Introduction to Software Testing
(*2nd edition*)
Chapter 2

Model-Driven Test Design

Paul Ammann & Jeff Offutt

http://www.cs.gmu.edu/~offutt/softwaretest/

Updated September 2016

Complexity of Testing Software

- No other engineering field builds products as complicated as software

در عوض، سعی کنید »رفتار« نرم افزار را ارزیابی کنید تا تصمیم بگیرید که آیا رفتار با توجه به تعداد زیادی از عوامل قابل قبول است یا خیر.
بدیهی است که این پیچیده تر است.
یک راه حل این است که سطح انتزاع خود را بالا ببریم"

- The term correctness has no meaning
  - Is a building correct?
  - Is a car correct?
  - Is a subway system correct?

هیچ رشته مهندسی دیگری به اندازه نرم افزار محصولی پیچیده نمی سازد
اصطلاح صحت معنا ندارد-
آیا ساختمان درست است؟
- ماشین درسته؟-
آیا سیستم مترو درست است؟

- Instead, try to evaluate software's "behavior" to decide if the behavior is acceptable within consideration of a large number of factors.
- Obviously, this is more complex.
- One solution is to "raise our level of abstraction. "

اگر طراحان بتوانند سطح انتزاع خود را بالا ببرند، کارآمدتر و مؤثرتر هستند.
مانند سایر مهندسان، ما باید از انتزاع برای مدیریت پیچیدگی استفاده کنیم
- این هدف از فرآیند طراحی تست مدل محور است
- مدل« یک ساختار انتزاعی است

Designers are more efficient and effective if they can

raise their level of abstraction.

- Like other engineers, we must use abstraction to manage complexity
  - This is the purpose of the model-driven test design process
  - The "model" is an abstract structure

Model-Driven Test Design (MDTD) process(1)

- Breaks testing into a series of small tasks that simplify test generation.

تست را به یک سری کارهای کوچک تقسیم می کند که تولید تست را ساده می کند.

- Then test designers isolate their task, and work at a higher level of abstraction by using mathematical engineering structures to design test values independently of the details of software or design artifacts, test automation, and test execution.

سپس طراحان آزمون وظیفه خود را جدا میکنند و با استفاده از ساختارهای مهندسی ریاضی برای طراحی مقادیر آزمون مستقل از جزئیات نرم افزار یا مصنوعات طراحی، اتوماسیون تست و اجرای آزمون، در سطح بالاتری از انتزاع کار میکنند.

The Model-Driven Test Design (MDTD) process(1)

- A key intellectual step in MDTD is test case design.
- Test case design can be the primary determining factor in whether tests successfully find failures in software.

یک مرحله فکری کلیدی در MDTD طراحی مورد آزمایشی است.طراحی کیس آزمایشی میتواند عامل اصلی تعیینکننده در یافتن موفقیت آمیز اینکه آیا تستها به خوبی فیلیورها را کشف کردند یا نه در نرمافزار باشد.

- Tests can be designed with a "human-based" approach.

آزمون ها را می توان با رویکرد »مبتنی بر انسان« طراحی کرد.

  - A test engineer uses domain knowledge of the software's purpose and his or her experience to design tests that will be effective at finding faults.
- Alternatively, tests can be designed to satisfy well-defined engineering goals such as coverage criteria.

یک مهندس تست از دانش دامنه در مورد هدف نرم افزار و تجربه خود برای طراحی آزمایش هایی استفاده می کند که در یافتن عیوب موثر باشد.
از طرف دیگر، تست ها را میتوان برای برآورده کردن اهداف مهندسی به خوبی تعریف شده مانند معیارهای پوشش طراحی کرد.

Software Testing Foundations (1)

Testing can only show the presence of failures,

Not their absence

آزمایش فقط می تواند وجود شکست ها را نشان دهد، نه عدم وجود آنها را

Software Testing Foundations (2)

- the problem of finding all failures in a program is undecidable.

- Testers often call a test successful (or effective) if it finds an error.

مساله ی یافتن تمام فیلیور ها در یک برنامه غیرقابل تصمیم گیری است.
تسترها معمولاً در صورت یافتن خطا، آزمایشی را موفق (یا مؤثر) مینامند.

Software Faults, Errors & Failures

- Software Fault : A static defect in the software
- Software Error : An incorrect internal state that is the manifestation of some fault
- Software Failure : External, incorrect behavior with respect to the requirements or other description of the expected behavior

فالت نرم افزار: یک نقص استاتیک در نرم افزار
ارور نرم افزار: یک حالت داخلی نادرست که مظهر ونشانه ی فالت است
فیلیور نرم افزار: رفتار خارجی و نادرست با توجه به الزامات یا سایر توضیحات رفتار مورد انتظار

Testing & Debugging

- Testing : Evaluating software by observing its execution

تست: ارزیابی نرم افزار با مشاهده نحوه اجرای آن

- Test Failure : Execution of a test that results in a software failure

Test Failure : اجرای آزمایشی که منجر به خرابی نرم افزار می شود

- Debugging : The process of finding a fault given a failure

اشکال زدایی: فرآیند یافتن عیب در صورت شکست

for a given fault, not all inputs will “trigger” the fault into creating incorrect output (a failure)

برای یک خطای معین، همه ورودیها باعث ایجاد خطا در ایجاد خروجی نادرست (شکست) نمیشوند.

