آزمون نرم افزار

SOFTWARE TESTING

- مهندسی نرم افزار۲

فهرست مطالب

اهمیت آزمون و طراحی آزمون

- ✓ **آزمون:** اجرای مجموعه ای از فعالیت ها برای اطمینان یافتن از اینکه نرم افزار عملکرد مورد نظر را به خوبی انجام می دهد.
- ✓ **هدف آزمون:** یافتن حداکثر خطاها با حداقل تلاش و زمان
- ✓ آزمون نرم افزار، آخرین فرصت برای سنجش کیفیت و کشف خطاها قبل از تحویل محصول به مشتری است.
- ✓ در هنگام توسعه نرم افزار، مرحله آزمون بیشترین کار فنی را نیاز دارد.

درستی سنجی و اعتبار سنجی (V & V)

- ✓ درستی سنجی و اعتبار سنجی شامل مجموعه وسیعی از فعالیت ها است که از آنها به عنوان تضمین کیفیت نرم افزار نام برده می شود.

❖ درستی سنجی (Verification)

- مجموعه ای از فعالیت ها که پیاده سازی صحیح یک عملکرد خاص توسط نرم افزار را تضمین می کند.
- آیا محصول به درستی ایجاد شده است؟

❖ اعتبار سنجی (Validation)

- مجموعه ای از فعالیت ها که تضمین می کند نرم افزار تولید شده با خواسته های مشتری مطابقت دارد.
- آیا محصولِ درستی ایجاد شده است؟

- ✓ آزمون نرم افزار بخشی از موضوع درستی سنجی و اعتبار سنجی نرم افزار است.

مورد آزمون (Test-Case)

❖ مورد آزمون

- نقطه شروع برای اجرای آزمون
- شامل مجموعه ای از ورودی ها (داده های آزمون)، پیش شرط ها (شرایط پیش از اجرای آزمون) و نتایج مورد انتظار

❖ خصوصیات موارد آزمون خوب

- احتمال یافتن خطا را افزایش می دهند.
- شامل موارد تکراری و اضافی نیستند.
- بیش از حد ساده و بیش از حد پیچیده نیستند.
- قابل اطمینان و قابل نگهداری هستند.

در طراحی موارد آزمون تنها یک قاعده وجود دارد:
همه چیز را پوشش دهید اما موارد آزمون را بیش از حد زیاد نکنید.

مورد آزمون ...

❖ **مثال**

```
if (a == b)
      print(a + b);
else
      print(a * b);
```

	داده های آزمون		نتیجه مورد انتظار
مورد آزمون 1	a=3	b=4	12
مورد آزمون 2	a=3	b=3	6
✖ مورد آزمون 3	a=5	b=12	60
✖ مورد آزمون 4	a=18	b=18	36

آزمون پذیری (Testability)

- هر چه میزان کار لازم برای آزمون نرم افزار کمتر باشد، قابلیت آزمون آن بالاتر است.
- یک نرم افزار آزمون پذیر باید دارای صفات زیر باشد:

صفت	توضیح
قابلیت عملیاتی کردن *Operability*	سهولت راه اندازی، کار با نرم افزار و آماده کردن ورودی های لازم برای انجام آزمون
قابلیت فهم *Understandability*	قابل فهم بودن کد برنامه، طراحی، وابستگی میان مولفه های داخلی و خارجی و مستندات
سادگی *Simplicity*	کوچک بودن مجموعه نیازمندی ها، رعایت استاندارد های کد نویسی و طراحی پیمانه ای (قابلیت تجزیه) موجب کوچکتر شدن موارد آزمون و تسریع در انجام آزمون می شود.

آزمون پذیری ...

صفت	توضیح
قابلیت مشاهده *Observability*	حالات نرم افزار در هنگام اجرا به راحتی قابل مشاهده و تشخیص باشد. خطاهای داخلی نرم افزار به طور خودکار آشکار و گزارش شود. خروجی های نادرست به آسانی قابل مشاهده باشد. به ازای ورودی های متمایز، خروجی های متمایز تولید شود.
قابلیت کنترل *Controllability*	سهولت تعریف و یا تغییر آزمون ها. آزمون ها در صورت تمایل به صورت خودکار انجام شوند. همه خروجی های ممکن را بتوان از طریق ترکیبی از ورودی ها تولید کرد. سازگار و ساخت یافته بودن فرمت های ورودی و خروجی. حالات نرم افزار در هنگام اجرا توسط آزمونگر قابل کنترل باشند.
پایداری *Stability*	تغییرات نرم افزار چندان زیاد نباشد. در صورت وقوع تغییر، تغییر قابل کنترل باشد و آزمون های قبلی را بی اعتبار نکند. در صورت وقوع شکست، نرم افزار قابل ترمیم باشد.

راهبرد کلی آزمون نرم افزار

❖ **هر راهبرد آزمون، راهنما و الگویی را برای انجام آزمون معرفی می کند.**

- برنامه ریزی و زمانبندی مراحل اجرای آزمون
- چگونگی طراحی موارد آزمون
- زمان و منابع لازم برای اجرای آزمون
- معیار ها و روش اندازه گیری پیشرفت آزمون

❖ **چهار مرحله اصلی برای انجام آزمون نرم افزار**

آزمون های سطح پایین

۱- آزمون واحد

۲- آزمون انسجام

آزمون های سطح بالا

۳- آزمون اعتبار سنجی

۴- آزمون سیستم

راهبرد کلی آزمون نرم افزار ...

❖ **آزمون واحد (Unit Test)**

- یک آزمون سطح پایین است که بر هر واحد (مولفه) از نرم افزار تاکید دارد.
- ساختار کنترلی، منطق و ساختمان داده های هر یک از مولفه ها به طور جداگانه تست می شود.

❖ **آزمون انسجام (Integration Test)**

- یک آزمون سطح پایین است که بر طراحی و معماری نرم افزار تاکید دارد.
- بررسی نحوه ترکیب مولفه ها با یکدیگر و اثرات جانبی آن
- تست روابط میان مولفه ها
- تلاش برای کشف خطا ها در واسط بین مولفه ها

راهبرد کلی آزمون نرم افزار ...

❖ آزمون اعتبار سنجی (Validation Test)

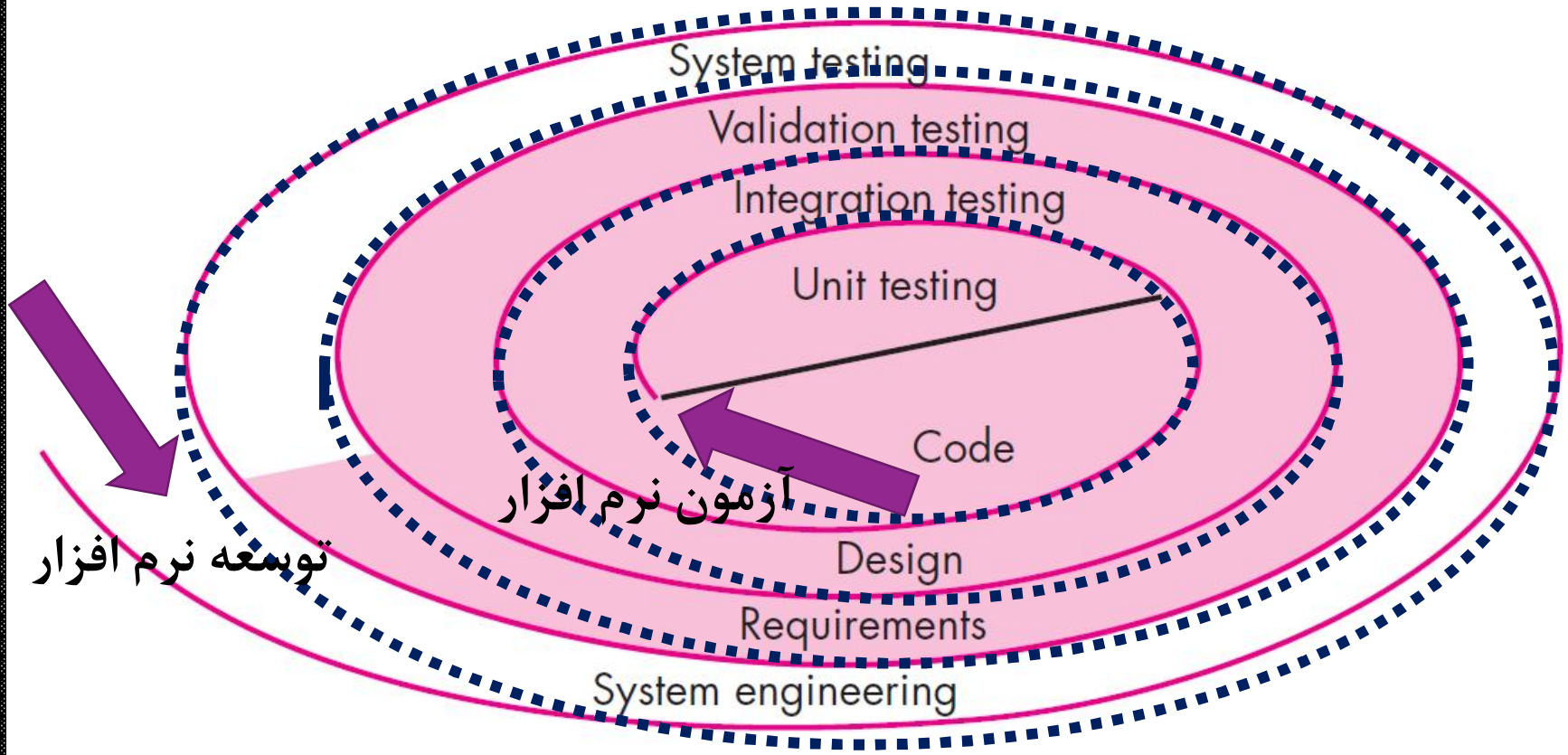
- یک آزمون سطح بالا است که هدف آن اطمینان از رعایت همه نیازمندی های عملیاتی، رفتاری و کارایی مورد نظر مشتری می باشد.
- آیا نرم افزار تولید شده با نیازمندی های توافق شده در مرحله تحلیل نرم افزار مطابقت دارد؟

❖ آزمون سیستم (System Test)

- یک آزمون سطح بالا است که خارج از مرز مهندسی نرم افزار و در حیطه وسیع تر یعنی مهندسی سیستم های کامپیوتری قرار می گیرد.
- اطمینان از اینکه همه عناصر سیستمی (مثل سخت افزار، نرم افزار، بانک های اطلاعاتی، افراد و غیره) در ترکیب با یکدیگر به درستی عمل می کنند و عملکرد نهایی سیستم قابل دستیابی است.

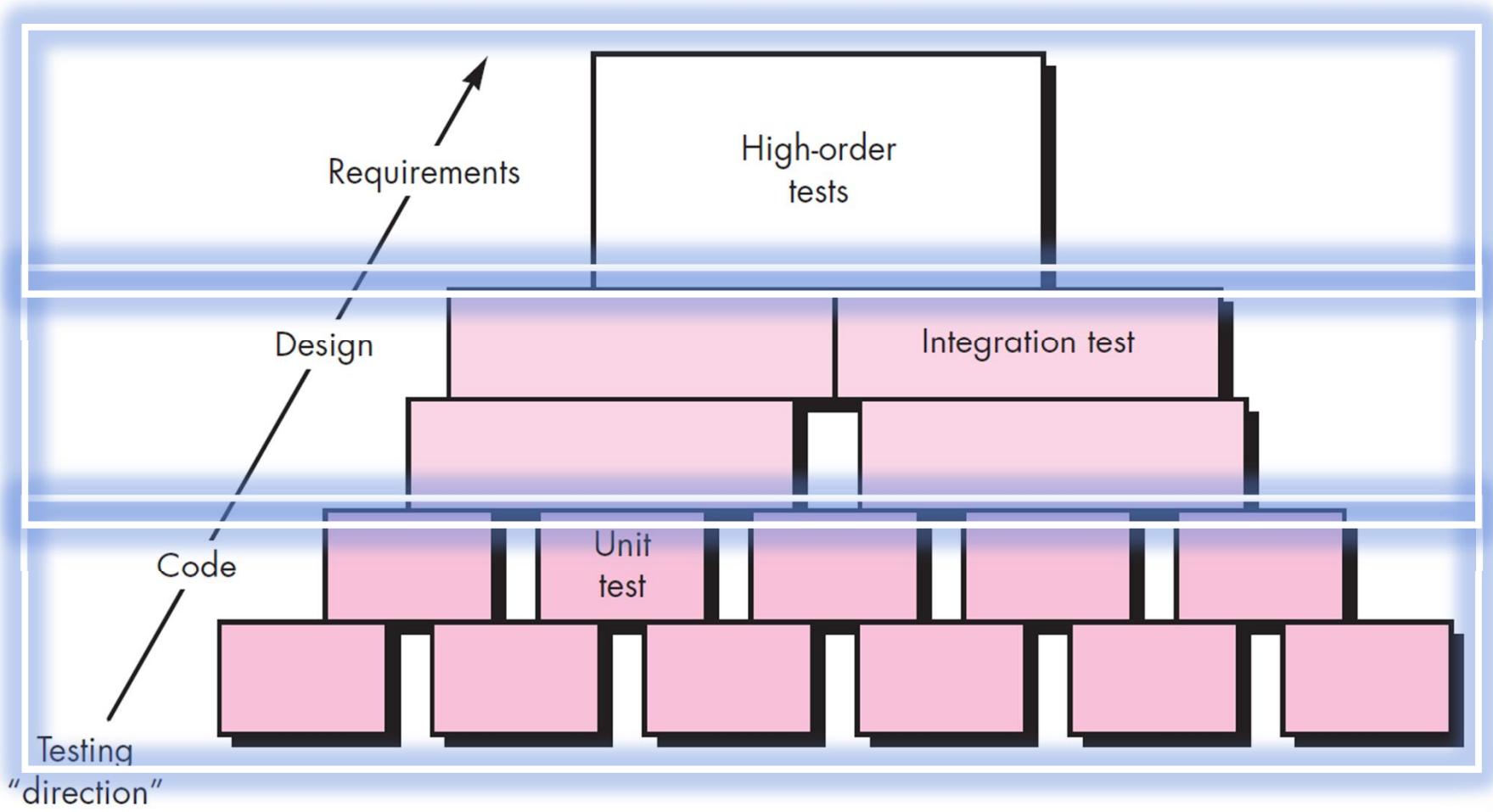
راهبرد کلی آزمون نرم افزار ...

- آزمون در ابعاد کوچک آغاز می شود و به ابعاد بزرگ پیشرفت می کند.



راهبرد کلی آزمون نرم افزار ...

- آزمون در ابعاد کوچک آغاز می شود و به ابعاد بزرگ پیشرفت می کند ...

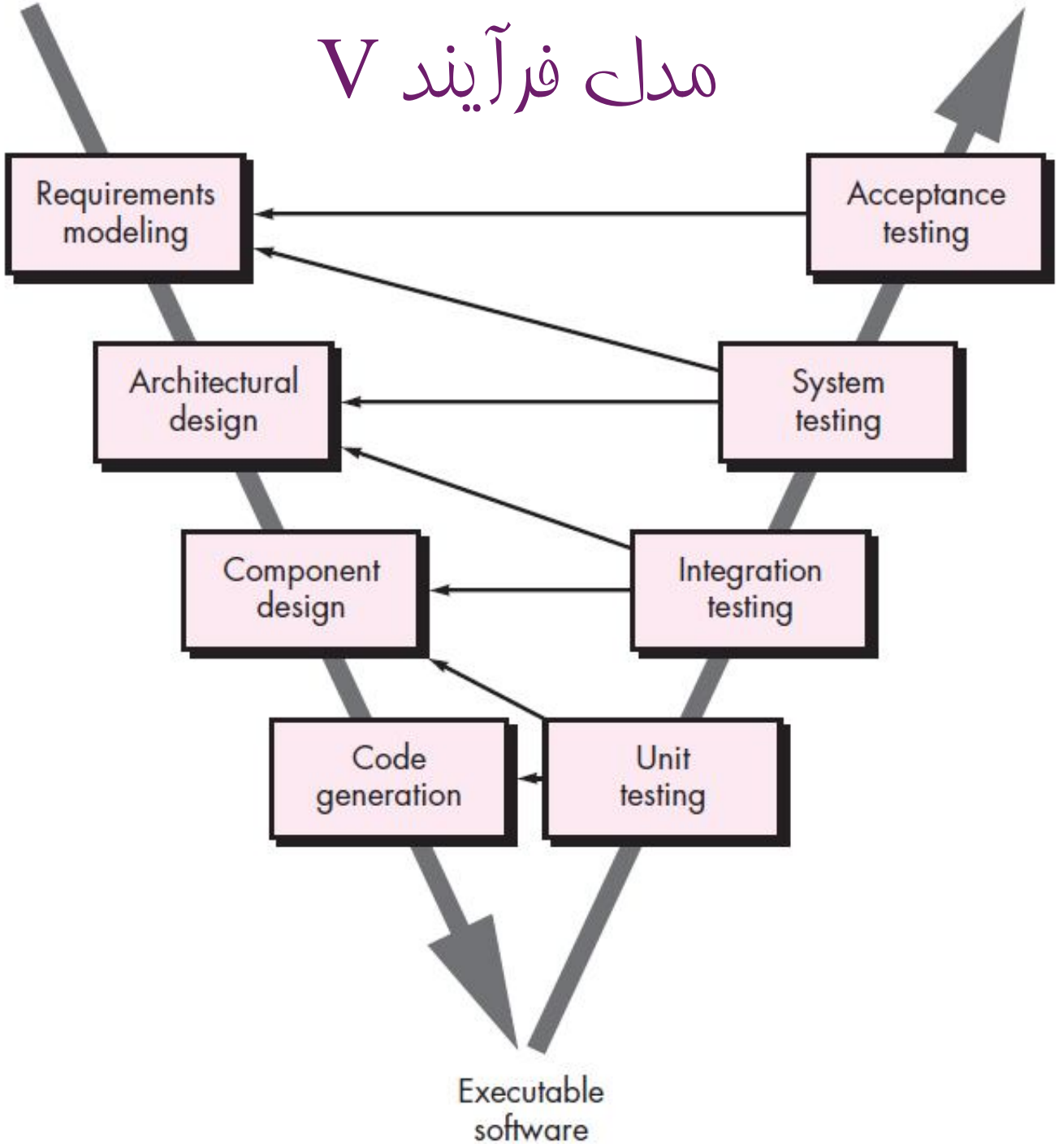


راهبرد کلی آزمون نرم افزار ...

❖ نکاتی برای موفقیت راهبرد آزمون

- تعیین نیازمندی های عملیاتی و غیر عملیاتی نرم افزار به شیوه کمیت پذیر (قبل از شروع آزمون)
- تولید نرم افزار به شیوه مستحکم (robust)
- بازبینی فنی راهبرد آزمون و موارد آزمون طراحی شده (قبل از شروع آزمون)
- بیان اهداف و راهبرد آزمون به صورت واضح و قابل اندازه گیری
  - تعداد ساعات کار برای انجام آزمون، میانگین زمان شکست، درصد خطاهای کشف شده، هزینه کشف و تصحیح خطاها، چگالی نقایص باقیمانده و ...
- شناسایی کاربران نرم افزار و ایجاد یک پروفایل برای هر گروه از کاربران
- تاکید بر چرخه های سریع آزمون

مدل فرآیند V


Requirements modeling
Acceptance testing
Architectural design
System testing
Component design
Integration testing
Code generation
Unit testing
Executable software

مدل فرآیند V ...

- شکل دیگری از مدل آبشاری است که رابطه فعالیت های کنترل و تضمین کیفیت را با فعالیت های فرآیند توسعه نرم افزار نشان می دهد.
- ابتدا با حرکت به سمت پایین مدل، نیازمندی های مساله پالایش شده و جزئیات بیشتری از آنها تعیین می شود و در نهایت کد برنامه پیاده سازی می شود.
- سپس با حرکت به سمت بالای مدل، مجموعه ای از آزمون ها اجرا می شود تا کیفیت محصولات کاری ایجاد شده را مورد بررسی قرار دهد.

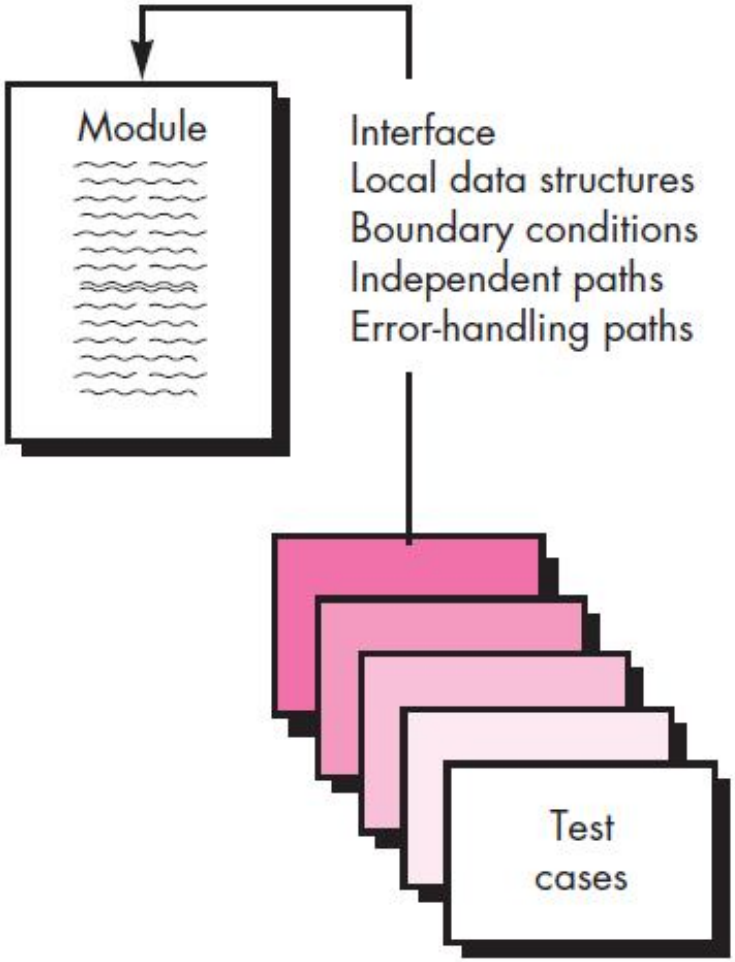
آزمون گران (tester) نرم افزار

- آزمون می تواند توسط سازنده نرم افزار و یا متخصص آزمون انجام شود.
- سازنده نرم افزار
  - آزمون هر یک از مولفه های نرم افزار (آزمون واحد)
  - آزمون انسجام
- گروه آزمون مستقل (ITG)
  - آزمون پروژه های بزرگ را می توان توسط یک تیم آزمونگر مستقل (Independent Test Group) انجام داد.
  - معمولا این تیم پس از تکمیل معماری نرم افزار، کار خود را شروع می کنند.
  - سازنده نرم افزار و ITG باید در سراسر پروژه با هم در ارتباط باشند.

آزمون های سطح پایین
در رویکرد سنتی

آزمون واحد در رویکرد سنتی

- آزمون واحد بر کوچکترین واحد طراحی، یعنی پیمانه تمرکز دارد.
- بخش هایی از پیمانه که مورد آزمون واحد قرار می گیرند:
  - واسط پیمانه
  - ساختمان داده های محلی
  - مسیر های مستقل
  - شرایط مرزی
  - مسیر های کنترل خطا



آزمون واحد در رویکرد سنتی ...

❖ **واسط پیمانه (Interface)**

- اطمینان از اینکه اطلاعات به طور مناسب به درون و بیرونِ پیمانه جریان می یابند.
- جریان داده هایی که از واسط پیمانه عبور می کنند، باید پیش از شروع هر آزمون دیگری، تست شوند.
- اگر داده ها به طور مناسب وارد یا خارج نشوند، همه آزمون های دیگر بیهوده خواهد بود.

