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

Old View : Colored Boxes

- Black-box testing : Derive tests from external descriptions of the software, including specifications, requirements, and design
- White-box testing : Derive tests from the source code internals of the software, specifically including branches, individual conditions, and statements

تست جعبه سیاه: آزمایش ها را از توضیحات خارجی نرم افزار، از جمله مشخصات، الزامات، و طراحی استخراج کنید
تست جعبه سفید: آزمایشها را از کد منبع داخلی نرم افزار استخراج کنید، به ویژه از جمله شاخه ها، شرایط فردی و عبارات

MDTD makes these distinctions less important.

The more general question is:

from what abstraction level do we derive tests?

MDTD این تمایزات را کمتر اهمیت می دهد.
سوال کلی تر این است:
از چه سطح انتزاعی تست ها را استخراج می کنیم؟

MBT and MDTD

- Model-based testing : Derive tests from a model of the software (such as a UML diagram)
  - *Typically assumes that the model has been built to specify the behavior of the software and was created during a design stage of development.*
- Our way is much overlap with MDTD and most of the concepts in this book can be directly used as part of MBT.

آزمایش مبتنی بر مدل: آزمایشها را از یک مدل نرم افزار استخراج میکند (مانند نمودار UML)
- معمولاً فرض میکند که مدل برای مشخص کردن رفتار نرم افزار ساخته شده است و در مرحله طراحی توسعه ایجاد شده است.
روش ما بسیار با MDTD همپوشانی دارد و بیشتر مفاهیم این کتاب را می توان مستقیماً به عنوان بخشی از MBT استفاده کرد.

Difference between MDTD and MBT

- We derive our tests from abstract structures that are very similar to models
- *Differences*
  - *These structures can be created **after the software is implemented**.*
  - *Create idealized structures that are more abstract than most modeling languages.*
  - *MBT explicitly does not use the source code implementation to design tests. In this book, abstract structures can be created from the implementation.*

ما تست های خود را از ساختارهای انتزاعی که بسیار شبیه به مدل ها هستند استخراج می کنیم

– این ساختارها را می توان پس از پیاده سازی نرم افزار ایجاد کرد.
- ساختارهای ایده آلی ایجاد میکند که انتزاعی تر از بسیاری از زبان های مدل سازی هستند.
- MBT به صراحت از پیاده سازی کد منبع برای طراحی تست ها استفاده نمی کند.
در این کتاب می توان ساختارهای انتزاعی از پیاده سازی ایجاد کرد.

Model-Driven Test Design (2.5)

- *Test Design* is the process of designing input values that will effectively test software

- Test design is one of several activities for testing software
  - Most mathematical
  - Most technically challenging

طراحی تست فرآیند طراحی مقادیر ورودی است که به طور موثر نرم افزار را آزمایش می کند
طراحی تست یکی از چندین فعالیت برای تست نرم افزار است.
- ریاضی ترین
- از نظر فنی چالش برانگیزترین

Types of Test Activities

- Testing can be broken up into four general types of activities
  1. Test Design
  2. Test Automation
  3. Test Execution
  4. Test Evaluation

بر اساس معیارها
مبتنی بر انسان

→ I.a) Criteria-based
→ I.b) Human-based

1. طراحی تست
2. تست اتوماسیون
3. اجرای تست
4. ارزیابی تست

هر نوع فعالیت به مهارت ها، دانش پیش زمینه، آموزش متفاوتی نیاز دارد
هیچ سازمان توسعه نرم افزار معقولی از افراد مشابه برای نیازمندی ها، طراحی، پیاده سازی، یکپارچه سازی و کنترل پیکربندی استفاده نمی کند

- Each type of activity requires different skills, background knowledge, education and training
- No reasonable software development organization uses the same people for requirements, design, implementation, integration and configuration control

Why do test organizations still use the same people for all four test activities??