A Concrete Example

Fault: Should start searching at 0, not 1

```
public static int numZero (int [ ] arr)
{  // Effects: If arr is null throw NullPointerException
   // else return the number of occurrences of 0 in arr
   int count = 0;
   for (int i = 1; i < arr.length; i++)
   {
      if (arr [ i ] == 0)
      {
         count++;
      }
   }
   return count;
}
```

Test 1
[2, 7, 0]
Expected: 1
Actual: 1

Error: i is 1, not 0, on the first iteration
Failure: none

Test 2
[0, 2, 7]
Expected: 1
Actual: 0

Error: i is 1, not 0
Error propagates to the variable count
Failure: count is 0 at the return statement

So,

- it is often very difficult to relate a failure to the associated fault.

- Analyzing these ideas leads to the fault/failure model, which states that four conditions are needed for a failure to be observed.

اغلب بسیار دشوار است که یک شکست را به خطای مرتبط مرتبط کنیم.
تجزیه و تحلیل این ایدهها منجر به مدل خطا/شکست میشود که بیان میکند چهار شرط برای مشاهده شکست لازم است.

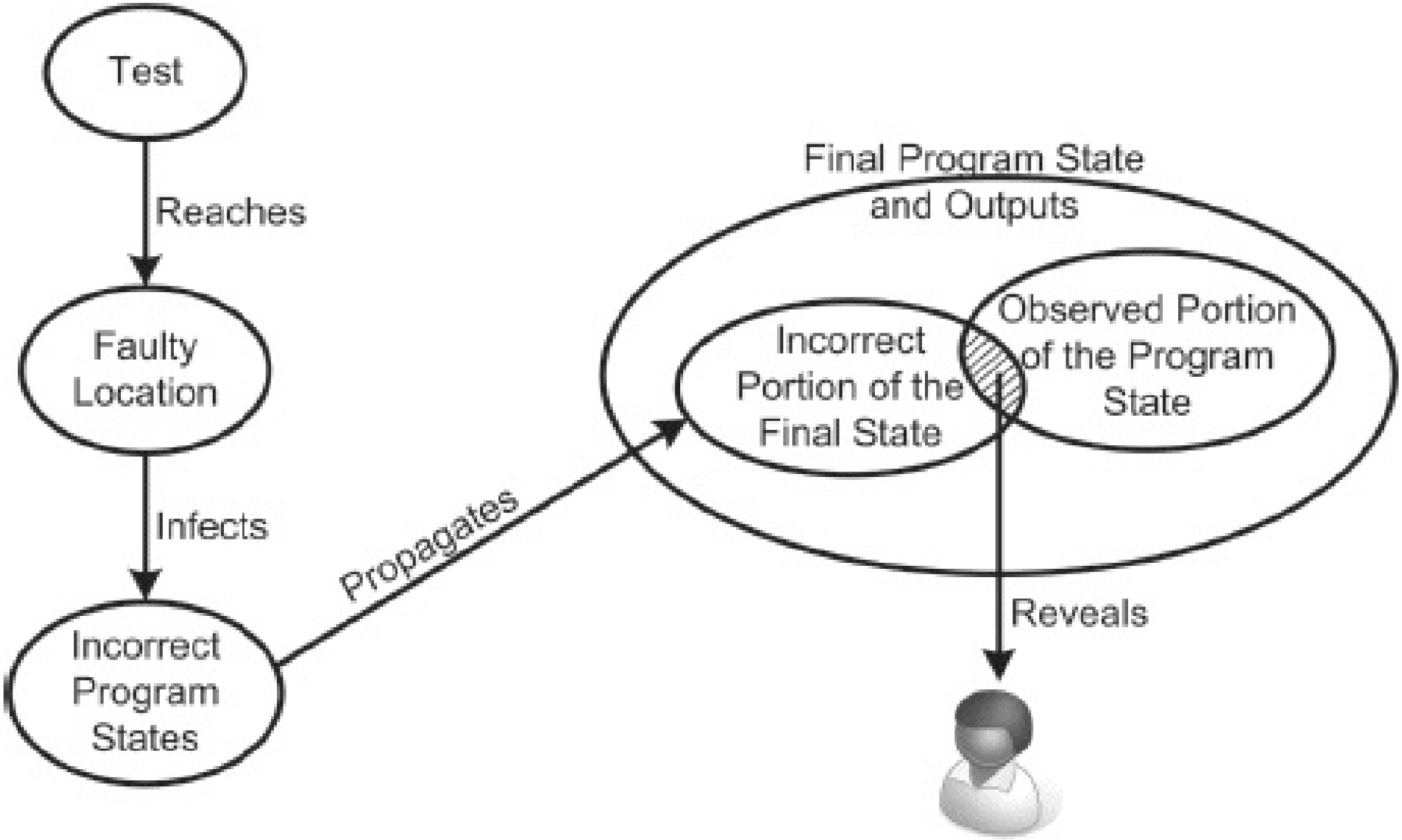
Fault & Failure Model (RIPR)

Four conditions necessary for a failure to be observed

1. Reachability : The location or locations in the program that contain the fault must be reached
2. Infection : The state of the program must be incorrect
3. Propagation : The infected state must cause some output or final state of the program to be incorrect
4. Reveal : The tester must observe part of the incorrect portion of the program state

1. قابلیت دسترسی: باید به مکان یا مکان هایی در برنامه که حاوی خطا هستند دسترسی داشت
2. عفونت: وضعیت برنامه باید نادرست باشد
3. انتشار: حالت آلوده باید باعث شود برخی خروجی ها یا وضعیت نهایی برنامه نادرست باشد.
4. آشکار: آزمایشگر باید بخشی از قسمت نادرست وضعیت برنامه را مشاهده کند

RIPR Model


Test
Reaches
Faulty Location
Infects
Incorrect Program States
Propagates
Final Program State and Outputs
Incorrect Portion of the Final State
Observed Portion of the Program State
Reveals

Is RIPR model can detect missing code?

