

فصل چهارم – تحلیل و طراحی شی گرا

همان‌طور که در فصل‌های پیشین مطرح شد، دو روش برای ساخت مدل تحلیل وجود دارد:

۱. **مدل ساخت‌یافته (Procedural):** در این روش تحلیل‌گر به دنبال فرایندها است. دیدگاه تحلیل‌گر این است که بدنبال فرایندهای موجود در سیستم بگردد و پس از یافتن آن‌ها به کمک نمودارهای (DFD و ...) آن‌ها را مدل‌سازی کند. نتیجه‌ی این روش مجموعه‌ای از ماژول‌ها و توابع است که ساختار برنامه را تشکیل می‌دهند.

۲. **مدل شی گرا (Object Oriented):** در این روش دیدگاه تحلیل‌گر این است که باید به دنبال اشیای موجود در سیستم بگردد و اشیاء و ویژگی‌ها و وظایف آن‌ها را مدل‌سازی کند. نتیجه‌ی این روش تعریف ساختار برنامه است که به صورت مجموعه‌ای از کلاس‌ها و روابط بین آن‌هاست.

۱. متدهای مختلف شی‌گرایی: Code-yourdon

۲. متد Fusion

۳. متد Jacobson

۴. متد OMT (Object Modeling Technique)

۵. متد Booch

۶. متد UML

* ما در این دوره تنها با متد UML آشنا خواهیم شد.

* هر مدل تحلیل شامل مجموعه‌ای از نمودارهاست.

مفاهیم اولیه شی‌گرایی:

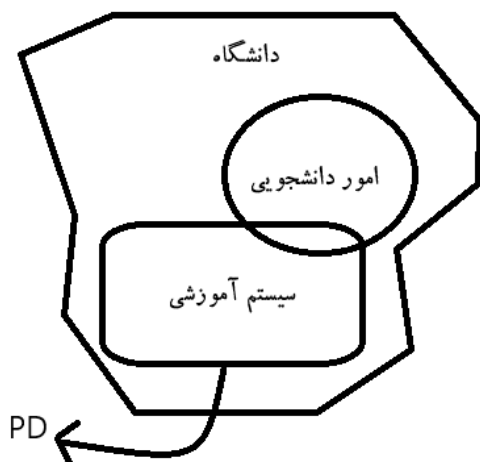
۱. **شی:** هر جزء مهم سیستم که نشان‌دهنده‌ی بخشی از توانایی آن است و می‌تواند اطلاعاتی درمورد آن را نگهداری کند. هر شی بصورت یک Abstract از صفات و وظایف نشان داده می‌شود.

۲. **کلاس:** توصیف مجموعه‌ای از اشیاء با صفات و وظایف مشترک. کلاس وجود خارجی ندارد و تنها یک توصیف است. اما شی نمود بیرونی و واقعی دارد.

مثال: در کلاس دانشجو مجموعه‌ی دانشجویان توصیف می‌شوند و صفات آن‌ها تعریف می‌شود (مثلاً: تمام دانشجویان دارای یک کد دانشجویی هستند و یا وظیفه دارند دروس خود را پاس کنند). اما یک شی دانشجو به یک دانشجو بخصوص اشاره دارد

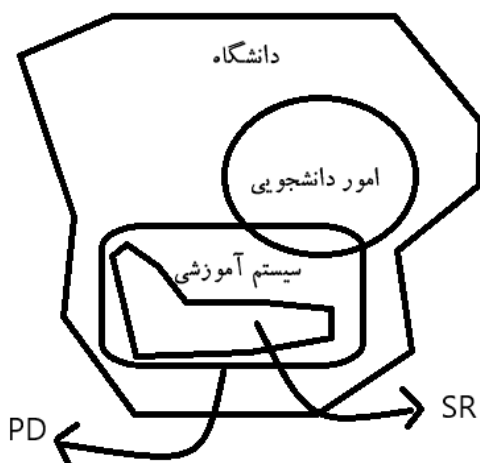
(مثلا: دانشجو با کد دانشجویی ۱۱۱۲ توانسته ۲۰ واحد را پاس کند). در نتیجه شی وجود و نمود بیرونی دارد اما کلاس يك مفهوم انتزاعی است که وظیفه وصف اشیا بصورت عمومی را دارد.

۳. **Problem Domain (PD):** بخشی از يك سیستم است که برای اجرای پروژه مورد بررسی قرار می گیرد. مثال برای سیستم آموزش:



PD محدوده‌ای وسیع تر از خود سیستم است که باید بررسی شود. مثلا سیستم آموزش که باید با سیستم نظام وظیفه که يك سیستم خارجی است نیز ارتباط برقرار کند. یا مثلا همانطور که در تصویر بالا می بینیم سیستم آموزشی با دیگر امور، مانند امور دانشجویی تلاقی دارد و نقطه مشترکی در این میان دیده می شود.

۴. **System Responsibilities (SR):** بخشی از PD که مربوط به پروژه است را SR می نامیم. مثال مجدد برای سیستم آموزش:



SR همان بخشی است که قرار است بصورت مدل ما پیاده سازی شود.

۱. تحلیل شی گرا (Object Oriented Analysis (OOA) خروجی: مدل تحلیل برخی دیگر از مفاهیم:

۲. طراحی شی گرا (Object Oriented Design (OOD) خروجی: مدل طراحی

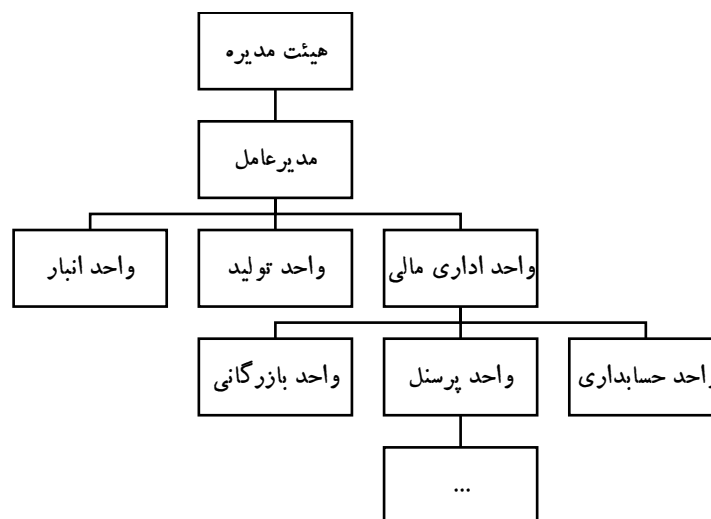
۳. برنامه نویسی شی گرا (Object Oriented Programming (OOP) خروجی: پیاده سازی

طبقه بندی انواع اشیاء:

۱. **Devices:** هر دستگاه، قطعه، جزئی از دستگاه، يك سیستم داخلی و ابزاری که در سیستم مهم باشد. مانند سیستم های اداری (پرینتر، دستگاه فکس و ...) در يك سیستم آموزشی.

۲. **Other Systems:** سیستم های دیگر نیز می توانند در مدل طراحی ما به عنوان اشیاء طبقه بندی شوند. هر سیستم یا سازمان خارجی که با سیستم مورد نظر در ارتباط باشند جزء این دسته قرار دارند. سیستم حسابداری، امور دانشجویی، نظام وظیفه و ... سیستم های دیگر در سیستم دانشجویی هستند که به عنوان شی تعریف می شوند.