This clearly wastes resources

چرا سازمان های آزمون هنوز از افراد یکسان برای هر چهار فعالیت آزمون استفاده می کنند؟
این به وضوح منابع را هدر می دهد

مقادیر تست را برای برآورده کردن معیارهای پوشش یا سایر اهداف مهندسی طراحی کنید

1. Test Design—(a) Criteria-Based

Design test values to satisfy coverage criteria or other engineering goal

- This is the most technical job in software testing
- Requires knowledge of :
  - Discrete math
  - Programming
  - Testing
- Requires much of a traditional CS degree
- This is intellectually stimulating, rewarding, and challenging
- Test design is analogous to software architecture on the development side
- Using people who are not qualified to design tests is a sure way to get ineffective tests

این فنی ترین کار در تست نرم افزار است
نیاز به دانش:- ریاضیات گسسته- برنامه نویسي- آزمایش کردن
به یک مدرک CS سنتی نیاز دارد
این از نظر فکری محرک، پاداش و چالش برانگیز است
طراحی تست مشابه معماری نرم افزار در سمت توسعه است
استفاده از افرادی که صلاحیت طراحی تست ها را ندارند یک راه مطمئن برای گرفتن تست های بی اثر است

1. Test Design—(b) Human-Based

Design test values based on domain knowledge of the program and human knowledge of testing

- This is much harder than it may seem to developers
- Requires knowledge of :
  - Domain, testing, and user interfaces
- Requires almost no traditional CS
  - A background in the domain of the software is essential
  - An empirical background is very helpful (biology, psychology, …)
  - A logic background is very helpful (law, philosophy, math, …)

این بسیار سخت تر از آن چیزی است که ممکن است برای توسعه دهندگان به نظر برسد
نیاز به دانش:
– دامنه، تست و رابط کاربری
تقریباً به هیچ CS سنتی نیاز ندارد
– داشتن پیشینه در حوزه نرم افزار ضروری است
- پیشینه تجربی بسیار مفید است (زیست شناسی، روانشناسی، ...)
- پیشینه منطقی بسیار مفید است (حقوق، فلسفه، ریاضی، ...)
مقادیر آزمون را بر اساس دانش دامنه برنامه و دانش انسانی آزمایش طراحی کنید

1. Test Design—(b) Human-Based

- This is intellectually stimulating, rewarding, and challenging
  - But not to typical CS majors – they want to solve problems and build things
- Human-based test designers explicitly attempt to find *stress tests*,
  - Tests that stress the software by including very large or very small values, boundary values, invalid values, or other values that the software may not expect during typical behavior.
  - Criteria-based approaches can be blind to special situations

این از نظر فکری محرک، پاداش و چالش برانگیز است
- اما نه برای رشته های اصلی CS - آنها می خواهند مشکلات را حل کنند و چیزهایی بسازند
طراحان آزمون های مبتنی بر انسان به صراحت تلاش می کنند تا تست های استرس را بیابند،
- تست هایی که با گنجاندن مقادیر بسیار بزرگ یا بسیار کوچک، مقادیر مرزی، مقادیر نامعتبر یا سایر مقادیری که نرم افزار ممکن است در رفتار معمولی انتظار نداشته باشد، به نرم افزار فشار وارد می کند.
- رویکردهای مبتنی بر معیار می توانند نسبت به موقعیت های خاص کور باشند

Comparison

- Many people think of criteria-based test design as being used for unit testing and human-based test design as being used for system testing.
- This is an artificial distinction.
- When using criteria, a graph is just a graph and it does not matter if it started as a control flow graph, a call graph, or an activity diagram.
- Likewise, human-based tests can and should be used to test individual methods and classes.

بسیاری از مردم تصور می کنند که طراحی آزمون مبتنی بر معیار برای زمایش واحد و طراحی آزمایش مبتنی بر انسان برای آزمایش سیستم مورد استفاده قرار می گیرد.این یک تمایز مصنوعی است.

هنگام استفاده از معیارها، یک نمودار فقط یک نمودار است و فرقی نمی کند که به عنوان یک نمودار جریان کنترلی، یک نمودار فراخوانی یا یک نمودار فعالیت شروع شده باشد.