- The RIPR model applies even when the fault is missing code (so-called faults of omission).
- when execution passes through the location where the missing code should be, the program counter, which is part of the program state, necessarily has the wrong value.

آیا مدل RIPR می تواند کد گم شده را تشخیص دهد؟
مدل RIPR حتی زمانی که خطا فاقد کد باشد (به اصطلاح خطاهای حذف) اعمال می شود.
هنگامی که اجرا از محلی می گذرد که کد گم شده باید باشد، شمارنده برنامه، که بخشی از وضعیت برنامه است، لزوماً دارای اشتباه است.

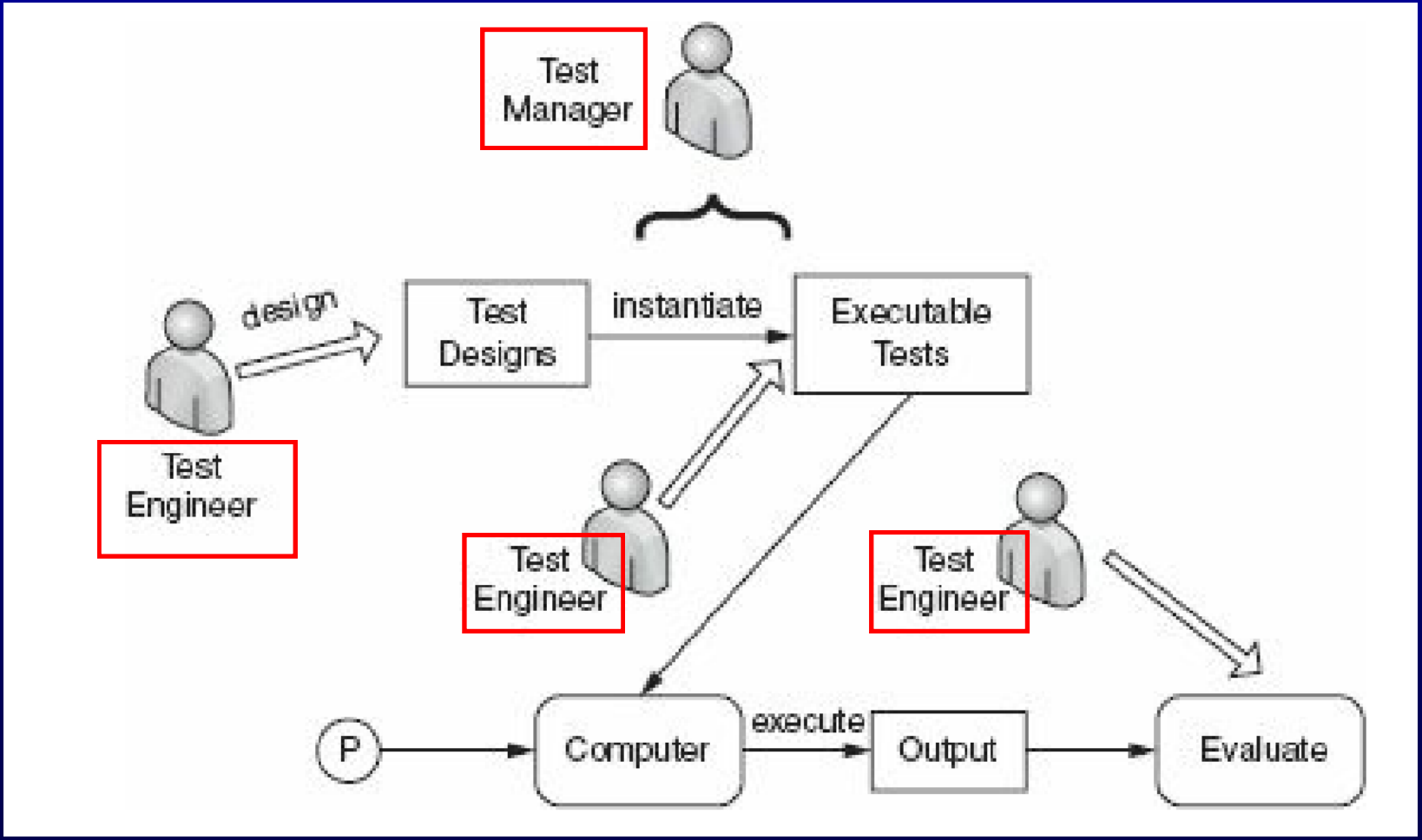
Software Testing Activities (2.2)

- Test Engineer : An IT professional who is in charge of one or more technical test activities
  - Designing test inputs
  - Producing test values
  - Running test scripts
  - Analyzing results
  - Reporting results to developers and managers
  - every engineer involved in software development can wear the hat of a test engineer. Because the person best positioned to define these test cases is often the designer of the artifact.

