **قابلیت استفاده مجدد**

انسجام(Cohesion)

انسجام یا چسبندگی، بسط طبیعی مفهوم پنهان‌سازی اطلاعات است. یک پیمانه یکپارچه، یک وظیفه منفرد را در یک رویه نرم‌افزاری و با برقراری ارتباط محدود با رویه‌های در حال اجرای سایر بخش‌های برنامه، انجام می‌دهد. به بیان ساده‌تر، یک پیمانه منسجم (به طور ایده‌آل) باید تنها یک کار را انجام دهد.

انسجام را می‌توان به صورت یک " طیف " نشان داد. ما همیشه تلاش می‌کنیم تا به **انسجام و پکپارچگی زیاد** دست یابیم. هر چند که دامنهٔ متوسط طیف نیز اغلب قابل قبول است. مقیاس انسجام، غیرخطی است. یعنی آن که، انسجام انتهای پایانی به مراتب بدتر از دامنهٔ متوسط (اواسط طیف) است که تقریباً به اندازهٔ انسجام انتهای بالایی، قابل قبول می‌باشد. عملاً لزومی ندارد که طراح به طبقه‌بندی انسجام در یک پیمانه خاص بپردازد، بلکه باید مفهوم کلی را درک نموده و هنگام طراحی پیمانه‌ها بایستی از سطوح پایین انسجام اجتناب کرد.

انواع پیمانه‌ها از نظر انسجام

- تصادفا منسجم (Coincidentally Cohesive)

در انتهای پایینی (نامطلوب) طیف, با پیمانه‌های سرو کار داریم که مجموعه ای از وظایف را انجام می‌دهد که ارتباط محکمی با هم ندارند.

- منطقا منسجم (Logically Cohesive)

اگر پیمانه ای وظایفی انجام دهد که با هم ارتباط منطقی دارند (مثل پیمانه هایی که خروجی ها را بدون در نظر گرفتن نوع آنها تولید می‌کنند), آن پیمانه را منطقا منسجم گویند.

- انسجام موقتی (Temporal Cohesive)

هنگامی که پیمانه ای حاوی وظایفی است که باید در یک گستره زمانی مشترک اجرا شوند, آن پیمانه دارای انسجام موقتی است. مثال

- انسجام رویه ای (Procedural Cohesive)

سطوح میانه ای از انسجام از لحاظ استقلال پیمانه ها نسبتا به هم نزدیکند. هنگامی که عناصر پردازشی یک پیمانه با هم ارتباط داشته باشند و باید به ترتیب مشخصی انجام شوند , انسجام رویه ای وجود دارد.

- انسجام ارتباطی (Communicational Cohesive)

همه عناصر پردازشی به یک ناحیه از ساختمان داده ها متمرکز شوند.

مثالی از سطوح پایین انسجام

به عنوان مثالی از انسجام پایین , پیمانه ای را در نظر می‌گیریم که پردازش خطا را برای یک برنامه پکیج نرم‌افزاری تحلیل مهندسی انجام می‌دهد.این پیمانه زمانی فراخوانی می‌شود که داده های محاسبه شده، از مرزهای از پیش تعیین شده تجاوز نمایند. وظایف ان عبارتند از:

۱. محاسبه داده های مکمل بر اساس داده های محاسبه شده اولیه
۲. تولید گزارش خطا (با محتویات گرافیکی) روی ایستگاه کاری کاربر
۳. اجرای محاسبات بعدی که مورد درخواست کاربر است
۴. بهنگام سازی یک بانک اطلاعاتی
۵. فراهم آوردن امکان انتخاب از منو برای پردازش‌های بعدی

گرچه وظایف قبلی ارتباط محکمی با هم ندارند، هر کدام یک نهاد عملیاتی مستقل هستند که ممکن است به بهترین نحو به صورت پیمانه‌ای جداگانه اجرا شوند. ترکیب عملکردها در یک پیمانه واحد، تنها می‌تواند به افزایش احتمال انتشار خطا به هنگام اصلاح یکی از وظایف پردازشی فوق کمک کند.

اتصال(Coupling)

اتصال، مقیاس ارتباط بین پیمانه‌ها در ساختار نرم‌افزار است. اتصال به پیچیدگی واسط بین پیمانه‌ها، نقطه ورود (دخول) و ارجاع به یک پیمانه و نوع داده‌های عبوری از واسط، بستگی دارد.

در طراحی نرم‌افزار، تلاش ما در جهت **پایین‌ترین سطح ممکن اتصال** است. اتصال سادهٔ بین پیمانه‌ها باعث ایجاد نرم‌افزاری می‌شود که فهم آن ساده‌تر بوده و هنگام وقوع خطاها در یک محل و پخش آنها در سیستم، کمتر در معرض « اثر فزونگری» ایجاد شده قرار دارد.

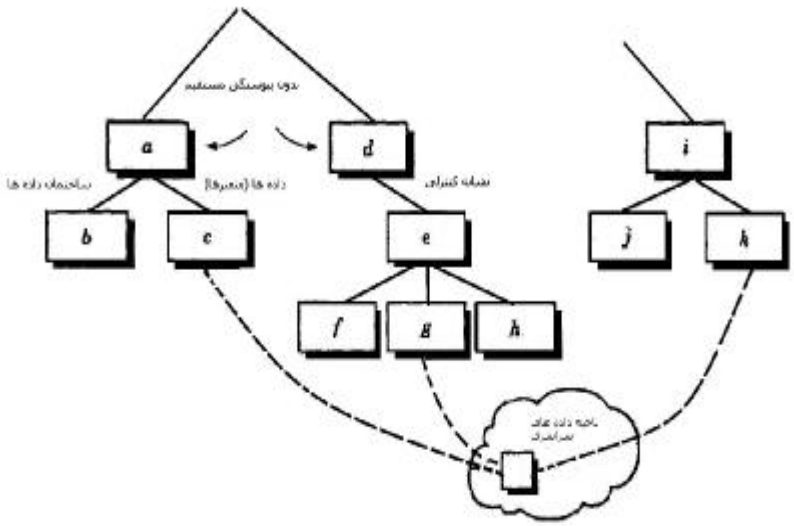
در شکل زیر پیمانه c تابع پیمانه a بوده و از طریق لیست قراردادی ارگومان که داده‌ها از طریق آن منتقل می‌شوند، قابل دسترسی است. تا مادامی که لیست آرگومان ساده وجود دارد(یعنی داده‌های ساده انتقال یافته و ارتباطی یک به یک بین اقلام برقرار است)، اتصال ضعیف یا(**اتصال داده‌ها**) در این بخش از ساختار به نمایش گذاشته می‌شود. نوعی اتصال داده‌ها به نام " **اتصال برچسبی**" زمانی ظاهر می‌شود که بخشی از ساختار داده‌ای(به جای آرگومان‌های ساده) از طریق واسط پیمانه، منتقل می‌گردد. این اتصال بین پیمانه‌های a وb بوجود می‌آید.

در سطوح متوسط " اتصال با انتقال کنترل بین پیمانه‌ها " توصیف می‌گردد. " **اتصال کنترل**" در اکثر طراحی‌های نرم‌افزاری بسیار رایج است و در شکل زیر با عبور " نشانه نمای کنترل " (متغیری که تصمیمات را در یک پیمانه تابع یا حاکم کنترل می‌کند) بین پیمانه‌های d و e نشان داده شده است.

در صورت **ارتباط پیمانه‌ها با محیط خارج از نرم‌افزار، سطوح نسبتاً بالای اتصال**، به وجود می‌آیند. بعنوان مثال I/O، پیمانه‌ها را به دستگاه‌های خاص، قالب‌ها و پروتوکل‌های ارتباطاتی، متصل می‌کند.

" اتصال خارجی " ضروری است اما باید به تعداد کمی از پیمانه‌ها محدود شود. اتصال سطح بالا نیز زمانی پدید می‌آید که تعدادی از پیمانه‌ها به ناحیهٔ سراسری داده‌ها ارجاع می‌کنند " **اتصال مشترک**" که نام این حالت می‌باشد.

بالاترین میزان اتصال با عنوان " **اتصال محتوا** " زمانی به وجود می‌آید که یک پیمانه، از داده‌ها یا اطلاعات کنترلی موجود در حد و مرز پیمانه‌ای دیگر، استفاده می‌کند. ثانیاً اتصال محتوایی به هنگام ایجاد شاخه‌هایی در میان یک پیمانه نیز اتفاق می‌افتد. این حالت اتصال، قابل پیشگیری بوده و باید از آن اجتناب کرد.



شکل ۵: " نمودار نخست " ساختار برنامه برای حس‌گرهای نمایش دهنده

انواع اتصال (Coupling)

- **اتصال داده‌ای** (Structural Coupling)

مادمی که لیستی از آرگومانهای ساده وجود داسته باشند (یعنی داده های ساده عبور داده شوند، و بین عناصر موجود، تناظر یک به یک برقرار باشد)، در این بخش از برنامه اتصال اندکی موسوم به اتصال داده‌ای به چشم می‌خورد.(درشکل بین a و c)

- **اتصال برچسبی** (Stamp Coupling)

هنگامی یافت می‌شود که بخشی از ساختمان داده‌ها (به جای آرگومانهای ساده) از طریق یک واسط پیمانه عبور داده شوند. (درشکل بین a و b)

- **اتصال کنترلی** (Control Coupling)

در سطوح میانی، اتصال با تبادل کنترل بین پیمانه‌ها مشخص می‌شود. اتصال کنترلی در اکثر طراحی‌های نرم افزاری بسیار متداول است و در شکل بین c-d موجود است.

- **اتصال خارجی** (External Coupling)

سطوح نسبتا بالای اتصال هنگامی رخ می‌دهد که پیمانه‌ها با محیط خروجی نرم‌افزار ارتباطی تنگاتنگ داشته باشند. اتصال خارجی ضروری است، ولی باید به تعداد اندکی از پیمانه‌ها با یک ساختار محدود شود.

- **اتصال مشترک** (Common Coupling)

زمانی که پیمانه یک دسته از داده‌های سرتا سری را آدرس‌دهی می‌کند , اتصال بالایی رخ می‌دهد .

- **اتصال محتویات** (Content Coupling)

بالاترین درجه اتصال، اتصال محتویات است و هنگامی رخ می‌دهد که یک پیمانه از داده‌های اطلاعات کنترلی نگهداری شده در مرز پیمانه دیگر استفاده کند. از این نوع اتصال می‌توان و باید حذر کرد.

ابتکار در طراحی پیمانه‌های موثر و کاراً

پس از ایجاد ساختار برنامه، پیمانه‌ای کردن مؤثر و کارآمد به واسطهٔ اجرای مفاهیم طراحی که در ابتدای فصل معرفی شدند، تحقق می‌یابد. ساختار برنامه براساس مجموعه ذهنیت‌های زیر، قابل دست‌کاری است:

۱. " **اولین تکرار**" **ساختار برنامه را به منظور کاهش اتصال و بهبود انسجام، مورد ارزیابی قرار دهید.** پس از ساخت برنامه، می‌توان پیمانه‌ها را از جهت بهبود استقلال آنها، تفکیک یا ترکیب کرد.
یک " **پیمانه ترکیبی**" در ساختار نهایی برنامه، تبدیل به دو یا چند پیمانه می‌گردد. **پیمانه تفکیکی** اغلب زمانی حاصل می‌شود که پردازش مشترک در دو یا چند پیمانه وجود داشته و به صورت یک پیمانه منسجم و جداگانه، مجدداً قابل تعریف است. هنگامی که اتصال در سطح بالا مورد نظر می‌باشد، می‌توان گاهی پیمانه‌های را برای کاهش انتقال کنترل، ارجاع به داده‌های سراسری و پیچیدگی واسط، ترکیب کرد.

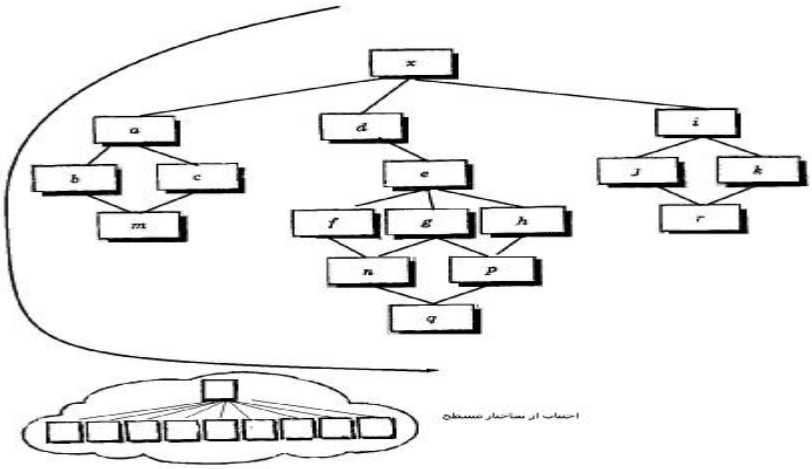
۲. **سعی کنید ساختارهایی با گنجایش خروجی زیاد را به حداقل رسانده و همان‌طور که عمق افزایش می‌یابد، برای گنجایش ورودی تلاش کنید.** ساختار داخل ابر در شکل زیر از تجزیهٔ عوامل استفادهٔ مفیدی نمی‌کند. تمامی پیمانه‌ها، زیر یک تکه پیمانه کنترل به طور یکنواخت قرار دارند. به طور کلی، توزیع معقول‌تر کنترل در ساختار سمت راست، نشان داده شده است. این ساختار بیضی شکل بوده و تعدادی لایهٔ کنترل و پیمانه‌های کاملاً سودمند را در سطوح پایین‌تر نمایش می‌دهد.

۳. **حوزه تأثیر یک پیمانه را در محدودهٔ دامنهٔ کنترل همان پیمانه حفظ کنید.** حوزهٔ تأثیر یک پیمانه e، به تمام پیمانه‌هایی اطلاق می‌شود که تحت تأثیر تصمیم اتخاذ شده در پیمانه e هستند. دامنهٔ کنترل پیمانه e، همه پیمانه‌هایی هستند که به طور مستقیم یا با واسطه تابع پیمانه e می‌باشند. اگر تصمیم اتخاذ شده در پیمانه e بر پیمانه r تأثیر بگذارد، ذهنیت ۳ نقض شده، زیرا پیمانه r در حوزهٔ کنترل پیمانه e نمی‌باشد.

۴. **واسطهای پیمانه را به منظور کاهش پیچیدگی و بهبود سازگاری مورد ارزیابی قرار دهید.** پیچیدگی واسط، پیمانه، عامل عمدهٔ خطاهای نرم‌افزاری است. واسطها باید برای انتقال راحت اطلاعات طراحی شده و با وظیفهٔ پیمانه هماهنگ و سازگار باشند. ناسازگاری واسط (یعنی داده‌های ظاهراً نامربوط که از طریق لیست آرگومان یا تکنیک دیگری منتقل می‌شوند) نشانهٔ انسجام و یکپارچگی است. پیمانه مورد نظر بایستی مجدد مورد ارزیابی واقع شود.

۵. **پیمانه‌هایی را تعریف کنید که کارشان قابل پیش‌بینی است اما از پیمانه‌های بسیار محدودکننده اجتناب نمایید.** یک پیمانه زمانی قابل پیش‌بینی است که به صورت یک جعبهٔ سیاه تلقی شود، یعنی، داده‌های یکسان خارجی بدون در نظر داشتن جزییات داخلی پردازشی، تولید خواهند شد. پیمانه‌ای که پردازش را فقط بر یک عمل فرعی محدود می‌کند، انسجام زیادی را نشان می‌دهد و طراح نسبت به آن تمایل دارد. با این وجود آن پیمانه‌ای که به طور دلخواه اندازهٔ ساختاری داده‌ای محلی را و گزینه‌های جریان کنترل یا حالات واسط خارجی را محدود می‌کند، همواره مستلزم مراقبت و نگهداری خواهد بود تا چنین محدودیت‌هایی رفع گردند.

۶.**برای پیمانه‌هایی با " ورودی کنترل شده " تلاش نموده و از " اتصالات نامعقول" خودداری کنید.** این ذهنیت طراحی، دربارهٔ اتصال محتوایی، هشدار می‌دهد. در صورت محدودسازی و کنترل واسطهای پیمانه، درک و در نتیجه نگهداری و ترمیم نرم‌افزار ساده‌تر می‌گردد.



شکل ۵: " نمودار نخست " ساختار برنامه برای حس‌گرهای نمایش دهنده

بنابراین در طراحی پیمانه

- تقطیع وتلفیق (Exploded/ Imploded)

تقطیع پیمانه باعث تبدیل پیمانه به دو یا چند پیمانه در برنامه نهایی می‌شود و تلفیق پیمانه به ترکیب پردازش حاصل از دو یا چند پیمانه می انجامد.

- کوشش برای به حداقل رساندن ساختارهایی با توان خروجی بالا و تلاش برای افزایش دادن توان ورودی به موازات افزایش عمق شکل
- حفظ حوزه یک پیمانه در دامنه کنترل آن پیمانه: حوزه پیمانه عبارتست از کلیه پیمانه های دیگری که از تصمیم‌گیری انجام شده توسط یک پیمانه تاثیر بپذیرند. در شکل اگر r از e تاثیر بپذیرد، از این قاعده سرپیچی شده است.
- ارزیابی واسطهای پیمانه‌ها برای کاهش پیچیدگی و زواید و بهبود بخشیدن به سازگاری
- تعریف پیمانه‌هایی که عملکرد آنها قابل پیش‌بینی است و پرهیز از پیمانه هایی که همپوشانی دارند. پیمانه‌هایی که حافظه داخلی دارند می‌توانند غیرقابل پیش‌بینی باشند.
- کوشش برای دستیابی به پیمانه هایی با مدخل کنترل شده و پرهیز از اتصالات آسیب شناختی (Pathological Connection). اتصالات آسیب شناختی عبارت از ایجاد انشعاب یا رجوع به میانه یک پیمانه است.

مستند سازی طراحی

تعیین مشخصات طراحی با جنبه‌های متفاوتی از مدل طراحی سرو کار دارد و به موازات این که طراح، نمایش خود از نرم افزار را پالایش می‌کند، کامل می‌شود. ابتدا دامنه کلی کار شرح داده می‌شود. اکثر اطلاعاتی که در این مرحله ارایه می‌شود از مشخصات سیستم و مدل تحلیل(مشخصات خواسته های نرم افزار) بدست می‌آید.

سپس طراحی داده ها مشخص می‌شود. ساختار بانک اطلاعاتی، هر ساختار فایل خارجی، ساختمان داده‌های داخلی و یک ارجاع متقابل(Cross Reference) که اشیای داده‌ای را به فایل‌های مشخصی متصل می‌کند، همگی تعریف می‌شوند.

طراحی معماری نشان می‌دهد که معماری برنامه چگونه از مدل تحلیل بدست آمده است. بعلاوه از نمودارهای ساختاری برای نشان دادن سلسله مراتب پیمانه استفاده می‌شود.

طراحی واسط‌های داخلی و خارجی برنامه نمایش داده می‌شود و جزییات طراحی واسط انسان _ ماشین شرح داده می‌شود.

مولفه‌ها، عناصر جداگانه از سیستم مثل زیربرنامه‌ها، توابع یا رویه‌ها، ابتدا با زبان مادری شرح داده می‌شوند. در این مرحله عملکرد رویه‌ای یک مولفه (پیمانه) توضیح داده می‌شود. سپس از یک ابزار طراحی رویه‌ای برای ترجمه این نسخه با توصیفی ساختیافته استفاده می‌شود.

مشخصات طراحی شامل یک ارجاع متقابل (Cross Reference) خواسته ها است . هدف از این ارجاع متقابل که توسط یک ماتریس متقابل نشان داده می‌شود:

- اثبات بر آورده شدن همه خواسته‌ها توسط طراح نرم‌افزار
- نشان دادن اینکه کدام مولفه‌ها برای پیاده سازی خواسته‌های مشخص اهمیت حیاتی دارد.

آخرین بخش از مشخصات طراحی شامل داده های مکمل است. توصیفات الگوریتم‌ها، رویه‌های دیگر، داده‌های جدولی، نیازهایی از مستندات دیگر به عنوان یادداشت خاص با پیوستی جداگانه ارایه می‌شوند. بهتر است که یک جزوه راهنمای نصب و راه اندازی مقدماتی تهیه و ضمیمه مستند طراحی شود.

چارچوب گزارش طراحی سیستم در صفحه بعد ارایه شده است.

تست‌های فصل سیزده: اصول و قواعد طراحی

۱- کدام یک از موارد زیر در مدل طراحی مورد توجه قرار نمی‌گیرند؟

الف- معماری ب) داده ج) واسطها د) محیط پروژه

۲- اهمیت طراحی نرم‌افزار را می‌توان درکدام کلمه خلاصه نمود؟

الف- صحت ب- پیچیدگی ج- کارایی د- کیفیت

۳- کدام یک از موارد زیر از ویژگی‌های مشترک تمام روش‌های طراحی نمی‌باشد؟

الف- مدیریت پیکربندی ب- نشانه مولفه عملیاتی ج- راهنماهای ارزیابی کیفیت د- پالایش ذهنی

۴- قبل از شروع به کار، باید مجموعه‌ای از قواعد طراحی وضع شوند تا انسجام و جامعیت طراحی را تضمین نماید.

الف- درست ب) نادرست

۵- کدام یک از انواع انتزاع در طراحی نرم‌افزار به کار می‌رود؟

الف- کنترلی ب) داده‌ای ج- رویه‌ای د- همه موارد

۶- هنگام استفاده از متدولوژی‌های طراحی ساختیافته، فرآیند پالایش مرحله ای غیر الزامی است.

الف- درست ب) نادرست

۷- از آنجایی که پیمانه‌ای بودن از اهداف مهم طراحی است، وجود پیمانه‌های متعدد در یک طراحی پیشنهادی ممکن نیست.

الف- درست ب) نادرست

۸- کدام یک از انواع مدل‌های زیر نمایانگر یک معماری نرم‌افزار نمی‌باشد؟

الف- داده ب- پویا ج- فرآیند (رویه) د- ساختاری

۹- سلسله مراتب کنترل نمایانگر کدام مورد زیر است؟

الف- ترتیب تصمیم‌گیری ب- سازماندهی پیمانه‌ها ج- تکرار عملیات د- ترتیب رویه‌ها

۱۰-افراز افقی برای عملیات برنامه، شاخه‌های جداگانه‌ای تعریف می‌کند. در حالی که افراز عمودی کنترل را به صورت بالا و پایین توزیع می‌کند.

الف- درست ب) نادرست

۱۱-طراحی ساختمان داده‌ها نسبت به طراحی الگوریتم زمان کمتری می‌گیرد. در نتیجه ممکن است به آخر کار موکول شود.

الف- درست ب) نادرست

۱۲-رویه نرم‌افزار بر کدام مورد زیر تاکید دارد؟

الف- سلسله مراتب کنترل در حالت انتزاعی‌تر ب- پردازش جزییات هر پیمانه به طور مجزا

ج- پردازش جزییات هر مجموعه از پیمانه‌ها به طور جمعی د- رابطه بین کنترل و رویه

۱۳-انسجام(Cohesion) نمایشگر کیفی درجه‌ای است که پیمانه در آن:

الف- میتواند به طور فشرده‌تر نوشته شود. ب- تنها بر یک مورد تاکید دارد.

ج- قادر به اتمام به موقع عملیات خود می‌باشد. د- به پیمانه‌های دیگر و جهان خارج متصل می‌شود.

۱۴-اتصال(Coupling) نمایشگر کیفی درجه‌ای است که یک پیمانه در آن:

الف- میتواند به طور فشرده‌تر نوشته شود. ب- تنها بر یک مورد تاکید دارد.

ج- قادر به اتمام به موقع عملیات خود می‌باشد. د- به پیمانه‌های دیگر و جهان خارج متصل می‌شود.

۱۵-ذهنیات طراحی معمولاً تنها مورد استفاده دانشجویان است و مهندسین نرم افزار مجرب از آن بی‌نیازند.

الف- درست ب) نادرست

۱۶- دلیل نادرستی طراحی سطح مولفه قبل از طراحی داده این است که:

الف- طراحی مولفه وابسته به زبان برنامه‌سازی است ولی طراحی داده این طور نیست.

ب- طراحی داده آسان‌تر است.

ج- طراحی داده سخت‌تر است.

د- ساختمان داده معمولاً بر روش طراحی سطح مولفه تاثیر می‌گذارد.

۱۷- پیمانه‌های c، g و k هرکدام به داده‌ای (مانند فایل یا ناحیه‌ای از حافظه که دستیابی سراسری بـه آن وجـود دارد) در ناحیه سراسری داده‌ها دستیابی دارند. پیمانه c، داده‌ای را مقدار دهی مـی‌نمایـد. سـپس پیمانـه g مقدار آن را محاسبه و بهنگام می‌کند. در این حالت کدام یک از سطوح اتصال (Coupling) وجود دارد؟

الف- Data Couplig
ب- External Coupling
ج- Common Coupling
د- Content Coupling

فصل ۱۴: طراحی معماری نرم‌افزار

معماری چیست؟

وقتی در مورد معماری یک ساختمان یا بنا صحبت می‌کنیم، نگرش‌های متفاوتی به ذهنمان می‌رسد. در ساده‌ترین سطح، شکل کلی ساختمان فیزیکی را در نظر می‌گیریم. اما در واقع معماری آن، بسیار پیچیده‌تر از اینهاست. این حالتی است که در آن اجزای مختلف ساختمانی به صورتی با هم ترکیب می‌شوند که یک کل سازمان‌یافته و منسجم را تشکیل می‌دهند. این شیوه‌ای است که با آن ساختمان، همراه با ساختمان‌های مجاور خود در محیط سازگاری می‌یابد و هماهنگ می‌شود. در این جا این ساختار سیستم تا حدی هدف مشخص شده خود را نشان داده و نیازهای صاحبش را برطرف می‌کند. در این جا حس زیباشناسی ساختمان یعنی تأثیر بصری ساختمان، و شیوه ترکیب بافت‌ها، رنگ‌ها و مواد برای خلق نما و محیط زنده درونی مطرح است. در این جا جزییات کوچک مثل طراحی تسهیلات نور، نوع کف‌پوش و ... فهرستی بلند بالا را به وجود می‌آورند و نهایتاً، موضوع اصلی هنر است.

در مورد ساختار نرم‌افزاری چه می‌توان گفت؟ Bass و همکارانش [BAS98] این عبارت دشوار و دیرفهم را به شکل زیر تعریف کرده‌اند:

معماری نرم‌افزار در یک برنامه یا سیستم محاسباتی عبارتست از ساختار یا ساختارهای سیستم که شامل اجزای نرم‌افزاری، مشخصه‌های مشهود برونی این اجزاء و ارتباطات میان آنها می‌باشد.

چرا معماری؟

معماری، یک نرم‌افزار عملیاتی نیست. بلکه نمودی است که مهندس نرم‌افزار را قادر می‌سازد:

۱)میزان تأثیر طرح را در مرتفع نمودن نیازمندی‌های بیان شده، تحلیل کند.

۲)معماری‌های جایگزین دیگر را در مرحله‌ای که تغییر طرح هنوز نسبتاً آسان است، بررسی کند.

۳)خطرات مربوط به ساخت نرم‌افزار را کاهش دهد.

طراحی داده‌ها

طراحی داده‌ها، هم چون دیگر فعالیتهای مهندسی نرم‌افزار (که گاهی به آن معماری داده‌ها نیز می‌گویند) مدلی از داده‌ها و/ یا اطلاعات را در سطح بالایی از حالت انتزاعی، ایجاد می‌کند. سپس مدل داده‌ای به صورت بازنمایی خاص پیاده‌سازی درمی‌آید که می‌توان آن را به وسیله سیستم مبتنی بر کامپیوتر پردازش کرد. در بسیاری از برنامه‌های کاربردی نرم‌افزار، معماری داده‌ها تأثیر شگرفی بر معماری نرم‌افزار دارد که باید آن را پردازش کند.

- اشیاء داده‌ای را پالایش کرده و مجموعه‌ای از انتزاعات داده‌ای را توسعه می‌دهد.
- صفات اشیاء داده‌ای را به عنوان یک یا چند ساختمان داده‌ای پیاده‌سازی می کند.
- ساختمان‌های داده‌ای به منظور اطمینان از ارتباط‌های مناسب بازنگری می‌گردد.
- ساختمان داده‌ها تا حد ممکن ساده‌سازی می گردند.

طراحی مبتنی بر داده ها یک روش طراحی معماری نرم افزار است که به راحتی امکان گذر از مرحله تحلیل نیاز به طراحی توصیف ساختمان برنامه را می دهد.

مدل سازی داده، ساختار داده، پایگاه داده، و انبار داده‌ها

اشیای داده‌ای که در طول تحلیل نیازهای نرم‌افزاری تعریف شده‌اند، با استفاده از نمودارهای موجودیت/ رابطه و فرهنگ داده‌ها، مدلسازی می‌شوند. کار طراحی داده‌ها، عناصر مورد نیاز مدل نیازها را در سطح جزء نرم‌افزاری به ساختمان داده‌ها و وقتی لازم باشد معماری پایگاه داده‌ای را در سطح برنامه کاربردی، تبدیل می‌کند.

طراحی تفصیلی داده‌ها (در سطح اجزاء)

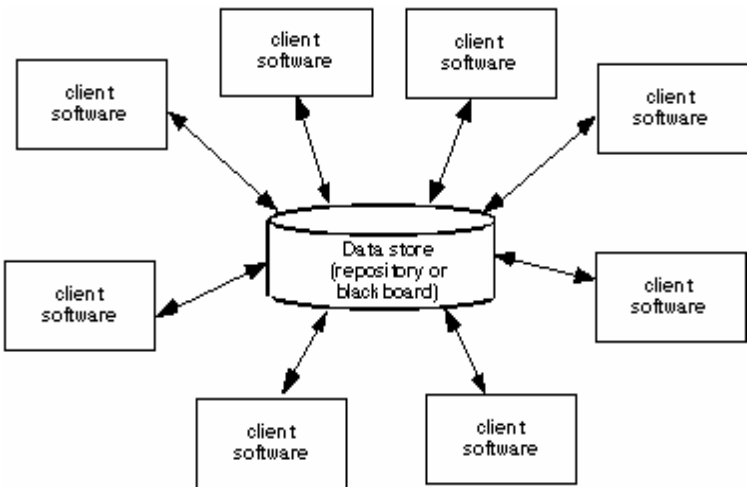
طراحی داده‌ها در این سطح روی ارایه و نمایش ساختمان داده‌هایی متمرکز می‌شود که مستقیماً توسط یک یا چند جزء نرم‌افزاری قابل دسترسی هستند. Wasserman [WAS80] مجموعه اصولی را پیشنهاد کرد که ممکن است برای مشخص کردن و طراحی چنین ساختارهایی به کار روند. در واقع، طراحی داده‌ها در طول خلق مدل تحلیل شروع می‌شود. فراخوانی تحلیل نیازها و طراحی اغلب هم‌پوشانی می‌شود. مجموعه اصول زیر را برای مشخصه‌های داده‌ای، در نظر بگیرید:

۱- اصول تحلیل نظام‌مند که در مورد عملکرد و رفتار به کار می‌رود باید در مورد داده‌ها نیز به کار رود.
۲- تمام عملیات و ساختمان داده‌ها باید شناسایی شوند، در طراحی یک ساختار داده‌ای کارآمد باید عملیاتی که روی ساختار داده‌ها صورت می‌گیرد را در نظر گرفت.
۳- یک فرهنگ داده‌ای به وجود آورده و از آن برای تعریف طراحی داده‌ها و برنامه استفاده کنید.
۴- تصمیمات مربوط به سطوح پایین طراحی داده‌ها را باید تا اواخر فرآیند طراحی به تعویق انداخت.
۵- نمودار ساختاری داده‌ها تنها باید برای پیمانه‌هایی شناخته شده باشد که مستقیماً از داده‌های موجود در ساختار استفاده می‌کنند.
۶- کتابخانه‌ای از ساختارهای داده‌ای مفید و عملیاتی که ممکن است در مورد آنها به کار رود، باید ایجاد گردد.
۷- طراحی نرم‌افزار و زبان برنامه‌نویسی باید خصوصیات و شناسایی انواع داده‌های انتزاعی را پشتیبانی کند.

سبک‌های معماری

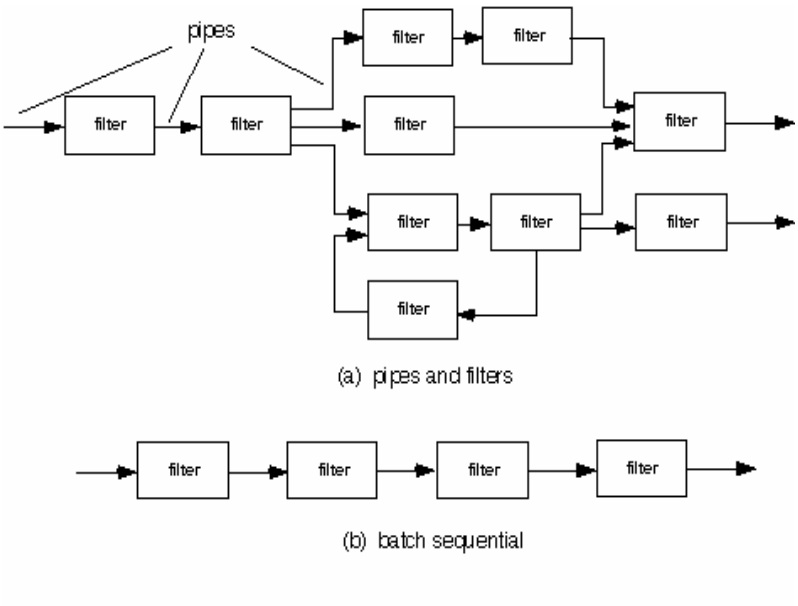
- **Data-centered architectures**
- **Data flow architectures**
- **Call and return architectures**
- **Object-oriented architectures**
- **Layered architectures**

معماری‌های متمرکز بر داده‌ها- یک مخزن داده‌ای در مرکز این معماری قرار دارد و اغلب توسط دیگر اجزایی که به روزسازی، افزودن، حذف یا کارهای دیگر اصلاحی را در مورد مخزن انجام می‌دهند، قابل دسترسی است.



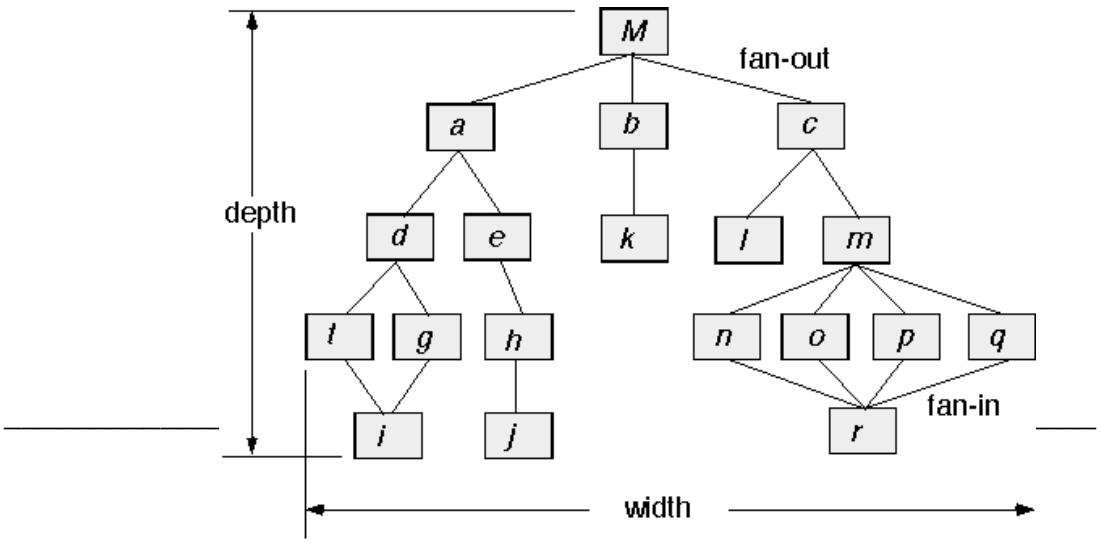
شکل ۱: نمودار معماری داده‌گرا

معماری‌های جریان داده‌ها. این معماری وقتی به کار گرفته می‌شود که داده‌های ورودی قرار است از طریق یک سری اجزاء محاسباتی یا تغییراتی، به داده‌های خروجی تبدیل شوند.



شکل ۲: نمودار معماری جریان داده

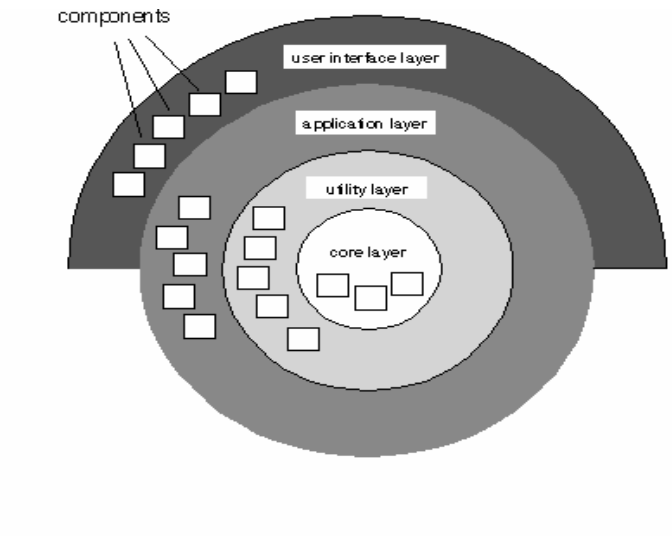
معماری فراخوانی و بازگشت- این سبک از معماری طراح نرم‌افزار (معمار سیستم) را قادر می‌سازد تا به ساختار برنامه‌ای دست یابد که از نظر اصلاح و ارزیابی نسبتاً ساده است.



شکل ۳: نمودار معماری فراخوانی و برگشت

معماری‌های شی‌گرا. اجزای یک سیستم در برگیرنده داده‌ها و عملیاتی هستند که باید برای تغییر داده‌ها به کار روند. ارتباط و هماهنگی بین اجزا از طریق عبور پیام‌ها حاصل می‌شود(این معماری در فصول آتی مورد بحث قرار می‌گیرد).

معماری‌های لایه‌ای. ساختار اصلی این معماری در شکل زیر نشان داده شده است. تعدادی لایه‌های مختلف تعریف شده‌اند که هر کدام به عملیاتی دست می‌یابند که به طور گسترده‌ای به مجموعه دستورات ماشین نزدیک‌تر می‌شوند. در لایه خارجی‌تر، اجزاء در خدمت عملیات واسط کاربر هستند. در لایه داخلی‌تر، اجزاء کار ارتباط سیستم عامل را انجام می‌دهند. لایه‌های میانی خدمات استفاده و بهره‌برداری و عملیات کارکردی نرم‌افزار را مهیا می‌کنند.



شکل ۴: نمودار معماری لایه‌ای

روش‌های طراحی معماری (An Architectural Design Method)

فعالیت‌های تحلیل طراحی به صورت تکراری انجام می‌گیرند:

۱- **جمع‌آوری (سناریوها) طرح‌ها.** مجموعه‌ای از موارد مورد استفاده ارایه شده‌اند تا نمایانگر سیستم از دیدگاه کاربر باشند.

۲- **بدست آوردن نیازها، محدودیت‌ها و توصیف محیط.** این اطلاعات به عنوان بخشی از مهندسی نیازها، لازمند و برای اطمینان از این امر استفاده می‌شوند که تمام نگرانی‌های مشتری، کاربر وسهامداران مورد توجه قرار گرفته‌اند.

۳- **توصیف سبک‌ها/ الگوهای معماری که برای بررسی طرح‌ها و نیازها انتخاب شده‌اند.** این سبک‌ها را باید با استفاده از دیدگاه‌های معماری مورد توصیف قرار داد مثل:

- **دیدگاه پیمانه**
- **دیدگاه پردازشی**
- **دیدگاه جریان داده‌ای**

۴- **ارزیابی صفات خاصه کیفی با در نظر گرفتن هر یک به طور جداگانه.** تعداد صفات خاصه انتخابی برای تحلیل، تابعی از زمان موجود برای بازبینی و مقداری است که نسبت به آن این نگرش‌ها با سیستم مورد نظر مرتبط می‌شوند. نگرش‌های کیفی برای برآورد طراحی معماری شامل قابلیت اطمینان، عملکرد، امنیت، قابلیت نگهداری، انعطاف‌پذیری، آزمون‌پذیری، قابلیت حمل، استفاده مجدد و قابلیت عملیاتی در همکاری می‌باشد.

۵- **شناسایی حساسیت صفات خاصه به صفات خاصه مختلف معماری برای در یک سبک به خصوص.** این کار با ایجاد تغییرات کوچکی در ساختار و تعیین چگونگی حساسیت یک صفت خاصه کیفی مثلاً عملکرد نسبت به تغییر، انجام می‌گیرد. صفت خاصه‌ای که تا حد زیادی تحت تأثیر تنوع معماری قرار دارد، " نقاط حساسیت " نامیده می‌شود.

۶- **متخصصان با استفاده از تحلیل حساسیت که در مرحله ۵ صورت گرفته، معماری‌های کاندیدا را پیشنهاد می‌کنند.**

پیچیدگی معماری

یک روش مفید برای ارزیابی پیچیدگی کلی معماری پیشنهادی، عبارتست از در نظر گرفتن وابستگی میان اجزاء درون معماری. این وابستگی‌ها ناشی از جریان کنترل/ اطلاعات درون سیستم می‌باشند.

ZHAO سه نوع وابستگی ارایه می‌دهد: [ZHA98]

- **وابستگی‌های مشترک** نمایان‌گر ارتباطات وابسته‌ای در میان مصرف‌کنندگانی هستند که از منبعی یکسان استفاده کرده یا تولیدکنندگانی که برای مصرف‌کنندگانی یکسان تولید می‌کنند. مثلاً، در مورد دو جزء، v،u اگر u و v هر دو به یک سری داده سرتاسری یکسانی رجوع کنند، یک واسطه وابستگی مشترک بین v،u وجود دارد.
- **وابستگی‌های جریان** نمایان‌گر ارتباطات وابستگی میان تولیدکننده و مصرف‌کننده‌های منبع است. در مورد دو جزء v،u اگر باید u قبل از جریان یافتن کنترل در v تکمیل شود یا اگر u به وسیله پارامترهایی با v ارتباط برقرار می‌کند، در این صورت یک جریان وابستگی میان این دو وجود دارد.
- **وابستگی‌های محدود شده** نمایان‌گر محدودیت‌ها و قیودی در جریان نسبی کنترل میان مجموعه‌ای از فعالیت‌ها هستند. مثلاً در مورد دو جزء v،u ، آنها نمی‌توانند در یک زمان اجرا شوند پس یک واسطه وابستگی محدود و مقید شده بین آنها وجود دارد.

وابستگی‌های **مشترک و جریان** که مورد توجه ZAHO قرار گرفت از بعضی جهات شبیه مفهوم انسجام و اتصال هستند.

نگاشت نیازها در یک معماری نرم‌افزار

انتقال از جریان داده ها به ساختمان نرم افزار با طی ۶ مرحله انجام می گیرد.

طراحی ساختیافته اغلب به عنوان یک روش طراحی مبتنی بر جریان داده‌ای معرفی می‌گردد زیرا انتقال راحت‌تر از نمودار جریان داده‌ای (DFD) را به معماری نرم‌افزار فراهم می‌سازد. کار انتقال از جریان اطلاعات به ساختار برنامه به عنوان بخشی از فرآیند مرحله ۶ با موفقیت به انجام می‌رسد:

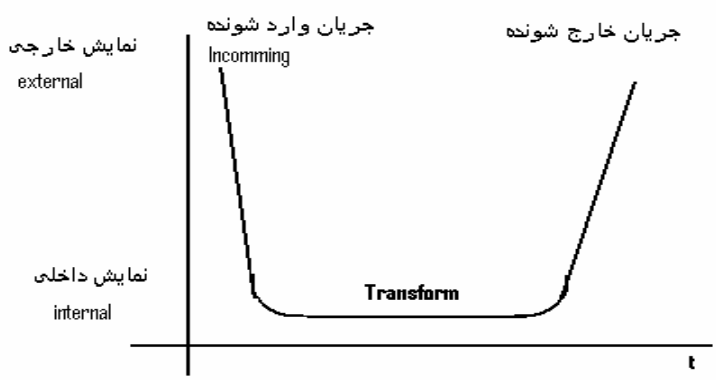
۱) نوع جریان اطلاعاتی تعیین می‌گردد.
۲) سرحدات جریان مشخص می‌شوند.
۳) DFD در ساختمان برنامه نگاشت می‌شود.
۴) سلسله مراتب کنترلی تعیین می‌گردد.
۵) ساختمان بدست آمده با استفاده از معیارهای طراحی و علوم تجربی بازنگری می‌شود.
۶) توصیف معماری بازنگری و تعیین می‌گردد.

تعیین نوع جریان اطلا عات

دو نوع جریان اطلاعات وجود دارد:

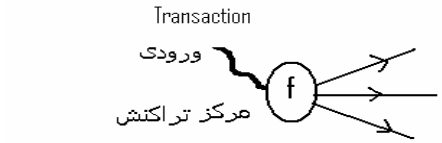
۱- جریان تبدیلی(Transform Flow)،
۲- جریان تراکنشی (Transaction Flow)

در جریان تبدیلی روال به صورت زیر است:



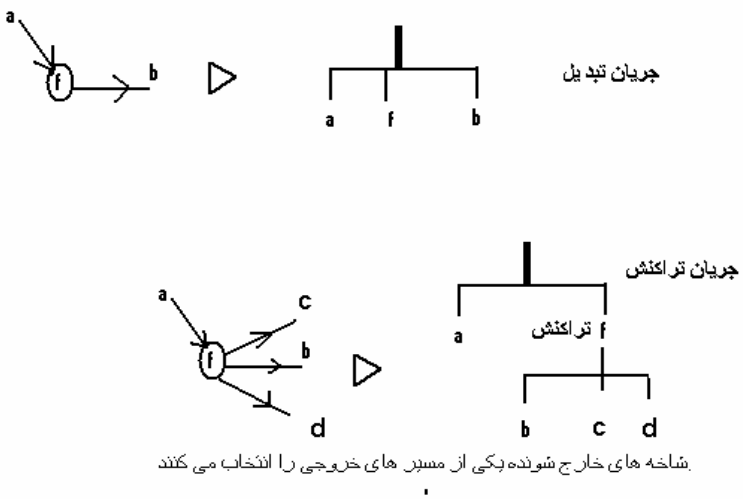
شکل ۵: نمودار جریان تبدیلی

در روش تراکنشی، جریانی وارد سیستم می شود و سیستم انتخاب می کند که از کدام شاخه خارج شود.



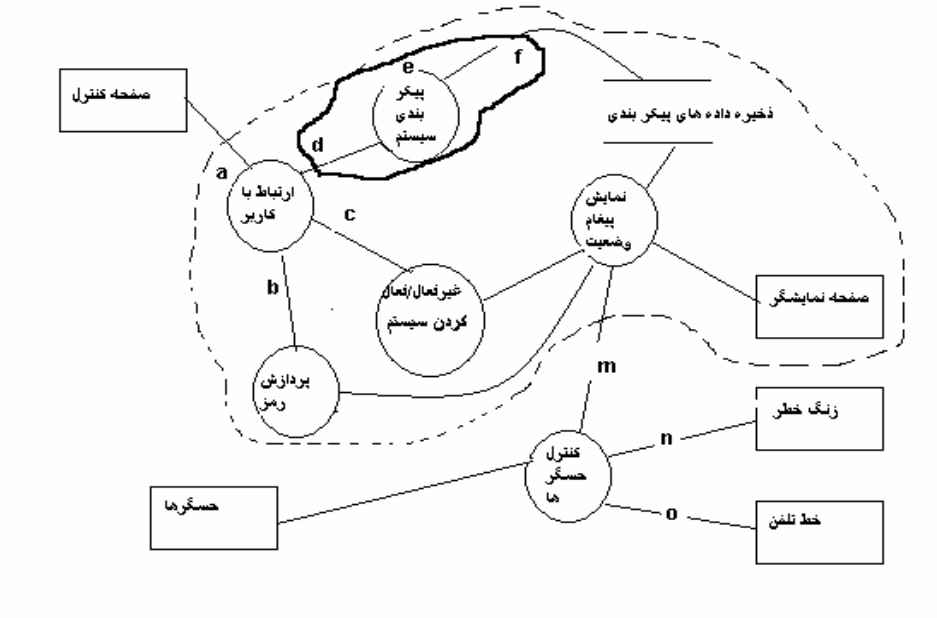
شکل ۶: نمودار جریان تراکنشی

مثال: تبدیل جریان داده به معماری نرم افزار:



شکل ۷: نحوه تبدیل جریان‌های تبدیلی و تراکنشی به معماری(ساختار) برنامه

مثال: سطح ۱ DFD خانه امن را در نظر بگیرید. داریم:

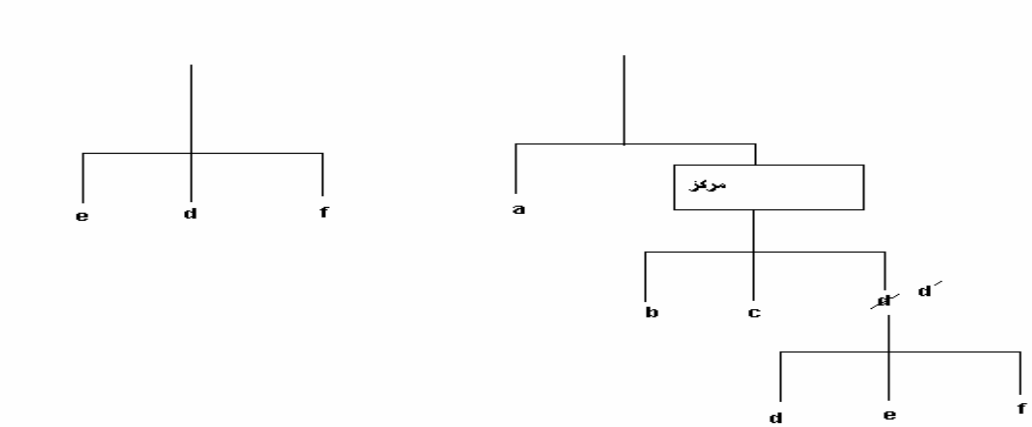


شکل ۸: نمودارجریان داده سیستم خانه امن

حال با توجه به DFD فوق مراحل طراحی معماری را پیش می‌گیریم.