به همین ترتیب، آزمونهای مبتنی بر انسان میتوانند و باید برای آزمایش متودها و کلاسهای فردی مورد استفاده قرار گیرند.

The main point is that the approaches are complementary and we need both to fully test software.

نکته اصلی این است که رویکردها مکمل یکدیگر هستند و برای آزمایش کامل نرم افزار به هر دو نیاز داریم.

2. Test Automation

Embed test values into executable scripts

تست اتوماسیون
مقادیر تست را در اسکریپت های اجرایی جاسازی کنید

- This is slightly less technical
- Requires knowledge of programming
- Requires very little theory
- Often requires solutions to difficult problems related to observability and controllability
- Can be boring for test designers
- Programming is out of reach for many domain experts

این کمی کمتر فنی است
نیاز به دانش برنامه نویسی دارد
به نظریه بسیار کمی نیاز دارد
اغلب نیاز به راه حل برای مشکلات دشوار مربوط به قابلیت مشاهده و کنترل
می تواند برای طراحان آزمون خسته کننده باشد
برنامه نویسی برای بسیاری از کارشناسان حوزه دور از دسترس است

3. Test Execution

Run tests on the software and record the results

- This is easy – and trivial if the tests are well automated
- Requires basic computer skills
  - Interns
  - Employees with no technical background
- Asking qualified test designers to execute tests is a sure way to convince them to look for a development job

تست ها را روی نرم افزار اجرا کنید و نتایج را ثبت کنید

اگر تست ها به خوبی خودکار شوند، این آسان و بی اهمیت است
به مهارت های اولیه کامپیوتر نیاز دارد
- کارآموزان
- کارکنان بدون سابقه فنی
درخواست از طراحان آزمون واجد شرایط برای اجرای تست ها راهی
مطمئن برای متقاعد کردن آنها به جستجوی شغل توسعه است

3. Test Execution

- If, for example, GUI tests are not well automated, this requires a lot of manual labor
- Test executors have to be very careful and meticulous with bookkeeping

برای مثال، اگر تستهای رابط کاربری گرافیکی به خوبی خودکار نباشند، این کار به کار دستی زیادی نیاز دارد

مجریان آزمون باید در حسابداری بسیار دقیق عمل کنند

4. Test Evaluation

Evaluate results of testing, report to developers

- This is much harder than it may seem
- Requires knowledge of :
  - Domain
  - Testing
  - User interfaces and psychology
- Usually requires almost no traditional CS
  - A background in the domain of the software is essential
  - An empirical background is very helpful (biology, psychology,…)
  - A logic background is very helpful (law, philosophy, math, …)
- This is intellectually stimulating, rewarding, and challenging
  - But not to typical CS majors – they want to solve problems and build things

نتایج آزمایش را ارزیابی کنید، به توسعه دهندگان گزارش دهید.

این بسیار سخت تر از آن چیزی است که ممکن است به نظر برسد\
یاز به دانش:\
- دامنه\
- آزمایش کردن\
- رابط کاربری و روانشناسی\
معمولاتقریباً به هیچ CS سنتی نیاز ندارد

– داشتن پیشینه در حوزه نرم افزار ضروری است\
- پیشینه تجربی بسیار مفید است (زیست شناسی، روانشناسی، ...)\
- پیشینه منطقی بسیار مفید است (حقوق، فلسفه، ریاضی، ...)

Other Activities

- Test management : Sets policy, organizes team, interfaces with development, chooses criteria, decides how much automation is needed,

مدیریت تست: خطمشی را تنظیم میکند، تیم را سازماندهی میکند، با توسعه ارتباط برقرار میکند، معیارها را انتخاب میکند، تصمیم میگیرد که چقدر اتوماسیون مورد نیاز است، ...