مهندس تست: یک متخصص فناوری اطلاعات که مسئول یک یا چند فعالیت تست فنی است
– طراحی ورودی های تست
- تولید مقادیر تست
– اجرای اسکریپت های تست
- تجزیه و تحلیل نتایج
- گزارش نتایج به توسعه دهندگان و مدیران

- هر مهندس درگیر در توسعه نرم افزار می تواند کلاه یک مهندس آزمایشی را بر سر بگذارد.
زیرا شخصی که بهترین موقعیت را برای تعریف این موارد تست دارد، اغلب طراح مصنوع است.

Activities os Test Engineers



Important points

هدف اصلی-
طراحی تست هایی که به طور سیستماتیک طبقات مختلف خطاها را با حداقل زمان و تلاش آشکار می کند
- یک تست خوب احتمال خطایابی بالایی دارد
- یک آزمایش موفق یک خطا را آشکار می کند

- **Main objective**
  - Design tests that systematically uncover different classes of errors with a minimum amount of time and effort
  - A good test has a high probability of finding an error
  - A successful test uncovers an error

- **Secondary benefits**
  - Demonstrate that software appears to be working according to specification (functional and non-functional)
  - Data collected during testing provides indication of software reliability and software quality
  - Good testers clarify the specification (creative work)

- **Testing is done best by independent testers.**

مزایای ثانویه
- نشان میدهد که به نظر می رسد نرم افزار مطابق با مشخصات کار می کند (عملکردی و غیر کاربردی)
- دادههای جمع آوریشده در طول آزمایش، نشاندهنده قابلیت اطمینان و کیفیت نرم افزار است.

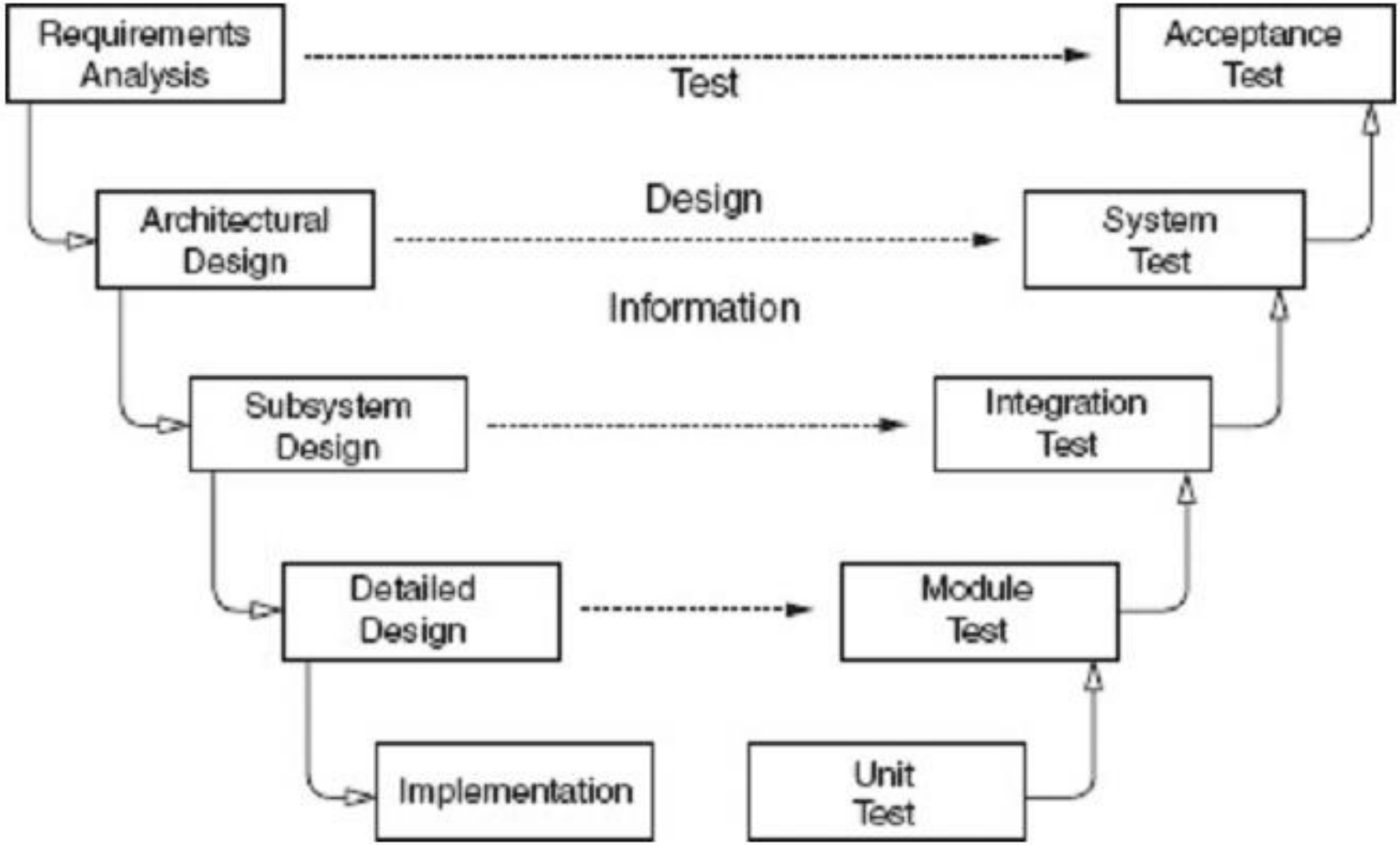
- آزمایش کننده های خوب مشخصات را روشن می کنند (کار خلاقانه)
تست به بهترین وجه توسط آزمایش کنندگان مستقل انجام می شود.