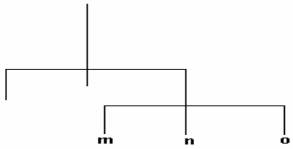
مرحله ۱و۲: تعیین مرز و نوع جریان:

در ابتدا قسمت خط چین در شکل را در نظر می‌گیریم. به نمونه مثال زیر توجه کنید.



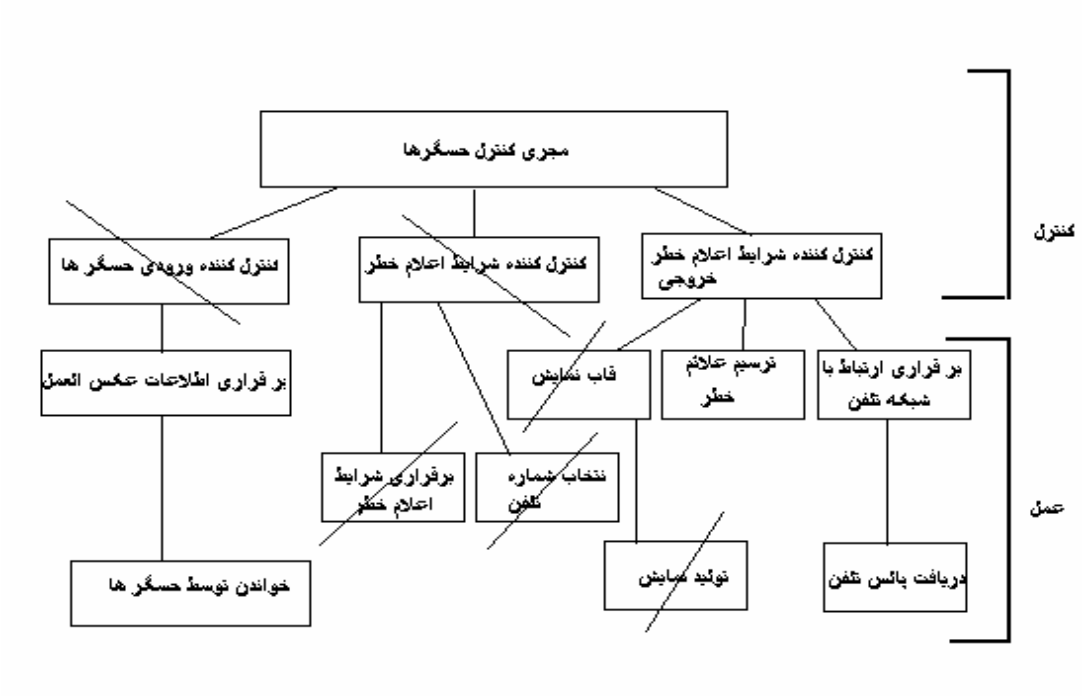
شکل ۹: مثالی از نحوه نگاشت جریان داده به ساختار برنامه

در سطوح بالایی معماری کنترل مطرح است ولی در سطوح پایینی نحوه تبدیل اطلا عات را می‌توان دید.
و یا قسمت کنترل حسگر یک تبدیل است چون در تراکنش یکی از خروجی‌ها انتخاب می شود ولی در کنترل حسگر هر سه خروجی همزمان اتفاق می افتد.



شکل ۱۰: مثالی از نحوه نگاشت جریان تراکنشی به ساختار برنامه

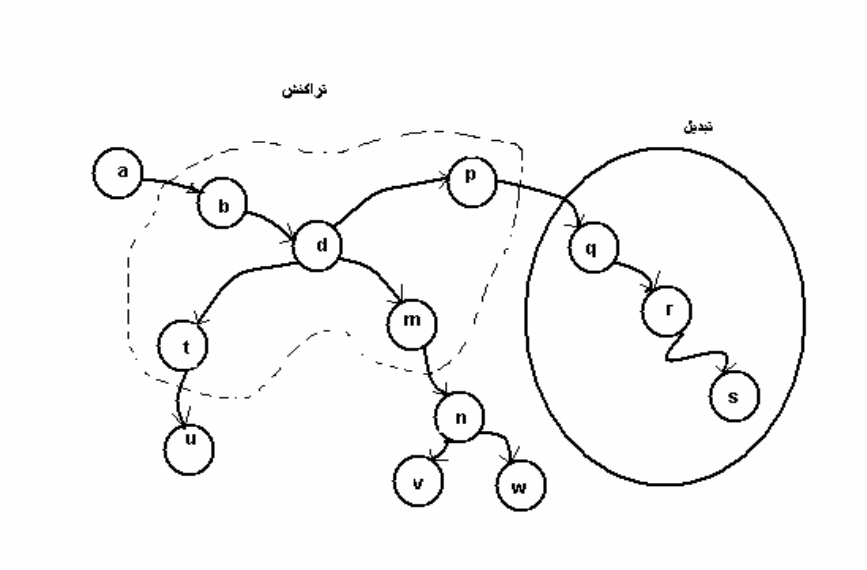
اگر مراحل ۳ تا ۵ که عبارت است از رسم تفصیلی و تعیین جریان و مرز آن را انجام دهیم در مرحله ۶ خواهیم داشت:



شکل ۱۱: مثالی از نحوه نگاشت جریان تراکنشی و تبدیلی سیستم خانه امن به ساختار

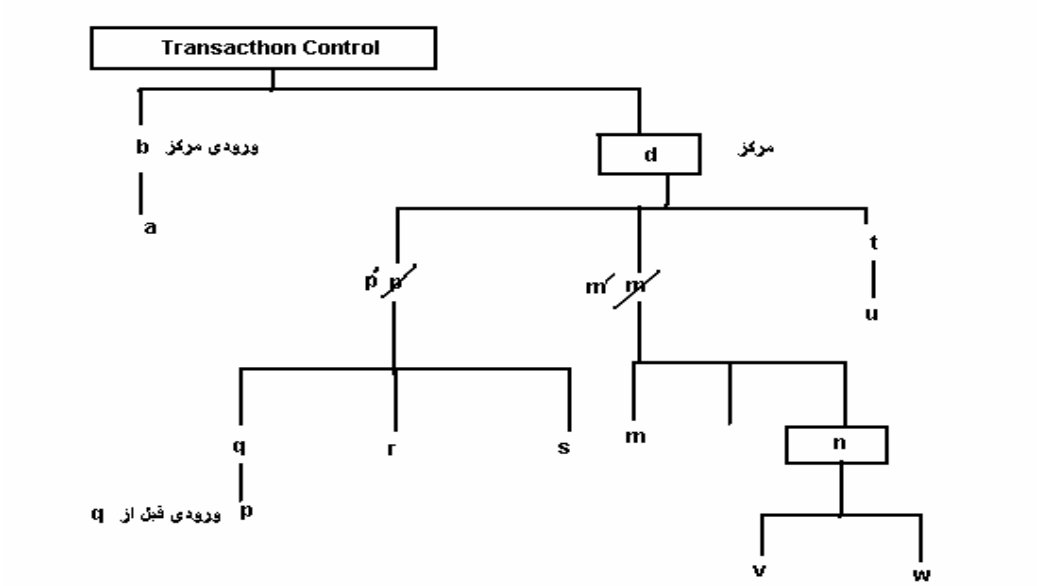
باز نگری حاصل ممکن است نتیجه را تعدیل کند(می‌توان این عمل را در مرحله دیگری مثل مرحله ۷ انجام داد.)
به عنوان مثال می توان بخش"قالب نمایش" و" تولید نمایش" را در یک مرحله با عنوانی جدید مثل "پردازش نمایش اطلاعات انجام داد."...
در این مرحله بازنگری انجام می گیردو نیز پالایش ساختار اولیه نرم افزار به منظور بهبود کیفیت نرم افزار انجام می‌گیرد.
در بازنگری باید اصول و ضوابط طراحی، درک، مهارت، تجربه و قواعد بازنگری .....دخالت داشته باشند.

مثال: معماری متناظر با DFD داده شده را به دست آورید.



شکل ۱۲: مثالی از جریان های تراکنشی و تبدیلی

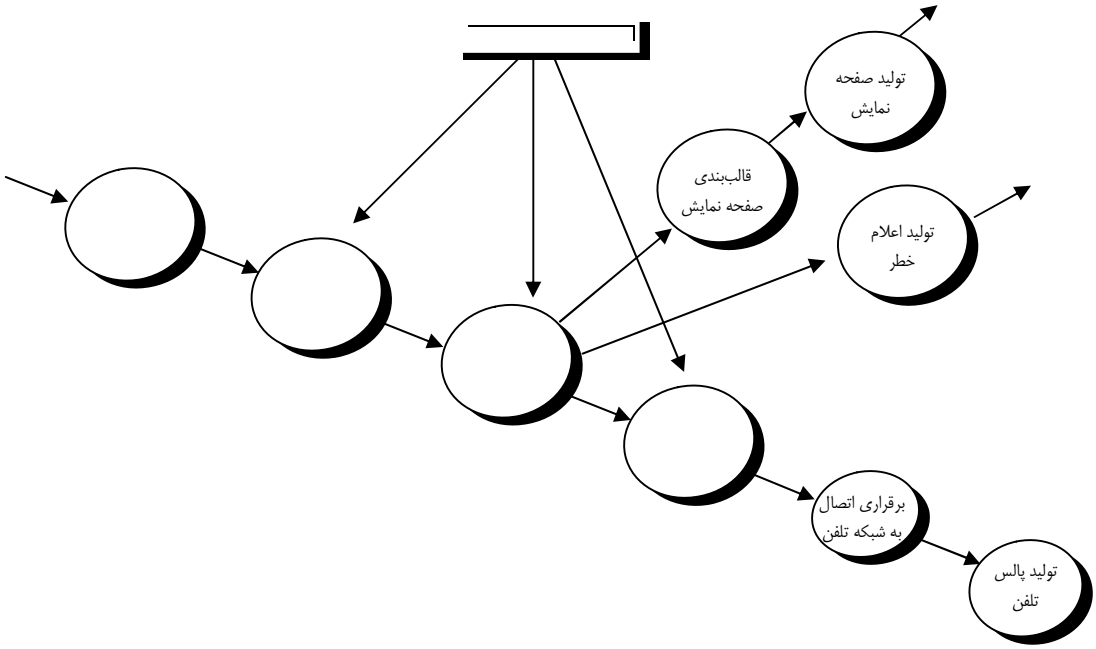
حل: از روی شکل در می‌یابیم که DFD ما تراکنشی است مگر آنکه شرحی روی آن نوشته شده باشد مبنی بر اینکه هر سه خروجی همزمان اتفاق می افتند در این صورت می گوییم جریان تبدیلی است.



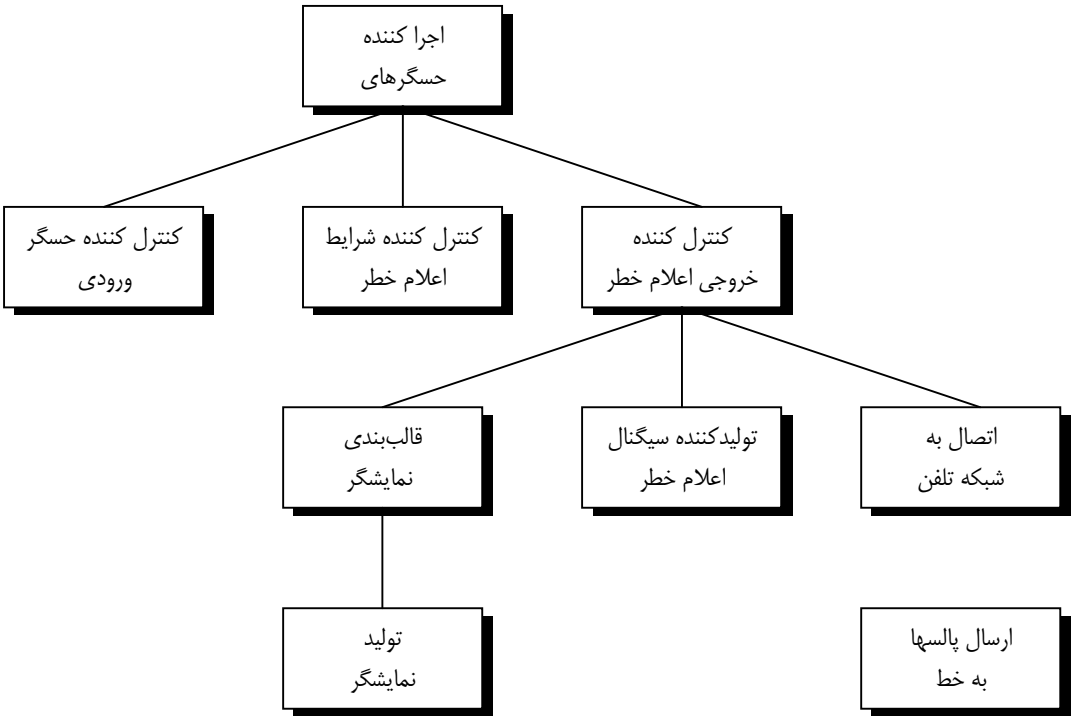
شکل ۱۳: نگاشت جریان‌ها به ساختار برنامه

مثال: معماری برنامه کتناظر با سیستم خانه امن را با استفاده از DFD های داده شده به دست آورید.

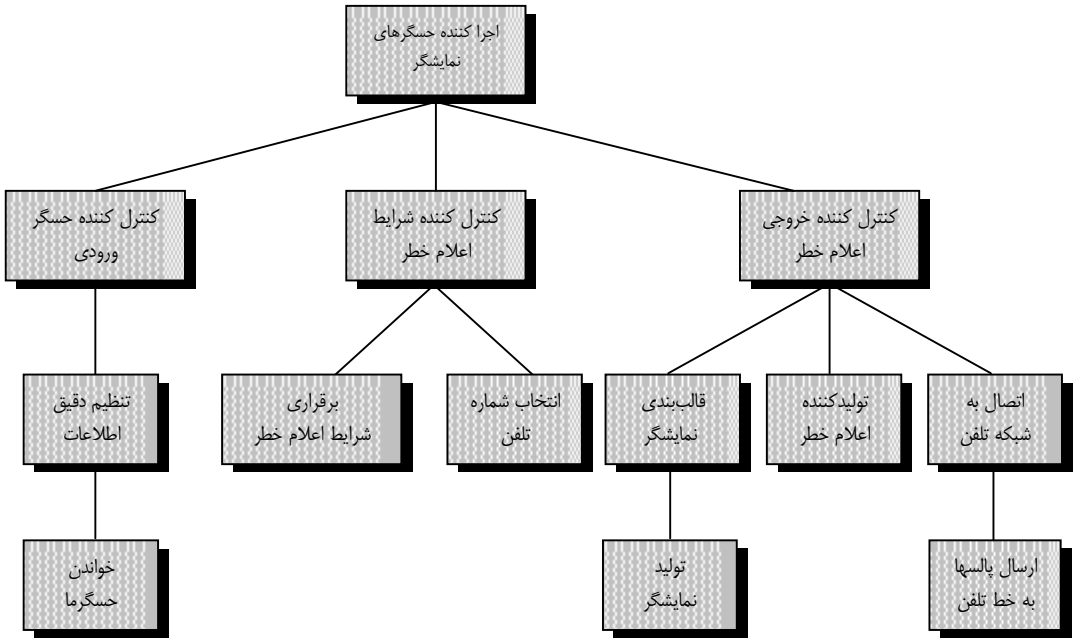
حل: بخشی از جریان داده‌ها سیستم خانه امن در سطح یک در صفحه بعد نشان داده شده است.



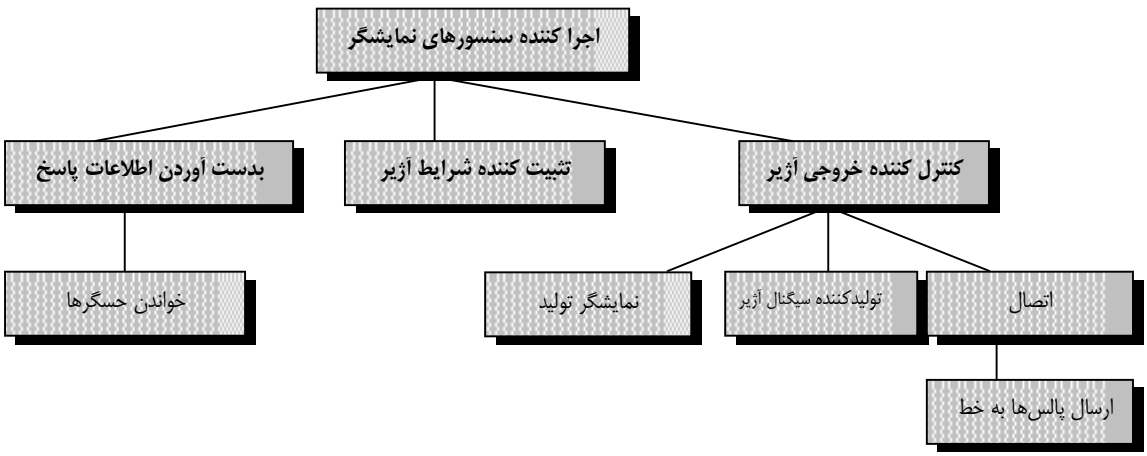
شکل ۱۴: نمودار جریان داده‌های سطح ۱ برای حسگرهای خانه امن



شکل ۱۵: نمودار ساختار برنامه برای حسگرهای نمایش دهنده



شکل ۱۶: ساختار قبل از پالایش برای حسگرهای نمایش دهنده



شکل ۱۷: ساختار پالایش شده برنامه برای حسگرهای نمایش دهنده

استفاده موفق از نگاشت تراکنش یا تبدیل با کارهای اضافی دیگری تکمیل می‌شود که بـه عنـوان بخشـی از طراحـی معماری ضروری هستند. بعد از ساخت ساختار برنامه و اصلاح آن، کارهای زیر باید تکمیل گردد:

- **گزارش از پردازش‌های هر پیمانه ارایه شود.**
- **یک توضیح در مورد واسط برای هر پیمانه داده شود.**
- **ساختارهای داده‌ای محلی و سراسری تعریف شوند.**
- **تمام محدودیت‌های طراحی ذکر شوند.**
- **مجموعه‌ای از بازنگری طراحی‌ها آماده شود.**
- **اصلاح در صورت لزوم در نظر گرفته شود.**

تست‌های فصل چهارده: طراحی معماری

۱- کدامیک از گزینه‌های زیر قسمتی از معماری نرم‌افزار نیست؟
الف) جزئیات الگوریتم ب) پایگاه داده‌ها ج) طراحی داده‌ها د) ساختمان برنامه

۲- طراحی داده در واقع در ضمن تحلیل مدل معماری نرم‌افزار بوجود می‌آید.
الف) درست ب) غلط

۳- معیاری که برای تشخیص کیفیت یک طراحی معماری استفاده می‌شود باید بر مبنای چه ویژگی‌های از سیستم باشد؟
الف) کنترل و داده ب) تابعی بودن ج) جزئیات پیاده‌سازی د) واسطها

۴- یک تکنیک مفید برای ارزشیابی پیچیدگی کلی یک معماری پیشنهادی با نگاه به مولفه‌ها چیست؟
الف) تعداد و اندازه مولفه‌ها ب) جریان وابستگی‌ها و تقسیم وابستگی‌ها
ج) اندازه و هزینه د) هیچکدام

۵- هدف پالایش DFD در ضمن نگاشت آن به معماری نرم‌افزار کوشش برای دستیابی به دایره‌هایی است که وابستگی زیاد را نشان دهند.
الف) درست ب) غلط

۶- وقتی با هر دو جریان تبدیل و تراکنش در یک DFD مواجه می‌شویم آنرا تقسیم‌بندی می‌کنیم و تکنیک نگاشت مناسب را برای هر قسمت از DFD به کار می‌بریم.
الف) درست ب) غلط

۷- در پالایش DFD ضمن نگاشت تراکنش ضروریست که یک PSPEC تا وقتی که تنها یک CSPEC در رابطه با نوع روش معماری نرم‌افزار است، بوجود آوریم.
الف) درست ب) غلط

۸- در نگاشت تراکنش اولین سطح، تولید نتایج در کدامیک از موارد زیر حاصل می‌گردد؟
الف) بوجود آوردن یک CFD ب) بدست آوردن سلسله مراتب کنترل
ج) توزیع ماژولهای کاری د) پالایش دیدگاه ماژول

۹- کدامیک از موارد زیر در ارتباط با انبار داده‌ها درست ولی در مورد پایگاه داده‌ها درست نیست.
الف) مبتنی بر سطح کسب و کار و اندازه پروژه ب) دقت و صحت اطلاعات
ج) یکپارچگی و پایداری د) تمام موارد فوق

۱۰- سبک‌های معماری کدامیک از مؤلفه‌های زیر را در خود جای داده است؟
الف) محدودیت‌ها ب) مجموعه‌ای از مولفه‌ها ج) مدل‌های معنایی (Semantic) د) تمام موارد فوق

۱۱- برای تعیین سبک معماری یا ترکیبی از سبک‌ها که مناسب‌ترین حالت برای سیستم پیشنهادی باشد مهندسی نیازها باید برای کشف کدام یک از موارد زیر انجام گیرد:
الف) پیچیدگی الگو ب) ویژگی‌ها و محدودیت‌ها ج) کنترل و داده د) الگوهای طراحی

۱۲- معیارهای ارزیابی کیفیت طراحی معماری باید مبتنی بر سیستم ..... باشد؟
الف) قابلیت دسترسی و قابلیت اعتماد ب) داده و کنترل ج) قابلیت کارکردها د) جزییات پیاده‌سازی

۱۳-وقتی یک جریان کلی در قسمتی از نمودار جریان داده، خیلی طولانی و پشت سرهم باشد این امر نشان دهنده ..... است.
الف) اتصال پایین ب) پیمانه‌پذیری خوب ج) جریان تراکنشی (Transaction) د) جریان تبدیلی (Transform)

۱۴- وقتی جریان اطلاعات در قسمتی از یک نمودار جریان داده‌ها توسط ویژگی یک مؤلفه تنها شناسایی می‌شود که سایر جریان داده‌ها در طول یک یا چند مسیر است، این امر نشان دهندهٔ ..... است.
الف) اتصال بالا ب) پیمانه‌پذیری ضعیف ج) جریان تراکنشی د) جریان تبدیلی

۱۵- در طی فرآیند نگاشت وقتی DFD مورد پالایش قرار می‌گیرد، هدف آن است که به حباب‌هایی دست یابیم که دارای انسجام (cohesion) بالا هستند.
الف) درست ب) نادرست

فصل ۱۵: طراحی واسط کاربر

طراحی واسط بر سه حوزه موضوع مهم به شرح زیر تأکید دارد:

۱. **طراحی واسط بین اجزای نرم‌افزار،**
۲. **طراحی واسطها بین نرم‌افزار و سایر تولیدکنندگان و مصرف‌کنندگان اطلاعاتی غیر بشری(یعنی سایر اشیاء خارجی) و**
۳. **طراحی واسط بین یک انسان(یعنی کاربر) و کامپیوتر.**

در این فصل صرفاً بر سومین مقولهٔ طراحی واسط ها یعنی طراحی واسط کاربر توجه خواهیم داشت.

قواعد طلایی

Mendal در کتاب خود با عنوان طراحی واسط، [MAN 97] سه "**قانون طلایی**" را به صورت زیر ارایه می کند:

۱- **واگذاری کنترل به کاربر**
۲- **کاهش بار حافظهٔ کاربر**
۳- **سازگار کردن واسط ها**

این قوانین طلایی، عملاً مبنایی برای مجموعه اصول طراحی واسط کاربر هستند که این فعالیت مهم طراحی نرم‌افزاری را هدایت می کنند. پ

واگذاری کنترل به کاربر

در این قانون رعایت نکات زیر الزامی است.

۱- **تعیین شیوه‌های تعاملی به نحوی که کاربر را مجبور به اعمال غیر ضروری یا نامطلوب نکند.**
۲- **ایجاد تعامل انعطاف‌پذیر در ارتباط با کاربر**
۳- **امکان ایجاد وقفه و خنثی‌سازی (بازگشت) در تعامل کاربر.**
۴- **کارآمد ساختن تعامل همراه با پیشرفت سطوح مهارتی و امکان سفارشی کردن آن.** کاربران اغلب در می‌یابند که زنجیرهٔ یکسانی از تعاملات را به طور مکرر انجام می‌دهند. طراحی یک ماکرو، مکانیزمی که به کاربر پیشرفته امکان می‌دهد برای تسهیل تعامل، واسط را سفارشی کند، ارزشمند و مفید است.
۵- **مخفی کردن موارد فنی داخلی از کاربران عادی.** واسط کاربر بایستی او را به درون عالم واقعی برنامهٔ کاربردی سوق دهد. کاربر نباید از سیستم عامل، وظایف مدیریت فایل یا سایر فنآوری‌ها و ... مطلع باشد.
۶- **طراحی تعامل مستقیم با اشیایی که روی صفحهٔ نمایش ظاهر می‌شوند.** کاربر با دستکاری اشیایی که برای انجام یک عمل ضروری هستند، احساس کنترل می‌کند، درست مثل زمانی که آن شیء یک شیء فیزیکی است. مثلاً واسط کاربری که امکان کشیدن یک شیء را (یعنی تغییر مقیاس آن از لحاظ سایر) به کاربر می‌دهد، نمونه‌ای از دستکاری مستقیم است.

کاستن از بار حافظه کاربر

در این قانون نیز رعایت نکات زیر الزامی است.

۱. **کاهش بار در حافظهٔ کوتاه‌مدت.** هنگامی که کاربران درگیر وظایف پیچیده هستند، باید بار حافظهٔ کوتاه مدت را کاهش دهند. این عمل با ایجاد علایم بصری میسر است که به کاربر امکان می‌دهند تا کارهای قبلی را شناسایی کند، نه این که مجبور باشد آنها را به یاد آورد.

۲. **ایجاد پیش گزیده‌های معنی‌دار.** مجموعهٔ آغازین پیش فرض‌ها باید برای کاربر متوسط معنی‌دار باشد، اما کاربر باید بتواند اولویت‌های فردی خود را مشخص کند. هر چند که امکان تنظیم مجدد، بایستی موجود باشد تا تعریف مجدد مقادیر اصلی پیش گزیده امکان‌پذیر گردد.

۴- **تعیین میان‌برهایی که شهودی هستند.** زمانی که برای انجام عملکرد سیستم از مجموعه ای از کلمات استفاده می‌شود، (مثلاً alt- P برای فعال کردن عمل چاپ)، کلمات حفظی باید به شیوه‌ای که به خاطر آوردن آن آسان باشد و به عمل مورد نظر مرتبط گردد. (به عنوان مثال، حرف اول آن عمل، فراخوانده شود.)

۵- **طرح بصری واسط باید براساس استعارهٔ جهان واقعی باشد.** به عنوان مثال سیستم پرداخت فاکتور باید برای هدایت کاربر طی فرایند پرداخت صورت‌حساب، از استعارهٔ دسته چک و ثبت چک استفاده کند. این مساله به کاربر امکان می‌دهد تا به جای حفظ سلسله کارهای غیر متعارف تعاملی، به علایم بصری شناخته شده متوسل شود.

۶- **آشکارسازی اطلاعات به شیوه‌ای تدریجی.** واسط باید به طور سلسله مراتبی سازمان‌دهی شود. یعنی آن که اطلاعات دربارهٔ یک عمل، شی یا یک شیوه باید ابتدا در سطح بالایی از انتزاع ارایه گردد. جزییات بیشتر باید پس از اعلام علاقهٔ کاربر با انتخاب وی از طریق ماوس، در اختیار او قرار گیرد. مثال رایج در بسیاری از برنامه‌های کاربردی واژه‌پردازی، عمل خط زیر است. این کارکرد یکی از چندین کارکردی است که در منوی قالب‌بندی متن قرار دارد. هر چند که تمامی امکانات خط زیر، فهرست نمی‌شوند. کاربر باید ابتدا خط زیر را انتخاب کند و سپس تمامی امکانات خط زیر (خط زیر تک خطی، دو خطی و نقطه‌چین) به نمایش درمی‌آیند.

سازگارسازی واسط

در این قانون نیز رعایت نکات زیر الزامی است.

۱- **قراردادن عمل فعلی در یک بافت معنی‌دار توسط کاربر.** بسیاری از واسط‌ها، لایه‌های پیچیدهٔ تعاملی را با تصاویر زیادی در صفحهٔ نمایش پیاده می‌کنند. تهیه نشان‌گرها (مثل عناوین پنجره، شمایل‌های گرافیکی، کدگذاری ثابت رنگ)، از این نظر که کاربر را قادر می‌سازند تا محیط کاری موجود را بشناسد، اهمیت دارند. به علاوه، کاربر باید بتواند مبدأ خود و امکانات موجود برای گذر به عملی جدید را تعیین کند.

۲- **حفظ ثبات در خانوادهٔ برنامه‌های کاربردی.** مجموعه‌ای از برنامه‌های کاربردی (یا محصولات) باید همگی قوانین یکسان طراحی را پیاده سازی کنند، به نحوی که سازگاری در تمامی تعاملات و محاوره‌ها حفظ شود.

۳- **اگر مدل‌های تعاملی پیشین انتظاراتی را در کاربر به وجود آورده‌اند، تا زمانی که دلیل قانع‌کننده‌ای ندارید از انجام تغییرات خودداری کنید.** پس از تبدیل یک ترتیب خاص تعاملی و یا یک استاندارد عملی (مثل کاربرد alt – S برای ذخیره فایل) از تغییر آن خودداری کنید. زیرا کاربر در مواجهه با هر برنامهٔ کاربردی دیگر همین انتظار را دارد و انجام یک تغییر (مثل کاربرد alt – S برای تغییر مقیاس) سبب آشفتگی و سردرگمی او خواهد شد.

اصول طراحی واسط که در این بخش و قسمت‌های قبلی مورد بحث قرار گرفتند، راهنمای اصلی یک مهندس نرم‌افزار به شمار می‌روند. در قسمت‌های بعدی، فرآیند طراحی واسط را بررسی خواهیم کرد.

طراحی واسط کاربر

فرآیند کلی طراحی واسط کاربر، با ایجاد مدل‌های مختلف کارکرد سیستم (آن طور که از بیرون مشاهده می‌شود) آغاز می‌گردد. سپس وظایف انسانی و کامپیوتری لازم برای تحقق کارکرد سیستم، توصیف می‌شوند، موضوعات طراحی

که در تمام طراحی‌های واسط کاربرد دارند مدنظر قرار می‌گیرند، برای الگوسازی و پیاده‌سازی نهایی مدل طراحی، ابزارهایی به کار می‌روند و نتیجه از لحاظ کیفی ارزیابی می‌گردد.
هنگام طراحی محیط تعامل از طریق تایپ فرمانها به مسایل زیر باید توجه داشت:

- **آیا هر یک از گزینه های منو دارای یک فرمان متناظر هست؟**
- **فرمانها چه شکلی به خود می گیرند؟**
- **فرا گیری و به خاطر سپردن فرمانها چقدر دشوار خواهد بود؟اگر فرمانی فراموش شد ،چه می توان کرد؟**
- **آیا کاربر می تواند فرمانها را به سلیقه خویش مختصر و کوتاه کند؟**

مدل‌های طراحی واسط

به هنگام طراحی یک واسط کاربر، چهار مدل مختلف به کار می‌آیند. مهندس نرم‌افزار **مدل طراحی** را ایجاد می‌کند، مهندس فاکتورهای انسانی(یا مهندس نرم‌افزار) **مدل کاربر** را تعیین می‌کند، کاربر نهایی یک تصویر ذهنی می‌سازد که غالباً **مدل ذهنی کاربر** را به وجود می‌آورند. پیاده کنندگان سی‌ستم نیز یک **تصویر سیستم** ایجاد می کنند. متأسفانه هر یک از این مدل‌ها ممکن است تفاوت قابل ملاحظه‌ای با یکدیگر داشته باشند. نقش طراح واسط، رفع اختلافات و به دست دادن یک نمایش منسجم و سازگار از واسط است.
مدل طراحی کل سیستم، تلفیقی از نمایش داده‌ها، معماری، واسط و بازنمایی رویه‌ای نرم‌افزار می‌باشد. تعیین نیازها ممکن است محدودیت‌های خاصی را مطرح کند که به تعیین کاربر سیستم کمک می‌کنند. اما طراحی واسط، اغلب تنها لازمهٔ مدل طراحی است.

مدل کاربر، نمایی از کاربران نهایی سیستم را ترسیم می‌کند. برای ساخت یک رابط کار مؤثر، " تمام کار طراحی باید با درک درستی از کاربران مورد نظر، از جمله مشخصات سن، جنسیت، توانایی‌های جسمی، سابقهٔ تحصیلی، فرهنگی یا قومی، انگیزه، اهداف و شخصیت آنها، آغاز گردد. به علاوه، کاربران را می‌توان در گروه‌های زیر طبقه‌بندی کرد:

۱- **کاربران مبتدی.** از دانش نحوی سیستم برخوردار نیستند و دانش معنایی آنها از برنامهٔ کاربردی یا کاربرد کامپیوتر به طور کلی، اندک است.
۲- **کاربران مطلع و دوره‌ای.** دانش معنایی معقول از برنامهٔ کاربردی دارند اما به یادآوری نسبتاً کم دانش نحوی لازم برای کاربرد رابط دارند.
۳- **کاربران مطلع و دایمی.** دانش نحوی و معنایی مناسب دارند که اغلب به " مشخصهٔ کاربر ماهر" منجر می‌شود، یعنی کاربرانی که به دنبال میان‌برها و حالت‌های اختصاری تعامل هستند.

ادراک سیستم (مدل ذهنی کاربر). تصویری از سیستم است که کاربر نهایی در ذهن خود ایجاد می‌کند. مثلاً اگر از کاربر یک واژه‌پرداز پرداز خاص بخواهیم تا کارکرد آن را توصیف کند، پاسخ او براساس درک او از سیستم خواهد بود. صحت توصیف بستگی به شرح حال کاربر (به عنوان مثال جواب کاربران مبتدی در نهایت مجمل و ناقص است) و آشنایی کلی با نرم‌افزار در حیطهٔ برنامهٔ کاربردی دارد.

کاربری که درک کاملی از واژه‌پردازها دارد. اما تنها با واژه‌پرداز خاصی کار کرده است، در عمل نسبت به تازه‌کاری که هفته‌ها وقت صرف یادگیری سیستم نموده ممکن است توصیف کامل‌تر و جامع‌تری را ارایه دهد.

تصویر سیستم، ترکیبی از نمود بیرونی سیستم کامپیوتری (یعنی ظاهر و عملکرد رابط) به همراه تمامی اطلاعات پشتیبان (کتاب‌ها، دستنویس‌ها، نوارهای ویدیویی و فایل‌ها راهنما) است که نحو و معنا شناسایی سیستم را توصیف می‌کنند. زمانی که تصویر سیستم و درک سیستم یکسان باشند، عموماً کاربران با نرم‌افزار احساس راحتی نموده و به طور مؤثر آن را به کار می‌برند. به منظور ادغام مدل‌ها، مدل طراحی باید سازگاری با اطلاعات موجود در مدل کاربر توسعه یافته باشد و تصویر سیستم اطلاعات نحوی و معنایی دربارهٔ واسط را دقیقاً منعکس کند.

چند مسأله طراحی راهنگام پرداختن به تسهیلات راهنما باید در نظر داشت:

- **آیا راهنما برای کلیه عملکردهای سیستم ودرهمه اوقات تعامل با سیستم در دسترس خواهد بود؟**
- **کاربرچگونه درخواست کمک وراهنمایی می کند؟**
- **راهنماچگونه ارایه خواهد شد؟**
- **کاربر چگونه به تعاملهای عادی باز می گردد؟**
- **اطلاعات راهنما چگونه ساختاری دارند؟**

فرایند طراحی واسط کاربر

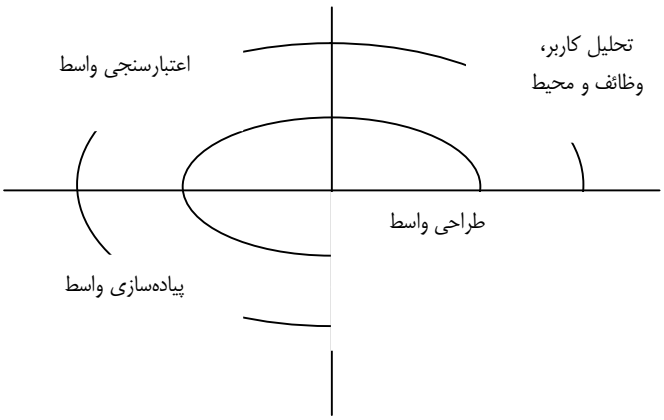
فرآیند طراحی واسط‌های کاربر، تکراری است و با استفاده از مدل حلزونی، مشابه آن چه در فصل ۲ مـورد بحـث قـرار گرفت، قابل ارایه است. با مراجعه به شکل ۱، روند طراحی واسط کاربر، چهار فعالیت مجزای ساختاری را دربردارد.

۱- **تحلیل و الگوسازی کاربر، وظیفه و محیط و مدل‌سازی**

۲- **طراحی واسط**

۳- **ساخت رابط**

۴- **اعتبارسنجی واسط**



شکل ۱: فرایند طراحی واسط کاربری

سیستم توسعه واسط کاربر(UIDS)با استفاده از مؤلفه های نرم افزاری،راهکاری برای موارد زیر فراهم می آورد:

- **مدیریت دستگاه های ورودی(مثل موس یا صفحه کلید)**
- **اعتبار سنجی ورودی کاربر**
- **کنترل خطاو نمایش پیامهای خطا**
- **فراهم آوردن باز خود(مثل بازتاب ورودی خودکار)**
- **فراهم آوردن راهنما و پیام**
- **کنترل پنجره هاو فیلدها،حرکت در داخل پنجره ها**
- **برقراری ارتباط میان نرم افزار کاربردی و واسط**
- **جدا کردن برنامه کاربردی از عملکردهای مدیریت واسط**
- **مجاز کردن کاربر به سفارشی کردن واسط**

بنابراین در طراحی واسط کاربری باید مراحل زیر طی گردد:

- **تعیین اهداف و مقاصد هر وظیفه**
- **نگاشت هر هدف/نیت در تعدادی عملیات مشخص**
- **مشخص کردن دنباله عملیاتی وظایف اصلی وفرعی ، که چون در سطح واسط ا جرا می شوند، سناریوی کاربر نیز خوانده می شوند**
- **نشان دادن حالت سیستم**
- **تعریف راهکارهای کنترلی**
- **نشان دادن چگونگی تاثیرپذیرفتن حالت سیستم از راهکارهای کنترلی**
- **نشان دادن چگونگی تفسیر حالت سیستم توسط کاربر با استفاده از اطلاعات به دست آمده از طریق واسط**

مسایل طراحی واسط ها

در حین تکمیل طراحی واسط کاربر، چهار مساله معمول طراحی تقریباً همیشه سطحی تلقی می‌شوند:

زمان پاسخ‌گویی سیستم، تسهیلات کمکی کاربر، خطاگردانی اطلاعات و برچسب‌گذاری فرمان.

متأسفانه، بسیاری از طراحان کمی دیر این مسایل را در فرآیند طراحی مدنظر قرار می‌دهند (گاهی قبل از در دسترس قرار گرفتن یک نمونهٔ عملی، اشارهٔ مختصر به یک مشکل صورت نمی‌گیرد.) تکرار غیر ضروری، تأخیرهای پروژه و نارضایتی مشتری، اغلب از پیامدهای حاصله است. پس بهتر آن است که هر یک از مسایل در آغاز طراحی نرم‌افزار و به هنگام انجام راحت تغییرات و پایین بودن هزینه‌ها، مورد توجه قرار گیرند.

زمان پاسخ‌گویی سیستم از زمانی که کاربر عمل کنترلی را انجام می‌دهد (مثلاً کلید بازگشت را زده یا روی ماوس کلیک می‌کند) تا زمان پاسخ‌گویی نرم‌افزار با اقدام یا خروجی مطلوب، اندازه‌گیری می‌شود.

زمان پاسخ‌گویی سیستم دو ویژگی مهم دارد: **طول و تغییرپذیری**. اگر **طول** پاسخ‌گویی سیستم بسیار طولانی باشد، ناامیدی و فشار روی کاربر، نتیجه‌ای اجتناب‌ناپذیر است. هر چند که اگر واسط، باعث دستپاچگی کاربر شود، زمان پاسخ‌گویی بسیار کوتاه نیز می‌تواند مضر باشد. پاسخ‌گویی سریع ممکن است موجب عجلهٔ کاربر و بنابراین انجام اشتباه از سوی او شود.

تغییرپذیری به انحراف از زمان میانگین پاسخ‌گویی اشاره داشته و از خیلی جهات، مهم‌ترین مشخصهٔ زمان پاسخ‌گویی به شمار می‌رود. تغییرپذیری کم، حتی در صورت طولانی بودن زمان پاسخ‌گویی، به کاربر امکان می‌دهد تا به طریق مناسب تعامل را برقرار کند. به عنوان مثال، پاسخ یک ثانیه‌ای به یک فرمان، به پاسخ‌گویی متغیر بین یک تا دو و نیم ثانیه، ترجیح دارد. کاربر همیشه بلاتکلیف است و نمی‌تواند که در پشت صحنه چه خبر است و چه اتفاقی افتاده است.

تقریباً تمامی کاربران یک سیستم تعاملی کامپیوتری، گهگاه به کمک نیاز دارند. دو نوع امکانات کمکی عبارتند از: **یکپارچه و افزودنی**، [RUB 88] **تسهیلات کمکی یکپارچه** از آغاز در داخل نرم‌افزار طراحی می‌شود. این نوع کمک به کاربر غالباً حساس به متن است و کاربر را قادر می‌سازد تا از میان موضوعات مرتبط با اعمال در حال اجرا، اقدام به انتخاب کند. واضح است که این امر، زمان لازم برای دریافت کمک توسط کاربر را کاهش داده و کاربرپسندی واسط را افزایش می‌دهد. **تسهیلات کمکی افزودنی**. پس از ساخت سیستم به نرم‌افزار افزوده می‌شود. ممکن است کاربر مجبور شود برای یافتن راهنمایی صحیح و مناسب، فهرستی با صدها موضوع را جستجو کند و اغلب با شروع نادرست، اطلاعات غیرمرتبط را دریافت نماید. شکی نیست که امکانات کمکی **یکپارچه** بر نوع **افزودنی** آن برتری دارد.

هنگام بروز خطا، پیغام‌های خطا و هشدارها. " اخبار بدی" است که به کاربران سیستم‌های تعاملی ارایه می‌گردد. در بدترین حالت، پیغام‌های خطا و اخطارها، اطلاعات بی‌فایده یا گمراه کننده را منتقل کرده و تنها باعث تشدید ناکامی کاربر می‌شوند. تعداد کاربران کامپیوتر که با خطایی از نوع زیر مواجه نشده باشند، بسیار اندک است.

فرمان تایپ شده زمانی، رایج‌ترین شیوهٔ محاوره بین کاربر و نرم‌افزار سیستم بود و در هر نوع برنامهٔ کاربردی به طور معمول به کار می‌رفت. امروزه، استفاده از رابطهای پنجره‌ای و اشاره و انتخاب، کاربرد فرامین تایپ شده را کاهش داده، اما بسیاری از کاربران ماهر هم‌چنان شیوهٔ ارتباطی مجهز به فرمان را ترجیح می‌دهند. به هنگام انتخاب فرامین تایپ شده به عنوان نوعی شیوهٔ محاوره، برخی مسایل طراحی مورد توجه قرار می‌گیرند:

- آیا هر یک از انتخاب‌های منو یک فرمان مرتبط خواهد داشت؟
- فرامین به چه شکلی خواهند بود؟ امکانات موجود عبارتند از: توالی کنترل (مثل Alt + P)؛ کلیدهای تابعی، واژهٔ تایپ شده.
- یادگیری و به خاطر سپردن فرامین تا چه حد دشوار است؟ در صورت فراموشی یک فرمان، چه می‌توان کرد؟
- آیا می‌توان فرامین را توسط کاربر سفارشی یا اختصاری کرد؟

تحلیل محیط کاربر بر محیط کارفیزیکی تکیه دارد- از جمله پرسشهایی که باید مطرح شوند، عبارتند از:

- **واسط از نظر فیزیکی در کجا قرار داده خواهد شد؟**
- **آیا کاربر به حالت نشسته کار می کند یا ایستاده،یا کارهای دیگری انجام می دهد که با واسط بی ارتباط هستند؟**
- **آیا سخت افزار واسط محدودیتهای جا،نور و سرو صدا را در بر می گیرد؟**

به طور کلی هر پیام خطا یا هشدار تولید شده توسط یک سیستم محاوره ای باید دارای ویژگیهای زیر باشد:

- **پیام باید مشکل را به زبانی شرح دهد که کار بر قادربه درک آن باشد**
- **پیام باید حاوی یک توصیه سازنده برای رهایی از وضعیت خطا باشد**
- **پیام باید هر گونه تبعات منفی خطا (مثلا فایلهای داده ای مخدوش شده)راخاطر نشان کند تا کاربر بتواند آنهاراچک کند**
- **پیام باید با یک نشانه سمعی یا بصری همراه باشد**
- **پیام باید قضاوت گونه باشد**

ارزیابی طراحی

پس از ایجاد الگوی عملی واسط کاربر، این مدل باید مورد ارزیابی قرار گیرد تا معلوم شود که آیا نیازهای کاربر را برطرف می‌کند یا خیر؟ ارزیابی می‌تواند در یک طیف رسمی صورت گیرد که گسترهٔ آن با انجام آزمونی غیر رسمی که ضمن آن کاربر بازتابی بدون فکر قبلی دارد شروع شده و به مطالعهٔ رسمی ختم می‌شود که از روش‌های آماری برای ارزیابی پرسش‌نامه‌های تکمیل شده توسط کاربران نهایی، استفاده می‌کند. چرخهٔ ارزیابی واسط کاربر مشابه شکل ۲ می باشد.

برخی معیارهای ارزیابی [MOR81] را هنگام بررسی‌های اولیه طراحی، می‌توان اعمال کرد:

۱- **طول و پیچیدگی مشخصات مکتوب سیستم و واسط آن**، بیانگر میزان یادگیری لازم توسط کاربران سیستم می‌باشد.

۲- **تعداد وظایف تعیین شدهٔ کاربر و میانگین اعمال در هر کار**، نشان دهندهٔ زمان محاوره و کارآیی کلی سیستم است.

۳- **تعداد اعمال، وظایف و وضعیت‌های سیستم که در مدل طراحی تعیین شده**، به بار حافظهٔ کاربران سیستم دلالت دارد.

۴- **روش ارتباط واسط، امکانات کمکی و خطاگردانی**، در کل بیانگر پیچیدگی واسط و میزان پذیرش از سوی کاربر می‌باشد.

پس از ساخته شدن اولین مدل، طراحی می‌تواند مجموعه‌ای از داده‌های کیفی و کمی را که به ارزیابی واسط کمک خواهند کرد، را جمع‌آوری می‌کند. ویژگی های پرسشنامه هایی که می توان بین کاربران توزیع کرد می تواند شامل موارد زیر باشد:

۱. پاسخ های ساده آری/ خیر داشته باشد.
۲. پاسخ عددی داشته باشد.
۳. پاسخ مقیاسی داشته باشد.
۴. پاسخ درصدی داشته باشد.

سوالات می توانند به صورت زیر باشند:

۱. آیا نمادها(ICONS) خود توصیف هستند؟ اگر نیستند کدام نماد نیاز به توصیف دارد ؟
۲. آیا عملیات را به آسانی می توان به خاطر سپرد و اجرا کرد؟
۳. تا کنون از چند عمل متفاوت استفاده کرده اید؟
۴. سهولت فراگیری عملیات اصلی سی‌ستم تا چه حدی است؟ (بین ۱ تا ۵)
۵. در مقایسه با واسط های دیگری که استفاده کرده اید، به این واسط چه امتیازی می دهید؟

پس از جمع آوری اطلاعات مربوطه باید نسبت به تحلیل و جمع بندی آن اقدام نمود.