۳. **Organizational Units:** هر مجموعه یا سیستمی که از بخش های مختلف تشکیل شده اند را واحدهای سازمانی می نامیم و می توانیم آن ها را به عنوان يك شی در سیستم در نظر بگیریم. هر يك از بخش های يك سازمان که در چارت سازمانی آن سازمان تعریف شده و به نوعی در سیستم در حال تحلیل مهم هستند عملاً يك شی در نظر گرفته می شوند. مانند واحد فارغ التحصیلان که زیرمجموعه ای از واحد آموزش است. يك مثال دیگر:

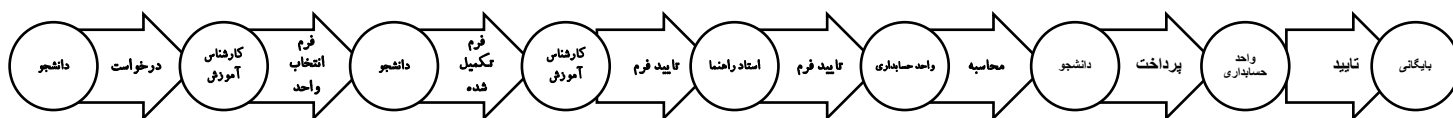


۴. **Roles:** در هر واحد سازمانی افراد دارای جایگاهی هستند که به آن جایگاه پست سازمانی و یا نقش گفته می شود. این ها نقش هایی هستند که برای آن واحد تعریف شده است. جایگاه افراد در هر واحد سازمانی، پست سازمانی یا نقش یا جایگاه افراد نامیده می شود. مثال برای واحد آموزش: رئیس آموزش (۱) - کارشناس آموزش (۱) و ...

هر پست سازمانی (نقش) می تواند يك شی دارای صفات و وظایف باشد.

۵. **Operational Procedures**: فرایندهای عملیاتی، بخشی از فرایندها یا عملیاتی هستند که در سیستم انجام می‌شود و می‌توانند بعنوان يك شی در نظر گرفته شوند.

مثال: فرایند انتخاب واحد در سیستم آموزشی:



فرض شود بخشی از فرایند بالا بعنوان يك شی در نظر گرفته شود. مثلاً شی آماده سازی فرم و یا شی تایید فرم.

۶. **Events**: وقایع اشاره دارد به اتفاقات مهمی که در سیستم می‌تواند اتفاق بیفتد و لازم است درمورد آن اتفاق اطلاعاتی ذخیره شود و یا کارهایی در ارتباط با آن اتفاق باید انجام شود. این اتفاقات را می‌توان به عنوان شی شناسایی کرد. مانند خطاها در سیستم. مثال: رخداد خطای سیستم، قطع ارتباط با شبکه، مشروط شدن يك دانشجو و ... می‌توانند به عنوان يك شی در نظر گرفته شوند.

۷. **Sites**: موقعیت مکانی مهم که لازم باشد درمورد آن مکان اطلاعاتی ذخیره شود و همچنین وظایفی برای آن تعریف شود می‌تواند به عنوان يك شی شناخته شود. مثال: چهارراه و میدان در سیستم‌های نرم‌افزارهای ترافیکی مانند نشان.

تحلیل شی گرا به وسیله ی متد UML

* UML == Unified Modeling Language

* این متد از ترکیب سه متد OMT، Booch و Jacobson حاصل شده است.

دیاگرام های UML به دو بخش تقسیم می شوند:

۱. دیاگرام های ساختاری:

– Class Diagram: مدل سازی ساختار کلاس ها و روابط میان آنها.

– Object Diagram

– Deployment Diagram: مدل سازی ساختار کلی اجزای سخت افزاری سیستم و محل استقرار آنها.

– Package Diagram

– Component Diagram

– Composite Structure Diagram

۲. دیاگرام های رفتاری:

– Use Case Diagram: مدل سازی تعامل های کاربر و سیستم.

– Activity Diagram: مدل سازی مراحل انجام فرایند.

– State Diagram: مدل سازی رفتار پویای سیستم.

– Interaction Diagram

– Sequence Diagram: مدل سازی تعامل های میان اشیای سیستم.

– Communication Diagram

– Timing Diagram

– Intractive Overview Diagram

دیاگرام Use Case:


این نمودار یک نمودار رفتاری است.

هدف: مدل سازی تعامل های بین کاربر و سیستم.

اجزای تشکیل دهنده:

۱. Use Cases - < 

به هر کاری که سیستم برای کاربر انجام می دهد یک use case گفته می شود. نحوه نام گذاری هر use case باید براساس فعل باشد زیرا نشان دهنده یک کار برای انجام شدن است. مانند (ثبت نام کردن، تغییر رمز دادن، رزرو کردن و ...)

۲. Actor - < 

کاربر سیستم که می تواند انسان، سخت افزار یا یک نرم افزار خارجی و ... باشد که با سیستم در ارتباط است و سیستم باید برای آن کاری انجام دهد. نحوه نام گذاری هر اکتور باید براساس نام باشد. مانند (دانشجو، مشتری و ...)

۳. نشان دهنده ارتباط بین use case و اکتور یک خط ساده است که با عنوان رابطه ی Association شناخته می شود.

گاهی ممکن است که اکتورها بعنوان ورودی و نیز خروجی یک use case باشند



راه های شناسایی اکتورها:

۱. تعیین مرز سیستم.

۲. کاربران سیستم همیشه اکتور هستند.

۳. هر سیستم که خارج از مرز سیستم باشد و با سیستم در ارتباط باشد یک اکتور است (سخت افزاری یا نرم افزاری).

۴. اسامی داخل سناریو می توانند اکتور باشند.

راه های شناسایی یوز کیس ها:

۱. تعیین مرز سیستم.

۲. تمام کارهایی که داخل سیستم انجام می شوند یوز کیس هستند.

۳. سیستم هایی که داخل سیستم ما قرار دارند یا به عبارتی زیرسیستم های ما نیز بعنوان یوز کیس شناسایی می شوند.

۴. در سناریوی سیستم فعل ها می توانند یوز کیس باشند (در صورتی که داخل سیستم باشند).

مثال در يك سناریو: مشتریان می توانند با ورود به سیستم لیست محصولات را مشاهده و از بین آن ها خرید کنند.

افعال (یوز کیس ها): مشاهده لیست محصولات – خرید محصول

اسم ها (اکتور): مشتری

روابط در دیاگرام Use Case:

۱. Include: وقتی استفاده می شود که use case شامل يك use case دیگر شود. مثال: در يك ساخت خرید آنلاین، برای ویرایش سبد خرید باید از پیش محصولی را به سبد خرید اضافه کرده باشیم. و یا مثلاً برای مشاهده نمرات درس در سامانه باید از پیش لاگین کرده باشیم. پس برای کامل شدن بعضی از use case ها باید use case دیگری نیز وجود داشته باشد.

۲. Extend: زمانی که بخواهیم يك use case را در شرایط خاص تغییر بدهیم یا عملکرد اضافی برای آن در نظر بگیریم، از این رابطه استفاده می کنیم. مثال: در يك سیستم رزرو هتل، ممکن است يك use case به نام "رزرو اتاق" داشته باشیم که در شرایط خاص (مانند درخواست خدمات لوکس) گسترش یابد. در این حالت، خدمات لوکس به عنوان يك use case اختیاری تعریف می شود که extend شده است.