- Test maintenance : Save tests for reuse as software evolves
  - Requires cooperation of test designers and automators
  - Deciding when to trim the test suite is partly policy and partly technical – and in general, very hard !
  - Tests should be put in configuration control

تست نگهداری: تست ها را برای استفاده مجدد با تکامل نرم افزار ذخیره کنید
- نیاز به همکاری طراحان تست و اتوماسیون دارد

- تصمیم گیری در مورد زمان کوتاه کردن مجموعه تست حدی سیاست و تا حدی فنی است - و به طور کلی، بسیار سخت است!

- Test documentation : All parties participate
  - Each test must document "why" – criterion and test requirement satisfied or a rationale for human-designed tests
  - Ensure traceability throughout the process
  - Keep documentation in the automated tests

- تست ها باید در کنترل پیکربندی قرار گیرند
ستندات آزمون: همه طرفین شرکت می کنند
- هر آزمون باید "چرا" را مستند کند - معیار و لزامات آزمون برآورده شده است یا منطقی برای تست های طراحی شده توسط انسان

از قابلیت ردیابی در طول فرآیند اطمینان حاصل کنید
اسناد را در تست های خودکار نگه دارید

سازماندهی تیم

Organizing the Team

یک سازمان تست بالغ تنها به یک طراح تست نیاز دارد تا با چندین خودکار تست، مجری و ارزیاب کار کند.

- A mature test organization needs only one test designer to work with several test automators, executors and evaluators
- Improved automation will reduce the number of test executors
  - Theoretically to zero … but not in practice

اتوماسیون بهبود یافته تعداد مجریان آزمون را کاهش می دهد - از نظر تئوری به صفر ... اما در عمل نه

- Putting the wrong people on the wrong tasks leads to inefficiency, low job satisfaction and low job performance
  - A qualified test designer will be bored with other tasks and look for a job in development
  - A qualified test evaluator will not understand the benefits of test criteria

قرار دادن افراد نامناسب بر روی وظایف اشتباه منجر به ناکارآمدی، رضایت شغلی پایین و عملکرد پایین شغلی می شود.

یک طراح آزمون واجد شرایط از انجام وظایف دیگر خسته می شود و به دنبال شغلی در حال توسعه می گردد

یک ارزیاب واجد شرایط آزمون مزایای معیارهای آزمون را درک نخواهد کرد

- Test evaluators have the domain knowledge, so they must be free to add tests that "blind" engineering processes will not think of
- The four test activities are quite different

چهار فعالیت تست کاملاً متفاوت هستند

Many test teams use the same people for ALL FOUR activities !!

ارزیابهای آزمون دانش حوزه را دارند، بنابراین باید آزاد باشند تا تستهایی را اضافه کنند که فرآیندهای مهندسی »کور« به آن فکر نمیکنند.

بسیاری از تیم های تست از افراد یکسان برای هر چهار فعالیت استفاده می کنند!!

Applying Test Activities

بکارگیری فعالیت های آزمایشی
برای استفاده مؤثر از افراد خود و آزمایش مؤثر، به فرآیندی نیاز داریم که به طراحان آزمون اجازه دهد سطح انتزاع خود را بالا ببرند.

To use our people effectively

and to test efficiently

we need a process that

lets test designers

raise their level of abstraction

Using MDTD in Practice

این رویکرد به یک طراح آزمون اجازه می دهد تا حساب را انجام دهد

- This approach lets one test designer do the math
- Then traditional testers and programmers can do their parts
  - Find values
  - Automate the tests
  - Run the tests
  - Evaluate the tests

سپس تسترها و برنامه نویسان سنتی می توانند قسمت های خود را انجام دهند
یافتن مقادیر
- تست ها را خودکار کنید
– تست ها را اجرا کنید
- آزمون ها را ارزیابی کنید