تست های فصل ۱۵ : طراحی واسط ها

۱. **سه قاعده طلایی مندل(mandel) در طراحی واسط‌ها شامل ۱- سپردن کنترل به کاربر ۲- کاستن از بار حافظه کاربر و سازگار ساختن واسط‌ها است.**
الف) درست ب) غلط

۲. **واسط‌های خارجی در تعامل با موجودیت‌های بیرونی سیستم در مدل تحلیل با شناسایی داده و کنترل مورد نیاز طراحی می‌گردد.**
الف) درست ب) غلط

۳. **در طراحی واسط‌های کاربر باید نحوه پیاده‌سازی واسط، محیط(فنآوری) مورد استفاده و عوامل حساس کاربر را درنظر گرفت.**
الف) درست ب) غلط

۴. **کاربر در زمان کار با سیستم، زمان پاسخگویی سیستم را مورد ارزیابی قرار می‌دهد. معیارهای قابل اندازه‌گیری برای این کار چیست؟**
الف) میانگین طول زمان پاسخ ب) طول زمان پاسخ ج) طول زمان پاسخ و تغییرپذیری د) انحرافات زمان پاسخ

۵. **واسط‌های کاربر باید با توجه به فرهنگ و دانش عمومی کاربران تهیه گردد.**
الف) درست ب) غلط

۶. **کدام یک از اصول طراحی واسط زیر اجازه نمی‌دهد که کاربر در تعامل کنترلی با کامپیوتر قرار داشته باشد.**
الف) تعاملی که بتواند اجرای برنامه را قطع نماید.
ب) تعاملی که بتواند عملیات را بازگردانی نماید.
ج) پنهان کردن امور فنی داخلی از دید کاربر
د) فقط شامل یک روش تعریف دشوار و سخت برای تکمیل یک وظیفه

۷. **کدام یک از اصول طراحی واسط زیر، بار حافظه کاربر را کاهش می‌دهد.**
الف) تعریف Shortcut های مستقیم
ب) فاش کردن اطلاعات به شیوه‌ای فزاینده
ج) استقرار پیش‌فرض‌های با معنی
د) تمام موارد فوق

۸. **دلیل کاستن از بار حافظهٔ کاربر آن است که وی بتواند در تعامل با کامپیوتر سریعتر کار را تمام کند.**
الف) درست ب) نادرست

۹. **سازگاری واسط‌ها منجر به**
الف) مکانیزم‌های ورودی در تمام نرم‌افزار کاربردی یکسان باقی می‌ماند.
ب) هر برنامهٔ کاربردی باید نگاه و احساس منحصر به خود را داشته باشد.
ج) روش‌های راهبردی حساس به محتوا هستند.
د) موارد الف و ب

۱۰. **اگر مدلهای تعاملی گذشته باری توقعات یک کاربر خاص ساخته شده باشند به طور کلی تغییر مدل‌ها ایدهٔ خوبی نیست.**
الف) درست ب) نادرست

۱۱. **کدام یک از مدل‌های زیر سودمندی کاربر نهایی یک سیستم کامپیوتری را شرح می‌دهد؟**
الف) مدل طراحی ب) مدل کاربری ج) مدل مربوط به کاربر(برداشت سیستمی د) تصویر سیستمی

۱۲. **کدام یک از مدل‌های زیر تصویری از یک سیستم را ارایه می‌دهد که کاربر نهایی در ذهن خود ایجاد کرده است؟**
الف) مدل طراحی ب) مدل کاربری ج) برداشت سیستمی د) تصویر سیستمی

۱۳. کدام یک ازمدل‌های زیر نگاه و نگرشی به واسط کاربر با پشتیبانی تمام اطلاعات را دارد؟

الف) مدل کاربری ب) مدل کاربر ج) برداشت سیستمی د) تصویر سیستمی

۱۴. کدام یک از فعالیت‌های چارچوبه‌ای(پشتیبانی) زیر به طور طبیعی در فرآیند طراحی واسط مورد استفاده قرار نمی‌گیرند:

الف) برآورد هزینه ب) ساخت واسط ج) اعتبارسنجی واسط د) تحلیل کاربر وظایف

۱۵. کدام نگرش بر تحلیل وظایف کاربر در طراحی واسط کاربر مفید است؟

الف) داشتن کاربرانی که ترجیحات خود را توسط پرسشنامه بیان کرده‌اند.

ب) تکیه بر قضاوت برنامه‌نویسان با تجربه

ج) مطالعه سیستم‌های مکانیزه شده مرتبط

د) مشاهدهٔ نحوهٔ انجام کار دستی کاربران

فصل۱۶: طراحی در سطح مولفه

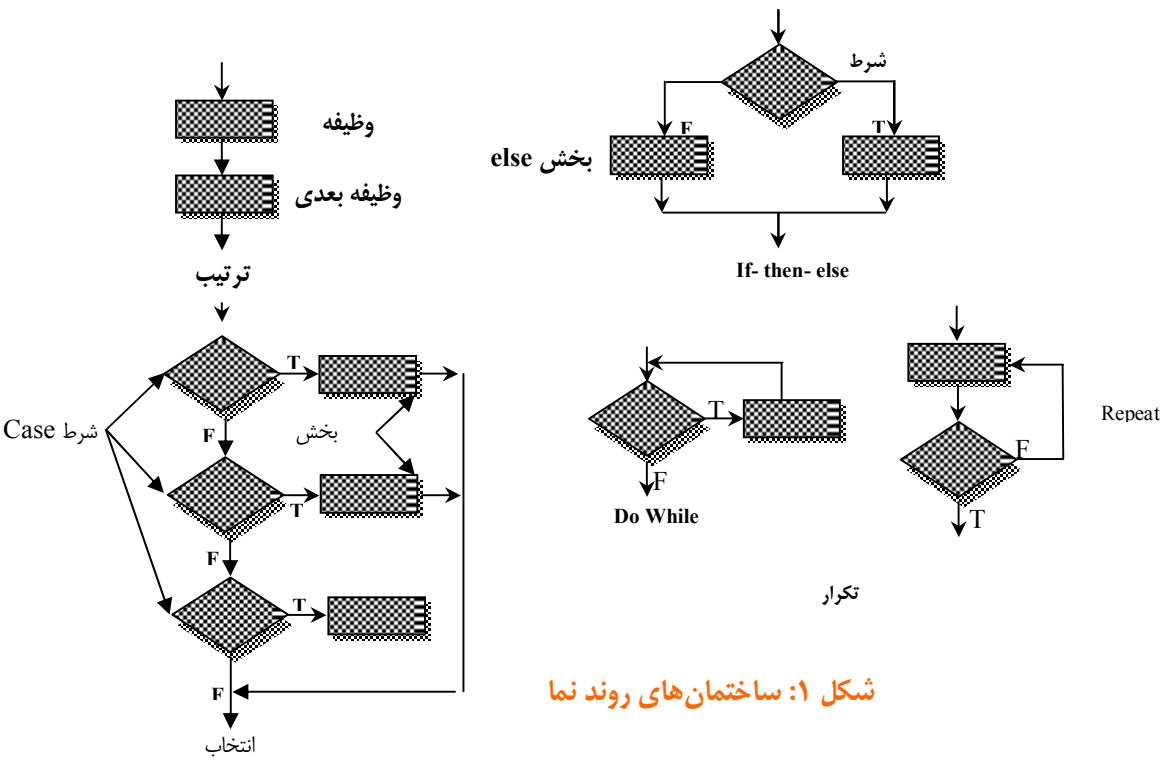
طراحی در سطح مؤلفه که «**طراحی رویه‌ای**» نیز نام دارد، پس از تعیین طراحی داده‌ها، معماری و واسط انجام می‌گیرد و هدف تبدیل مدل طراحی به نرم‌افزار عملیاتی می‌باشد. اما سطح انتزاعی مدل طراحی موجود، نسبتاً زیاد بوده و میزان انتزاع برنامهٔ عملیاتی کم است. تبدیل و برگردان، سخت و دشوار بوده و موجب ایجاد خطاهای ظریفی می‌شود که یافتن و تصحیح آنها در مراحل بعدی فرآیند نرم افزاری دشوار است.
نمایش طراحی لایه‌ای با استفاده از یک زبان برنامه‌سازی امکان‌پذیر است. اساساً، برنامه با به کارگیری مدل طراحی به عنوان یک راهنما، ایجاد می‌گردد. روش دیگر نمایش طراحی رویه‌ای با استفاده از نمایش واسطه (مثلاً گرافیکی، جدولی یا متنی است) که به راحتی قابل تبدیل به برنامهٔ منبع می‌باشد. صرف‌نظر از مکانیزم به کار رفته در نمایش طراحی رویه‌ای، ساختمان‌های داده‌ای، واسط‌های و الگوریتم‌های مشخص شده باید با مجموعه‌ای از رهنمودهای طراحی مطابقت داشته باشند تا ضمن پیشرفت و تکمیل طراحی رویه‌ای، مانع خطا گردند.

برنامه سازی ساختیافته

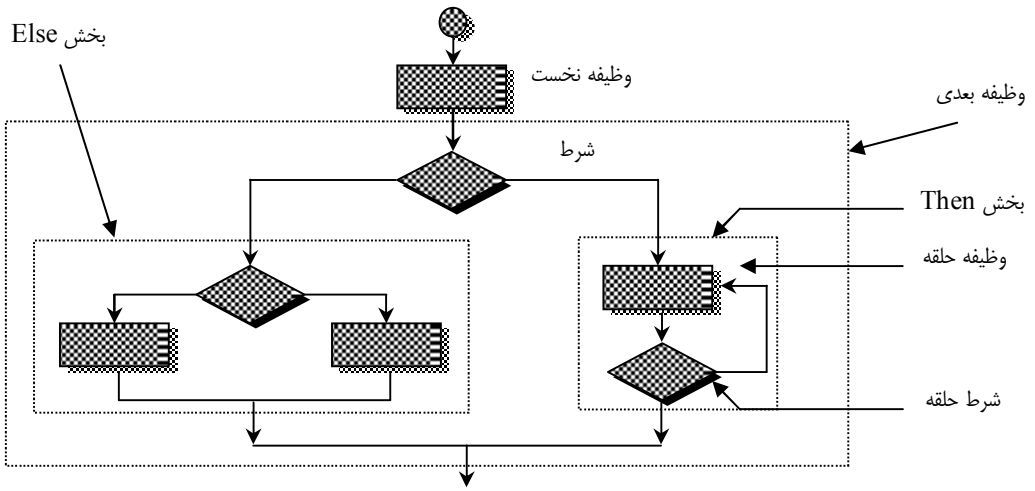
استفاده از ساختارهای منطقی پیشنهاد محققین می باشد که هر برنامه ای را با آن ها می توان نوشت. این ساختارها عبارتند از **توالی، شرط و تکرار**. **توالی**، مراحل پردازشی ضروری در تعیین هر الگوریتم را اجرا می‌کند. **شرط**، امکان پردازش انتخابی بر اساس رخداد منطقی را فراهم می‌کند و **تکرار**، ایجاد حلقه را امکان‌پذیر می‌سازد. این سه سازه در برنامه‌سازی ساختیافته که یک تکنیک مهم طراحی رویه می‌باشد، اساسی و ضروری به شمار می‌روند.

نشانه گذاری طراحی گرافیکی

هیچ تردیدی نیست که ابزارهای گرافیکی مانند روند نما(Flowchart) یا نمودارهای مستطیلی، الگوهای تصویری مفیدی هستند که به سادگی جزییات رویه‌ای را مصور می‌سازند. هر چند در صورت کاربرد غلط ابزارهای گرافیکی، تصویر اشتباه ممکن است به نرم‌افزاری نادرست منجر گردد.

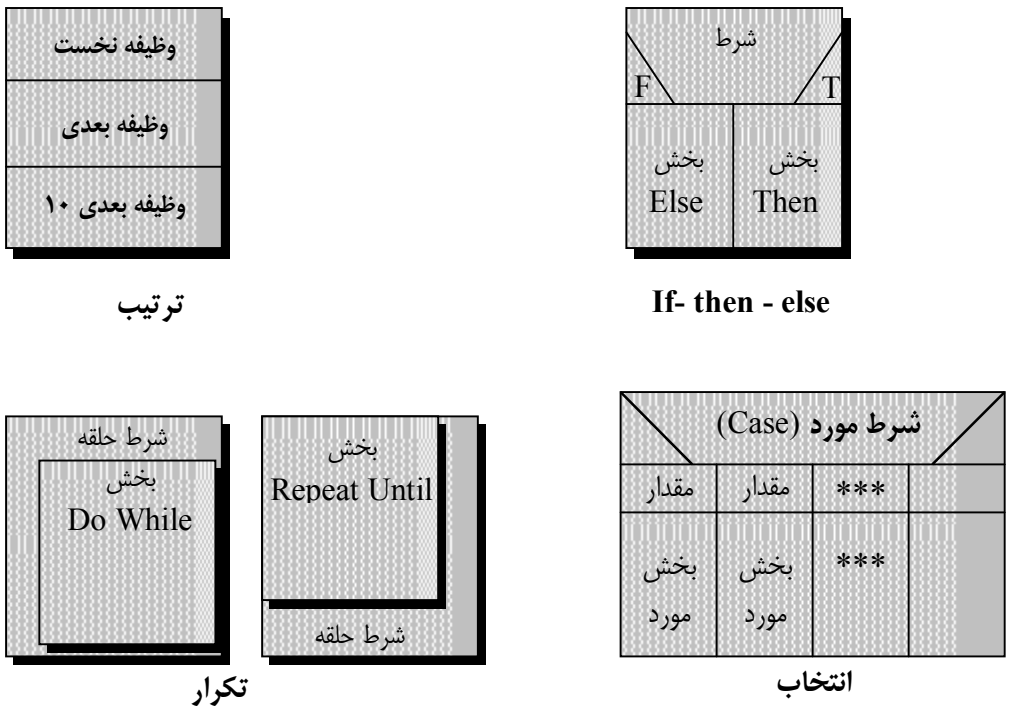


شکل ۱: ساختمان‌های روند نما



شکل ۲: ساختمانهای تو در تو

نمایش گرافیکی سازه‌های ساختیافته با استفاده از نمودار مستطیلی(یا جعبه‌ای) در شکل ۳ نشان داده شده است. عنصر اصلی نمودار یک کادر یا جعبه است. برای نمایش توالی، دو مستطیل از پایین به بالا به یکدیگر متصل می‌شوند. برای نمایش if-Then- else ، مستطیل else- Part, Then- Part به دنبال Condition قرار می‌گیرد. تکرار با یک الگوی محدودکننده که فرآیند قابل تکرار را محصور می‌کند. (Repeat-Until–Part DO–Wile-Part) به نمایش در می‌آید. در نهایت، انتخاب با به کارگیری فرم گرافیکی نشان داده شده در پایین تصویر، به نمایش در می‌آید.
نمودار کادری نیز مانند فلوچارت‌ها به هنگام پالایش عناصر پردازشی یک پیمانه، در چندین صفحه لایه‌بندی می‌شود. یک فراخوانی پیمانه توسط کادری در داخل پیمانه که نام آن درون یک بیضی محصور شده، قابل نمایش است.



شکل ۳ : ساختمان های نمودار جعبه‌ای

علایم طراحی جدولی

در بسیاری از کاربردهای نرم‌افزاری، به منظور ارزیابی ترکیب پیچیدهٔ شرایط و انتخاب اعمال مناسب مبتنی بر این حالت‌ها، ممکن است به پیمانه ای نیاز باشد. جداول تصمیم‌گیری، علایمی را به کار می‌برد که اعمال و شرایط (توصیف شده در شرح پردازش) را به شکل جدولی تبدیل می‌کند. تعبیر غلط جدول، دشوار است و حتی ممکن است به عنوان ورودی برای یک الگوریتم قابل خواندن ولی برای یک دستگاه قابل خواندن نباشد.

سازمان‌دهی جدول تصمیم‌گیری در شکل۴ به نمایش درآمده است. مطابق شکل، این جدول به چهار بخش تقسیم می‌شود. یک چهارم سمت چپ بالایی شامل لیستی از تمامی شرایط است. یک چهارم سمت چپ پایینی لیستی ازتمامی اعمال و اقدامات ممکن و مبتنی به ترکیب شرایط را در بردارد. یک چهارم‌های سمت راست، ماتریسی را تشکیل می‌دهند که نشان دهندهٔ ترکیب شرایط و اعمال و اقدامات متناظر با یک ترکیب خاص هستند. هر ستون ماتریس ممکن است به عنوان یک قانون پردازشی تفسیر شود.

در ایجاد جدول تصمیم‌گیری، مراحل زیر طی می‌شود:

۱- تمامی اعمال مرتبط با یک رویه (یا پیمانه) خاص را فهرست کنید.

۲- تمامی شرایط (یا تصمیمات اتخاذ شده) طی اجرای رویه را لیست نمایید.

۳- مجموعه خاص شرایط را با اعمال و اقدامات بخصوصی که ترکیبات غیر ممکن شرایط را برطرف می‌سازند، مرتبط نمایید یا این که هرگونه جابه‌جایی ممکن در شرایط ایجاد کنید.

۴- با بیان اعمال انجام شده در ارتباط با مجموعه شرایط، قوانین را تعریف کنید.

شرایط	۱	۲	۳	۴					n
شرط شماره ۱	✓			✓	✓				
شرط شماره ۲		✓		✓					
شرط شماره ۳			✓		✓				
شرط شماره ۴									
شرط شماره ۵									
اقدام شماره ۱	✓			✓	✓				
اقدام شماره ۲		✓		✓					
اقدام شماره ۳			✓						
اقدام شماره ۴			✓	✓	✓				
اقدام شماره ۵	✓	✓			✓				

شکل ۴: ساختار جدول تصمیم‌گیری

نمونه ای دیگر از جدول تصمیم گیری

شرایط	۱	۲	۳	۴	۵	
نرخ ثابت حساب	T	T	F	F	F	
نرخ متغیر حساب	F	F	T	T	F	
مصرف > کیلو وات ساعت ۱۰۰	T	F	T	F		
مصرف >= ۱۰۰ کیلو وات ساعت	F	T	F	T		
اقدامات						
حداقل هزینه ماهانه						
صورت حساب زمانبندی الف						
صورت حساب زمانبندی ب						
دیگر اقدامات						

شکل ۵: ساختار جدول تصمیم‌گیری در مورد مصرف برق

زبان طراحی برنامه (PDL)

الف) " زبان طراحی برنامه (PDF) که انگلیسی ساخت‌یافته یا شبه کد نیز نام دارد " یک زبان آمیخته است به طوری که واژگان یک زبان (یعنی انگلیسی) و نحوه کلی زبانی دیگر (یعنی زبان برنامه‌سازی ساختیافته) را به کار می‌برد.

ب) یک زبان طراحی باید ویژگیهای زیر را داشته باشد:

- یک نحو ثابت از کلمات کلیدی که تمام سازه‌های ساخت‌یافته، تعاریف داده‌ها و ویژگیهای پیمانه‌ای شدن را تهیه کند.
- یک نحو آزاد از زبان طبیعی که ویژگیهای پردازشی را تشریح کند.
- تسهیلات تعریف داده‌ها که باید مشتمل بر ساختارهای داده‌ای ساده (اسکالر، آرایه) و پیچیده (لیست پیوندی یا درخت) باشد.
- تعریف زیر برنامه و فنون فراخوانی که حالات مختلف توصیف رابط را پشتیبانی کند.

```
PROCEDURE security.monitor;
INTERFACE RETURNS system.status;
TYPE signal 1S STRUCTURE DEFINED
name 1S STRING LENGTH VAR;
address 1S HEX device location;
bound. Value IS upper bound SCALAR;
message IS STRING LENGTH VAR;
      END signal TYPE;
      TYPE system.status IS BIT (4);
        TYPE alarm. Type DEFINED
```

```
Smoke. Alarm IS INSTANCE OF signal;
Fire.alarm IS INSTANCE OF signal;
Water.alarm IS INSTANCE OF signal;
temp.alarm IS INSTANCE OF signal;
burglar.alarm IS INSTANCE OF signal;
    TYPE phone. Number IS area code + 7-digit number;
```

ج) نشان‌گذاری طراحی باید به نمایش رویه‌ای منجر شود که درک و بررسی آن آسان باشد. به علاوه، نشان‌گذاری باید توانایی " code to " را تقویت کند به طوری که برنامه یا کد در واقع به محصول جانبی و طبیعی طراحی تبدیل شود. و نهایتاً این که نمایش طراحی بایستی به راحتی قابل نگهداری باشد به نحوی که طراحی همواره به طور صحیح نشان دهندهٔ برنامه باشد.

مقایسه نشانه گذاری های طراحی

خصوصیات زیر در مورد نشان‌گذاری طراحی در زمینهٔ مشخصات کلی فوق‌الذکر تعیین شده‌اند:

قابلیت پیمانه‌ای: نشان‌گذاری طراحی باید ایجاد نرم‌افزار پیمانه‌ای را حمایت کرده و شیوه‌ای برای تعیین رابط را فراهم آورد.

سادگی همه جانبه: یادگیری نشانه‌گذاری طراحی باید تقریباً آسان بوده و کاربرد آن نیز نسبتاً راحت باشد و به طور کلی از لحاظ خواندن نیز دشوار نباشد.

سهولت ویرایش: ضمن پیشرفت روند نرم‌افزار، طراحی رویه‌ای ممکن است نیاز به اصلاح و تغییر داشته باشد. سهولت و راحتی در ویرایش نمایش رویه‌ای، موجب تسهیل وظایف مهندسی نرم‌افزار گردد.

قابلیت خواندن سیستم: نشان‌گذاری که بتواند مستقیماً ورودی یک سیستم توسعه کامپیوتری باشد، مزایای قابل توجهی را به همراه دارد.

قابلیت نگهداری: نگهداری نرم‌افزار پر هزینه‌ترین مرحله در دورهٔ زندگی نرم‌افزاری است. نگهداری پیکربندی نرم‌افزار تقریباً همواره به معنای نگهداری از نمایش طراحی رویه‌ای می‌باشد.

تقویت ساختار: مزایای یک رهیافت طراحی که از مفاهیم برنامه‌نویسی ساخت یافته را به کار می‌برد، قبلاً مورد بحث قرار گرفته است. نشان‌گذاری طراحی که کاربرد صرف سازه‌های ساخت یافته را تقویت می‌کند، یک شیوهٔ خوب طراحی را توسعه می‌دهد.

پردازش خودکار: طراحی رویه‌ای دربردارنده اطلاعاتی است که قابل پردازش بوده و می‌تواند دربارهٔ صحت و کیفیت طراحی، بینش و درک بهتر یا جدیدی را در اختیار طراح قرار دهد. این شناخت از طریق گزارش‌های حاصل از ابزارهای طراحی نرم‌افزار، قابل بهبود و تقویت می‌باشد.

بازنمایی داده‌ها: توانایی نمایش داده‌های محلی و سراسری، عنصر اصلی و ضروری طراحی سطح مؤلفه‌هاست. در وضعیت ایده‌آل، نشان‌گذاری طراحی باید این داده‌ها را به طور مستقیم نمایش دهد.

تأیید منطق: تأیید خودکار منطق طراحی هدفی مهم در مرحلهٔ آزمون نرم‌افزار می‌باشد. نشان‌گذاری که باعث تقویت توانایی تأیید منطق شود، کفایت آزمون را به شدت افزایش می‌دهد.

توانایی " تبدیل به برنامه ": وظیفهٔ بعدی مهندس نرم‌افزار پس از طراحی اجزاء، تولید برنامه است. نشان‌گذاری که به راحتی قابل تبدیل به برنامهٔ منبع باشد، میزان تلاش و خطا را کاهش می‌دهد.

تست‌های فصل شانزده: طراحی درسطح مولفه

۱- کدام طراحی از نظر زمانی در آخرین مرحله انجام می‌گیرد؟

الف) طراحی داده‌ها ب) طراحی معماری

ج) طراحی رویه‌ای د) طراحی واسط

۲- سطح انتزاع در کدام طراحی پایین‌تر است؟

الف) طراحی داده‌ها ب) طراحی معماری

ج) طراحی رویه‌ای د) طراحی واسط

۳- کدام یک از موارد زیر جزء ساختارهای منطقی پیشنهادی دیکسترا (Dijkstre) برای نوشتن برنامه نمی‌باشد؟

الف) Sequence ب) State

ج) Conditional د) Loop

۴- پر هزینه ترین مرحله از چرخه حیات نرم‌افزار کدام است؟

الف) طراحی داده‌ها ب) طراحی معماری

ج) طراحی رویه‌ای د) نگهداری آن

۵- کدام یک از ویژگی‌های نمایش نمودار مستطیلی در مقایسه با نمایش نمودارگردشی(روند نما) وجود ندارد؟

الف) سادگی ب) قابل فهم

ج) غیر ممکن بودن انتقال اختیاری کنترل د) به سهولت تعیین کردن دامنه داده‌ها

۶- کدام یک از موارد زیر از ویژگی‌های PDL نمی‌باشد؟

الف) سادگی ب) قابل فهم بودن

ج) شبه کد بودن د) کامپایلری بودن

۷- کدام یک از مراحل زیر برای توسعه یک جدول تصمیم‌گیری اجرا نمی‌شود؟

الف) لیست کردن عملیاتی که به یک رویه مشخص ربط دارد.

ب) لیست کردن کلیه شرطها

ج) توسعه هر جای‌گشت ممکن از شرطها

د) تعیین قواعد با مشخص کردن این که چه عمل‌هایی برای یک مجموعه از شرایط رخ می‌دهد.

۸- کدام یک از موارد زیر پایه‌ای برای تهیه برنامه‌ای ساختیافته نیست؟

الف) برگشتی ب) شرطی ج) تکرار د) توالی

۹- کدام یک از موارد زیر یک نمایش گرافیکی برای توصیف رویه‌ای است؟

الف) نمودار مستطیلی ب) جدول ج) نمودار ER د) ماتریس گرافیک

۱۰-به طور کلی نمودارهای مستطیلی و روند نماهافلوچارت‌ها باید

الف) به جای زبان‌های طراحی برنامه‌نویسی استفاده شوند.

ب) برای مستندسازی کامل طراحی استفاده شوند یا اصلاً نشوند.

ج) فقط برای مستندسازی و یا ارزیابی طراحی در یک مورد خاص استفاده شوند.

د) هیچ یک از موارد فوق

۱۱-یک جدول تصمیم در موارد زیر استفاده می‌شود.

الف) تمام شرایط مستندسازی می‌شوند.

ب) راهنمایی برای تهیهٔ برنامهٔ مدیریت پروژه است.

ج) فقط در زمانی تهیه می‌شود که موضوع کار، یک سیستم خبره می‌باشد.

د) وقتی که مجموعهٔ پیچیده از شرایط و اعمال در مؤلفه‌ها ظاهر می‌شود.

۱۲-اغلب زبان طراحی برنامه (PDL) عبارت است از

الف) ترکیبی از ساختارهای برنامه نویسی و توصیف متنی

ب) زبان برنامه نویسی مجاز در قالب درست خود

ج) زبان توسعه نرم‌افزاری که قابلیت خواندن توسط ماشین را دارد.

د) راه مفیدی برای نمایش معماری نرم‌افزار است.

۱۳-چون یک زبان طراحی برنامه، زبان برنامه‌نویسی واقعی نیست، بنابراین طراحان آزاد هستند طراحی رویه‌ای خود را بدون نگرانی از خطاهای معنایی (syntax) بنویسند.

الف) درست ب) نادرست

۱۴-مهندسین نرم‌افزار مدرن معتقدند که مفیدترین نماد طراحی برای نمایش رویه فقط تولید شبه کد است.

الف) درست ب) نادرست

۱۵-کدامیک از معیارهای زیر برای ارزیابی موثر بودن یک نماد طراحی رویه خاص مفید است.

الف) قابلیت نگهداری ب) قابلیت پیمانه‌بندی ج) سادگی د) همه موارد فوق

بخش چهارم: مهندسی نرم افزار

فصل ۱۷: اصول و مفاهیم تحلیل مهندسی نرم افزار شیء گرا

ما در جهانی از اشیاء زندگی می‌کنیم. این اشیاء در طبیعت، در نهادهای ساخت دست بشر، در تجارت و در محصولاتی که استفاده می‌کنیم، وجود دارند. آنها را می‌توان بسته‌بندی کرد، توصیف نمود، سازماندهی کرد، ترکیب کرد، دستکاری کرد و ایجاد نمود. بنابراین، تعجبی ندارد که برای ایجاد نرم‌افزارهای کامپیوتری نیز دیدگاهی شیءگرا پیشنهاد شود– این شکل انتزاعی ما را قادر می‌سازد تا جهان را به شیوه‌ای مدلسازی کنیم که بهتر قابل درک و کاوش باشد.

روش شیءگرا در توسعهٔ نرم‌افزار اولین بار در اواخر دههٔ ۱۹۶۰ برای توسعهٔ نرم‌افزار به کار گرفته شد. ولی ۲۰ سال طول کشید تا فنآوری شیءگرا به طور گسترده مورد استفاده قرار گیرد. در سرتاسر دههٔ ۱۹۹۰، مهندسی نرم‌افزار شیءگرا الگوی انتخابی بسیاری از نرم‌افزار نویسان شد و تعداد فزاینده‌ای از سیستم های اطلاعاتی و مهندسان حرفه‌ای به آن روی آوردند. به مرور زمان، فنآوری های شیءگرا جایگزین روش های کلاسیک توسعهٔ نرم‌افزار می‌شوند. سوال مهم این است: چرا؟

پاسخ این سؤال (همانند پاسخ بسیاری از سوالات دیگر در مهندسی نرم‌افزار) پاسخ ساده‌ای نیست. برخی استدلال می‌کنند که نرم‌افزارنویسان حرفه‌ای صرفاً به دنبال یک روش جدید بودند، ولی این دیدگاه بیش از حد ساده‌نگرانه است. فنآوری های شیءگرا به چندین مزیت ذاتی منجر می‌شوند که هم در سطح مدیریتی و هم فنی مزایایی به همراه دارد.

فنآوری های شیءگرا منجر به استفاده مجدد می‌شود و استفاده مجدد (از مؤلفه‌های برنامه) منجر به توسعه سریعتر نرم‌افزارها و برنامه‌هایی با کیفیت بالاتر می‌شود. نگهدرای نرم‌افزارهای شیءگرا آسانتر است زیرا ساختار آن ذاتاً فاقد پیوستگی است. این موضوع، به هنگام اعمال تغییرات، اثرات جانبی کمتری به وجود می‌آورد و برای مهندس نرم‌افزار و مشتری دردسر کمتری ایجاد می‌کند. به علاوه، تطبیق دادن و تغییر دادن اندازهٔ سیستم های شیءگرا آسانتر است (یعنی سیستم های بزرگ را می‌توان با مونتاژ کردن زیرسیستم های قابل استفاده مجدد ایجاد کرد).

سال ها بود که اصطلاح شیءگرا (OO) برای مشخص کردن روشی به کار می‌رفت که در آن از زبانهای برنامه‌نویسی شیءگرا (مثل اِدا ۹۵، جاوا، C++، ایفل و اسمالتاک) استفاده می‌شود. امروزه الگوی OO شامل دیدگاهی کامل از مهندسی نرم‌افزار می‌شود. Berar به این نکته چندین اشاره دارد [BER93]:

مزایای فنآوری شیءگرا اگر به طور زودهنگام و در سرتاسر فرآیند نرم‌افزار به آن پرداخته شود. بهبود می‌یابد. آنها که به فنآوری شیءگرا روی می‌آورند، باید تأثیر آن را بر کل فرآیند مهندسی نرم‌افزار مورد سنجش قرار دهند. فقط استفاده از برنامه‌نویسی شیءگرا (OOP) نیست که بهترین نتایج را ببار دهد. مهندسان نرم‌افزار و مدیران آنها باید چنین عناصری را به عنوان تحلیل نیازهای شیءگرا (OORA)، طراحی شیءگرا (OOD)، تحلیل دامنه شیءگرا (OODA)، سیستم های بانک اطلاعاتی شیءگرا (OODBMS) و مهندسی نرم‌افزار شیءگرا به کمک کامپیوتر (OOCASE) در نظر بگیرند.

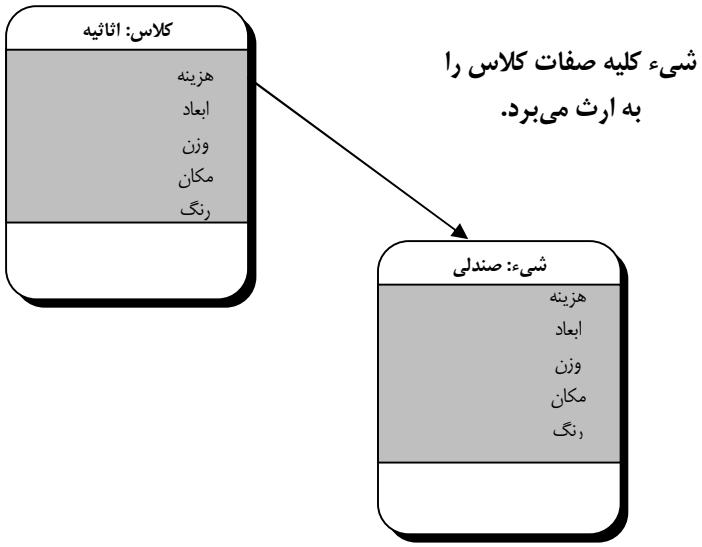
مفاهیم و قواعد کلی شیء گرایی

برای درک موضوع از دیدگاه شیء گرایی ، مثالی از یکی از اشیای دنیای واقعی – یعنی صندلی – را در نظر بگیرید.

- صندلی یک **عضو**(member) یا یک **نمونه**(instance) از یک کلاس بزرگتر از اشیاء بنام اثاث خانه است.
- در کلاس اثاث خانه مجموعه ای از **صفات**(attributes) کلی و مشترک به تمام اشیای این کلاس نسبت داده می شود. بعنوان مثال تمام اثاث خانه قیمت، ابعاد، وزن، محل، رنگ و خیلی صفات احتمالی دیگر دارند.
- چون صندلی عضوی از اثاث خانه است، تمام صفاتی را که برای کلاس اثاث خانه تعریف شده است به **ارث** می برد(inherits).

کوشش کرده‌ایم تا تعریفی حکایت‌وار از کلاس را با توصیف صفات آن ارایه کنیم. ولی چیزی کم است. هر کدام از اعضای کلاس اثاثیه را می‌توان به چندین شیوه دستکاری کرد. می‌توان آن را خرید، فروخت، تغییر فیزیکی در آن ایجاد کرد (مثلاً پایه‌ها را اره کرد یا آن را ارغوانی رنگ کرد) یا از مکانی به مکان دیگر جابجا کرد. هر یک از **عملیات** (یا **متدها** یا **سرویس ها**) یک یا چند صفت شیء را تغییر می‌دهند. برای مثال، اگر صفت مکان یک عنصر داده‌ای مرکب به صورت زیر باشد:

اتاق + طبقه + ساختمان= مکان



شکل ۱ : وراثت بین کلاس ها

در این صورت، عملی که جابجایی نام دارد، یک یا چند مورد از این عناصر داده‌ای (ساختمان، طبقه، اتاق) را که مکان را تشکیل می‌دهند، تغییر می‌دهد. برای انجام این کار، عمل جابجایی باید از این عناصر داده‌ای **آگاه** باشد. مادامی که صندلی و میز هر دو نمونه‌های کلاس اثاثیه باشند، از عمل جابجایی می‌توان برای آنها استفاده کرد. همهٔ عملیات معتبر (مثل خریدن، فروختن، توزیع کردن) برای کلاس اثاثیه به تعریف شیء **متصل** هستند و برای کلیه نمونه‌های این کلاس به ارث گذاشته می‌شوند.

شیء صندلی (و کلاً همهٔ اشیاء) داده‌ها (مقادیر صفاتی که صندلی را تعریف می‌کنند)، عملیات (عملیاتی که برای تغییر دادن صفات صندلی به کار می‌روند)، اشیای دیگر (اشیای مرکبی که قابل تعریف هستند)، ثابت ها(مقادیر ثابت) و اطلاعات مربوطه دیگر را **بسته‌بندی** می‌کنند. بسته‌بندی بدان معنا است که کلیهٔ این اطلاعات تحت یک نام بسته‌بندی شوند و به عنوان یک مشخصه یا قطعه برنامه به کار برده شوند.

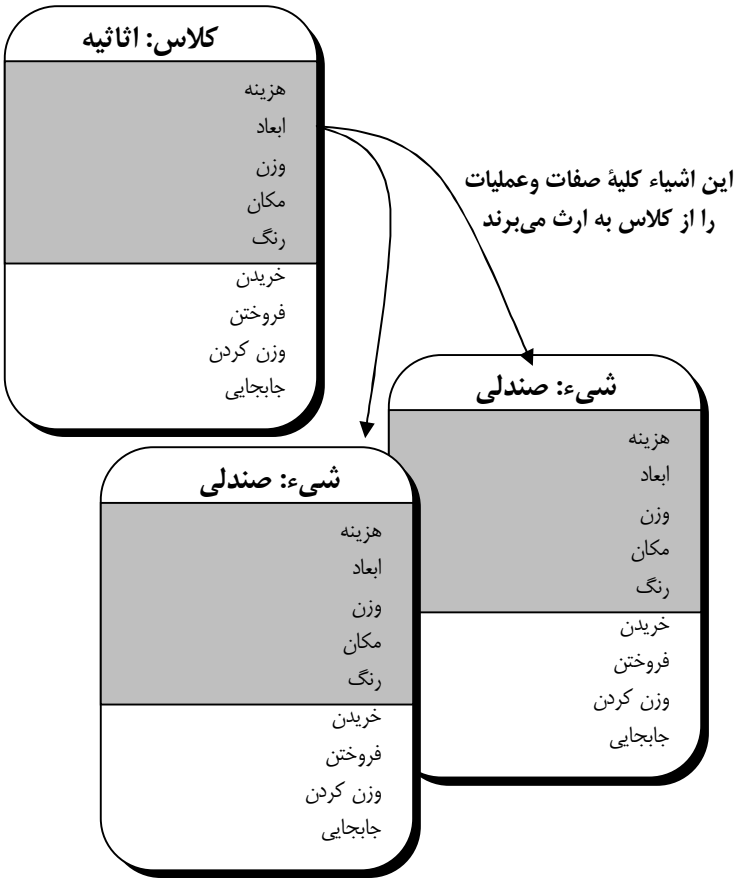
اکنون که با چند مفهوم اساسی آشنا شدیم، تعریفی رسمی‌تر از شیءگرایی، بی‌مناسبت نخواهد بود. Coad و Yourdon [COA91] این اصطلاح را چنین تعریف می‌کنند:

ارتباطات + وراثت + طبقه‌بندی+ اشیاء= شیءگرایی

بنابراین هر شیء از کلاس **اثاث خانه** به روش های مختلف قابل تغییر است : خریده شود ، فروخته شود یا از یک مکان به مکان دیگر منتقل شود.

- هر کدام از این عملیات(operations) یا خدمات(services) یا متدها(methods) یک یا چند صفت از صفات شیء را تغییر می دهند.

- **اشیا** در درون خود داده ها (مقادیر صفات)، عملیات (اعمالی که بر شیء وارد می شوند تا صفات آن را تغییر دهد)، اشیای دیگر(اشیای ترکیبی)، ثابت ها و سایر اطلاعات مربوط را بسته بندی می کنند.

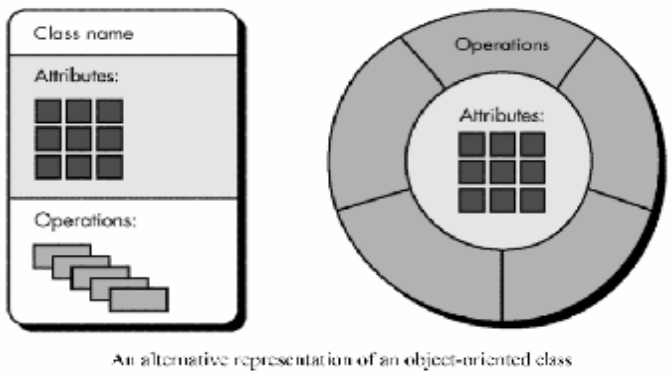


شکل ۲ : ارث بری عملیات از کلاس به عملیات

کلاس ها و اشیاء

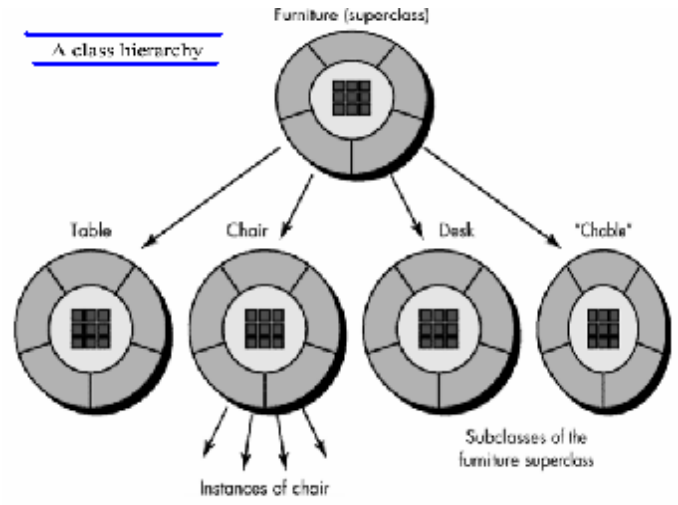
کلاس یک مفهوم شی گرایی است که داده ها و رویه هایی را که برای توصیف محتوا(content) و رفتار(behaviour) یک موجودیت دنیای واقعی لازم است ، بسته بندی می کند.

انتزاع داده ای(صفات) که کلاس را توصیف می کند در میان یک "دیوار" به نام **انتزاع رویه ای**(که **عملیات، خدمات** یا **متدها** نامیده می شود) قرار گرفته اند که قادر به تغییر داده ها در بعضی موارد است.



شکل ۳ : نمایشی از کلاس ها

تنها راه برای دسترسی به صفات از طریق یکی از متدهاست(کلاس داده ها و متدها را **بسته بندی می کند**).
این کار باعث **پنهان سازی اطلاعات(Information Hiding)** می شود و باعث کمترشدن تاثیر اثرات جانبی مرتبط با تغییرات می شود.
از آنجایی که متدها تعداد محدودی از صفات را تغییر می دهند، آنها **منسجم(Cohesive)** هستند؛ و بخاطر اینکه ارتباطات فقط از طریق متدها رخ می دهد ، کلاس از سایر اجزای سیستم **جدا(Decoupled)** می شود. این خصوصیات منجر به شکل گیری یک **نرم افزار با کیفیت بالا(High-Quality Software)**می شود.
زبر کلاس (Superclass) مجموعه ای از کلاس هاست، و **زیر کلاس(Subclass)** یک نمونه خاص از یک کلاس است.
این تعاریف دلالت بر وجود **سلسله مراتب کلاس(** Class Hierarchy) می کند که در آن صفات و عملیات زبر کلاس بوسیله زیر کلاس ها به ارث برده می شوند. البته هر کدام از این زیر کلاس ها ممکن است داده ها و متدهای "خصوصی" داشته باشند.



شکل ۴ : نمایشی از سلسه مراتب کلاس ها

صفات

پیش از این دیدیم که صفات به کلاس ها واشیاء متصل هستند و شیء یا کلاس را به نحوی توصیف می‌کنند. بحثی دربارهٔ صفات توسط de Champeaux و همکاران وی [CHA93] ارایه شده است:

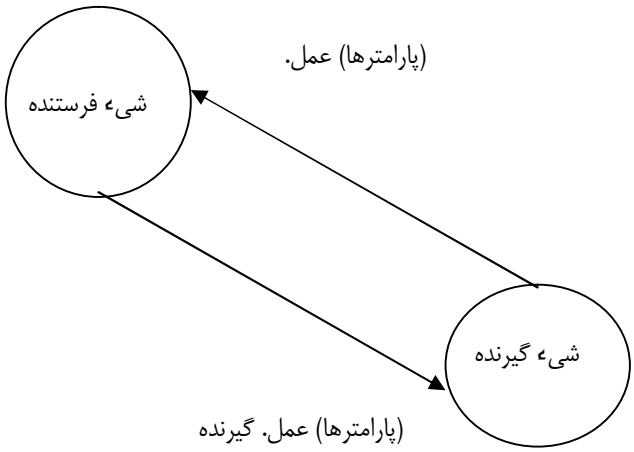
نهادهای موجود در زندگی واقعی، غالباً با واژه‌هایی توصیف می‌شوند که نشانگر ویژگی های پایدارند. اکثر اشیای فیزیکی دارای ویژگی هایی از قبیل رنگ، شکل، وزن و جنس مواد هستند. افراد دارای ویژگی هایی مثل تاریخ تولد، والدین، نام و رنگ چشم هستند. هر ویژگی را می‌توان به عنوان رابطه‌ای دودویی میان یک کلاس و یک دامنه معین در نظر گرفت.

- صفات به کلاس ها و اشیا متصل هستند، و آنها کلاس یا شی را به نوعی تعریف می کنند.
- یک صفت مقدار خود را از یک میدان مقادیر می گیرد. در اکثر حالات، دامنه فقط مجموعه ای از مقادیر است. مثال: دامنه مقادیر برای رنگ عبارتند از { سفید ، سیاه ، نقره ، خاکستری ، آبی ، ... } .
- در حالات پیچیده تر، دامنه می تواند مجموعه ای از کلاس ها باشد. مثال: دامنه مقادیر برای کلاس خودرو عبارت است از { ۴ سیلندر، ۶ سیلندر، ۸ سیلندر، ۱۰ سیلندر، ۲۴ سیلندر، و ... }.

عملیات، متدها و خدمات

یک شیء، داده‌ها را (که به صورت مجموعه‌ای از صفات نمایش داده می‌شود) و الگوریتم هایی که این داده‌ها را پردازش می‌کنند، بسته‌بندی می‌کند. این الگوریتم ها را که عملیات، متدها یا سرویس ها می‌نامند، می‌توان از دیدگاه سنتی به عنوان پیمانه در نظر گرفت.

هر کدام از این عملیات که در شیء بسته بندی شده است، در واقع یک نوع نمایش برای یکی از رفتارهای شیء را تامین می کند.



شکل ۵ : مبادله پیام ها بین اشیاء

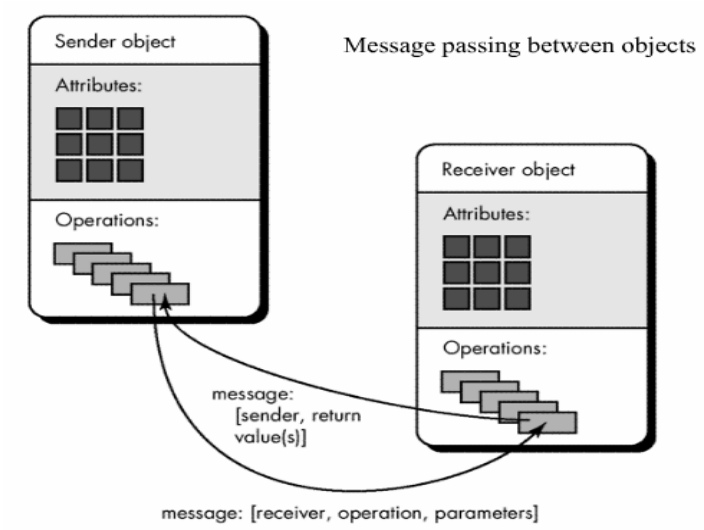
پیغام ها (Messages)

پیغام ها ابزارهای تعامل اشیاء هستند. با استفاده از فناوری معرفی شده در بخش قبل، پیغام باعث برانگیختن رفتاری خاص در شیء گیرنده می‌شود. این رفتار زمانی مشاهده می‌شود که عملی اجرا شود. بنابراین:

- پیغام ها ابزاری هستند که اشیا از طریق آنها با یکدیگر ارتباط برقرار می کنند.
- عملیات موجود در شی فرستنده پیغامی به فرم زیر تولید می کند:
- پیغام : [مقصد ، عملیات ، پارامترها]

که **مقصد** شی گیرنده را – که با دریافت پیغام فعال می شود – تعریف می کند ، **عملیات** به عملیاتی اشاره می کند که پیغام را قرار است دریافت کند، و **پارامترها** اطلاعاتی را تامین می کنند که برای درست انجام شدن عملیات لازمند.

شی گیرنده با انتخاب عملیاتی که نام پیغام را پیاده سازی می کند، اجرای آن، و بازگرداندن کنترل به فراخواننده؛ به پیغام پاسخ می دهد.



شکل٦ : مبادله پیام ها بین دو شیء

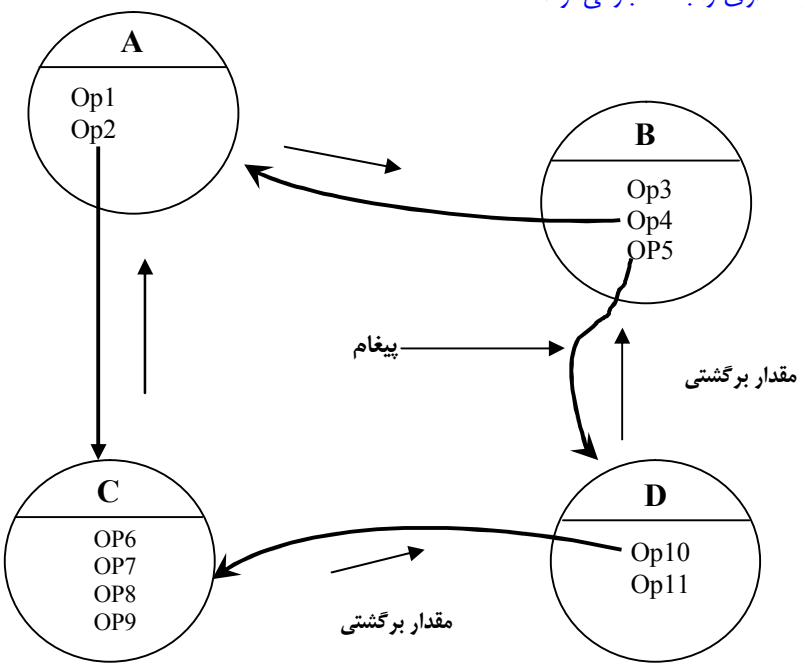
به عنوان مثالی از مبادله پیغامها در داخل یک سیستم OO، اشیای شکل ۷ را در نظر بگیرید. چهار شیء A، B، C و D با مبادله پیغام با یکدیگر ارتباط برقرار می‌کنند. برای مثال، اگر شیء B بخواهد عملیات op10 از شیء D را اجرا کند، پیغامی به شکل زیر به D ارسال می‌کند:

D . op10 (data)

شیء D نیز به عنوان بخشی از اجرای op10 ممکن است پیغامی به شکل زیر به C بفرستد:

C . op08 (data)

C عمل op08 را می‌یابد، آن را اجرا می‌کند، و سپس یک مقدار بازگشتی مناسب به D ارسال می‌کند. عمل op10 کامل می‌شود و مقداری را به B بازمی‌گرداند.



شکل۷ : مبادله پیام ها بین اشیاء

بسته بندی (Encapsulation)

گرچه ساختار و اصطلاحات معرفی شده در بخش های قبل سیستم های OO را از همتاهای سنتی آنها متفاوت می‌سازد، سه ویژگی سیستم های شیءگرا آنها را منحصر به فرد می‌سازد. چنان که پیش از این نیزگفتیم، کلاس و اشیایی که از آن ایجاد می‌شوند، داده‌ها و عملیاتی را که بر روی آنها عمل می‌کنند، یک جا **بسته‌بندی** می‌کنند.

مزایای بسته بندی

۱. جزییات داخلی پیاده سازی داده ها و رویه ها از دنیای خارج پنهان هستند (پنهان سازی اطلاعات). این باعث می شود که گسترش اثرات جانبی در هنگام تغییرات کم شود.
۲. ساختارهای داده و عملیاتی که آنها را تغییر می دهند در یک موجودیت تکی به نام کلاس با یکدیگر ادغام می شوند. این باعث می شود که استفاده مجدد از مولفه (Component) راحت تر شود.
۳. واسط های بین اشیای بسته بندی شده ساذه تر می شوند. شیء فرستنده یک پیغام لازم نیست که به جزییات داخلی ساختارهای داده توجه کند. این باعث می شود که اتصال(Coupling) سیستم کاهش یابد.

