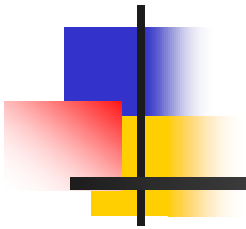
تکنیک های تست نرم افزار



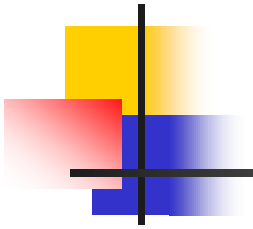
اهداف تست

- نگرش ما در تست نرم افزار
  - تست: اجرای یک برنامه با هدف پیدا کردن خطا
  - تست خوب: احتمال پیدا کردن خطاهای کشف نشده توسط ارزیابی زیاد است.
  - تست موفق: که حداقل یک خطای کشف نشده را بیابد
- تست فقط وجود خطا را نشان می دهد و نه عدم وجود آن را. پیدا نشدن خطا در تست به معنای بدون خطا بودن برنامه نیست.

اصول تست

- تست با توجه به نیازمندیهای کاربر
- برنامه ریزی قبل از اجرا
  - نوشتن برنامه تست (test plan)
- قانون پارتو
  - 80% خطاهای کشف نشده در 20% کد است
- تست باید از اجزای کوچک شروع شود
- تست کامل (exhaustive) ممکن نیست
- برای موثر بودن باید توسط شخص ثالث بیطرف انجام شود

معیارهای تست پذیر بودن نرم افزار



۱. قابلیت اجرا Operability

- هرچه نرم افزار بهتر کار کند و در محیط های بیشتری قابل اجرا باشد، بهتر قابل ارزیابی است

۲. مشاهده پذیری Observability

- قابلیت مشاهده نتایج ارزیابی

۳. کنترل پذیری Controlability

- قابلیت اجرای تستهای خودکار (مثل امکان اجرای خودکار تست های واحد توسط jUnit برای زبان جاوا)

۴. تجزیه پذیری Decomposability

- ارزیابی می تواند هدفمند تر شود

معیارهای تست پذیر بودن نرم افزار (ادامه)

۵. سادگی Simplicity

- کاهش پیچیدگی معماری و منطق برنامه

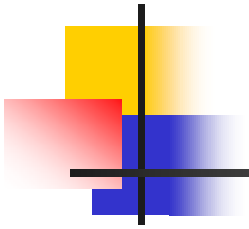
۶. پایداری Stability

- برای ارزیابی تغییرات کمی بخواهد

۷. درک پذیری Understandability

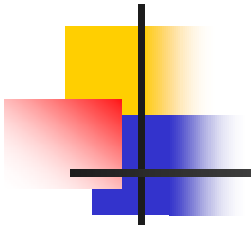
- قابلیت درک طراحی و وابستگیهای بین اجزا

سطوح مختلف تست



- تست واحد (Unit testing)
- تست یکپارچه سازی (Integration testing)
  - تست یکپارچه سازی افزایشی
  - تست یکپارچه سازی
- تست سیستم (System testing)
- تست پذیرش (Acceptance testing)
  - تست آلفا
  - تست بتا

تست واحد



- پایین ترین سطح تست است. (micro level)
- هر کد تست واحد، یک قطعه کد یا یک تابع (متد) خاص را تست می کند.
- این تست نیاز به دانش در مورد طراحی و نحوه عملکرد داخلی تابع یا قطعه کد دارد.
- توسط برنامه نویس (و نه تست کننده) انجام می شود.

تست یکپارچه سازی افزایشی

- با افزوده شدن قابلیت جدید به نرم افزار، مجددا نرم افزار تست می شود.
- هدف این تست، بررسی درستی نرم افزار پس از افزوده شدن امکان جدید است.
- امکانات نرم افزار باید از هم استقلال داشته باشند تا بتوان پیش از تکمیل کل نرم افزار و به صورت افزایشی نرم افزار را تست کرد.
- توسط برنامه نویس یا تیم تست انجام می شود.