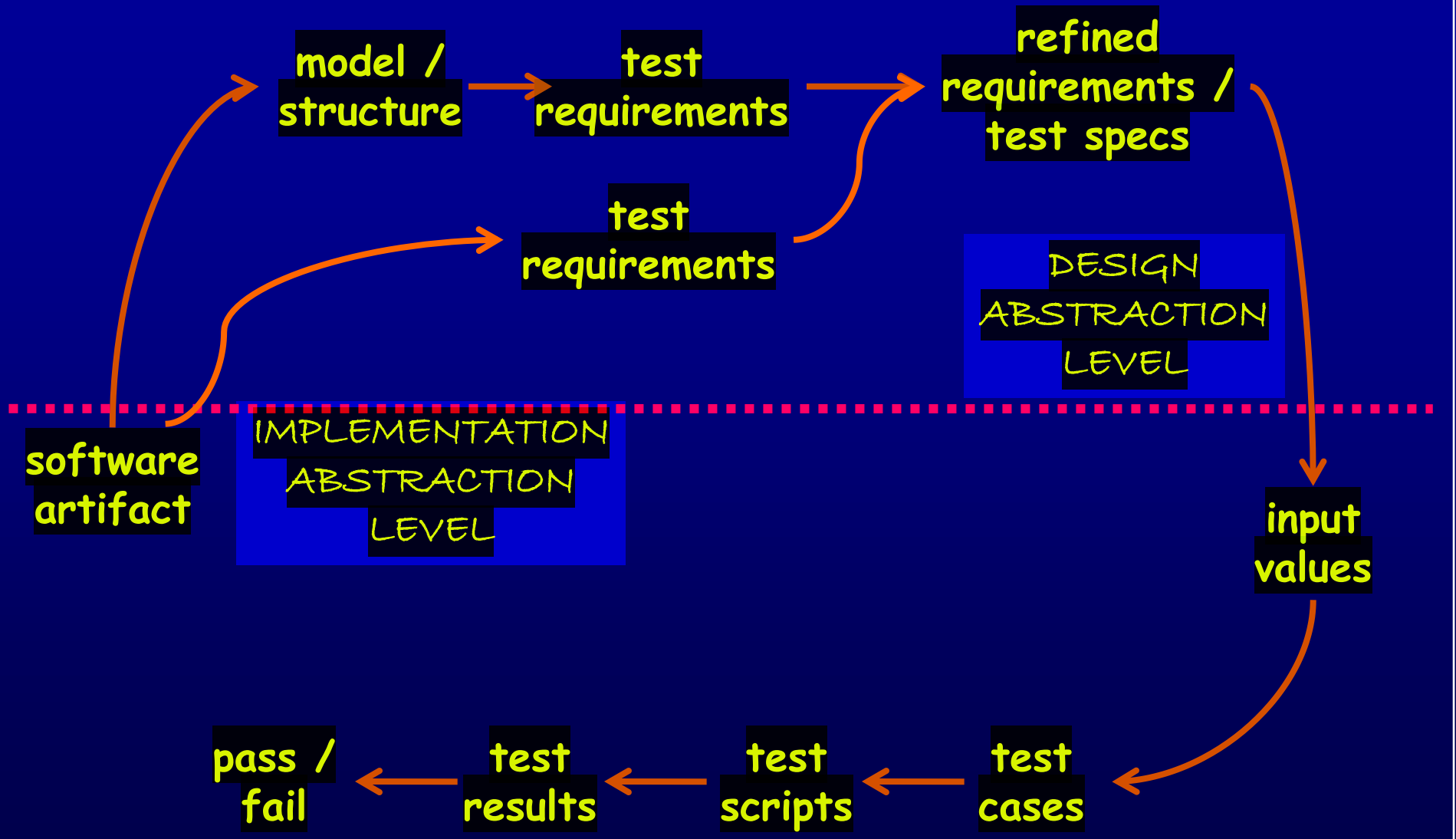
- Just like in traditional engineering … an engineer constructs models with calculus, then gives direction to carpenters, electricians, technicians, …

درست مانند مهندسی سنتی، یک مهندس با حساب دیفرانسیل و انتگرال مدل ها را می سازد، سپس به نجار، برق، تکنسین ها جهت می دهد.