وراثت (Inheritance)

وراثت یکی از مهمترین تفاوت ها بین سیستم های سنتی و سیستم های شی گراست. یک زیر کلاس Y تمام صفات و عملیات را از زبر کلاس خود X به ارث می برد. این بدین معنی است که تمام ساختارهای داده و الگوریتم هایی که در ابتدا برای X طراحی و پیاده سازی شده اند ، اکنون برای Y نیز موجود می باشند.
هر تغییر در داده ها یا عملیات موجود در زبرکلاس سریعا بوسیله تمام زیرکلاسهایی که از آن زبرکلاس ارث می برند، به ارث برده می شود. بنابراین سلسله مراتب کلاس تبدیل به مکانیزمی شده است که بوسیله آن تغییرات (در سطوح بالا) سریعا می توانند در سیستم پخش شوند.

- **گزینه های لازم برای تولید یک کلاس جدید**

۱. می توان یک کلاس را از ابتدا طراحی و ساخت در این حالت از وراثت استفاده نمی شود.
۲. سلسله مراتب کلاس را جستجو کنیم تا کلاسی بالاتر را در سلسله مراتب پیدا کنیم که اکثر صفات و عملیات مورد نیاز را داشته باشد. کلاس جدید از کلاس بالاتر ارث می برد و در صورت نیاز صفات و عملیات جدید را اضافه می کنیم.
۳. سلسله مراتب کلاس را دوباره سازماندهی کنیم تا صفات و عملیات مورد نیاز قابل ارث بری شوند.
۴. خصوصیات یک کلاس موجود را می توان دوباره نویسی (Override) کرد و نسخه های خصوصی صفات و عملیات را برای این کلاس جدید پیاده سازی کرد.

چند ریختی (Polymorphism)

چند ریختی باعث می شود که چندین عملیات بتوانند از یک نام استفاده کنند. این کار باعث می شود که تعداد خطوط برنامه برای پیاده سازی کاهش یابد و اعمال تغییرات را تسهیل کنند.

- برای درک چند ریختی ، یک برنامه سنتی(رویه ای) را در نظر بگیرید که چهار نوع گراف را باید بکشد : گراف‌های خطی، pie charts ، histograms و Kiviat diagrams . ایده آل این است که هنگامی که داده ها برای یک نوع خاص گراف فراهم شد، گراف باید خودش را بکشد. برای انجام این کار در یک برنامه سنتی(رویه ای) باید برای هر نوع، یک پیمانه را در نظر بگیریم و توسعه دهیم.

- سپس، در داخل طراحی هر نوع گراف، منطق کنترلی مشابه زیر باید تعبیه شود :

```
case of graphtype:
    if graphtype = linegraph then DrawLineGraph (data);
    if graphtype = piechart then DrawPieChart (data);
    if graphtype = histogram then DrawHisto (data);
    if graphtype = kiviat then DrawKiviat (data);
end case;
```

برای حل این مشکل، تمام گراف ها را زیر کلاس هایی از یک کلاس کلی به نام **graph** قرار می دهیم. با استفاده از مفهوم overloading هر زیر کلاس یک عملیات به نام draw تعریف می کند. یک شی می تواند پیغام draw را به هریک از اشیایی که از هر یک از زیر کلاس ها نمونه سازی (instantiate) شده است بفرستد. شیئی که پیغام را دریافت می کند، عملیات draw مربوط به خود را فراخوانی می کند تا گراف مناسب را تولید کند.

شناسایی عناصر یک مدل شیء گرا

عناصر مدل تحلیل شیء گرا شامل موارد زیر است:

- شناسایی کلاس ها و اشیاء
- مشخص کردن صفات
- تعریف عملیات
- به پایان رساندن تعریف اشیاء

در ادامه به بحث در مورد هریک از این عناصر خواهیم پرداخت.

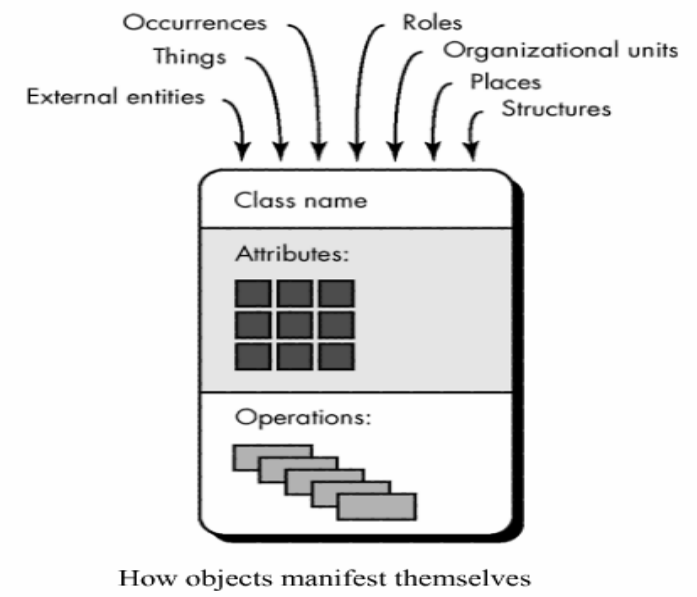
شناسایی کلاس ها و اشیاء

شناسایی اشیاء را با بررسی بیان مساله یا (با استفاده از اصطلاحی که در فصل ۱۲ معرفی شد) با اجرای **تجزیه گرامری** روی شرح پردازشی سیستمی که قرار است ساخته شود، آغاز می‌کنیم. برای تعیین اشیاء، زیر اسم ها یا عبارات اسمی خط کشیده، آنها را در جدولی قرار می‌دهیم. به اسم های مترادف نیز باید توجه داشت. اگر یک شیء برای پیاده‌سازی یک راهکار، مورد نیاز باشد، در آن صورت بخشی از فضای حل را تشکیل می‌دهد؛ در غیر این صورت، اگر شیء فقط برای توصیف راهکار لازم باشد، بخشی از فضای مساله به شمار می‌رود. ولی هنگامی که کلیه اسم ها را مشخص کردیم، چه باید بکنیم؟

اشیاء می توانند یکی از طرق زیر خود را نشان دهند:

۱. **موجودیت های خارجی**(مثلا سیستم های دیگر، دستگاه ها ،افراد) که اطلاعات مورد استفاده سیستم کامپیوتری را تولید یا مصرف می کنند.
۲. **چیزهایی**(مثلاً گزارش ها، صفحات نمایش، نامه ها، علایم الکترونیکی) که بخشی از دامنه اطلاعاتی مسأله به شمار می روند.
۳. **رخدادها** یا **وقایعی**(مثلاً کامل شدن حرکات روبات) که در حیطه عملکرد سیستم رخ می دهند.
۴. **نقش هایی**(مثل مدیر، مهندس، فروشنده) که افراد در حال تعامل با سیستم، بر عهده دارند.
۵. **واحدهای سازمانی**(مثل بخش، گروه ، تیم) که با یک کاربرد خاص سروکار دارند.
۶. **مکان هایی**(مثل سالن تولید یا بارانداز) که حیطه مسأله و وظیفه کلی سیستم را مشخص می کند.

۷. **ساختارهایی**(مثل حسگرها، وسایل نقلیه چهار چرخ یا کامپیوترها) که کلاسی از اشیاء یا کلاس های مرتبطی از اشیاء را تعریف می کنند.



شکل۸ : چگونگی نشان دادن اشیاء

توجه به این نکته نیز مهم است که اشیاء چه چیزهایی نیستند. به طور کلی یک شیء هیچگاه نباید یک "نام رویه ای دستوری" داشته باشد. یک تجزیه گرامری را می توان برای تفکیک اشیاء (اسامی) و عملیات به کار برد.

- شش خصوصیت گزینشی که تحلیلگر باید در مورد هر شیء بالقوه ای که در مدل تحلیل بکار می برد در نظر بگیرد :

۱. **اطلاعات ذخیره شده**– شیء بالقوه فقط در صورتی هنگام تحلیل مفید خواهد بود که برای عملکرد سیستم به اطلاعات آن نیاز باشد.

۲. **خدمات مورد نیاز**– شیء بالقوه باید مجموعه ای از عملیات قابل شناسایی داشته باشد که می توانند صفات آنرا به طریقی تغییر دهند.

۳. **صفات چندگانه**– یک شیء با یک صفت ممکن است در طول طراحی مفید باشد ولی ممکن است بهتر باشد در هنگام تحلیل به عنوان صفتی از شیء دیگر منظور شود.

۴. **صفات مشترک**– مجموعه ای از صفات را می توان برای یک شیء بالقوه تعریف کرد و این صفت ها در تمام نمونه های شیء بکار می روند.

۵. **عملیات مشترک**– مجموعه ای از عملیات را می توان برای یک شیء بالقوه تعریف کرد و این عملیات در تمام نمونه های شیء بکار می روند.

۶. **نیازهای ضروری**– موجودیت های خارجی که در فضای مسأله ظاهر می شوند و اطلاعات ضروری برای کارکرد هر راهکار را تولید یا مصرف می کنند، باید به عنوان شیء در مدل نیازها تعریف شوند.

مشخص کردن صفات

- صفات شیء آن را طوری تعریف و توصیف می کنند که برای ورود به مدل تحلیل انتخاب شده است.

- برای توسعه یک مجموعه بامعنی از صفات، تحلیلگر می تواند شرح پردازش مسأله را مطالعه کند و چیزهایی را انتخاب کند که به طور منطقی به شیء تعلق دارند.
- به علاوه باید به این سؤال پاسخ داد: «چه عناصر داده ای (مرکب ویا ساده) این شیء را در حیطه مسأله به طور کامل تعریف می کنند؟»

تعریف عملیات

عملیات، رفتار یک شیء را تعریف می کنند و صفات آنرا به طریقی تغییر می دهند. به طور مشخص، یک عمل مقدار یک یا چند صفت موجود در شیء را تغییر می دهد. یک عمل باید از ماهیت صفات شیء آگاه باشد و باید به شیوه ای پیاده سازی شود که دستکاری داده های به دست آمده از صفات را میسر سازد.

- **رده بندی عملیات**

۱.عملیاتی که با داده ها را به طریقی کار می کنند(مثلاً اضافه کردن، حذف کردن، قالب بندی دوباره و گزینش)،
۲.عملیاتی که محاسبه ای را انجام می دهند، و
۳.عملیاتی برای رخ دادن یک رویداد کنترلی بر شیء نظارت می کنند.

نهایی سازی تعریف اشیاء

تعریف عملیات، آخرین مرحله در کامل کردن تشخیص اشیاء است. عملیات اضافه ای را می توان با در نظر گرفتن **تاریخچه حیات** یک شیء و پیغام هایی که در میان اشیاء تعریف شده در سیستم مبادله می شوند، تعیین کرد. تاریخچه حیات کلی یک شیء را می توان با در نظر گرفتن این نکته تعیین کرد که شیء باید به شیوه های دیگری ایجاد، اصلاح، دستکاری یا خوانده شود و احتمالاً حذف شود.

مدیریت پروژه های نرم افزاری شیءگرا

- **فعالیت های مدیریت مدرن پروژه نرم افزاری**

۱. ساختن یک چهارچوب فرایند مشترک برای پروژه.
۲. استفاده از چهارچوب و معیارهای تاریخی برای براورد کار و زمان لازم.
۳. ایجاد نقاط عطف که اندازه گیری پیشرفت کار را میسر می کنند.
۴. تعریف یکسری نقاط کنترلی برای مدیریت ریسک، تضمین کیفیت و کنترل.
۵. مدیریت تغییراتی که به طور ذاتی در پیشرفت پروژه رخ می دهند.
۶. پیگیری، نظارت و کنترل پیشرفت.

در ادامه عناوین زیر را بررسی می کنیم:

- **چهارچوب فرایند مشترک برای OO**
- **معیارها و برآورد پروژه شیءگرا**
- **یک رهیافت براورد و زمانبندی OO**
- **پیگیری پیشرفت برای یک پروژه شیءگرا**

چهارچوب فرایند مشترک(Common Process Framework-CPF)

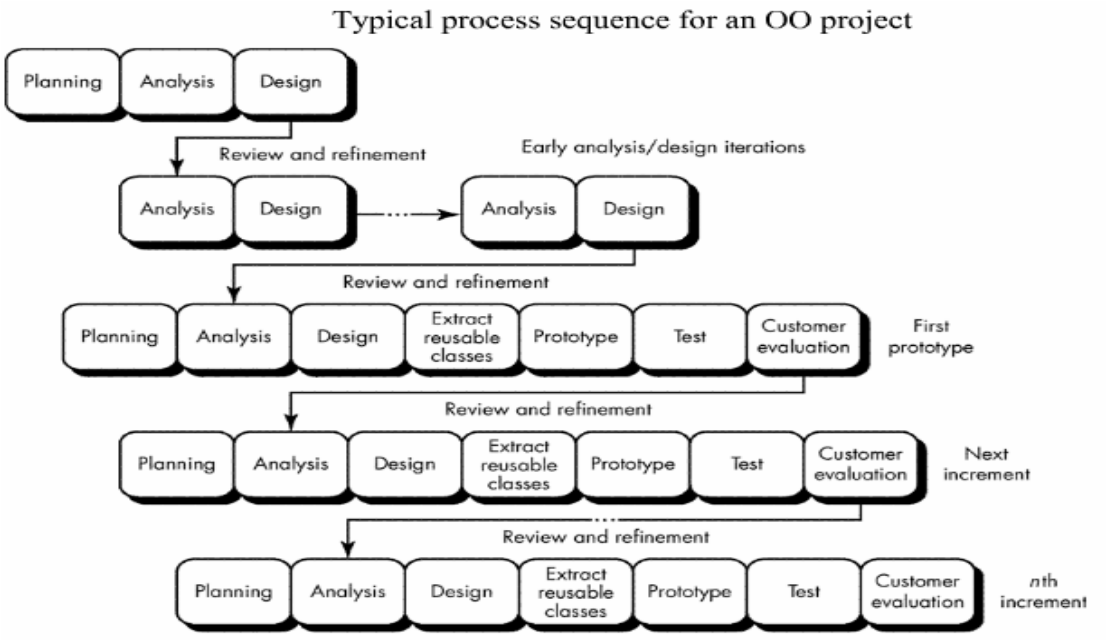
- یک چهارچوب فرایند مشترک (CPF) روشی را تعیین می کند که یک سازمان برای مهندسی نرم افزار تعیین می کند.
- (CPF) تعیین می کند که چه الگویی برای ساختن و نگهداری نرم افزار باید اعمال شود و چه وظایف و نقاط عطفی مورد نیاز خواهد بود.
- CPF درجه سختی که پروژه های مختلف دارند را تعیین می کند.
- CPF همواره قابل انطباق است و می تواند نیازهای فردی تیم پروژه را برآورده سازد. این مهمترین خصوصیت آن است.
- مهندسی نرم افزار شیءگرا مدل بازگشتی/ موازی را برای توسعه نرم افزار تقویت می کند.
- حال چگونه ما مدل بازگشتی/موازی را برای مهندسی نرم افزار شیءگرا به کار بریم؟

پاسخ

۱. انجام تحلیل کافی برای جداکردن کلاس ها و ارتباطات اصلی مسأله
۲. انجام اندکی طراحی برای تعیین اینکه آیا کلاس ها و ارتباطات را می توان عملاً پیاده سازی کرد یا خیر.
۳. استخراج اشیای قابل استفاده مجدد از یک کتابخانه، برای ساختن یک نمونه اولیه تقریبی
۴. اجرای چند آزمون جهت کشف خطاها در نمونه اولیه.
۵. دریافت نظرات مشتری در خصوص نمونه اولیه.
۶. اصلاح مدل تحلیل براساس آنچه که از نمونه اولیه، از انجام طراحی و از نظرات مشتری فرا گرفته اید.
۷. پالایش طراحی به منظور انجام دادن تغییرات.
۸. برنامه نویسی اشیای خاص (که در کتابخانه موجود نیست.)
۹. مونتاژ یک نمونه اولیه جدید با استفاده از اشیای کتابخانه و اشیای جدیدی که ایجاد کرده اید.
۱۰. اجرای چند آزمون برای کشف خطاها در نمونه اولیه.
۱۱. دریافت نظریات مشتری در خصوص نمونه اولیه.

- هردور تکرار نیاز به برنامه ریزی، مهندسی(تحلیل، طراحی، استخراج کلاس ها، ایجاد نمونه اولیه و آزمون) و فعالیتهای ارزیابی دارد.

- در اثنای طرح ریزی، فعالیت های مرتبط با هر یک از مؤلفه های مستقل، برنامه ریزی و زمانبندی می شود.
- طی مراحل اولیه مهندسی، تحلیل و طراحی به طور تکراری انجام می شود. هدف آن است که کلیه عناصر مهم مدل های تحلیل و طراحی OO مشخص شوند.
- به موازات پیشرفت کارهای مهندسی، نسخه های تدریجی از نرم افزار ساخته می شود.
- در اثنای این تکامل، برای هر گام یک سری بازبینی، ارزیابی مشتری و آزمون اجرا می شود و نتیجه آن بر فعالیت برنامه ریزی بعدی تأثیر خواهد گذاشت.



شکل۹ : فرایندهای متداول برای یک پروژه OO

معیارها و برآورد پروژه

- تکنیک های سنتی برآورد پروژه نرم افزاری، نیازمند برآورد کردن تعداد خطوط کد(LOC) یا ارزش تابعی(عملکرد) یعنی FP به عنوان مبنای برآورد است.
- از آنجا که یکی از اهداف پروژه های OO استفاده مجدد است ، برآوردهای مبتنی بر LOC چندان معنی پیدا نمی کند.
- برآوردهای FP را می توان به طور کارامد استفاده نمود، زیرا شمارش های دامنه اطلاعاتی مورد نیاز را به راحتی از روی بیان مسأله می توان به دست آورد.

یک مجموعه پیشنهادی از معیارهای پروژه

تعداد متون سناریو– یک متن سناریو، مراحلی است که تعامل میان کاربر و برنامه کاربردی را توصیف می کند. هر متن به شکل زیر سازماندهی می شود.

{**آغازگر، عمل، شرکت کننده**}

که در آن **آغازگر** یک شیء است که سرویسی را درخواست می کند، **عمل** نتیجه درخواست و **شرکت کننده** شیء سرویس دهنده ای است که درخواست را برآورده می کند.

- تعداد متون سناریو مستقیماً با اندازه برنامه کاربردی و با تعداد نمونه های آزمونی که باید پس از ساخته شدن سیستم روی آن امتحان شوند، ارتباط دارد.

معیارها و برآورد پروژه

۱. **تعداد کلاس های کلیدی.** کلاس های کلیدی **«مؤلفه های بسیار مستقل»** هستند که در اوایل OOA تعریف می شوند.

از آنجا که کلاس های کلیدی، در دامنه مسأله اهمیت اساسی دارند، تعداد این گونه کلاس ها شاخصی از مقدار کار لازم برای توسعه نرم افزار و نیز شاخصی از مقدار بالقوه استفاده مجدد در اثنای توسعه سیستم است.

۲. **تعدادکلاس های پشتیبان.** کلاس های پشتیبان برای پیاده سازی سیستم لازمند ولی ارتباط مستقیم با دامنه مسأله ندارند.

مثال ها عبارتند از کلاس های GUI، کلاس های دستیابی به بانک های اطلاعاتی و کلاس های محاسباتی.

۳. **تعداد میانگین کلاس های پشتیبان به ازای هر کلاس کلیدی**– اگر تعداد میانگین کلاس های پشتیبان به ازای هر کلاس برای یک مسأله مشخص باشد، برآورد بسیار ساده می شود. پیشنهاد می شود که در برنامه کاربردی با GUI تعداد کلاس های پشتیبان دو تاسه برابر کلاس های کلیدی و در برنامه های فاقد GUI یک تا دو برابر کلاس های کلیدی باشد.

۴. **تعداد زیرسیستم ها**– یک زیرسیستم، مجموعه ای از کلاس هاست که پشتیبان عملکردی است که برای کاربر نهایی سیستم قابل مشاهده است. هنگامی که زیرسیستم ها شناسایی شدند، تنظیم یک زمانبندی منطقی، آسانتر می شود.

رهیافت براورد و زمانبندی

رهیافت پیشنهادی

۱. ایجاد برآوردهایی با استفاده از تجزیه کار، تحلیلFP و هر روش دیگری که برای کاربردهای سنتی قابل اجرا باشد.
۲. با استفاده از OOA، متون سناریو را تهیه کرده یک شمارش تعیین کنید. تعداد متون سناریو ممکن است با پیشرفت پروژه تغییر کند.
۳. با استفاده از OOA تعداد کلاسهای کلیدی را تعین کنید.
۴. نوع واسط را برای برنامه کاربردی دسته بندی کرده برای کلاسهای پشتیبان ضریبی ایجاد کنید.

جزئیات مرحله

نوع واسط	ضریب
بدون GUI	2.0
واسط کاربر متنی	2.25
GUI	2.5
GUI پیچیده	3.0

۵. تعداد کل کلاس ها (کلیدی+ پشتیبان) را در تعداد میانگین واحدهای کاری ضرب کنید.Lorenz و Kidd به ازای هر کلاس ۱۵ تا ۲۰ نفر–روز کار پیشنهاد می کنند.
۶. برآورد مبتنی بر کلاسها را با ضرب تعداد میانگین واحدهای کار به ازای هر متن سناریو، چک کنید.

زمانبندی پروژه های شیءگرا با توجه به ماهیت تکراری چهارچوب فرآیند، دشوار است. Lorenz و Kidd مجموعه ای از معیارها را پیشنهاد می کنند که به زمانبندی پروژه کمک می کند:

۱. **تعدادتکرارهای اصلی**- با یادآوری مدل حلزونی یک دور تکرار اصلی متناظر با طی کردن ۳۶۰ درجه از حلزون است. Lorenz و Kidd پیشنهاد می کنند تکرارهایی با طول ۲,۵ تا ۴ ماه را به بهترین وجه می توان پیگیری و مدیریت کرد.
۲. **تعداد قراردادهای کامل شده**- قرارداد به گروهی از مسؤولیت های عمومی مرتبط اطلاق می شود که توسط زیرسیستم ها و کلاس ها فراهم می شوند. قرارداد یک نقطه عطف عالی است و حداقل یک قرارداد باید با هربار تکرار مرتبط شود.

پیگیری پیشرفت برای یک پروژه شیءگرا

با اجرای همزمان فعالیت های یک پروژه OO نقاط عطفی وجود دارد که باید توسط مدیر پروژه کنترل گردد. این نتقاط عطف و موارد مربوطه در ادامه ارایه شذه است.

۱- نقاط عطف تکنیکی: تکمیل شدن تحلیل شیءگرا

- همه کلاس ها و سلسله مراتب کلاسها تعریف و بازبینی شده اند.
- صفات و عملیات کلاس های مرتبط با یک کلاس تعریف و بازبینی شده اند.
- روابط کلاس ها تعیین و بازبینی شده اند.
- یک مدل رفتاری ایجاد و بازبینی شده است.
- کلاس های قابل استفاده مجدد مشخص شده اند.

۲- نقاط عطف تکنیکی: تکمیل شدن طراحی شیءگرا

- مجموعه ای از زیرسیستم ها تعریف و بازبینی شده است.
- کلاس ها به زیرسیستم ها تخصیص داده و بازبینی شده است.
- تخصیص وظایف انجام و بازبینی شده است.
- مسؤولیت ها و مشارکت ها مشخص شده اند.
- صفات و عملیات طراحی و بازبینی شده اند.
- مدل رد و بدل کردن پیغام ایجاد و بازبینی شده است.

۳- نقاط عطف تکنیکی: تکمیل شدن برنامه نویسی شیءگرا

- هر یک از کلاس های جدید از مدل طراحی به کد تبدیل شده است.
- کلاسهای استخراج شده (از کتابخانه) پیاده سازی شده اند.
- نمونه ساخته شده است.

۴- نقاط عطف تکنیکی: انجام آزمون شیءگرا

- درستی و کامل بودن مدل تحلیل و طراحی OO مورد بازبینی قرار گرفته است.
- یک شبکه کلاس- مسؤولیت- مشارکت توسعه یافته و مورد بازبینی قرار گرفته است.
- موارد آزمون طراحی شده اند و آزمون هایی در سطح کلاس برای هر کلاس اجرا شده اند.
- موارد آزمون طراحی شده و آزمون های خوشه ای کامل شده و کلاس ها یکپارچه شده اند.
- آزمون های در سطح سیستم کامل شده اند.

تست های فصل ۱۷: اصول و مفاهیم تحلیل شیء گرا

۱- توصیف کلی مجموعه‌ای از اشیای مشابه ... نام دارد.

الف) کلاس ب) نمونه ج) زیر کلاس د) زیر کلاس

۲- مقادیری که به صفات اشیا نسبت داده می‌شود آن شی را یکتا می‌کند:

الف- درست ب- نادرست

۳- عملیات، رویه‌های اشیا هستند که وقتی شی پیغام دریافت می‌کند فعال می‌شوند:

الف- درست ب- نادرست

۴- کدام یک از موارد زیر جزء مزیت‌های عمده معماری شی‌گرا نیست:

الف) آسانی استفاده مجدد از مولفه‌ها ب) پیشرفت کارایی و اجرایی

ج) پنهان‌سازی اطلاعات د) واسطهای ساده شده

۵- وراثت مکانیزمی است که بوسیله آن تغییرات در سطوح پایین سریعاً می‌توانند در سیستم پخش شوند:

الف) درست ب) نادرست

۶- کدام یک از گزینه‌های زیر باید بعنوان اشیای کاندید در فضای مساله دیده شوند:

الف) رویدادها ب) افراد ج) ساختارها د) هرسه

۷- صفات بوسیله آزمایش دیکشنری داده‌ها و مشخص کردن موجودیت‌های مربوط، انتخاب می‌شوند:

الف) درست ب) نادرست

۸- کدام یک از گزینه‌های زیر جزء دسته‌هایی که برای طبقه‌بندی عملیات بکار می‌روند نیست:

الف) محاسبه ب) عملیات در داده‌ها ج) event monitors د) transformer

۹- پروژه‌های شی‌گرا احتیاج به مدیریت و برنامه‌ریزی کمتری نسبت به پروژه‌های سنتی دارند.

الف) درست ب) نادرست

۱۰- کدام یک از گزینه‌های زیر از خصوصیات اصلی شی‌گرایی نیست:

الف) بسته‌بندی ب) وراثت ج) چسبندگی د) چند ریختی

۱۱- کدام گزینه از مراحل زیر مرحله مهمی از توسعه یک سیستم شی‌گراست:

الف) پایان تحلیل شی‌گرا ب) پایان طراحی شی‌گرا ج) پایان برنامه‌نویسی شی‌گرا د) هرسه

۱۲- چون هدف عمده سیستم‌های شی‌گرا، قابلیت استفاده مجدد است LOC معیار خوبی برای این پروژه‌هاست:

الف) درست ب) نادرست

فصل ۱۸ : مدلسازی تحلیل شیء گرا

هدف ازمدلسازی تحلیل شیء گرا چیست؟

وقتی قرار است محصول یا سیستم جدیدی ایجاد شود، چگونه آن را به نحوی مشخص کنیم که بتواند به صورت شیءگرا مهندسی شود؟ آیا پرسش‌هایی وجود دارند که باید از مشتریان پرسیده شوند؟ اشیاء چگونه با یکدیگر ارتباط دارند؟ اشیاء در حیطهٔ سیستم چگونه رفتار می‌کنند؟ چگونه مساله‌ای را مشخص یا مدلسازی کنیم تا بتوانیم یک طراحی کارآمد ایجاد کنیم؟

هر یک از این پرسش‌ها در حیطهٔ تحلیل شیءگرا (Object Oriented Analysis- OOA)- نخستین فعالیت تکنیکی که در مهندسی نرم‌افزار شیگرا (Object Oriented-OO) اجرا می‌شود- پاسخ داده می‌شود.

OOA به جای بررسی یک مساله با استفاده از مدل جریان اطلاعات کلاسیک، چند مفهوم جدید را معرفی می‌کند.

OOA ریشه در مجموعه‌ای از اصول بنیادی دارد که در فصل ۱۱ معرفی شد. برای ساخت یک مدل تحلیل، پنج اصل بنیادی به کار برده می‌شود:

۱. دامنه اطلاعاتی مدلسازی می‌شود؛

۲. عملکرد توصیف می‌شود؛

۳. رفتار نمایش داده می‌شود؛

۴. مدل‌های داده‌ای، عملیاتی و رفتاری افزار می‌شوند تا جزییات بیشتری در معرض دید قرار گیرند، و

۵. مدل‌های اولیه، بنیاد و ماهیت مساله را نشان می‌دهند، حال آنکه مدل‌های نهایی، جزییات ساده‌ای را نمایش می‌دهند.

این اصول، مبنای روش OOA را تشکیل می‌دهند.

هدف OOA تعریف کلیه کلاس‌هایی است که به نوعی با مساله ارتباط دارند- عملیات و صفات مرتبط با آنها و روابط میان آنها و رفتاری که از خود نشان می‌دهند. برای این منظور، چند کار باید صورت گیرد:

- **خواسته های اصلی کاربر باید بین مشتری و مهندس تبادل شود.**
- **کلاس‌ها باید شناسایی شوند.**
- **سلسله مراتب کلاس‌ها باید مشخص شود.**
- **روابط شیء با شیء باید نشان داده شود.**
- **رفتار اشیا باید مدلسازی شود.**

شهرت OOA

محبوبیت فنآوری‌های شیءگرا منجر به ابداع دهها روش OOA در اواخر دههٔ ۱۹۸۰ و اوایل دههٔ ۹۰ شد. هر یک از این روش‌ها معرف فرآیندی برای تحلیل یک محصول یا سیستم، یک مجموعه نمودار که از فرآیند به دست می‌آیند، و نشانه گذاری هایی است که مهندس نرم‌افزار را قادر به ایجاد مدل تحلیل می‌کند. روش های زیر طی دهه اخیرکاربرد بیشتری دارند.

روش بوچ(Booch) [BOO94]: این روش شامل یک **فرآیند توسعهٔ میکرو** و یک **فرآیند توسعهٔ ماکرو** است. سطح میکرو، مجموعه‌ای از وظایف تحلیل را تعریف می‌کند که برای هر مرحله از فرآیند ماکرو دوباره اجرا می‌شود. از این رو، یک روش تکاملی صورت می گیرد. در فرآیند توسعهٔ میکرو بوچ برای OOA، کلاس ها، اشیاء و معانی آنها تعیین می‌شود؛ روابط میان کلاس ها و اشیاء تعیین می‌گردد و پالایش های لازم صورت می‌پذیرد تا مدل تحلیل شناسایی شود.

روش رومبو و همکاران(Rambaugh) [RUM91]: رومبو و همکاران وی تکنیک مدل‌سازی شیءگرا (OMT) را برای تحلیل، طراحی سیستم و طراحی در سطح اشیاء توسعه دادند. نتیجه فعالیت تحلیل، سه مدل است: مدل اشیاء (نمایشی از اشیاء، کلاس ها، سلسله مراتب و روابط)، مدل پویا (نمایشی از رفتار شیء و سیستم) و مدل عملیاتی (نمایشی سطح بالا و به شکل DFD از جریان اطلاعات در سیستم).

روش جاکوبسون(Jacobson). این روش که آن را OOSE (مهندس نرم‌افزار شیءگرا) نیز می‌نامند، نسخه ساده‌ای از یک روش شیءگرای قدیمی‌تر است که آن را نیز جاکوبسون ابداع کرد. تفاوت این روش با روش های دیگر در تأکید فراوان آن بر موارد کاربرد(Use Case) است- یعنی شرح یا سناریویی که چگونگی تعامل کاربر با محصول یا سیستم را تصویر می‌کند.

روش کود و یوردون(Coad & Yourdon) [COA91]: این روش غالباً به عنوان یک از آسانترین روش های OOA در نظر گرفته می‌شود. نشانه‌گذاری مدلسازی نسبتاً ساده است و دستورالعمل های توسعه مدل تحلیل، صریح هستند. فرآیند OOA کود و یوردون را به اختصار در زیر مطرح می‌کنیم:

- شناسایی اشیاء با استفاده از ملاک های ((جستجو))
- تعریف یک ساختار تعمیم- تعیین مشخصات
- تعریف ساختار کامل مؤلفه
- شناسایی موضوع‌ها (نمایشی از مؤلفه‌های زیرسیستم)
- تعریف صفات
- تعریف سرویس ها

روش ویرف- بروک(Wriffs & Brok) [WIR90]: ویرف- بروک بین وظایف طراحی و تحلیل، تفاوت چندانی قایل نمی‌شوند. در عوض، فرآیندی پیوسته را پیشنهاد می‌کنند که با ارزیابی مشخصات مشتری آغاز می‌شود و با طراحی پایان می‌یابد. وظایف مرتبط با تحلیل در این روش به اختصار به شرح زیر می باشد:

- ارزیابی مشخصات مشتری
- استخراج کلاس های نامزد از روی مشخصات، از طریق تجزیه گرامری
- گروه‌بندی کلاس ها به نیت شناسایی کلاس های پایه
- تعریف مسوولیت ها برای هر کلاس
- نسبت دادن مسوولیت ها به هر کلاس
- تعیین روابط میان کلاس ها
- تعیین مشارکت میان کلاس ها براساس مسوولیت
- ساخت نمایش های سلسله مراتبی از کلاس ها
- تولید گراف مشارکت برای سیستم

گرچه مراحل، اصطلاح شناسی و فرآیندهای هر یک از این روش های OO متفاوت است، ولی فرآیندهای کلی OOA بسیار مشابهند. مهندس نرم‌افزار برای اجرای تحلیل شیءگرا باید مراحل کلی زیر را دنبال کند:

1. **روشن کردن خواسته‌های مشتری برای سیستم**
2. **شناسایی سناریو یا موارد کاربرد**
3. **انتخاب کلاس ها و اشیاء با استفاده از نیازها(خواسته‌ها) به عنوان یک راهنما**
4. **شناسایی صفات و عملیات مربوط به هر یک از اشیای سیستمی**
5. **تعریف ساختارها و سلسله مراتبی که کلاس ها را سازماندهی می‌کنند.**

۶. **ساخت یک مدل شیء- واسط**
۷. **ساخت یک مدل شیء- رفتار**
۸. **بازبینی مدل تحلیل OO نسبت به موارد کاربرد/ سناریوها**

این مراحل کلی را با جزییات بیشتری در ادامه این فصل ارایه شده است.

چرا از مدل شیء گرا استفاده می کنیم؟

۱. **روش یکنواختی برای تحلیل شیء گرا.**
۲. **استفاده از تحلیل دامنه.**
۳. **استفاده مجدد.**

منابع آگاهی از دامنه

۱. ادبیات فنی
۲. نرم افزارهای کاربردهای موجود
۳. تحقیق از مشتری
۴. وسایل هوشمند و خبره
۵. شناسایی نیازمندی های فعلی و آینده

مدل تحلیل دامنه

۱. شناسایی انواع کلاس ها
۲. داشتن استانداردهای استفاده مجدد
۳. استخراج مدل های عملیاتی
۴. دامنه های مورد نظر

مراحل فرآیند تحلیل دامنه

۱. **تعریف و تعیین دامنه ای که باید مورد بررسی قرار گیرد.**
۲. **گروه بندی عناصر استخراج شده از دامنه.**
۳. **جمع آوری نمونه ای از کاربردهای موجود در دامنه که نماینده کل دامنه باشد.**
۴. **تحلیل هر کاربرد در نمونه.**
   - شناسایی اشیای کاندیدا برای استفاده مجدد.
   - ذکر دلایل برای انتخاب شیء جهت استفاده مجدد.
   - تعریف تطابق هایی برای شیء که ممکن است بعدا استفاده شود.
   - برآورد درصدی از کاربردهای موجود در دامنه که ممکن است از شیء استفاده کنند.
   - شناسایی اشیاء از طریق نام و استفاده از تکنیک های مدیریت پیکربندی برای مدیریت آنها.

مولفه های عمومی در مدل تحلیل شیء گرا

۱. نمای ایستا از کلاس های معنایی
۲. نمای ایستا از صفات
۳. نمای ایستا از روابط

۴. نمای ایستا از رفتارها
۵. نمای پویا از رفتارها
۶. نمای پویا از ارتباطات
۷. نمای پویا از کنترل و زمان

مدلسازی مسوولیت و مشارکت کلاس ها

کلاسها

خصوصیات کلاسهای گزینشیء

۱. اطلاعات نگهداری شده
۲. اطلاعات مورد نیاز
۳. صفات چندگانه
۴. صفات متداول
۵. عملیات متداول
۶. خواسته های اساسی

انواع کلاس ها

۱. کلاس های دستگاهی
۲. کلاس های خواص
۳. کلاس های تعامل
۴. عینی بودن
۵. شمول
۶. دوام
۷. آسیب پذیری

مسئولیت ها

۱. هوشمندی سیستم باید از توزیعی مناسب برخوردار باشد.
۲. هر مسوولیتی را باید هر چه کلی تر بیان کرد.
۳. اطلاعات و رفتاری که به آن مربوط می شود باید در یک کلاس قرار گیرد.
۴. اطلاعات مربوط به یک چیز باید در یک کلاس متمرکز شود.
۵. مسوولیت ها باید در صورت امکان بین کلاس ها پخش شود.

مشارکت کننده ها

کلاس ... از کلاس ... آگاه است.
کلاس ... بخشیء از کلاس ... است.
کلاس ... به کلاس ... وابسته است.

تعریف ساختارها و سلسله مراتب

تعریف زیر سیستم

- **مدل روابط میان اشیاء**
- **مدل رفتار اشیاء**
- **شناسایی رویدادها با موارد کاربرد**
- **نمایش حالت ها**

مروری بر زبان مدلسازی یکنواخت(UML-Unified Modelling Language)

زبان یکنواخت و یکپارچه مدلسازی(UML) زبان مدلسازی گرافیکی است که با ترکیبی از نمادها برای تشریح عناصر اصلی سیستم های نرم افزاری (که در UML به آن محصول گفته می شود) تهیه گردیده است.

UML شامل سه مدل به شرح زیر است:

۱. مدل تابعی(Functional Model): عملکرد سیستم را از دید کاربر شرح می دهد. در UML برای این مدلسازی از نمودارهای Use-Case استفاده می شود.
۲. مدل شیء (Object Model): ساختار سیستم را به صورت مجموعه ای از اشیا، صفات، عملیات و ارتباطات شرح می دهد. درUML برای این مدلسازی از نمودار کلاس(Class Diagram) استفاده می شود.
۳. مدل پویا(Dynamic Model): رفتار داخلی سیستم را شرح می دهد. درUML برای این مدلسازی از نمودارهایِ Sequence, Statechart,Activity استفاده می شود.

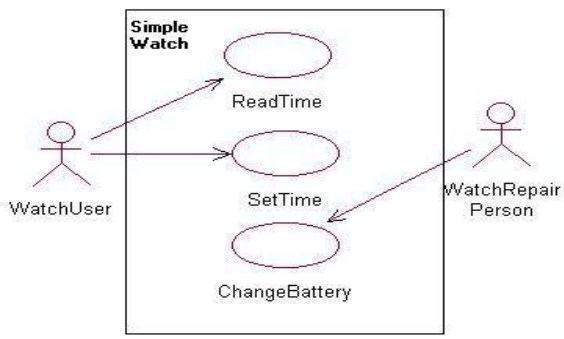
نمودار مورد کاربرد(Use Case Diagram)

نمودار Use Case در طول استخراج نیازها و تحلیل سیستم برای مشخص کردن عملکرد برنامه به کار می رود.

Use Case ها روی رفتار سیستم از یک دیدگاه خارج از سیستم تمرکز می کنند.

یک Use Case در واقع عملیاتی را شرح می دهد که توسط سیستم تهیه شده و نتیجه ای مشخص برای یک بازیگر(Actor) دارد.

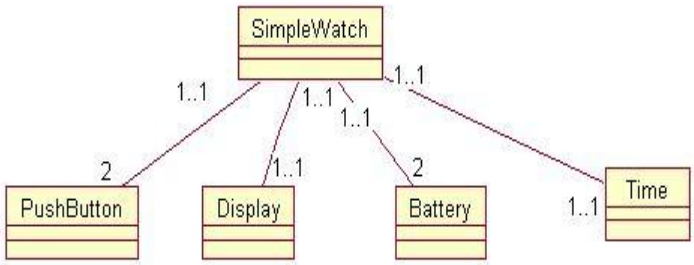
بازیگر یک **موجودیت** است که با **سیستم** در حال تعامل می باشد.(**کاربر ، یک سیستم** دیگر و ...) نمونه ای از نمودار مورد کاربرد در شکل ۱ نشان داده شده است. در این مورد کاربرد Watch User و WatchRepair Person دو بازیگر هستند که با سیستم در تعامل می باشند.



شکل ۱: نمودار یک مورد کاربرد

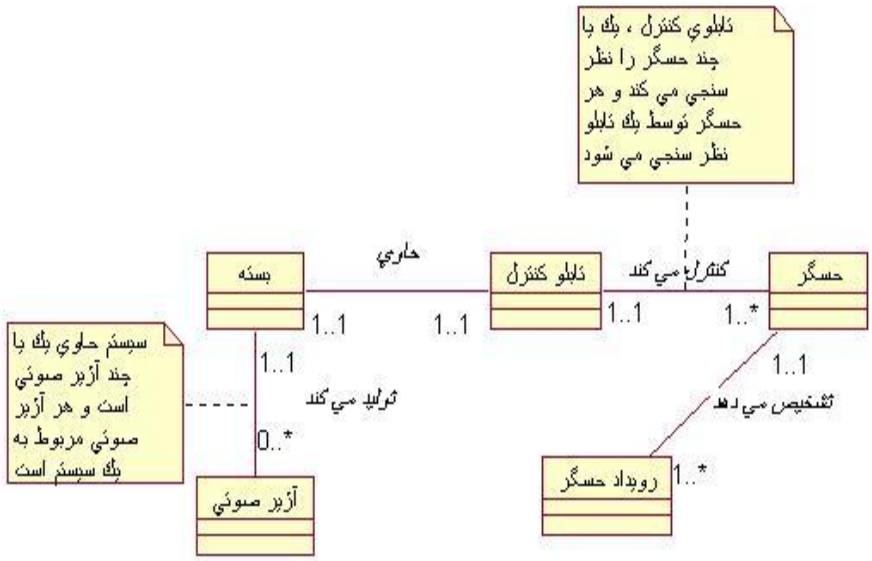
نمودار کلاس(Class Diagram)

از نمودار کلاس برای نشان دادن ساختار سیستم استفاده می شود. کلاس ها انتزاعی از ساختارهای مشترک و رفتار مجموعه ای از اشیا می باشند. اشیا نمونه هایی از کلاس ها می باشند که در حین اجرای سیستم ساخته می شوند، تغییر می کنند و غالبا بعد از پایان اجرای سیستم از بین می روند. شکل ۲ نیز نمونه ای از نمودار کلاس را برای مورد کاربرد نشان داده شده در شکل ۱ نشان می دهد.



شکل ۲: نمودار سلسله مراتبی کلاس ها

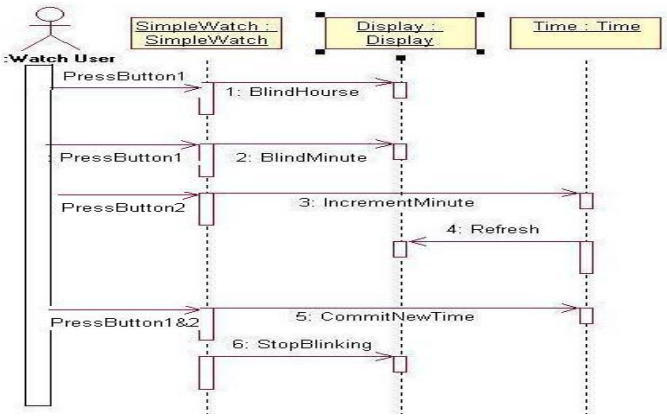
مجموعه ای از کلاس های مرتبط با ارتباطات نامگذاری شده و چندی های مشخص



شکل ۳: نمودار کلاس ها به همراه چندی بین انها

نمودار توالی(Sequence Diagram)

نمودار توالی برای فرموله کردن رفتار سیستم و همچنین واضح کردن ارتباطات بین اشیا به کار می رود. نمودار توالی برای نشان دادن اشیایی که در یک Use Case سهیم هستند مفید است. به همین دلیل به اشیایی که در یک نمودار توالی نشان داده می شوند اشیا شرکت کننده(Participating Object) می گویند. در شکل ۴ نمونه ای از نمودار توالی رسم شده است.

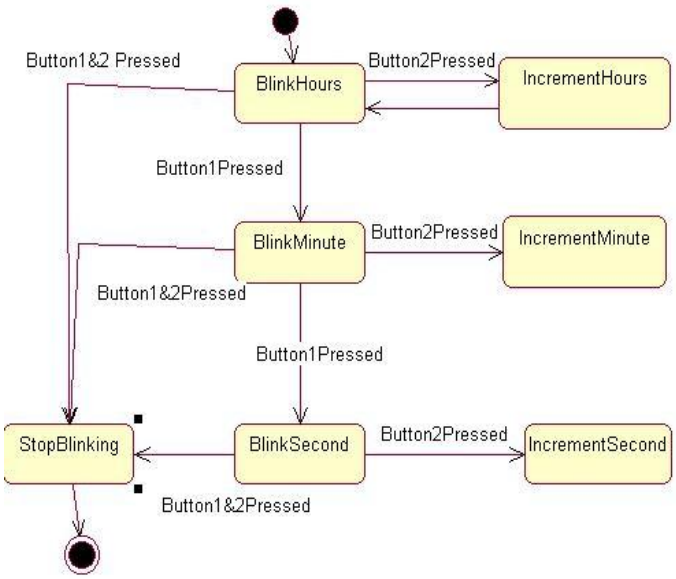


شکل ۴: نمودار کلاس ها به همراه چندی بین آنها

نمودار حالت(StateChart Diagram)

نمودار حالت رفتار یک شیء مشخص را با تعدادی حالت(State) و حرکت بین این حالات شرح می دهد. یک شیء در یک حالت، مقادیر مشخصی برای ویژگی های خودش دارد. منظور از حرکت یعنی رفتن شیء از یک حالت به حالت دیگری با روی دادن یک رویداد خاص می باشد.

در نمودار حالت تمرکز روی پیغام‌هایی است که تحت تاثیر یک رویداد بوجود آمده توسط بازیگر بین اشیا مختلف مبادله می شود اما در نمودار حالت تمرکز بر انتقال بین حالت ها می باشد که این انتقال ناشی از روی دادن یک رویداد برای آن شیء خاص می باشد. شکل ۵ را ببینید.



شکل ۵: نمودار کلاس‌ها به همراه چندی بین آنها

اکنون در ادامه بحث فرایند تحلیل شیء گرایی را با مثال سیستم خانه امن یکبار دیگر ادامه می دهیم.

فرایند تحلیل شیء‌گرا (OOA Process)

فرایند تحلیل شیء‌گرا در آغاز کاری با خود اشیا ندارد و ابتدا با درک شیوه استفاده از سیستم توسط کاربران(انسان، ماشین و یا برنامه های دیگر) آغاز می شود .

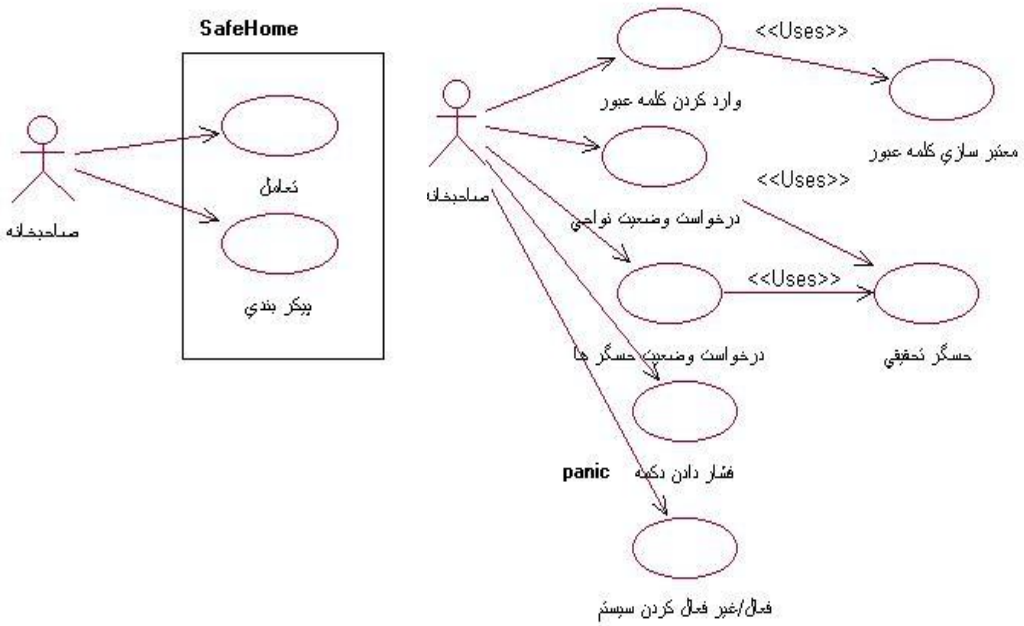
تکنیک‌هایی برای جمع آوری خواسته‌های مشتری و سپس تعریف یک مدل تحلیل برای یک سیستم شیء‌گرابه شرح زیر وجود دارد:

- **Use-Case ها**
- **مدلسازی مسوولیت و مشارکت کلاس ها(Class-Responsibility-Collaborator Modelling)**
- **تعریف ساختارها و سلسله مراتب(Defining Structurers and Hierachies)**
- **تعریف موضوع و زیر سیستم ها(Defining Subjects and Subsystems)**

موارد کاربرد(Use-Case)

مورد کاربرد یکی از نمودارهایUML می باشد که در واقع شرحی از مجموعه تعاملات بین کاربران و سیستم است. Use-Case سیستم را از دیدگاه کاربر نهایی مدلسازی می کند. Use-Caseها باید اهداف زیر را برآورده سازند:

- تعریف خواسته های عملیاتی سیستم با تعریف یک سناریوی کاربرد که مشتری و تیم مهندسی نرم‌افزار در مورد آن به توافق رسیده اند.
- فراهم آوردن توصیفی واضح و عاری از ابهام از چگونگی تعامل سیستم و کاربر نهایی با یکدیگر
- فراهم آوردن مبنایی برای انجام آزمون‌های اعتبار سنجی



شکل ۶: نمودار مورد کاربرد سیستم خانه امن

مدلسازی مسولیت ومشارکت کلاس‌ها

مدلسازی مسوولیت و مشارکت کلاس‌ها(CRC) وسیله‌ای ساده برای شناسایی وسازماندهی کلاس‌های مرتبط با خواسته های سیستم فراهم می آورد.

