Introduction to Software Testing
(*2nd edition*)
Chapter 5

Criteria-Based Test Design

Paul Ammann & Jeff Offutt

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

20 September 2015

Abstraction

should be used to handle complexity,

not to ignore it.

انتزاع باید برای رسیدگی به پیچیدگی استفاده شود، نه برای نادیده گرفتن آن.

Introduction

- The number of potential inputs for most programs is so large as to be effectively infinite and cannot be explicitly enumerated.

تعداد ورودی های بالقوه برای اکثر برنامه ها به قدری زیاد است که به طور موثر بی نهایت است و نمی توان به طور صریح آنها را برشمرد.

- This is where formal coverage criteria come in.

اینجاست که معیارهای پوشش رسمی وارد می شود.

- Since we cannot test with all inputs, coverage criteria are used to decide which test inputs to use.

از آنجایی که نمیتوانیم با همه ورودیها آزمایش کنیم، معیارهای پوشش برای تصمیم گیری برای استفاده از ورودی های تست استفاده میشود.

Introduction(Cnt'd)

- The rationale behind coverage criteria is that they divide up the input space to maximize the number of faults found per test case.

- From a practical perspective, coverage criteria also provide useful rules for when to stop testing.

منطق پشت معیارهای پوشش این است که آنها فضای ورودی را برای به حداکثر رساندن تعداد خطاهای یافت شده در هر مورد آزمایشی تقسیم می کنند.
از منظر عملی، معیارهای پوشش نیز قوانین مفیدی را برای زمان توقف آزمایش ارائه می کنند.

Changing Notions of Testing

- Old view focused on testing at each software development phase as being very different from other phases
  - Unit, module, integration, system .

تغییر مفاهیم تست
دیدگاه قدیمی بر آزمایش در هر مرحله توسعه نرم افزار متمرکز بود زیرا بسیار متفاوت از سایر مراحل است
– واحد، ماژول، یکپارچه سازی، سیستم ...

- New view is in terms of structures and criteria
  - input space, graphs, logical expressions, syntax
- Test design is largely the same at each phase
  - Creating the model is different
  - Choosing values and automating the tests is different

دیدگاه جدید از نظر ساختارها و معیارها است
- فضای ورودی، نمودارها، عبارات منطقی، نحو طراحی آزمون تا حد زیادی در هر مرحله یکسان است
– ایجاد مدل متفاوت است
- انتخاب مقادیر و خودکار کردن تست ها متفاوت است

New : Test Coverage Criteria

A tester's job is simple : Define a model of the software, then find ways to cover it

کار تستر ساده است: یک مدل از نرم افزار را تعریف کنید، سپس راه هایی برای پوشش آن پیدا کنید