۳. Generalization یا Specification: به معنای وراثت است. يك use case کلی و عمومی تعریف می شود و سپس حالات مختلف و تخصصی آن در use case های فرزند مشخص می شوند. مثال: اگر در يك سیستم دانشگاهی يك use case کلی به نام "مدیریت اطلاعات کاربران" داشته باشیم، این use case می تواند شامل موارد خاص تر مانند "مدیریت اطلاعات دانشجویان" و "مدیریت اطلاعات اساتید" باشد.

۴. Association: این رابطه نشان دهنده تعامل بین يك بازیگر (Actor) و يك Use Case است. این ارتباط، معمولاً با يك خط ساده بین بازیگر و Use Case نشان داده می شود و بیانگر این است که بازیگر می تواند يك Use Case خاص را اجرا کند یا با آن در تعامل باشد. مثال: در يك سیستم بانکداری، "مشتری" (Actor) ممکن است با Use Case "مشاهده موجودی" ارتباط داشته باشد. این ارتباط فقط تعامل مستقیم مشتری با این عملیات را نشان می دهد.

* در نمودار Use Case، رابطه Extend اختیاری است، به این معنا که بدون آن نیز فرایند اصلی قابل انجام است. در مقابل رابطه Include بخشی از فرایند اصلی است و رسم آن اجباری است، زیرا بدون آن اجرای کامل فرایند ممکن نیست.

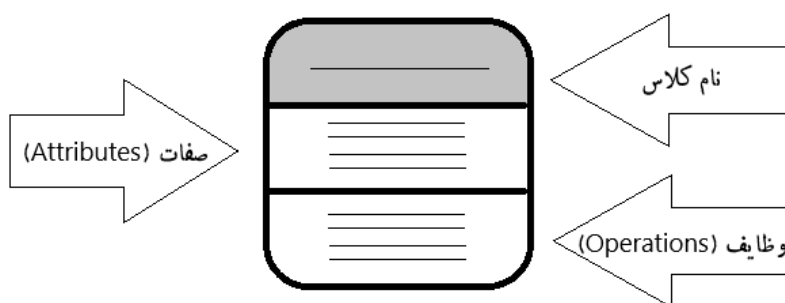
دیاگرام Class:

این نمودار یک نمودار ساختاری است.

هدف: مدل سازی ساختار کلاس ها و روابط آنها.

اجزای تشکیل دهنده:

۱. Class:

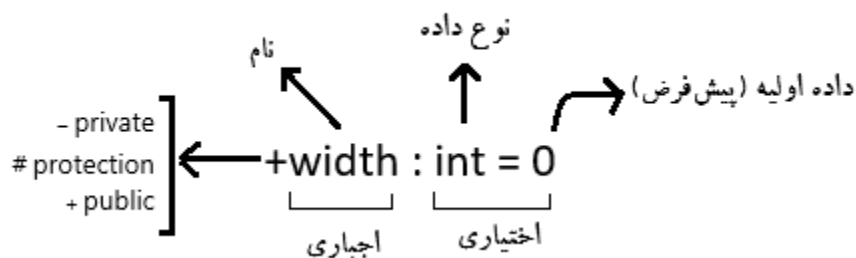


راه های شناسایی صفات یک کلاس:

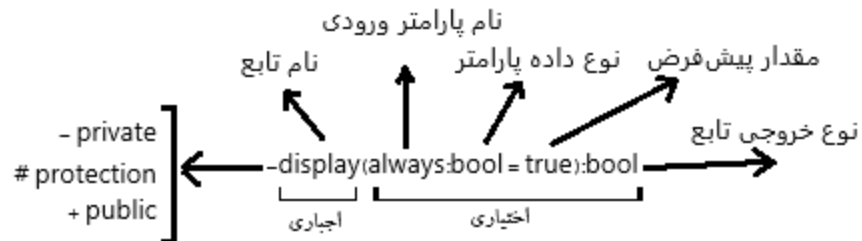
۱. اطلاعاتی که برای آن کلاس باید ذخیره شوند (مثال برای کلاس دانشجو: تعداد واحد، معدل و ...).

۲. داده های توصیف کننده ی اشیای یک کلاس (مثال برای کلاس دانشجو: نام، کد دانشجویی و ...).

نحوه ی نمایش صفات:



نحوه ی نمایش وظایف (توابع):

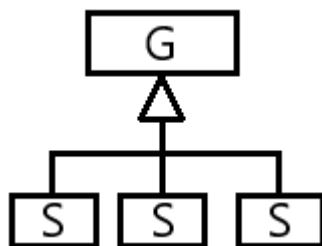


۲. روابط بین کلاس‌ها:

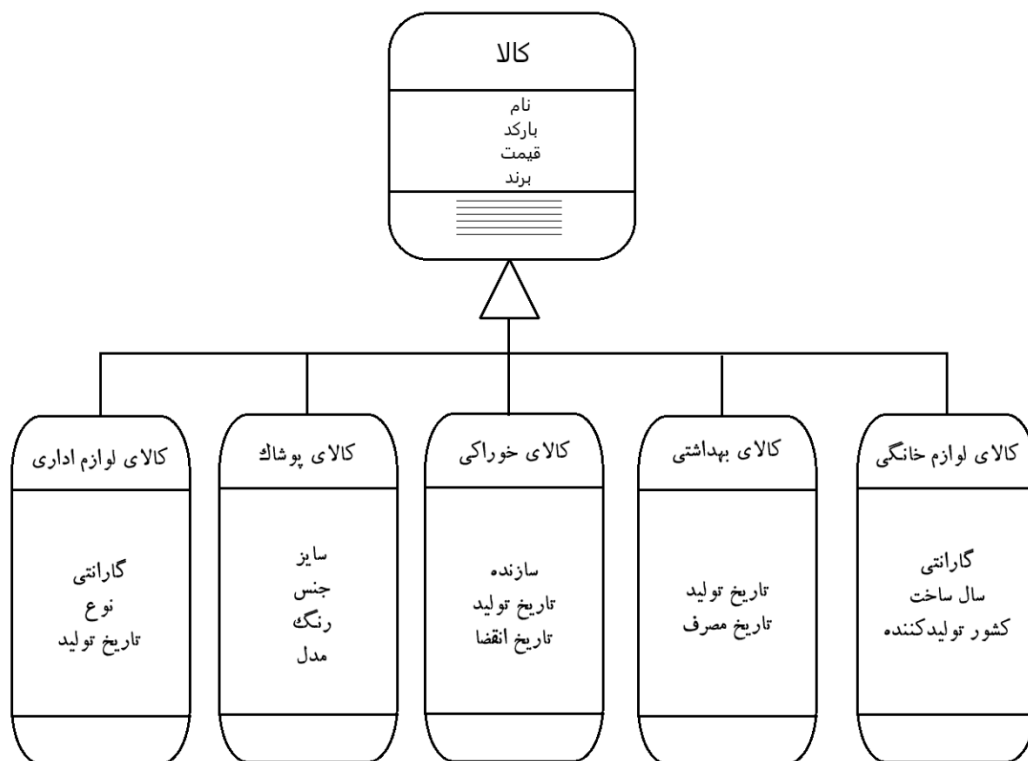
۱.۲. رابطه‌ی عام به خاص (Generalization): این رابطه به عنوان رابطه‌ی عام به خاص یا رابطه‌ی وراثت شناخته می‌شود. در این حالت یک کلاس عام داریم و تعدادی کلاس خاص. کلاس عام کارهای کلی را انجام می‌دهد اما کلاس‌های خاص افزون بر کارهای کلاس عام، کارهای خاص دیگری نیز انجام می‌دهند. هر کلاس خاص نوع خاصی از کلاس عام هستند. تمام صفات و متدهای کلاس عام به تمام کلاس‌های خاص به ارث می‌رسد. کلاس‌های خاص می‌توانند صفات و متدهای خاص خود را نیز داشته باشند.