Software Testing Activities (2.2)

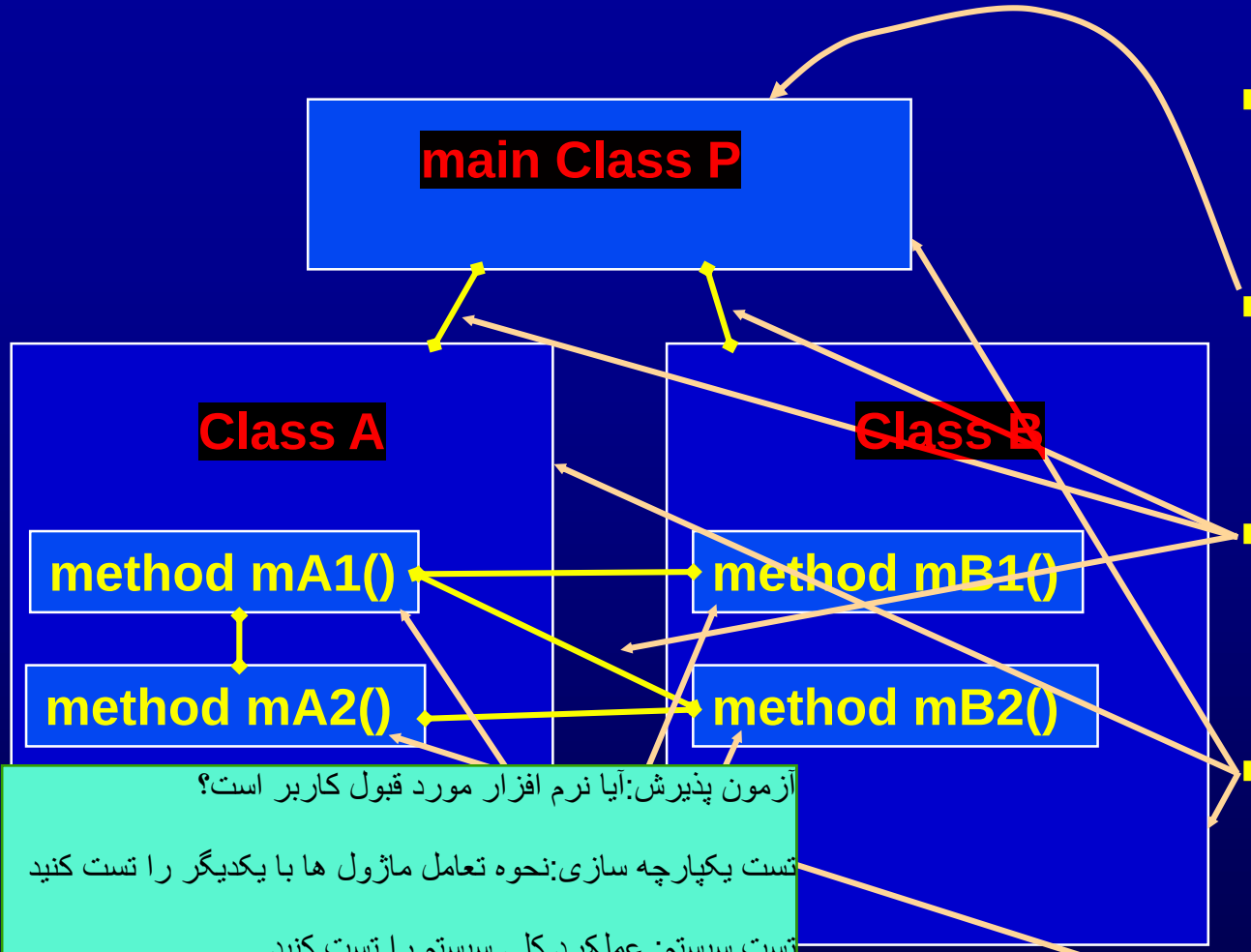
- Test Manager : In charge of one or more test engineers
  - Sets test policies and processes
  - Interacts with other managers on the project
  - Otherwise supports the engineers

مدیر آزمون: مسئول یک یا چند مهندس آزمون
- سیاست ها و فرآیندهای تست را تنظیم می کند
- با سایر مدیران پروژه تعامل دارد
- در غیر این صورت از مهندسین پشتیبانی می کند

Software development activities and testing levels – the "V Model"



Traditional Testing Levels (2.3)



- **Acceptance testing** : Is the software **acceptable** to the **user**?
- **System testing** : Test the **overall functionality** of the system
- **Integration testing** : Test how **modules interact** with each other
- **Module testing (developer testing)** : Test each **class**, **file**, **module**, **component**
- **Unit testing (developer testing)** : Test **each unit (method) individually**