❖ **ساختمان داده های محلی (Local Data Structures)**

- اطمینان از اینکه داده های ذخیره شده در پیمانه در طی همه مراحل اجرا، یکپارچگی و صحت خود را حفظ می کنند.

آزمون واحد در رویکرد سنتی ...

❖ شرایط مرزی (Boundary Condition)

- نرم افزار ها اغلب در مرز ها دچار شکست می شوند. برای مثال:
  - پردازش n امین عنصر از یک آرایه n بعدی
  - تکرار n ام در حلقه ای با حداکثر n مرتبه گذر مجاز
  - حداقل و حداکثر مقدار یک متغیر
- اطمینان از اینکه پیمانه در مرز های تعیین شده (برای محدود کردن پردازش) به درستی عمل می کند.

❖ مسیر های مستقل (Independent Paths)

- همه مسیر های مستقل که از ساختار کنترلی پیمانه عبور می کنند بررسی می شوند تا اطمینان حاصل شود که همه دستورات داخل پیمانه حداقل یک بار اجرا می شوند.

آزمون واحد در رویکرد سنتی ...

❖ مسیر های کنترل خطا (Error-handling Paths)

✓ *ضد اشکال سازی (antibugging)*

- *طراحی خوب حکم می کند که شرایط خطا پیش بینی شود و مسیر هایی برای اداره خطا مشخص شود تا در صورت بروز خطا، برنامه بتواند پردازش را دوباره جهت دهی کند یا به آن پایان دهد.*

- مسیر اداره خطا ممکن است هنگام فراخوانی با شکست روبرو شود.
- خطاهای بالقوه در مسیر های کنترل خطا
  - توصیف خطا پیچیده و غیر قابل فهم است.
  - توصیف خطا، اطلاعات کافی را برای ارزیابی مکانی که موجب خطا شده است ارائه نمی دهد.
  - خطایی که به آن اشاره شده است متناظر با خطای رخ داده نیست.
  - شرایط خطا موجب مداخله سیستم قبل از بخش مدیریت خطا شده است.
  - پردازش شرایط خطا نادرست است.

آزمون واحد در رویکرد سنتی ...

- ✓ آزمون واحد معمولا همگام با مرحله کد نویسی انجام می شود.
  - طراحی آزمون های واحد ممکن است قبل از شروع کد نویسی یا پس از آن انجام شود.
- ✓ می توان از اطلاعات طراحی به عنوان راهنما برای تعیین موارد آزمون واحد استفاده کرد.
- ✓ هر چه یکپارچگی (cohesion) مولفه ها بالاتر باشد، طراحی آزمون واحد ساده تر خواهد بود.
- ✓ اگر منابع لازم برای انجام آزمون های واحد به صورت کامل و فراگیر وجود نداشته باشد، می توان تنها پیمانه های بحرانی را مورد آزمون واحد قرار داد.
  - پیمانه بحرانی (Critical Module)
    - با چندین نیازمندی نرم افزار سر و کار دارد.
    - در معماری برنامه جایگاه و سطح کنترل بالایی دارد.
    - پیچیده یا مستعد خطا است.
    - مشخصات کارایی خاصی برای آن در نظر گرفته شده است.

آزمون جعبه سفید (White-box Testing)

- آزمون جعبه سفید = آزمون ساختار کنترلی = آزمون شیشه ای
- برای طراحی موارد آزمون های واحد معمولا از تکنیک های آزمون جعبه سفید استفاده می شود.
- با استفاده از این تکنیک ها می توان منطق داخلی برنامه را تست کرد.
- از آزمون های جعبه سفید تنها پس از طراحی در سطح مولفه ها یا پیاده سازی کد منبع می توان استفاده کرد زیرا جزئیات منطق برنامه باید در دسترس باشد.

آزمون جعبه سفید ...

❖ **تکنیک های آزمون جعبه سفید**

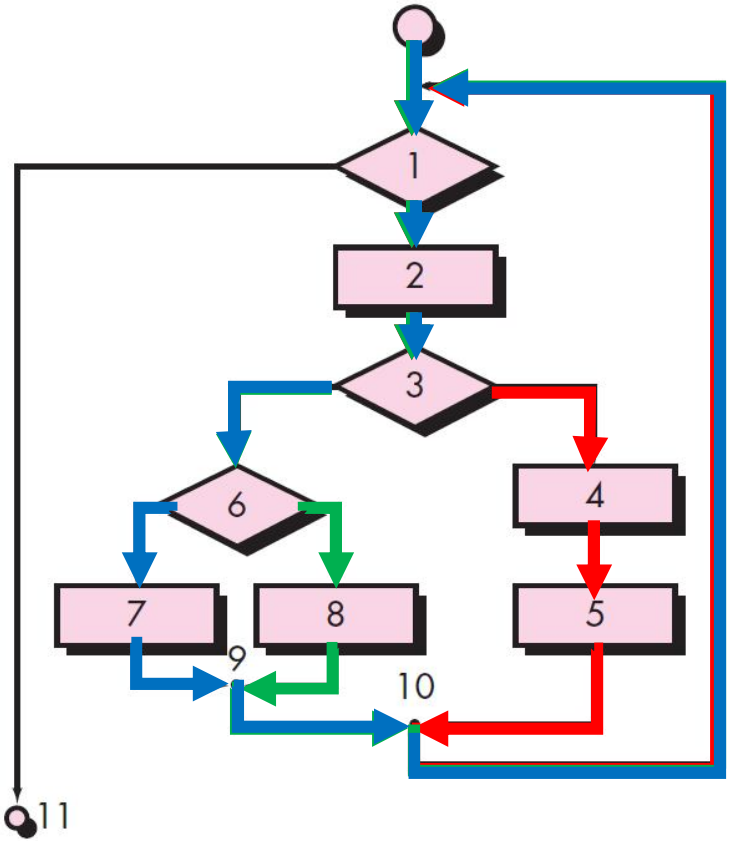
- آزمون مسیر پایه
- آزمون شرط ها
- آزمون حلقه ها
- آزمون جریان داده ها

❖ موارد آزمون طراحی شده توسط تکنیک های جعبه سفید تضمین می کنند که:

- همه مسیر های مستقل در یک پیمانه را حداقل یک بار امتحان کنند.
- همه تصمیم گیری های منطقی را در دو حالت درست و غلط بررسی کنند.
- همه حلقه ها را در مرزها و در داخل مرز های عملیاتی آنها امتحان کنند.
- ساختمان داده های داخلی را امتحان کنند تا درستی آنها ثابت شود.

آزمون مسیر پایه (Basis Path Testing)

- اگر حلقه حداکثر ۲۰ بار تکرار شود، برنامه معادل با نمودار جریان (Flow Chart) زیر حدودا دارای 3^{20} (۳۴۸۶۷۸۴۴۰۱) مسیر اجرایی است.
- آزمون همه این مسیر ها دشوار و غیر ممکن است. (Complete Testing)
- بنابراین تنها می توان زیر مجموعه ای از مسیر های اجرایی را تست کرد. (Selective Testing)



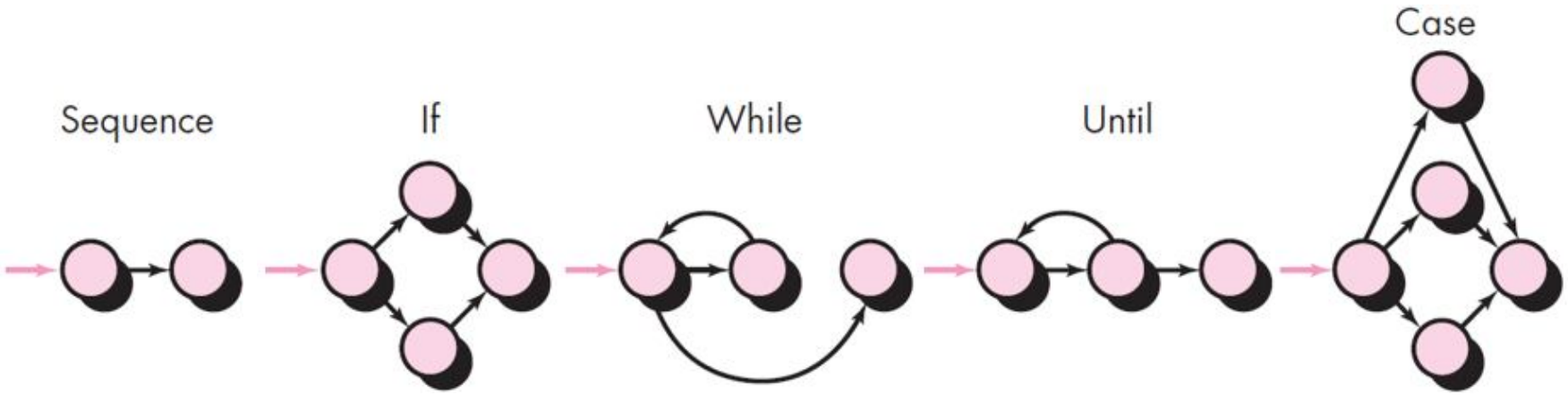
آزمون مسیر پایه ...

- در آزمون مسیر پایه، میزان پیچیدگی منطقی برنامه محاسبه می شود و از آن به عنوان راهنمایی جهت تعریف یک مجموعه پایه از مسیر های اجرایی برنامه استفاده می شود.
- اگر موارد آزمون به گونه ای طراحی شوند که اجرای مسیر ها در مجموعه پایه را تضمین کنند، هر دستور از برنامه حداقل یک بار اجرا خواهد شد و هر دستور شرطی در هر دو حالت درست یا نادرست خود بررسی می شود.
- آزمون مسیر پایه، ساده و اثر بخش است اما کافی نیست.

آزمون مسیر پایه ...

❖ گراف جریان (Flow Graph)

- یک نمایش انتزاعی از ساختار و جریان کنترل برنامه
- در گراف جریان، برای هر ساختار ساخت یافته یک نماد متناظر وجود دارد.
- ساختار های ساخت یافته
  - ترتیب (sequence)
  - شرط (if و case)
  - تکرار (while و until)



آزمون مسیر پایه ...

❖ گره *node*

- نشان دهنده یک یا چند دستور
- دنباله ای از مستطیل های پردازشی و یک لوزی تصمیم گیری در نمودار جریان را می توان به یک گره در گراف جریان نگاشت داد.

❖ گره گزاره ای *predicate node*

- هر گره که حاوی یک شرط باشد.
- دارای دو یا چند پیکان خروجی

❖ یال *edge*

- نمایش دهنده جریان کنترل
- هر یال باید در یک گره پایان یابد؛ حتی اگر آن گره شامل هیچ دستوری نباشد.

❖ ناحیه *region*

- مساحت های محصور شده توسط یال ها و گره ها
- مساحت خارج از گراف نیز به عنوان یک ناحیه در نظر گرفته می شود.

آزمون مسیر پایه ...

❖ شرط ساده *simple condition*

- یک متغیر بولین (Boolean Variable) : مقدار آن برابر با true یا false است.
- یک عبارت رابطه ای (Relational Expression)

E_1 *relational-operator* E_2

- E_1 و E_2 : عبارات محاسباتی
- relational-operator (عملگر رابطه ای): < > = != >= <=
- مثال: a+3<4

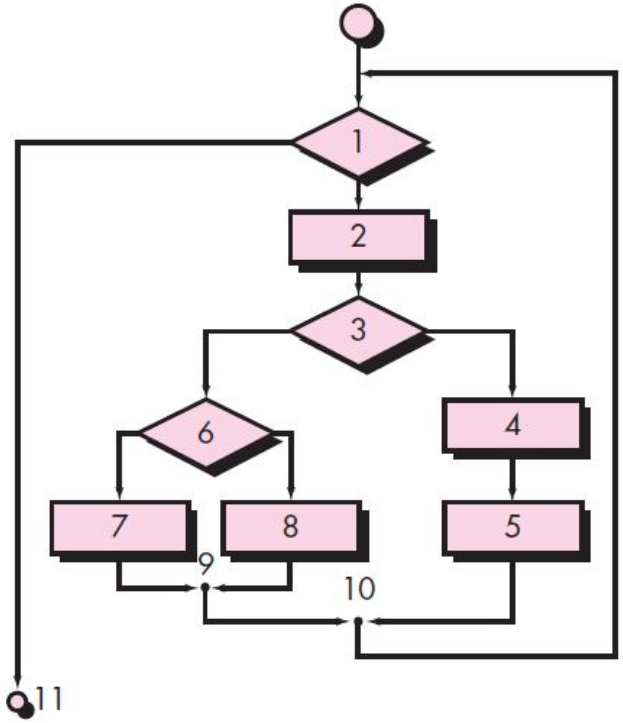
❖ شرط مرکب *compound condition*

- از دو یا چند شرط ساده، عملگر های بولین و پرانتز تشکیل می شود.
- عملگر های بولین (Boolean Operators): OR، AND و NOT
- مثال: (x+5<3 OR y>0) AND (z!=3)
- مثال: a OR b (a و b متغیر های بولین هستند)

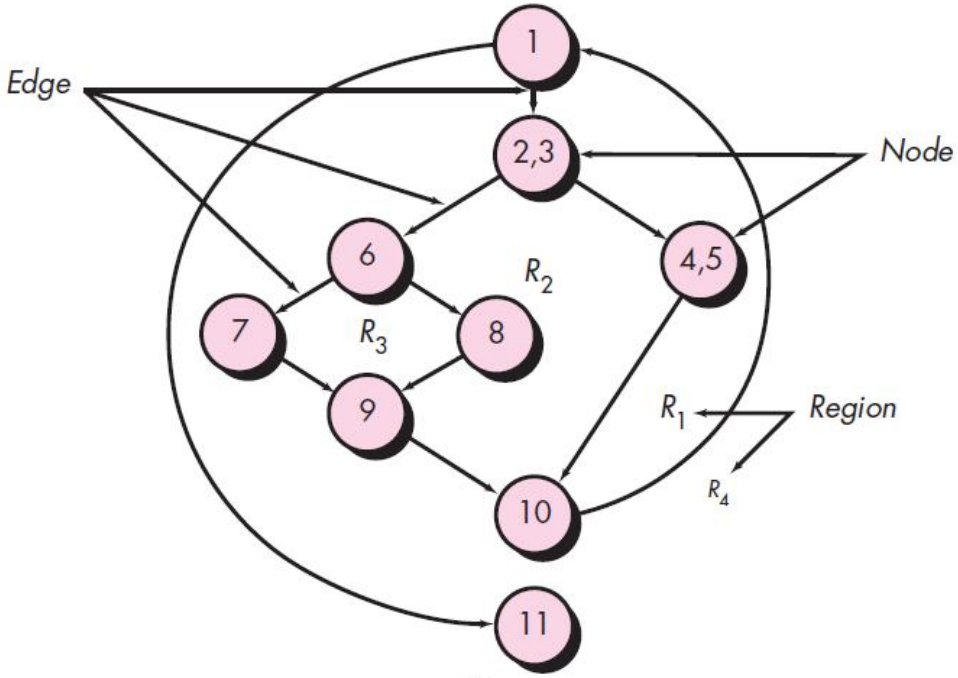
آزمون مسیر پایه ...

❖ مثال

✓ با این فرض که هیچ شرط مرکبی در لوزی های تصمیم گیری وجود ندارد.



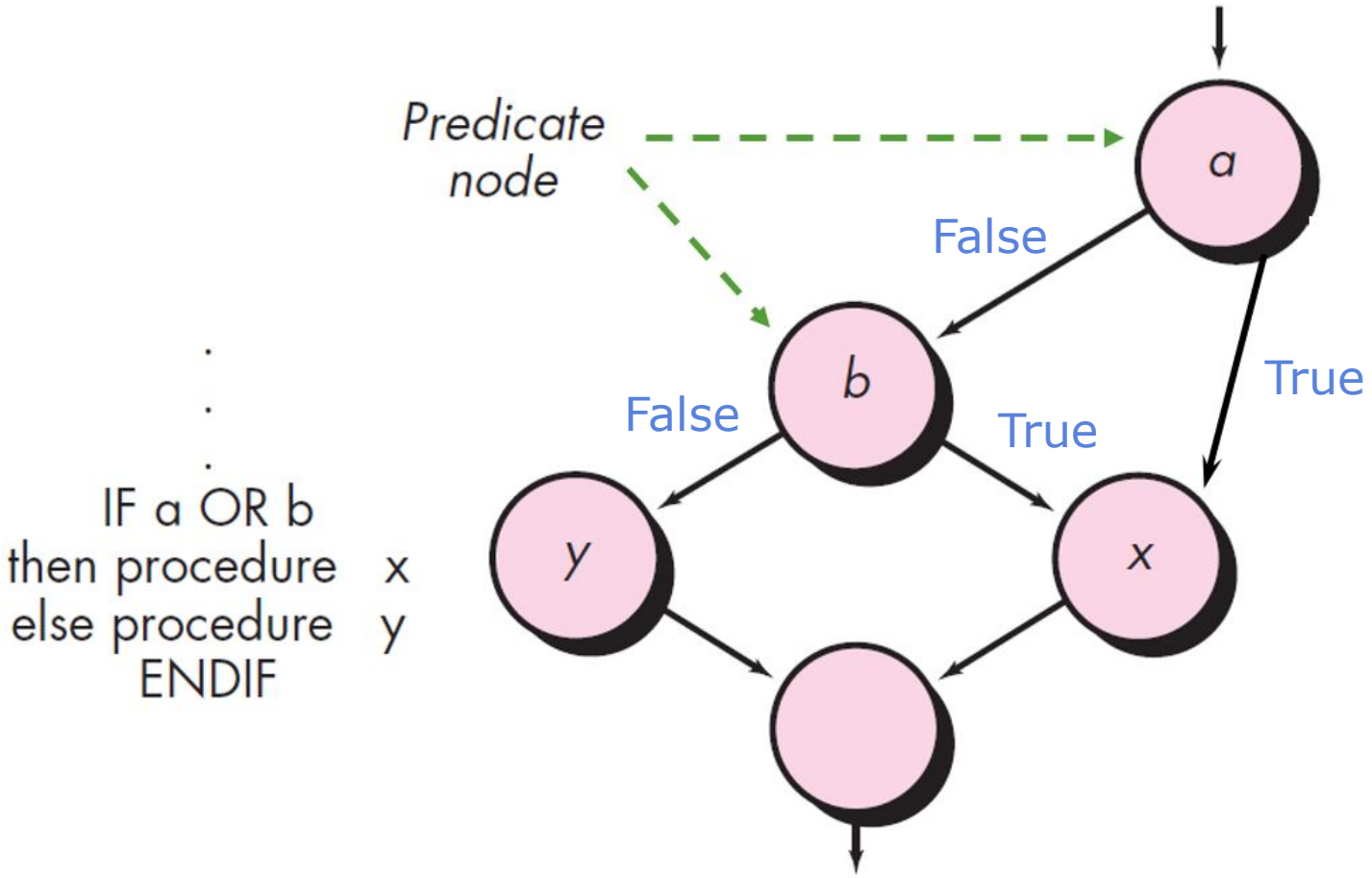
نمودار جریان



گراف جریان

آزمون مسیر پایه ...

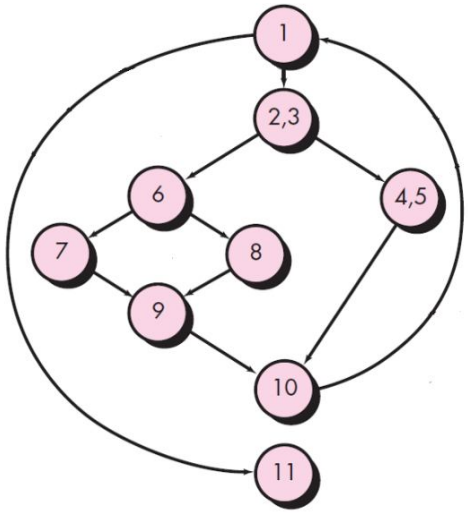
❖ **مثال:** گراف جریان برای ساختار if-else همراه با شرط مرکب



آزمون مسیر پایه ...

❖ پیچیدگی سیکلوماتیک (Cyclomatic Complexity)

E: تعداد یال های گراف جریان N: تعداد گره های گراف جریان	V(G) = E – N + 2
R: تعداد نواحی در گراف جریان	V(G) = R
P: تعداد گره های گزاره ای در گراف جریان	V(G) = P + 1
هر بخش از شرط مرکب به عنوان یک واحد مجزا شمارش می شود.	V(G) = تعداد عبارات شرطی + 1



❖ مثال

V(G)=E-N+2=11-9+2=4

V(G)=R=4

V(G)=P+1=3+1=4

آزمون مسیر پایه ...

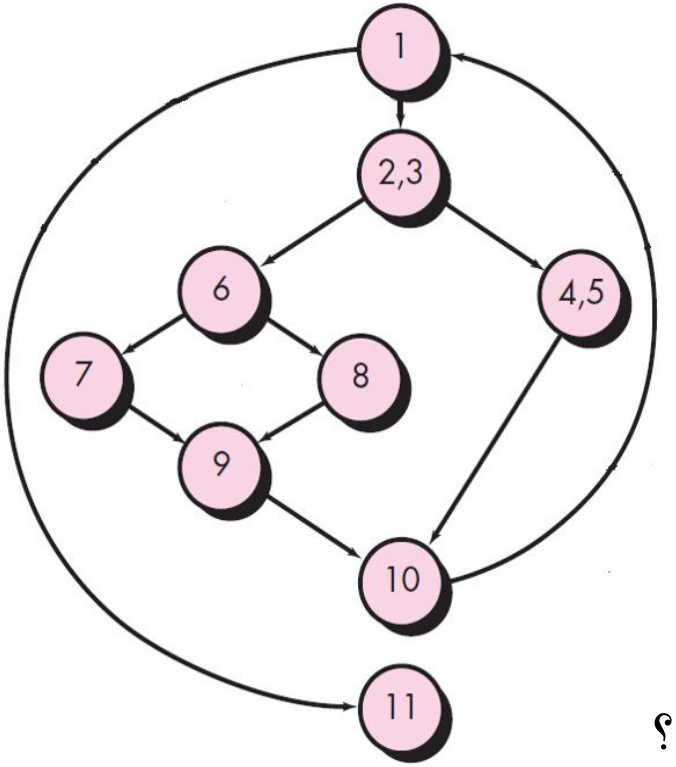
❖ **کاربرد های پیچیدگی سیکلوماتیک**

- شناسایی مولفه های مستعد خطا
  - در مولفه های با V(G) بالاتر، احتمال وجود خطا بیشتر است.
- یک حد فوقانی برای تعداد مسیر های تشکیل دهنده مجموعه پایه
  - مجموعه پایه باید حداکثر شامل V(G) مسیر مستقل باشد.

آزمون مسیر پایه ...

❖ **مسیر مستقل (Independent Path)**

- هر مسیری از برنامه که حداقل یک مجموعه جدید از دستور های پردازشی یا یک دستور شرطی جدید را معرفی کند.



- **مثال:** مسیر های مستقل

Path 1: 1-11

Path 2: 1-2-3-4-5-10-1-11

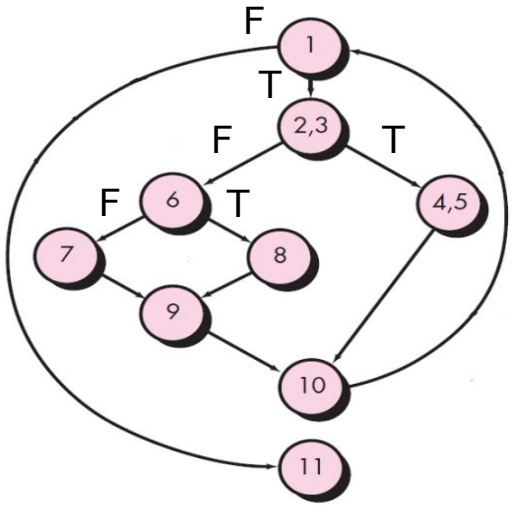
Path 3: 1-2-3-6-8-9-10-1-11

Path 4: 1-2-3-6-7-9-10-1-11

- **سوال:** آیا مسیر ۵ یک مسیر مستقل است؟

Path 5: 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11

آزمون مسیر پایه ...