نحوه‌ی شناسایی این رابطه: اگر بشود از اصطلاح "نوع خاصی از" استفاده کرد، این نوع رابطه برقرار است. مثال: لپ‌تاپ نوع خاصی از کامپیوتر است. درس عمومی نوع خاصی از درس است.

شکل این رابطه:



مثال برای شناسایی کلاس عام:

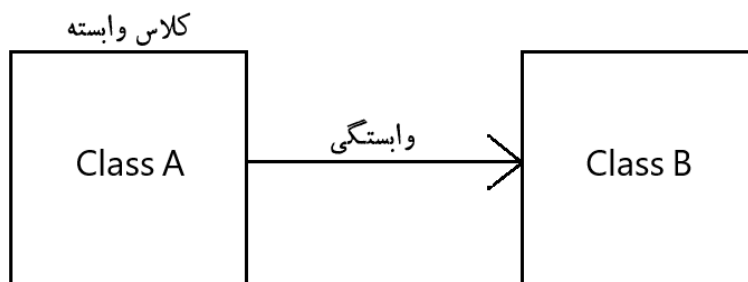


* صفاتی که از کلاس جنرال (کالا) به ارث می‌برند، دیگر در کلاس‌های خاص نوشته نمی‌شود. امکان دارد که این رویه تا چند سطح پیش برود. برای مثال: کالای لوازم خانگی خودش نیز یک کلاس جنرال باشد که دو کلاس خاص (برقی و غیر برقی) صفات او را به ارث می‌برند.

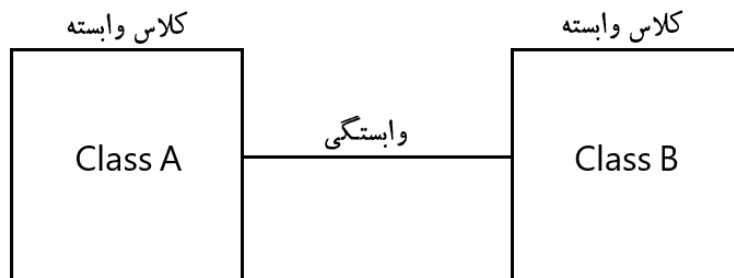
۲.۲. **رابطه‌ی وابستگی (Association):** هر وابستگی که بین کلاس‌ها باشد و ربطی به ارث‌بری نداشته باشد جزء این رابطه است. شکل رسم با یک خط ساده است. مثال: وقتی یک کلاس بخواهد از صفات و متدهای یک کلاس دیگر استفاده کند از این رابطه استفاده می‌شود.

انواع:

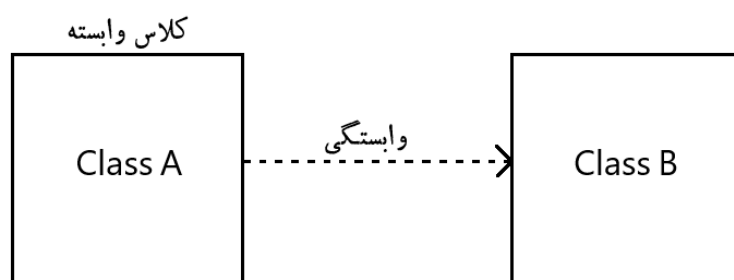
رابطه یک‌طرفه:



رابطه دوطرفه:



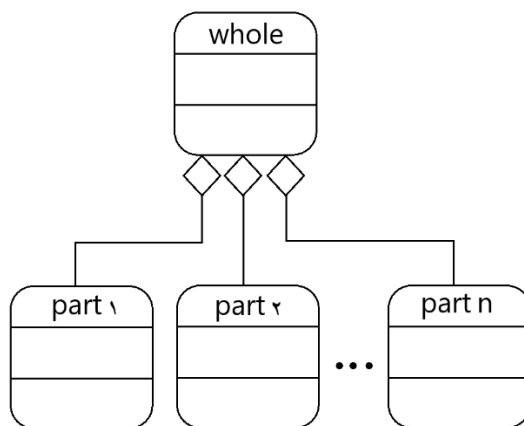
رابطه موقت وابستگی:



* در این مدل وابستگی بعد مدتی از بین می‌رود.

* رابطه‌های وابستگی می‌توانند یک به یک، یک به چند و چند به چند باشند.

۳.۲. رابطه‌ی کل به جزء (Aggregation): رابطه‌ی کل به جزء (یا Whole-Part) نوعی از رابطه‌ی وابستگی (Association) است. در این رابطه هر کلاس از نوع کل تشکیل شده از کلاس‌های جزء. یعنی کلاس‌های جزء، اجزای تشکیل‌دهنده‌ی کلاس کل هستند.



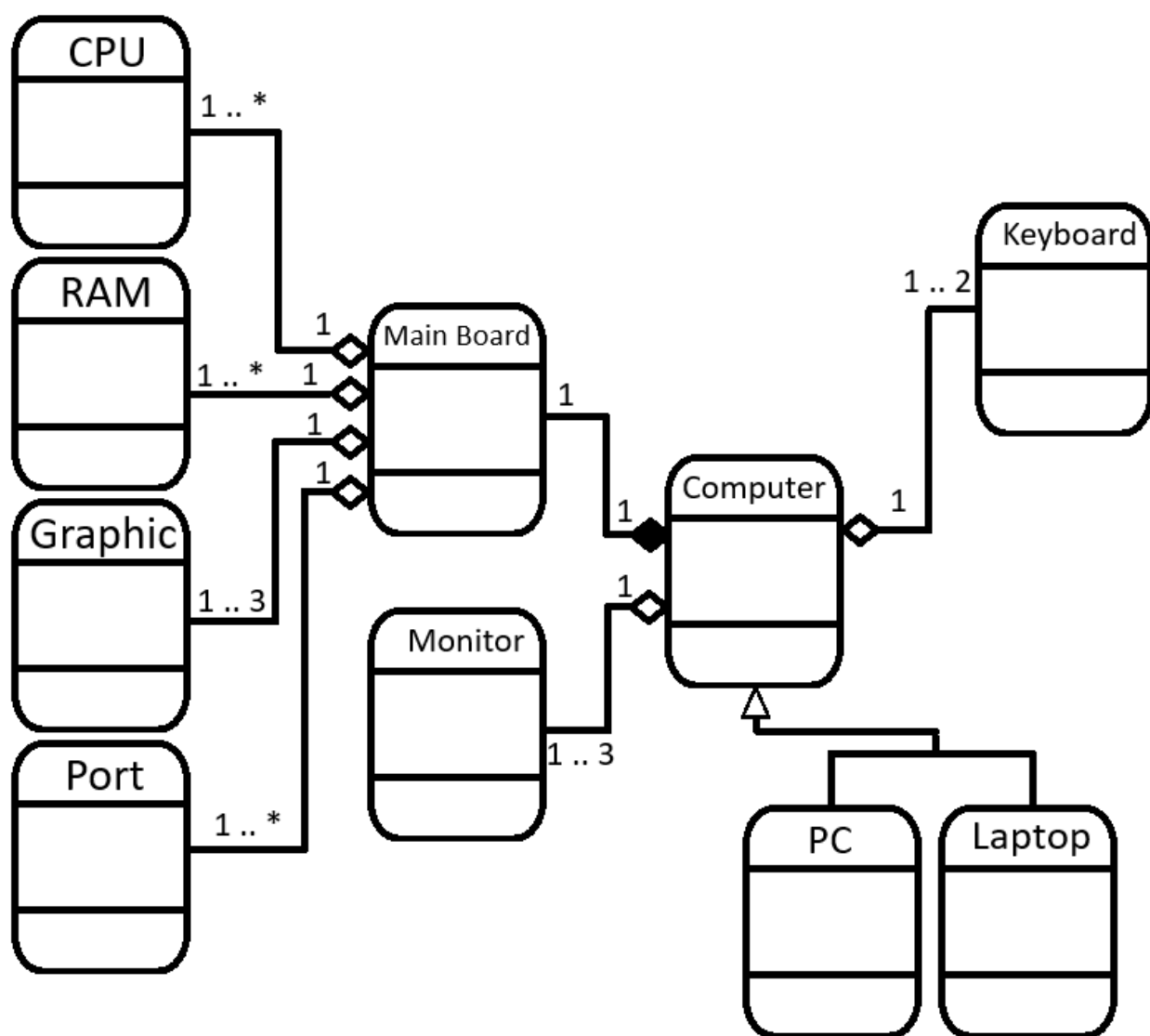
اگر بتوان گزاره‌ی "---- جزئی از ----" گفت پس این ارتباط Aggregation است. مثال: دانشکده جزئی از دانشگاه است.