مراحل کار:

- **شناسایی کلاس‌ها**
- **شناسایی مسوولیت‌ها (صفات و عملیات)**
- **شناسایی مشارکت کننده‌ها**

شناسایی کلاس‌ها

شش خصوصیت برای شناسایی کلاس‌های بالقوه تعریف می‌شود:

۱- اطلاعات نگهداری شده

۲- اطلاعات مورد نیاز

۳- صفات چندگانه

۴- صفات مشترک

۵- عملیات مشترک

۶-خواسته‌های اساسی

انواع کلاسها

۱- کلاس‌های دستگاهی (Device Classes)

۲- کلاس‌های خواص (Property Classes)

۳- کلاس‌های تعامل (Interaction Classes)

ویژگیهای یک کلاس

۱- عینی یا انتزاعی

۲- اتمی یا مجتمع

۳- ترتیبی یا غیر ترتیبی

۴-گذرا ، موقت یا دائمی

۵-جامعیت کلاس

شناسایی مسوولیت‌ها

پنج دستور العمل برای تخصیص مناسب مسوولیت‌ها به کلاس‌ها به شرح زیر می‌باشد:

۱- توزیع مناسب هوشمندی سیستم

۲- بیان کلی مسولیت‌ها

۳- قرار گرفتن اطلاعات و رفتار مربوط به هر کلاس در همان کلاس

۴- متمرکز شدن اطلاعات مربوط به هم در یک کلاس

۵- به اشتراک گذاشتن مسوولیت‌ها بین کلاس‌های مرتبط

شناسایی مشارکت کننده‌ها

برای شناسایی مشارکت کننده‌ها سه رابطه کلی میان کلاس‌ها بررسی می شود:

۱- رابطه بخشی از .... است.

۲- رابطه از .....آگاه است.

۳- رابطه وابسته به ...است.

کارت شاخص

کارت‌هایی که برای نشان دادن کلاس‌ها به کار می روند و دارای سه بخش هستند:

۱- نام کلاس در بالای کارت

۲- لیستی از مسوولیت‌های کلاس در طرف چپ

۳- لیستی از مشارکت‌کننده ها در طرف راست

نمونه‌ای از یک کارت شاخص در شکل نشان داده شده است.

	نام کلاس:	
	نوع کلاس (مثلا دستگاهی ، خواص ، نقش ، رویداد)	
	خصوصیت کلاس (مثلا ملموس ، اتمی ، همزمان و ..)	
	مسئولیتها:	مشارکت کننده ها :

شکل ۷: نمونه‌ای از یک کارت CRC

بررسی مدل (CRC)

مراحل کار برای تولید مدل CRC به صورت زیر خواهد بود:

۱- به همه شرکت کنندگان در بررسی، زیر مجموعه‌ای از کارت‌های شاخص مدل CRC داده می‌شود.

۲- همه سناریوهای Use-Case باید گروه بندی شوند.

۳- شخص مسوول بررسی، Use-Case ها را به دقت می خواند و همین که به نام یک کلاس رسید نوبت را به کسی می دهد که کارت شاخص کلاس مربوطه در دست اوست.

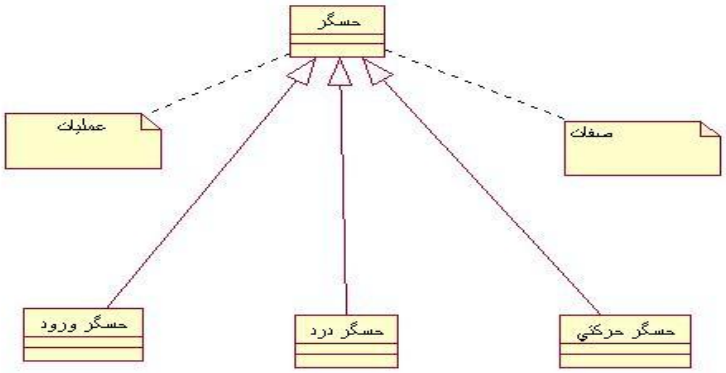
۴- هنگام تحویل کارت‌ها از دارنده کارت خواسته می شود تا مسوولیت‌های ذکر شده در کارت را شرح دهد و گروه بررسی می کندکه آیا تمام خواسته های Use-Case برآورده می‌شود یا نه؟

۵- اگر مسوولیت‌ها و مشارکت‌های ذکر شده روی کارت‌های شاخص نتوانند جوابگوی نیازهای موارد کاربردها(Use-Cases) باشند انجام اصلاحات در کارت‌ها ضروی است و این کار انجام می‌گیرد.

تعریف ساختارها و سلسله مراتب کلاس‌ها

۱- ساختار کلاس تعمیم تخصیص(Is A)

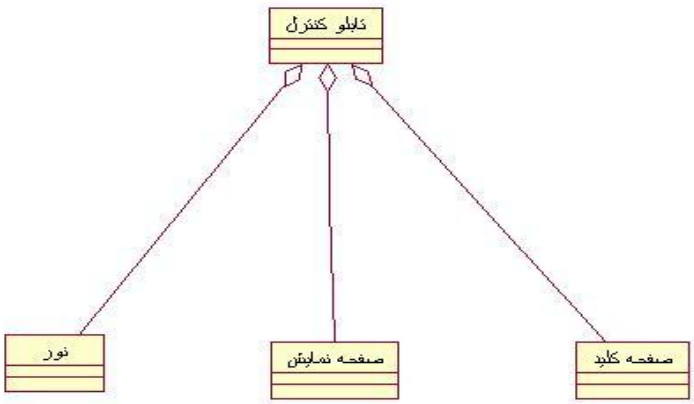
در این ساختار یک کلاس به عنوان زیر کلاس کلاس دیگر مطرح می شود و تمام صفات و عملیات‌های آن را به ارث می برد. مثال:



شکل ۸: نمودار کلاس‌ها برای تعمیم- تخصیص

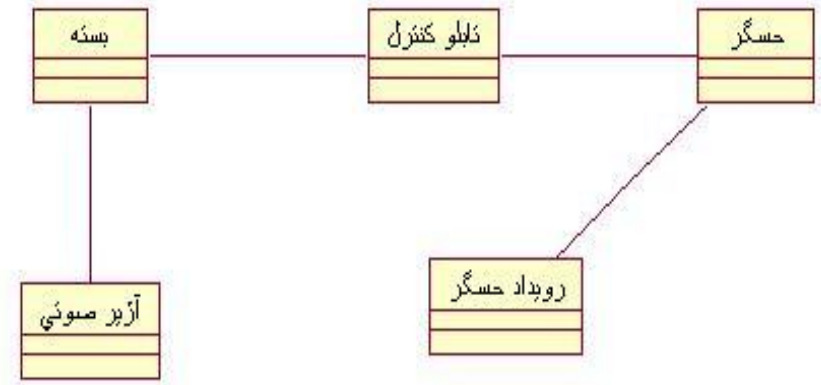
۲- ساختار کلاس مجتمع مرکب(Is A Part Of)

در این ساختار یک شیء از چند مولفه تشکیل شده است که این مولفه‌ها را می توان به عنوان یک شیء تعریف کرد. مثال:



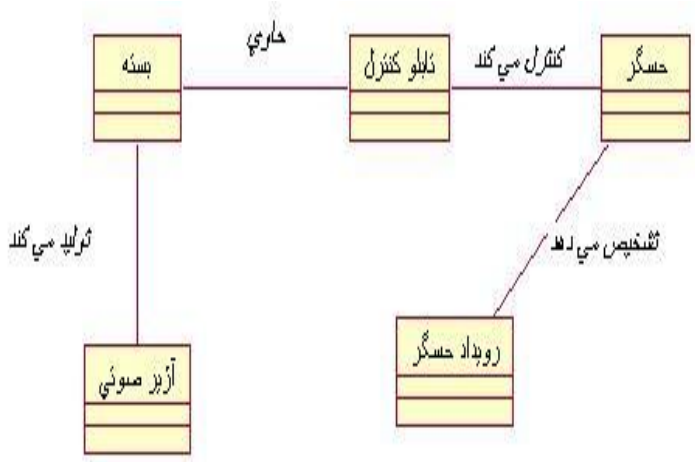
شکل ۹: نمودار کلاس‌ها برای کلاس‌های مجتمع مرکب

مجموعه ای از کلاس‌های مرتبط



شکل ۱۰: نمودار کلاس‌های مرتبط

مجموعه ای از کلاس‌های مرتبط با ارتباطات نامگذاری شده



شکل ۱۱: نمودار کلاس‌های مرتبط با تعریف ارتباطات

تعریف زیر سیستم (SubSystem)

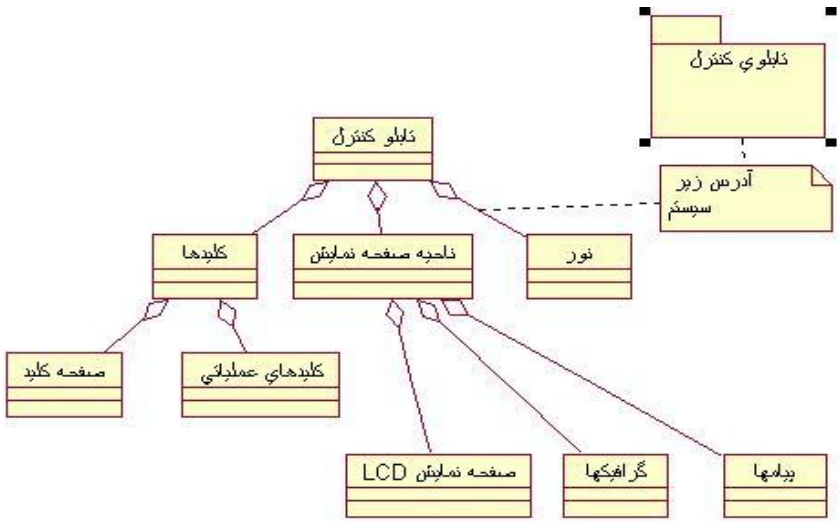
هرگاه گروهی از کلاس‌ها با یکدیگر مشارکت کنند تا مجموعه‌ای از مسوولیت‌های منسجم را بوجود آورند، می‌توان آنها را به عنوان یک زیر سیستم در نظر گرفت.

کارت شاخص زیر سیستم

شامل نام زیر سیستم ، توافقنامه های مربوطه و کلاس‌ها و یا زیر سیستم‌های دیگری که باید توافق را پشتیبانی کنند.

بسته (Package)

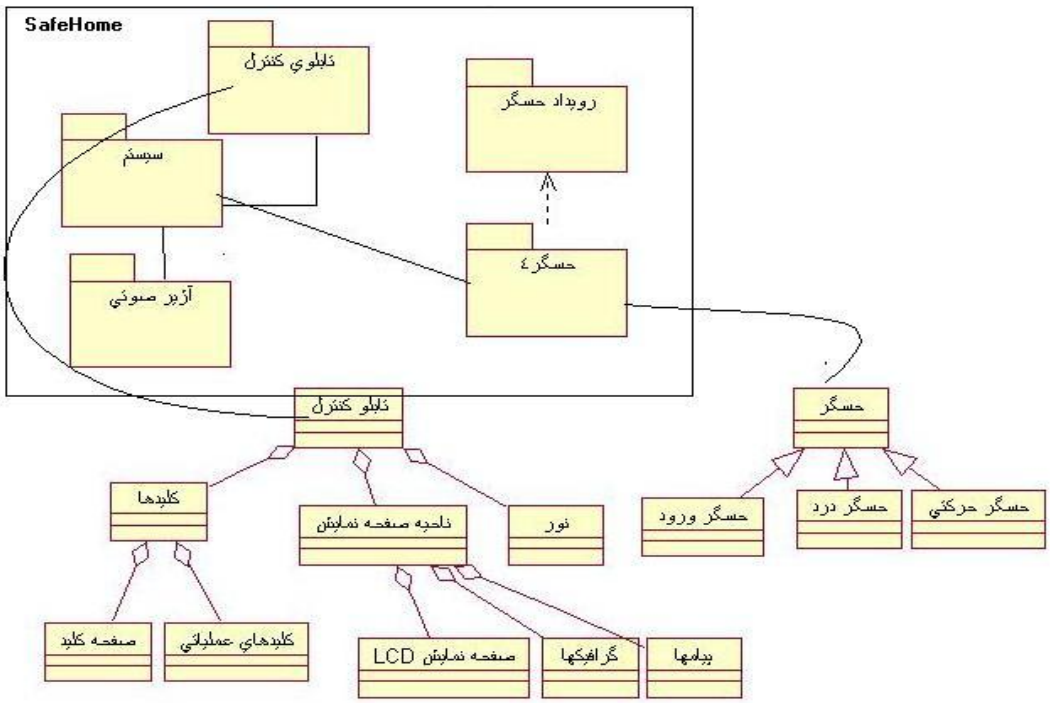
نمایش گرافیکی زیر سیستم در UML است.



شکل ۱۲: نمودار زیر سیستم و بسته در سیستم خانه امن

مدل OOA در انتزاعی ترین سطح

مدل OOA در بالاترین سطح انتزاع فقط شامل بسته‌ها است. هریک از آدرس‌ها به یک ساختار بسط می‌یابد. شکل را ببینید.



شکل ۱۲: نمودار زیر سیستم و بسته در سیستم خانه امن

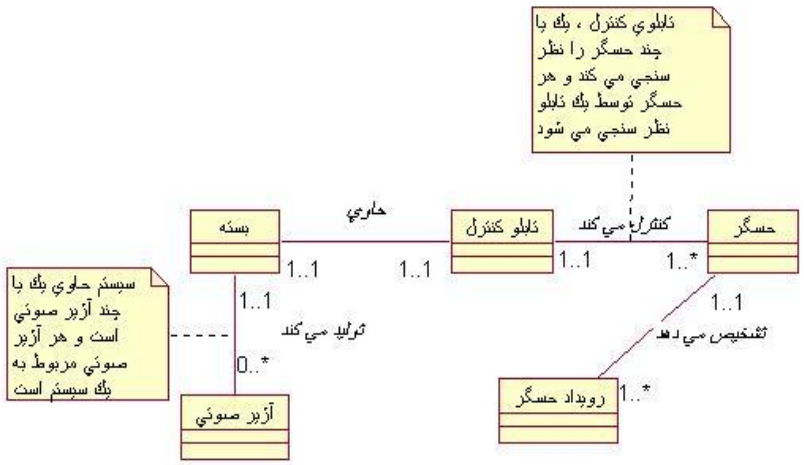
مدل روابط بین اشیاء

مدل روابط میان اشیا در سه مرحله به دست می آید:

۱- با استفاده از کارت‌های شاخص CRC مجموعه ای از اشیای مشارکت کننده رسم می شود.

۲- با بازبینی کارت‌های شاخص مدل CRC مسوولیت‌ها و مشارکت‌کننده ها مورد ارزیابی قرار گرفته و ارتباطات نامگذاری می شوند.

۳- هنگامی که روابط با نام مشخص شد نحوه مشارکت کلاس‌ها در ارتباط ارزیابی می شود. شکل زیر را ببینید.



شکل ۱۳: نمودار روابط بین اشیاء

مدل رفتار اشیا (Object –Behavior Model)

مدل رفتار اشیا نشان می دهد که سیستم شی‌گرا چگونه به رویداد ها و محرک‌های خارجی پاسخ خواهد داد.

مراحل ایجاد مدل

۱- ارزیابی کلیه Use-Case ها برای درک کامل ترتیب تعامل‌ها در سیستم

۲- شناسایی رویدادهایی که ترتیب تعامل‌ها را هدایت می کنند و درک چگونگی ارتباط این رویداد ها با اشیا خاص

۳- ایجاد یک پیگرد رویداد برای هر Use-Case

۴- ساختن یک نمودار گذار حالت برای سیستم

۵- بازبینی مدل رفتار اشیا به منظور اعتبار سنجی، صحت و سازگاری

مثال SafeHome

۱- صاحبخانه به تابلوی کنترل نگاه می کند تا تعیین کند که آیا سیستم آماده دریافت ورودی هست یا خیر . اگر سیستم آماده نباشدصاحبخانه باید از نظر فیزیکی درها یا پنجره ها را ببنددتا نشانگر آمادگی ارائه شود.

۲- صاحبخانه با استفاده از صفحه کلید ، کلمه عبور چهار رقمی را وارد می کند . این کلمه با کلمه عبور موجود در سیستم مقایسه می شود. اگر کلمه عبور نادرست باشد ، تابلوی کنترل بوق می زند و آماده ورودی بعدی خواهد شد .اگر کلمه عبور درست باشد ، منتظر فعالیتهای دیگر می ماند.

۳- صاحبخانه با استفاده از صفحه کلید ، Stay یا Away را وارد می کند تا سیستم فعال شود. Stay فقط حسگر های محیطی را فعال می کند . Away تمام حسگر ها را فعال می کند.

۴- پس از فعال سازی صاحبخانه می تواند تمام حسگرها را فعال کند.

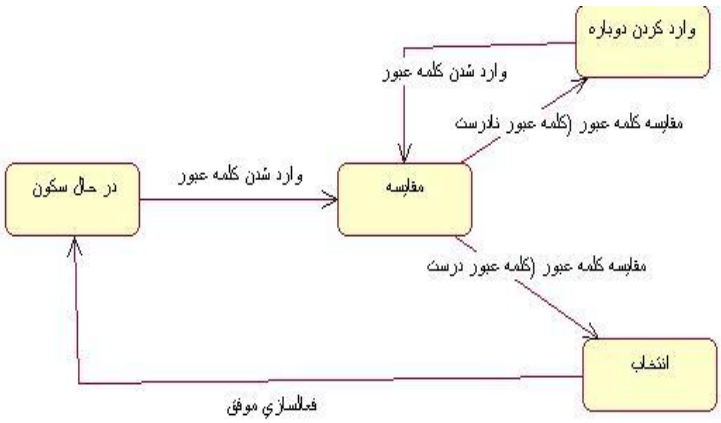
نمودار نمایش حالت (State Diagram)

برای هر سیستم شی‌گرا حالت سیستم توسط دو مورد زیر مشخص می شود:

۱- حالت هر شیء در هنگامی که سیستم در حال اجرای عملکرد خود می باشد. هر شی خود دارای دو گونه حالت می‌باشد:

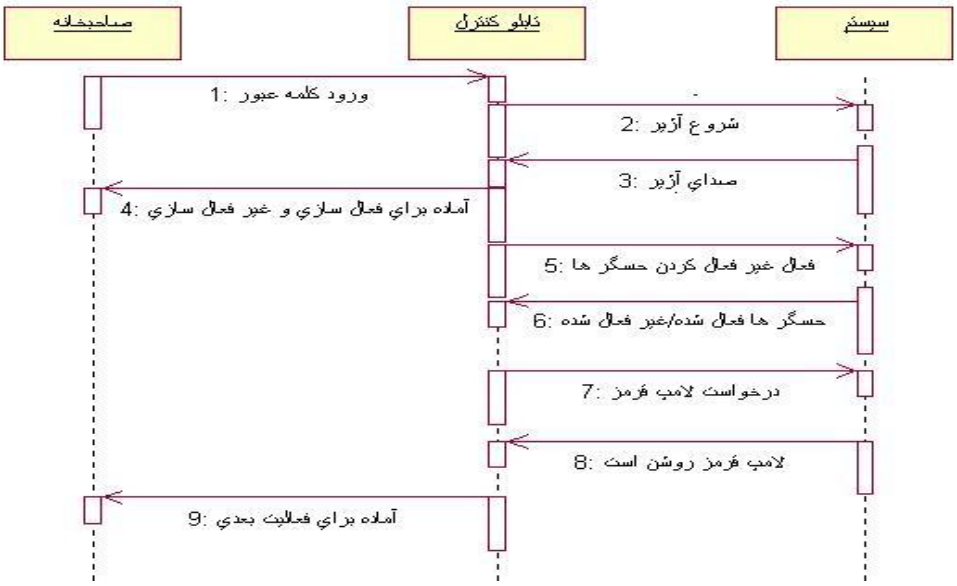
۱- حالت انفعالی

۲- حالت فعال



شکل ۱۴: نمودار انتقال حالت‌های سیستم

۲- حالت هر سیستم از دیدگاه خارجی و در حالی که سیستم عملکرد خود را اجرا می کند. نمودار ترتیب (Sequence Diagram)



تست‌های فصل ۱۸: مدلسازی تحلیل شی‌گرا

۱- کدام یک از اصول بنیادی برای تحلیل توسط کود و یوردون بکار برده می‌شوند.

الف) توصیف عملکرد ب) مدلسازی دامنه اطلاعات ج) دیده شدن جزئیات بیشتر د) تمام موارد

۲- کدام یک از گزینه‌های زیر نادرست است؟

الف) هیچ توافق جهانی بر سر مفاهیم مبنایی OOA وجود ندارد.

ب) اتصالات اشیا در OOA باید نشان داده شود.

ج) نظریه کود و یوردون بعنوان مبنای OOA است.

د) رفتار اشیا در OOA باید مدلسازی شود.

۳- چه کسی OOA را انجام می‌دهد؟

الف) مهندس نرم‌افزار ب) طراح ج) کاربر د) ۲و ۳

۴- کدام یک از گزینه‌های زیردر مورد تحلیل شی‌گرا نادرست است؟

الف) روش آن نسبت به مهندسی اطلاعات اندکی تفاوت دارد.

ب) روش آن تغییر بنیادی نسبت به تحلیل ساختیافته نشان می‌دهد.

ج) محصول آن یک مدل شی‌گرا است.

د) فرآیند توسعه میکرو روش رومبو است.

۵- کدامیک از گزینه‌های زیر از وظایف مهندس نرم‌افزار برای اجرای تحلیل شی‌گرا نیست؟

الف) ساخت یک مدل شی- رفتار ب) شناسایی موارد کاربرد

ج) ارزیابی مشخصات مشتری د) ساخت یک مدل شی- واسط

۶- فرآیند OO در کار خود به اشیا نیاز ندارد.

الف) درست ب) نادرست

۷- فرآیند OO در آغاز کار خود توسط ... (اگر در کنترل فرآیند بکار گرفته شود.) شروع می‌شود.

الف) انسان ب) ماشینها ج) بعضی برنامه‌ها د) هر ۳ مورد

۸- در اثنای تحلیل خواسته‌ها باید به اطلاعات اصلی تاکید شود.

الف) درست ب) نادرست

۹- اطلاعات مربوط به یک چیز باید در میان چند کلاس توزیع شوند.

الف) درست ب) نادرست

۱۰-نمایش ساختار افراز، CRC را در اختیار تحلیلگر قرار می‌دهد.

الف) درست ب) نادرست

۱۱-پکیج ....

الف) از لحاظ هدف مانند زیرسیستم است. ب) در UML نشان داده نمی‌شود.

ج) از لحاظ محتویات مانند سیستم است. د) همان سیستم است.

۱۲- حسگر به وضعیت ... وابسته است.

الف) پکیج ب) سیستم ج) حسگرها د) الف و ج

۱۳-ساختن یک نمودار گذرا برای سیستم وظیفه کیست.

الف) مهندس نرم‌افزار ب) طراح ج) تحلیل‌گر د) کاربر

۱۴- UML .....

الف) برای نشان دادن برخی از رفتار پویای اشیا و کلاس‌هایی است.

ب) تلفیقی از نمودارهای حالت، مشارکت، ترتیب و فعالیت است.

ج) زبانی برای مدلسازی است.

د) هر ۳ مورد

۱۵- مدل رفتار اشیا رفتار کلی سیستم OO است.

الف) درست ب) نادرست

فصل ۱۹: طراحی شیء گرا(OOD-Object Oriented Design)

مدل تحلیل ایجاد شده با استفاده از تحلیل شیء گرا را به یک مدل طراحی تبدیل می کند که بـه عنـوان نقشـه راهنمـایی برای ساخت نرم افزار عمل می کنـد. بـا ایـن وجـود، وظیفـه مهنـدس نـرم افـزار مـی توانـد دشـوار باشـد. Gamma [GAM95] و همکاران وی تصویر خوبی از OOD را ارایه می دهند:

طراحی نرم افزارهای شیء گرا دشوار است و طراحی نرم افزار شیء گرا با قابلیت اسـتفاده مجـدد از آن هـم دشـوارتر. بایـد اشیای مرتبط را بیابید، آنها را در کلاس‌هایی مناسب دسته بندی کنید و روابط کلیدی میان آنها را مشـخص کنیـد. طراحـی شما باید مختص مساله مورد نظر باشد، ولی در عین حال آنقدر عمومیت داشته باشد که مسایل و خواسته های آینـده را نیـز پاسخگو باشد. باید از طراحی دوباره پرهیز کنید یا حداقل آن را به کمترین میزان برسانید. طراحان شیء گـرای کـار آزمـوده معتقدند که دستیابی به طراحی انعطاف پذیر و قابل استفاده مجدد، برای بار اول اگر غیر ممکن نباشد، بسـیار دشـوار اسـت. پیش از آن که طراحی پایان یابد، معمولاً سعی می کنند از آن چند بار استفاده به عمل آورند و آن را هر بار اصلاح کنند.

برخلاف مدل‌های طراحی نرم افزار سنتی، OOD منجر به یک طراحی می شود که شامل سـطوح متفـاوتی از پیمانـه هـا است. مولفه های سیستم اصلی به صورت زیرسیستم یا **پیمانه** سطح سیستمی سازماندهی می شـوند. داده هـا و عملیـاتی که داده ها را دستکاری می کنند، در اشیاء- یک شکل پیمانه ای که مولفه سازنده سیستم‌های OO است- **بسته بنـدی** می شوند. به علاوه، OOD باید سازمان داده های مربوط به صفات و جزییات رویه‌ای هر یک از عملیات را توصـیف کنـد. اینها نشانگر مولفه های داده ها و الگوریتمی سیستم OO بوده در پیمانه‌ای کردن سیستم سهم دارد.

طراحی با استفاده از روش‌های شیء گرا

در فصل ۱۳ با مفهوم **هرم طراحی برای نرم افزارهای سنتی** آشنا شـدیم. **چهار لایه طراحی شامل داده هـا، معماری، واسط و سطح مولفه‌ها** تعریف و مورد بحث قرار گرفت. برای سیستم‌های شیء گرا نیز می توان یـک هـرم طراحی تعریف کرد، ولی لایه ها در آن قدری تفاوت دارند. در شکل ۱، چهار لایه هرم طراحی OO عبارتند از:

لایه زیرسیستم: شامل نمایشی از هر یک از زیر سیستم‌هاست که نرم افزار را قادر می سازد تا به خواسـته هـای تعیـین شده توسط مشتری دست پیدا کند و زیر ساخت فنی را که خواسته های مشتری را پشتیبانی می‌کنند، پیاده سازی کنند.

لایه کلاس‌ها و اشیاء: شامل سلسله مراتبی از کلاس‌هاسـت کـه امکـان ایجـاد سیسـتم را بـا اسـتفاده از تعمـیم‌هـا و تخصصی کردن هدفمند فراهم می آورد. این لایه همچنین شامل نمایشی از کلیه اشیا است.

لایه پیغام رسانی: شامل جزییاتی از طراحی است که هر کلاس را قادر به برقراری ارتباط با مشارکت کننده هایش مـی سازد. این لایه واسطهای داخلی و خارجی را برای سیستم برقرار می سازد.

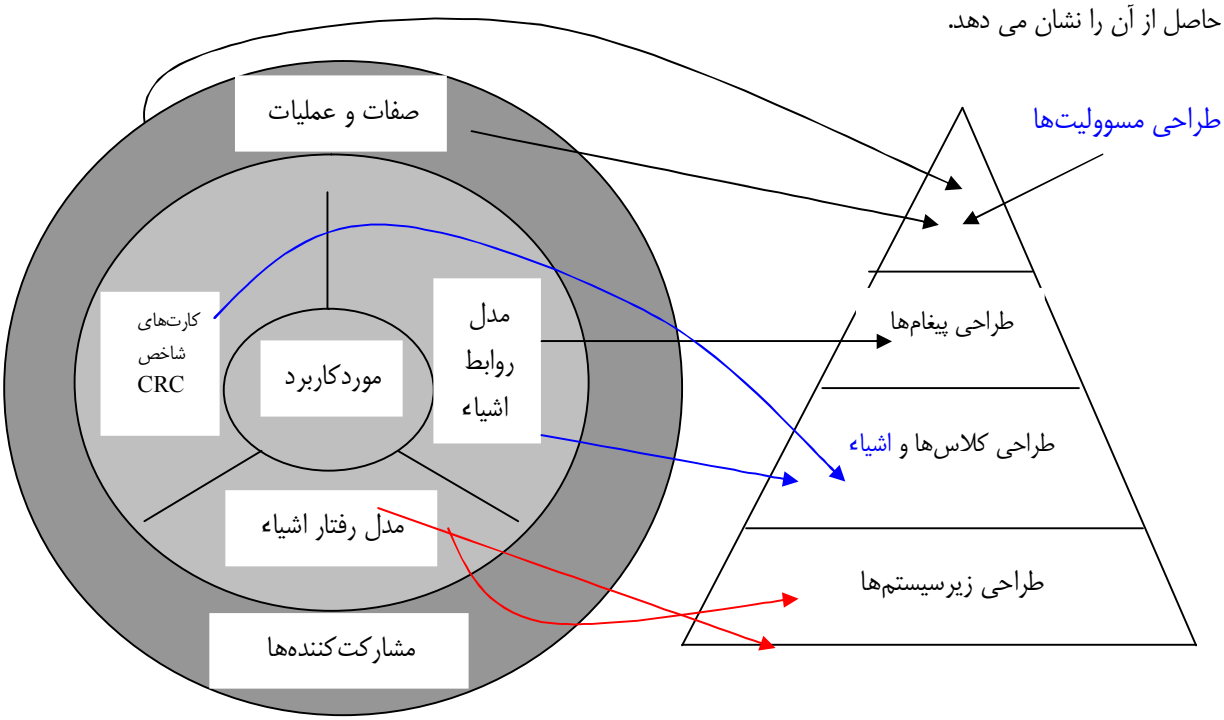
لایه مسوولیت‌ها: شامل طراحی ساختمان داده ها و الگوریتم برای کلیه صفات و عملیات مربوط به هر شیء می‌شود.

هرم طراحی انحصاراً بر طراحی محصول یا سیستمی مشخص تأکید دارد. ولی لازم به ذکر است که لایه دیگری از طراحـی نیز وجود دارد و این **لایه مبنا**یی تشکیل می دهد که هرم بر آن استوار است. لایه مبنا بر **طراحی اشـیای دامنـه** (کـه بعداً در همین فصل **الگوهای طراحی** نامیده خواهند شد) تأکید دارد. اشیای دامنه نقشی کلیدی در ایجاد زیرساختی بـرای سیستم OO ایفا می کنند. زیرا فعالیت‌های واسـط انسـان- کـامپیوتر، مـدیریت وظـایف و مـدیریت داده هـا را پشـتیبانی می‌کنند. اشیای دامنه را می توان برای افزودن اطلاعات بیشتر به خود برنامه کاربردی نیز به کار برد.

تبدیل مدل تحلیل OO به مدل طراحی OO

در OOD نیز همانند طراحی نرم افزار سنتی، هنگامی طراحی داده ها اجرا می شود که صفات موجود باشند؛ طراحی واسط زمانی صورت می پذیرد که مدل پیغام رسانی توسعه یافته باشد و طراحی در سطح مولفه ها (رویه‌ای) برای طراحی عملیات انجام می شود. لازم به ذکر است که معماری یک طراحی OO بیشتر با مشارکت میان اشیاء سر و کار دارد تا با جریان کنترل میان مولفه های سیستم.

گرچه میان مدل‌های طراحی سنتی و OO شباهت وجود دارد، تصمیم گرفتیم لایه های هرم طراحی را تغییر نام دهیم تا ماهیت طراحی OO را به طور صحیحی منعکس کند. شکل ۱ رابطه میان مدل تحلیل OO (فصل ۱۸) و مدل طراحی حاصل از آن را نشان می دهد.



شکل ۱: رابطه بین مدل‌های تحلیل و طراحی شیءگرا

طراحی زیرسیستم‌ها با در نظر گرفتن خواسته های مشتری (که در موارد کاربرد ارایه شدند) و رویدادها و حالت‌هایی به دست می آید که قابل مشاهده هستند(مدل رفتار اشیاء). طراحی کلاس‌ها و اشیاء از توصیف صفات، عملیات و مشارکت‌های موجود در مدل CRC تصویر برداری می شود. طراحی پیغام‌ها توسط مدل روابط میان اشیاء به دست می آید و طراحی مسوولیت‌ها با استفاده از صفات، عملیات و مشارکت‌های شرح داده شده در مدل CRC به دست می آید.

Fishman و Kemerer [FIC92] ده مولفه مدلسازی طراحی را پیشنهاد می کنند که می توان آنها را برای مقایسه روش‌های طراحی شیء گرا و سنتی به کار برد:

۱. نمایش سلسله مراتب پیمانه ها
۲. مشخص سازی تعاریف داده ها
۳. مشخص سازی منطق رویه ای

۴. نشان دادن دنباله ای از پردازش انتها به انتها
۵. نمایش حالت‌ها و گذارهای اشیاء
۶. تعریف کلاس‌ها و سلسله مراتب‌ها
۷. انتساب عملیات به کلاس‌ها
۸. تعریف مشروح کلاس‌ها
۹. تعیین مشخصات اتصالات پیغام رسانی
۱۰. شناسایی سرویس‌های انحصاری

مسایل طراحی

Bertrand و Meyer [MYE90] پنج ملاک برای قضاوت در خصوص توانایی روش‌های طراحی برای دستیابی به پیمانه‌ای بودن پیشنهاد می‌کند و آنها را به طراحی شیء گرا رابط می دهد:

- **تجزیه پذیری:** میزان سهولتی که روش طراحی به طراح کمک می کند تا مساله ای بزرگ را به چند مساله کوچکتر تجزیه کند که راحت تر قابل حل باشند؛
- **ترکیب پذیری.** حدی که روش طراحی اطمینان می دهد تا مولفه های برنامه (پیمانه ها) پس از طراحی و ساخته شدن، در ایجاد سیستم‌های دیگر قابل استفاده باشند؛
- **درک پذیری:** سهولت درک یک مولفه از برنامه بدون رجوع به اطلاعات دیگر یا پیمانه های دیگر؛
- **تداوم:** توانایی ایجاد تغییرات کوچک در برنامه، به طوری که این تغییرات خودشان را با تغییرات متناظر در یک یا چند پیمانه نشان دهنده؛
- **محافظت:** خصوصیتی از معماری که انتشار اثرات جانبی حاصل از خطا را در یک پیمانه کاهش می‌دهد.

Meyer [MEY90] پیشنهاد می کند که پنج اصل طراحی زیر را می توان برای معماری پیمانه ای بودن به دست آورد:

۱- واحدهای پیمانه ای بودن زیانی؛
۲- واسطهای معدود؛
۳- واسطهای کوچک (اتصال ضعیف)؛
۴- واسطهای مشخص؛
۵- مخفی سازی اطلاعات(انسجام بالا)

دورنمای OOD

در دهه ۱۹۸۰ و اوایل دهه ۱۹۹۰، گستره وسیعی از روش‌های تحلیل و طراحی OO، پیشنهاد و به کار گرفته شدند. این روش‌ها مولد نشانه گذاری، اصول طراحی و مدل‌های مربوط به OOD نوین شدند. نگاهی اجمالی به روش‌های اولیه OOD خالی از لطف نبوده و می‌تواند آموزنده نیز باشد:

روش Booch: همان طور که در فصل ۱۸ گفته شد، این روش شامل یک **فرآیند توسعه میکرو** ویک **فرآیند توسعه ماکرو** است. در زمینهٔ طراحی، توسعهٔ ماکرو شامل یک فعالیت طراحی معماری است که اشیای مشابه را در افرازهای معماری جداگانه، گروه‌بندی می‌کند؛ اشیاء را از نظر سطح انتزاع، لایه‌بندی می‌کند، سناریوهای مرتبط را شناسایی؛ یک

نمونه اولیه برای طراحی ایجاد و نمونه اولیه را با اعمال آن روی سناریوهای کاربرد، اعتبارسنجی می‌کند. توسعه میکرو، مجموعه‌ای از **قواعد** را تعیین می‌کند که حاکم بر استفاده از عملیات و صفات هستند و سیاست‌های خاص دامنه را برای مدیریت حافظه، کنترل خطا و عملکردهای زیرساختی دیگر وضع می‌کنند؛ سناریوهایی را توسعه می‌دهد که معانی این قواعد و سیاست‌ها را تبیین می‌کند؛ برای هر سیاست یک نمونه اولیه می‌سازد؛ نمونه اولیه را دستکاری و پالایش می‌کند و هر سیاست را مورد بازبینی قرار می‌دهد. به طوری که **تصویر معماری سیستم آشکار گردد**[BOO94].

روش Rambough. تکنیک مدلسازی اشیا (OMT) شامل یک فعالیت طراحی است که اجرای طراحی در سطح انتزاع متفاوت را تشویق می‌کند. طراحی سیستم بر آرایش قطعاتی تأکید دارد که برای ساخت محصول یا سیستم کامل مورد نیاز است. مدل تحلیل، به زیرسیستم‌هایی افراز می‌شود که به پردازنده‌ها و وظایف اختصاص داده می‌شوند. راهبردی برای پیاده‌سازی مدیریت داده‌ها تعریف شده منابع سرتاسری و راهکارهای کنترلی مورد نیاز برای دستیابی به آنها شناسایی می‌شوند.

طراحی اشیاء بر آرایش مشروح یک شیء منفرد تأکید دارد. عملیات از مدل تحلیل انتخاب می‌شوند و برای هر عمل الگوریتم‌هایی تعیین می‌شوند. ساختمان داده‌های مناسب برای صفات و الگوریتم‌ها ارایه می‌شوند. کلاس‌ها و صفات کلاس‌ها به شیوه‌ای طراحی می‌شوند که دستیابی به داده‌ها را بهینه کرده بازدهی کار کامپیوتری را بهبود می‌بخشند. برای پیاده‌سازی روابط میان اشیاء یک مدل پیغام‌رسانی ایجاد می‌شود[RUM91].

روش Jacobson. فعالیت طراحی برای OOSE (مهندسی نرم‌افزار شیءگرا)، نسخه ساده‌ای از یک روش شیءگرایی مقدماتی است که آن را نیز Jacobson ابداع کرده است. مدل طراحی بر قابلیت پیگیری مدل تحلیل OOSE تأکید دارد. نخست، مدل تحلیل ایده‌آلی انتخاب می‌شود که در محیط جهان واقعی بگنجد. سپس اشیای طراحی اصلی، موسوم به بلوک ایجاد می‌شوند و به عنوان بلوک‌های واسط، بلوک‌های موجودیت و بلوک‌های کنترل گروه‌بندی می‌شوند. ارتباط میان بلوک‌ها در حین اجرا تعیین می‌شود و بلوک‌ها به صورت زیر سیستم سازماندهی می‌شود [JAC92].

روش Coad و Yourdon. این روش برای OOD با مطالعه چگونگی انجام کار طراحی توسط **طراحان کارآمد شیءگرا** توسعه یافته است. این روش طراحی نه تنها به کاربرد می‌پردازد، بلکه به زیرساخت کاربر نیز توجه دارد و بر نمایش چهار مولفه اصلی سیستم، یعنی **دامنه مساله، تعامل با انسان، مدیریت وظایف و مدیریت داده‌ها** تأکید دارد[COA91].

روش Wirfs-Brock. این روش طیف پیوسته‌ای از وظایف را تعریف می‌کند که در آن، تحلیل بلافاصله منجر به طراحی می‌شود. پروتکل‌های مربوط به هر کلاس با پالایش پروتکل‌های میان اشیاء ایجاد می‌شوند. هر عمل (مسوولیت) و پروتکل (طراحی واسط) در سطحی از جزییات طراحی می‌شودکه قابل پیاده‌سازی باشند. مشخصه‌های مربوط به هر کلاس (تعریف مسئولیت‌های خصوصی و جزییات مربوط به عملیات) و هر زیرسیستم شناسایی همهٔ کلاس‌های بسته‌بندی شده و تعامل میان زیرسیستم‌ها) تهیه می‌شود[WIR90].

گرچه اصطلاحات و مراحل فرآیند برای هر یک از این روش‌های OOD متفاوت است، ولی کل فرآیند OOD یکی است. مهندس نرم‌افزار برای اجرای مهندسی شیءگرا باید مراحل زیر را اجرا کند:

1. **توصیف کلیه زیرسیستم‌ها و اختصاص دادن آن به پردازنده‌ها و وظایف.**
2. **انتخاب یک راهبرد طراحی برای پیاده‌سازی مدیریت داده‌ها، پشتیبانی واسطها و مدیریت وظایف.**
3. **طراحی یک راهکار کنترلی مناسب برای سیستم.**

۴. **اجرای طراحی اشیاء با ایجاد یک نمایش رویه‌ای برای هر عمل و ساختمان داده‌ها برای صفات کلاس‌ها.**
۵. **اجرای طراحی پیغام‌ها با استفاده از مشارکت میان اشیاء و روابط میان اشیاء.**
۶. **ایجاد مدل طراحی.**
۷. **بازبینی مدل طراحی و تکرار آن تا حد کفایت.**

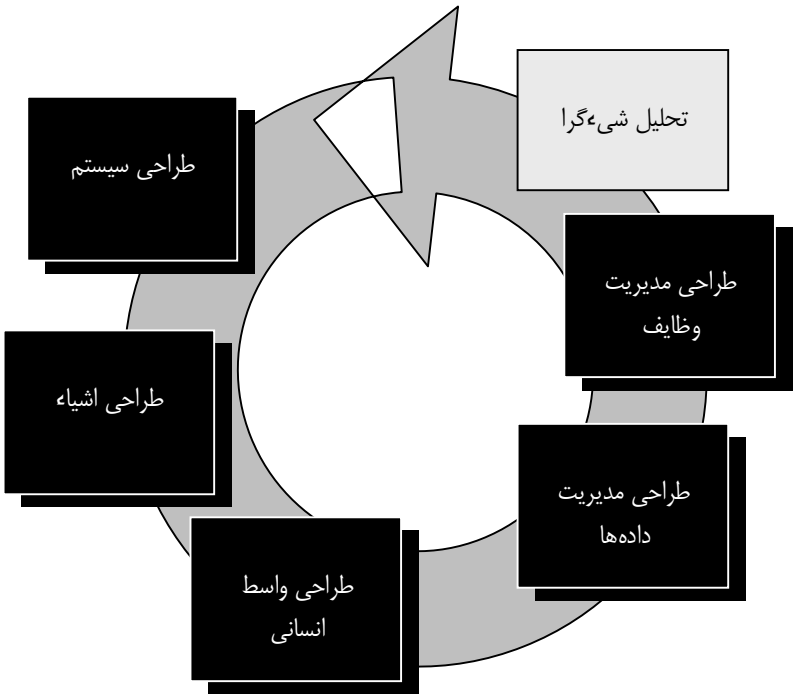
روش یکنواخت برای OOD

Grady، Booch، Rumbough و Jacobson بهترین ویژگی‌های روش‌های تحلیل و طراحی شیءگرای خود را کنار هم قرار دادند و روشی یکنواخت حاصل شد. نتیجه کار که زبان مدلسازی یکنواخت، UML، خوانده می‌شود، در سرتاسر صنعت، کاربردی گسترده یافته است.

در فرایند مدلسازی تحلیل (فصل ۱۸)، دیدگاه‌های مدل کاربر و مدل تحلیل ارایه می‌شود. این دیدگاه‌ها شناختی از سناریوهای استفاده (که راهنمایی برای مدلسازی رفتاری هستند) ارایه داده و با شناسایی و توصیف عناصر ساختاری ایستای سیستم، دیدگاه‌هایی از مدل پیاده‌سازی و مدل رفتاری ارایه می‌کند.

UML در دو فعالیت طراحی عمده سازماندهی می‌شود: **طراحی سیستم و طراحی اشیاء**.

هدف اصلی طراحی سیستم UML، نمایش دادن معماری نرم‌افزار است. جریان فرآیند از تحلیل به طراحی در شکل زیر نیز نشان داده شده است. در سرتاسر فرآیند طراحی UML، دیدگاه مدل کاربر و دیدگاه مدل ساختار، به نمایش طراحی بسط داده می‌شود.



شکل ۲: فریان فرایند طراحی شیءگرا

فرایند طراحی سیستم

فرآیند طراحی سیستم شامل فعالیت‌های زیر می‌شود:

- **افزار مدل تحلیل به زیرسیستم‌ها**
- **شناسایی همزمانی که توسط مساله دیکته می‌شود**
- **تخصیص زیرسیستم‌ها به پردازنده‌ها و وظایف**
- **توسعه یک طراحی برای واسط کاربر**
- **انتخاب یک راهبرد پایه برای پیاده‌سازی مدیریت داده‌ها**
- **شناسایی منابع سرتاسری و راهکاریهای کنترلی مورد نیاز برای دستیابی به آنها**
- **طراحی یک راهکار کنترلی مناسب برای سیستم از جمله مدیریت وظایف**
- **در نظر گرفتن چگونگی پرداختن به شرایط مرزی**
- **بازبینی و در نظر گرفتن مطالعات**

افراز مدل تحلیل

یکی از اصول بنیادی تحلیل (فصل ۱۱)، افراز سیستم است. در طراحی سیستم‌های OO، مدل تحلیل را افراز می‌کنیم تا مجموعه‌های منسجمی از کلاس‌ها، روابط و رفتارها را تعیین کنیم. این عناصر طراحی به صورت یک زیرسیستم بسته‌بندی می‌شوند.

زیرسیستم‌هایی که تعریف (و طراحی) می‌شوند، باید با ملاک‌های زیر نیز مطابقت کنند:

- زیرسیستم باید دارای یک واسط کاملاً تعریف شده باشد که ارتباط با بقیهٔ سیستم، از طریق آن انجام شود.
- به استثنای تعداد کوچکی از ((کلاس‌های ارتباطات))، کلاس‌های درون یک زیرسیستم باید فقط با کلاس‌های موجود در همان زیرسیستم مشارکت کنند.
- تعداد زیرسیستم‌ها نباید زیاد شود.
- زیرسیستم را می‌توان برای کاهش دادن پیچیدگی افراز کرد.

هنگامی که سیستمی به صورت زیرسیستم افراز می‌شود، یک فعالیت طراحی دیگر، موسوم به لایه‌بندی نیز رخ می‌دهد. هر لایه از سیستم OO شامل یک یا چند زیرسیستم است و سطح متفاوتی از انتزاع را برای قابلیت عملیاتی نشان می‌دهد که برای دستیابی به عملکردهای سیستم مورد نیاز است. در اکثر موارد، سطح انتزاع برحسب میزان وابستگی پردازش به زیرسیستمی سنجیده می‌شود که در معرض دید کاربر نهایی قرار دارد.

برای مثال، یک معماری چهار لایه‌ای ممکن است شامل این موارد شود:

۱- **لایه ارایه (زیرسیستم مرتبط با واسط کاربر)؛**
۲- **لایه کاربرد (زیرسیستمی که پردازش مرتبط با کاربرد را انجام می‌دهند)؛**
۳- **لایه قالب‌بندی داده‌ها (زیرسیستمی که داده‌ها را برای پردازش آماده می‌کند)؛**
۴- **لایه بانک اطلاعاتی (زیرسیستم مرتبط با مدیریت داده‌ها).**

هر لایه به طور عمیقتر وارد سیستم می‌شود و پردازشی را نشان می‌دهد که بیشتر خاص محیط است.

همزمانی و تخصیص زیرسیستم‌ها

جنبه پویای مدل رفتار اشیاء، شاخصی از همزمانی میان کلاس‌ها (یا زیرسیستم‌ها) فراهم می‌آورد. اگر کلاس‌ها (یا زیرسیستم‌ها) در یک زمان فعال نباشند، نیازی به پردازش همزمانی نیست. این بدان معنا است که کلاس‌ها (یا زیرسیستم‌ها) را می‌توان روی یک سخت‌افزار مشترک پیاده‌سازی کرد. از طرفی دیگر، اگر کلاس‌ها (یا زیرسیستم‌ها) باید در یک زمان، روی رویدادها عمل کنند، آنها را همزمان در نظر می‌گیرند. هنگامی که زیرسیستم‌ها همزمان هستند، دو گزینه برای تخصیص داریم: ۱. تخصیص هر زیرسیستم به یک پردازنده مستقل یا ۲. تخصیص زیرسیستم‌ها به یک پردازنده مشترک و فراهم آوردن پشتیبانی همزمانی از طریق ویژگی‌های سیستم عامل.

مؤلفه مدیریت وظایف

Coad و Yourdon برای طراحی اشیایی که وظایف جاری را مدیریت می‌کنند، راهبرد زیر را پیشنهاد کرده‌اند:

- **خصوصیات وظیفه تعیین می‌شود.**
- **وظیفه هماهنگ‌سازی و اشیای مرتبط با آن تعریف می‌شود.**
- **هماهنگ‌سازی و دیگر وظایف با یکدیگر مجتمع می‌شوند.**

هنگامی که خصوصیات وظیفه تعیین شد، صفات و عملیاتی از شیء که برای هماهنگی و برقراری ارتباط با وظایف دیگر مورد نیاز هستند، تعیین خواهند شد. الگوی اصلی وظایف(برای یک شیء) به شکل زیر است:

نام وظیفه – نام شیء
توصیف – شرحی که هدف شیء را بیان می‌کند.
اولویت – شامل مثلاً کم، متوسط و زیاد
سرویس – لیستی از عملیات و مسوولیت‌های شیء
هماهنگی – شیوه رفتار شیء
برقراری ارتباط از طریق – مقادیر داده‌های ورودی و خروجی مربوط به وظیفه

مؤلفه واسط کاربر

گرچه مؤلفه واسط کاربر درحیطه دامنه مساله پیاده‌سازی می‌شود، خود واسط، یک زیرسیستم بسیار مهم برای اکثر برنامه‌های کاربردی مدرن به شمار می‌رود. مدل تحلیل OO (فصل ۱۸) شامل سناریوهای استفاده (موارد کاربرد) و شرحی از نقش‌هایی است که کاربر هنگام تعامل با سیستم باید ایفا کند. این‌ها به عنوان ورودی فرآیند طراحی واسط کاربر عمل می‌کنند.