❖ **مجموعه پایه (Basic Set)**

- مجموعه ای از مسیر های مستقل که به ازای آنها:
  - هر دستور از برنامه حداقل یک بار اجرا می شود.
  - هر دستور شرطی در هر دو حالت درست و نادرست بررسی می شود.

Path 1: 1-11

Path 2: 1-2-3-4-5-10-1-11

Path 3: 1-2-3-6-8-9-10-1-11

Path 4: 1-2-3-6-7-9-10-1-11

	دستورات									شرط ها		
	1	2,3	4,5	6	7	8	9	10	11	1	2,3	6
Path 1	✓								✓	F		
Path 2	✓ ✓	✓	✓					✓	✓	T F	T	
Path 3	✓ ✓	✓		✓		✓	✓	✓	✓	T F	F	T
Path 4	✓ ✓	✓		✓	✓		✓	✓	✓	T F	F	F

آزمون مسیر پایه ...

❖ **مراحل بدست آوردن موارد آزمون مسیر پایه**

۱- رسم گراف جریان بر مبنای طراحی سطح مولفه یا کد منبع

۲- تعیین پیچیدگی سیکلوماتیک

۳- تعیین مجموعه پایه برای مسیر های مستقل

۴- طراحی موارد آزمون که اجرای همه مسیر ها در مجموعه پایه را تضمین می کنند.

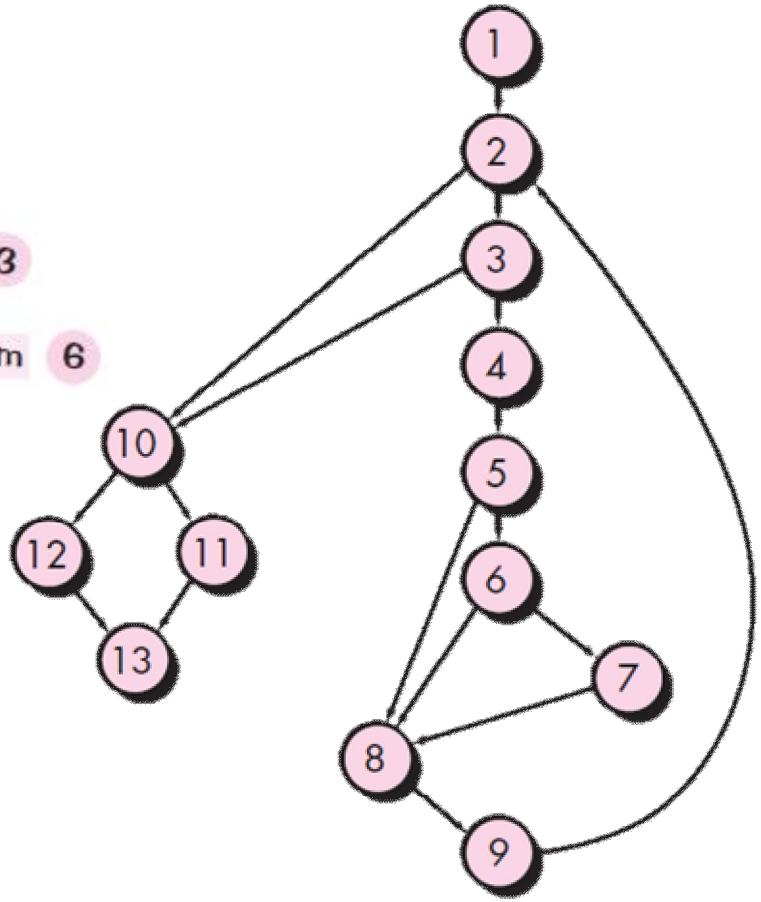
✓ آزمون مسیر پایه بدون استفاده از گراف جریان نیز قابل اجرا است. اما این گراف به درک بهتر جریان کنترل برنامه و نمایش آن کمک می کند.

❖ مثال

شبه کد

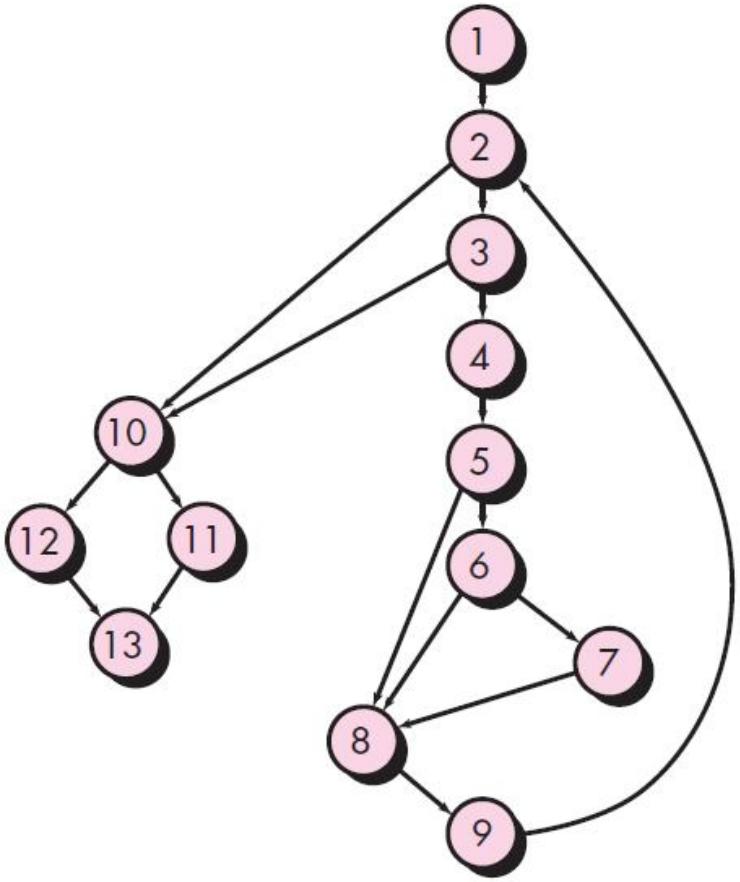
محاسبه میانگین

```
PROCEDURE average;

*  This procedure computes the average of 100 or fewer
   numbers that lie between bounding values; it also computes the
   sum and the total number valid.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
   minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
1  i = 1;
   total.input = total.valid = 0;
   sum = 0;
   DO WHILE value[i] <> -999 AND total.input < 100   3
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum   6
5        THEN increment total.valid by 1;
7                sum = s sum + value[i]
         ELSE skip
8     ENDIF
      increment i by 1;
9  ENDDO
   IF total.valid > 0   10
11    THEN average = sum / total.valid;
12    ELSE average = -999;
13 ENDIF
END average
```



آزمون مسیر پایه ...

❖ **مثال ...**



مرحله ۱- رسم گراف جریان

مرحله ۲- محاسبه پیچیدگی سیکلوماتیک
V(G)=6

مرحله ۳- تعیین مجموعه مسیر های پایه

Path 1: 1-2-10-11-13

Path 2: 1-2-10-12-13

Path 3: 1-2-3-10-11-13

Path 4: 1-2-3-4-5-8-9-2- ...

Path 5: 1-2-3-4-5-6-8-9-2- ...

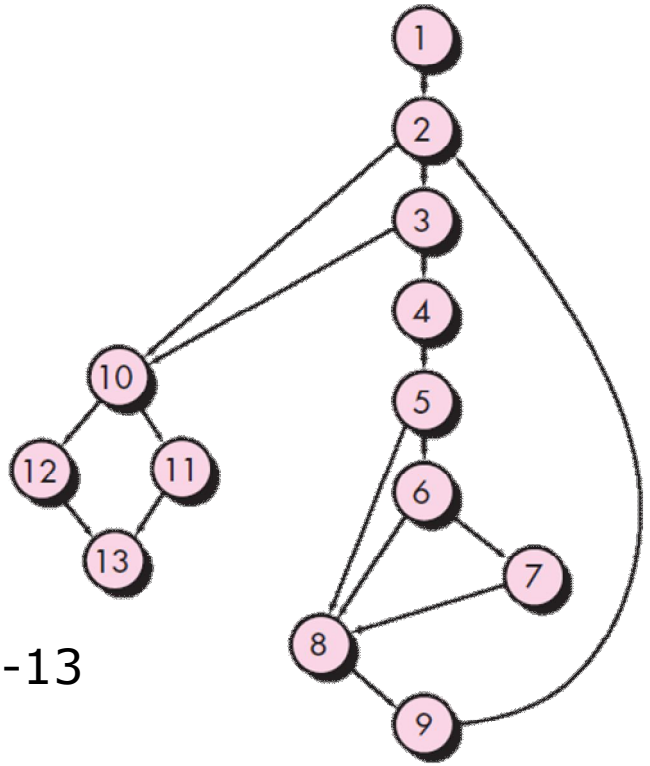
Path 6: 1-2-3-4-5-6-7-8-9-2- ...

✓ سه نقطه (...) در مسیر های ۴، ۵ و ۶ بیانگر این است که هر مسیری (مبتنی بر ساختار کنترلی) می تواند در ادامه ی آن نوشته شود.

آزمون مسیر پایه ...

❖ **مثال ...**

مرحله ۳- تعیین مجموعه مسیر های پایه ...



Path 1: 1-2-10-11-13

Path 2: 1-2-10-12-13

Path 3: 1-2-3-10-11-13

Path 4: 1-2-3-4-5-8-9-2-10-12-13

Path 5: 1-2-3-4-5-6-8-9-2-10-12-13

Path 6: 1-2-3-4-5-6-7-8-9-...-2-3-10-11-13

100 بار

آزمون مسیر پایه ...

❖ **مثال ...**

مرحله ۴- طراحی موارد آزمون

✓ داده های آزمون را باید به گونه ای انتخاب کرد که به موازات آزمون هر مسیر، شرایط لازم در گره های گزاره ای به طور مناسب برقرار باشد.

✓ برخی از مسیر های مستقل را نمی توان به شیوه مستقل تست کرد.

- به عبارت دیگر، ترکیب داده های لازم برای عبور از مسیر را نمی توان در جریان عادی برنامه بدست آورد. بنابراین مسیر به صورت مستقل قابل دسترس نیست.
- این مسیر ها معمولا به عنوان بخشی از یک مسیر دیگر، مورد آزمون قرار می گیرند.
- مانند مسیر ۱ و مسیر ۳

آزمون مسیر پایه ...

❖ **مثال ...**

مرحله ۴- طراحی موارد آزمون ...

	دستورات													شرط ها				
	1	2	3	4	5	6	7	8	9	10	11	12	13	2	3	5	6	10
Path 1	✓	✓								✓	✓		✓	F				T
Path 2	✓	✓								✓		✓	✓	F				F
Path 3	✓	✓	✓							✓	✓		✓	T	F			T
Path 4	✓	✓ ✓	✓	✓	✓			✓	✓	✓		✓	✓	T F	T	F		F
Path 5	✓	✓ ✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	T F	T	T	F	F
Path 6	✓	✓ ۱۰۱ بار	✓ ۱۰۱ بار	✓ ۱۰۰ بار	✓ ۱۰۰ بار	✓ ۱۰۰ بار	✓ ۱۰۰ بار	✓ ۱۰۰ بار	✓ ۱۰۰ بار	✓	✓		✓	T ۱۰۱ بار	T ۱۰۰ بار F	T ۱۰۰ بار	T ۱۰۰ بار	T

❖ مثال

شبه کد

محاسبه میانگین

```
PROCEDURE average;

   * This procedure computes the average of 100 or fewer
     numbers that lie between bounding values; it also computes the
     sum and the total number valid.

   INTERFACE RETURNS average, total.input, total.valid;
   INTERFACE ACCEPTS value, minimum, maximum;

   TYPE value[1:100] IS SCALAR ARRAY;
   TYPE average, total.input, total.valid;
      minimum, maximum, sum IS SCALAR;
   TYPE i IS INTEGER;
1  i = 1;
   total.input = total.valid = 0;
   sum = 0;
   DO WHILE value[i] <> -999 AND total.input < 100   3
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum   6
5        THEN increment total.valid by 1;
7                 sum = s sum + value[i]
         ELSE skip
8     ENDIF
      increment i by 1;
9  ENDDO
   IF total.valid > 0   10
11    THEN average = sum / total.valid;
12    ELSE average = -999;
13 ENDIF
END average
```

Path 1: 1-2-10-11-13

Path 3: 1-2-3-10-11-13

آزمون مسیر پایه ...

❖ **مثال ...**

مرحله ۴- طراحی موارد آزمون ...

	دستورات													شرط ها				
	1	2	3	4	5	6	7	8	9	10	11	12	13	2	3	5	6	10
Path 1	✓	✓								✓	✓		✓	F				T
Path 2	✓	✓								✓		✓	✓	F				F
Path 3	✓	✓	✓							✓	✓		✓	T	F			T
Path 4	✓	✓ ✓	✓	✓	✓			✓	✓	✓		✓	✓	T F	T	F		F
Path 5	✓	✓ ✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	T F	T	T	F	F
Path 6	✓	✓ ۱۰۱ بار	✓ ۱۰۱ بار	✓ ۱۰۰ بار	✓ ۱۰۰ بار	✓ ۱۰۰ بار	✓ ۱۰۰ بار	✓ ۱۰۰ بار	✓ ۱۰۰ بار	✓	✓		✓	T ۱۰۱ بار	T ۱۰۰ بار F	T ۱۰۰ بار	T ۱۰۰ بار	T

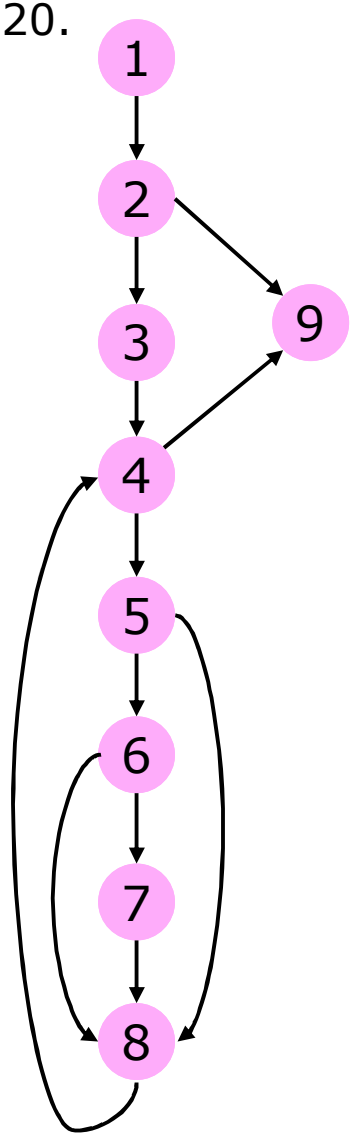
آزمون مسیر پایه ...

❖ **مثال ...** **مرحله ۴- طراحی موارد آزمون ...**

	داده های آزمون	نتیجه مورد انتظار	توضیح
مورد آزمون 1	value[1]=-999 minimum=0 maximum=20	average=-999 total.input=0 total.valid=0	پوشش مسیر ۲
مورد آزمون 2	value[1]=-1 value[2]=-999 minimum=0 maximum=20	average=-999 total.input=1 total.valid=0	پوشش مسیر ۴
مورد آزمون 3	value[1]=21 value[2]=-999 minimum=0 maximum=20	average=-999 total.input=1 total.valid=0	پوشش مسیر ۵
مورد آزمون 4	value[i]=1 (1<= i <=100) minimum=0 maximum=20	average=1 total.input=100 total.valid=100	پوشش مسیر ۶

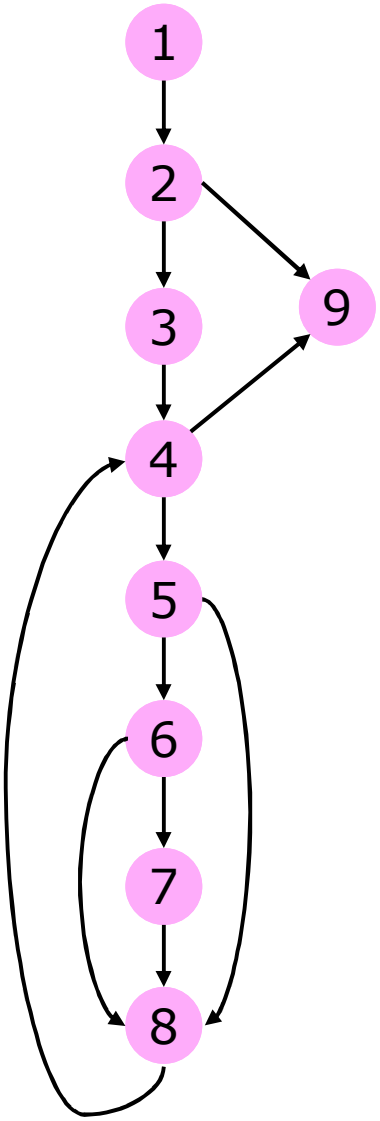
❖ مثال: طراحی موارد آزمون مسیر پایه برای تابع FindMax

```
/* Compute max of numItems positive numbers in the array
*  @param numItems how many items to find max, maximum of 20.
*/
public int FindMax(int [] numbers, int numItems) {
1   int max= -1;
2   if (numItems <= 20)
    {
        for (int count=0; count < numItems; count = count + 1)
        {
            if (numbers[count] > 0 && numbers[count] > max) {
                max = numbers[count];
            }
        }
    }
9   return max;
}
```



گراف جریان کنترل

آزمون مسیر پایه ...



گراف جریان کنترل

❖ **مثال ...**

- محاسبه پیچیدگی سیکلوماتیک

V(G) = 5

- تعیین مجموعه مسیر های پایه

Path 1: 1-2-9

Path 2: 1-2-3-4-9

Path 3: 1-2-3-4-5-6-7-8-4-9

Path 4: 1-2-3-4-5-8-4-9

Path 5: 1-2-3-4-5-6-7-8-4-5-6-8-4-9

❖ مثال: طراحی موارد آزمون مسیر پایه برای تابع FindMax

```
/* Compute max of numItems positive numbers in the array
*  @param numItems how many items to find max, maximum of 20.
*/
public int FindMax(int [] numbers, int numItems) {
1  int max= -1;
2  if (numItems <= 20)
   {
       for (int count=0; count < numItems; count = count + 1)
       {
            if (numbers[count] > 0 && numbers[count] > max) {
                 max = numbers[count];
             }
       }
    }
9  return max;
}
```

Is it Accessible? 1-2-3-4-5-6-8-4-9

آزمون مسیر پایه ...

❖ **مثال ...**

- طراحی موارد آزمون ...

مورد آزمون	توضیح
مورد آزمون ۱ داده های آزمون: numItems=21 , numbers={60} نتیجه مورد انتظار: max=-1	پوشش مسیر ۱
مورد آزمون ۲ داده های آزمون: numItems=0 , numbers={60} نتیجه مورد انتظار: max=-1	پوشش مسیر ۲
مورد آزمون ۳ داده های آزمون: numItems=1 , numbers={60} نتیجه مورد انتظار: max=60	پوشش مسیر ۳
مورد آزمون ۴ داده های آزمون: numItems=1 , numbers={-60} نتیجه مورد انتظار: max=-1	پوشش مسیر ۴
مورد آزمون ۵ داده های آزمون: numItems=2 , numbers={60,30} نتیجه مورد انتظار: max=60	پوشش مسیر ۵

آزمون شرط (Condition Testing)

- بررسی هر یک از شرط های منطقی در پیمانه
- خطا در مجاورت شرایط منطقی نسبت به دستورات پردازش ترتیبی رایج تر است.
- انواع خطا در شرط ها
  - خطا در عملگر های بولین
  - خطا در متغیر های بولین
  - خطا در پرانتز ها
  - خطا در عملگر های رابطه ای
  - خطا در عبارات محاسباتی

آزمون شرط ...

- در آزمون شرط، موارد آزمون به گونه ای طراحی می شوند تا هر حالت ممکن برای شرایط، حداقل یک بار بررسی و اجرا شود.

- **مثال**

شرط	حالات ممکن
E1 == E2	E1 == E2 E1 != E2
E1 < E2	E1 < E2 E1 == E2 E1 > E2

❖ مثال: طراحی موارد آزمون شرط برای تابع FindMax

```
/* Compute max of numItems positive numbers in the array
 *  @param numItems how many items to find max, maximum of 20.
 */
public int FindMax(int [] numbers, int numItems) {
    int max= -1;
    if (numItems <= 20)
   {
          for (int count=0; count < numItems; count = count + 1)
          {
               if (numbers[count] > 0 && numbers[count] > max) {
                    max = numbers[count];
               }
          }
    }
    return max;
}
```

آزمون شرط ...

شرط numItems <= 20	مورد آزمون
numItems < 20	**مورد آزمون ۱** داده های آزمون: numbers={60} , numItems=1 نتیجه مورد انتظار: max=60
numItems == 20	**مورد آزمون ۲** داده های آزمون: numbers[i]=i+1 (0<= i <20) , numItems=20 نتیجه مورد انتظار: max=20
numItems > 20	**مورد آزمون ۳** داده های آزمون: numbers[i]=i+1 (0<= i <21) , numItems=21 نتیجه مورد انتظار: max=-1

شرط count < numItems	مورد آزمون
count < numItems	**مورد آزمون ۱**
count == numItems	**مورد آزمون ۱**
count > numItems	**مورد آزمون ۴** داده های آزمون: numbers={} , numItems= -1 نتیجه مورد انتظار: max=-1

آزمون شرط ...

شرط numbers[count] > 0 && numbers[count] > max	مورد آزمون
numbers[count] > 0 && numbers[count] > max	**مورد آزمون ۱**
numbers[count] > 0 && numbers[count] == max	**مورد آزمون ۵** داده های آزمون: numbers={60,60} numItems=2 نتیجه مورد انتظار: max=60
numbers[count] > 0 && numbers[count] < max	**مورد آزمون ۶** داده های آزمون: numbers={60,30} numItems=2 نتیجه مورد انتظار: max=60
numbers[count] == 0 && numbers[count] > max	**مورد آزمون ۷** داده های آزمون: numbers={0} numItems=1 نتیجه مورد انتظار: max=-1
numbers[count] == 0 && numbers[count] == max	-----

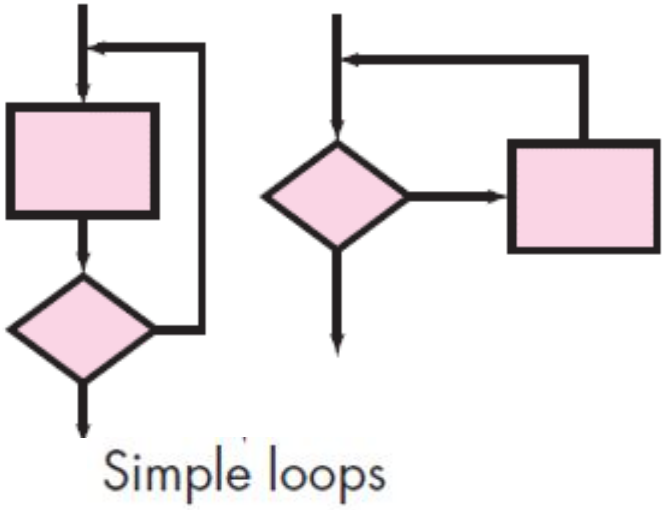
آزمون شرط ...