انواع:

Composition (◆—): رابطه منحصر به فرد.

Aggregation (◇—): رابطه‌ی کل به جزئی که منحصر به فرد نیست.

مثال:



دیاگرام Activity:

نمودار فعالیت یک نمودار رفتاری است.

هدف: مدل سازی فرایندهای سیستم.

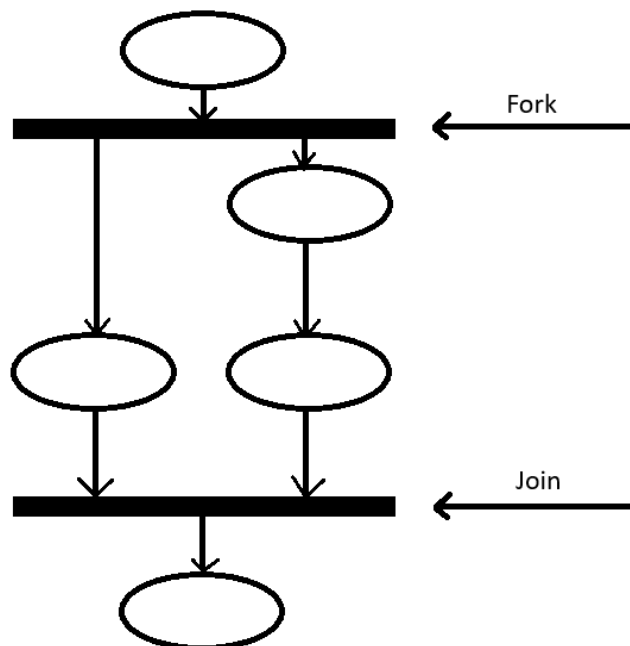
اجزای تشکیل دهنده:

۱. کارهای اولیه ی هر فرایند (Action States) ○
۲. وضعیت شروع فرایند (Start State) ●
۳. عبور بین وضعیت ها (Transition) ↓
۴. پایان فرایند (Final State) ⊙
۵. اعمال شرط در رویه فرایند (Condition Box) ◇

۶. پردازش موازی (Fork & Join)

۱.۶ Fork: ایجاد thread جدید.

۲.۶ Join: متصل شدن threadها.



۷. تفکیک گر فرایند به بخش هایی که در هر کدام یک object پیش می رود (Swimlane)

فصل چهاردهم – استراتژی‌های آزمایش نرم‌افزار (Software Testing)

تعریف تست نرم‌افزار: فرایند اجرای نرم‌افزار با هدف یافتن خطاهای پنهان آن.

تعریف تست خوب: تستی که با بیشترین احتمال خطاهای پنهان نرم‌افزار را بیابد.

برای انجام تست باید داده‌هایی آماده شود و با کمک آن داده‌ها نرم‌افزار تست شود. مجموعه داده‌های خوب برای تست، داده‌هایی هستند که با بیشترین احتمال خطاهای پنهان نرم‌افزار را بیابند.

دو گروه داده داریم: ۱. مشخصات تست (Test Specification):

۱. تکنیک‌ها و روش‌های برنامه‌ریزی برای تست

۲. نمونه‌های یا داده‌های تست

۳. نتایج موردانتظار

۲. مشخصات نرم‌افزار (Software Specification):

۱. مستندات تحلیل نرم‌افزار

۲. مستندات طراحی

۳. سورس کد

مراحل تست: چهار مرحله‌ی تست، تست واحدها، تست مجتمع‌سازی، تست اعتبارسنجی و تست سیستم نام دارند.

۱. تست واحدها (Unit Test): هر واحد نرم‌افزار (مثلاً یک تابع، متد یا کلاس و ...) در این مرحله بصورت مجزا باید تست شود. هر واحد جداگانه ارزیابی می‌شود. در این مرحله نیاز به سورس کد داریم.

۲. تست مجتمع‌سازی (Integration Test): در این مرحله واحدها با هم ترکیب شده و تست می‌شوند. در این مرحله به مستندات طراحی نیاز است که شامل نمودارهای طراحی است. با کمک این مستندات تست رخ می‌دهد. نتیجه و برآیند این مرحله یک نرم‌افزار اسمبل شده می‌باشد.

۳. تست اعتبارسنجی (Validation Test): در این مرحله که نرم‌افزار به‌صورت فنی تست شده است، به کاربر ارائه می‌شود. این مرحله تست تایید کاربر است. با این که نرم‌افزار کارایی لازم را دارد، اما باید کاربر نیز آن را تست کند و باید دید که نرم‌افزار از دید او تأیید می‌گردد یا خیر. در این مرحله به مستندات نیازمندی‌های نیاز است.

۴. تست سیستم (System Test): این تست، تست ارتباطات نرم‌افزار با نرم‌افزارها، سخت‌افزارها و سرورهای خارجی است. نیازمندی این مرحله مستندات طراحی سیستم است.

دسته‌بندی تست‌ها:

۱. تست بازبینی (Verification Test)

- تست واحدها (Unit Test)
- تست مجتمع‌سازی (Integration Test)

۲. تست اعتبارسنجی (Validation Test)

۳. تست سیستم (System Test)

* تست بازبینی مربوط به درست کار کردن نرم‌افزار است. یعنی که برنامه بدون خطا اجرا شود.

* تست اعتبارسنجی مربوط به کار درست کردن نرم‌افزار است. یعنی خروجی نرم‌افزار مورد تأیید کاربر باشد.