تست یکپارچه سازی

- تست نرم افزار حاصل از کنار هم قرار گرفتن قطعات مختلف آن
- به منظور بررسی درستی عملکرد نرم افزار یکپارچه شده
- قطعات مختلف شامل
  - قطعه کدها (ماژول هایی از کد)
  - برنامه های مجزا که در کنار هم برنامه اصلی را تشکیل می دهند
  - برنامه های مشتری-کارگزار عمل کننده در یک شبکه
- پس از تست واحد انجام می شود

تست سیستم

- به منظور بررسی عملکرد نرم افزار بر روی پلتفرم های مختلف انجام می شود
  - پلتفرم: سخت افزار + نرم افزار (شامل OS و نرم افزارهای کاربردی مورد نیاز برنامه)
- به منظور اطمینان از اینکه برنامه با مولفه های دیگر محیط اجرایش به خوبی کار می کند
- به منظور اطمینان از اینکه نرم افزار ارائه شده در محیط مورد نظر قابل استفاده است.

مثالی از مشکل حاصل از انجام ندادن تست سیستم

- **نرم افزار بازی شیر شاه دیزنی** (Disney's Lion King Game)
- در پاییز سال 1994 شرکت دیزنی اولین CD بازی خود تحت عنوان شیر شاه (Lion King) که بر اساس کارتونی به همین نام ساخته شده بود را وارد بازار کرد. بسیاری از شرکتهای دیگر تا آن زمان اقدام به ساخت بازیهای رایانه ای کرده بودند اما این اولین بار بود که شرکت دیزنی وارد این تجارت شده بود. دیزنی برای فروش این بازی دست به تبلیغات گسترده ای زد و در نتیجه این محصول با فروش بسیار بالایی مواجه شد. اما اتفاقات پس از آن تبدیل به کابوسی برای این شرکت شد. در 26 دسامبر، روز پس از کریسمس تلفن های بخش پشتیبانی مشتریان شرکت دیزنی شروع کرد به زنگ زدن و زنگ زدن و زنگ زدن! متصدیان پاسخگویی به تماس ها با خیل عظیمی از والدین عصبانی با بچه های گریان مواجه شدند که ادعا می کرند نرم افزار مزبور کار نمی کند. این خبر به سرعت در مطبوعات و تلویزیون نیز پخش شد و کریسمس آن سال را برای بسیاری از پرسنل دیزنی تلخ کرد.
- علت چه بود؟ پس از بررسی مشخص شد که دیزنی نرم افزار خود را بر روی بسیاری از مدل های PCتست نکرده بود و در نتیجه تنها بر روی سیستمهایی کار می کرد که برنامه نویسان دیزنی روی آن سیستم ها نرم افزار خود را توسعه داده بودند و نه دستگاههای متداولی که عموم مردم از آن استفاده می کردند.
-
- (منبع: Software Testing,Ron Patton, Sams Publishing, 2005)

تست پذیرش

- به منظور بررسی اینکه نرم افزار نیازهای مشتری را برآورده می کند، انجام می شود
- بعد از تست سیستم انجام می شود. شامل:
  - تست آلفا: تست آلفا در سایت توسعه دهنده نرم افزار و در اغلب موارد توسط کارمندان داخلی و در بعضی از موارد توسط مشتری (تعدادی از کاربرانش که به محل دعوت می شوند) انجام می گیرد.
  - تست بتا: در تست بتا نسخه هایی از نرم افزار در اختیار تعدادی از کاربران قرار می گیرد تا در بازه ای با آن کار کنند و خطاها را گزارش دهند.

روشهای ارزیابی

- روش جعبه سفید
  - دانستن نحوه کار داخلی برنامه
  - امکان تایید نحوه عمل هر تکه کد و مسیر اجرا
  - مراحل اولیه ارزیابی
- روش جعبه سیاه
  - دانستن عمل مورد انتظار و مطلوب
  - امکان تایید کاری که سیستم باید انجام دهد
  - مراحل انتهایی ارزیابی

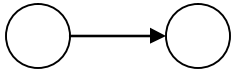
روش تست جعبه سفید

ارزیابی مسیر پایه (Basis Path)