هنگامی که بازیگر و سناریوی آن تعریف شدند، سلسله مراتبی از فرمان‌ها تعیین می‌شود. این سلسله مراتب فرمان‌ها، گروه‌های اصلی منوی سیستم (منوی میله‌ای، یا جعبه ابزار)، و کلیه عملکردهای فرعی را تعریف می‌کنند که از طریق یک گروه منوی سیستم (پنجره‌های منو) در دسترس هستند. سلسله مراتب منوها به طور تکراری مورد پالایش قرار می‌گیرد تا اینکه کلیه موارد کاربرد را بتوان با حرکت در سلسله مراتب عملکردها پیاده‌سازی کرد.

از آنجا که گسترهٔ وسیعی از محیط‌های توسعهٔ واسط‌های کاربری از قبل وجود دارند، نیازی به طرحی عناصر GUI نیست. کلاس‌های قابل استفاده مجدد (با صفات و عملیات مناسب) از قبل برای پنجره‌ها، نمادها(Icons)، عملیات ماوس و گستره وسیعی از امور تعاملی دیگر وجود دارد. برای پیاده‌سازی، فقط کافی است از اشیایی که دارای خصوصیات مناسب هستند، نمونه‌برداری شود.

مؤلفه مدیریت داده‌ها

مدیریت داده‌ها شامل دو زمینه کاری متمایز است:

۱- مدیریت داده‌هایی که در خود برنامه کاربردی اهمیت بحرانی دارند و

۲- خلق زیر ساختی برای ذخیره‌سازی و بازیابی اشیاء.

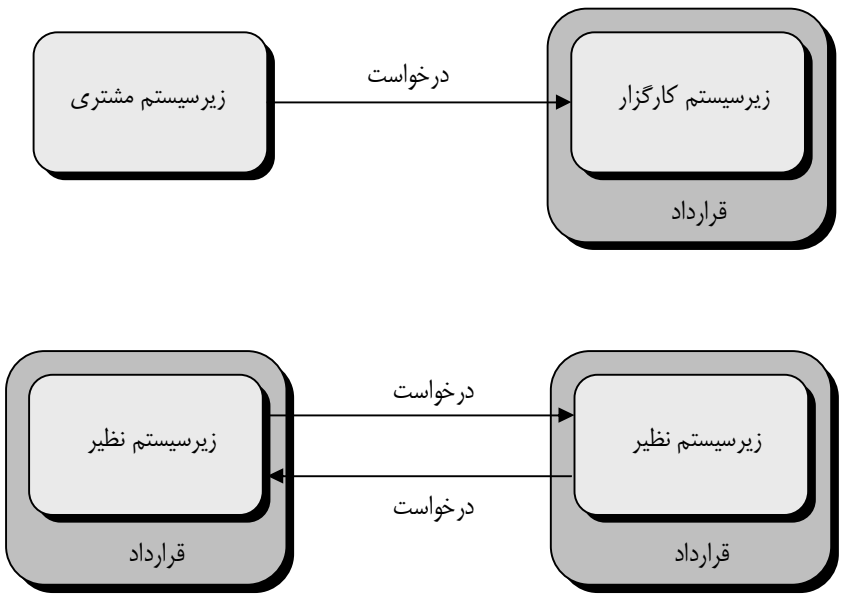
به طور کلی، مدیریت داده‌ها به شیوه‌ای لایه‌ای طراحی می‌شود. ایده اصلی، جداسازی خواسته‌های سطح پایین برای دستکاری ساختمان داده‌ها، از خواسته‌های سطح بالا برای در دست گرفتن صفات سیستم است.

مؤلفه مدیریت منابع

منابع متفاوتی در دسترس یک سیستم یا محصول OO قرار دارند و در بسیاری از موارد، زیرسیستم‌ها بر سر این منابع به رقابت می‌پردازند. منابع سیستمی سرتاسری می‌توانند نهادهای خارجی (مثل گرداننده دیسک، پردازنده، یا خط ارتباطی) یا انتزاعی (مثلاً یک بانک اطلاعاتی یا شیء) باشند. ماهیت منبع هرچه که باشد، مهندس نرم‌افزار باید یک راهکار کنترلی برای آن طراحی کند. Rumbaugh و همکاران وی [RUM91] پیشنهاد می‌کنند که هر منبعی باید به یک شیء نگهبان تعلق داشته باشد. شیء نگهبان، در واقع مراقب و حافظ منبع بوده و دستیابی به آن را کنترل و از تضاد تقاضاها جلوگیری می‌کند.

برقراری ارتباط میان زیرسیستم‌ها

هنگامی که همهٔ زیرسیستم‌ها مشخص شدند، لازم است مشارکت‌های موجود بین آنها نیز تعیین گردد. مدلی که ما برای مشارکت شیء با شیء به کار می‌بریم، به کل زیرسیستم‌ها نیز قابل بسط است. شکل زیر یک مدل مشارکت را نشان می‌دهد.



شکل ۳: مدلی از مشارکت میان زیر سیستم‌ها

فرآیند طراحی اشیاء

سیستم طراحی OO را با توجه به استعاره‌ای که در این متن به کار گرفته شد، می‌توان به عنوان نقشهٔ پلان یک خانه در نظر گرفت. نقشهٔ پلان، هدف از ساخت هر اتاق و ویژگی‌های معماری، اتصال اتاق‌ها به یکدیگر و به محیط خارج را مشخص می‌سازد. اکنون زمان آن فرا رسیده که جزییات مورد نیاز برای ساخت هر اتاق تهیه شود. در حیطهٔ OOD، **طراحی اشیاء** بر **اتاق ها** تأکید دارد.

در این مرحله است که اصول و مفاهیم پایه‌ای مرتبط با طراحی در ساختار مؤلفه‌ها مطرح می‌شوند. ساختمان داده‌های محلی (برای صفات) تعیین و الگوریتم‌ها (برای عملیات) طراحی می‌شوند.

توصیف اشیاء

توصیف طراحی یک شیء (نمونه‌ای از یک کلاس یا زیرکلاس) می‌تواند یکی از دو شکل زیر انجام گیرد[GOL83]:

۱. **توصیف قرارداد** که واسط شیء را با تعریف هر پیغامی که شیء می‌تواند دریافت کند و عملی که شیء به هنگام دریافت آن پیغام اجرا می‌کند، برقرار می‌سازد، یا

۲. **توصیف پیاده‌سازی** مربوط به عمل درخواست شده توسط پیغام را نشان می‌دهد. جزییات پیاده‌سازی شامل اطلاعاتی درباره بخش خصوصی شیء می‌شود، یعنی جزییاتی درباره ساختمان داده‌هایی که صفات شیء و جزییات رویه‌ای توصیفگر عملیات را توصیف می‌کنند.

طراحی الگوریتم‌ها و ساختمان داده‌ها

تنوع نمایش‌های موجود در مدل تحلیل و طراحی سیستم، مشخصه‌ای برای کلیه عملیات و صفات فراهم می‌آورند. الگوریتم‌ها و ساختمان‌های داده‌ای با استفاده از روشی طراحی می‌شوند که قدری با روش‌های طراحی داده‌ها و طراحی در سطح مؤلفه‌های بحث شد، تفاوت دارد.

برای پیاده‌سازی مشخصات مربوط به هر عمل، الگوریتمی ایجاد می‌شود. در بسیاری از موارد، الگوریتم یک دنباله محاسباتی یا رویه‌ای است که می‌توان آن را به صورت یک پیمانه نرم‌افزاری مستقل پیاده کرد. ولی، اگر مشخصات عملی پیچیده باشد، ممکن است نیاز به پیمانه کردن عمل باشد. برای این منظور می‌توان از تکنیک‌های سنتی طراحی در سطح مؤلفه‌ها استفاده کرد.

ساختمان داده‌ها همزمان با الگوریتم‌ها طراحی می‌شوند. از آنجا که عملیات، صفات یک کلاس را دستکاری می‌کنند، طراحی ساختمان داده‌ها که صفات را به خوبی منعکس کند، طراحی الگوریتم‌های عملیات مربوط را بسیار دشوار می‌سازد.

گرچه انواع مختلفی از عملیات وجود دارند، با این وجود می‌توان آنها را به ۳ گروه تقسیم کرد:

۱. عملیاتی که به طریقی داده‌ها را دستکاری می‌کنند (مثل اضافه کردن، حذف کردن، قالب‌بندی، گزینش)؛

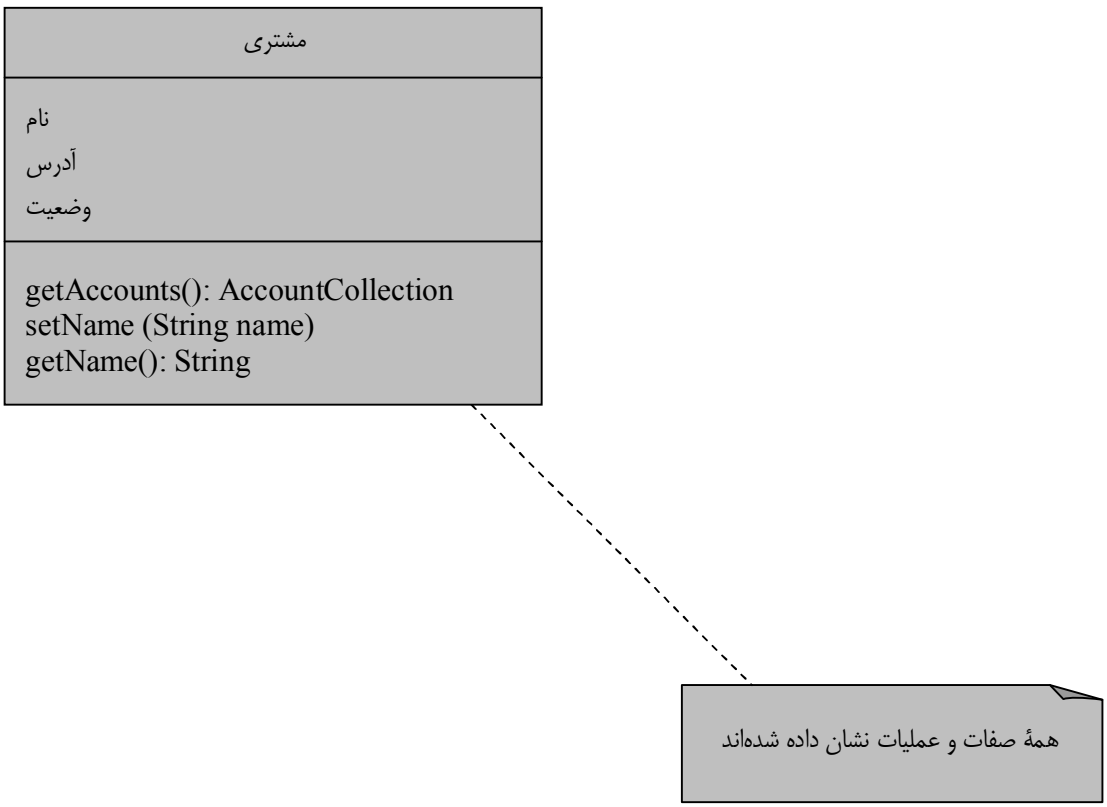
۲. عملیاتی که یک کار محاسباتی انجام می‌دهند؛

۳. عملیاتی که شیء را از لحاظ رخ دادن یک رویداد کنترلی مورد نظارت قرار می‌دهند.

مدل کلاس‌ها

مدل کلاس، توصیفی از کلاس‌ها و روابط میان آنها در یک سیستم است. این مدل، رفتار پویای سیستم، مثلاً رفتار تک تک اشیاء را توصیف نمی‌کند. نخستین عنصر نمودار کلاس‌ها، شرحی از تک تک کلاس‌ها است. شکل ۴ چگونگی توصیف یک کلاس را نشان می‌دهد. این کلاس به مشتریان یک بانک مربوط می‌شود.

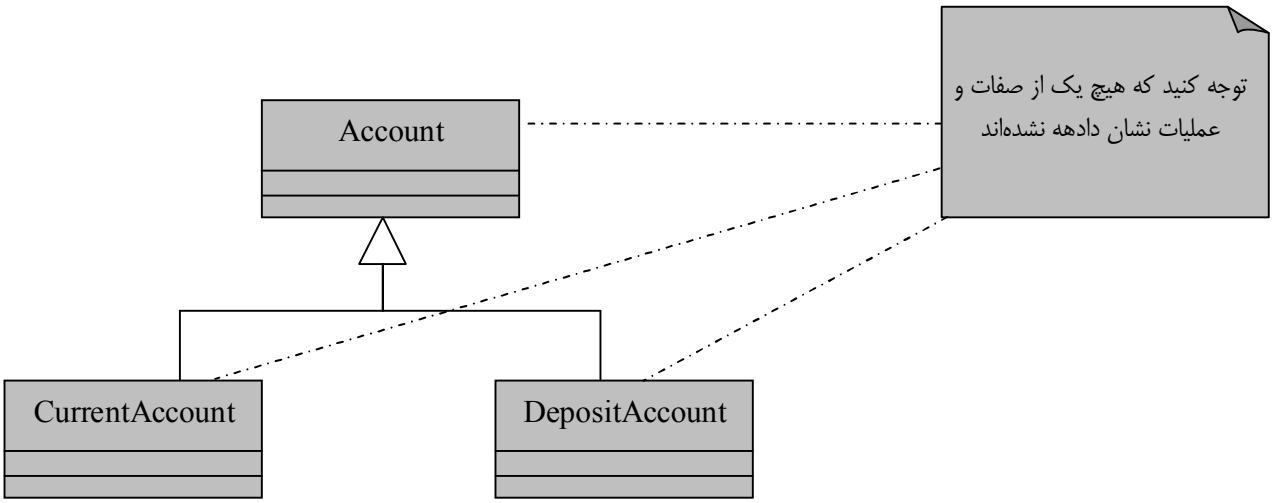
این شکل بسیار ساده است، زیرا تنها یک کلاس دارد. این مدل شامل نام کلاس (Customer)، نام برخی صفات آن (مثلاً صفت address یعنی نشانی مشتری است) و لیستی از عملیات (مثلاً getName، نام مشتری را برمی‌گرداند) است. پس هر مستطیلی که نشانگر یک کلاس است شامل قسمتی برای نام کلاس، قسمتی برای صفات اشیای تعریف شده توسط آن کلاس و قسمتی برای لیست عملیات مرتبط با این شیء است.



شکل ۴: مثالی از بخش از یک کلاس توصیف شده در UML

تعمیم

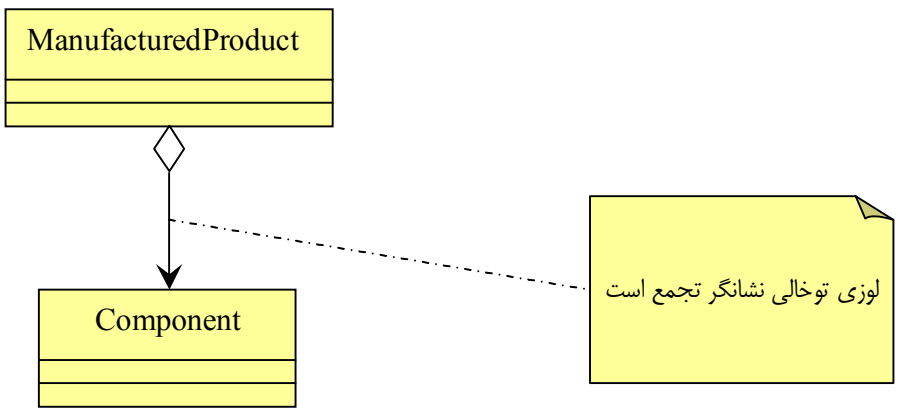
این رابطه میان کلاس X و کلاس Y زمانی برقرار است که کلاس Y نمونه خاص از کلاس X باشد. برای مثال، بین کلاس Account که نشانگر یک حساب بانکی عمومی است و یک حساب جاری Current Account که نمونه خاصی از یک حساب است، رابطه **تعمیم** وجود دارد. شکل ۵ چگونگی نمایش این رابطه را در یک نمودار کلاس‌های UML نشان می‌دهد.



شکل ۵: مثالی از یک تعمیم در UML.

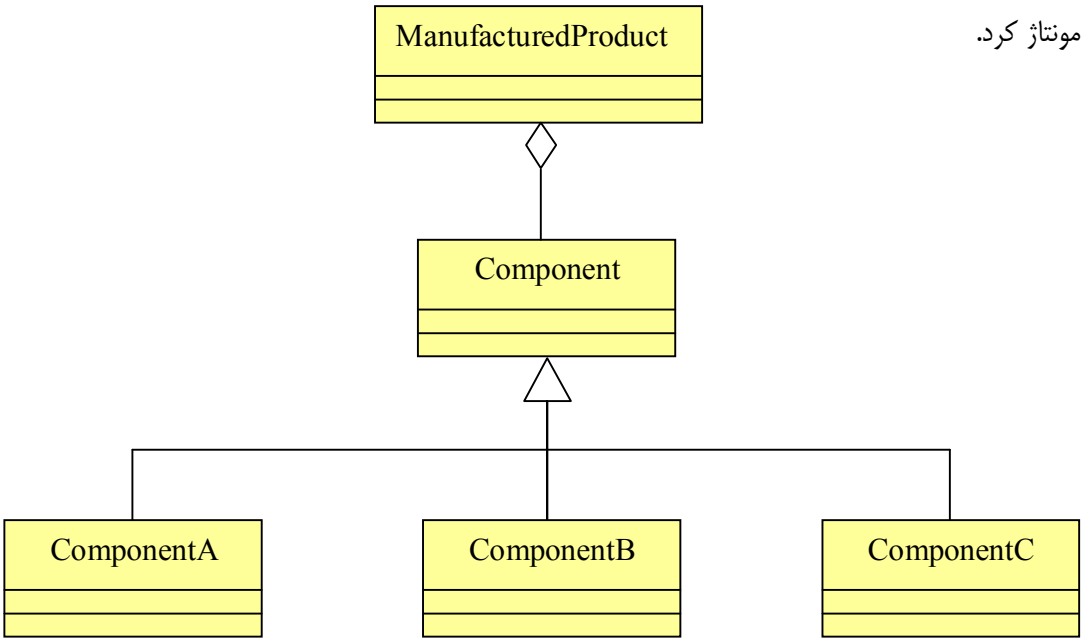
تجمع و ترکیب

روابط مهم دیگر عبارتند از **تجمع** و **ترکیب**. دو رابطه وجود دارند که نشان می‌دهند یک کلاس اشیایی تولید می‌کند که بخشی از یک شیء است که توسط کلاس دیگری تعریف شده است. برای مثال، سیستمی برای یک تولیدکننده باید داده‌های مربوط به اقلامی را نگهداری کند که تولید می‌شوند و اینکه از چه مواردی ساخته می‌شوند. مثلاً یک کامپیوتر از قطعاتی مثل دستگاه اصلی، حافظه جانبی، کارت‌های حافظه و غیره ساخته می‌شود. کامپیوتر از قطعات تشکیل می‌شود و در یک سیستم شیءگرا که برای پشتیبانی تولید به کار برده می‌شود، یک رابطهٔ تجمعی میان کلاس توصیف کننده محصول تولید شده و هر یک از قطعات آن وجود دارد. بنابراین می‌گوییم یک رابطه تجمع وجود دارد. شکل ۶ چگونگی نمایش این رابطه **تجمع** را در نمودار کلاس UML نشان می‌دهد.



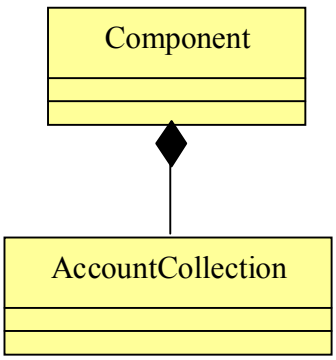
شکل ۶: رابطه تجمع در UML

در اینجا،خطی که لوزی توخالی به آن متصل است، نشان می‌دهد که کلاس، اشیایی را توصیف می‌کند که اشیای دیگر را با هم مجتمع می‌کنند. کلاسی که لوزی به آن متصل است، اشیایی را توصیف می‌کند که شامل اشیایی است که توسط کلاس‌هایی دیگر تعریف شده‌اند. در UML، روابط معمولاً به هم آمیخته‌اند. برای مثال در شکل ۱۲-۲۲، چند مؤلفه وجود دارند که با کلاس Component رابطه تعمیم دارند (شکل ۱۳-۲۲). دراین شکل Cmponent با چند کلاس اختصاصی‌تر مرتبط است که توصیفگر قطعاتی هستند که محصول رامی‌توان از آنها مونتاژ کرد.



شکل ۷: نمودار کلاس‌های UML که تعمیم و تجمع را نشان می‌دهد.

شکل خاصی از **تجمع** هست که به آن **ترکیب** گفته می‌شود. این رابطه زمانی استفاده می شود که یک شیء شامل چند شیء دیگر باشد و وقتی که شیء اصلی حذف شود، همهٔ اشیای موجود در آن نیز از بین می‌روند. برای مثال کلاس Customer که نشانگر مشتریان بانک است، با حساب‌های مشتری رابطه‌ای ترکیبی دارد، زیرا اگر مشتری حذف شود، همهٔ حساب‌های وی نیز حذف خواهد شد. این رابطه مشابه با رابطه تجمیع نشان داده می‌شود، ولی با یک لوزی توپر (شکل ۸)



شکل ۸: نمودار کلاس UML که ترکیب را نشان می‌دهد

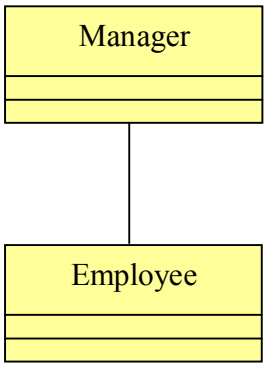
همبستگی‌ها

تجمع و ترکیب، مثال‌های خاصی از رابطه میان دو کلاس هستند. وقتی یک رابطه میان دو کلاس برقرار می‌شود که بین آن دو اتصالی برقرار باشد؛ این اتصال در UML به عنوان همبستگی شناخته می‌شود. برخی از مثال‌های همبستگی عبارتند از:

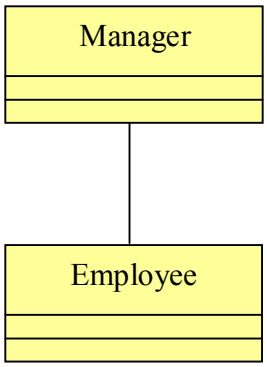
- کلاس Manager (مدیر) با کلاس Employee (کارمند) رابطه دارد، زیرا یک مدیر چند کارمند را مدیریت می‌کند.
- کلاس Flight (پرواز) با کلاس Plane (هواپیما) بستگی دارد، زیرا هواپیما یک پرواز خاص را به انجام می‌رساند.
- کلاس Computer با کلاس Message (پیغام) رابطه دارد، زیرا مجموعه‌ای از پیغام‌ها منتظرند تا توسط کامپیوتر پردازش شوند.
- کلاس BankStatement (صورت حساب) با کلاس Transaction (تراکنش) رابطه دارد، زیرا صورت حساب شامل جزییات هر تراکنش است.

از میان این روابط، فقط آخری رابطه تجمعی است. همهٔ روابط دیگر، همبستگی‌های ساده‌اند. این همبستگی‌ها در UML به صورت یک خط راست نشان داده می‌شوند. برای مثال، در شکل ۹ نخستین همبستگی نشان داده شده است.

همبستگی میان کلاس‌ها برحسب چندگانگی همبستگی و نام همبستگی نیز مستندسازی می‌شود. با در نظر گرفتن مثال‌های نشان داده شده در شکل ۹، نگاهی به چندگانگی خواهیم داشت.

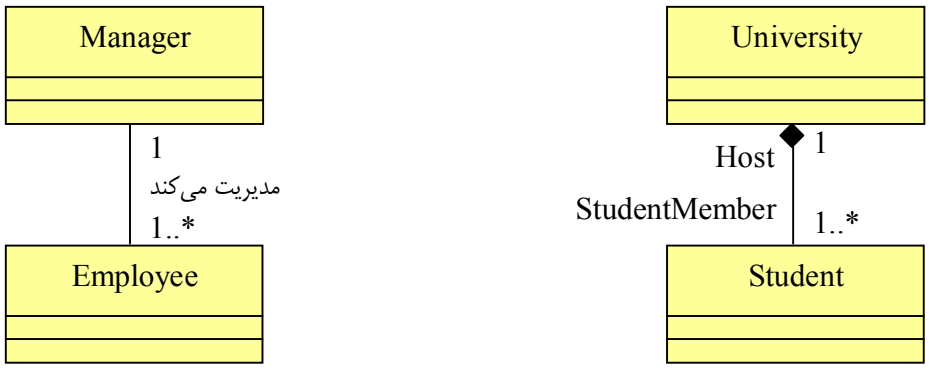


نمونه‌ای از رابط هواپیما در UML.



تعدد در نمودار کلاس UML.

شکل ۹: رابطه‌های همبستگی

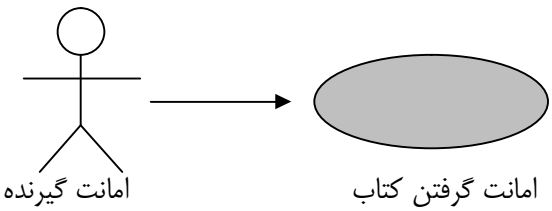


مستندسازی همبستگی. **مستندسازی نقش**

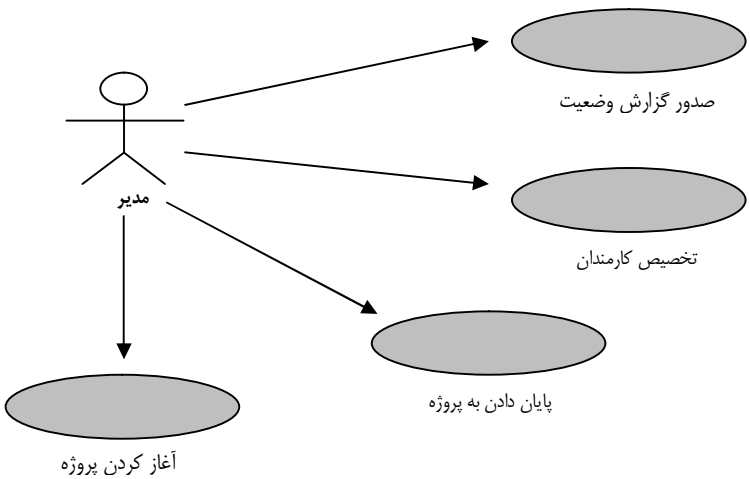
شکل ۱۰: رابطه‌های همبستگی با تعیین چندی

موارد کاربرد

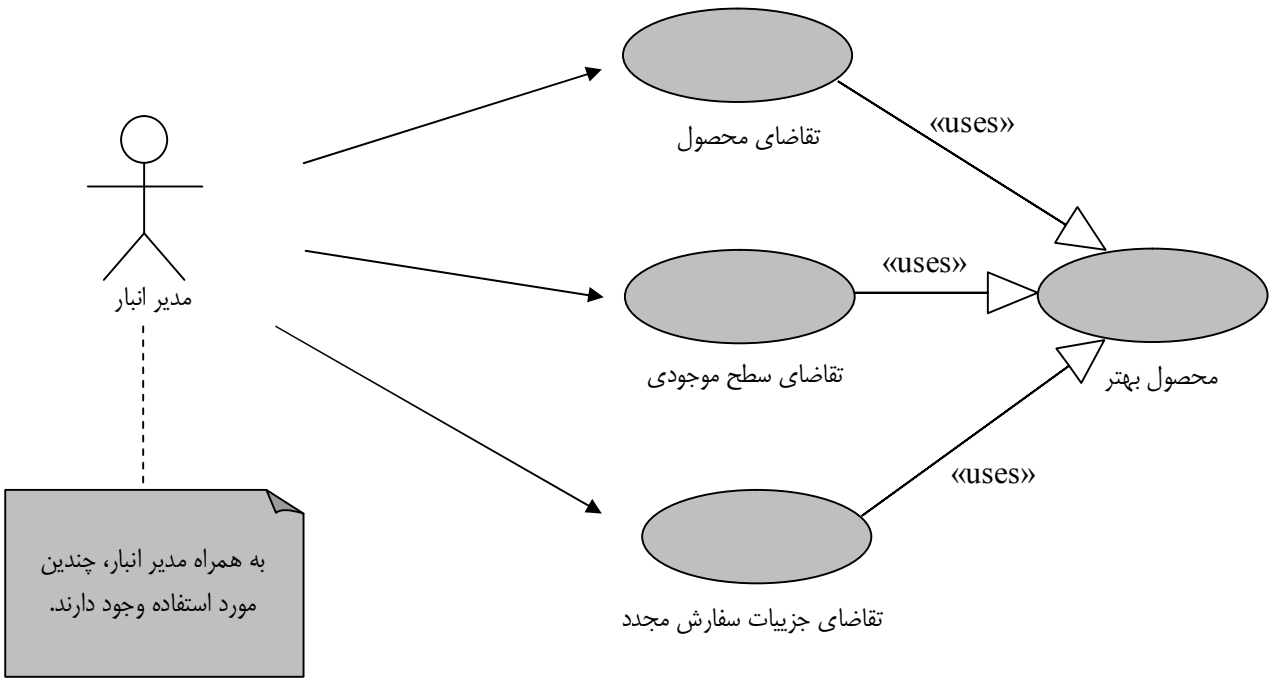
در فصل ۱۸ موارد کاربرد را مورد بحث قرار دادیم. در UML، مورد کاربرد بسیار ساده، برحسب بازیگر (Actor) و یک مورد کاربرد مستندسازی می‌شود. بازیگر، عاملی است که با سیستمی که در حال ساخته‌شدن است، تعامل می‌کند؛ مثلاً خلبان یک هواپیما، کسی که از کتابخانه کتاب به امانت می‌گیرد یا مدیر چند کارمند در یک شرکت. هر مورد کاربرد، عملی را که بازیگر انجام می‌دهد مستندسازی می‌کند؛ مثل تغییر دادن جهت حرکت هواپیما، امانت گرفتن کتاب از کتابخانه یا افزودن یک عضو جدید به تیم برنامه‌نویسی. نمونه‌هایی از مورد کاربردها در شکل‌های ۱۱ تا ۱۳ نشان داده شده است. این مثال، کاربری را نشان می دهد که در حال امانت گرفتن یک کتاب از کتابخانه است.



شکل ۱۱: یک مورد کاربرد ساده



شکل ۱۲: چند مورد کاربرد



شکل ۱۳: مثالی از موارد کاربرد دیگر

مشارکت‌ها

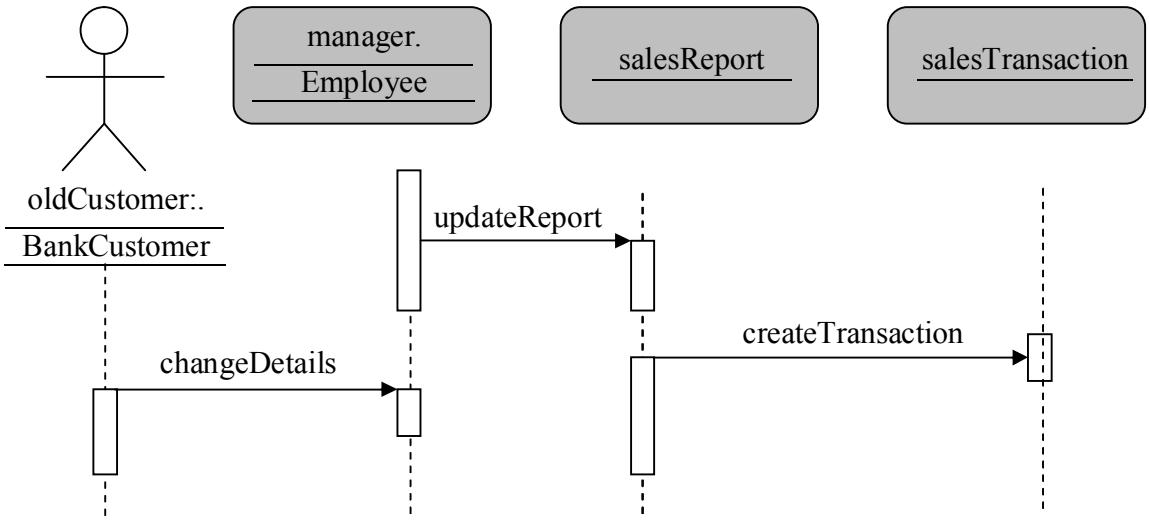
در فرایند اجرای یک سیستم شیءگرا، اشیای سیستم با یکدیگر تعامل می‌کنند. برای مثال، در یک سیستم بانکداری، شیء Account ممکن است پیغامی به یک شیء تعاملی ارسال نماید تا تراکنشی را ایجاد کند که در آن حساب رخ داده است. برای مثال، از حساب برداشت شده است. این نوع اطلاعات برای طراحی یک سیستم شیءگرا در حین فرآیند شناسایی و اعتبارسنجی کلاس‌ها مهم است. از این رو، UML دو نشانه‌گذاری متفاوت برای تعریف تعامل‌ها دارد. **نمودار توالی** (Sequence Diagram) و نمودار دیگری که به عنوان **نمودار مشارکت**(Collaboration Diagram) شناخته می‌شود و هم ارز نمودار توالی است؛ در واقع، آنها چنان شبیه یکدیگرند که ابزارهای Case غالباً می‌توانند یک نمودار را از روی دیگری ایجاد کنند. شکل‌های ۱۴ و ۱۵ نمونه‌هایی ساده از نمودار توالی را نشان می‌دهند.

در نمودار ۱۴، سه شیء وجود دارد که در یک تعامل شرکت دارند. اولی، شیء manager است که توسط کلاس Employee توصیف می‌شود. این شیء یک پیغام updateReport را به شیء salesReport ارسال می‌کند که سپس آن شیء پیغام‌های CreatTransaction را به شیء دیگری به نام salesTransaction ارسال می‌کند. در نمودار توالی، سه شیء موجودند که یکی از آنها (manager) دارای کلاس مشخص خود(Employee) است ولی بقیه خیر.

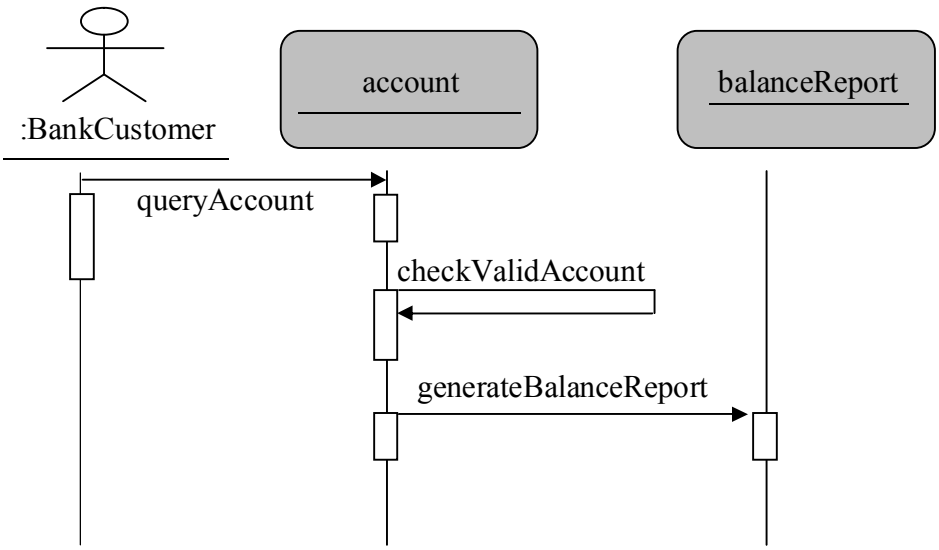
محتویات یکی از کادرهای موجود در نمودار توالی می‌تواند فقط شامل نام یک شیء به همراه نام کلاس آن که توسط دو نقطه (:) از هم جدا شده یا تنها نام کلاس و قبل از آن دو نقطه (:) باشد؛ در موردآخر، شیء بدون نام است.

شکل ۱۴ نقش یک بازیگر را نیز در مشارکت نشان می‌دهد؛ در اینجا، بازیگر BankCustomer با ارسال پیغام changeDetails با مدیر شیء Employee تعامل می‌کند.

شکل ۱۵ مثال دیگری از یک نمودار توالی را نشان می‌دهد. در اینجا، بازیگری که توسط شیء بدون نام تعریف شده است به وسیله کلاس BankCustomer نمایش داده می‌شود، به شیء account پیغامی ارسال می‌کند که حساب را تقاضا می‌کند. این شیء چک می‌کندکه آیا حساب معتبر است و سپس پیغام generateBalanceReport به شیء balanceReport ارسال می‌کند که شامل داده‌هایی است که مشتری بانک درخواست کرده است.



شکل ۱۴: نمودار توالی ساده



شکل ۱۵: مثال دیگری از نمودار توالی

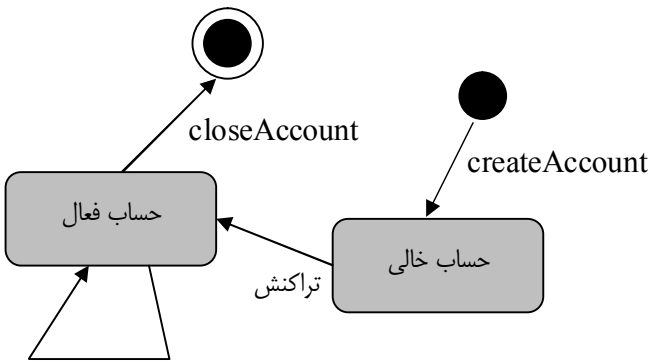
نمودارهای حالت

یکی دیگر از اجزای مهم UML، نمودار حالت است. این نمودار، حالت‌های گوناگونی را نشان می‌دهد که شیء می‌تواند آن حالت‌ها را دارا باشد و نشان می‌دهد که چگونه هر حالت به حالت دیگر گذار می‌کند. چنین نموداری شامل چند مؤلفه است:

- **حالت‌ها که به صورت کادرهایی با گوشه‌های گرد نشان داده می‌شوند.**
- **گذارهای میان حالت‌ها که به صورت خطوط پیکان‌دار نشان داده می‌شوند.**
- **رویدادها که باعث گذار میان حالت‌ها می‌شوند.**
- **علامت شروع که حالت اولیهٔ شیء را به هنگام ایجاد نشان می‌دهد.**
- **علامت توقف که نشان می‌دهد شیئی به پایان حیات خود رسیده است.**

نمونه‌ای از نمودار حالت در شکل ۱۶ نشان داده شده است.

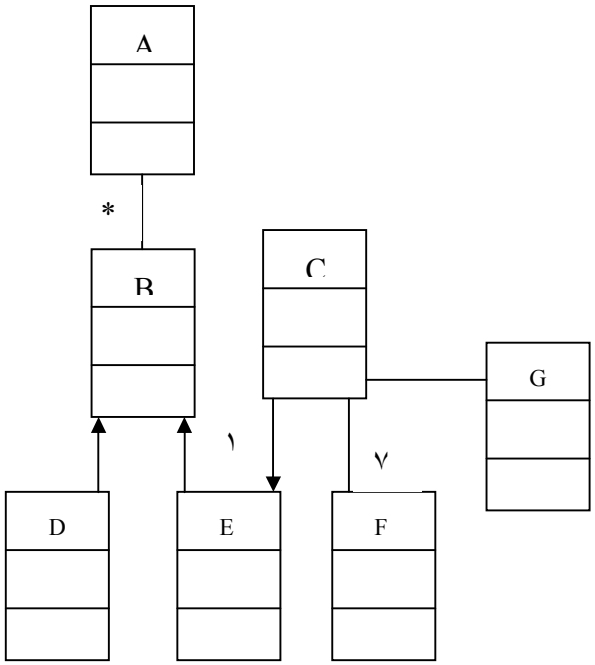
در اینجا چرخه حیات یک حساب بانکی نشان داده شده است. هنگامی که حساب ایجاد شد، به عنوان یک حساب خالی در نظر گرفته می‌شود. به محض آنکه یک تراکنش روی حساب رخ داد، حساب، فعال در نظر گرفته می‌شود. نمودار حالت نشان می‌دهد که وقتی حساب بسته شد، باید از بین برود.



شکل ۱۶: نمونه‌ای از نمودار حالت

تست‌های فصل ۱۹: طراحی شیءگرا

۱- با توجه به نمودار کلاس زیر گزینه صحیح را انتخاب کنید.



الف) E بخشی از C می‌باشد. E نوعی از B می‌باشد. یک شی از نوع G با چند شی از نوع C رابطه دارد.
ب) C بخشی از E می‌باشد. D نوعی از B می‌باشد. یک شی از نوع C با یک تا هفت نوع از شی F رابطه دارد.
ج) E بخشی از C می‌باشد. E نوعی از B می‌باشد. یک شی از نوع C با یک تا هفت نوع از شی F رابطه دارد.
د) C بخشی از E می‌باشد. B نوعی از D می‌باشد. یک شی از نوع C با یک تا هفت نوع از شی F رابطه دارد.

۲- چهار لایه تعریف شده برای طراحی شیءگرا مشابه لایه‌های تعریف شده در طراحی نرم‌افزار سنتی است.

الف) درست ب) غلط

۳- کدامیک از گزینه‌های زیر در مدل‌های طراحی شیءگرا وجود دارند ولی در مدل‌های طراحی سنتی نیستند؟

الف) نمایش سلسله مراتب پیمانه‌ها ب) مشخصات تعاریف داده‌ها
ج) مشخصات ارتباط پیغام‌ها د) مشخصات منطق رویه‌ای

۴- در طراحی شیء گرا به اتصال پایین پیمانه‌ها دست می‌یابیم که این امر شامل پنهان‌سازی اطلاعات بهتر نسبت به سایر روش‌ها است.

الف) درست ب) نادرست

۵- مراحل عمومی یکسانی برای طراحی شیءگرا بکار گرفته می‌شود بجز اینکه روش طراحی خاصی انتخاب می‌گردد.

الف) درست ب) نادرست

۶- نگرش UML به طراحی شیءگرا دارای دو فعالیت عمده است. این فعالیت‌ها کدامند؟

الف) طراحی معماری و طراحی اشیاء ب) طراحی واسط و طراحی پیغام
ج) طراحی واسط و طراحی سیستم د) طراحی سیستم و طراحی اشیاء

۷- کدامیک از فعالیت‌های زیر بخشی از فعالیت طراحی سیستم با نگرش UML به OOD است؟

الف) انتخاب استراتژی برای مدیریت داده‌ها ب) افراز مدل تحلیل به زیرسیستم‌ها

ج) طراحی واسط کاربر د) تمام موارد فوق

۸- اولین مرحله از طراحی سیستم در OOD، افراز مدل تحلیل به مجموعه‌ای از کلاس‌ها، روابط بین آنها، و رفتارها است. این کار را ..... می‌نامند؟

الف) سلسله مراتبی کلاس‌ها ب) ارتباطهای مشتری/ کارگزار

ج) زیرسیستم‌ها د) لایه‌های سیستمی

۹- وقتی زیرسیستم‌ها همروند (همزمان) هستند باید آنها را به پردازنده‌های مجزایی اختصاص داد.

الف) درست ب) نادرست

۱۰- واسط‌های کاربران معمولاً با استفاده از ابزارهای اتوماتیک ساخته می‌شود که شامل کلاس‌های قابل استفادهٔ مجدد است به طوری‌که پیاده‌ساز فقط باید اشیاء مناسب با محدودهٔ مسأله را در آن تعریف کند.

الف) درست ب) نادرست

۱۱- کدامیک از زمینه‌های زیر به عنوان بخشی از مدیریت داده مولفه‌های طراحی سیستم OOD است؟

الف) ایجاد زیرساختاری برای ذخیره و بازیابی اشیاء.

ب) مدیریت داده‌هایی که برای نرم‌افزار کاربردی حیاتی و حساس هستند.

ج) نرمال‌سازی صفات کلاس‌های داده.

د) موارد الف و ب

۱۲- هر قراردادی بین زیرسیستم‌ها دقیقاً از طریق یک پیغام که بین اشیاء زیر سیستم قرار دارد ایجاد می‌گردد.

الف) درست ب) نادرست

۱۳- طراح توصیف یک شیء یکی از دو صورت زیر می‌باشد:

الف) الگوی شیء یا شبه کد ب) توالی اپراتور یا گراف‌های صفات

ج) توصیف پروتکل و یا توصیف شیء د) گراف همکاری زیرسیستم یا گراف پروتکل

۱۴- در OOD عملیات با اعمال زیر پالایش می‌شوند.

الف) ایزوله کردن عملیات جدید در پایین‌ترین سطح انتزاع ب) تهیه تجزیه گرامری

ج) نوشتن خلاصه فرآیند د) تمام موارد

۱۵- طراحی الگوها برای طراحی نرم‌افزار شیءگرا قابل استفاده و بکارگیری نیست؟

الف) درست ب) نادرست

۱۶-طراحی‌های شیءگرا نیازی به استفاده از تکنیک‌های برنامه‌سازی شیءگرا در پیاده‌سازی ندارند.

الف) درست ب) نادرست

فصل ۲۰: آزمون نرم افزار و راهبردها

اهمیت آزمایش نرم‌افزار و اثرات آن بر کیفیت نرم‌افزار نیاز به تأکید بیشتر ندارد. Deutch در این باره این گونه بیان می‌نماید:

توسعهٔ سیستم‌های نرم‌افزاری شامل یک‌سری فعالیت‌های تولید می‌باشد که امکان اشتباهات انسانی در آن زیاد است. خطاها در ابتدای یک فرآیند و مراحل توسعه بعدی آن ظهور می‌نمایند. به دلیل عدم توانایی انجام کارها و برقراری ارتباط به صورت کامل، توسعهٔ نرم‌افزار همواره با فعالیت تضمین کیفیت همراه است.

آزمایش نرم‌افزار عنصری حیاتی از تضمین کیفیت نرم‌افزار می‌باشد و مرور تقریبی مشخصه، طراحی، و تولید کد را نشان می‌دهد.

یک شیوهٔ استراتژیک برای آزمایش نرم‌افزار

آزمایش، مجموعه فعالیت‌هایی است که می‌تواند از قبل به صورت سیستماتیک برنامه‌ریزی و هدایت شوند. به این دلیل، الگویی برای آزمایش نرم‌افزار باید برای فرآیند نرم‌افزار تعریف شود. این الگو شامل مجموعه مراحلی است که می‌توان تکنیک‌های خاص طراحی نمونه های آزمایش و روش های آزمایش را در آن قرار داد.

چند استراتژی آزمایش نرم‌افزار در این رابطه پیشنهاد شده است. همهٔ آنها برای توسعه دهندهٔ نرم‌افزار، الگویی را به منظور آزمایش فراهم می‌کنند و همگی دارای خصوصیات زیر هستند:

- آزمایش از سطح مؤلفه شروع می‌شود به سمت خارج در جهت مجتمع‌سازی کل سیستم کامپیوتری پیش می‌رود.
- تکنیک‌های متفاوت آزمایش، در نقاط زمانی مختلف مناسب می‌باشند.
- آزمایش توسط توسعه دهندهٔ نرم‌افزار و برای پروژه‌های بزرگ توسط گروه مستقل آزمایش، هدایت می‌شود.
- آزمایش و اشکالزدایی فعالیت‌های متفاوتی هستند، اما اشکالزدایی باید با هر استراتژی آزمایش همراه باشد.

یک استراتژی برای آزمایش نرم‌افزار باید آزمایش های سطح پایینی را هدایت کند که برای بازبینی صحت پیاده‌سازی یک قطعه کد کوچک لازم می‌باشند. همچنین این استراتژی باید آزمایش های سطح بالایی را سازماندهی کند که اکثر توابع سیستم را در رابطه با نیازهای مشتری اعتبارسنجی می‌نمایند. یک استراتژی باید راهنمایی‌هایی را برای مجری و مجموعه‌ای از علایم نشان دهنده را برای مدیر فراهم نماید. چون این مراحل استراتژی آزمایش زمانی انجام می‌شوند که فشار مربوط به پایان مهلت، شروع به افزایش می‌نماید، پیشرفت باید قابل اندازه‌گیری باشد و مشکلات باید تا حد امکان به سادگی برطرف شوند.

اصول آزمایش نرم‌افزار

آزمایش، موارد غیر معمول جالبی را برای مهندس نرم‌افزار آشکار می‌نماید. در ضمن فعالیت‌های اولیهٔ مهندسی نرم‌افزار، مهندس، سعی در ایجاد نرم‌افزار با استفاده از مفهومی مجرد و بدست آوردن محصولی واضح و کامل دارد. اینک آزمایش باید انجام شود. این مهندس یک‌سری نمونه های آزمایش ایجاد می‌کند که باید نرم‌افزار ایجاد شده را با شکست روبرو نماید. در واقع، آزمایش، یک مرحله در فرآیند نرم‌افزار است که می‌تواند به عنوان فرآیندی مخرب به جای سازنده در نظر گرفته شود (حداقل از نظر روانشناسی). به هر حال هدف از آزمایش چیزی متفاوت از آنچه انتظار می رود!