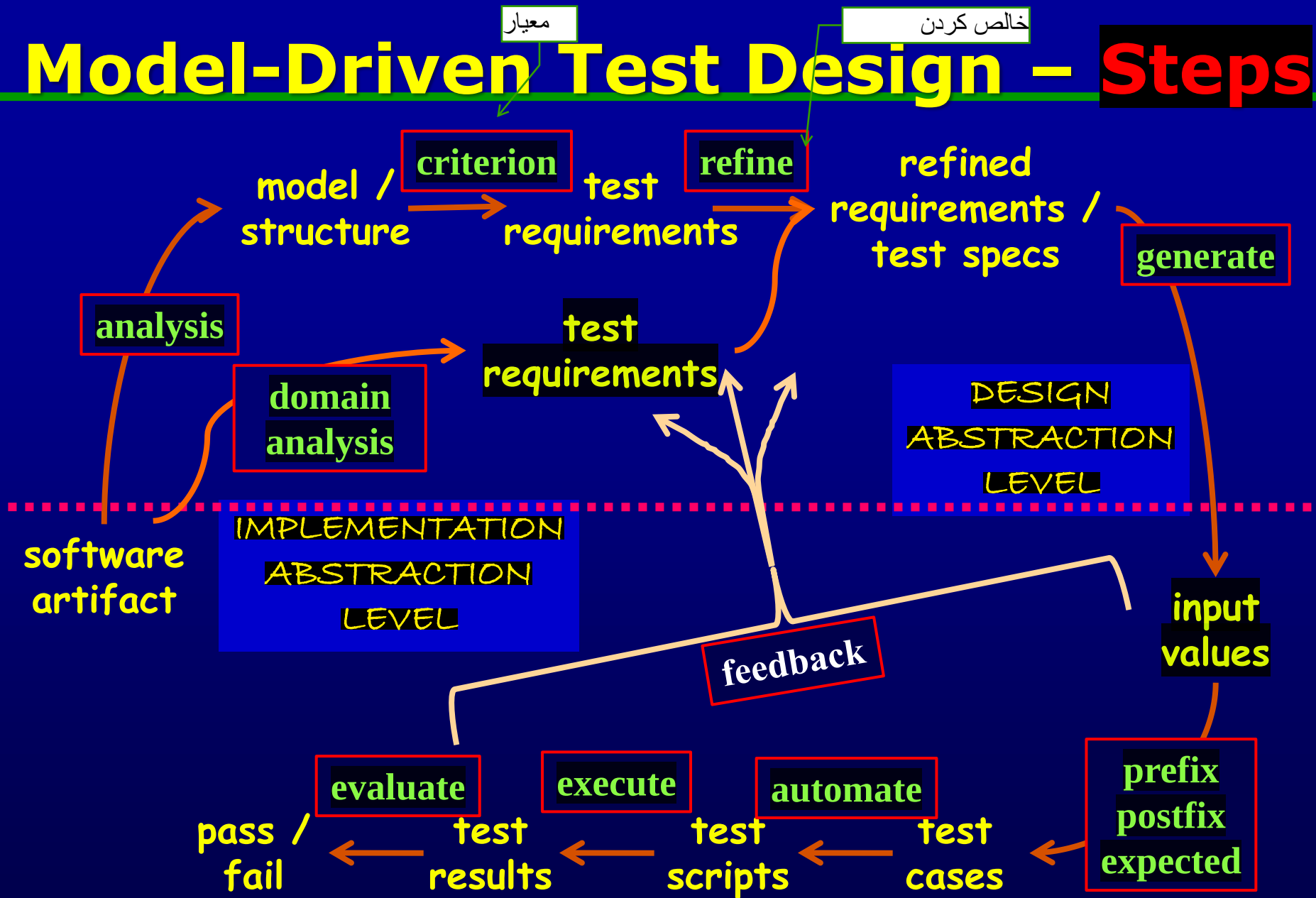
Test designers become technical experts

طراحان آزمون به کارشناسان فنی تبدیل می شوند

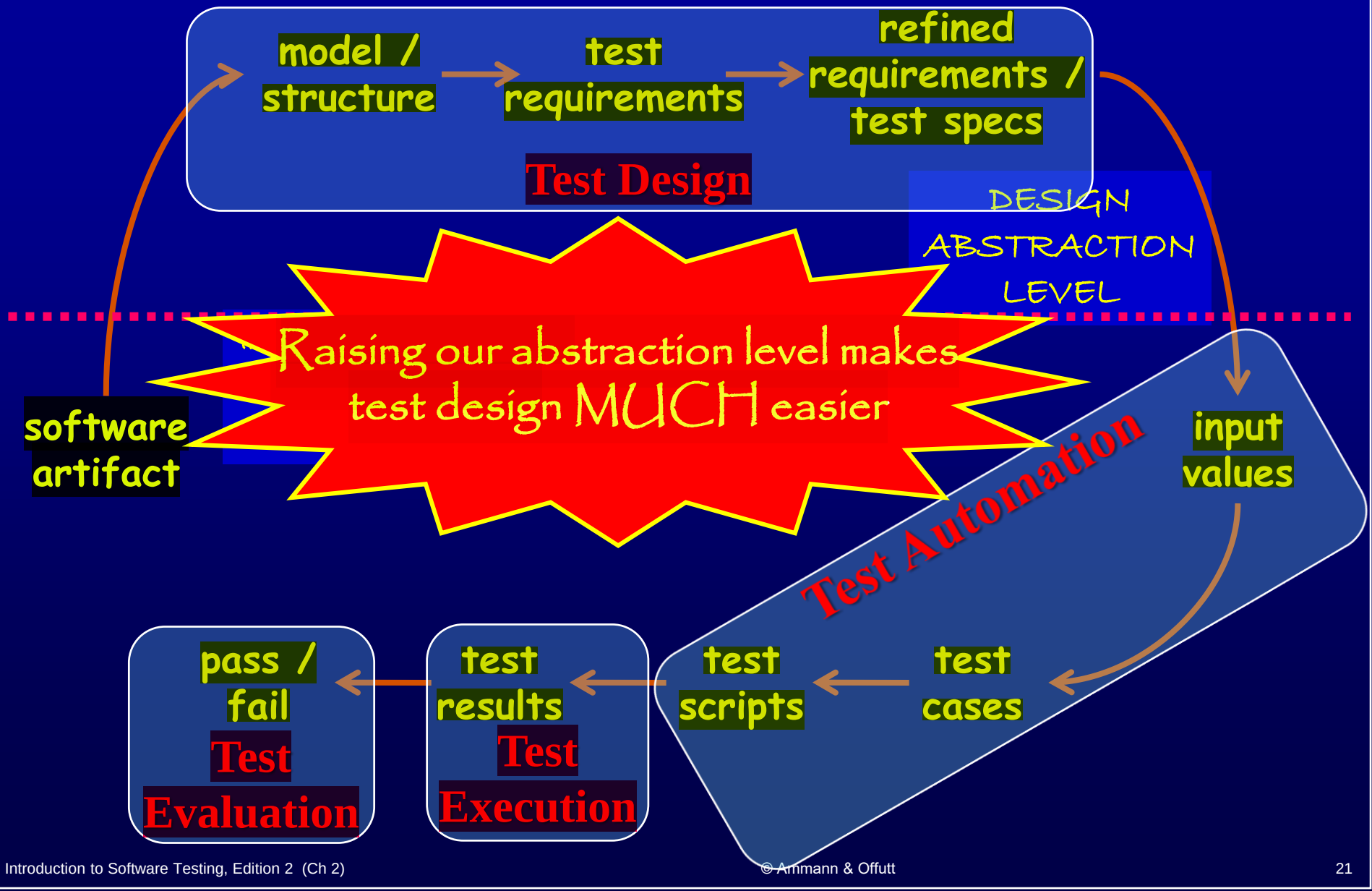
Model-Driven Test Design



Model-Driven Test Design – Steps



Model-Driven Test Design–Activities

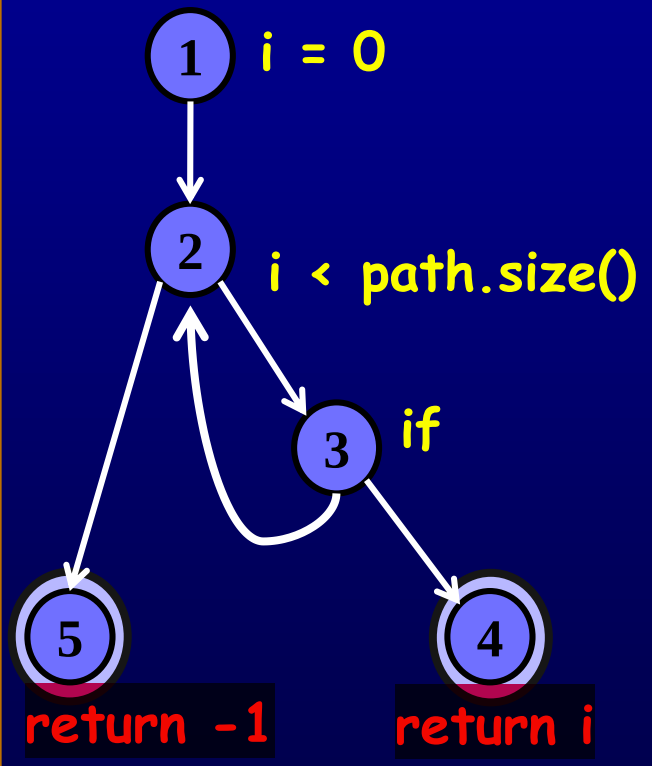


Small Illustrative Example