شرط numbers[count] > 0 && numbers[count] > max	مورد آزمون
numbers[count] == 0 && numbers[count] < max	**مورد آزمون ۸** داده های آزمون: numbers={60,0} numItems=2 نتیجه مورد انتظار: max=60
numbers[count] < 0 && numbers[count] > max	-----
numbers[count] < 0 && numbers[count] == max	**مورد آزمون ۹** داده های آزمون: numbers={-1} numItems=1 نتیجه مورد انتظار: max=-1
numbers[count] < 0 && numbers[count] < max	**مورد آزمون ۱۰** داده های آزمون: numbers={-60} numItems=1 نتیجه مورد انتظار: max=-1

آزمون حلقه (Loop Testing)

- بررسی صحت هر یک از حلقه ها در پیمانه
- انواع حلقه ها
  - حلقه های ساده
  - حلقه های تو در تو
  - حلقه های متصل
  - حلقه های غیر ساخت یافته

آزمون حلقه ...



❖ آزمون حلقه های ساده

- عدم اجرای حلقه (صفر بار گذر)
- یک بار گذر از حلقه
- دو بار گذر از حلقه
- m بار گذر از حلقه (m<n)
- n-1، n و n+1 گذر از حلقه

✓ n : حداکثر تعداد گذر های مجاز از میان حلقه

✓به ازای مورد آزمون n+1، حلقه نباید قادر به اجرا باشد و یا در صورت اجرا، برنامه باید در یکی از دستورات متوقف (break) شود.

❖ مثال : طراحی موارد آزمون حلقه برای تابع FindMax

```
/* Compute max of numItems positive numbers in the array
 *  @param numItems how many items to find max, maximum of 20.
 */
public int FindMax(int [] numbers, int numItems) {
    int max= -1;
    if (numItems <= 20)
   {
          for (int count=0; count < numItems; count = count + 1)
          {
               if (numbers[count] > 0 && numbers[count] > max) {
                     max = numbers[count];
               }
          }
    }
    return max;
}
```

آزمون حلقه ...

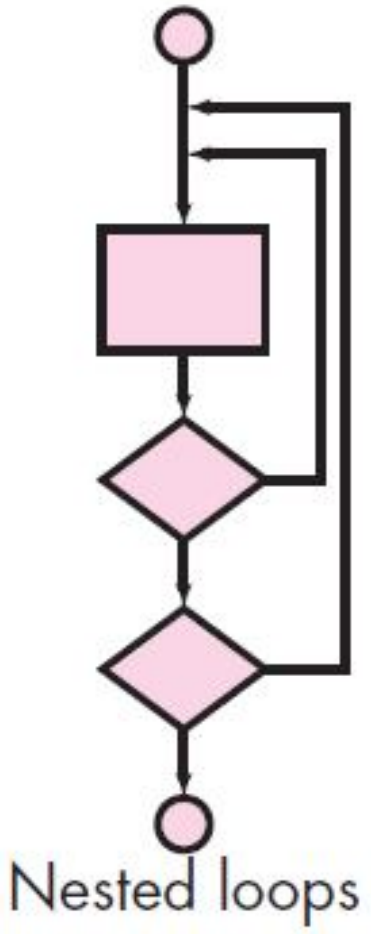
مورد آزمون	توضیح
مورد آزمون ۱ داده های آزمون: numItems=0 , numbers={} نتیجه مورد انتظار: max=-1	عدم اجرای حلقه
مورد آزمون ۲ داده های آزمون: numItems=1 , numbers={60} نتیجه مورد انتظار: max=60	یک بار اجرای حلقه
مورد آزمون ۳ داده های آزمون: numbers={60,30} , numItems=2 نتیجه مورد انتظار: max=60	دو بار اجرای حلقه
مورد آزمون ۴ داده های آزمون: numbers={5,-3,8,-12,4,0,-98,6,-2,1} , numItems=10 نتیجه مورد انتظار: max=8	m<n بار اجرای حلقه

آزمون حلقه ...

مورد آزمون	توضیح
مورد آزمون ۵ داده های آزمون: numbers[i]=60 (0<= i <19) , numItems=19 نتیجه مورد انتظار: max=60	n-1 بار اجرای حلقه
مورد آزمون ۶ داده های آزمون: numbers[i]=i+1 (0<= i <20) , numItems=20 نتیجه مورد انتظار: max=20	n بار اجرای حلقه
مورد آزمون ۷ داده های آزمون: numbers[i]=i+1 (0<= i <21) , numItems=21 نتیجه مورد انتظار: max=-1	n+1 بار اجرای حلقه

آزمون حلقه ...

❖ آزمون حلقه های تو در تو

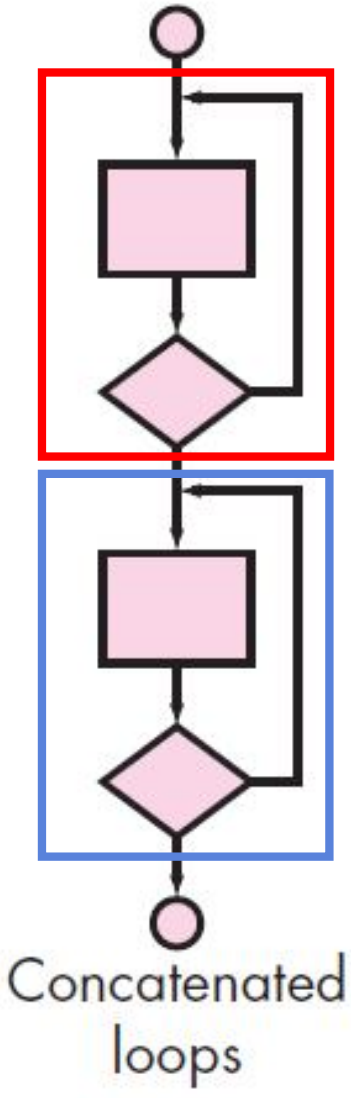


۱. آزمون حلقه ساده را برای داخلی ترین حلقه اجرا کنید در حالی که تعداد تکرار سایر حلقه ها در حداقل مقدار غیر صفر ممکن قرار دارد.

۲. به سمت بیرون حلقه ها حرکت کرده و همین روند را برای حلقه بعدی تکرار کنید؛ یعنی آزمون حلقه ساده را برای حلقه بعدی اجرا کنید در حالی که حلقه های بیرونی به ازای حداقل مقدار غیر صفر ممکن و حلقه های داخلی به ازای یکی از مقادیر معمولی شمارنده تکرار می شوند.

۳. مرحله ۲ را تا آزمون همه حلقه ها ادامه دهید.

آزمون حلقه ...



❖ آزمون حلقه های متصل

- **حلقه های مستقل از هم**
  - هر یک از حلقه ها مستقل از دیگری است.
    - شمارنده حلقه دوم به حلقه اول وابسته نیست.
  - استفاده از روش آزمون حلقه های ساده
- **حلقه های وابسته به هم**
  - مقدار شمارنده حلقه دوم به حلقه اول وابسته است.
  - روش آزمون

۱. آزمون حلقه ساده را برای آخرین (پایین ترین) حلقه اجرا کنید در حالی که تعداد تکرار حلقه های قبلی (حلقه های بالاتر) در کمترین مقدار ممکن قرار دارد.

۲. به سمت بالا حرکت کنید و همین روند را برای حلقه های بالاتر اجرا کنید (تعداد تکرار حلقه های بالایی را در کمترین مقدار ممکن و تعداد تکرار حلقه های پایینی را در یک مقدار معمولی تنظیم کنید).

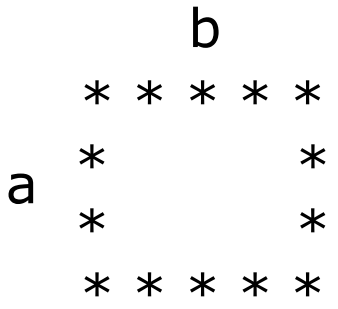
آزمون حلقه ...

❖ **مثال: آزمون حلقه های تو در تو و متصل (مستقل از هم)**

```
cin >> a >> b;
if (a>=2 && a<=100 && b>=1 && b<=100)
{
  1 for (j = 1;j <= b;j++)
        cout << '*' << ' ';
    cout << '\n';
  2 for (i = 2;i <= a - 1;i++)
    {
        cout << '*';
      3 for (j = 1;j <= b - 2;j++)
            cout << ' ' << ' ';
        cout << ' ' << '*' << '\n';
    }
  4 for (j = 1;j <= b;j++)
        cout << '*' << ' ';
    cout << '\n';
}
```

حلقه های متصل و مستقل از هم

حلقه های تو در تو



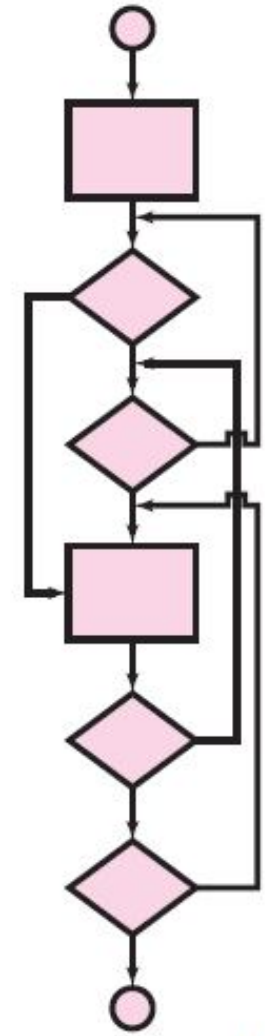
آزمون حلقه ...

❖ **مثال ...**

	داده های آزمون		تعداد تکرار حلقه ها			
	a	b	حلقه 1 b بار	حلقه 2 a-2 بار	حلقه 3 b-2 بار	حلقه 4 b بار
مورد آزمون 1	3	2	2	1	0	2
مورد آزمون 2	3	3	3	1	1	3
مورد آزمون 3	3	4	4	1	2	4
مورد آزمون 4	3	50	50	1	48	50
مورد آزمون 5	3	99	99	1	97	99
مورد آزمون 6	3	100	100	1	98	100
مورد آزمون 7	3	101	0	1	0	0
مورد آزمون 8	4					
مورد آزمون 9	50					
مورد آزمون 10	99					
مورد آزمون 11	00					
مورد آزمون 12	101					
مورد آزمون 13	2					

```
2  for (i = 2;i <= a - 1;i++)

1  4   for (j = 1;j <= b;j++)
                cout << '*' << ' ';

          cout <<      <<  <<  << '\n';
    }
```

آزمون حلقه ...



Unstructured loops

❖ آزمون حلقه های غیر ساخت یافته

- حلقه غیر ساخت یافته: استفاده از دستورات goto، break، continue، exit و ... در ساختار حلقه
- آزمون حلقه های غیر ساخت یافته به صورت موثر امکان پذیر نیست.
- این نوع حلقه ها معمولا بازآرایی (refactoring) شده و به حلقه های ساخت یافته (ساده، تودرتو و متصل) تبدیل می شوند.

آزمون جریان داده ها (Data Flow Testing)

❖ مسیر های لازم برای آزمون، مطابق با مکان تعریف و کاربرد متغیر ها در برنامه انتخاب می شود.

مرحله ۱ – به هر دستور از برنامه یک شماره منحصر بفرد داده می شود.

مرحله ۲ – به ازای هر دستور با شماره S دو مجموعه تعریف می شود:

- **DEF(S)**
  - مجموعه متغیر های مقدار دهی شده توسط دستور S (*write*)

DEF(S) = {X | statement S contains a definition of X}

- **USE(S)**
  - مجموعه متغیر های استفاده شده در دستور S (*read*)

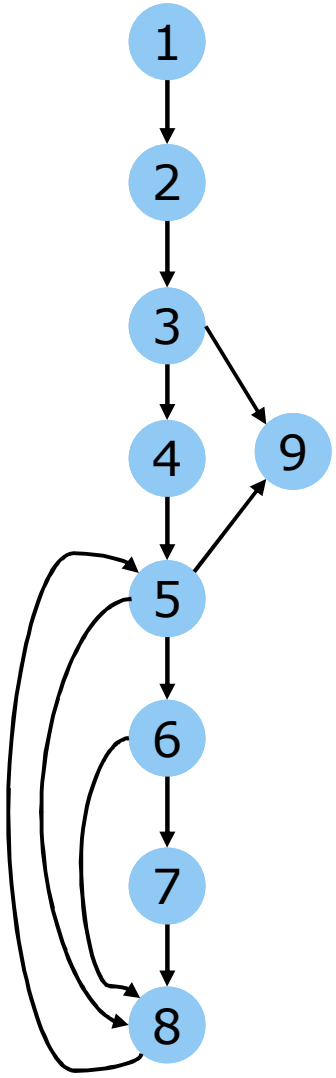
USE(S) = {X | statement S contains a use of X}

❖ مثال: آزمون جریان داده برای تابع FindMax

مرحله ۱

```
/* Compute max of numItems positive numbers in the array
 *  @param numItems how many items to find max, maximum of 20.
 */
1 public int FindMax(int [] numbers, int numItems) {
2     int max= -1;
3     if (numItems <= 20)
      {                          4          5                8
            for (int count=0; count < numItems; count = count + 1)
            {
             6 if (numbers[count] > 0 && numbers[count] > max) {
                      7 max = numbers[count];
                  }
            }
      }
9 return max;
}
```

آزمون جریان داده ها ...



گراف جریان داده

❖ **مثال ...**

مرحله ۲ – تعریف مجموعه های DEF و USE

DEF(1)={numbers,numItems}	USE(1)={}
DEF(2)={max}	USE(2)={}
DEF(3)={}	USE(3)={numItems}
DEF(4)={count}	USE(4)={}
DEF(5)={}	USE(5)={count,numItems}
DEF(6)={}	USE(6)={numbers,count,max}
DEF(7)={max}	USE(7)={numbers,count}
DEF(8)={count}	USE(8)={count}
DEF(9)={}	USE(9)={max}

آزمون جریان داده ها ...

❖ *تعریف متغیر X در دستور S، در دستور S' **زنده** است اگر مسیری از دستور S به دستور S' وجود داشته باشد که حاوی هیچ تعریف دیگری از X نباشد.*

- **مثال:** تعریف متغیر max در دستور ۲، در دستورات ۳، ۴، ۵ و ۶ زنده است اما در دستور ۷ زنده نیست.

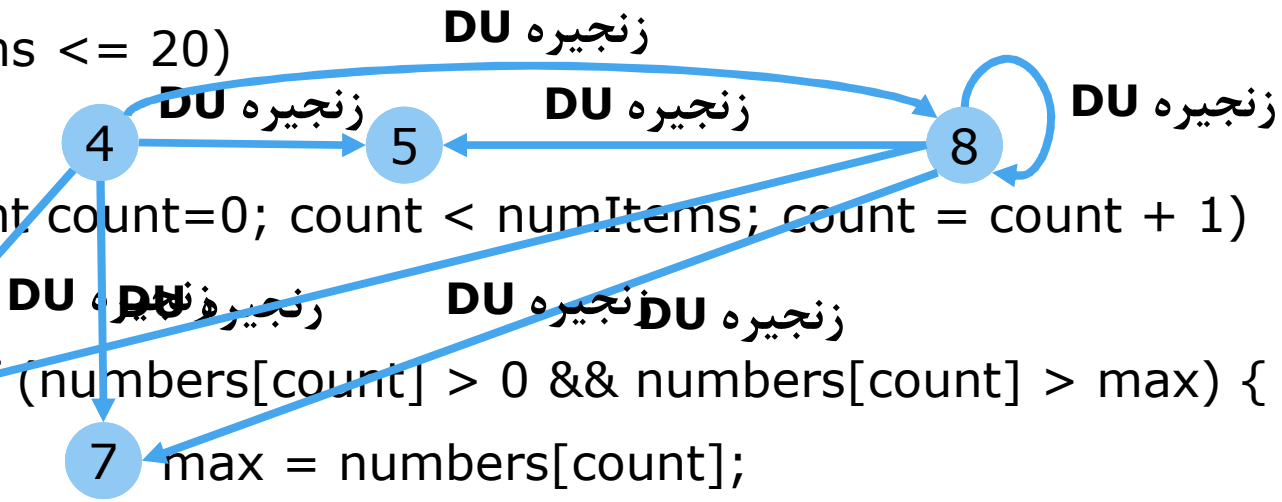
مرحله ۳- تعیین زنجیره تعریف-کاربرد (Definition-Use) برای متغیر ها

- یک زنجیره تعریف-کاربرد (DU) برای متغیر X به صورت [X,S,S'] تعیین می شود که در آن:
  - X عضو مجموعه DEF(S) است.
  - X عضو مجموعه USE(S') است.
  - تعریف X در دستور S، در دستور S' زنده است.

✓ به عبارت دیگر، در دستور S یک مقدار به X نسبت داده شده است و از همان مقدار در دستور S' استفاده شده است.

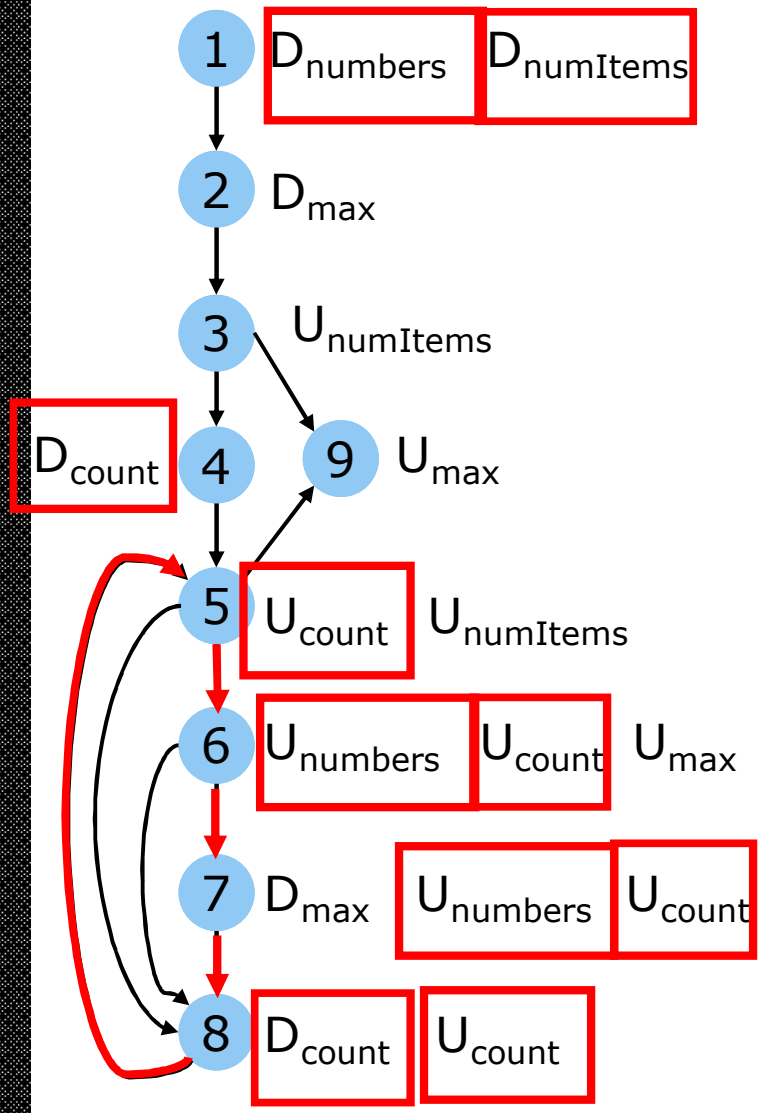
❖ مثال: آزمون جریان داده برای تابع FindMax

```
/* Compute max of numItems positive numbers in the array
 *  @param numItems how many items to find max, maximum of 20.
 */
1 public int FindMax(int [] numbers, int numItems) {
2     int max= -1;
3     if (numItems <= 20)
      {
            for (int count=0; count < numItems; count = count + 1)
            {
6               if (numbers[count] > 0 && numbers[count] > max) {
7                   max = numbers[count];
                }
            }
      }
9 return max;
}
```



آزمون جریان داده ها ...

❖ **مثال ...**



مرحله ۳ – تعیین زنجیره های DU

1. [numbers , 1 , 6]	9. [count , 4 , 5]
2. [numbers , 1 , 7]	10. [count , 4 , 6]
3. [numItems , 1 , 3]	11. [count , 4 , 7]
4. [numItems , 1 , 5]	12. [count , 4 , 8]
5. [max , 2 , 9]	13. [count , 8 , 5]
6. [max , 2 , 6]	14. [count , 8 , 6]
7. [max , 7 , 6]	15. [count , 8 , 7]
8. [max , 7 , 9]	16. [count , 8 , 8]

آزمون جریان داده ها ...

مرحله ۴ – طراحی موارد آزمون به گونه ای که هر زنجیره DU حداقل یک بار پوشش داده شود.

- ✓مسیر هایی را از گراف جریان داده انتخاب می کنیم که در نهایت مطمئن باشیم که از هر زنجیره DU حداقل یک بار عبور کرده ایم.
- ✓به این شیوه آزمون، آزمون DU نیز گفته می شود.

آزمون جریان داده ها ...

❖ **مثال ...**

مرحله ۴ – طراحی موارد آزمون

مورد آزمون
مورد آزمون ۱ داده های آزمون: numItems=21 , numbers={60} نتیجه مورد انتظار: max=-1
مورد آزمون ۲ داده های آزمون: numItems=0 , numbers={60} نتیجه مورد انتظار: max=-1
مورد آزمون ۳ داده های آزمون: numItems=1 , numbers={60} نتیجه مورد انتظار: max=60
مورد آزمون ۴ داده های آزمون: numItems=2 , numbers={60,30} نتیجه مورد انتظار: max=60
مورد آزمون ۵ داده های آزمون: numItems=20 , numbers[i]=i+1 (0<=i<20) نتیجه مورد انتظار: max=20

آزمون جریان داده ها ...

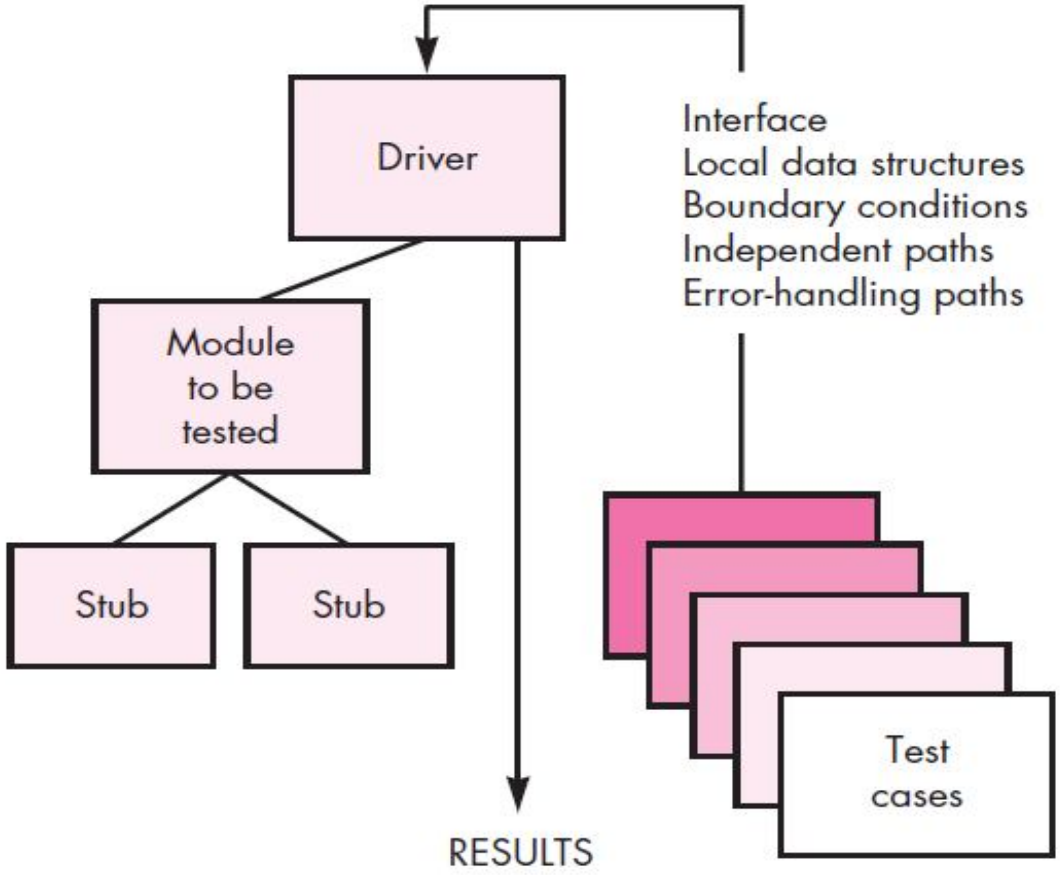
❖ **مثال ...**

✓ به ازای موارد آزمون طراحی شده هر یک از زنجیره های DU حداقل یک بار پوشش داده می شوند.