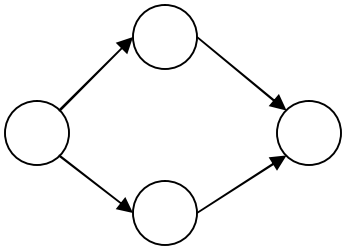
- پیچیدگی سیستم را مشخص می کند
- تمامی مسیرهای اجرا ارزیابی میشوند (هر دستور یکبار اجرا میشود) این کار با کمک گراف جریان انجام می شود
- گراف جریان کنترل
  - گره ها: یک یا چند دستور برنامه
  - کمان ها: جریان برنامه (پیکانهای فلو چارت)
  - کمانها حتما باید از دو طرف به گره ها وصل باشند

گرافها

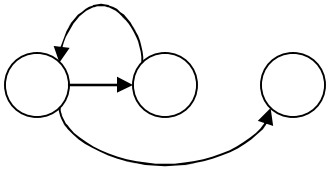
Sequence



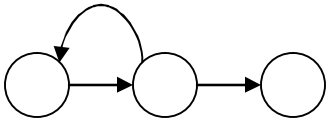
If



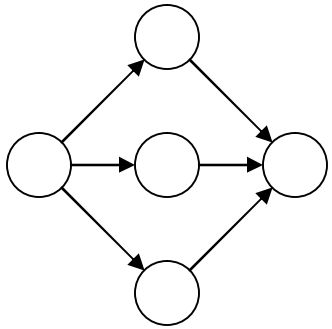
While



Do-while



Case



برخی تعاریف

- **ناحیه** : فضایی که توسط لبه ها و گره ها محدود شده باشد (فضای خارج هم یک ناحیه محسوب می شود)
- **گره مسند** (predicate node): گرهی که دارای یک شرط باشد

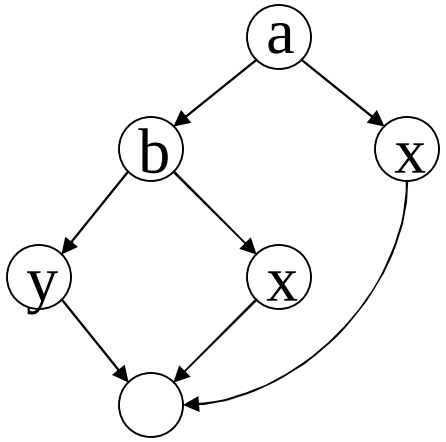
حالت خاص

شروط ترکیبی باید تجزیه شوند

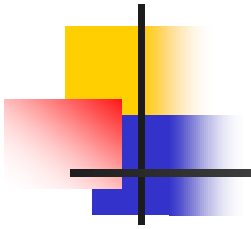
“if a or b then x else y”

نحوه تجزیه:

“if a then x”
else if (b) then x
else y



پیچیدگی سیکلوماتیک



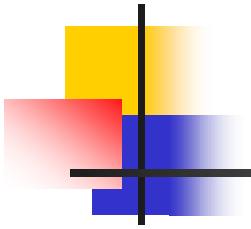
- یک معیار عددی از میزان پیچیدگی منطقی
- تعداد مسیرهای مستقل و لذا حد بالای تعداد تست ها را مشخص می کند
- یک مسیر مستقل مسیری است که یک سری جدید از دستورات یا یک شرط جدید داشته باشد.
- مجموعه پایه (basis set): مجموعه همه مسیرهای مستقل