آزمون پذیرش:آیا نرم افزار مورد قبول کاربر است؟

تست یکپارچه سازی:نحوه تعامل ماژول ها با یکدیگر را تست کنید

تست سیستم: عملکرد کلی سیستم را تست کنید

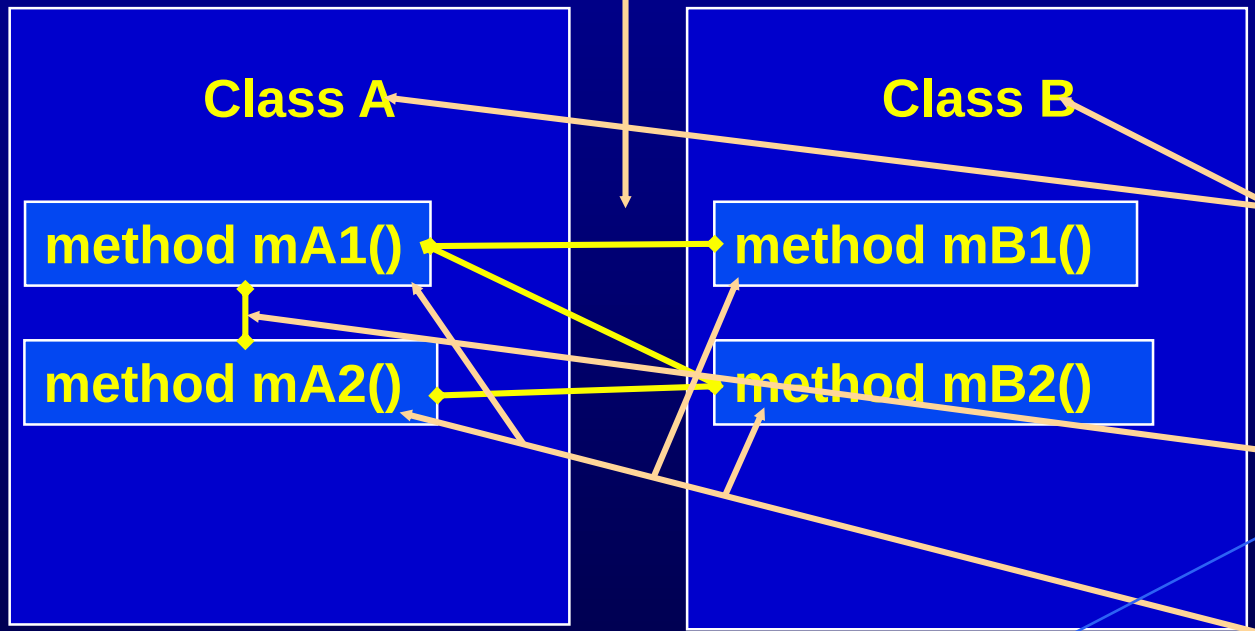
تست ماژول (تست توسعه دهنده): هر کلاس، فایل، ماژول، کامپوننت را تست کنید

تست واحد (تست توسعه دهنده):هر واحد (متد) را جداگانه تست کنید

Object-Oriented Testing Levels

تست بین کلاسی: چندین کلاس را با هم آزمایش کنید

Inter-class testing : Test multiple classes together



تست درون کلاسی:کل کلاس را به عنوان دنباله ای از فراخوانی ها آزمایش میکند

Intra-class testing : Test an entire class as sequences of calls

Inter-method testing : Test pairs of methods in the same class

Intra-method testing : Test each method individually

ست بین متدی:جفت متدها را در یک کلاس آزمایش کنید

تست درون متدی:هر متد را جداگانه تست کنید

Unit Testing

- Testing individual subsystems (collection of classes)
- Goal: Confirm that subsystem is correctly coded and carries out the intended functionality.
- To achieve a reasonable test coverage, one has to test each method with several inputs
  - To cover valid and invalid inputs
  - To cover different paths through the method

- آزمایش زیرسیستم های فردی و منحصربه فرد(مجموعه ای از کلاس ها)
- هدف: تأیید کنید که زیرسیستم به درستی کدگذاری شده است و عملکرد مورد نظر را انجام می دهد.

- برای دستیابی به پوشش تست معقول، باید هر روش را با چندین ورودی آزمایش کرد
  - برای پوشش ورودی های معتبر و نامعتبر
  - برای پوشش مسیرهای مختلف از طریق متد

This slide is brought from “Software Architecture and Engineering Testing”, Dr. P. Muller, ETH Zurich: 2018.