	زنجیره های تعریف - کاربرد (DU)															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
مورد آزمون ۱			✓		✓											
مورد آزمون ۲			✓	✓	✓				✓							
مورد آزمون ۳	✓	✓	✓	✓		✓		✓	✓	✓	✓	✓	✓			
مورد آزمون ۴	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
مورد آزمون ۵	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

نرم افزار های درایور و استاب

- پیمانه، یک برنامه مستقل نیست و در ارتباط با پیمانه های دیگر عمل می کند.
- در آزمون واحد (و آزمون انسجام)، گاهی اوقات نیاز است که یک برنامه درایور و/یا استاب ایجاد شود.



نرم افزار های درایور و استاب ...

❖ **درایور (Driver)**

- یک برنامه کنترلی برای اجرای آزمون
  - پذیرش داده های مورد آزمون
  - فراخوانی پیمانه ای که باید تست شود
  - نمایش و اعتبار سنجی نتایج حاصل از اجرای مورد آزمون
- درایور معمولا به صورت برنامه اصلی (main program) پیاده سازی می شود.
- در آزمون های انسجام، درایور ها ممکن است جایگزین پیمانه های سطح بالایی شوند که هنوز پیاده سازی یا تست نشده اند.

```
static void Main(string[] args) {
    int[] numbers = { 60, 30 };
    int numItems = 2;
    int expected = 60;
    int actual = FindMax(numbers, numItems);
    if (expected == actual)
        Console.WriteLine("Passed Test");
    else
        Console.WriteLine("Failed Test: actual={0}  expected={1}"
                                    ,actual, expected);
 }
```

نرم افزار های درایور و استاب ...

❖ استاب (Stub)

- پیمانه تحت آزمون ممکن است به پیمانه های دیگر وابسته باشد (برای مثال، برخی از توابع پیمانه های دیگر را فراخوانی کند).
- استاب یک برنامه ساختگی است که جایگزین پیمانه ای می شود که توسط پیمانه تحت آزمون لازم است فراخونی شود اما هنوز پیاده سازی یا تست نشده است.
- به عبارت دیگر استاب، پیمانه فراخوانی شده را شبیه سازی می کند.
- در آزمون های انسجام، استاب ها ممکن است جایگزین پیمانه های سطح پایینی شوند که هنوز پیاده سازی یا تست نشده اند.

نرم افزار های درایور و استاب ...

❖ مشکلات

- درایور ها و استاب ها ایجاد سربار می کنند.
  - هر دو برنامه هایی هستند که باید نوشته شوند (البته طراحی رسمی برای آنها اعمال نمی شود) ولی با محصول نهایی تحویل داده نمی شوند.
- می توان برای کاهش سربار، درایور ها و استاب ها را به صورت ساده پیاده سازی کرد.
  - اما بسیاری از نرم افزار ها را نمی توان به طور مناسب با نرم افزار های سربار ساده تست کرد. در چنین مواردی، آزمون واحد را می توان تا مرحله آزمون انسجام به تعویق انداخت.

آزمون انسجام در رویکرد سنتی

- در انسجام نرم افزار، پیمانه ها مطابق با طراحی معماری با هم ترکیب می شوند تا زمانی که نرم افزار به طور کامل ساخته شود.

- **آزمون انسجام**
  - یک آزمون سطح پایین است که بر طراحی و معماری نرم افزار تاکید دارد.
  - بررسی نحوه ترکیب پیمانه ها با یکدیگر و اثرات جانبی آن
  - کشف خطاهای ممکن در روابط و واسط میان پیمانه های ترکیب شده

- **دلایل نیاز به آزمون انسجام**
  - ممکن است داده ها در گذر از یک واسط از بین بروند.
  - یک پیمانه می تواند اثر وارونه بر پیمانه دیگر داشته باشد.
  - پس از ترکیب پیمانه ها، عملکرد های فرعی ممکن است مانع از اجرای مطلوب عملکرد اصلی شوند.
  - ساختمان داده های سراسری ممکن است باعث ایجاد مشکلات شود.

آزمون انسجام در رویکرد سنتی ...

❖ **انواع روش های انسجام**

- انسجام غیر افزایشی
- انسجام افزایشی
  - انسجام بالا به پایین
    - انسجام عمقی
    - انسجام عرضی
  - انسجام پایین به بالا
  - انسجام ترکیبی (آزمون ساندویچ)

✓ انتخاب شیوه انسجام به ویژگی های نرم افزار و زمان بندی پروژه بستگی دارد.

✓ در زمان بندی پروژه باید شیوه انسجام در نظر گرفته شود تا پیمانه ها در صورت نیاز در دسترس باشند.

آزمون انسجام در رویکرد سنتی ...

❖ **انسجام غیر افزایشی**

- به این روش، انفجار بزرگ (big bang approach) نیز گفته می شود.
- ابتدا همه پیمانه ها با هم ترکیب (منسجم) می شوند. سپس کل نرم افزار (همه پیمانه ها) به صورت یکجا تست می شود.

- **مشکلات**
  - معمولا بی نظمی ایجاد می شود.
  - با مجموعه زیادی از خطاها روبرو می شود.
  - جداسازی علت ها و تصحیح خطاها در کل برنامه، پیچیده و دشوار است.

آزمون انسجام در رویکرد سنتی ...

❖ انسجام افزایشی (Incremental Integration)

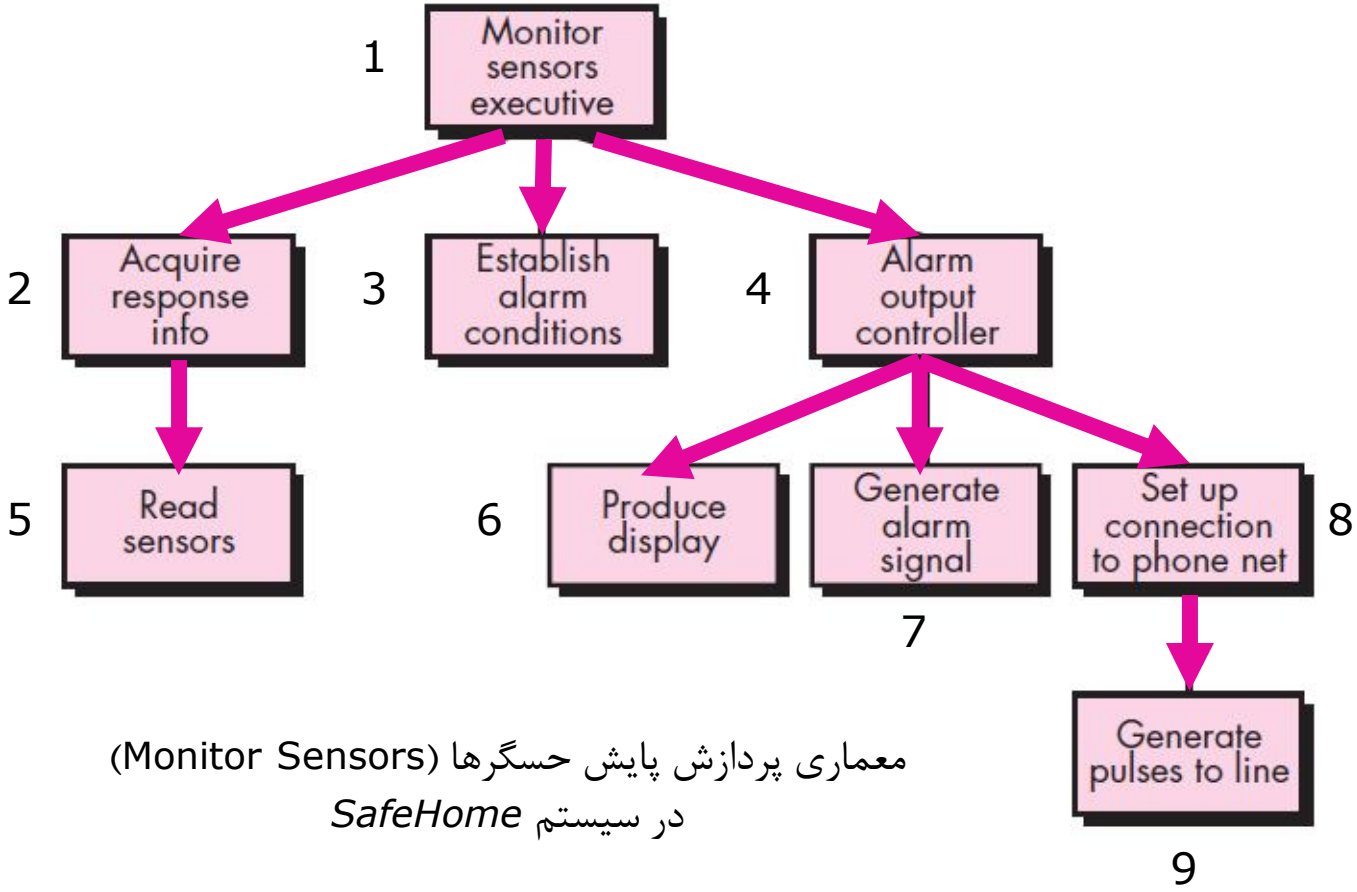
- پیمانه ها به صورت گام به گام (تدریجی) با هم ترکیب می شوند و در هر گام، روابط و واسط میان پیمانه های ترکیب شده تست می شود.

- **مزایا**
  - جدا کردن خطاها و تصحیح آنها آسان تر است.
  - واسط ها با احتمال بیشتری می توانند به طور کامل تست شوند.

- **روش های انسجام افزایشی**
  - انسجام بالا به پایین
  - انسجام پایین به بالا
  - انسجام ترکیبی (آزمون ساندویچ)

انسجام بالا به پایین (Top-Down Integration)

- پیمانه ها، در راستای سلسله مراتب کنترلی، با شروع از پیمانه اصلی (ریشه) و با حرکت به طرف پایین با هم ترکیب می شوند.
  - پیمانه های سطح بالاتر زودتر از پیمانه های دیگر تست شده و با هم ترکیب می شوند.



معماری پردازش پایش حسگرها (Monitor Sensors)
در سیستم *SafeHome*

انسجام بالا به پایین ...

- روش های انسجام بالا به پایین
  - انسجام عمقی (Depth-first Integration)
    - انسجام در راستای یک مسیر کنترلی و با حرکت در عمق انجام می شود.
    - **مثال:** M1 - M2 - M5 - M3 - M4 - M6 - M7 - M8 - M9
  - انسجام عرضی (Breadth-first Integration)
    - انسجام سطح به سطح و با حرکت افقی در سطوح انجام می شود.
    - **مثال:** M1 - M2 - M3 - M4 - M5 - M6 - M7 - M8 - M9

انسجام بالا به پایین ...

❖ مراحل

۱- پیمانه اصلی به عنوان درایور در نظر گرفته می شود. استاب ها جایگزین پیمانه هایی می شوند که مستقیما زیر دست (subordinate) پیمانه اصلی هستند.

۲- با توجه به روش انسجام انتخاب شده (عمقی یا عرضی)، استاب های زیر دست به نوبت جای خود را به پیمانه های واقعی (پیاده سازی شده) می دهند.

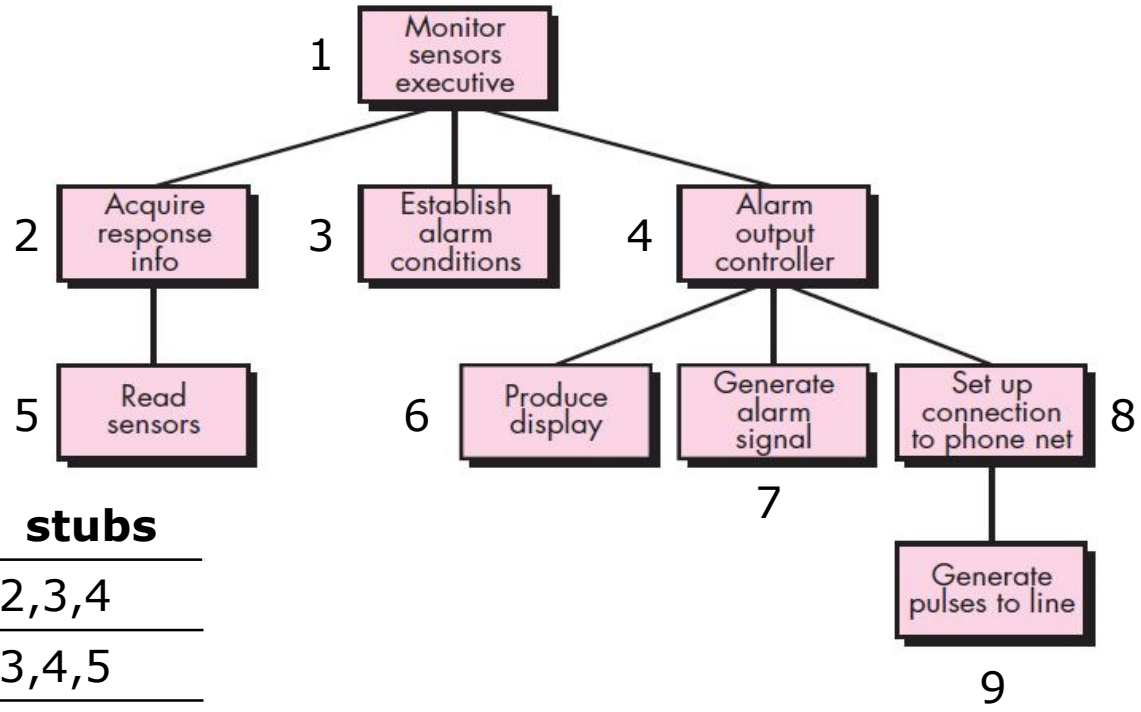
۳- همزمان با انسجام هر پیمانه جدید، روابط و واسط میان پیمانه های ترکیب شده تست می شود.

۴- در زمان انسجام، می توان با اجرای آزمون رگرسیون اطمینان حاصل کرد که خطاهای جدید وارد پیمانه ها نشده اند.

۵- مراحل ۲ تا ۴ تا زمانی که کل ساختار برنامه ساخته شود، تکرار می شود.

انسجام بالا به پایین ...

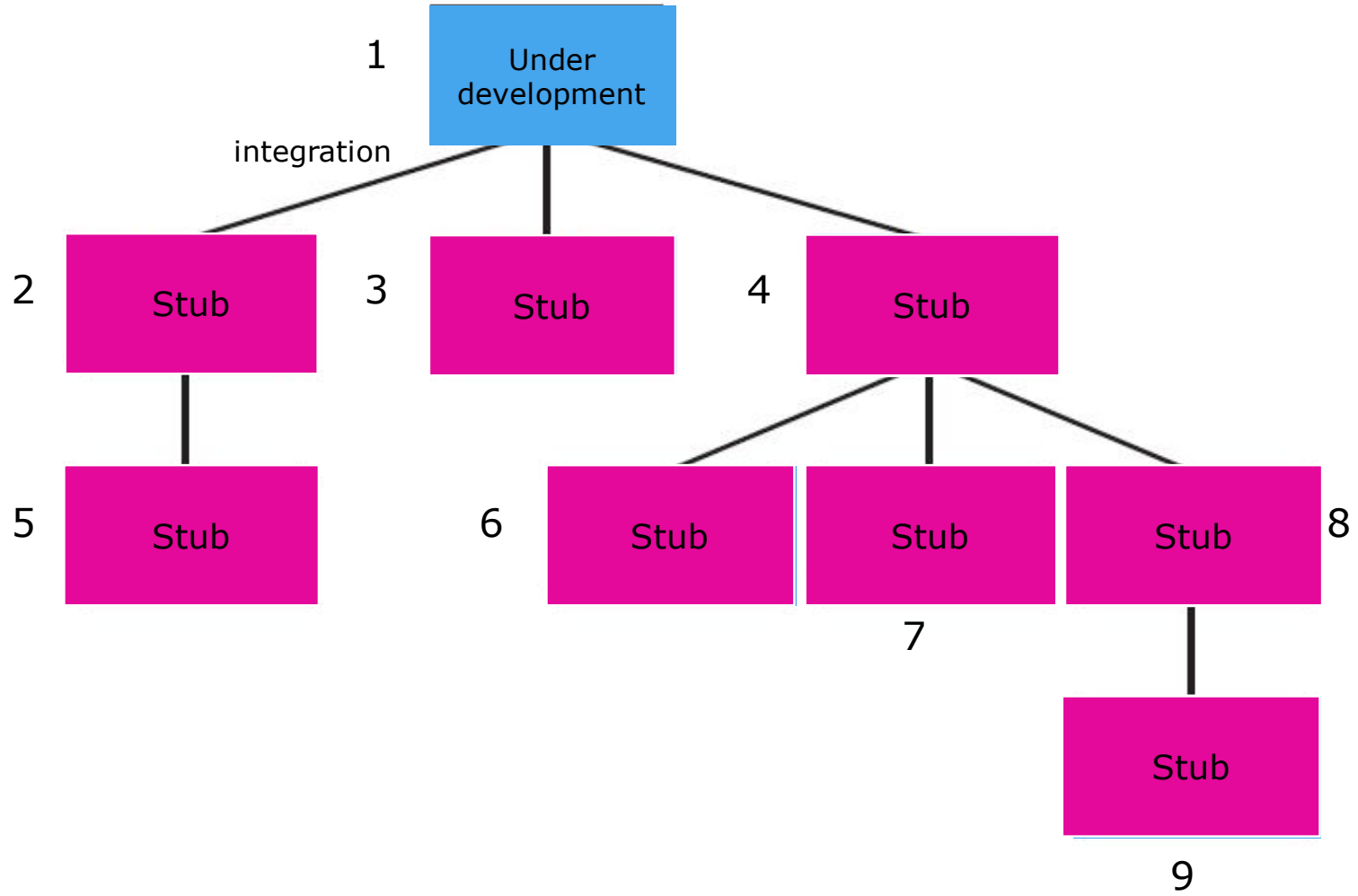
❖ **مثال:** انسجام بالا به پایین با رویکرد سطحی



step	integration	stubs
1	1	2,3,4
2	1,2	3,4,5
3	1,2,3	4,5
4	1,2,3,4	5,6,7,8
5	1,2,3,4,5	6,7,8
6	1,2,3,4,5,6	7,8
7	1,2,3,4,5,6,7	8
8	1,2,3,4,5,6,7,8	9
9	1,2,3,4,5,6,7,8,9	-

انسجام بالا به پایین ...

❖ **مثال ...**



انسجام بالا به پایین ...

❖ **مزایا**

- نقاط کنترلی، تصمیم گیری ها و عملکرد های اصلی در ابتدای فرآیند آزمون بررسی می شوند.
- در صورت استفاده از انسجام عمقی، این امکان وجود دارد که یک قابلیت عملیاتی کامل از نرم افزار پیاده سازی شده و تست شود و سپس به مشتری ارائه شود. این امر موجب رضایت مشتری و سازنده می شود.
- کاهش تعداد درایورها

انسجام بالا به پایین ...

❖ مشکلات

- نیاز به استاب های زیاد
- پیچیده تر شدن استاب ها و موارد آزمون

❖ راهکار ها

- به تاخیر انداختن آزمون ها تا زمانی که استاب ها جای خود را به پیمانه های واقعی می دهند.
  - دشوار شدن تعیین علت خطاها و تغییر ماهیت روش انسجام بالا به پایین
- توسعه استاب هایی که با اجرای عملیاتی محدود، پیمانه واقعی را شبیه سازی می کنند.
  - این کار عملی است اما سربار بالایی دارد زیرا استاب ها پیچیده می شوند.
- انسجام پایین به بالا

انسجام پایین به بالا (Bottom-Up Integration)

- ساخت و آزمون پیمانه ها با پیمانه های ساده تر (که در پایین ترین سطح سلسله مراتب ساختار برنامه قرار دارند) آغاز می شود.
  - پیمانه های سطوح پایین زودتر از پیمانه های دیگر تست شده و با هم ترکیب می شوند.

- **مراحل**

۱- پیمانه های سطح پایین به صورت خوشه (cluster) هایی با هم ترکیب می شوند. این پیمانه ها یک عملکرد فرعی از نرم افزار را انجام می دهند.

۲- به ازای هر خوشه، یک درایور نوشته می شود تا ورودی و خروجی موارد آزمون را هماهنگ کند.

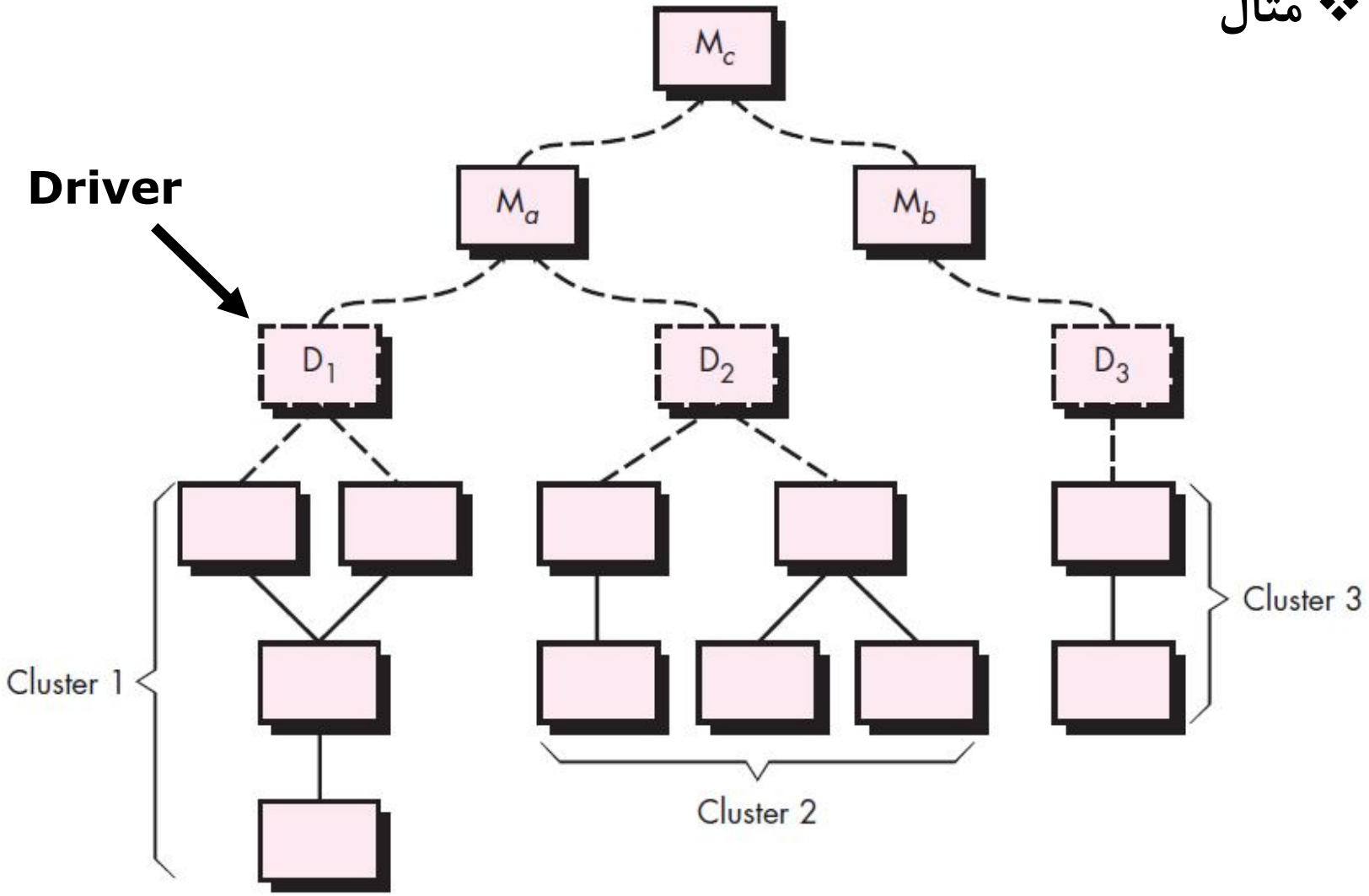
۳- خوشه مورد آزمون قرار می گیرد.