مراحل تست بازبینی (Verification Test)

تست بازبینی دو مرحله‌ی اصلی دارد:

۱. تست واحدها (Unit Test):

- تست جعبه سفید (Black Box Test)
- تست جعبه سیاه (White Box Test)

۲. تست مجتمع‌سازی (Integration Test):

- تست از بالا به پایین (Top Down Test)
- تست پایین به بالا (Bottom Up Test)

در تست واحدها، هر جز یا واحد (Unit) به‌عنوان يك ماژول (Module) در نظر گرفته می‌شود. این ماژول می‌تواند يك تابع، متد و یا هر جزء قابل تست دیگری از سیستم باشد. برای آزمایش هر ماژول در روش تست واحدها مراحل و روش‌های مختلفی باید انجام شود:

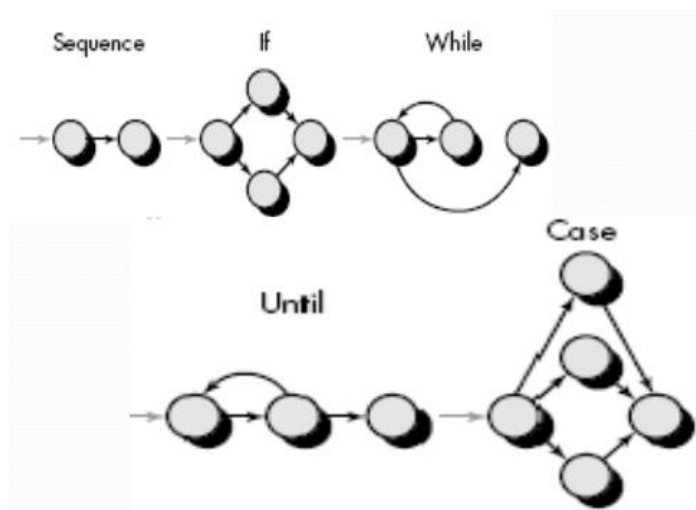
- تست Interface: بررسی صحت و سازگاری ورودی‌ها و خروجی‌های ماژول، شما پارامترهای ورودی و مقادیر بازگشتی.
- تست Local Data Structures: بررسی ساختمان‌های داده داخلی که توسط ماژول استفاده می‌شوند تا اطمینان حاصل شود که داده‌ها به درستی مدیریت می‌شوند.

- تست Boundry Conditions: شناسایی و بررسی شرایط مرزی یا خاص، مانند حداقلها، حداکثرها و مقادیر خارج از محدوده مجاز.
- تست Independent Paths: آزمایش مسیرهای اجرایی مستقل در کد برای اطمینان از پوشش تمام شاخه‌های ممکن.
- تست Error Handling Paths: بررسی مسیرهای مدیریت خطا برای اطمینان از رفتار مناسب ماژول در مواجهه با خطاها.

در بخش بالا به مواردی که در آزمایش و تست باید بررسی شوند اشاره کردیم. برای تست واحدها به دو ابزار دیگر نیازمندیم که Driver و Stub نام دارند. Driver ماژولی است که نقش ورودی دهنده و فراخواننده ماژول تحت آزمایش را ایفا می‌کند. معمولاً شبیه به یک ماژول Main طراحی می‌شود. از طرف دیگر Stub به ماژول‌های شبیه‌سازی شده می‌گویند که توسط ماژول تحت آزمایش فراخوانی می‌شوند و وظیفه بازگشت مقادیر پیش‌بینی شده (بدون تولید خطا) را بر عهده دارند.

تست جعبه سفید: در این تست تمام مسیرهای اجرایی، تمام دستورات و تمام شرطها، تمام ساختارها و جزئیات داخلی ماژول باید تست شود.

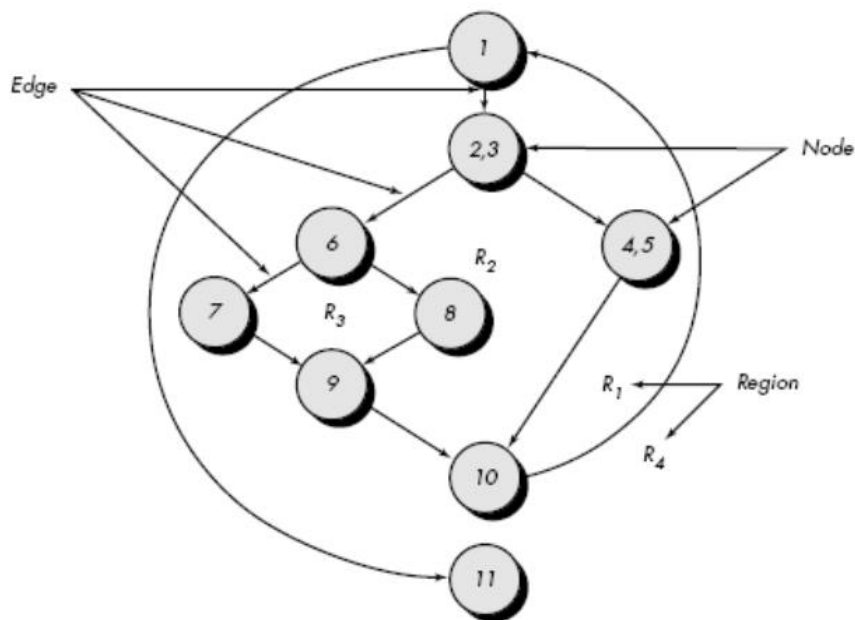
روش‌ها: تست مسیر مبنا (Basis Path Testing): برای این روش ابتدا باید گراف جریان رسم شود که شامل Sequence، If، (then - else)، While، (do while) Case و switch case است. برای رسم گراف جریان باید به خاطر داشت که شرطها باید ساده باشند و اگر شروط پیچیده شوند باید بطور جداگانه و در گراف‌های متفاوت رسم شوند (Combined Logic).



از تست مسیر مبنا می‌توان برای محاسبه پیچیدگی دوره‌ای ماژول یا برنامه استفاده کرد.

تعریف Planer Graph: به گرافی گفته می‌شود که کمان‌هایش از روی یک دیگر عبور نکنند.

روش محاسبه پیچیدگی: با توجه به تعداد نواحی گراف که با خطوط کمان‌ها از هم جدا شده‌اند می‌توان پیچیدگی گراف را محاسبه کرد. مثلاً در مثال زیر چهار ناحیه (R) وجود دارد و پیچیدگی گراف ما نیز ۴ محاسبه می‌شود. هرچقدر تعداد نواحی بیشتر باشد یعنی گراف جریان ما دارای پیچیدگی بیشتری است.



سه روش دیگر برای محاسبه پیچیدگی دوره‌ای نیز وجود دارد:

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

روش دوم:

در این روش E به معنای کمان‌ها و N به معنای گره‌هاست. در تصویر بالا ۱۱ کمان و ۹ گره داریم. پس در نتیجه:

$$11 - 9 + 2 = 4$$

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

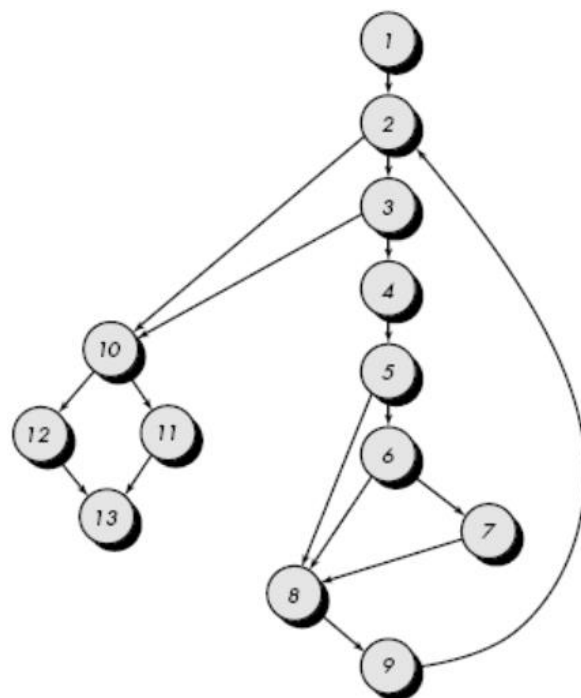
روش سوم:

در این روش P تعداد گره‌هایی هستند که مسیرهای بیشتر از یکی را در پیش دارند. در تصویر بالا گره‌های ۱ و (۲ و ۳) و ۶ سه گره‌ای هستند که در گراف جریان ما به دو مسیر متفاوت رفته‌اند. پس در نتیجه:

$$3 + 1 = 4$$

در هر سه روش برای محاسبه پیچیدگی مثالی که زده شد به یک جواب (۴) رسیدیم.