پیچیدگی سیکلوماتیک

V(G) = تعداد نواحی

V(G) = تعداد گره ها - تعداد لبه ها + 2

V(G) = تعداد گره های مسند + 1

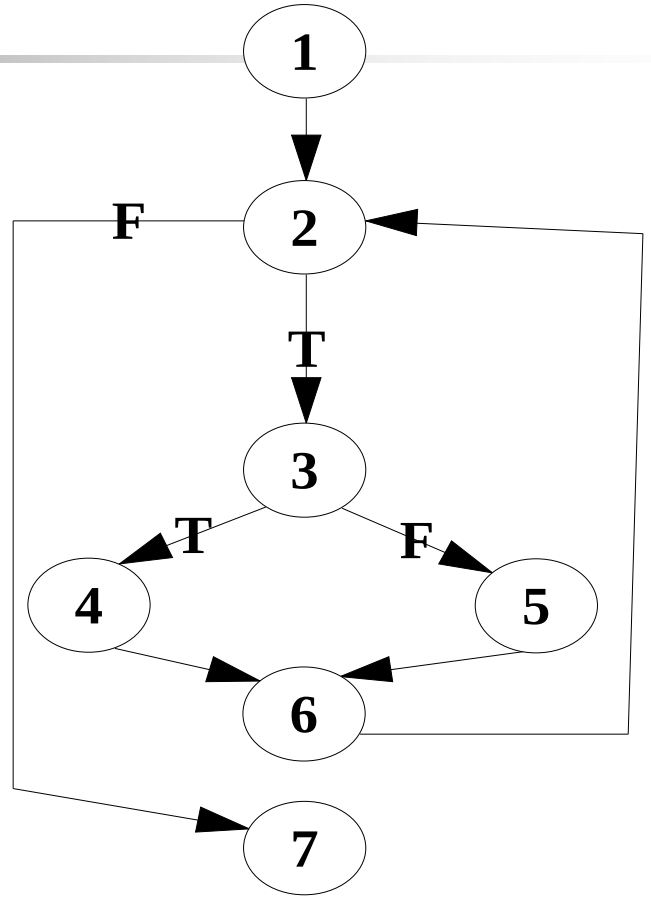


استخراج نمونه های تست

- استفاده از کد و یا دیاگرام
- محاسبه پیچیدگی سیکلوماتیک
- محاسبه مجموعه مسیرهای پایه
- تهیه داده تست برای هر مسیر

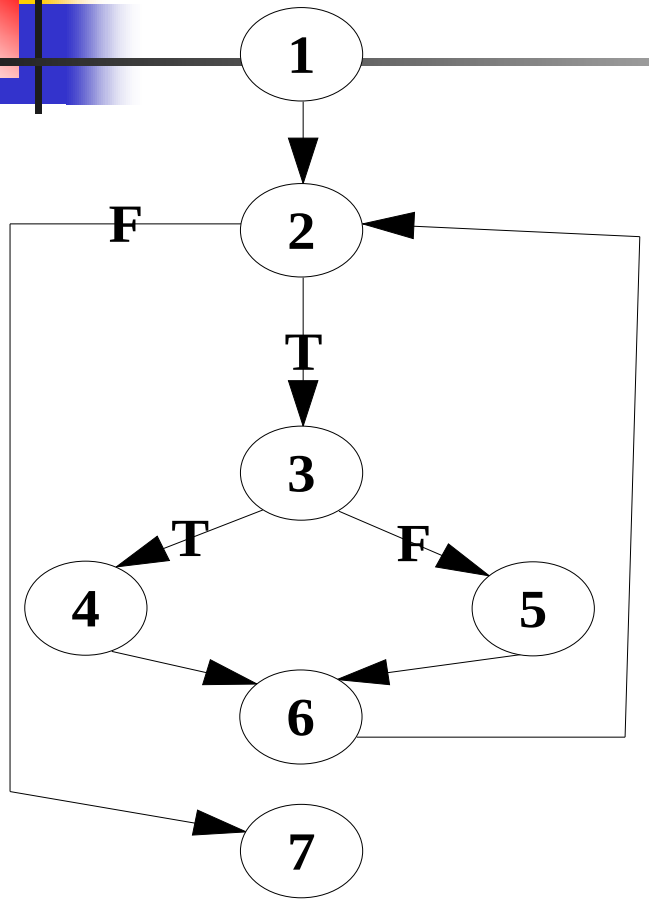
مثال 1

```
(1)        read (i, y)
           a := 1
           b := 1
(2)        while (i < 100) do
           c := i + 1
           z := f()
(3)        if (y < 0)
(4)                  a := 2
                     b := 2
           else
(5)                  a := 3
                     b := 3
           endif
(6)        i := i + 1
           y := y + c
      endwhile
(7)        print(a,b)
```



V(G) = 3

مثال 1-ادامه



- مجموعه مسیر های پایه
  - ۱, ۲, ۷
  - ۱, ۲, ۳, ٤, ٦, ۲, ..., ۷
  - ۱, ۲, ۳, ٥, ٦, ۲, ..., ۷

مثال 1-ادامه

Test case 1, purpose: skip the loop

inputs: i >= 100; any value for y

expected results: a = 1; b = 1

Path covered: 1, 2, 7

Test case 2, purpose: execute loop with y < 0

inputs: i < 100; y < 0

expected results: (a = 2 or a = 3) and (b = 2 or b = 3)