۴- درایور ها حذف می شوند و خوشه ها در حرکت به طرف بالای ساختار برنامه با یکدیگر ترکیب می شوند.

✓ هر چه بیشتر به سمت بالای ساختار حرکت کنیم، نیاز به درایور های جداگانه برای آزمون کمتر می شود.

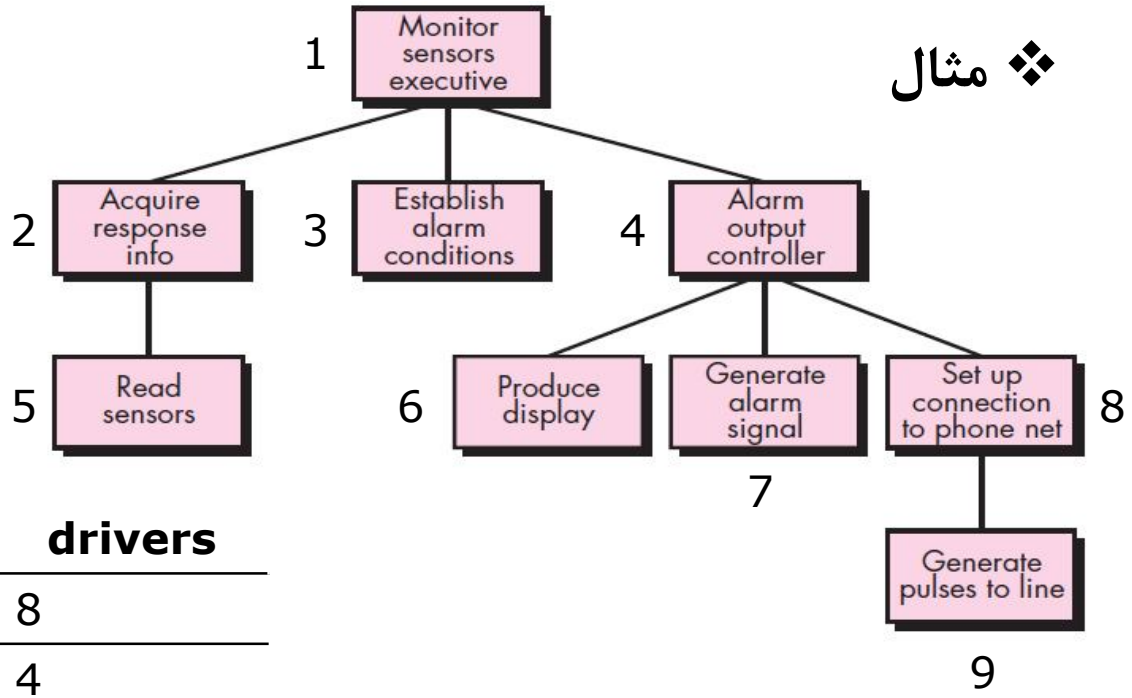
انسجام پایین به بالا ...

❖ **مثال**



انسجام پایین به بالا ...

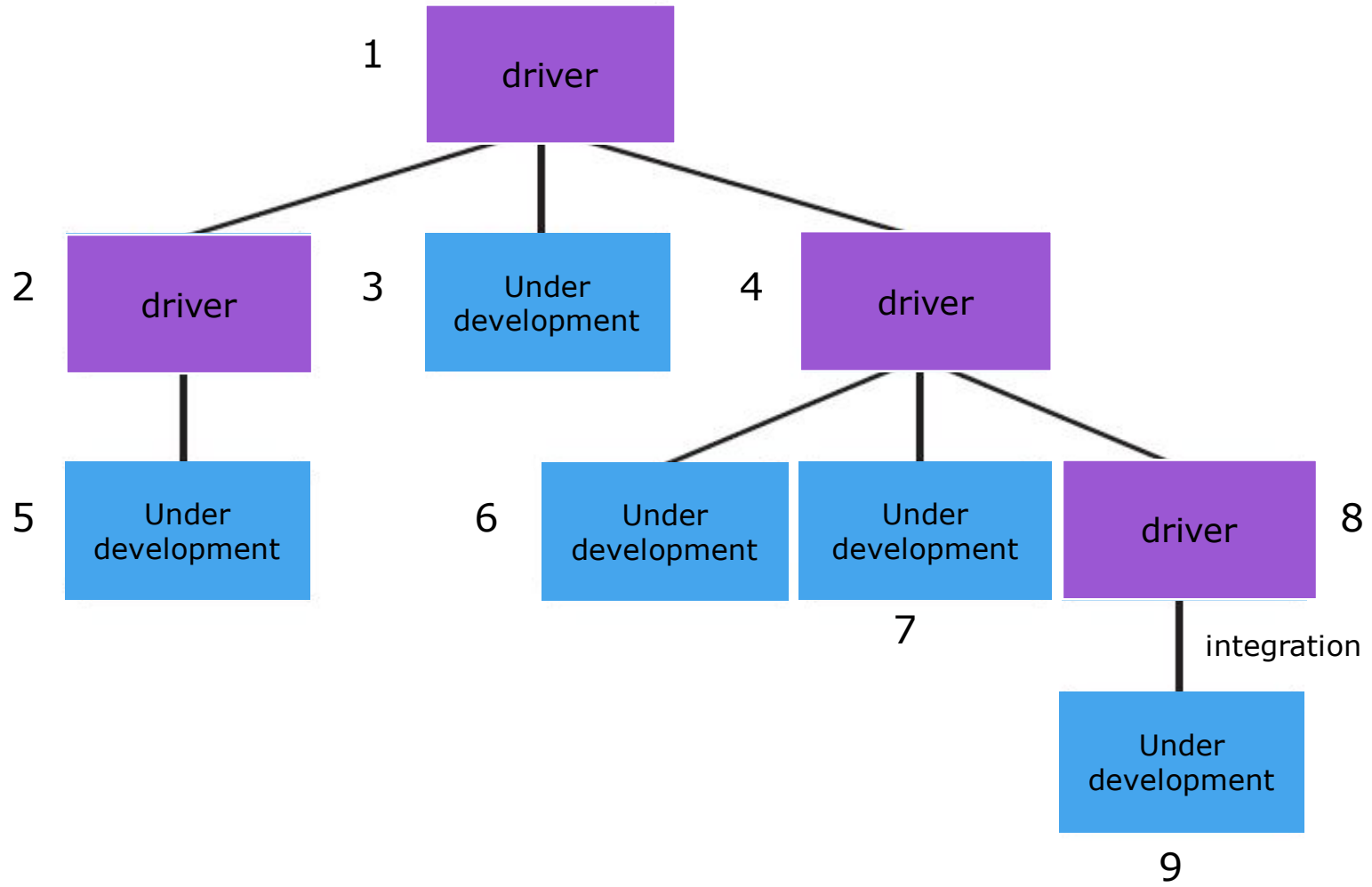
❖ **مثال**



step	integration	drivers
1	9	8
2	8,9	4
3	6	4
4	7	4
5	5	2
6	2,5	1
7	3	1
8	4,6,7,8,9	1
9	1,2,3,4,5,6,7,8,9	-

انسجام پایین به بالا ...

❖ **مثال ...**



انسجام پایین به بالا ...

❖ مشکلات

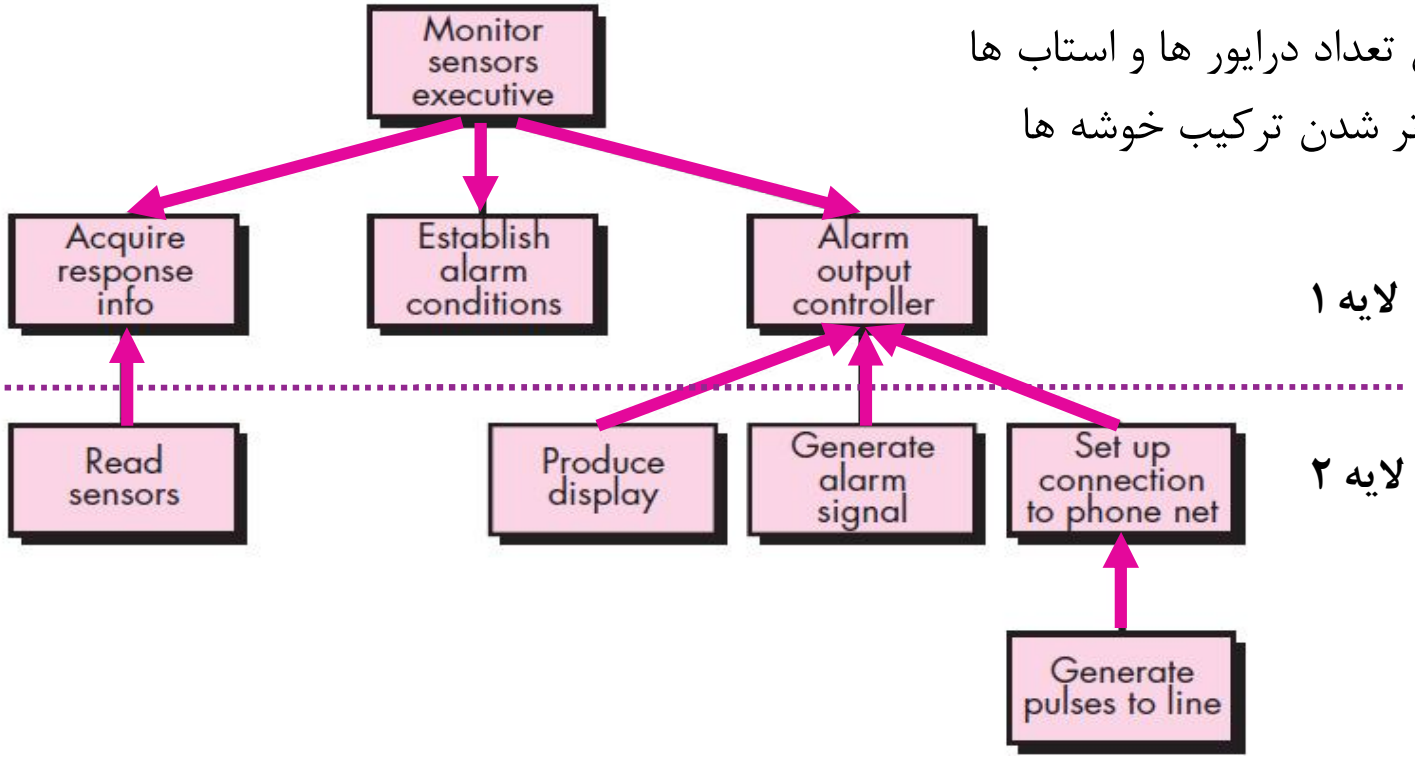
- تا زمان افزوده شدن آخرین پیمانه، برنامه به عنوان یک موجودیت کامل وجود ندارد (به عبارت دیگر، تا زمان منسجم نشدن پیمانه ریشه با سایر پیمانه ها، عملکرد های اصلی برنامه قابل استفاده نیستند).
- افزایش و پیچیده تر شدن درایورها

❖ مزایا

- طراحی آسان تر موارد آزمون و عدم نیاز به استاب ها
  - مولفه ها از پایین به بالا منسجم می شوند، در نتیجه مولفه های زیر دست در یک سطح مفروض، همیشه در دسترس هستند و نیازی به استاب ها نیست.

انسجام ترکیبی (Hybrid Integration)

- به این نوع انسجام، آزمون ساندویچ (sandwich testing) نیز گفته می شود.
- به معماری سیستم به صورت چند لایه نگاه می شود.
- در ساختار برنامه، برای سطوح بالاتر از روش بالا به پایین و برای سطوح زیرین از روش پایین به بالا استفاده می شود.
- **مزایا**
  - کاهش تعداد درایور ها و استاب ها
  - ساده تر شدن ترکیب خوشه ها



آزمون دود (Smoke Testing)

- یک راهکار انسجام گام به گام و روزانه برای پروژه هایی که زمان در آنها اهمیت دارد.
- مراحل
  - مولفه های نرم افزاری که تا کنون به کد ترجمه شده اند، منسجم می شوند و یک سازه (build) را تشکیل می دهند.
    - *یک سازه شامل فایل های داده ای، کتابخانه ها، پیمانه های قابل استفاده مجدد و مولفه هایی است که برای پیاده سازی یک یا چند عملکرد از محصول مورد نیاز هستند.*
  - مجموعه ای از آزمون ها طراحی می شود تا خطاهایی را پیدا کنند که مانع از اجرای درست عملکرد سازه می شوند.
  - سازه با سازه های دیگر منسجم می شود و محصول جاری به صورت روزانه مورد آزمون قرار می گیرد.
    - *روش انسجام می تواند بالا به پایین یا پایین به بالا باشد.*

آزمون دود ...

❖ مزایا

- ریسک انسجام به حداقل می رسد.
- کیفیت محصول نهایی بهبود می یابد.
- تشخیص و تصحیح خطا آسان می شود.
- ارزیابی پیشرفت پروژه، آسان تر می شود.

✓ معمولا از آزمون دود برای پروژه های پیچیده و حساس به زمان استفاده می شود.

آزمون رگرسیون (Regression Testing)

- افزودن یک مولفه جدید در هنگام انسجام و یا تغییر مولفه ها ممکن است بر عملکرد سایر مولفه ها و کل سیستم تاثیر داشته باشد.
  - جریان های داده ای جدید ایجاد شود.
  - I/O های جدید ممکن است رخ دهد.
  - یک فراخوانی جدید انجام شود.
- تغییرات ممکن است باعث مشکلات در عملکرد هایی شود که قبلا بدون نقص کار می کرده اند.
- هر زمان که تغییری عمده (از جمله انسجام مولفه های جدید) در نرم افزار رخ دهد، لازم است که آزمون رگرسیون اجرا شود.
- در آزمون رگرسیون مجموعه ای از آزمون ها که قبلا اجرا شده اند، بار دیگر اجرا می شوند.

آزمون رگرسیون ...

❖ نکات

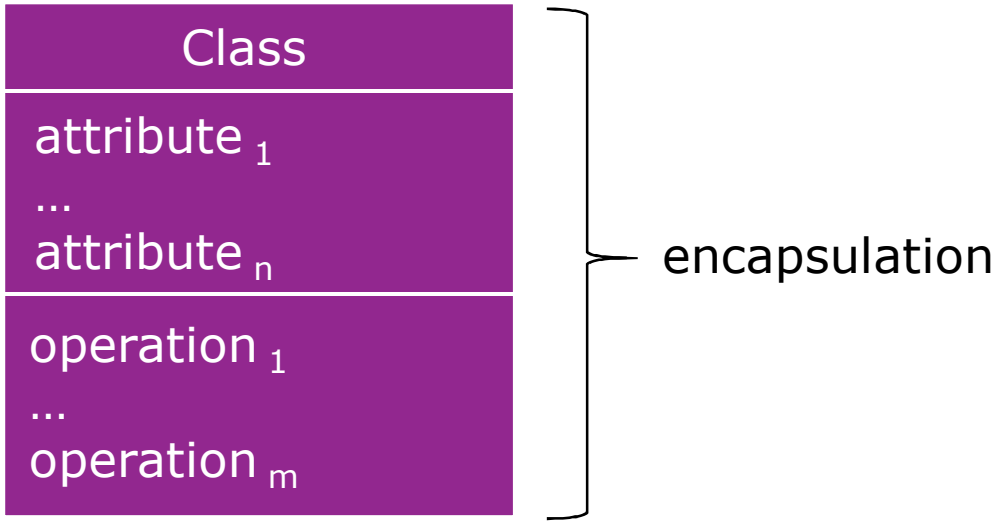
- ✓ برای آزمون مجددِ یک عملکرد، ممکن است:
  - بعضی از موارد آزمون که قبلا اجرا شده اند تغییر کرده و دوباره اجرا شوند.
  - نیاز به طراحی موارد آزمون جدید باشد.
- ✓ به موازات پیشرفت آزمون، تعداد آزمون های رگرسیون می تواند بسیار زیاد شود. اجرای دوباره همه آزمون ها برای هر یک از عملکرد های برنامه (پس از اعمال تغییر) معمولا غیر عملی است و بازدهی چندانی هم ندارد.
  - می توان تنها آزمون هایی را دوباره اجرا کرد که بر مولفه های تغییر یافته تاکید دارند.
  - آزمون های رگرسیون را باید طوری طراحی کرد که به ازای هر یک از عملکرد های اصلی، فقط آزمون های مرتبط با یک یا چند دسته از خطاها مجددا اجرا شود.

آزمون های سطح پایین
در رویکرد شی گرا

آزمون واحد در رویکرد شی گرا

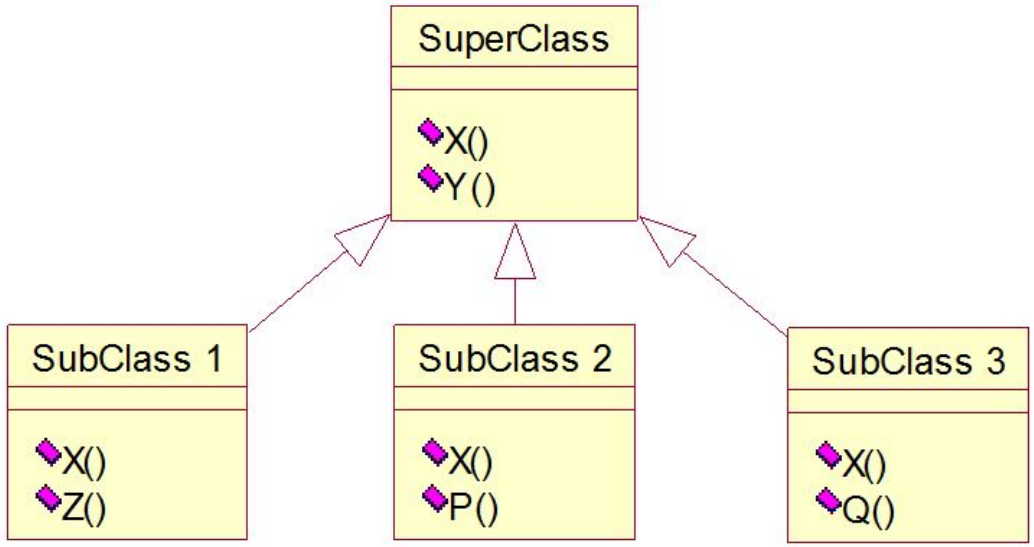
❖ **معماری شی گرا (Object Oriented Architecture)**

- هر مولفه شامل مجموعه ای از کلاس ها است.
- هر کلاس یا هر نمونه از کلاس (شی)، داده ها (صفات) و عملیات (توابع) مرتبط با این داده ها را بسته بندی می کند.
- هر یک از عناصر سیستمی (زیر سیستم ها و کلاس ها) وظایف مرتبط با بخشی از نیازمندی های سیستم را اجرا می کنند.



آزمون واحد در رویکرد شی گرا ...

- مفهوم واحد در رویکرد شی گرا با رویکرد سنتی متفاوت است.
  - یک کلاس می تواند حاوی چند عمل (operation) متفاوت باشد.
  - یک عمل خاص ممکن است در چند کلاس و با پیاده سازی های متفاوت وجود داشته باشد و یا با داده های متفاوت کار کند.
- نمی توان یک عمل را به طور جداگانه (دیدگاه سنتی) تست کرد بلکه باید آن را به عنوان جزئی از کلاس مورد آزمون قرار داد.



آزمون واحد در رویکرد شی گرا ...

- **واحد آزمون در نرم افزار های شی گرا = کلاس**
- آزمون واحد در نرم افزار های سنتی بیشتر بر جزئیات الگوریتمی یک پیمانه و داده هایی تاکید دارد که از میان واسط پیمانه جریان پیدا می کند، اما آزمون واحد در نرم افزار های شی گرا بر عملیات بسته بندی شده توسط کلاس و رفتار کلاس تاکید دارد.
- برای آزمون واحد در رویکرد شی گرا از آزمون کلاس (class testing) استفاده می شود.

آزمون کلاس (Class Testing)

- هر کلاس به همراه عملیات داخل آن، به طور جداگانه تست می شود.
- هدف
  - بررسی هر یک از عملیات کلاس
  - بررسی حالات رفتاری کلاس در هنگام فراخوانی عملیات آن
    - حالت کلاس (state) = مقادیر صفات کلاس در یک لحظه خاص

✓ از آزمون کلاس پس از طراحی در سطح مولفه ها یا پیاده سازی کد منبع می توان استفاده کرد زیرا جزئیات منطق کلاس باید در دسترس باشد.

✓ می توان از تکنیک های آزمون جعبه سفید برای آزمون عملیات تعریف شده در یک کلاس استفاده کرد. اما باید توجه داشت که این تکنیک ها، تعامل میان عملیات یک کلاس را تست نمی کنند.

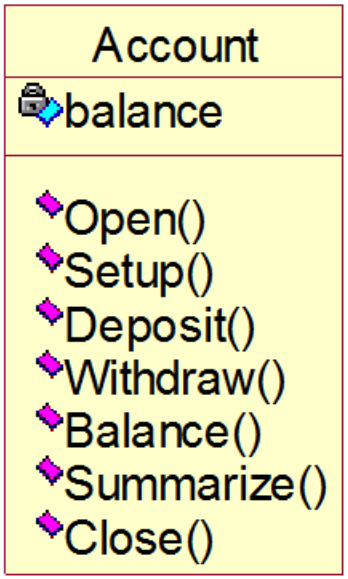
آزمون کلاس ...

- هر مورد آزمون کلاس (یا شی) باید شامل اطلاعات زیر باشد:
  - هدف آزمون
  - فهرستی از حالت های مشخص شده برای شی ای که باید تست شود.
  - فهرستی از پیام ها و عملیاتی که در طی آزمون تست می شوند.
  - فهرستی از شرایط استثنا که ممکن است در حین آزمون شی رخ دهند.
  - فهرستی از شرایط خارجی (تغییرات در محیط خارجی نرم افزار) که باید وجود داشته باشند تا آزمون به طور مناسب اجرا شود.
  - سایر اطلاعاتی که به درک یا پیاده سازی آزمون کمک می کنند.
- تکنیک های آزمون کلاس
  - آزمون تصادفی
  - آزمون افراز

آزمون تصادفی (Random Testing)

- تولید چند دنباله از عملیات قابل اجرا در کلاس به طور تصادفی
  - هر مورد آزمون، دنباله ای از عملیات را معرفی می کند.
- مشکل
  - تعداد تبدیلات جایگشت برای آزمون می تواند بسیار زیاد باشد.