صحت و اعتبارسنجی

آزمایش نرم‌افزار یک عنصر از عنوان گسترده‌تری است که اغلب با **صحت و اعتبارسنجی** (V&V-Validation & verification) شناخته می‌شود. صحت اشاره به مجموعه فعالیت‌هایی دارد که مطمئن می‌سازند نرم‌افزار به

درستی یک تابع خاص را پیاده‌سازی می‌نماید. اعتبارسنجی اشاره به مجموعه‌ای متفاوت دارد که مطمئن می‌سازند نرم‌افزاری که ایجاد شده منطبق بر نیازهای مشتری است. Boehm این مطلب را این گونه بیان می‌کند:

صحت: " آیا محصول را درست ایجاد می‌کنیم؟

اعتبارسنجی: " آیا محصول درستی را ایجاد می‌کنیم؟

تعریف V&V شامل بسیاری از فعالیت‌هایی است که تضمین کیفیت نرم‌افزار (SQA) نامیده می‌شوند. صحت و اعتبارسنجی شامل گروه وسیعی از فعالیت‌های SQA می‌باشد شامل: مرورهای فنی رسمی، بررسی کیفیت و پیکربندی، نظارت بر کارایی، شبیه‌سازی، امکان‌سنجی، مرور مستندات، مرور بانک اطلاعاتی، تحلیل الگوریتم، آزمایش توسعه، آزمایش کیفی، و آزمایش نصب. اگرچه آزمایش نقش بسیار مهمی را در V&V دارد، بسیاری از فعالیت‌های دیگر نیز لازم می‌باشند.

اهداف آزمایش

در مورد آزمایش نرم‌افزار، Myers چند قانون زیر را بیان می‌کند که اهداف مناسبی برای آزمایش هستند:

۱- **آزمایش فرآیندی است شامل اجرای برنامه با هدف یافتن خطا.**

۲- **یک نمونه آزمایش خوب، نمونه ای است که با احتمال بالایی خطاها را بیابد.**

۳- **آزمایش موفق، آزمایشی است که خطاهای یافت نشده تاکنون را بیابد.**

این اهداف تغییری دراماتیک را در دیدگاه ایجاد می‌نمایند. این اهداف باعث تغییر در دیدگاه متداولی می‌شوند که آزمایش موفق را آن نوع آزمایش می‌داند که در آن خطایی یافت نشود. هدف، طراحی آزمایش هایی است که به طور سیستماتیک رده‌های متفاوتی از خطاها را آشکار نمایند، و این عمل را با حداقل مقدار زمان و فعالیت انجام دهند.

اصول آزمایش

قبل از بکارگیری روش های طراحی نمونه های مؤثر آزمایش، مهندس نرم‌افزار باید اصول اولیه‌ای را که آزمایش نرم‌افزار را هدایت می‌کنند بفهمد. Davis مجموعه‌ای از اصول آزمایش را پیشنهاد می‌کند:

- **تمام آزمایش های باید براساس نیازهای مشتری قابل پیگیری باشند.**
- **آزمایش ها باید مدتی طولانی قبل از شروع آزمایش برنامه‌ریزی شوند.**
- **اصل Pareto برای آزمایش نرم‌افزار بکار گرفته شود.**
- **آزمایش باید با " توجه به اجزاء " شروع شود و به سمت آزمایش " کلی " پیش رود.**
- **آزمایش کامل و جامع امکان‌پذیر نیست.**
- **به منظور داشتن بیشترین تأثیر، آزمایش باید توسط تیم مستقلی هدایت شود.**

قابلیت آزمایش

در موارد ایده‌آل، مهندس نرم‌افزار، برنامه‌ای کامپیوتری، سیستم، یا محصولی را با در نظر داشتن **قابلیت آزمایش** طراحی می‌کند. این مساله باعث می‌شود افرادی که مسوول آزمایش هستند، نمونه هاای آزمایشی مؤثر را ساده‌تر ایجاد نمایند. اما قابلیت آزمایش چیست؟ Bach James قابلیت آزمایش را این گونه توصیف می‌کند:

عملیاتی بودن(Operability). " نرم افزار هرچه بهتر کارکند، با کارایی بالاتری آزمایش می‌شود. "

- سیستم اشکالات اندکی دارد (اشکالات، تحلیل و گزارش اضافی را بر فرآیند آزمایش تحمیل می‌کنند).
- هیچ اشکالی، اجرای آزمایشات را متوقف نکند.
- محصول در مراحل عملیاتی تکامل می‌یابد (توسعه و آزمایش همزمان را امکان‌پذیر می‌نماید).

قابلیت مشاهده(Observability). " آنچه می‌بینید آزمایش می‌کنید ".

- خروجی‌های مجزا برای هر ورودی تولید می‌شوند.
- حالت‌های سیستم و متغیرها در ضمن اجرا قابل رؤیت و قابل پرس و جو باشند.
- حالت‌های قبلی سیستم و متغیرها قابل پرس و جو می‌باشند (برای مثال، ثبت تراکنشها).
- تمام فاکتورهای مؤثر بر خروجی قابل رؤیت باشند.
- خروجی غلط به راحتی مشخص شود.
- خطاهای داخلی به طور خودکار از طریق مکانیزم‌های خودآزمایی آشکار شوند.
- خطاهای داخلی به طور خودکار گزارش شوند.
- برنامه های مبدأ قابل دسترسی باشد.

قابلیت کنترل(Controllability). " هر چه نرم‌افزار بهتر کنترل شود، آزمایش بیشتر به طور خودکار و بهینه قابل انجام است. "

- تمام خروجی‌های ممکن نمی‌توانند از طریق برخی ترکیبات ورودی تولید شوند.
- تمام دستورات از طریق برخی ترکیبات ورودی قابل اجرا باشند.
- حالت‌ها و متغیرهای نرم‌افزار و سخت‌افزار مستقیماً توسط آزمایش کننده قابل کنترل باشند.
- قالب‌های ورودی و خروجی یکنواخت و ساختیافته باشند.
- آزمایش ها می توانند به طور مناسبی مشخص شوند، و به طور خودکار انجام گیرند و دوباره تولید گردند.

تجزیه‌پذیری(Decomposability). " با کنترل نمودن محدودهٔ آزمایش، با سرعت بیشتری مسایل تجزیه می‌شوند و آزمایش های هوشمندانه‌تری انجام می‌گیرد ".

- سیستم نرم‌افزار از پیمانه‌های مستقل ساخته می‌شود.
- پیمانه‌های نرم‌افزاری به طور مستقل قابل آزمایش هستند.

سادگی(Simplicity). هر چه مورد برای آزمایش کمتر باشد، آزمایش با سرعت بیشتری انجام می گیرد. "

- سادگی تابعی (برای مثال، مجموعه جنبه‌هایی که حداقل لازم برای دستیابی به نیازها هستند).
- سادگی ساختاری (برای مثال، معماری پیمانه‌بندی می‌شود تا انتشار اشکالات را محدود نماید).
- سادگی برنامه های مبداء (برای مثال، استاندارد کدنویسی برای سهولت بازبینی و نگهداری تعریف می‌شود).

پایداری(Stability). " هر چه تغییرات کمتر باشد، انحراف از آزمایش کمتر است. "

- تغییرات در نرم‌افزار غیرمتداول هستند.
- تغییرات در نرم‌افزار کنترل شده هستند.
- تغییرات در نرم‌افزار، آزمایش های موجود را نامعتبر نمی‌سازند.
- نرم‌افزار از شکست‌ها به خوبی خارج می‌شود.

قابلیت فهم(Understandability). " هر چه اطلاعات بیشتری در اختیار داشته باشیم، آزمایش هوشمندانه‌تری انجام می‌شود. "

- طراحی کاملاً قابل فهم است.
- وابستگی‌های بین مؤلفه‌های داخلی، خارجی، و اشتراکی کاملاً قابل فهم هستند.
- تغییرات طراحی منتقل می‌شوند.
- مستندات فنی قابل دسترسی است.
- مستندات فنی به طور مناسبی سازماندهی شده است.
- مستندسازی فنی دقیق انجام شده است.

طراحی نمونه های آزمایش

طراحی آزمایش هایی برای نرم‌افزار و محصولات مهندسی دیگر می‌تواند به اندازهٔ طراحی اولیهٔ خود محصول متغیر باشد. با این وجود، مهندسین نرم‌افزار اغلب با آزمایش به عنوان فعالیتی نهایی برخورد می‌نمایند، و نمونه های آزمایشی طراحی می‌کنند که ظاهراً درست هستند اما اطمینان کمی از کامل بودن آنها وجود دارد. اهداف آزمایش را به خاطر آورید، براساس آنها آزمایش هایی باید طراحی شوند که احتمال بالایی برای یافتن اکثر خطاها، با حداقل مقدار زمان و فعالیت داشته باشند.

مجموعه‌ای غنی از روش های طراحی نمونه های آزمایش برای نرم‌افزار تکامل یافته‌اند. این روش ها برای توسعه دهنده، روشی سیستماتیک را برای آزمایش فراهم می‌کنند. مهمتر این که، این روش ها مکانیزمی را فراهم می‌کنند که به اطمینان از کامل بودن آزمایش ها کمک می‌کند و احتمال بالایی برای کشف خطاهای نرم‌افزاری را نیز تضمین می‌نمایند.

هر محصول مهندسی (و اکثر چیزهای دیگر) می‌تواند به یکی از این دو روش زیر آزمایش شود:

۱. **با دانستن تابع خاصی که یک محصول برای انجام آن طراحی شده،** آزمایش هایی طراحی می‌شوند که مشخص کنند هر تابع کاملاً عملیاتی است در حالی که در عین حال در هر تابع برای یافتن خطاها جستجو نیز انجام می‌گیرد**(آزمایش جعبه سیاه)**.

۲. **با دانستن عملکرد داخلی محصول،** آزمایش ها به گونه‌ای طراحی می‌شوند که تعیین نماید اعمال داخلی مطابق با مشخصه‌ها انجام می‌شوند و تمام مؤلفه‌های داخلی به طور مناسبی آزمایش می گردند**(آزمایش جعبه سفید)**.

آزمایش جعبهٔ سفید

آزمایش جعبهٔ سفید، که گاهی آزمایش جعبهٔ شیشه‌ای نامیده می‌شود، یک روش طراحی نمونه های آزمایش است که از ساختار کنترل طراحی رویه‌ای برای هدایت نمونه های آزمایش استفاده می‌کند. با استفاده از روش های آزمایش جعبهٔ سفید، مهندس نرم‌افزار می‌تواند نمونه های آزمایشی را بدست آورد که

۱- تضمین نمایند که تمام مسیرهای مستقل داخل پیمانه حداقل یک بار آزمایش شوند،

۲- تمام تصمیمات شرطی را در دو بخش درست و غلط بررسی نمایند،

۳- تمام حلقه‌ها را در شرایط مرزی و در محدوده‌های عملیاتی اجرا کنند، و

۴- ساختمان داده‌های داخلی را بررسی نمایند تا از اعتبار آنها مطمئن شوند.

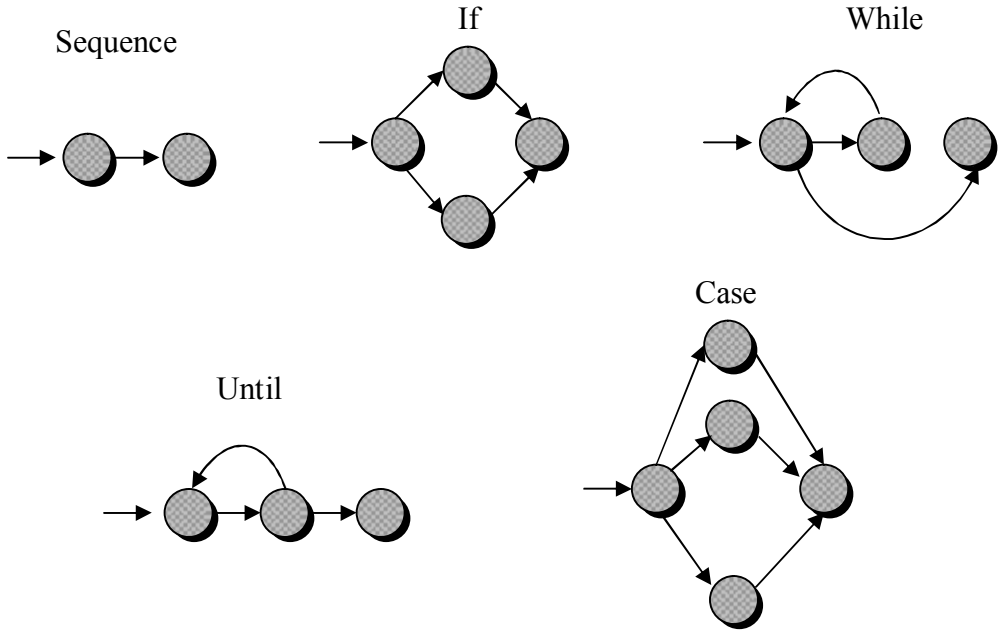
به منظور انجام آزمایش جعبه سفید با دو بحث مسیر پایه و گراف جریان مطرح گردد.

آزمایش مسیر پایه

آزمایش مسیر پایه یک تکنیک آزمایش جعبهٔ سفید است که ابتدا توسط Mc Cabe پیشنهاد شد. روش مسیر پایه، طراح نمونه های آزمایش را وادار می‌نماید که اندازهٔ پیچیدگی منطقی طراحی رویه‌ای را بدست آورد و این اندازه را به عنوان راهنمایی برای تعریف مجموعهٔ پایه مسیرهای اجرایی به کار ببرد. مسیر پایه تضمین می کند که با اجرای نمونه های آزمایش بدست آمده، هر دستور برنامه حداقل یک بار در ضمن آزمایش اجرا می گردد.

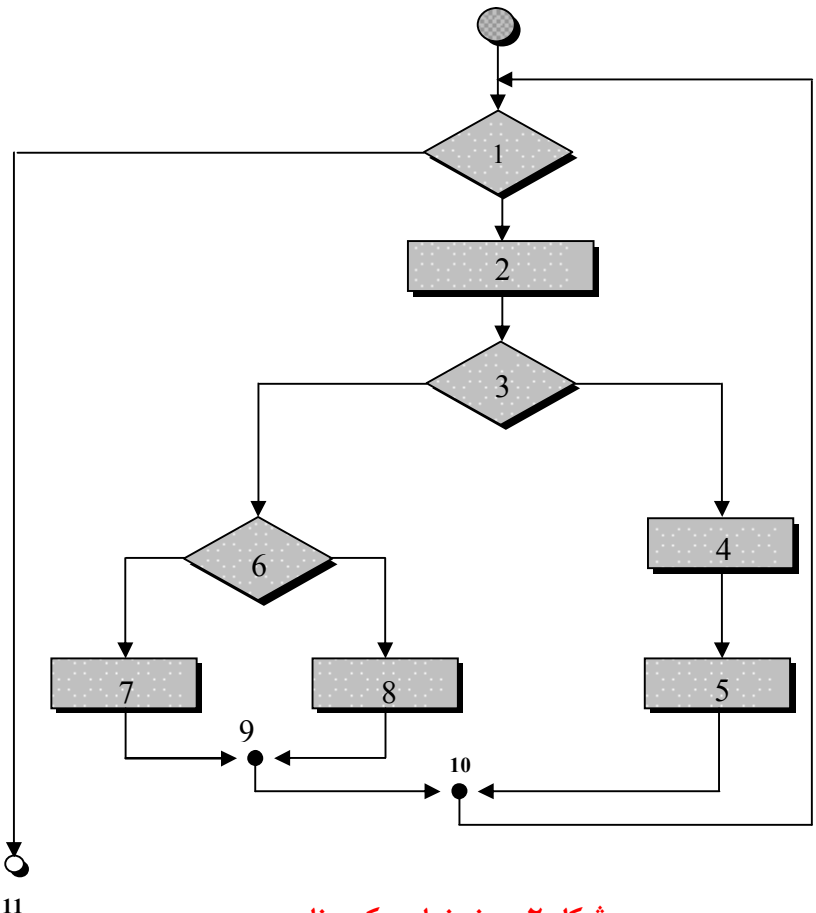
گراف جریان

قبل از معرفی روش مسیر پایه، یک نشان‌گذاری ساده برای نمایش جریان کنترل به نام **گراف جریان**(یا گراف برنامه) باید معرفی شود. گراف جریان، جریان کنترل منطقی را با استفاده از نشان‌گذاری نمایش داده شده در شکل ۱ نشان می‌دهد. هر واحد ساختاری (فصل ۱۶) یک نماد متناظر گراف جریان دارد.

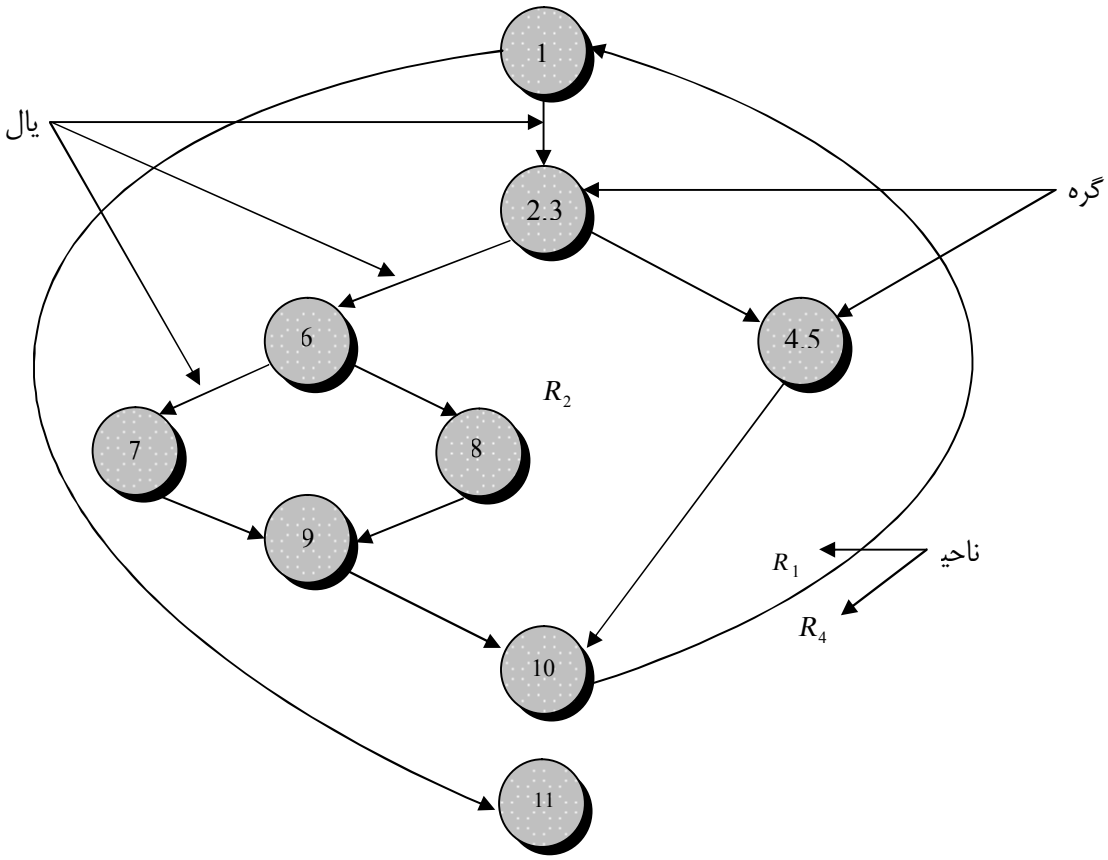


شکل ۱ : نمادگذاری گراف جریان

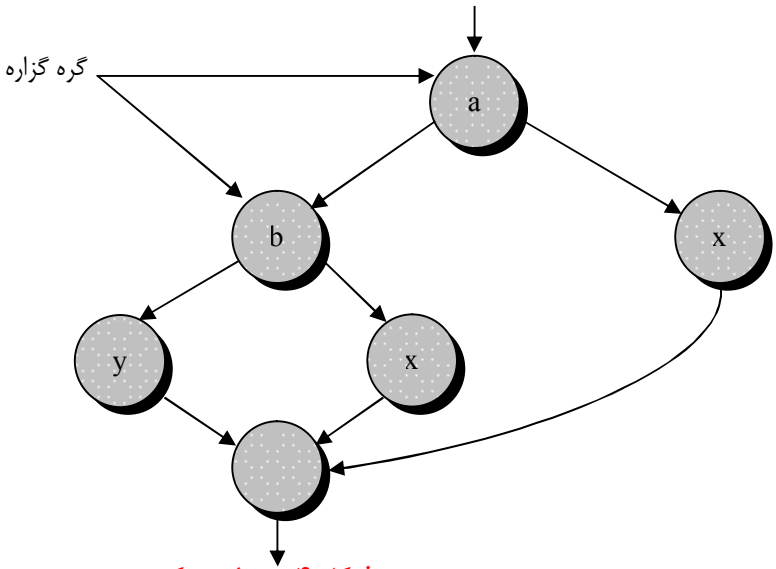
هردایره نشان دهندهٔ یک یا چند PDL بدون انشعاب یا دستورات برنامه مبداء می‌باشد. به مثال زیر توجه کنید.



شکل ۲ : روند نمای یک برنامه



شکل ۳ : گراف جریان مساله فوق



IF a OR b
then procedure x
else procedure y
ENDIF

شکل ۴ : منطق ترکیبی

پیچیدگی دورانی(Cyclomatic Complexity)

پیچیدگی دورانی، معیاری از نرم‌افزار است که اندازه‌ای مقداری از پیچیدگی منطقی برنامه را مشخص می‌کند. وقتی در رابطه با روش آزمایش مسیر پایه استفاده می‌شود، مقدار محاسبه شده برای پیچیدگی دورانی، تعداد مسیرهای مستقل

را در مجموعهٔ پایه برنامه مشخص می‌کند و حد بالایی را برای تعداد آزمایش هایی مشخص می‌نماید که باید برای اطمینان از اجرای حداقل یک بار هر یک از دستورات انجام شوند.

یک مسیر مستقل، هر مسیری در برنامه است که حداقل یک مجموعهٔ جدید از احکام پردازش یا شرط جدیدی را مشخص می‌کند. هنگامی که مسیر مستقل، برحسب گراف جریان بیان می‌شود، باید حداقل در مسیر یالی حرکت کند که قبل از تعریف آن مسیر، از آن عبور نشده باشد. برای مثال، مجموعه‌ای از مسیرهای مستقل برای گراف جریان در شکل ۳ عبارتند از:

مسیر ۱: ۱ – 11

مسیر ۲ : ۱ – 2 – 3 – 4 – 5 – 10 – 1 – 11

مسیر ۳ : ۱ – 2 – 3 – 6 – 8 – 9 – 10 –1– 11

مسیر ۴ : ۱ – 2 – 3 – 6 – 7 – 9 – 10 –1– 11

توجه داشته باشید که هر مسیر جدید، یک یال جدید را شامل می‌شود. مسیر زیر:

1 – 2 – 3 – 4 – 5 – 10 – 1 – 2 – 3 – 6 – 8 – 9 – 10 – 1 – 11

به عنوان مسیر مستقل در نظر گرفته نمی‌شود زیرا فقط ترکیبی از مسیرهای مشخص شدهٔ قبلی است و یال جدیدی را پیمایش نمی‌کند.

وقتی مسیرهای پایه تعیین گردند، باید برای هر مسیر پایه نمونه های آزمایش طراحی کرد و سپس آنها را برای یافتن خطا های احتمالی اجرا نمود.

پیچیدگی دورانی به یکی از این سه شکل زیر محاسبه می‌شود:

۱- **تعداد نواحی گراف جریان متناظر با پیچیدگی دورانی می‌باش**د.

۲- **پیچیدگی دورانی، V(G)، برای گراف جریان، G، به این صورت تعریف می‌شود:**

$$V(G) = E - N + 2$$

که E تعداد یال های گراف جریان، و N تعداد گره‌های گراف جریان می‌باشد.

۳- **پیچیدگی دورانی، V(G)، برای گراف جریان، G، به این صورت نیز تعریف می‌شود:**

$$V(G) = P + 1$$

که P تعداد گزاره‌های موجود در گراف جریان G می‌باشد.

با مراجعه مجدد به گراف جریان شکل ۳، پیچیدگی دورانی با استفاده از هر یک از الگوریتم‌های ذکر شده این گونه محاسبه می‌شود:

۱- گراف جریان دارای چهار ناحیه است.

۲- $V(G) = 11 - \text{یال} + 9 + 2 = 4$ گره

۳- $V(G) = \text{گزاره} + 3 + 1 = 4$ گرهٔ

بنابراین، پیچیدگی دوره‌ای گراف جریان شکل ۳ برابر 4 است.

ماتریس‌های گراف

رویه‌ای برای بدست آوردن گراف جریان و حتی مشخص نمودن مجموعه‌ای از مسیرهای پایه، برای مکانیزه نمودن مناسب می‌باشد. به منظور توسعهٔ ابزاری نرم‌افزاری که بر پایه آزمایش مسیر عمل می‌کند، ساختمان داده‌ای به نام **ماتریس گراف** بسیار مفید است.

ماتریس گراف، ماتریس مربعی است که اندازهٔ آن (یعنی تعداد سطرها و ستونها) برابر است با تعداد گره‌ها در گراف جریان که هر سطر و ستون معادل یک گرهٔ مشخص شده است، و وارده‌های ماتریس معادل ارتباطات (یال های) بین گره‌ها می‌باشند. یک مثال ساده از گراف جریان و ماتریس گراف معادل آن در شکل ۵ نشان داده شده است.

گره \ متصل به گره	۱	۲	۳	۴	۵
۱			a		
۲					
۳		d		b	
۴		c			f
۵		g	e		

شکل ۵ : ماتریس گراف متناظر با گراف جریان

با مراجعه به این شکل، هر گره در گراف جریان با عدد مشخص می‌شود، در حالی که هر یال با یک حرف مشخص می‌گردد. یک حرف در ماتریس در محلی متناظر با ارتباط بین دو گره وارد می‌شود. برای مثال، گرهٔ 3 به گرهٔ 4 با یال b متصل می‌گردد. تا این مرحله، ماتریس گراف چیزی بیش از نمایش جدولی گراف جریان نمی‌باشد. به هر حال، با افزودن ارزش اتصال به هر واردهٔ ماتریس، این ماتریس گراف می‌تواند به ابزاری قدرتمند برای ارزیابی ساختار کنترل برنامه در ضمن آزمایش تبدیل شود. این ماتریس ارزش اتصال اطلاعاتی اضافی در مورد جریان کنترل را فراهم می‌نماید. در ساده‌ترین شکل، ارزش اتصال، 1 (وجود ارتباط) یا صفر (عدم وجود ارتباط) می‌باشد. اما به ارزش های ارتباطی خواص جالب دیگری نیز نسبت داده می‌شود:

- احتمال این که یک اتصال (یال) اجرا می‌شود.
- زمان پردازش صرف شده در ضمن پیمایش اتصال.
- حافظهٔ لازم در ضمن پیمایش اتصال.
- منابع لازم در ضمن پیمایش آن اتصال.

به منظور نمایش این مطلب، ساده‌ترین ارزش را به کار می‌بریم تا ارتباط را نشان دهیم (0 یا 1). ماتریس گراف شکل ۵ دوباره در شکل۶ رسم شده است. هر حرف با 1 جایگزین شده است، و نشان می‌دهد که یک ارتباط وجود دارد (صفرها برای وضوح حذف شده‌اند). با نمایشی به این شکل، ماتریس گراف، ماتریس ارتباط نیز نامیده می‌شود.

گره \ متصل به گره	۱	۲	۳	۴	۵	
۱			1			1 – 1 = 0
۲						
۳		1		1		2 – 1 = 1
۴		1			1	2 – 1 = 1
۵		1	1			2 – 1 = 1
						3 + 1 = 4 ■ ← پیچیدگی دوره‌ای

شکل ۶ : ماتریس گراف با در نظر گرفتن ارتباط

آزمایش ساختار کنترل

تکنیک آزمایش مسیر پایه توصیف شده، یکی از چند تکنیک آزمایش ساختار کنترلی است. اگرچه آزمایش مسیر پایه ساده و بسیار مفید است، ولی به تنهایی کافی نیست. در این بخش، حالت‌های دیگر آزمایش ساختار کنترلی بحث می‌شوند. این حالت‌ها پوشش آزمایش را گسترش داده و کیفیت آزمایش جعبه سفید را افزایش می‌دهند.

آزمایش شرط

آزمایش شرط روشی برای طراحی نمونه های آزمایش است که شرط‌های منطقی موجود در یک پیمانهٔ برنامه را بررسی می‌نماید. یک شرط ساده، متغیری بولی یا عبارتی رابطه‌ای است، احتمالاً با عملگر NOT که قبل از آن قرار گرفته. عبارت رابطه‌ای به این شکل است:

$$E_1 < \textbf{عملگر رابطه‌ای} > \mathrm{E}_2$$

که E_1 و E_2 عبارت های محاسباتی هستند و < عملگرد رابطه‌ای > شامل یکی از عملگرهای > و ≤ و = ≠ (نامساوی) <، یا ≥ می‌باشد. یک شرط مرکب تشکیل شده است از دو یا چند شرط ساده، عملگرد بولی، و پرانتزها. فرض می‌کنیم که عملگرهای بولی امکان شرط‌های ترکیبی را با اضافه نمودن OR (|)، AND (&) و NOT (¬) می‌دهند. شرطی بدون عبارت های رابطه‌ای، عبارتی بولی است.

اگر یک شرط غلط باشد، حداقل یک مؤلفهٔ آن شرط غلط خواهد بود. بنابراین، انواع خطاها در شرط به شرح زیر هستند که باید مورد آزمایش قرار گیرند:

- خطاهای عملگر منطقی (غلط/ حذف شده/ عملگرهای بولی اضافی).
- خطای متغیر بولی.
- خطای پرانتزهای بولی.
- خطای عملگر رابطه‌ای.
- خطای عبارت محاسباتی.

آزمایش جریان داده

روش آزمایش جریان داده، مسیرهای آزمایش برنامه را طبق مکانهای تعاریف و استفاده از متغیرهای برنامه انتخاب می‌کند. چند استراتژی آزمایش جریان داده، مطالعه و مقایسه شده‌اند.

به منظور نمایش شیوهٔ آزمایش جریان داده، فرض کنید که به هر دستور در برنامه یک شمارهٔ دستور منحصر به فرد داده شده است و این که هر تابع، پارامترهای خود یا متغیرهای سراسری را تغییر نمی‌دهد.

برای دستوری با شمارهٔ دستورS:

$\mathrm{DEF(S)} = \{\mathrm{X} \mid$ دستور S حاوی تعریف X باشد $\}$

$\mathrm{USE(S)} = \{\mathrm{X} \mid$ دستور S حاوی استفاده‌ای از X باشد $\}$

اگر دستور S، یک دستور if یا حلقه باشد، مجموعهٔ DEF آن خالی است و مجموعهٔ USE آن وابسته به شرط دستور S می‌باشد. تعریف متغیر X در دستور S، در دستور S' زنده است اگر مسیری از دستور S به دستور S' وجود داشته باشد که حاوی تعریف دیگری از X نباشد.

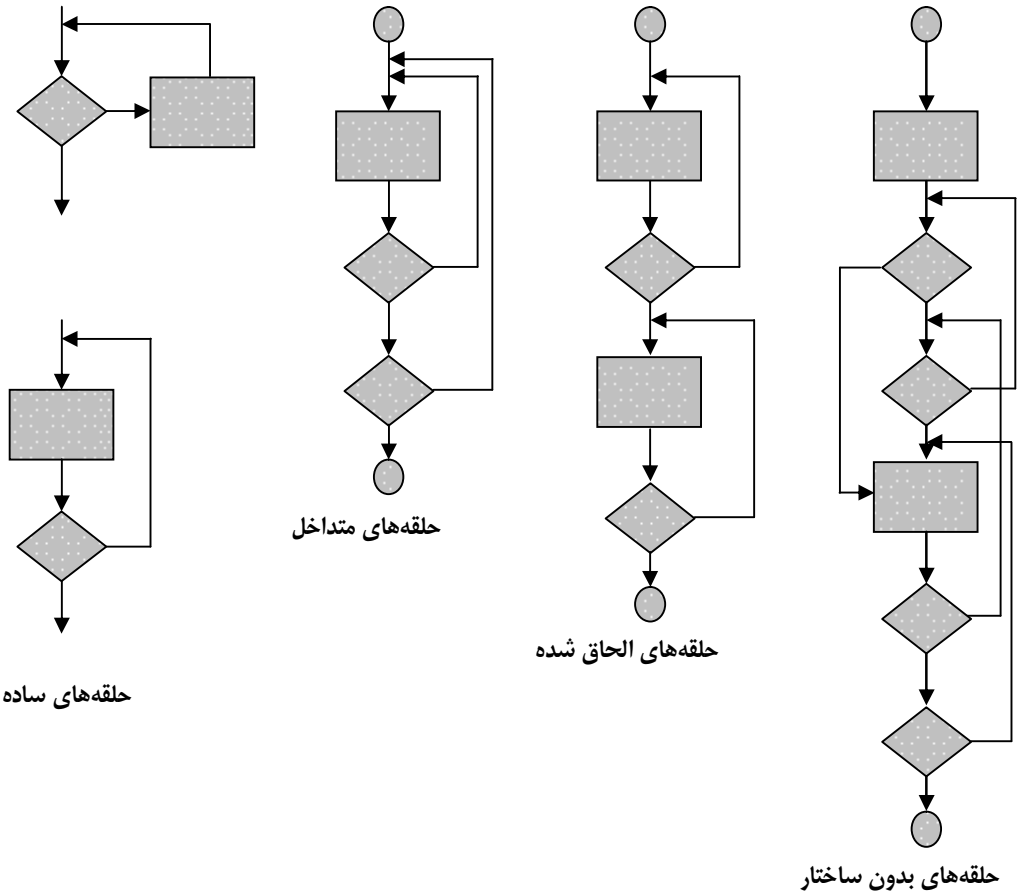
زنجیرهٔ تعریف– استفادهٔ (DU) متغیر X به شکل $\{S'$ و S و $\mathrm{x}\}$ می‌باشد که S و S' شماره‌های دستورات هستند، X در $\mathrm{DEF(S)}$ و $\mathrm{USE}\ (S')$ قرار دارد و تعریف X در دستورS ، در دستور S' زنده است.

یک استراتژی سادهٔ آزمایش جریان داده، نیاز دارد که هر زنجیرهٔ DU حداقل یک دفعه پوشش داده شود. به این استراتژی، **استراتژی آزمایش DU** نیز گفته می‌شود.

نشان داده شده است که آزمایش DU پوشش تمام انشعاب‌های برنامه را تضمین نمی‌کند. به هر حال، تضمینی وجود ندارد که یک انشعاب توسط آزمایش DU پوشش داده شود فقط در موارد نادری مانند ساختارهای if-then-else که در آن، بخش then فاقد تعریف متغیر است و بخش else وجود ندارد. در چنین موقعیتی انشعاب else از دستور if لزوماً توسط آزمایش DU پوشش داده نمی‌شود.

آزمایش حلقه

حلقه‌ها برای اکثریت الگوریتم‌های پیاده‌سازی شده در نرم‌افزار نقش محوری دارند. با این وجود، اغلب در ضمن هدایت آزمایش های نرم‌افزار، توجه کمی به آنها می‌شود.



شکل ۷ : انواع حلقه ها

آزمایش حلقه تکنیکی بر پایه آزمایش جعبه سفید می‌باشد که منحصراً بر اعتبار ساختارهای حلقه تأکید دارد. چهار رده از حلقه‌ها قابل تعریف هستند: **حلقه‌های ساده، حلقه‌های الحاق شده، حلقه‌های متداخل و حلقه‌های بدون ساختار** که در شکل ۷ نشان داده شد است.

آزمایش جعبه سیاه

آزمایش جعبه سیاه که آزمایش رفتاری نیز نامیده می‌شود، بر نیازهای تابعی نرم‌افزاری تأکید دارد. یعنی، آزمایش جعبه سیاه باعث می‌شود مهندس نرم‌افزار مجموعه‌هایی از شرایط ورودی را بدست آورد که کاملاً تمام نیازهای تابعی برنامه را بررسی می‌کنند. آزمایش جعبهٔ سیاه راه جایگزینی برای تکنیک جعبهٔ سفید نیست. در عوض، روشی تکمیلی است که احتمالاً ردهٔ متفاوتی از خطاها را نسبت به روش های جعبهٔ سفید آشکار می‌کند.

آزمایش جعبهٔ سیاه سعی در یافتن خطاهایی در دسته‌بندی‌های زیر دارد:

۱- توابع غلط یا حذف شده،
۲- خطاهای واسط ها،
۳- خطا در ساختمان داده‌ها یا دسترسی به بانک اطلاعاتی خارجی،
۴- خطاهای رفتاری یا کارایی، و
۵- خطاهای آماده‌سازی و اختتامیه.

برخلاف آزمایش جعبهٔ سفید که در اوایل فرآیند آزمایش انجام می‌شود، آزمایش جعبهٔ سیاه در مراحل آخر آزمایش به کار گرفته می‌شود. چون آزمایش جعبهٔ سیاه عمداً به ساختارکنترلی توجهی ندارد، توجه بر دامنهٔ اطلاعات متمرکز می‌باشد. آزمایش های برای پاسخگویی به سؤالات زیر طراحی می شوند:

- **چگونه اعتبار عملکردی آزمایش می‌شود؟**
- **چگونه رفتار و کارایی سیستم آزمایش می‌شود؟**
- **چه رده‌هایی از ورودی، نمونه های آزمایش خوبی می‌سازند؟**
- **آیا سیستم مخصوصاً به مقادیر خاص ورودی حساس است؟**
- **چگونه مرزهای یک رده از داده‌ها مجزا می‌شود؟**
- **سیستم چه نواساناتی برای سرعت و حجم داده‌ها دارد؟**
- **ترکیبات خاص داده‌ها چه اثری بر عملکرد سیستم دارند؟**

با بکارگیری روش های آزمایش جعبه سیاه، مجموعه‌ای از نمونه های آزمایشی بدست می‌آیند که معیارهای زیر را برآورده می‌سازند:

۱- نمونه های آزمایشی که باعث کاهش بیش از حد یک واحد از تعداد نمونه های آزمایشی می‌شوند که برای رسیدن به آزمایش قابل قبول مورد نیاز می‌باشند، و
۲- نمونه های آزمایشی که چیزی در مورد حضور یا عدم حضور رده‌هایی از خطاها ارایه دهند. به جای اینکه یک خطا مربوط به یک آزمایش خاص در حال انجام را آشکار نمایند.

تحلیل مقدار مرزی

به دلایلی که کاملاً واضح نیستند، تعداد زیادی از خطاها در مرزهای دامنه ورودی اتفاق می‌افتند، به جای این که در مرکز آن اتفاق بیفتند. به این دلیل است که تحلیل مقدار مرزی (BVA-Boundary Value Analysis) به عنوان یک روش آزمایش توسعه داده شده است. تحلیل مقدار مرزی باعث انتخاب نمونه های آزمایشی می‌شود که مقادیر مرزی را مورد آزمایش قرار می‌دهند.

تحلیل مقدار مرزی یک تکنیک طراحی نمونه های آزمایش می‌باشد که مکمل تقسیم‌بندی مساوی است. به جای انتخاب هر عنصر از ردهٔ مساوی، BVA، انتخاب نمونه های آزمایش را به لبه‌های این رده هدایت می‌کند. به جای تمرکز بر شرایط ورودی، BVA، نمونه های آزمایش را از دامنهٔ خروجی بدست می‌آورد. رهنمودهای زیر در این آزمون راهگشا است:

۱- اگر شرط ورودی محدوده‌ای را با مقادیر a و b مشخص نماید، نمونه های آزمایش باید با مقادیر a و b و اندکی بالاتر و پایین تر از a و b طراحی شوند.
۲- اگر شرط ورودی چند مقدار را مشخص نماید، نمونه های آزمایش باید به گونه‌ای توسعه یابند که اعداد حداکثر و حداقل را بررسی نمایند. مقادیر اندکی بالاتر و پایین‌تر از حداقل و حداکثر نیز آزمایش می‌شوند.
۳- به کارگیری راهنمایی‌های 1 و 2 برای شراط خروجی. برای مثال، فرض کنید که جدول دما در مقابل فشار، به عنوان خروجی یک برنامهٔ تحلیل مهندسی لازم است. نمونه های آزمایش باید برای تولید یک گزارش خروجی طوری ایجاد شوند که حداقل و حداکثر عدد مجاز وارده‌های جدول را تولید نمایند.

۴- اگر ساختمان داده‌های داخل برنامه مرزهای مشخصی دارند (برای مثال، آرایه با محدودیت 100 وارده تعریف شده باشد)، از طراحی نمونه های آزمایشی که این ساختمان داده‌ها و مرزهای آنها را بررسی می‌کند مطمئن شوید.

اکثر مهندسین نرم‌افزار BVA را با درجه‌ای خاص اجرا می‌کنند. با به کارگیری این راهنمایی‌ها، آزمایش مرزی کامل‌تر خواهد شد، و احتمال بیشتری برای آشکارسازی خطا وجود دارد.

آزمایش برای محیط‌ها، معماری‌ها و کاربردهای خاص

نرم‌افزارهای کامپیوتر پیچیده‌تر شده، و نیاز برای شیوه‌های آزمایش خاص نیز رشد نموده است. روش های آزمایش جعبهٔ سیاه و جعبهٔ سفید بحث شده، برای تمام محیط‌ها، معماری‌ها، و کاربردها قابل به کارگیری هستند، اما راهنمایی‌های منحصر به فرد و شیوه‌هایی برای آزمایش گاهی توصیه می‌شوند. در این بخش، راهنمودهای آزمایش محیط‌ها، معماری‌ها و کاربردهای خاصی که به طور متداول مهندسین نرم‌افزار با آنها روبرو می‌شوند ارایه شده است.

آزمایش واسط های گرافیکی کاربران(GUI-Graphical User Interface)

واسط‌های گرافیکی کاربران(GUI ها) زمینهٔ جالبی را برای مهندسین نرم‌افزار ارایه می‌نمایند. به علت اجزاء قابل استفادهٔ مجددی که به عنوان بخشی از محیط‌های توسعهٔ GUI فراهم می‌شوند، ایجاد واسط کاربر زمان کمتری نیاز دارد و دقیق‌تر است. اما در عین حال، پیچیدگی GUIها نیز افزایش یافته است، و باعث مشکلات بیشتر در طراحی و اجرای نمونه های آزمایش می‌شود.

چون بسیاری از GUIهای مدرن، احساس و جلوهٔ یکسانی دارند، یک سری از آزمایش های استاندارد قابل انجام است. گراف های مدلسازی حالت محدود می‌توانند مورد استفاده واقع شوند تا یکسری آزمایش هایی را ایجاد کنند که اشیاء و داده‌های خاص مربوط به GUI برنامه را مورد توجه قرار دهند.

به دلیل ترکیبات زیاد اعمال GUI، آزمایش باید با نمونه های خودکار انجام شود. دستهٔ بزرگی از نمونه های آزمایش GUI در بازار در چند سال گذشته عرضه شده‌اند.

آزمایش معماری مشتری/ کارگزار

معماری‌های مشتری/ کارگزار (C-S) موارد مهمی را برای آزمایش کننده‌های نرم‌افزار نشان می‌دهند. ماهیت توزیع شدهٔ محیط‌های C-S، موارد کارایی مربوط به پردازش تراکنش، حضور بالقوهٔ سکوهای سخت‌افزاری متفاوت، پیچیدگی های ارتباط شبکه، نیاز به رایه سرویس به چندین مشتری از بانک اطلاعاتی متمرکز (یا توزیع شده)، و نیازهای هماهنگ‌سازی تحمیل شده بر کارگزار، همگی ترکیب می‌شوند و باعث می‌شوند آزمایش معماری‌های C-S، و نرم‌افزاری که بر روی آنها قرار می‌گیرد، تا حد قابل توجهی مشکل‌تر از کاربردهای مجزا باشد. در واقع، مطالعات اخیر صنایع نشان می‌دهد که افزایش عمده‌ای در زمان و هزینهٔ آزمایش محیط‌های C-S وجود دارد.

مستندسازی آزمایش و امکانات کمک

واژهٔ آزمایش نرم‌افزار، شامل تصاویری از تعداد زیادی از نمونه های آزمایش می‌باشد که برای بررسی برنامه‌های کامپیوتری و داده‌هایی که دستکاری می‌کنند آماده شده‌اند. تعریف نرم‌افزار را که در اولین فصل ارایه شد به خاطر آورید، توجه به این نکته مهم است که آزمایش باید به سومین عنصر پیکربندی نرم‌افزار، یعنی مستندات، توسعه یابد.

خطاها در مستندات می‌توانند به همان اندازهٔ خطاها که در برنامه های مبدأ یا داده‌ها، باعث اشکالاتی در قبول برنامه شوند. هیچ نگران کننده‌تر از این نمی‌باشد که راهنمای کاربر یا امکان کمک سیستم دقیقاً دنبال شود و به نتایج یا رفتاری منتهی شود که منطبق با آنچه توسط مستندات پیش‌بینی شده نباشد. به این دلیل است که آزمایش مستندات باید بخشی معنی‌داری برای هر طرح آزمایش نرم‌افزار باشد.

آزمایش مستندات در دو فاز قابل انجام است. اولین فاز که مرور و بازبینی نام دارد (فصل ۸)، مستندات را برای وضوح ویرایشی بررسی می‌کند. فاز دوم، که آزمایش زنده نام دارد، مستندات را همراه با استفاده از برنامهٔ واقعی، مورد استفاده قرار می‌دهد.

به شکل تعجب‌آوری، آزمایش زنده برای مستندات با استفاده از تکنیک‌هایی مشابه بسیاری از روش های جعبهٔ سیاه، قابل انجام است. آزمایش بر مبنای گراف می‌تواند استفاده از برنامه را توصیف کند. تقسیم‌بندی مساوی و تحلیل مقدار مرزی برای تعریف رده‌های گوناگون ورودی و ارتباطات مربوط به آنها استفاده می‌شوند. سپس استفاده از برنامه از طریق مستندات پیگیری می‌شود. سؤالات زیر باید در طول هر فاز پاسخ داده شوند:

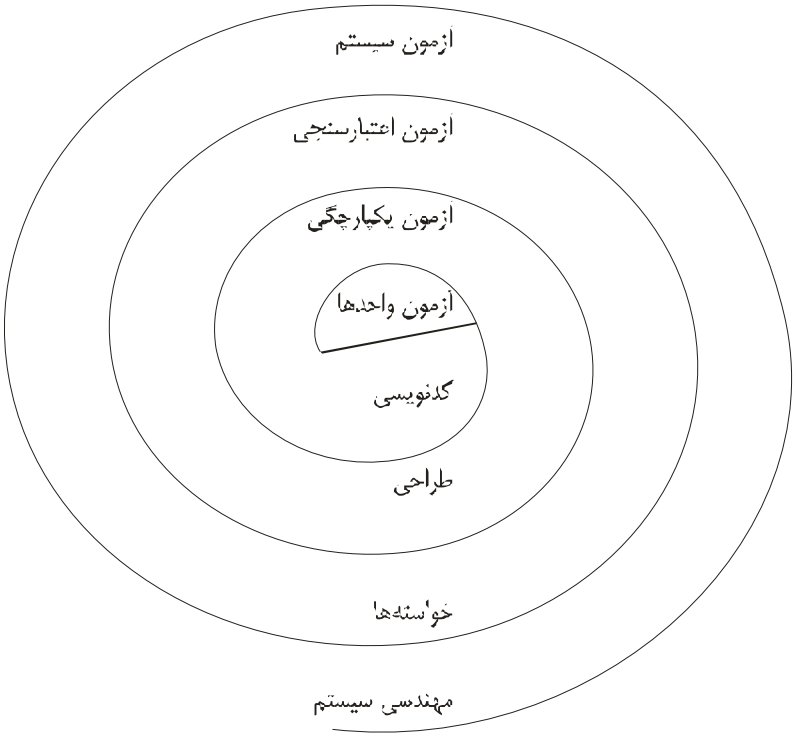
- **آیا مستندسازی به طور دقیق چگونگی استفاده از هر روش را توصیف می کند؟**
- **آیا توصیف هر دنبالهٔ ارتباط دقیق است؟**
- **آیا مثال ها دقیق هستند؟**
- **آیا بکارگیری لغات، توصیف‌های منوها، و پاسخ‌های سیستم منطبق بر برنامهٔ واقعی است؟**
- **آیا یافتن راهنمایی در مستندات نسبتاً ساده است؟**
- **آیا رفع اشکال با استفاده از مستندات به راحتی انجام می‌شود؟**
- **آیا فهرست مندرجات و اندیس مستندات کامل و دقیق است؟**
- **آیا طراحی مستندات (اجزاء، تایپ ظاهری، کنگره‌بندی، گرافیک‌ها) برای فهم و درک سریع اطلاعات مؤثر هستند؟**
- **آیا تمام پیغام های خطای نرم‌افزار که برای کاربر ظاهر می‌شوند، با جزییات بیشتر در مستندات توصیف شده‌اند؟ آیا اعمال قابل انجام در نتیجهٔ پیغام خطا، به وضوح بیان شده‌اند؟**
- **اگر ارتباطات فرامتنی (Hypertext) استفاده می‌شوند، آیا دقیق و کامل هستند؟**
- **اگر فرامتنی استفاده می‌شود، آیا طراحی نحوه حرکت در اتصالات، برای اطلاعات مورد نیاز مناسب است؟**

تنها راه عملی برای پاسخ به این سؤالات، وجود گروهی مستقل (برای مثال، کاربران انتخاب شده) است که مستندات را در رابطه با استفاده از برنامه آزمایش نمایند. تمام تفاوت ها تعیین و ذکر شوند و زمینه‌های ابهام یا ضعف، برای بازنویسی مجدد تعریف گردند.

استراتژی آزمایش نرم‌افزار

فرآیند مهندسی نرم‌افزار می‌تواند به صورت یک مارپیچی در نظر مانند شکل ۸ در نظر گرفته شود. در ابتدا، مهندس سیستم، نقش نرم‌افزار را تعریف می‌کند و به تحلیل نیازهای نرم‌افزار وارد می‌شود، که دامنه اطلاعات، عملکرد، رفتار، کارایی، محدودیت‌ها، و معیارهای اعتبارسنجی برای نرم‌افزار ایجاد می‌شود. با حرکت در مارپیچ به سمت طراحی و در نهایت برنامه نویسی می‌رسیم. به منظور توسعهٔ نرم‌افزار کامپیوتر، به سمت داخل مارپیچ حرکت می‌کنیم، در طول خطی که سطح انتزاع را در هر دور کاهش می‌دهد.