Path covered: 1, 2, 3, 4, 6, 2, ..., 7

Test case 3, purpose: execute loop with y >= 0

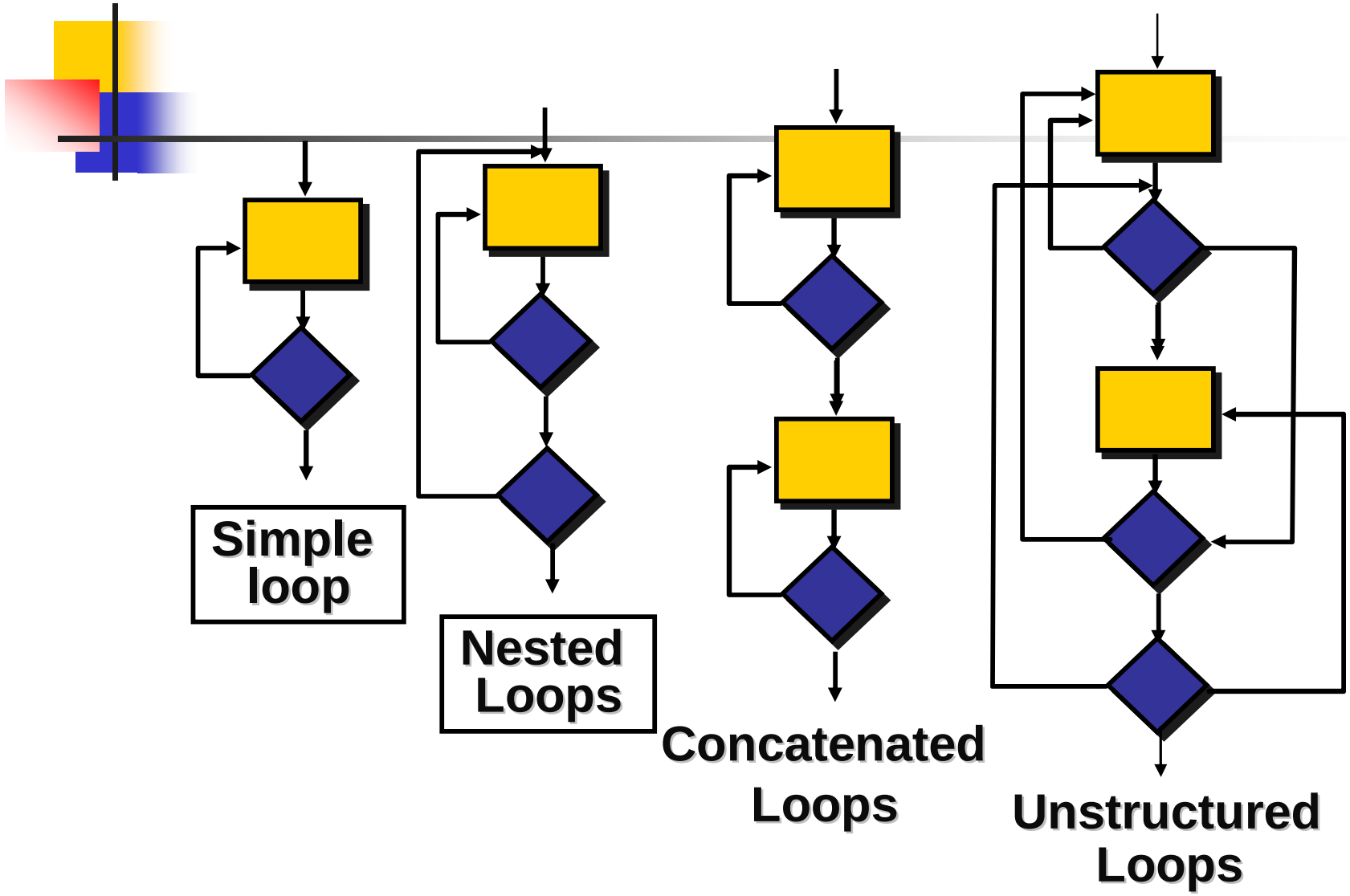
inputs: i < 100; y >= 0

expected results: (a = 2 or a = 3) and (b = 2 or b = 3)