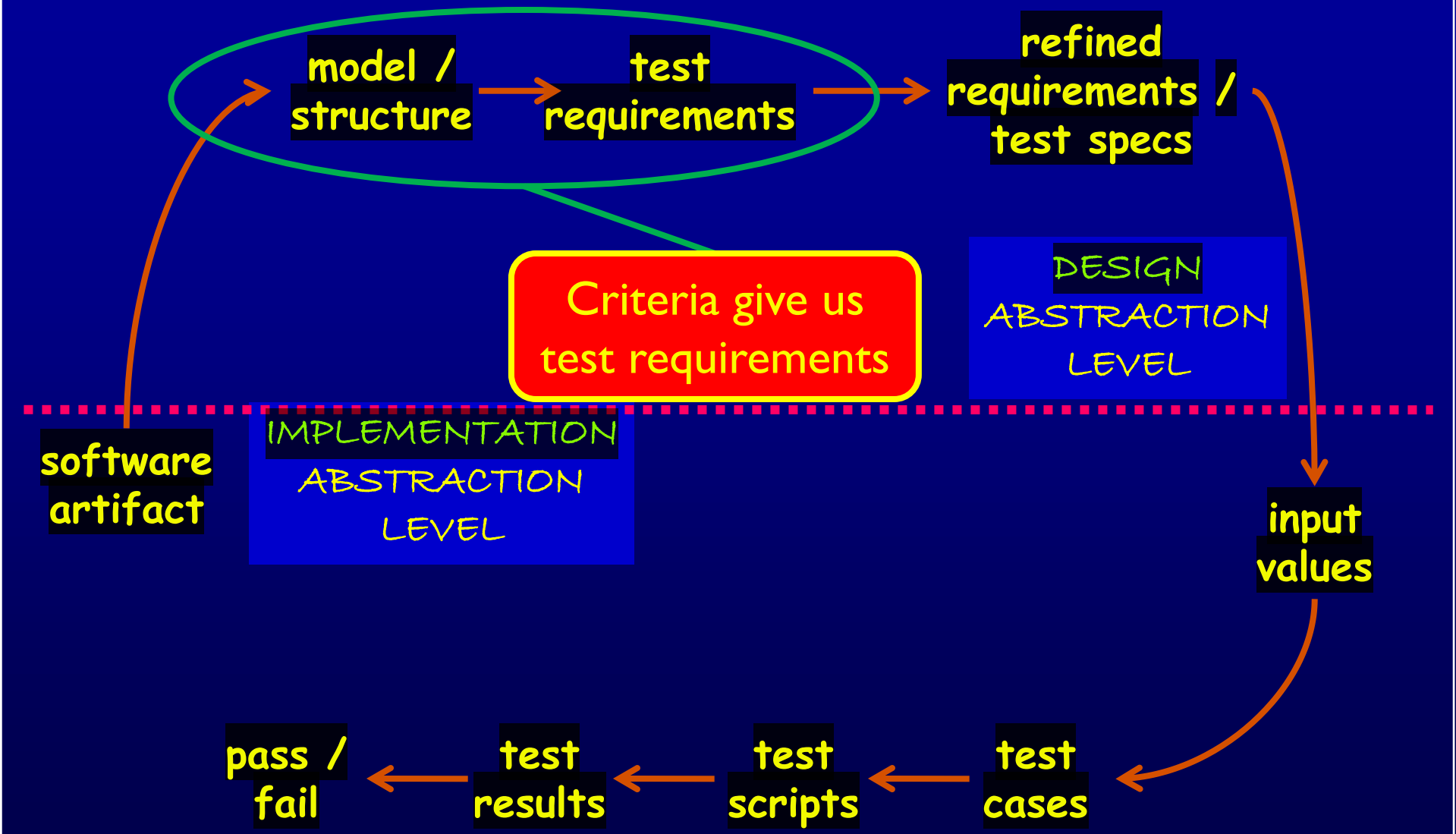
- Test Requirements : A specific element of a software artifact that a test case must satisfy or cover

الزامات تست: یک عنصر خاص از یک مصنوع نرم افزاری که یک مورد آزمایشی باید آن را برآورده یا پوشش دهد
معیار پوشش: قاعده یا مجموعه ای از قوانینی که الزامات آزمون را بر یک مجموعه آزمایشی تحمیل می کند.

- Coverage Criterion : A rule or collection of rules that impose test requirements on a test set

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

Model-Driven Test Design



Source of Structures

- These structures can be extracted from lots of software artifacts
  - Graphs can be extracted from UML use cases, finite state machines, source code, …
  - Logical expressions can be extracted from decisions in program source, guards on transitions, conditionals in use cases, …
- This is not the same as "*model-based testing*," which derives tests from a model that describes some aspects of the system under test
  - The model usually describes part of the behavior
  - The source is explicitly *not* considered a model

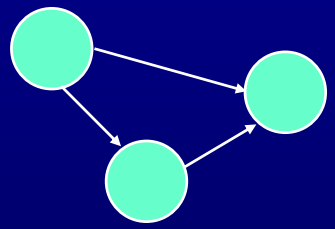
Criteria Based on Structures

Structures : Four ways to model software

1. Input Domain Characterization (sets)

   A: {0, 1, >1}
   B: {600, 700, 800}
   C: {swe, cs, isa, infs}

2. Graphs



3. Logical Expressions

   (not X or not Y) and A and B

4. Syntactic Structures (grammars)

```
if (x > y)
    z = x - y;
else
   z = 2 * x;
```

Example : Jelly Bean Coverage

Flavors :