مسیر مستقل: مسیری که از node ورودی (شروع) به node خروجی مسیر داشته باشد و نباید متشکل از چند مسیر مستقل دیگر باشد. مسیرهای مستقل تصویر زیر بدین صورت می‌باشد:



مسیر مستقل ۱: ۱ - ۲ - ۱۰ - ۱۱ - ۱۳

مسیر مستقل ۲: ۱ - ۲ - ۱۰ - ۱۲ - ۱۳

مسیر مستقل ۳: ۱ - ۲ - ۳ - ۱۰ - ۱۱ - ۱۳

و ...

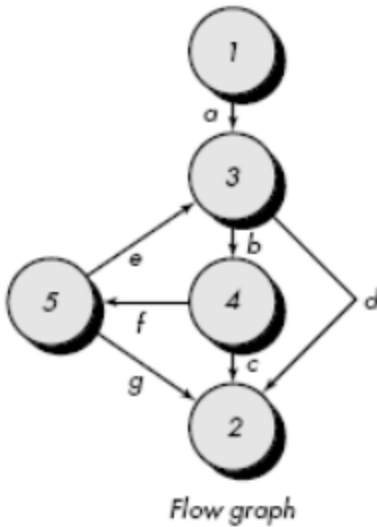
سه روش محاسبه پیچیدگی برای مثال بالا:

$$V(G) = 6 \text{ regions}$$

$$V(G) = 17 \text{ edges} - 13 \text{ nodes} + 2 = 6$$

$$V(G) = 5 \text{ Predicate nodes} + 1 = 6$$

روش چهارم محاسبه پیچیدگی گراف توسط ماتریس انجام می‌شود: در این روش ماتریس مربعی می‌کشیم که به تعداد nodeها متغیر دارد. خانه‌هایی که در آن nodeهای ستون و سطر به هم متصل هستند را پر می‌کنیم (با نام متغیر یا عدد دلخواه). در نهایت تعداد خانه‌های پر در هر سطر را منهای یک کرده و بعد حاصل هر سطر را با سطور دیگر به صورت ستونی جمع می‌کنیم و حاصل به دست آمده به علاوه یک می‌شود.



Connected to node		1	2	3	4	5	Connections
Node							
1				1			1 - 1 = 0
2							
3							
4							
5							

	1	2	3	4	5	
1			1			2 - 1 = 1
2						2 - 1 = 1
3		1		1		2 - 1 = 1
4		1			1	
5		1	1			

Graph matrix

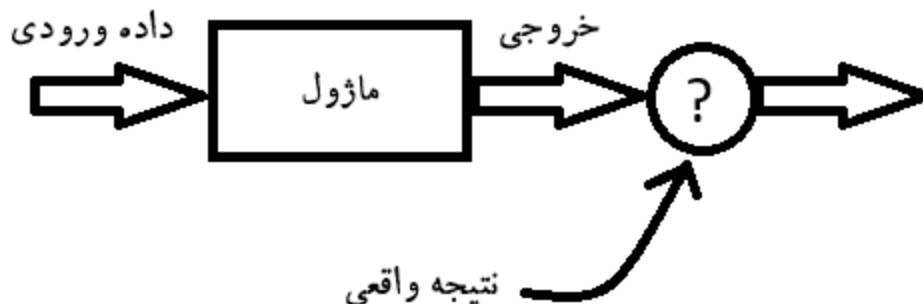
$\overline{3} + 1 = 4$ ← Cyclomatic complexity

تست ساختارهای کنترلی: به تستی که عملگرهای ارتباطی، متغیرهای بولی و خطاهای منطقی و ... را چک می کند. در این خطاها سینتکس درست است اما برنامه از لحاظ منطقی اشتباه است. بعنوان مثال فرض کنید به جای نوشتن $if(l + j < (k * m))$ به اشتباه $if(l + (j < k * m))$ نوشته شود. درست است که هر دو تکه کد بدون ایراد اجرا می شوند اما مورد دوم از لحاظ منطقی ایجاد ایراد می کند.

تست حلقه ها: به این خاطر که حلقه ها در برنامه بارها تکرار می شوند باید در فرایند تست به خوبی ارزیابی شوند. برای هر حلقه باید مراحل زیر به دقت ارزیابی شود تا از صحت عملکرد آن حلقه اطمینان حاصل کنیم:

۱. حلقه بکلی نادیده گرفته شود: شرایطی را چک کنیم که جریان برنامه نباید وارد حلقه شود.
 ۲. شرایط ارزیابی را طوری ارائه دهیم که تنها یک بار در شرط حلقه قرار بگیریم و حلقه یک بار اجرا شود.
 ۳. شرایط ارزیابی را طوری ارائه دهیم که دوبار در شرط حلقه قرار بگیریم و حلقه دوبار اجرا شود.
 ۴. شرایط ارزیابی را طوری ارائه دهیم که اگر حلقه n بار امکان اجرا دارد، ما m بار در شرط حلقه قرار بگیریم و حلقه m بار اجرا شود (تنها در صورتی که $m < n$)
 ۵. شرایط ارزیابی را طوری ارائه دهیم که بیشتر از n بار در حلقه قرار بگیریم.
- * اگر حلقه های تو در تو داشته باشیم در ابتدا باید حلقه های داخلی تست شوند و سپس حلقه های خارجی.

تست جعبه سیاه: برای برنامه هایی استفاده می شود که به ضریب ۱۰۰ درصدی درستی احتیاجی ندارند. یعنی که اگر برنامه در ۹۸ درصد شرایط هم به درستی کار کند ایرادی ندارد. شمای کلی این روش بصورت زیر است:

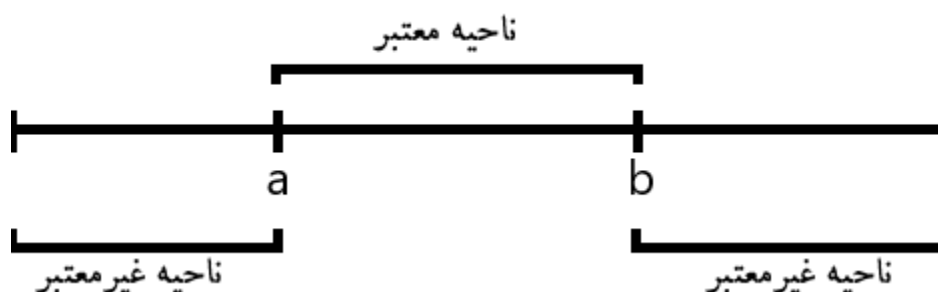


خروجی با نتیجه واقعی مقایسه می گردد.