یک استراتژی برای آزمایش نرم‌افزار می‌تواند در زمینهٔ این مارپیچ مورد توجه قرار گیرد. **آزمایش واحد،** از مرکز مارپیچ شروع می‌شود و بر روی هر واحد(یعنی مؤلفه) نرم‌افزار متمرکز می‌باشد که با برنامه های مبدأ پیاده‌سازی شده است. با حرکت به سمت خارج در مارپیچ، آزمایش به سمت آزمایش **یکپارچه‌سازی** ادامه می‌یابد. این آزمایش بر طراحی و ساخت معماری نرم‌افزار تأکید دارد. با حرکت به اندازه یک دور دیگر به سمت خارج در طول مارپیچ، به آزمایش **اعتبارسنجی** می‌رسیم.

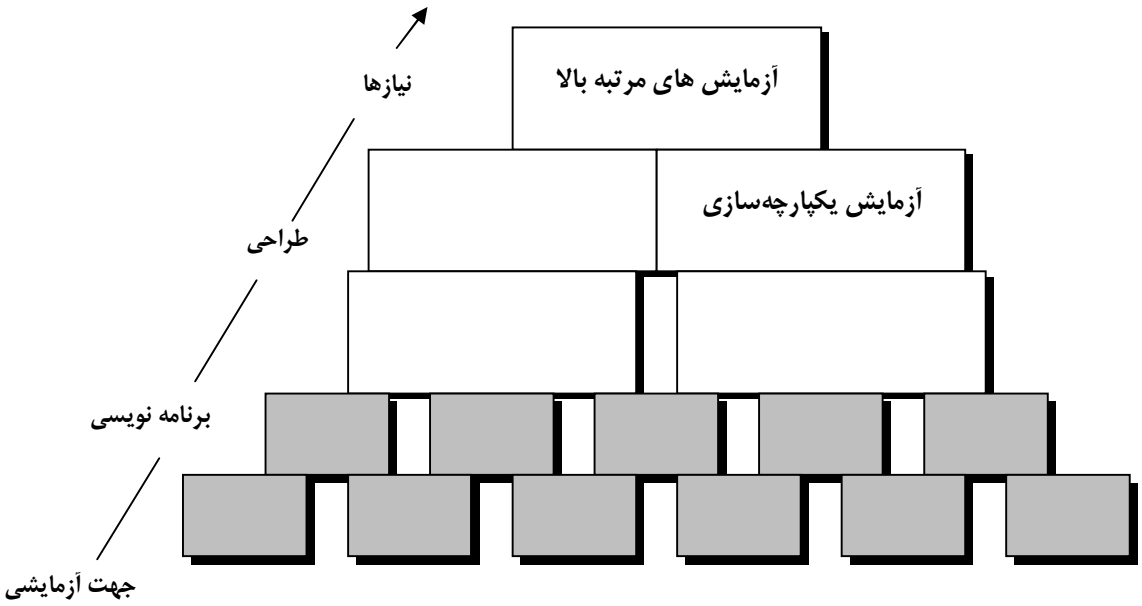


شکل ۸ : استراتژی آزمایش

نیازهایی که به عنوان بخشی از تحلیل نیازهای نرم‌افزار ایجاد شده‌اند، در مقابل نرم‌افزاری که ساخته شده اعتبارسنجی می‌شوند. در نهایت، به آزمایش **سیستم** می‌رسیم، که در آن نرم‌افزار و عناصر دیگر سیستم به صورت یک مجموعه آزمایش می‌شوند. به منظور آزمایش نرم‌افزار کامپیوتر، در طول مارپیچ حرکت نموده و در هر دور، محدودهٔ آزمایش گسترده‌تر می‌شود.

این فرآیند را از نطقه نظر رویه‌ای در نظر بگیرید، آزمایش در زمینهٔ مهندسی نرم‌افزار در واقع شامل چهار مرحله است که به صورت ترتیبی پیاده‌سازی شده‌اند. این مراحل در شکل ۹ نشان داده شده‌اند. در ابتدا، آزمایش بر هر مؤلفه به صورت منفرد تمرکز دارد، تا اطمینان حاصل شود به صورت یک واحد، درست کار می‌کند. در نتیجه، نام این مرحله، آزمایش واحد می‌باشد. **آزمایش واحد، استفاده زیادی از تکنیک‌های آزمایش جعبهٔ سفید می‌برد،** و مسیرهای خاصی را در ساختار کنترلی پیمانه بررسی می‌کند تا اطمینان حاصل شود پوشش کاملی داده شده و حداکثر خطاها آشکار می‌شوند. سپس، مؤلفه‌ها باید مونتاژ یا مجتمع شوند تا بستهٔ نرم‌افزاری کامل را تشکیل دهند. **آزمایش یکپارچه‌سازی، مسایل مربوط به مشکلات دوگانهٔ بازبینی و ساخت برنامه را مورد توجه قرار می‌دهد.** تکنیک‌های **طراحی نمونه های آزمایش جعبهٔ سیاه، در زمان یکپارچه‌سازی بیشترین استفاده را دارند،** اگرچه مقدار محدودی آزمایش جعبهٔ سفید نیز می‌تواند استفاده شود تا از پوشش اکثر مسیرهای کنترلی اطمینان حاصل شود. پس از یکپارچه‌سازی نرم‌افزار (ساخته شدن آن)، مجموعه‌ای از آزمایش های مرتبه بالا هدایت می‌شوند. معیارهای اعتبارسنجی (ایجاد شده در ضمن تحلیل نیازها) باید آزمایش شوند. **آزمایش اعتبارسنجی، اطمینان نهایی را ایجاد می‌کند که نرم‌افزار تمام نیازهای عملکردی، رفتاری، و کارایی را برآورده می‌نماید. تکنیک‌های آزمایش جعبهٔ سیاه به طور انحصاری در ضمن اعتبارسنجی استفاده می‌شوند.**

آخرین مرحلهٔ آزمایش مرتبه بالا، خارج از مرز مهندسی نرم‌افزار است و در مرز زمینهٔ مهندسی سیستم کامپیوتری است. نرم‌افزار، پس از اعتبارسنجی، باید با عناصر دیگر سیستم ترکیب شود(برای مثال، سخت‌افزار، افراد، بانک‌های اطلاعاتی). **آزمایش سیستم بازبینی می‌کند که تمام عناصر به طور منظم مرتبط شده باشند و این که عملکرد و کارایی کل سیستم نیز بدست آمده باشد.**



شکل ۹ : مراحل آزمایش نرم افزار

آزمایش واحد(Unit Testing)

آزمایش واحد، بر فعالیت بازبینی کوچکترین واحد طراحی نرم‌افزار، که مؤلفهٔ نرم‌افزار یا پیمانه نامیده می‌شود تمرکز دارد. با استفاده از توصیف طراحی در سطح مؤلفه به عنوان راهنما، مسیرهای کنترلی مهم آزمایش می‌شوند تا خطاهای موجود در مرز پیمانه پیدا شوند. پیچیدگی نسبی آزمایش ها و خطاهای آشکار شده، با محدودهٔ ایجاد شده برای آزمایش واحد، محدود می‌شود. آزمایش واحد، گرایش به جعبهٔ سفید دارد، و این مرحله می‌تواند به موازات برای چند مؤلفه هدایت شود.

در بین خطاهای متداول محاسبه، این موارد مشاهده می‌شوند:

۱- اولویت محاسباتی نادرست،

۲- اعمال ترکیبی،

۳- آماده‌سازی غلط،

۴- عدم دقت کافی در محاسبه،

۵- نمایش غلط یک نماد محاسباتی.

مقایسه و کنترل جریان تا حد زیادی به یکدیگر نزدیک هستند (یعنی تغییر جریان، معمولاً بعد از مقایسه انجام می‌شود). نمونه های آزمایش باید خطاهایی را از این قبیل آشکار نمایند:

(۱) مقایسهٔ انواع دادهٔ متفاوت،

(۲) عملگرهای منطقی یا اولویت نادرست،

(۳) داشتن انتظار تساوی در زمانی که خطا در دقت محاسبه باعث عدم تساوی می‌شود،

(۴) مقایسهٔ غلط متغیرها،

(۵) خاتمهٔ حلقهٔ نامناسب یا عدم وجود خاتمهٔ حلقه،

(۶) شکست در خروج، زمانی که حلقهٔ نامتناهی تشخیص داده می‌شود،

(۷) اصلاح نامناسب متغیرهای حلقه.

آزمایش یکپارچه سازی(Integration Testing)

یک فرد مبتدی در دنیای نرم‌افزار ممکن است سؤالی به ظاهر ساده را بعد از این که آزمایش واحد بر روی تمام پیمانه‌ها صورت گرفت بپرسد: " اگر همهٔ آنها به تنهایی کار می‌کنند، چرا مشکوک هستید که آنها وقتی کنار هم قرار گیرند کار می‌کنند یا خیر؟ " این مشکل، کنار هم قرار دادن مولفه ها، یعنی ارتباط بین آنها است. داده ممکن است در یک ارتباط ناپدید شود. یک پیمانه ممکن است اثر نامطلوب و ناخواسته‌ای را بر دیگری داشته باشد. زیر توابع، زمانی که با هم ترکیب می‌شوند، ممکن است تابع بزرگتر مورد نظر را تولید نکنند. مقادیر نادقیقی که در هر یک به تنهایی پذیرفته شده‌اند، ممکن است بزرگ شوند و به سطوح غیرقابل قبول برسند. ساختمان داده‌های سراسری ممکن است مشکل‌ساز شود. این لیست همچنان به طور نگران‌کننده‌ای ادامه دارد.

آزمایش یکپارچه‌سازی، روشی سیستماتیک برای ایجاد ساختار برنامه است در حالی که، آزمایش‌ها نیز انجام می‌شوند تا خطاهای مربوط به واسط‌ها آشکار شوند. هدف، دریافت مؤلفه‌های آزمایش واحد و ایجاد ساختار برنامه‌ای است که توسط طراح دیکته شده است.

یکپارچه سازی بالا به پایین

آزمایش یکپارچه‌سازی بالا به پایین، روشی افزایشی برای ایجاد ساختار برنامه است. پیمانه‌ها با حرکت به سمت پایین در سلسله مراتب کنترل، مجتمع می‌شوند. کار، با پیمانهٔ کنترل اصلی (Main Program) شروع می‌شود. پیمانه‌های پایین‌تر(تقریباً پایین‌تر) نسبت به پیمانهٔ کنترل اصلی در این ساختار به صورت عمقی یا سطحی یکپارچه می‌شوند.

یکپارچه‌سازی شامل پنج مرحله است:

۱- پیمانهٔ کنترل اصلی (Main) به عنوان گردانندهٔ آزمایش استفاده می‌شود و جانگهدارها(Stubs) به جای تمام مؤلفه‌هایی که مستقیماً در سطح بعدی پیمانه کنترل اصلی قرار دارند، جایگزین می‌شوند.
۲- برحسب روش مجتمع‌سازی انتخاب شده، (یعنی، سطحی یا عمقی)، در هر مرحله، یک جانگهدار با پیمانهٔ اصلی در سطوح بعدی جایگزین می‌شود.
۳- آزمایش ها در ضمن مجتمع شدن هر مؤلفه هدایت می‌شوند.
۴- با تکمیل هر مجموعه از آزمایش ها، جانگهدار دیگری با مؤلفه اصلی جایگزین می‌شود.
۵- آزمایش رگرسیون انجام می‌شود تا اطمینان حاصل شود خطاهای جدیدی هنوز شناسایی نشده‌اند.

یکپارچه‌سازی پایین به بالا

آزمایش یکپارچه‌سازی پایین به بالا، همانطوری که از نامش مشخص می‌باشد، ساخت و آزمایش را با پیمانه‌های اتمی شروع می‌کند (یعنی، مؤلفه‌های پایین‌ترین سطوح ساختار برنامه). چون مؤلفه‌ها از پایین به بالا کنارهم قرار می‌گیرند، پردازش‌های لازم برای مؤلفه‌های سطح بعدی همیشه در دسترس می‌باشند و نیاز به جانگهدار مرتفع می‌گردد.

یک استراتژی یکپارچه‌سازی پایین به بالا با مراحل زیر پیاده‌سازی می‌شود:

۱- مؤلفه‌های سطح پایین در قالب خوشه‌هایی(Clusters) ترکیب می‌شوند که زیر تابع خاصی از نرم‌افزار را انجام دهند.
۲- گردانندهای (برنامهٔ کنترل کنندهٔ آزمایش)(Drivers) نوشته می‌شود تا ورودی- خروجی نمونه های آزمایش را هماهنگ نماید.
۳- خوشه آزمایش می‌شود.
۴- گردانندهها حذف می‌شوند، خوشه‌ها ترکیب می‌شوند، و حرکت به سمت بالا در ساختار کنترلی ادامه می‌یابد.

آزمایش رگرسیون(Regression Testing)

هر دفعه که پیمانهٔ جدیدی به عنوان بخشی از آزمایش یکپارچه‌سازی افزوده می‌شود، نرم‌افزار تغییر می‌کند. مسیرهای جریان دادهٔ جدیدی ایجاد می‌شوند، I/O جدیدی انجام می‌گیرد، و منطق کنترل جدیدی فراخوانی می‌شود. این تغییرات ممکن است باعث بروز مشکلاتی با توابعی شوند که قبلاً بدون خطا کار می‌کردند. در رابطه با استراتژی آزمایش یکپارچه سازی، آزمایش رگرسیون، اجرای مجدد زیر مجموعه‌ای از آزمایش هایی است که قبلاً انجام شده‌اند تا اطمینان حاصل شود که تغییرات، باعث انتشار اثرات جانبی ناخواسته نشده‌اند. در محدودهٔ وسیع‌تر، آزمایش های موفقیت‌آمیز (از هر نوع) باعث کشف خطاها می‌شوند، و خطاها باید اصلاح شوند. هر زمانی که نرم‌افزار اصلاح می‌شود، جنبه‌ای از پیکربندی نرم‌افزار (برنامه، مستندات، یا داده‌هایی که آنها را حمایت می‌کنند) تغییر می‌نماید. آزمایش رگرسیون می‌تواند به صورت دستی هدایت شود. این عمل با اجرای مجدد زیر مجموعه‌ای از تمام نمونه‌های آزمایش استفاده از ابزارهای خودکار Capture/Playback انجام می‌شود. ابزارهای capture/playback باعث می‌شوند مهندس نرم‌افزار نمونه‌های آزمایش را دریافت کند و با حرکت به عقب، مقایسه‌هایی را انجام دهد.
مجموعه آزمایش رگرسیون (زیرمجموعه‌ای از آزمایش هایی که باید انجام شوند) شامل سه ردهٔ متفاوت از نمونه‌های آزمایش می‌باشد:

- **نمونه‌هایی از آزمایش‌هایی که تمام عملکرد نرم‌افزار را بررسی می‌نمایند.**
- **آزمایش‌های اضافی که بر عملکردهایی از نرم‌افزار تأکید دارند که احتمالاً با این تغییرات تحت تأثیر قرار می‌گیرند.**
- **آزمایش‌هایی که بر مؤلفه‌های تغییر یافته در نرم‌افزار تأکید دارند.**

آزمایش دوُد(Smoke Test)

آزمایش دوُد یک روش یکپارچه‌سازی است که به طور متداول زمانی استفاده می‌وشد که محصولات نرم‌افزاری کم اهمیت توسعه داده می‌شوند. به عنوان مثال، مکانیزمی سریع و مرحله‌ای برای پروژه‌هایی که حساسیت زمانی دارند استفاده می‌شود و به تیم نرم‌افزار امکان می‌دهد پروژه را به تدریج انجام دهد. در نتیجه، روش آزمایش دود شامل فعالیت‌های زیر است:

۱- مؤلفه‌های نرم‌افزاری که به کد ترجمه شده‌اند، در قالب یک " بنا " یکپارچه می شوند. یک بنا شامل تمام فایل‌های داده، کتابخانه‌ها، پیمانه‌های قابل استفاده مجدد، و مؤلفه‌های ایجاد شده با فرآیند مهندسی است که برای پیاده‌سازی یک یا چند تابع محصول مورد نیاز می‌باشند.
۲- یک سری از آزمایش ها طراحی می‌شوند تا خطاهایی را آشکار نمایند که باعث می‌شوند یک بنا به طور منظم عمل خود را انجام ندهد. هدف، یافتن خطاهای بازدارنده‌ای است که بالاترین احتمال به تأخیر انداختن پروژه را دارند.
۳- این بنا، با بناهای دیگر یکپارچه می‌شود و محصول کامل (به شکل جاری) به صورت روزانه با این روش آزمایش می‌گردد. روش یکپارچه‌سازی می‌تواند بالا به پایین یا پایین به بالا باشد.

آزمایش اعتبارسنجی(Validation Testing)

در نتیجهٔ آزمایش یکپارچه‌سازی، نرم‌افزار به طور کامل به صورت یک بسته مونتاژ می‌شود، خطاهای واسطها آشکار و برطرف می‌شوند، و سری نهایی آزمایش‌های نرم‌افزار با عنوان آزمایش اعتبارسنجی شروع می‌شود. اعتبارسنجی می‌تواند به چندین روش تعریف شود، اما یک تعریف ساده این است که اعتبارسنجی موفق است اگر عملکرد نرم‌افزار به صورتی باشد که مورد انتظار کاربر می‌باشد. در این نقطه، یک توسعه‌دهنده نرم‌افزار پرخاشگر ممکن است اعتراض نماید، "چه کسی یا چه چیزی مشخص کنندهٔ انتظارات منطقی است؟ ".

انتظارات منطقی در مشخصهٔ نیازهای نرم‌افزار تعریف شده‌اندکه سندی است توصیف کنندهٔ تمام صفات قابل رؤیت نرم‌افزار. این مشخصه شامل بخشی است به نام معیارهای اعتبارسنجی، اطلاعات موجود در این بخش، مبنایی برای روش آزمایش اعتبارسنجی خواهد بود.

معیارهای آزمایش و اعتبارسنجی

اعتبارسنجی نرم‌افزار از طریق یک سری آزمایش های جعبه سیاه بدست می‌آید که تطابق با نیازها را مشخص می‌کند. یک طرح آزمایش، رده‌هایی از آزمایش را مشخص می‌کند که باید هدایت شوند، و یک رویهٔ آزمایش نمونه‌های آزمایش خاصی را تعریف می‌کند که برای نمایش تطابق با نیازها استفاده می‌شوند. این طرح و رویه، هر دو طراحی می‌شوند تا مطمئن حاصل گردد که تمام نیازهای تابعی برآورده شده‌اند، تمام خصوصیات رفتاری بدست آمده‌اند، تمام نیازهای کارایی حاصل شده‌اند، مستندسازی صحیح است، و نیازهای دیگر برآورده شده‌اند (برای مثال، قابلیت حمل، سازگاری، پوشش دادن به خطا و قابلیت نگهداری).

مرور پیکربندی

یک عنصر مهم فرآیند اعتبارسنجی، مرور پیکربندی است. ماهیت این مرور، حصول اطمینان از این است که تمام عناصر پیکربندی نرم‌افزار به طور مناسب توسعه داده شده باشند، ثبت شده باشند، و شامل جزییات لازم برای مرحله حمایت در دوره زندگی نرم‌افزار باشند. مرور پیکربندی گاهی تطبیق نامیده می‌شود و با جزییات در فصل ۹ بحث شده است.

آزمایش های آلفا و بتا(Alpha & Beta Testing)

برای توسعه دهندهٔ نرم‌افزار غیر ممکن است که پیش‌بینی نماید مشتری به طور صحیح و واقعی برنامه را مورد استفاده قرار می‌دهد. دستورات ممکن است به غلط تعبیر شوند، ترکیبات عجیبی از داده‌ها ممکن است به طور معمول استفاده شود، یک خروجی که برای آزمایش کننده واضح است، ممکن است برای کاربر غیرقابل درک باشد.

هنگامی که نرم‌افزاری متداول برای مشتری ایجاد می‌شود، یک سری آزمایش‌های پذیرش انجام می‌شوند تا باعث شوند مشتری تمام نیازها را اعتبارسنجی نماید. **آزمایش پذیرش** به جای مهندسین نرم‌افزار، توسط کاربر نهایی انجام می‌شود، و می‌تواند شامل هدایت غیر رسمی یا یک سری آزمایش‌های برنامه‌ریزی شده و سیستماتیک باشد. در واقع، آزمایش پذیرش می‌تواند در بازهٔ زمانی هفته‌ها یا ماه‌ها انجام گردد، و خطاهای موجود که ممکن است کارایی سیستم را در طول زمان کاهش دهند، کشف گردند.

اگر نرم‌افزار به صورت بسته نرم‌افزاری توسعه داده شود که توسط کاربران متعددی اجرا می‌گردد، اجرای آزمایش‌های پذیرش با هر یک، غیر عملی خواهد بود. اکثر سازندگان محصولات نرم‌افزاری، فرآیندی را به نام **آزمایش آلفا و بتا** استفاده می‌کنند تا خطاهایی را که به نظر می‌رسد فقط کاربر نهایی می‌تواند بیابد کشف نمایند.

آزمایش آلفا در سایت توسعه دهنده توسط مشتری انجام می‌شود. نرم‌افزار با تنظیمات معمول استفاده می‌شود و توسعه‌دهنده بر آن نظارت دارد و خطاها را ثبت می‌نماید. آزمایش های آلفا در **محیطی کنترل شده** انجام می‌شوند.

آزمایش بتا در یک یا چند سایت مشتری توسط کاربر نهایی نرم‌افزار انجام می‌شود. برخلاف آزمایش آلفا، توسعه دهنده عموماً حضور ندارد. بنابراین آزمایش بتا، بکارگیری زندهٔ نرم‌افزار در محیطی است که توسط توسعه دهنده قابل کنترل نیست. مشتری تمام مشکلات را (واقعی یا خیالی) که در طول آزمایش بتا شناسایی می‌شوند ثبت می‌کند و این گزارشات را به توسعه دهنده در بازه‌های زمانی منظم تحویل می‌دهد. در نتیجهٔ مشکلات گزارش شده در ضمن آزمایش های بتا، مهندسین نرم‌افزار اصلاحات را انجام می‌دهند و برای انتشار محصول نرم‌افزار به مشتری آماده می‌شوند.

آزمایش سیستم(System Testing)

در ابتدای بحث، بر این حقیقت تأکید داشتیم که نرم‌افزار فقط یک عنصر از یک سیستم بزرگ کامپیوتری است. به طور تقریبی، نرم‌افزار با عناصر دیگر سیستم یکپارچه می‌شود(برای مثال، سخت‌افزار، افراد، اطلاعات)، و یک سری آزمایش های یکپارچه‌سازی و اعتبارسنجی نیز انجام می‌گردد. این آزمایش‌ها خارج از محدودهٔ فرآیند نرم‌افزار هستند و کاملاً توسط مهندس نرم‌افزار صورت نمی‌گیرد. به هرحال، مراحل انجام شده در ضمن طراحی و آزمایش نرم‌افزار، تا حد زیادی احتمال یکپارچه‌شدن نرم‌افزار موفق در یک سیستم بزرگ ارتقاء می‌دهند.

یک مشکل کلاسیک آزمایش سیستم انداختن تقصیر به گردن دیگری است. این حالت زمانی اتفاق می‌افتد که خطایی یافت شود و هر عضو توسعهٔ سیستم، دیگری را مسئول آن مشکل می‌داند. به جای پرداختن به این مسائل بی‌ارزش، مهندس نرم‌افزار باید متوجه مشکلات ارتباطی باشد و

۱- مسیرهای ادارهٔ خطایی طراحی کند که تمام اطلاعات دریافت شده از عناصر دیگر سیستم را آزمایش می‌کنند،

۲- یک‌سری آزمایش‌ها را انجام ده که داده‌های نامناسب یا خطاهای بالقوه دیگر در رابطه نرم‌افزار را شبیه‌سازی کنند،

۳- نتایج آزمایش‌ها برای استفاده به عنوان شاهد ثبت شوند، تا تقصیر به گردن دیگری انداخته نشود،

۴- در برنامه‌ریزی و طراحی آزمایش‌های سیستم شرکت کند تا مطمئن شود که نرم‌افزار به طور مناسبی آزمایش می‌شود.

آزمایش احیاء(Recovery Testing)

بسیاری از سیستم‌های کامپیوتری باید بعد از بروز خطا قابل بازیافت باشند و پردازش را در زمان مشخص شده ادامه دهند. سیستم باید در مقابل اشکال مقاوم باشد، یعنی، خطاهای پردازش نباید باعث شوند کل عملکرد سیستم خاتمه یابد. در موارد دیگر، شکست سیستم باید در بازهٔ زمانی مشخص اصلاح شود در غیر این صورت ضربهٔ اقتصادی شدیدی ایجاد می‌شود. آزمایش بازیافت نوعی آزمایش سیستم است که باعث شکست نرم‌افزار به روش های گوناگون می‌شود و بازبینی می‌کند که آیا بازیافت به طور مناسبی انجام می‌شود.

آزمایش امنیت(Security Testing)

هر سیستم کامپیوتری که اطلاعات حساس را مدیریت می‌کند یا اعمالی انجام می‌دهد که می‌تواند باعث ضرر رساندن (یا فایده رساندن) به افراد شود، هدفی برای نفوذ غیرقانونی یا نامناسب می‌باشد. نفوذ شامل محدوده وسیعی از فعالیت‌ها است:

۱- افراد مهاجمی که سعی در نفوذ به سیستم‌ها را برای تفریح دارند،

۲- کارمندان ناراضی که سعی در نفوذ برای انتقام دارند،

۳- افراد متقلبی که سعی در نفوذ برای رسیدن به اهداف شخصی دارند.

آزمایش امنیت سعی در بازبینی مکانیزم‌های امنیتی ایجاد شده در سیستم دارد تا مطمئن شود سیستم را از نفوذ غیرقانونی محافظت می‌نمایند. Beizer چنین می‌گوید: *" امنیت سیستم باید برای آسیب‌پذیر نبودن از حملهٔ مستقیم آزمایش شود، اما باید برای آسیب‌پذیر نبودن از حملهٔ جانبی یا کناری نیز آزمایش شود".*

آزمایش فشار(Stress Testing)

در ضمن مراحل اولیهٔ آزمایش نرم‌افزار، تکنیک‌های جعبهٔ سفید و جعبهٔ سیاه، عملکردهای معمول و کارایی متداول سیستم را ارزیابی می‌نمایند. آزمایش‌های فشار طراحی می‌شوند تا برنامه‌ها را با موقعیت‌های غیر معمول مواجه نمایند. در نتیجه، آزمایش کننده‌ای که آزمایش فشار را انجام می‌دهد سؤال می‌نماید که: " **قبل از شکست تا چه مدت می‌توان آن را در حال کار نگهداشت؟**".

آزمایش فشار سیستم را به روشی اجرا می‌کند که منابع با کمیت، تکرار، یا حجم غیرمعمول درخواست شوند. برای مثال،

۱- آزمایش های خاصی می‌توانند طراحی شوند تا ده وقفه را در ثانیه تولید کنند، در زمانی که یک یا دو عدد وقفه سرعت متوسط باشد،

۲- سرعت داده‌های ورودی افزایش داده می‌شود تا حدی که مشخص شود چگونه توابع ورودی پاسخ می‌دهند،

۳- نمونه‌های آزمایش که حداکثر حافظه یا منابع دیگر را نیاز دارند اجرا می‌شوند،

۴- نمونه‌های آزمایشی طراحی می گردد که باعث صدمه به سیستم عامل شوند،

۵- نمونه‌های آزمایشی طراحی می‌شوند که باعث دستیابی مداوم به داده‌های ریسک می‌شوند. ضرورتاً، آزمایش کننده سعی در خراب کردن داده‌های سیستم دارد.

آزمایش کارایی(Performance Testing)

برای سیستم‌های بلادرنگ و توکار، نرم‌افزاری که تابع مورد نیاز را فراهم می‌کند اما منطبق بر لوازم کارایی نمی‌باشد، قابل قبول نیست. آزمایش کارایی برای آزمایش کارایی زمان اجرای نرم‌افزار در رابطه با سیستم یکپارچه شده است. آزمایش کارایی در طول تمام مراحل فرآیند آزمایش صورت می‌گیرد. حتی در سطح واحد، کارایی هر پیمانه ممکن است با آزمایش های جعبهٔ سفید بدست آید. به هر حال، تا زمانی که تمام عناصر سیستم به طور کامل یکپارچه نشده باشند، کارایی کامل سیستم قابل دستیابی نیست.

هنر اشکالزدایی(The Art of Debugging)

اشکالزدایی در نتیجهٔ آزمایش موفق انجام می‌شود. یعنی، هنگامی که نمونه های آزمایش خطایی را کشف می‌کند، اشکالزدایی فرآیندی است که باعث حذف خطا می‌شود. اگرچه اشکالزدایی فرآیندی پوششی است، ولی تا حد زیادی هنری است. مهندس نرم‌افزاری که نتایج آزمایش را ارزیابی می‌کند، اغلب با علایم یک مشکل نرم‌افزاری مواجه می‌شود. یعنی، وضوح خارجی خطا و علت داخلی خطا ممکن است رابطه واضحی با یکدیگر نداشته باشند. فرآیند رفتاری مرتبط کنندهٔ یک علامت ظاهر شده با علت آن که اندکی درک شده باشد، اشکالزدایی نامیده می‌شود.

فرآیند اشکال‌دایی

چرا اشکالزدایی این چنین مشکل است؟ شکل ۹ فرایند اشکالزدایی را نشان می دهد. در تمام احتمالات، پاسخ به این سؤال، بیشتر به روانشناسی انسان مربوط می‌شود تا تکنولوژی نرم‌افزار. به هر حال، چند خصوصیت اشکالات، کلیدهایی را فراهم می‌نمایند:

۱- علامت و علت ممکن است از نظر جغرافیایی از یکدیگر فاصله داشته باشند. یعنی، این علامت ممکن است در یک بخش برنامه ظاهر شود، در حالی که علت آن ممکن است در سایتی قرار داشته باشد که بسیار دور است. ساختارهای برنامه‌ای که با اتصال زیاد مرتبط هستند (فصل ۱۳) این وضعیت را بدتر می‌نمایند.

۲- علامت ممکن است وقتی که خطای دیگری اصلاح می‌گردد، به طور موقت ناپدید شود.

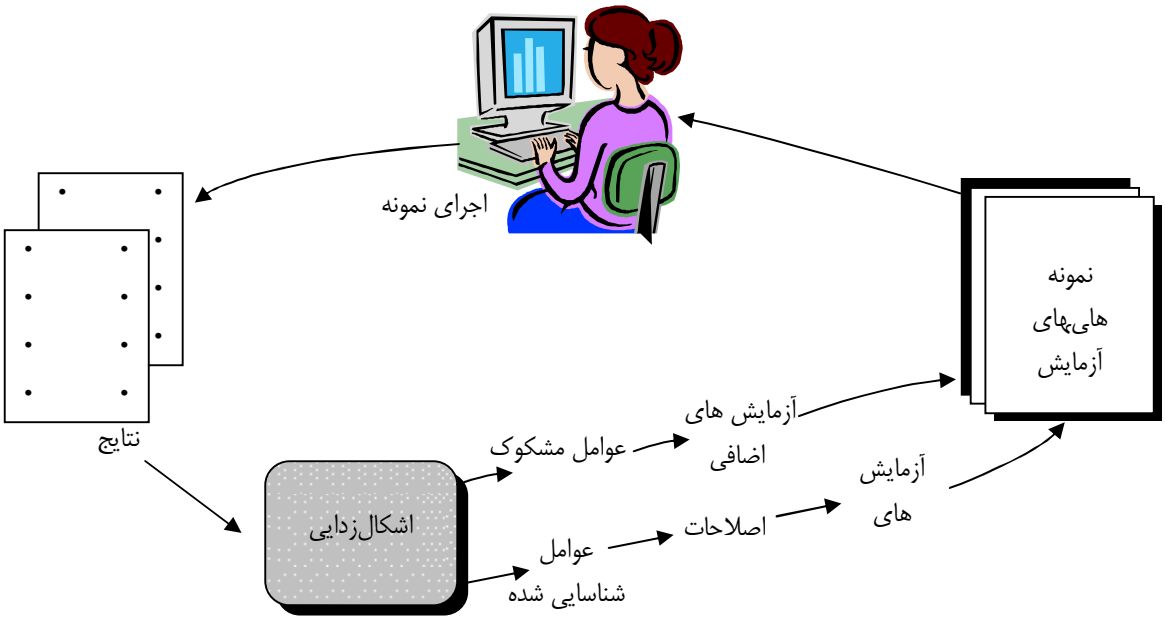
۳- علامت ممکن است توسط هیچ خطایی ایجاد نشده باشد (برای مثال، عدم دقت در نتیجهٔ گردکردن).

۴- علامت ممکن است ناشی از خطای انسانی باشد که به راحتی قابل پیگیری نیست.

۵- علامت ممکن است در نتیجهٔ مشکلات زمانبندی به وجود آید، به جای این که در اثر مشکلات پردازش پدید آمده باشد.

۶- ممکن است ایجاد مجدد شرایط ورودی با دقت، در سیستم‌ها جاسازی شده متداول است زیرا سخت‌افزار و نرم‌افزار کاملاً با یکدیگر متصل شده‌اند.

۷- علامت ممکن است در اثر عللی باشد که در چند task که بر روی پردازنده‌های متفاوت اجرا می‌شوند توزیع شده باشد.



شکل ۹ : ماتریس گراف با در نظر گرفتن ارتباط

شیوه‌های اشکالزدایی

علیرغم شیوه‌ای که به کار گرفته می‌شود، اشکالزدایی یک هدف پوشش دهنده دارد: **یافتن و تصحیح علت خطای نرم‌افزار**. این هدف با ترکیب ارزیابی سیستماتیک، ادارک، و شانس بدست می‌آید. Bradlley شیوهٔ اشکالزدایی را این گونه توصیف می‌کند:

در حالت کلی، سه دسته‌بندی برای روش های اشکالزدایی پیشنهاد می‌شود:

۱- **نیروی مطلق(Brute force)**،
۲- **عقبگرد(Backtracking)**، و
۳- **حذف علت(Cause elimination)**.

روش‌های نیروی مطلق اشکالزدایی احتمالاً متداول‌ترین و کم بازده‌ترین روش برای جدا نمودن علت خطای نرم‌افزار می‌باشد. روش های اشکالزدایی نیروی مطلق زمانی به کار گرفته می‌شوند که همهٔ روش های دیگر با شکست روبرو شده باشند. با فلسفهٔ " یافتن خطا توسط خود کامپیوتر "، محتویات حافظه بر روی صفحه نمایش داده می‌شود. پیگیری‌های زمان اجرا انجام می‌شوند، و احکام write در برنامه قرار داده می‌گیرند. انتظار می‌رود که جایی در اطلاعات تولید شده، کلیدی یافت شود که بتواند باعث هدایت به سمت خطا گردد. اگرچه حجم زیاد اطلاعات تولید شده ممکن است تا حدی به موفقیت منتهی شود، در اکثر موارد باعث اتلاف فعالیت و زمان می‌شود.

عقبگرد روشی نسبتاً متداول است که می‌تواند با موفقیت در برنامه‌های کوچک به کار گرفته شود. با شروع از محلی که علامت در آنجا ظاهر شده، برنامه مبدأ به سمت عقب (به صورت دستی) دنبال می‌شود تا زمانی که محل علت بروز خطا یافت شود. بدبختانه، با افزایش تعداد خطوط کد، تعداد مسیرهای عقبگرد بسیار زیاد خواهد بود.

روش سوم اشکالزدایی، **حذف علت**، با بسط یا حذف انجام می‌شود و مفهوم تقسیم‌بندی دودویی را به همراه دارد. داده‌های مرتب با خطا سازماندهی می‌شوند تا علل بالقوه را جدا نمایند. در یکی از حالات، لیستی از تمام علت‌های ممکن توسعه داده می‌شود و آزمایش‌های مربوطه انجام می‌شوند تا هر یک را حذف کنند. اگر آزمایش‌های اولیه نشان دهند که یک علت خاص، مربوط به علامت ظاهر شده است، داده‌ها پالایش می‌شوند تا خطا حذف شود.

تست‌های فصل ۲۰ : آزمون نرم‌افزار

۱. آخرین مرحله آزمون سطح بالا در چه حیطه‌ای قرار می‌گیرد؟

الف) کدنویسی ب) طراحی ج) خواسته‌ها د) مهندسی سیستم

۲. در حین آزمون واحدها کدام آزمون اهمیت ویژه‌ای دارد؟

الف) آزمون انتخابی(مسیرهای اجرایی کار) ب) آزمون جامعیت ج) آزمون رگرسیون د) آزمون دود

۳. کدام آزمون کوشش می‌کند تا وارسی کند که راهکارهای محافظ تعبیه شده در داخل سیستم واقعاً آن را از نفوذ نامناسب حفظ می‌کند؟

الف) آزمون امنیت ب) آزمون بازیابی ج) آزمون فشار د) آزمون کارایی

۴. کدام آزمون سیستم را به شیوه‌ای اجرا می‌کند که منابع را مقادیر غیر عادی، فراوانی غیر عادی یا حجم غیر عادی طلب کند؟

الف) آزمون کارایی ب) آزمون فشار ج) آزمون امنیت د) آزمون بازیابی

۵. در فرآیند اشکال‌زدایی کدام عمل مورد توجه قرار نمی‌گیرد؟

الف) آزمون‌های اضافی ب) تصحیحات ج) آزمون امنیت د) آزمون رگرسیون

۶. کم بازده‌ترین روش برای از بین بردن علت یک خطای نرم افزاری کدام است؟

الف) روش‌های نیروی مطلق ب) عقب‌گرد ج) حذف علت د) روش ردیاب

۷. بیشترین کار فنی در فرآیند نرم‌افزاری کدام است؟

الف) آزمون نرم‌افزار ب) طرح‌ریزی ج) اجرا د) کنترل

۸. کدام ویژگی باعث ایجاد طرح نرم‌افزار آزمون پذیر می‌شود؟

الف) قابلیت کار ب) قابلیت مشاهده ج) کنترل‌پذیری د) هر سه مورد

۹. آخرین وظیفه در مرحله آزمون واحدها چیست؟

الف) آزمون جامعیت ب) آزمون مرزی ج) آزمون رگرسیون د) آزمون سیستم

۱۰. اعتبارسنجی نرم‌افزار از طریق کدام آزمون انجام می‌پذیرد؟

الف) آزمون آلفا ب) آزمون بتا ج) آزمون جعبه سفید د) آزمون جعبه سیاه

۱۱. کدام آزمون نرم‌افزار را به طریق گوناگون وادار به شکست می‌کند و سپس در مورد اجرای مناسب بازیابی تحقیق می‌کند؟

الف) بازیابی ب) امنیت ج) کارایی د) فشار

۱۲. هدف اصلی اشکال‌زدایی چیست؟

الف) یافتن منبع مشکل از طریق استقراء

ب) یافتن منبع مشکل از طریق شانس قابل دستیابی

ج) یافتن و تصحیح علت یک خطای نرم‌افزاری توسط ترکیبی از سه روش نیروی مطلق، عقبگرد و حذف علت.

د) یافتن و تصحیح علت یک خطای نرم‌افزاری توسط ترکیبی از ارزیابی سیستماتیک و نبوغ و شانس قابل دستیابی است.

منابع

1- Pressman, R., Software Engineerig : A Practitioner's Approach, 5th editition, Mc Graw-Hill, 2005.
2- Bruegge Bernd, Dutoit, H.,Allen; Object Oriented Software Engineering, Prentice Hall, 2000.
3- Rumbaugh, J., Blaha Micheal, Premerlani William, Lorensen William; Object Oriented Moddeling and Design; Prentice Hall, 1991.
4- Kendall & Kendall; System Analysis and Design; 4th edition, Prentice Hall, 1999.
5- Sommerville, Ian; Software Engineering; 5th Edition, Addison-Wesley, 2000.
6- Parrington, N., Marc Roper; Understanding Software Testing; John Wiley & Sons, 1999.
7- Holmes, Jim; Object Oriented Computer Construction, Prentic Hall, 1995.

۸- محمد مهدی سالخورده حقیقی، مهندسی نرم‌افزار، ترجمه ویرایش ۵ مهندسی نرم‌افزار پرسمن، ۱۳۸۲

۹- تحلیل و طراحی سیستم‌ها در مهندسی نرم‌افزار، دکتر پارسا، ۱۳۷۷

۱۰- عین الله جعفر نژاد قمی، مهندسی نرم‌افزار، ترجمه ویرایش ۵ مهندسی نرم‌افزار پرسمن، ۱۳۸۱

۱۱- هاشمی طباء، مهندسی نرم‌افزار، ترجمه ویرایش ۵ مهندسی نرم‌افزار پرسمن، ۱۳۸۲

۱۲- اسلام ناظمی، مسعود زکی پور، امیرفرخ قنبرپور، ترجمه و تنظیم از ویراست های ۴ تا ۶ پرسمن، موسسه پارسه تابستان ۱۳۸۴

ضمیمه

آزمایش سیستم‌های بلادرنگ

ماهیت وابسته به زمان و غیرهمزمان بسیاری از کاربردهای بلادرنگ، عنصری جدید و احتمالاً مشکل را به نام زمان به آزمایش می‌افزاید. طراح نمونه های آزمایش باید نمونه های‌های آزمایش جعبهٔ سفید و جعبهٔ سیاه را همراه، با ادارهٔ واقعه (یعنی پردازش وقفه)، زمانبندی داده‌ها، و موازی بودن taskهای اداره کنندهٔ داده‌ها در نظر داشته باشد. در بسیاری از موارد، داده‌های آزمایشی زمانی فراهم می‌شوند که سیستم بلادرنگ در یک حالت خاص قرار دارد، ممکن است باعث بروز خطا شوند.

برای مثال، نرم‌افزار بلادرنگی که دستگاه کپی جدیدی را کنترل می‌کند، وقفه‌های اوپراتوری را بدون خطا می‌پذیرد (یعنی اوپراتور ماشین کلیدی کنترلی مانند RESET یا DARKEN را می‌زند)، وقتی ماشین در حال گرفتن کپی‌ها است (درحالت " کپی " قرار دارد). همین وقفه‌های اوپراتور، اگر در حالت " جمع شدن کاغذ " وارد شوند، باعث نمایش کد شناسایی می‌شوند تا محل جمع شدن کاغذ را که باید برطرف شود مشخص نمایند (خطا).

علاوه بر آن، رابط نزدیک بین نرم‌افزار بلادرنگ و محیط سخت‌افزاری نیز باعث می‌شود آزمایش با مشکل روبرو شود. آزمایش های نرم‌افزار باید تأثیر خطاهای سخت‌افزار را بر پردازش نرم‌افزار در نظر بگیرند. شبیه‌سازی چنین اشکالاتی ممکن است بسیار مشکل باشد.

روش های طراحی نمونه های‌های طراحی برای سیستم‌های بلادرنگ هنوز در حال تکامل است. به هر حال، یک استراتژی کلی چهار مرحله‌ای پیشنهاد می‌شود:

آزمایش task: اولین مرحله در آزمایش نرم‌افزار بلادرنگ، آزمایش هر task به طور مستقل می‌باشد. یعنی آزمایش های جعبه سفید و جعبه سیاه برای هر task طراحی و اجرا شوند. هر task به طور مستقل در ضمن این آزمایشات اجرا می‌شود. آزمایش task خطاهایی را در منطق و عملکرد آشکار می‌کند، اما خطاهای زمانبندی و رفتاری آشکار نمی‌شوند.

آزمایش رفتاری: با استفاده از مدلهای سیستم‌هایی که با نمونه های‌های CASE ایجاد شده‌اند، امکان شبیه‌سازی رفتار سیستم بلادرنگ و آزمایش رفتار آن در نتیجهٔ وقایع خارجی، وجود دارد. این فعالیت‌های تحلیل، مبنایی را فراهم می‌کنند برای طراحی نمونه های‌های آزمایشی که در زمان ایجاد نرم‌افزار بلادرنگ هدایت می‌شوند. با استفاده از تکنیکی مشابه تقسیم‌بندی مساوی، وقایع (برای مثال، وقفه‌ها، سیگنالهای کنترل) برای آزمایش دسته‌بندی می‌شوند. برای مثال، وقایعی برای دستگاه فتوکپی عبارتند از: وقفه‌های کاربر (برای مثال، کم شدن پودر قرمز)، و مودهای شکست (برای مثال، سربار رولر). هر یک از این وقایع به طور مجزا آزمایش می‌شوند و رفتار سیستم اجرایی آزمایش می‌گردد تا خطاهایی در نتیجهٔ پردازش مربوط به این وقایع، آشکار شوند. رفتار مدل سیستم (که در ضمن فعالیت تحلیل توسعه داده شده) و نرم‌افزار اجرایی برای مسائل کارایی مقایسه می‌شوند. رفتار سیستم آزمایش می‌شود تا خطاهای رفتاری آشکار گردند.

آزمایش بین taskها. پس از جدا شدن خطاهای هر یک از task ها و رفتار سیستم، آزمایش به سمت خطاهای زمانی هدایت می‌شود. task های غیرهمزمان که با یکدیگر ارتباط برقرار می‌کنند، با سرعت انتقال داده‌ها و بار پردازش متفاوت آزمایش می‌شوند تا مشخص کنند آیا خطاهای همزمانی ارتباط بین task ها اتفاق می‌افتند یا خیر. علاوه بر آن، task هایی که با استفاده از صف پیغام یا حافظهٔ داده‌ها ارتباط برقرار می‌نمایند آزمایش می‌گردند تا خطاهای مربوط به اندازهٔ این ناحیه‌های حافظه آشکار شوند.

آزمایش سیستم. نرم‌افزار و سخت‌افزار مجتمع می‌شوند و محدودهٔ کاملی از آزمایش های سیستم هدایت می‌شوند تا خطاهای ارتباط سخت‌افزار– نرم‌افزار آشکار شود. اکثر سیستم‌های بلادرنگ، وقفه‌ها را پردازش می‌کنند. بنابراین، آزمایش ادراهٔ این وقایع بولی ضروری است. با استفاده از نمودار تغییر حالت و مشخصهٔ کنترل (فصل ۱۲)، آزمایش

کننده، لیستی از تمام وقفه‌های ممکن و پردازش‌هایی را که در نتیجهٔ آن وقفه‌ها انجام مـی‌شـوند و توسـعه مـی‌دهـد. سپس آزمایش هایی طراحی می‌شوند تا به خصوصیات سیستم که در زیر ارائه شده برسند:

- آیا اولویت‌های وقفه‌ها به طور منظم تخصیص داده شده و به طور منظم اداره می‌شوند؟
- آیا پردازش برای هر وقفه درست اداره می‌شود؟
- آیا کارایی (برای مثال، زمان پردازش) برای هر رویه اداره کنندهٔ وقفه با نیازها مطابقت دارد؟
- آیا حجم زیاد وقفه‌هایی که در زمانهای بحرانی دریافـت مـی‌شـوند مشـکلی را در عملکـرد و کـارایی ایجـاد می‌کنند؟

علاوه بر آن، ناحیه‌های داده‌های سراسری که بـرای انتقـال اطلاعـات بـه عنـوان بخشـی از پـردازش وقفـه اسـتفاده می‌شوند، باید آزمایش شوند تا پتانسیلی را برای تولید اثرات جانبی مشخص کنند.

نکات استراتژیک

در ادامهٔ این فصل، یک استراتژی سیستماتیک برای آزمایش نرم‌افزار ارائه مـی‌گـردد. امـا حتـی بهتـرین اسـتراتژی بـا شکست روبرو می‌شود اگر یک سرسی موارد عمده مورد توجه قرار نگیرند. TomGillb بحث می‌کند که مـوارد زیـر باید مورد توجه قرار گیرند اگر استراتژی آزمایش موفق نرم‌افزار قرار است پیاده‌سازی شود:

- مشخص نمودن نیازهای محصول به روش کمی، مدت طولانی قبل از شروع آزمایش.
- بیان صریح اهداف آزمایش.
- شناسایی خصوصیات کاربران نرم‌افزار و توسعهٔ پروفایلی برای هر دسته‌بندی از کاربران.
- توسعهٔ طرح آزمایشی که بر " دورهٔ سریع آزمایش" تأکید دارد.
- نرم‌افزاری تنومند ایجاد شود که برای آزمایش خودش طراحی شده باشد.
- از مرورهای تکنیکی رسمی مؤثر، به عنوان فیلتر، قبل از آزمایش استفاده شود.
- مرورهای تکنیکی رسمی به گونه‌ای هدایت شوند که به استراتژی آزمایش و خود نمونـه هایی‌هـای آزمـایش دست یابند.
- روشی پیوسته برای ارتقاء فرآیند آزمایش توسعه داده شود.

رویه‌های آزمایش واحد

آزمایش واحد به طور معمول با مرحلهٔ کدنویسی در نظر گرفته می‌شود. پس از توسعهٔ کد مبدأ، مـرور آن، و بـازبینی آن برای تطابق با طراحی در سطح مؤلفه، طراحی نمونه های آزمـایش واحـد شـروع مـی‌شـود. مـرور اطلاعـات طراحـی، راهنمایی‌هایی را برای ایجاد نمونه های آزمایشی فراهم می‌کند که احتمالاً خطاها را در هـر یـک از دسـته‌بنـدی‌هـای بحث شده آشکار می‌نمایند هر نمونه های آزمایش باید با مجموعه‌ای از نتایج مورد انتظار همراه شود.

چون یک مؤلفه، یک برنامهٔ مستقل نمی‌باشد، نرم‌افزار اداره کننده و stub باید بـرای هـر آزمـایش واحـد توسـعه داده شوند. این محیط آزمایش واحد در شکل نشان داده شده است. در اکثر کاربردها، اداره کننده چیزی بیش از یک "برنامـه اصلی" نیست که داده‌های نمونه های طراحی را دریافت می‌کند، این داده‌ها را به مؤلفهٔ مـورد نظـر (کـه بایـد آزمـایش شود) ارسال می‌کند، و نتایج بدست آمده را چاپ می‌کند stub ها برای جایگزین شدن با پیمانه‌هایی ایجـاد مـی‌شـوند که توسط مؤلفهٔ در حال آزمایش فراخوانی می‌شوند. یک stub یا " زیر برنامـه سـاختگی " بـا اسـتفاده از رابـط پیمانـهٔ فراخوانی شده، حداقل دستکاری بر روی داده‌ها را انجام می‌دهد، نتیجهٔ بازبینی ورودی را چاپ می‌کند، و کنتـرل را بـه پیمانهٔ در حال آزمایش باز می‌گرداند.