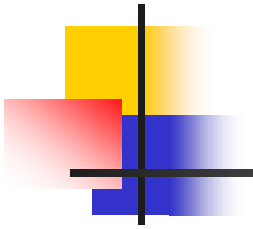
Path covered: 1, 2, 3, 5, 6, 2, ..., 7

```
(1)        read (i, y)
           a := 1
           b := 1
(2)        while (i < 100)
     c := i + 1
           z := f()
(3)        if (y < 0)
(4)                  a := 2
                     b := 2
           else
(5)                  a := 3
                     b := 3
           endif
(6)        i := i + 1
           y := y + c
     endwhile
(7)        print(a,b)
```

تست حلقه


Simple loop
Nested Loops
Concatenated Loops
Unstructured Loops

تست حلقه های ساده

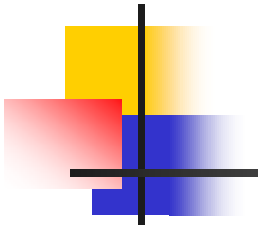


Minimum conditions—Simple Loops

1. **skip the loop entirely**
2. **only one pass through the loop**
3. **two passes through the loop**
4. **m passes through the loop m < n**
5. **(n-1), n, and (n+1) passes through the loop**

where n is the maximum number of allowable passes

تست حلقه های تودرتو



Nested Loops

Start at the innermost loop. Set all outer loops to their minimum iteration parameter values.

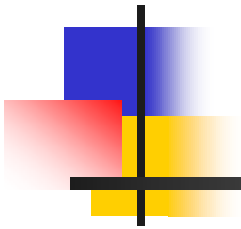
Test the min+1, typical, max-1 and max for the innermost loop, while holding the outer loops at their minimum values.

Move out one loop and set it up as in step 2, holding all other loops at typical values. Continue this step until the outermost loop has been tested.

Concatenated Loops

If the loops are independent of one another
then treat each as a simple loop
else* treat as nested loops
endif*

for example, the final loop counter value of loop 1 is used to initialize loop 2.