Software Artifact : Java Method

```
/**
    * Return index of node n at the
    * first position it appears,
    * -1 if it is not present
*/
public int indexOf (Node n)
{
    for (int i=0; i < path.size(); i++)
        if (path.get(i).equals(n))
            return i;
    return -1;
}
```

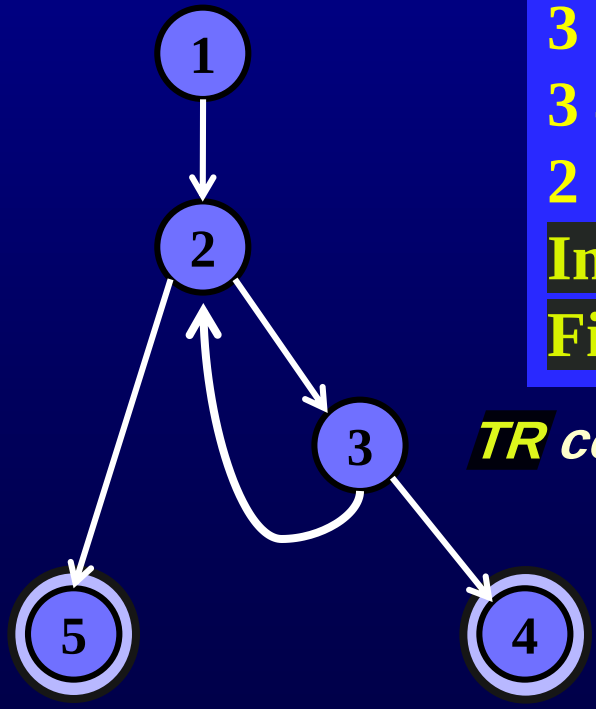
Control Flow Graph



Example (2)

Support tool for graph coverage
http://www.cs.gmu.edu/~offutt/softwaretest/

Graph
Abstract version



Edges
1 2
2 3
3 2
3 4
2 5
Initial Node: 1
Final Nodes: 4, 5

6 requirements for Edge-Pair Coverage
1. [1, 2, 3]
2. [1, 2, 5]
3. [2, 3, 4]
4. [2, 3, 2]
5. [3, 2, 3]
6. [3, 2, 5]

TR contains each reachable path of length up to 2 in graph G

Find values ...