Module Testing

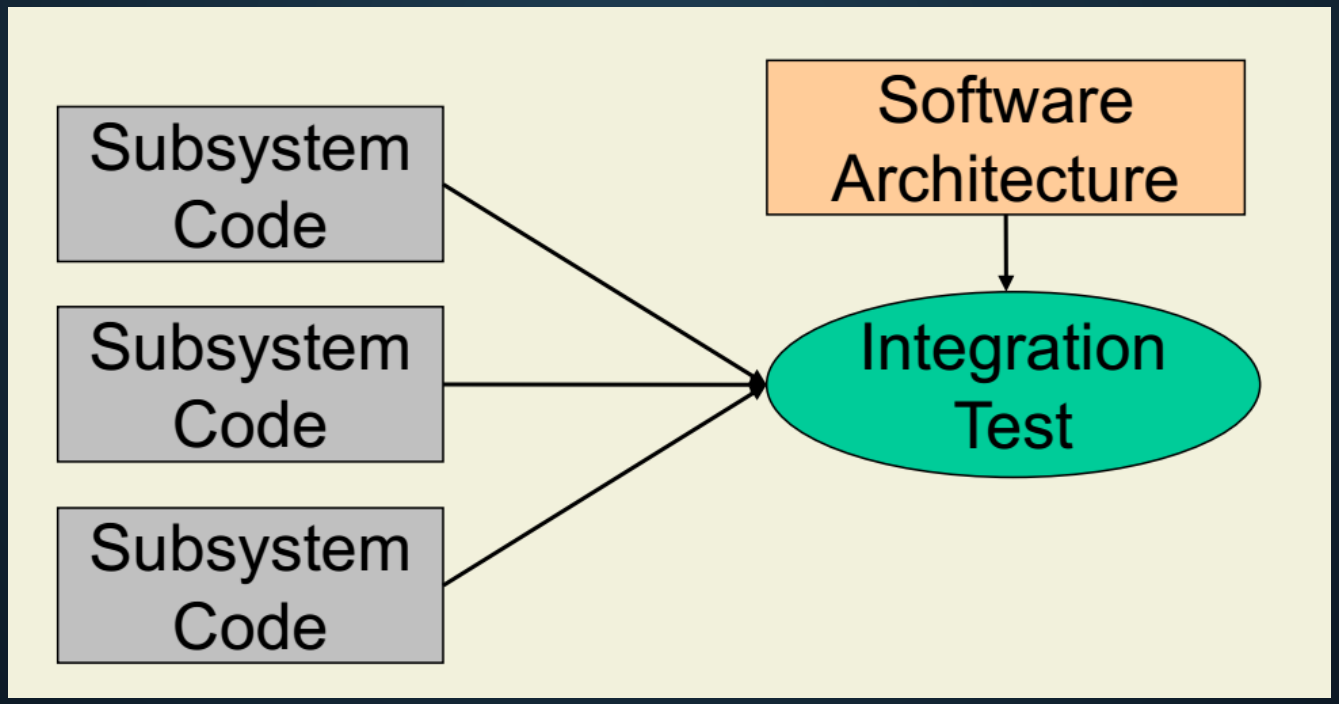
- Testing elements of each modules(classes, component, …)
- Goal: Confirm that module is correctly coded and carries out the intended functionality.

•تست عناصرهر ماژول (کلاس ها، مؤلفه ها، …)
• هدف: تأیید کنید که ماژول به درستی کدنویسی شده است و عملکرد مورد نظر را انجام می دهد.

Integration Testing

- Testing groups of subsystems and eventually the entire system
- Goal: Test interfaces between subsystems

- آزمایش گروه های زیرسیستم ها و در نهایت کل سیستم
- هدف: تست روابط بین زیرسیستم ها

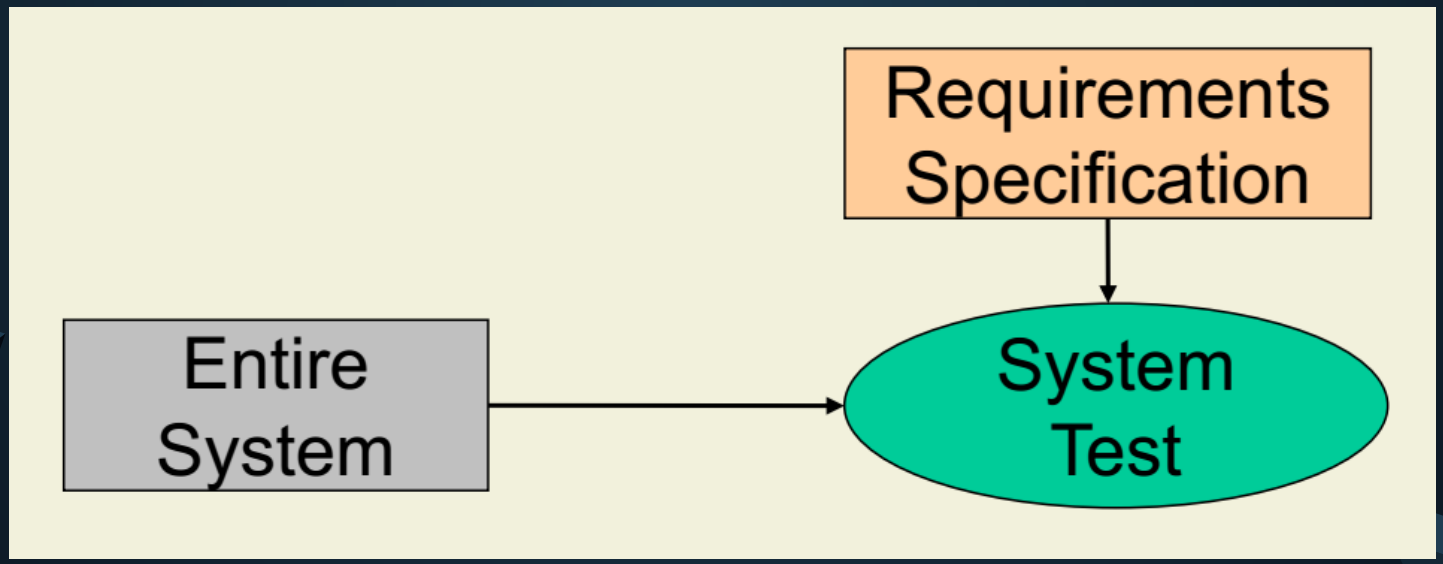


This slide is brought from “Software Architecture and Engineering Testing”, Dr. P. Muller, ETH Zurich: 2018.

System Testing

- تست کل سیستم
- هدف: تعیین اینکه آیا سیستم الزامات (عملکردی و غیرعملکردی) را برآورده می کند یا خیر.