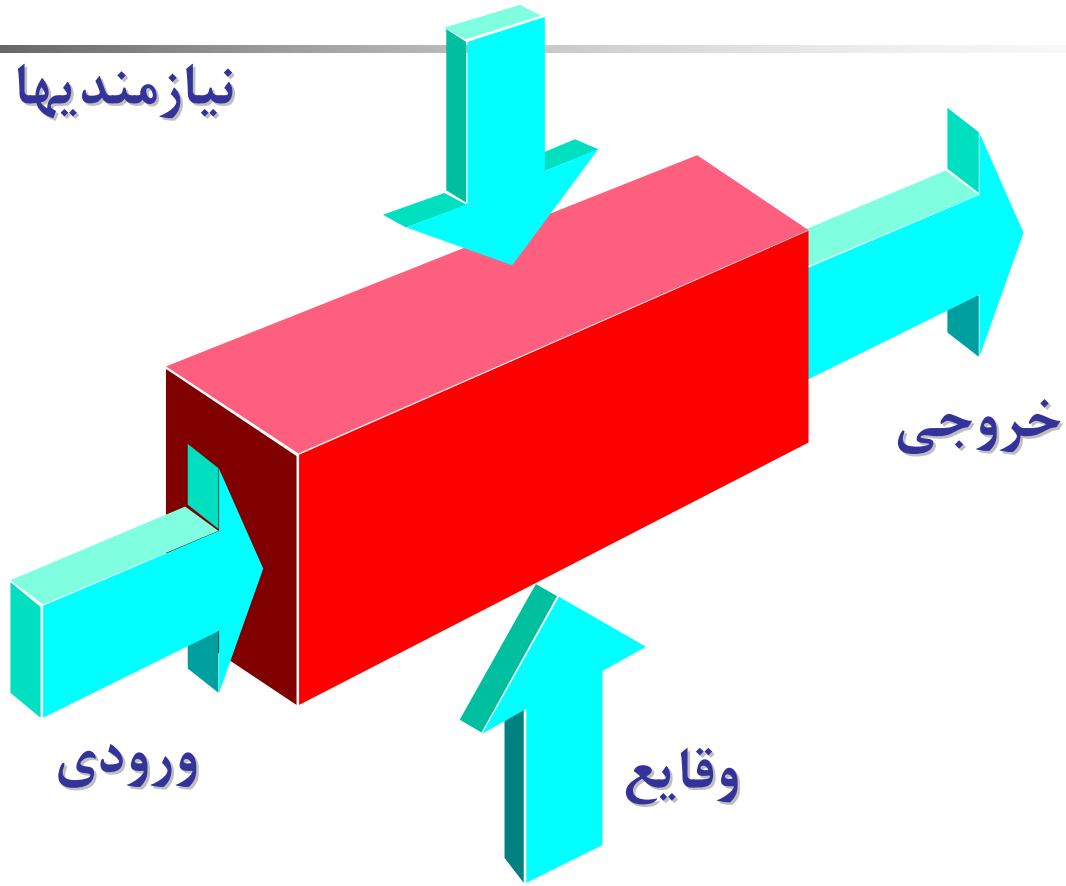


روش تست جعبه سیاه

تست جعبه سیاه یاتست رفتاری نرم افزار

- برخلاف تست جعبه سفید: از هیچ دانشی درباره نحوه رفتار کد استفاده نمی کند. مکمل تست جعبه سفید است.
- نمونه های تست براساس توصیف (specification) سیستم طراحی می شوند.
- این تست برخواسته های عملیاتی نرم افزار تکیه دارد.
- مثال: جستجوی یک مقدار در یک آرایه
  - پس شرط: مقدار برگشتی اندیس محل وقوع مقدار در آرایه است یا 1- در صورتی که این مقدار در آرایه نباشد.

تست جعبه سیاه



روش افراز هم ارزی (Equivalence Partitioning)

- دامنه ورودیها و خروجی ها را در نظر گرفته و آنها را به کلاسهای معادل دسته بندی می کنیم.
- برای مقادیر مختلف مربوط به یک کلاس، نرم افزار لازم است به صورت مشابه عمل کند
- هر کلاس داده یک رده از خطا ها را می پوشاند
- مقادیری از هر یک از کلاس ها انتخاب شده و تست می شوند

- مثال: برای ورودی در محدوده 2..5:
  - کلاس 1: اعداد کوچکتر از 2
  - کلاس 2: اعداد بین 2 و 5
  - کلاس 3: اعداد بزرگتر از 5

- احتمال کشف خطا در اجرای تست با مقادیری از کلاس های مختلف بیشتر از اجرای آن با مقادیری از یک کلاس داده است.

کلاس های هم ارزی (Equivalence Classe)

- مثال ها:
  - ورودی x در بازه [a..b] : سه کلاس x<a, a<=x<=b , x> b
  - یک مقدار boolean: دو کلاس false , true
  - می توان کلاس هایی برای ورودی های نامعتبر در نظر گرفت.
- انتخاب مقادیر تست:
  - یک مقدار معمولی در وسط کلاس (های)ی که مقادیر معتبر ورودی را شامل می شوند، انتخاب کنید.
  - موارد آزمون طوری انتخاب می شوند که بیشترین تعداد از صفات یک طبقه هم ارزی یکباره آزمایش شوند

روش بررسی مقادیر مرزی

در این روش تمرکز تست بر روی بررسی مقادیر مرزی است. تعداد خطاهای موجود در مرزهای دامنه ورودی نسبت به مقادیر مرکزی دامنه بیشتر است.

مثال:

- برای ورودی در بازه [a..b]: انتخاب ها: a-1، a، a+1 ، b-1، b، b+1
- Spec says that the code accepts between 4 and 24 inputs ; each is a 3-digit integer
  - One partition: number of inputs
    - Classes “x<4”, “4<=x<=24”, “24<x”
    - Chosen values: 3,4,5,14,23,24,25
  - Another partition: integer values
    - Classes: “x<100”, “100<=x<=999”, “999<x”
    - Chosen values: 99,100,101,500,998,999,1000

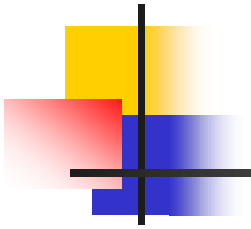
روش بررسی مقادیر مرزی (ادامه)