مواردی که در تست جعبه سیاه بررسی می شوند: توابع ناقص یا دارای ایراد - عیب های ورودی و خروجی های توابع - ایرادات ساختمان های داده یا ایرادات دسترسی به پایگاه های داده ی خارجی - ایرادات رفتاری یا اجرایی - ایرادات مربوط به مقداردهی اولیه.

روش های تست جعبه سیاه: متدهای تست بر پایه ی گراف - روش تقسیم بندی مساوی - تحلیل مقادیر مرزی - تست مقایسه ای - تست آرایه ی استاندارد

روش تقسیم بندی مساوی (Equivalence Partitioning): برای داده های ورودی در يك بازه (مثلا بین a و b) مشخص می شود. محدوده ی کوچک تر یا بزرگ تر از این بازه نواحی غیرمعتبر و این محدوده، ناحیه معتبر نام دارد. داده های در نواحی غیرمعتبر نیز باید در فرایند تست و ارزیابی جزء داده های ورودی باشند. برای مثال اگر برای ثبت نمره در سامانه بازه ی نمره های بین ۰ تا ۲۰ معتبر است، باید نمرات ۲۱ یا نمرات منفی نیز در فرایند تست امتحان شود تا عملکرد سیستم در این شرایط را مورد سنجش قرار دهیم.



* گاهی برای ناحیه معتبر ممکن است به جای يك بازه تنها داده ی ورودی معتبر يك مقدار مشخص باشد. در چنین شرایطی باید مقادیر بیشتر یا کوچکتر از آن مقدار مشخص نیز در فرایند ارزیابی تست شود.

* در این تست نوع داده ی ورودی نباید تغییر کند و مساله ارزیابی خود مقدار ورودی است و نه نوع آن.

تحلیل مقادیر مرزی (Boundry Value Analysis): تست مقادیر بسیار نزدیک و لب مرز در بازه‌ی a و b . مثال: ماژول محاسبه‌ی معدل را در نظر بگیرید. مقادیر حساس این ماژول می‌تواند ۱۰ (برای تشخیص پاس شده یا رد شده)، ۱۲ (برای تشخیص مشروطی)، ۱۷ (برای تشخیص معدل الف شدن) باشد. باید مقادیر لب مرزی به دقت چک شود. مثلاً اگر $a = 10$ باشد باید مقادیر $a + \epsilon$ و $a - \epsilon$ محاسبه گردد یعنی نمرات ۱۰/۰۱ و ۹/۹۹.

تست آرایه استاندارد (Orthogonal Array Testing): فرض کنید تابع زیر یکی از اجزا و ماژول‌های ماست:

```
function a(x,y,z) {  
    -----  
    -----  
    -----  
}
```

در این روش باید ترکیب تمامی مقادیر ممکن برای هر پارامتر چک شود.

تست مجتمع‌سازی (Integration Test):

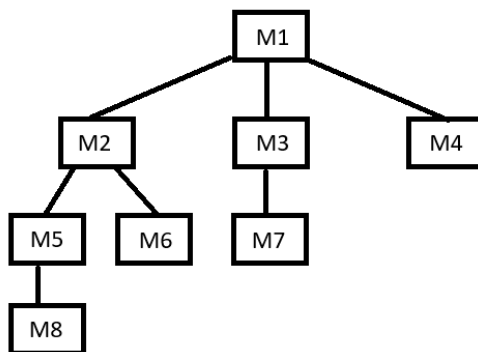
این تست دو روش دارد: ۱. Top Down Integration

- BFS
- DFS

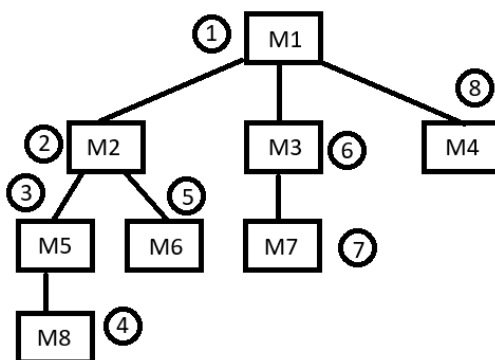
۲. Bottom Up Integration

ترتیب فراخوانی Top Down Integration: در این روش ماژول‌ها به ترتیب از بالا به پایین فراخوانی می‌شوند.

درخت زیر را در نظر بگیرید:

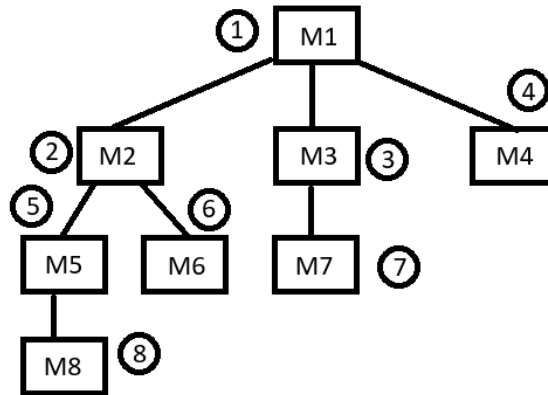


روش DFS: روش جستجوی اول عمق در ابتدا از ماژول بالا تا انتهای عمق درخت می‌رود و سپس به مرور به بالا برمی‌گردد. ترتیب فراخوانی ماژول‌ها در درخت موردنظر در روش dfs به این ترتیب است:



روش کار در این است که در ابتدا به جای هر ماژول يك Stub می‌گذاریم و بعد از اجرای بدون ایراد، خود ماژول را جایگزین می‌کنیم. مثلاً اول به جای m2 يك stub می‌گذاریم و اگر برنامه درست اجرا شد، ماژول m2 را در جای خود قرار داده و به جای فرزندانش stub قرار می‌دهیم. این روند تا اجرای تمام ماژول‌ها بدون ایراد ادامه خواهد داشت.

روش BFS: روش جستجوی اول سطح به همان ترتیب از بالا تا پایین مراحل را طی می‌کند؛ با این تفاوت که به جای کنکاش در عمق در ابتدا هر سطح درخت را طی می‌کند و بعد به سطح بعدی می‌رود. ترتیب فراخوانی ماژول‌ها در درخت موردنظر در روش bfs به این ترتیب است:



روش کار این گونه است که در ابتدا به جای هر ماژول يك Stub می گذاریم و بعد از اجرای بدون ایراد، خود ماژول را جایگزین می کنیم. مثلاً اول به جای m2 يك stub می گذاریم و اگر برنامه درست اجرا شد، ماژول m2 را در جای خود قرار داده و به جای ماژول مجاورش که هنوز چک نشده يك stub قرار می دهیم. این روند تا اجرای تمام ماژول ها بدون ایراد ادامه خواهد داشت.

تست Bottom Up Integration: این تست از روش خوشه ای (cluster) استفاده می کند. در این روش از ماژول نهایی شروع می کنیم و به جای ماژول والد يك Drive می گذاریم. اگر ماژول بدون ایراد اجرا شد، ماژول والد دوباره جایگزین می شود و تست دوباره اجرا می شود. این روش پیوسته و بصورت خوشه ای به سمت بالا حرکت می کند و به ترتیب به جای هر ماژول يك Drive گذاشته می شود و هنگامی که به نتیجه دلخواه رسیدیم به جای Drive ماژول اصلی می نشیند.