- Testing the entire system
- **Goal: Determine if the system meets the requirements (functional and non-functional)**



This slide is brought from “Software Architecture and Engineering Testing”, Dr. P. Muller, ETH Zurich: 2018.

Acceptance Testing

- هدف: نشان دادن اینکه سیستم با نیازهای مشتری مطابقت دارد و آماده استفاده است
- توسط مشتری انجام می شود، نه توسط توسعه دهنده

- **Goal:** Demonstrate that the system meets customer requirements and is ready to use
- Performed by the client, not by the developer

تست آلفا
- مشتری از نرم افزار در سایت توسعه دهنده استفاده می کند
- نرم افزار مورد استفاده در یک تنظیمات کنترل شده، با توسعه دهنده آماده برای رفع اشکالات

- **Alpha test**
  - Client uses the software at the developer's site
  - Software used in a controlled setting, with the developer ready to fix bugs
- **Beta test**
  - Conducted at client's site (developer is not present)
  - Software gets a realistic workout in target environment

تست بتا
- در سایت مشتری انجام شد (توسعه دهنده حضور ندارد)
- نرم افزار تمرینی واقع بینانه در محیط هدف می گیرد

Regression testing

- a standard part of the maintenance phase of software development.
- is done after changes are made to the software, to help ensure that the updated software still possesses the functionality it had before the updates.
- Testing that everything that used to work still works after changes are applied to the system

تست رگرسیون
بخشی استاندارد از مرحله تعمیر و نگهداری توسعه نرم افزار.
پس از ایجاد تغییرات در نرم افزار انجام می شود تا اطمینان حاصل شود که نرم افزار به روز شده همچنان عملکردی را که قبل از به روز رسانی داشته است دارد.
آزمایش اینکه همه چیزهایی که قبلاً کار میکردند، پس از اعمال تغییرات در سیستم همچنان کار میکنند

Coverage Criteria (2.4)

- Even small programs have too many inputs to fully test them all
  - private static double computeAverage (int A, int B, int C)
  - On a 32-bit machine, each variable has over 4 billion possible values
  - Over 80 octillion possible tests!!
  - Input space might as well be infinite
- Testers search a huge input space
  - Trying to find the fewest inputs that will find the most problems
- Challenges
  - How do we search?
  - When do we stop?

حتی برنامه های کوچک هم ورودی های زیادی دارند تا همه آنها را به طور کامل آزمایش کنند

در یک ماشین 32 بیتی، هر متغیر بیش از 4 میلیارد مقدار ممکن دارد.
- بیش از 80 اکتیلیون تست ممکن!!
- فضای ورودی نیز ممکن است بی نهایت باشد
آزمایشکنندگان فضای ورودی بزرگی را جستجو میکنند.
تلاش برای یافتن کمترین ورودی که بیشترین مشکل را پیدا کند

چالش ها
- چگونه جستجو کنیم؟
چه زمانی متوقف می شویم؟

Coverage Criteria (2.4)

- Coverage criteria give structured, practical ways to search the input space, how to search and when to stop.
  - Search the input space thoroughly, especially in the corners
  - Not much overlap in the tests

معیارهای پوشش روش های ساختاریافته و عملی برای جستجوی فضای ورودی، نحوه جستجو و زمان توقف را ارائه می دهد

– فضای ورودی را به طور کامل جستجو کنید، به خصوص در گوشه ها
- همپوشانی زیادی در تست ها وجود ندارد

Advantages of Coverage Criteria

- Maximize the "bang for the buck"

مزایای معیارهای پوشش
به حداکثر رساندن "Bang for Buck"

ارزش بیشتر برای پول شخص، بازگشت سرمایه بیشتر.

- Provide traceability from software artifacts to tests
  - Source, requirements, design models, …

ارائه قابلیت ردیابی از مصنوعات نرم افزاری تا آزمایشات
– منبع، نیازمندی ها، مدل های طراحی، …
تست رگرسیون را آسان تر میکند
پس از پایان آزمایش به آزمایشکنندگان یک »قانون توقف« میدهد
می توان به خوبی با ابزارهای قدرتمند پشتیبانی کرد

- Make regression testing easier
- Gives testers a "stopping rule" … when testing is finished
- Can be well supported with powerful tools

Test Requirements and Criteria

- Coverage Criterion : A collection of rules and a process that define test requirements
  - Cover every statement
  - Cover every functional requirement

معیار پوشش: مجموعه ای از قوانین و فرآیندی که الزامات آزمون را تعریف می کند.
ـهر عبارت را میپوشاند
ـهر نیاز کاربردی را پوشش میدهد
لزامات آزمون: موارد خاصی که باید در طول آزمایش رعایت شوند یا پوشش داده شوند

- Test Requirements : Specific things that must be satisfied or covered during testing
  - Each statement might be a test requirement
  - Each functional requirement might be a test requirement

- هر عبارت ممکن است یک نیاز آزمایشی باشد

هر نیاز کاربردی ممکن است یک نیاز آزمایشی باشد

Testing researchers have defined dozens of criteria, but they are all really just a few criteria on four types of structures …

محققان آزمایش ده ها معیار را تعریف کرده اند، اما همه آنها در واقع تنها چند معیار در چهار نوع ساختار هستند…

1. Input domains
2. Graphs
3. Logic expressions
4. Syntax descriptions

1. دامنه های ورودی
2. نمودارها گراف ها
3. عبارات منطقی
4. توضیحات نحوی و سینتکسی