- به طور مشابه برای خروجی نیز مقادیر مرزی بررسی می شود

- Spec: the output is between 3 and 6 integers, each in the range 1000-2500

- Try to design inputs that produce
  - 3 outputs with value 1000
  - 3 outputs with value 2500
  - 6 outputs with value 1000
  - 6 outputs with value 2500

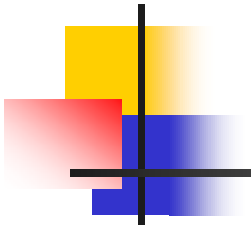
مثال: جستجو در آرایه



- Search for a value in an array
  - Return: index of some occurrence of the value, or -1 if the value does not occur
- One partition: size of the array
  - Programmer errors are often made for size 1: a separate equivalence class
  - Classes: “empty array”, “array with one element”, “array with many elements”
- Another partition: location of the value
  - “first element”, “last element”, “middle element”, “not found”

مثال: جستجو در آرایه (ادامه)

Array	Value	Output
empty	5	-1
[7]	7	0
[7]	2	-1
[1,6,4,7,2]	1	0
[1,6,4,7,2]	4	2
[1,6,4,7,2]	2	4
[1,6,4,7,2]	3	-1



Testing Levels/ Techniques	White Box	Black Box	Incremental
Unit Testing	X	-	-
Integration Testing	X	-	X
System Testing	-	X	-
Acceptance Testing	-	X	-

V & V

- تست نرم افزار، یکی از عناصر یک موضوع وسیع تر است که از آن با نام بازبینی و اعتبارسنجی (V&V) یاد می شود.

- Verification :Are you building it right ?
- بازبینی (وارسی): آیا محصول را به درستی می سازیم؟
  - عبارت از مجموعه فعالیت هایی است که پیاده سازی صحیح یک عملکرد خاص توسط نرم افزار را تضمین می کند.

- Validation: Are you building the right thing?
- اعتبار سنجی: آیا محصول درستی را می سازیم؟
  - عبارت از مجموعه متفاوتی از فعالیت هاست که تضمین می کند نرم افزار ساخته شده با خواست مشتری مطابقت دارد

Validation در مقایسه با Verification

- در آوریل 1990، تلسکوپ فضایی هابل به مدار زمین فرستاده شد. هدف اصلی این تلسکوپ فضایی، بزرگنمایی اهداف مورد مشاهده با استفاده از یک آینه بزرگ بود. تولید آینه مزبور امری دشوار و نیازمند صحت و دقت بسیار بالایی بود. آزمایش این آینه از این جهت که قرار بود محیط واقعی آن در فضا باشد، در محل ساخت آن در زمین بسیار مشکل می نمود. در نتیجه تنها راه تست، اندازه گیری دقیق مشخصات محصول ساخته شده و مقایسه آن با مشخصات تعیین شده برای محصول مورد نظر بود. این آزمایشات با موفقیت صورت گرفته و هابل آماده پرتاب اعلام شد.
- متاسفانه به محض عملیاتی شدن این تلسکوپ مشخص شد که تصاویر ارسال شده توسط آن به هیچ وجه فوکوس نیست. در تحقیقات صورت گرفته مشخص شد که ساخت آینه به درستی صورت نگرفته است. آینه مزبور بر اساس مشخصات تعیین شده (Specifications) تولید شده بود اما این مشخصات نادرست بوده است. این آینه به طور کامل دقیق (Precise) ساخته شده بود اما از صحت لازم (Accuracy) برخوردار نبود. آزمایشات تایید کرده بود که آینه مزبور مشخصات لازم را تامین کرده است (Verification) اما درستی خود مشخصات اولیه را مورد بررسی قرار نداده بود (Validation). به عبارت دیگر Verification به فرآیند مطابقت محصول با مشخصات آن گفته می شود اما Validation فرآیند مقایسه مشخصات محصول با نیازمندی های واقعی کاربر است.
- هر چند مثال بالا مثالی خارج از حوزه نرم افزار است اما به طور دقیق قابل نگاشت به محصولات نرم افزاری می باشد. در نتیجه هرگز تصور نکنید که در صورت مطابقت نرم افزار تولید شده با مشخصات آن، محصول مزبور بدون عیب است. بلکه بخش مهم دیگر در آزمایش نرم افزار، مطابقت مشخصات محصول با خواسته واقعی کاربر آن است.