- **مثال**
  - در یک سیستم بانکداری، برای ذخیره اطلاعات و انجام عملیات مرتبط با هر حساب بانکی از کلاسی به نام Account استفاده می شود.
  - محدودیت ها: پیش از آنکه عملیات دیگر قابل اجرا باشند باید حساب باز شود و در نهایت، حساب باید بسته شود.



آزمون تصادفی (Random Testing)

❖ **مثال ...**

- **چند مورد آزمون تصادفی برای بررسی عملیات کلاس Account**

Test case 1 *minimum life history of the class*

Open • Setup • Deposit • Withdraw • Close

Test case 2

Open • Setup • Deposit • Deposit • Balance • Summarize • Withdraw • Close

Test case 3

Open • Setup • Deposit • Withdraw • Deposit • Balance • Withdraw • Close

Test case 4

Open • Setup • Deposit • [Deposit|Withdraw|Balance|Summarize]^n • Withdraw • Close

آزمون افراز (Partition Testing)

❖ **هدف:** کاهش تعداد موارد آزمون لازم برای تست کلاس

❖ **مراحل**

- گروه بندی (افراز) ورودی ها و خروجی های کلاس
- طراحی موارد آزمون برای تست هر گروه

❖ **انواع افراز**

- افراز مبتنی بر حالت state-based partitioning
- افراز مبتنی بر صفت Attribute-based partitioning
- افراز مبتنی بر دسته Category-based partitioning

آزمون افراز ...

❖ افراز مبتنی بر حالت

- گروه بندی عملیات کلاس بر اساس توانایی آنها در تغییر دادن حالت کلاس
- طراحی موارد آزمون به گونه ای که عملیات تغییر دهنده حالت و عملیاتی که حالت را تغییر نمی دهند، به طور جداگانه بررسی شوند.

- **مثال:** کلاس Account
  - عملیات تغییر دهنده حالت کلاس Deposit() و Withdraw()
  - عملیاتی که حالت کلاس را تغییر نمی دهند. Balance() و Summarize()

Test case1

Open • Setup • Deposit • Deposit • Withdraw • Withdraw • Close

Test case2

Open • Setup • Deposit • Summarize • Withdraw • Close

آزمون افراز ...

- افراز مبتنی بر صفات
  - گروه بندی عملیات کلاس بر مبنای صفات
    - مثال: افراز عملیات کلاس Account بر مبنای صفت balance
      - عملیاتی که از صفت balance استفاده می کنند.
      - عملیاتی که صفت balance را تغییر می دهند.
      - عملیاتی که از صفت balance استفاده نمی کنند و آن را تغییر نمی دهند.
  - طراحی موارد آزمون برای هر گروه

آزمون افراز ...

- افراز مبتنی بر دسته
  - گروه بندی عملیات کلاس بر اساس عملکرد کلی هر یک از آنها
    - **مثال:** کلاس Account
      - عملیات آماده سازی : Setup() و Open()
      - عملیات محاسباتی : Withdraw() و Deposit()
      - عملیات پرسشی (گزارش گیری): Balance() و Summarize()
      - عملیات خاتمه دهنده: Close()
  - طراحی موارد آزمون برای هر گروه

چالش های موجود در طراحی موارد آزمون

❖ بسته بندی Encapsulation

- اگر عملیاتی در داخل یک کلاس برای گزارش مقادیر صفات کلاس وجود نداشته باشد، بدست آوردن حالت شی ممکن است دشوار باشد.

❖ وراثت Inheritance

- وراثت نیاز به آزمون کلاس های مشتق را برطرف نمی کند.
- آزمون های کلاس پایه (base) را می توان برای کلاس های مشتق (derived) نیز به کار برد اما در بعضی از موارد لازم است عملیات ارث برده شده دوباره تست شوند.
- داده های آزمون ممکن است هم برای کلاس پایه و هم برای کلاس مشتق مناسب باشد اما نتایج مورد انتظار ممکن است در کلاس مشتق متفاوت باشد.

چالش های موجود در طراحی موارد آزمون ...

❖ وراثت ...

نوع عملیات	توضیح	شیوه آزمون
عملیات جدید *New Operations*	تابع در کلاس تحت آزمون تعریف شده است و از کلاس دیگری ارث برده نشده است.	آزمون کامل تابع
عملیات ارث برده شده *Inherited Operations*	تابع در یک کلاس دیگر (کلاس پدر) تعریف شده است و در کلاس تحت آزمون ارث برده می شود.	اگر تابع ارث برده شده در کلاس تحت آزمون، با توابع جدید یا توابع دوباره تعریف شده در ارتباط باشد، لازم است تا دوباره تست شود.
عملیات دوباره تعریف شده *Redefined Operations*	تابع در یک کلاس دیگر (کلاس پدر) تعریف شده است. این تابع در کلاس تحت آزمون ارث برده شده اما پیاده سازی جدیدی برای آن تعریف می شود.	آزمون کامل تابع

چالش های موجود در طراحی موارد آزمون ...

❖ **سوال:** کلاس B از کلاس A ارث می برد. اگر کلاس A قبلا تست شده باشد، کدام یک از توابع کلاس B لازم است تست شوند؟

```
class A {
    public virtual void Mtd1() {
        Console.WriteLine("Mtd1 In A");
    }
    public virtual void Mtd2() {
        Console.WriteLine("Mtd2 In A");
    }
    public virtual void Mtd3() {
        Console.WriteLine("Mtd3 In A");
        Mtd1();
    }
}
```

```
class B : A {
    public override void Mtd1() {
        Console.WriteLine("Mtd1 In B");
    }
    public void Mtd4() {
        Console.WriteLine("Mtd4 In B");
    }
}
```

آزمون انسجام در رویکرد شی گرا

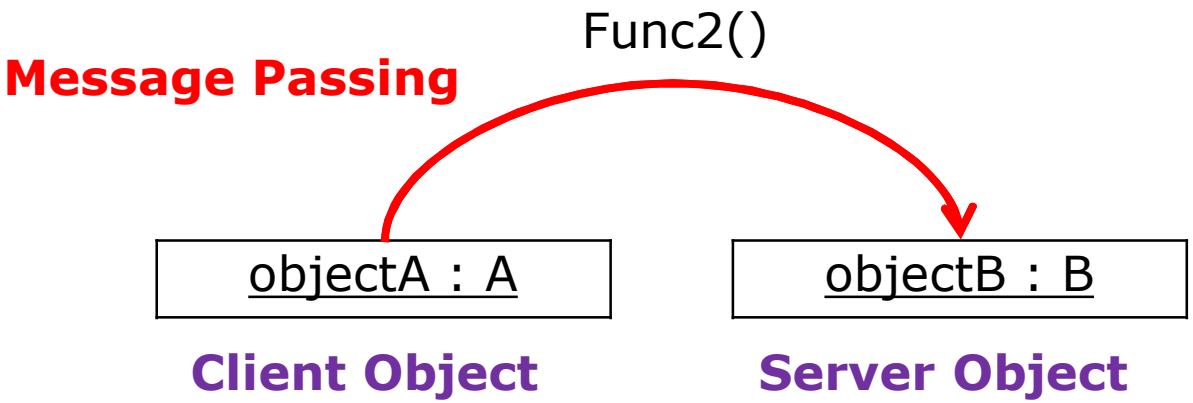
- راهبرد های انسجام افزایشی در نرم افزار های شی گرا چندان قابل استفاده نیستند.
  - نرم افزار های شی گرا فاقد ساختار کنترلی واضح و صریح هستند.
  - به دلیل تعاملات مستقیم و غیر مستقیم بین مولفه های تشکیل دهنده کلاس، عملیات انسجام به صورت «هر بار یک کلاس» اغلب غیر ممکن است.

- هدف آزمون انسجام
  - کشف خطاهای موجود در فراخوانی عملیات و ارتباط میان کلاس ها
    - استفاده از پیام یا عملیات اشتباه
    - فراخوانی نادرست عملیات
    - نتیجه غیر منتظره
  - آزمون انسجام سعی می کند خطاهای موجود در شی کلاینت را بیابد (نه شی سرور)

آزمون انسجام در رویکرد شی گرا

```
A objectA = new A();
objectA.Func1();
```

```
Class A
{
    ...
    Func1()
    {
        B objectB = new B();
        objectB.Func2();
    }
    ...
}
```



B is a collaborator class for A

روش های انسجام

❖ آزمون مبتنی بر نخ (Thread-based Testing)

- ✓ **نخ :** مجموعه ای از کلاس ها که به یک ورودی یا رویداد پاسخ می دهند.
  - هر نخ به طور جداگانه منسجم و تست می شود.
  - از آزمون رگرسیون استفاده می شود تا اطمینان حاصل شود انسجام نخ ها اثر جانبی به وجود نمی آورد.

❖ آزمون مبتنی بر کاربرد (Use-based Testing)

- ✓ **کلاس مستقل:** کلاسی که از هیچ یا تعداد کمی از کلاس های سرور استفاده می کند.
  - انسجام کلاس ها با آزمون کلاس های مستقل آغاز می شود.
  - پس از آزمون کلاس های مستقل، لایه بعدی کلاس ها، یعنی کلاس های وابسته که از کلاس های مستقل استفاده می کنند، تست می شوند.
  - آزمون کلاس های وابسته ادامه می یابد تا زمانی که کل سیستم تست شود.

آزمون خوشه ای (Cluster Testing)

- یک مرحله از آزمون انسجام در نرم افزار های شی گرا
- در این آزمون، خوشه ای از کلاس های همکار تست می شوند. در این راستا، موارد آزمونی طراحی می شوند که سعی در کشف خطاهای موجود در همکاری ها دارند.

✓ با بررسی کارت های CRC و مدل های مبتنی بر کلاس می توان کلاس های همکار را تعیین کرد.

✓ کلاس B همکار کلاس A است اگر برای انجام یکی از مسئولیت های کلاس A به اطلاعات کلاس B نیاز باشد.

- انواع همکاری
  - درخواست برای دریافت اطلاعاتی که کلاس آنها را در اختیار ندارد.
  - درخواست برای بروز رسانی اطلاعاتی که کلاس آنها را در اختیار ندارد.
  - درخواست برای انجام عملی که از مسئولیت های یک کلاس دیگر است.

طراحی موارد آزمون

❖ هر مورد آزمون، جریان عملیات در میان کلاس های همکار را تست می کند.

❖ **مراحل**

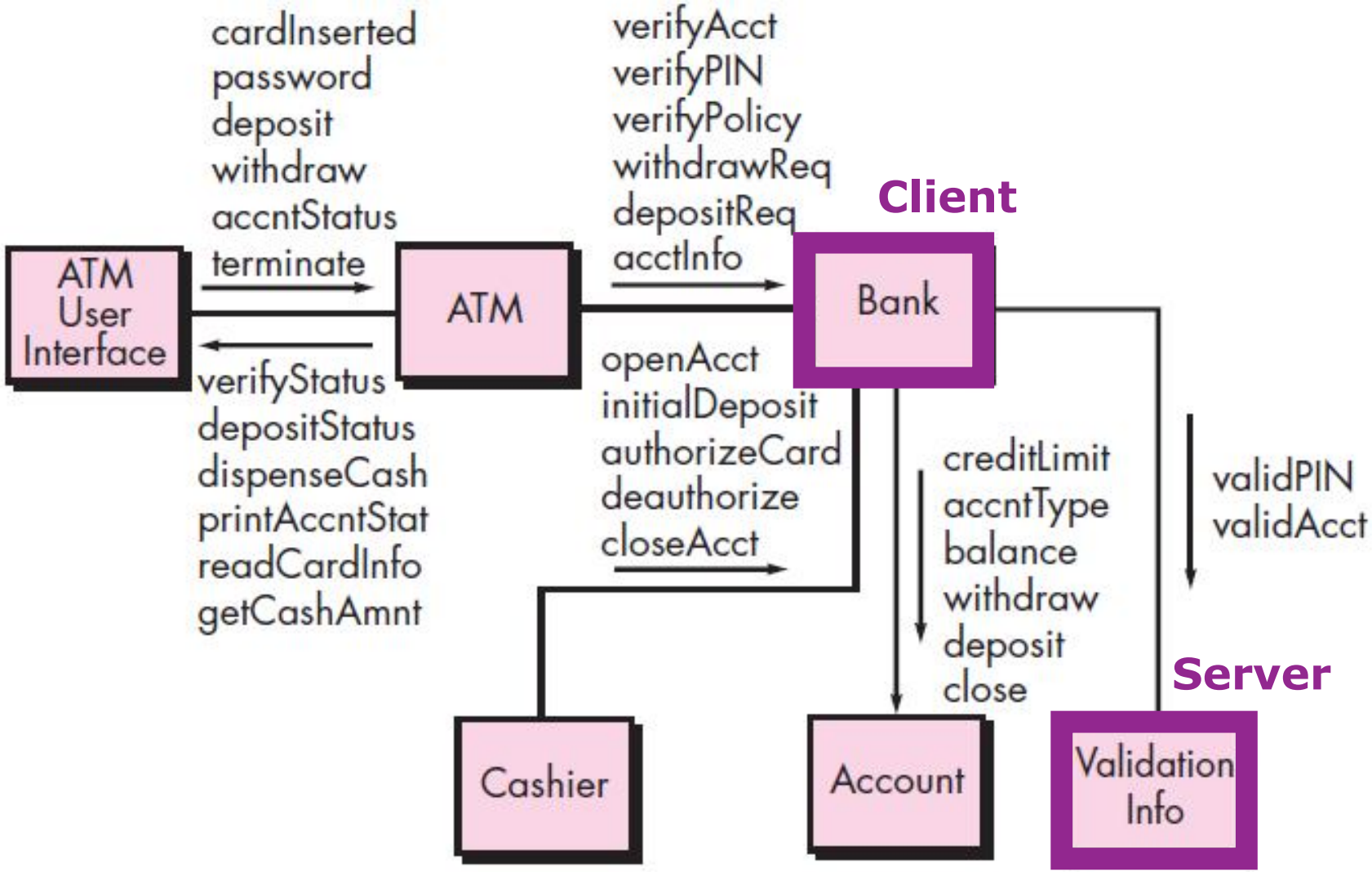
۱. برای هر کلاس کلاینت، با استفاده از عملیات کلاس، موارد آزمون را تولید کنید.
۲. عملیات کلاس ممکن است پیام هایی را به کلاس های سرور ارسال کنند. برای هر پیامی که تولید می شود، کلاس همکار و عملیات متناظر در شی سرور را تعیین کنید.
۳. عملیات شی سرور را در دنباله آزمون در نظر بگیرید.
۴. مراحل قبل را برای سطوح بعدی تکرار کنید.

❖ **روش های طراحی موارد آزمون**

- تصادفی
- افراز
- مبتنی بر رفتار
- مبتنی بر سناریو

طراحی موارد آزمون به صورت تصادفی

❖ **مثال :** نمودار همکاری برای یک سیستم بانکداری



طراحی موارد آزمون به صورت تصادفی ...

❖ **مثال ...**

۱. طراحی موارد آزمون تصادفی برای کلاس Bank

Test case : verifyAcct • verifyPIN • depositReq

۲. تعیین کلاس های همکار و عملیات متناظر در آن به ازای هر یک از عملیات ذکر شده در موارد آزمون مرحله قبل

- در اجرای عملیات verifyAcct() و verifyPIN()، کلاس ValidationInfo با کلاس Bank از طریق عملیات validAcct() و validPIN() همکاری می کند.
- در اجرای عملیات depositReq()، کلاس Account با کلاس Bank از طریق عملیات deposit() همکاری می کند.

۳. گسترش موارد آزمون طراحی شده بر اساس کلاس های همکار

Test case :

verifyAcct [ValidationInfo.validAcct] •

verifyPIN [ValidationInfo.validPIN] •

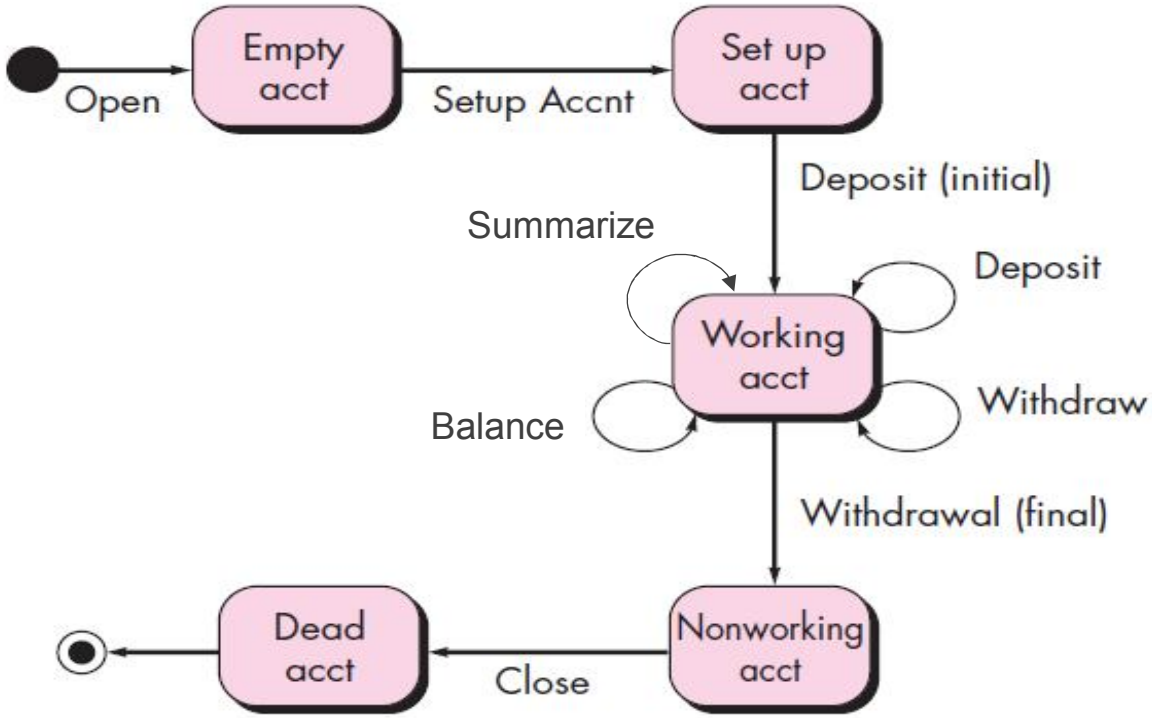
depositReq [Account.deposit]

طراحی موارد آزمون مبتنی بر افراز

- کلاس ها بر اساس روابطی که با یک کلاس خاص دارند، افراز می شوند.
- **مثال :** کلاس Bank پیام هایی را از کلاس های ATM و Cashier دریافت می کند.
  - افراز عملیات کلاس Bank بر اساس سرویس دهی آنها به کلاس ATM یا Cashier
  - طراحی موارد آزمون برای هر گروه

طراحی موارد آزمون مبتنی بر مدل های رفتاری

- تست رفتار پویای کلاس و کلاس هایی که با آن همکاری دارند.
- استفاده از مدل های رفتاری مانند نمودار حالت برای طراحی موارد آزمون
- **مثال:** نمودار حالت کلاس Account



طراحی موارد آزمون مبتنی بر مدل های رفتاری ...

❖ مثال ...

- دنباله ای از عملیات ها باید تست شوند به گونه ای که کلاس Account از همه حالت های مجاز گذر کند.
- در شرایطی که رفتار کلاس منجر به همکاری با یک یا چند کلاس می شود، از چندین نمودار حالت برای پیگیری جریان رفتاری سیستم استفاده می شود.

Test case 1

Open • Setup • Deposit(initial) • Withdraw(final) • Close

Test case 2

Open • Setup • Deposit(initial) • Deposit • Balance • Withdraw(final) • Close

Test case 3

Open • Setup • Deposit(initial) • Deposit • Withdraw • Summarize • Withdraw (final) • Close

طراحی موارد آزمون مبتنی بر سناریو

- استفاده از نمودار های موارد استفاده (use case) برای انتخاب کلاس های همکار و تست روابط میان آنها
- مراحل
  - تعیین کلاس های لازم و عملیات مرتبط با هر یک از آنها برای اجرای بخشی از یک use case
  - طراحی موارد آزمون به گونه ای که شامل دنباله ای از عملیات مورد نظر باشند.

نرم افزار های درایور و استاب

- از درایور می توان برای جایگزین کردن واسط کاربر، تست عملیات ها در پایین ترین سطوح و یا برای تست گروه کاملی از کلاس ها استفاده کرد.
- زمانی که همکاری بین کلاس ها مورد نیاز است ولی یک یا چند کلاس همکار هنوز پیاده سازی نشده اند، می توان از استاب ها استفاده کرد.
- در انسجام شی گرا، باید تا حد امکان از بکارگیری درایور ها و استاب ها به عنوان عملیات جایگزین پرهیز کرد.

آزمون اعتبار سنجی

آزمون اعتبار سنجی (Validation Testing)

- یک آزمون سطح بالا است که در آن نیازمندی های ثبت شده در سند تحلیل نرم افزار در مقابل نرم افزار ساخته شده بررسی می شود.
- تاکید بر ورودی ها و خروجی های قابل مشاهده توسط کاربر
- اطمینان نهایی از رعایت همه نیازمندی های رفتاری، عملیاتی و کارایی نرم افزار
- اعتبار سنجی هنگامی موفق است که نرم افزار بر اساس انتظار مشتری عمل کند.
- برای طراحی موارد آزمون اعتبار سنجی، از تکنیک های جعبه سیاه استفاده می شود. این موارد آزمون توسط سازنده نرم افزار اجرا می شوند.
- برای کشف خطاهایی که به نظر می رسد فقط کاربر نهایی قادر به یافتن آنها است، از آزمون پذیرش استفاده می شود.

آزمون پذیرش (Acceptance Test)

- آخرین مرحله از آزمون اعتبار سنجی
- آزمون توسط کاربر نهایی انجام می شود (نه تولید کننده نرم افزار)
- آزمون های پذیرش را می توان به شیوه غیر رسمی یا برنامه ریزی شده طراحی کرد.
- آزمون ممکن است در عرض یک هفته تا یک ماه انجام شود.

- **دلایل نیاز به آزمون پذیرش**
  - نمی توان پیش بینی کرد که مشتری چگونه از نرم افزار استفاده خواهد کرد.
  - راهنمای استفاده از نرم افزار ممکن است به درستی تفسیر نشود.
  - ممکن است ترکیبات نامشخصی از داده ها و ورودی ها توسط کاربر استفاده شود.
  - خروجی که در نظر آزمون گر واضح است ممکن است برای کاربر نهایی نامفهوم باشد.

- **انواع آزمون پذیرش**
  - آزمون آلفا
  - آزمون بتا

آزمون پذیرش ...

❖ آزمون آلفا (Alpha Testing)

- آزمون توسط مشتری یا یک گروه از کاربران نهایی انجام می شود.
- آزمون در محیط کنترل شده اجرا می شود.
- نرم افزار در شرایط طبیعی اجرا می شود و کاربران با آن کار می کنند.
- تولید کننده نرم افزار ناظر بر اجرای آزمون است و در هنگام تعامل کاربران با نرم افزار، خطاها و مشکلات رخ داده را ثبت می کند.

آزمون پذیرش ...

❖ آزمون بتا (Beta Testing)

- بر خلاف آزمون آلفا، در این آزمون تیم تولید کننده نرم افزار در حین اجرای آزمون حضور ندارند.
- در یک یا چند مکان متعلق به مشتری یا کاربران نهایی، نرم افزار اجرا شده و تست می شود.
- مشتری یا کاربر نهایی تمام مشکلات واقعی یا فرضی را که طی آزمون یافته است ثبت می کند و آنها را در فاصله های زمانی منظم به تیم تولید کننده گزارش می دهد.
- با توجه به مشکلات گزارش شده، تیم تولید کننده اصلاحات لازم را انجام می دهند و محصول را برای تحویل نهایی آماده می کنند.

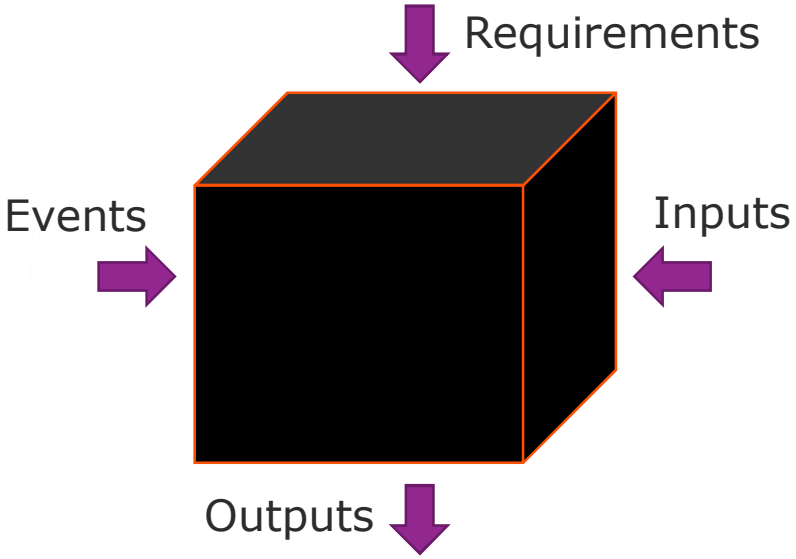
بازبینی پیکر بندی (Configuration Review)

- یک عنصر مهم از فرآیند اعتبار سنجی
- اطمینان از اینکه همه آیتم های پیکر بندی نرم افزار (برنامه ها، داده ها و مستندات) :
  - به طور مناسب توسعه یافته اند.
  - اطلاعات مرتبط با هر آیتم به طور کامل ثبت شده است.
  - تغییرات لازم در آنها به درستی اعمال و پیاده سازی شده است.
  - دارای جزئیات لازم برای سهولت در فاز نگهداری نرم افزار هستند.

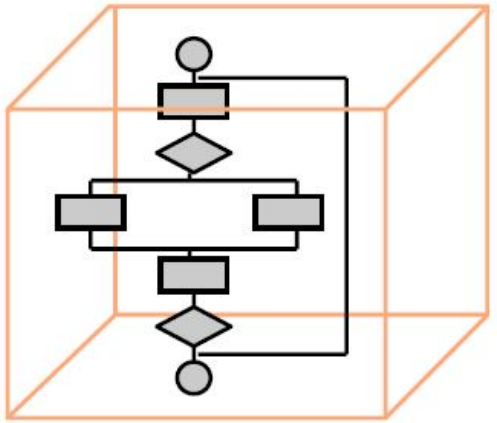
- به بازبینی پیکر بندی، بازرسی یا auditing نیز گفته می شود.

آزمون جعبه سیاه (Black-box Testing)

- اعتبار سنجیِ نیازمندی های عملیاتی نرم افزار بدون در نظر گرفتن منطق داخلی برنامه
- کشف خطاها بر مبنای نیازمندی ها و مشخصات برنامه
- آزمون جعبه سیاه = آزمون عملیاتی = آزمون رفتاری = آزمون مبتنی بر مشخصات



Black Box Testing



White Box Testing

آزمون جعبه سیاه ...

- ✓ تکنیک های جعبه سیاه شامل آزمون هایی است که بر روی واسط نرم افزار اجرا می شوند.
- ✓ برای طراحی موارد آزمون اعتبار سنجی و گاهی اوقات، در آزمون های انسجام و آزمون واحد از تکنیک های جعبه سیاه استفاده می شود.
- ✓ تکنیک های جعبه سیاه جایگزینی برای تکنیک های جعبه سفید نیست بلکه یک روش مکمل برای پیدا کردن گروه دیگری از خطاها است.

❖ انواع خطاهای قابل شناسایی توسط تکنیک های جعبه سیاه

- عملکرد های نادرست یا جا افتاده
- خطاهای واسط
- خطا در دسترسی به ساختمان داده ها یا پایگاه داده های خارجی
- خطاهای رفتاری یا کارایی
- خطا در شروع یا خاتمه برنامه

آزمون جعبه سیاه ...

- تکنیک های آزمون جعبه سیاه
  - افزار هم ارزی
  - تحلیل مقادیر مرزی
  - آزمون مبتنی بر مدل
  - آزمون آرایه متعامد
  - آزمون های مبتنی بر گراف
  - و ...

افراز هم ارزی (Equivalence Partitioning)

- دامنه ورودی (یا خروجی) برنامه به چند گروه تقسیم می شود و برای هر گروه، موارد آزمون طراحی می شود.
- به هر گروه، یک کلاس هم ارزی گفته می شود.
- هدف
  - از یک مجموعه مقادیر معادل (کلاس هم ارزی)، تنها یک مقدار تست شود.
  - کاهش تعداد موارد آزمون
- کلاس هم ارزی (*equivalence class*)
  - مجموعه ای از مقادیر معتبر یا نامعتبر برای دامنه ورودی (یا خروجی) برنامه
  - به ازای مقادیر مختلف از یک کلاس، انتظار می رود که نرم افزار رفتار معادلی از خود نشان دهد.

افراز هم ارزی ...

❖ چند قانون برای تعریف کلاس های هم ارزی

- اگر ورودی/خروجی بازه ای مانند [min,max] را مشخص کند، یک کلاس هم ارزی برای مقادیر معتبر و دو کلاس هم ارزی برای مقادیر نامعتبر تعریف می شود.
  - $a>max$ $a<min$ $min \leq a \leq max$
- اگر ورودی/خروجی به یک مقدار خاص مانند value نیاز داشته باشد، یک کلاس هم ارزی برای مقادیر معتبر و دو کلاس هم ارزی برای مقادیر نامعتبر تعریف می شود.
  - $a<value$ $a>value$ $a=value$
- اگر ورودی/خروجی عضوی از یک مجموعه مانند X را مشخص کند، یک کلاس هم ارزی برای مقادیر معتبر و یک کلاس هم ارزی برای مقادیر نامعتبر تعریف می شود.
  - $a \notin X$ $a \in X$
- اگر ورودی/خروجی یک مقدار بولین باشد، یک کلاس هم ارزی برای مقادیر معتبر و یک کلاس هم ارزی برای مقادیر نامعتبر تعریف می شود.
  - $a=true$ $a=false$

افراز هم ارزی ...

❖ **مثال:** تابع Func یک عدد چهار رقمی طبیعی مانند x را به عنوان پارامتر ورودی می پذیرد؛ ارقام اول و آخر عدد را با هم جابجا می کند و آن را به عنوان خروجی باز می گرداند. در صورت نامعتبر بودن ورودی، خروجی تابع برابر با 1- خواهد بود. کلاس های هم ارزی و موارد تست را برای این تابع مشخص کنید.

✓ x in [1000-9999]

- class 1: $1000 \leq x \leq 9999$ *(valid values)*
- class 2: x<1000 *(invalid values)*
- class 3: x>9999 *(invalid values)*

	داده های آزمون	نتیجه مورد انتظار	توضیح
مورد آزمون ۱	x=4507	7504	پوشش کلاس ۱
مورد آزمون ۲	x=800	-1	پوشش کلاس ۲
مورد آزمون ۳	x=12345	-1	پوشش کلاس ۳

تحلیل مقادیر مرزی (Boundary Value Analysis)

- آزمون BVA، موارد آزمونی را برای بررسی مقادیر مرزی طراحی می کند.
- این تکنیک مکمل افراز هم ارزی است و بر مقادیر مرزی یک کلاس هم ارزی تاکید دارد.
- **مثال:** اگر ورودی x یک مقدار در بازه [a,b] باشد، موارد آزمون BVA به صورت زیر تعریف می شوند:

Test case 1: a-1

Test case 2: a

Test case 3: a+1

Test case 4: b-1

Test case 5: b

Test case 6: b+1

تحلیل مقادیر مرزی ...

❖ **مثال:** تحلیل مقادیر مرزی در تابع Func ...

- ✓ x in [1000-9999]
- ✓ a = 1000 , b = 9999

	داده های آزمون	نتیجه مورد انتظار	توضیح
مورد آزمون ۱	x=999	-1	a-1
مورد آزمون ۲	x=1000	0001	a
مورد آزمون ۳	x=1001	1001	a+1
مورد آزمون ۴	x=9998	8999	b-1
مورد آزمون ۵	x=9999	9999	b
مورد آزمون ۶	x=10000	-1	b+1

آزمون مبتنی بر مدل (Model-based Testing)

- استفاده از اطلاعات موجود در مدل نیازمندی ها به عنوان مبنایی برای طراحی موارد آزمون
- هدف: کشف خطاها در رفتار نرم افزار به هنگام اجرا
- مراحل

۱. ایجاد یا تحلیل یک مدل رفتاری برای نرم افزار؛ مانند مدل حالت، مدل جریان داده (DFD)، مدل موارد استفاده (use case) و ...

۲. طراحی موارد آزمون بر مبنای مدل انتخاب شده

- مشخص کردن ورودی هایی که نرم افزار را وادار می سازند تا از حالتی به حالت دیگر منتقل شود.
- تعیین خروجی قابل انتظار از نرم افزار، هنگامی که از حالتی به حالت دیگر منتقل می شود.

آزمون سیستم

آزمون سیستم

- آخرین مرحله از آزمون های سطح بالا که خارج از مرز مهندسی نرم افزار و در حیطه وسیع تر یعنی مهندسی سیستم قرار می گیرد.
- اطمینان از اینکه همه عناصر سیستمی (مثل سخت افزار، نرم افزار، بانک های اطلاعاتی، افراد و غیره) در ترکیب با یکدیگر به درستی عمل می کنند و عملکرد کلی سیستم قابل دستیابی است.
- **انواع آزمون سیستم**
  - آزمون بازیابی (احیا)
  - آزمون امنیت
  - آزمون فشار
  - آزمون کارایی
  - آزمون استقرار

انواع آزمون سیستم

❖ آزمون بازیابی (Recovery Testing)

- بسیاری از سیستم های کامپیوتری باید خود را در شرایط شکست ترمیم کنند و در یک زمان از پیش تعیین شده پردازش را ادامه دهند؛ در غیر این صورت زیان های اقتصادی شدیدی به وجود خواهد آمد.

- آزمون بازیابی، نرم افزار را به روش های گوناگون وادار به شکست می کند و سپس مراحل اجرای ترمیم را بررسی می کند.
  - برای مثال، زمان میانگین برای ترمیم (MTTR) اندازه گیری شده و بررسی می شود که در حد قابل قبولی است یا خیر.
- انواع ترمیم
  - ترمیم خودکار: توسط سیستم اجرا می شود.
  - ترمیم با دخالت انسان

انواع آزمون سیستم ...

❖ آزمون امنیت (Security Testing)

- **امنیت:** احتمال اینکه سیستم بتواند در برابر حملات تصادفی یا عمدی از خود محافظت کند.
- **آزمون امنیت:** آیا راهکار های محافظت تعبیه شده در داخل سیستم واقعا آن را از نفوذ نامناسب حفظ می کنند؟
- در آزمون امنیت، آزمونگر نقش فردی را بازی می کند که مایل به نفوذ به سیستم است.
  - وارد کردن کلمات عبور به روش های مختلف
  - نفوذ به ساختار دفاعی سیستم با استفاده از یک سری نرم افزار ها
  - جلوگیری کردن از ارائه خدمات به کاربران
  - ایجاد خطا در سیستم تا در هنگام ترمیم و بازیابی خطا به آن نفوذ کند
  - جستجو در میان داده های غیر امن تا راهی برای ورود به سیستم پیدا کند

✓هر سیستمی با صرف هزینه و زمان قابل نفوذ است. طراح سیستم باید تلاش کند تا هزینه نفوذ به سیستم بیشتر از ارزش اطلاعاتی باشد که در اثر نفوذ به دست می آید.

انواع آزمون سیستم ...

❖ آزمون فشار (Stress Testing)

- در آزمون فشار، منابع مورد نیاز سیستم به میزان غیر عادی درخواست می شود. سپس بررسی می شود نرم افزار تا قبل از وقوع شکست حداکثر تا چه مدت زمانی می تواند به کار خود ادامه دهد.
- مثال
  - طراحی موارد آزمونی که مستلزم حداکثر حافظه یا منابع دیگر هستند.
  - طراحی موارد آزمونی که در هر ثانیه ده وقفه در سیستم ایجاد می کنند در حالی که میانگین آن در حالت نرمال یک یا دو وقفه در ثانیه است.
  - طراحی موارد آزمونی که نرخ ورود داده ها را چندین برابر (نسبت به حالت عادی) می کنند.
- **آزمون حساسیت (Sensitivity Testing)**
  - شکل دیگری از آزمون فشار است.
  - تلاش می کند تا ترکیباتی از داده های ورودی سیستم را کشف کند که ممکن است باعث ناپایداری یا پردازش نامناسب در سیستم شوند.

انواع آزمون سیستم ...

❖ آزمون کارایی (Performance Testing)

- آیا میزان منابع مصرفی سیستم در زمان اجرا، مطابق با اهداف کارایی از قبل تعیین شده است؟
- آزمون های کارایی غالبا در ترکیب با آزمون فشار انجام می شوند.

❖ آزمون استقرار (Deployment Testing)

✓ به این آزمون، آزمون پیکربندی (configuration testing) نیز گفته می شود.

- آزمون نرم افزار در هر کدام از محیط هایی که قرار است در آنها اجرا شود.
  - در بسیاری از موارد، لازم است نرم افزار بر روی انواع سخت افزار ها و در بیش از یک نوع سیستم عامل قابل اجرا باشد.
- بررسی مراحل نصب، نرم افزار های لازم برای نصب و مستنداتی که به مشتریان و کاربران نهایی ارائه می شوند و اطمینان از صحت آنها

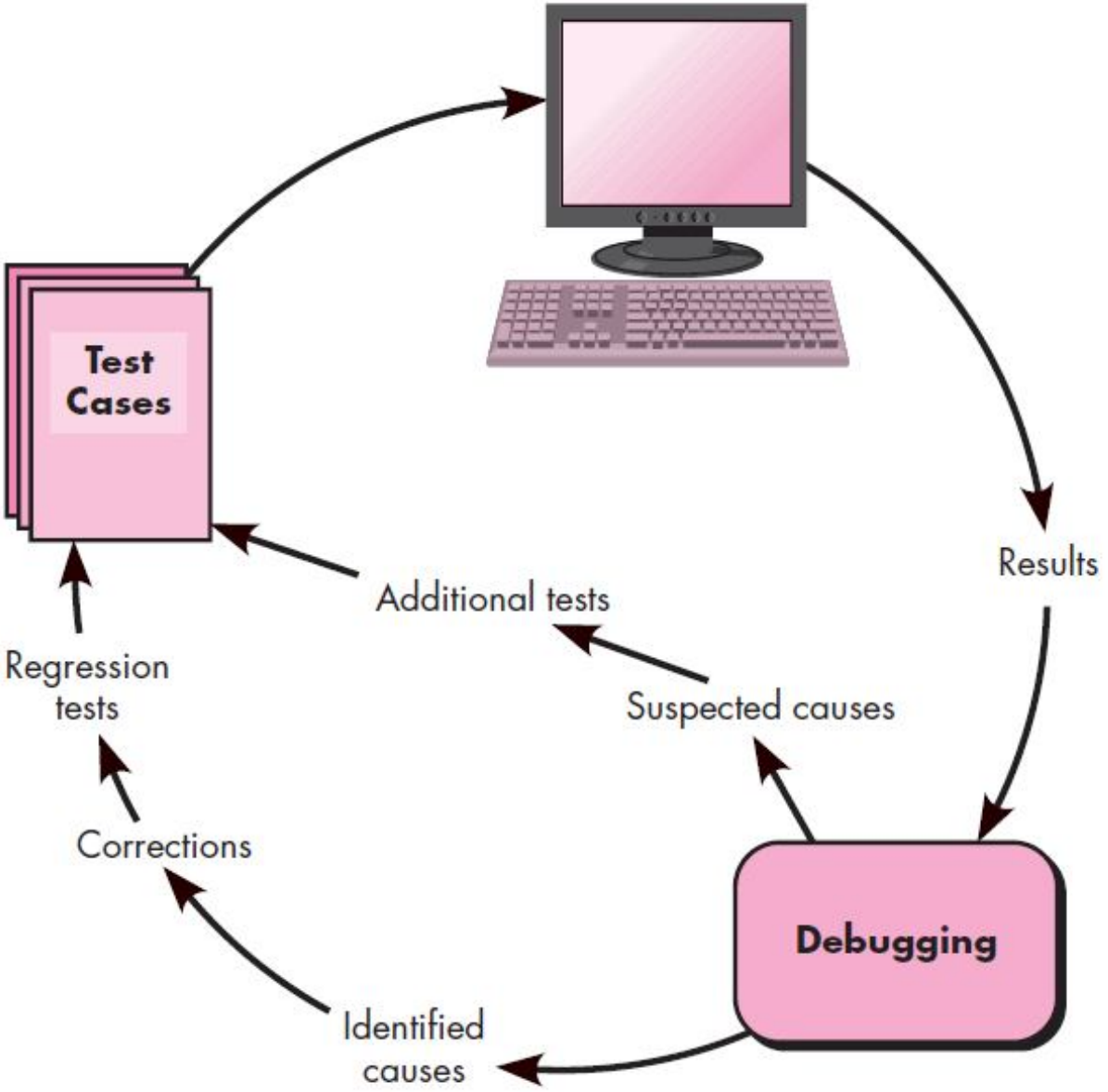
آزمون چه زمانی به پایان می رسد؟

- آزمون هیچ گاه تمام نمی شود، فقط مسئولیت آن از مهندس نرم افزار به مشتری (کاربر نهایی) منتقل می شود.
  - ✓هر بار که کاربر از برنامه استفاده می کند، یک آزمون انجام شده است.
- آزمون وقتی به پایان می رسد که زمان یا پول شما تمام شود.
- می توان از مدل سازی آماری و نظریه قابلیت اطمینان برای پیش بینی کامل بودن آزمون استفاده کرد.
- استفاده از قانون پارتو (Parto)
  - ۸۰ درصد از همه خطاهای کشف شده در طی فرآیند تست، احتمالا در ۲۰ درصد از روال های برنامه وجود دارند.
    - بهتر است که این روال ها را شناخته و آنها را کاملا تست کنیم.

اشکال زدایی (Debugging)

- **هدف آزمون:** یافتن خطا
- **هدف اشکال زدایی:** یافتن علت خطا و تصحیح آن
- در برخی از موارد، نشانه خطا (symptom) و علت اصلی آن ممکن است رابطه روشن و صریحی با یکدیگر نداشته باشند.
- **اشکال زدایی:** یک فرآیند ذهنی که نشانه خطا را به یک علت ربط می دهد.
  - ✓ اشکال زدایی آزمون نیست، بلکه پس از آزمون انجام می شود.
  - ✓ استفاده از یک روش اصولی + نبوغ + شانس = موفقیت در اشکال زدایی

اشکال زدایی ...


Test Cases
Results
Additional tests
Regression tests
Suspected causes
Corrections
Debugging
Identified causes

چرا اشکال زدایی دشوار است؟

- دور بودن نشانه خطا و علت آن از نظر مکانی
- ناپدید شدن نشانه خطا با تصحیح یک خطای دیگر (به طور موقت)
- خطاهای برنامه ممکن است ناشی از خطای انسانی باشد که به سادگی قابل ردگیری نیست. (خطاهای منطقی)
- نشانه ها ممکن است نتیجه مشکلات زمانبندی باشد نه مشکلات پردازشی
- نشانه ها ممکن است ناشی از عللی باشد که در میان چند وظیفه توزیع شده اند و در پردازنده های متفاوت اجرا می شوند.
- و ...

روش های اشکال زدایی

❖ روش بدون تفکر / جستجوی جامع (brute force)

- استفاده از فلسفه «یافتن خطا توسط خود کامپیوتر»
  - چاپ محتویات حافظه (مقادیر متغیر ها و ...) در صفحه نمایش و ردیابی دستورات برنامه در زمان اجرا
- متداول ترین و کم اثر بخش ترین روش
- اتلاف تلاش و زمان
- زمانی از این روش باید استفاده شود که همه روش های دیگر با شکست روبرو شده است.

❖ روش عقبگرد (backtracking)

- با شروع از محلی که نشانه خطا آشکار شده است، به صورت رو به عقب و به طور دستی، کد بررسی می شود تا محل علت خطا پیدا شود.
- یک روش اشکال زدایی نسبتا متداول که اغلب در برنامه های کوچک موفق است.
- با افزایش تعداد خطوط کد، تعداد مسیر های رو به عقب افزایش می یابد.

روش های اشکال زدایی ...

❖ **روش حذف علت (cause elimination)**

- داده های مرتبط با نشانه خطا جمع آوری می شوند.
- فهرستی از همه علل احتمالی تهیه می شود.
- با استفاده از داده های جمع آوری شده، آزمون هایی برای هر یک از علت ها (با هدف اثبات درستی یا رد آن) طراحی و اجرا می شود.
- اگر نتایج نشان دادند که یکی از علت ها با نشانه خطا مرتبط است، تلاش می شود تا آن خطا برطرف شود.

❖ نکات

✓برای مدتی که می خواهید صرف اشکال زدایی کنید یک محدودیت زمانی مثلا یک یا دو ساعت در نظر بگیرید. پس از آن، از اعضای تیم، شخص دیگر، اینترنت و غیره کمک بگیرید.

✓برنامه نویسی جفتی، روشی برای اشکال زدایی به موازات طراحی و کدنویسی است.

خلاصه فصل

- هدف آزمون نرم افزار، بدست آوردن مجموعه ای از موارد آزمون است که با حداقل تلاش و زمان، حداکثر خطاهای نرم افزار را کشف کند.
- آزمون در ابعاد کوچک (واحد) آغاز می شود و به ابعاد بزرگ (سیستم) پیشرفت می کند.
- آزمون واحد بر عملکرد انفرادی هر واحد (پیمانه، کلاس و ...) و آزمون انسجام بر نحوه ترکیب واحد ها مطابق با معماری نرم افزار تاکید دارد.
- در رویکرد سنتی، از تکنیک های جعبه سفید برای طراحی موارد آزمون واحد استفاده می شود. این تکنیک ها بر ساختار کنترلی برنامه (شرط ها، مسیر های پایه، حلقه ها و جریان های داده) تاکید دارند. زمانی که هر یک از واحد ها مورد آزمون قرار گرفتند، واحد ها به شیوه پایین به بالا، بالا به پایین و یا هر دو با هم ترکیب می شوند و و طی آزمون انسجام، خطاهای ناشی از ترکیب آنها و خطاهای موجود در واسط بین آنها کشف می شود.
- در رویکرد شی گرا، تاکید آزمون از مولفه رویه ای (پیمانه) به کلاس متمایل می شود. هنگامی که کد ها در دسترس باشند، علاوه بر تکنیک های جعبه سفید، از آزمون کلاس ها استفاده شده و هر کلاس به طور جداگانه تست می شود. آزمون انسجام را نیز می توان با استفاده از یک راهبرد مبتنی بر نخ یا مبتنی بر کاربرد انجام داد.
- آزمون اعتبار سنجی، میزان مطابقت نرم افزار تولید شده را با نیازمندی های مشتری بررسی می کند و برای طراحی موارد آزمون از تکنیک های جعبه سیاه استفاده می کند.
- آزمون سیستم، نرم افزار را پس از قرار گرفتن در یک سیستم بزرگ تر مورد بررسی قرار می دهد و عملکرد کلی سیستم را ارزیابی می کند.
- فرآیند اشکال زدایی با در نظر گرفتن نشانه های خطا، تلاش می کند تا علت خطا را بیابد و آن را تصحیح کند.

The best tester
is not the one who finds
the most bugs …
the best tester
is the one who gets the
most bugs fixed.

Question?