1. Lemon
2. Pistachio
3. Cantaloupe
4. Pear
5. Tangerine
6. Apricot

Colors :

1. Yellow (Lemon, Apricot)
2. Green (Pistachio)
3. Orange (Cantaloupe, Tangerine)
4. White (Pear)

- **Possible coverage criteria** :
  1. Taste **one jelly bean** of **each flavor**
     - Set of Test Requirements: TR={Lemon, Pistachio, Cantaloupe, Pear, Tangerine, Apricot}
  2. Taste **one jelly bean** of **each color**
     - Set of Test Requirements: TR={Yellow, Green, Orange, White}

Coverage

Given a set of test requirements *TR* for coverage criterion *C*, a test set *T* satisfies *C* coverage if and only if for every test requirement *tr* in *TR*, there is at least one test *t* in *T* such that *t* satisfies *tr*

با توجه به مجموعه ای از الزامات آزمون TR برای معیار پوشش C، یک مجموعه آزمایشی T پوشش C را رآورده می کند اگر و تنها اگر برای هر شرط آزمون tr در TR، حداقل یک آزمون t در T وجود داشته باشد به طوری که tr را ارضا کند.

More Jelly Beans

T1 = { three Lemons, one Pistachio, two Cantaloupes, one Pear, one Tangerine, four Apricots }

- Does test set T1 satisfy the flavor criterion ?
  - Set of Test Requirements: TR={Lemon, Pistachio, Cantaloupe, Pear, Tangerine, Apricot}

T2 = { One Lemon, two Pistachios, one Pear, three Tangerines }

- Does test set T2 satisfy the flavor criterion ?
- Does test set T2 satisfy the color criterion ?
  - Set of Test Requirements: TR={Yellow, Green, Orange, White}

Minimal Test Set

Given a set of test requirements *TR* and a test set *T* that satisfies all test requirements, *T* is minimal if removing any single test from *T* will cause *T* to no longer satisfy all test requirements.

T1 = { three Lemons, one Pistachio, two Cantaloupes, one Pear, one Tangerine, four Apricots }

- Checking to see if a test set is minimal is fairly easy, and deleting tests to make the set minimal is straightforward.
- We can delete **two Lemon, one Cantaloupe, and three Apricot** jelly beans to make the above set minimal.

Minimum Test Set

Given a set of test requirements *TR* and a test set *T* that satisfies all test requirements, *T* is minimum if there is no smaller set of tests that also satisfies all test requirements.

Coverage Level

Given a set of test requirements TR and a test set T, The ratio of the number of test requirements satisfied by *T* to the size of *TR*

T2 = { One Lemon, two Pistachios, one Pear, three Tangerines }

- T2 satisfies 4 of 6 test requirements.

Infeasible test requirements

- Test requirements that cannot be satisfied
  - No test case values exist that meet the test requirements
  - Example: Dead code
  - Detection of infeasible test requirements is formally undecidable for most test criteria.
- Thus, 100% coverage is impossible in practice

الزامات آزمایشی که نمی توان آنها را برآورده کرد
- هیچ مقدار مورد آزمایشی وجود ندارد که شرایط آزمون را برآورده کند
– مثال: کد مرده
- تشخیص الزامات آزمایش غیرممکن برای اکثر معیارهای آزمون به طور رسمی غیرقابل تصمیم گیری است.
بنابراین پوشش 100% در عمل غیرممکن است

Two Ways to Use Test Criteria

1. Directly generate test values to satisfy the criterion
   - Often assumed by the research community
   - Most obvious way to use criteria
   - Very hard without automated tools
2. Generate test values externally and measure against the criterion
   - Usually favored by industry
   - Sometimes misleading
   - If tests do not reach 100% coverage, what does that mean?

به طور مستقیم مقادیر تست را برای برآورده کردن معیار ایجاد کنید
- اغلب توسط جامعه پژوهشی فرض می شود
- واضح ترین راه برای استفاده از معیارها
- بدون ابزار خودکار بسیار سخت است

مقادیر تست را به صورت خارجی تولید کنید و بر اساس معیار اندازه گیری کنید
- معمولاً مورد علاقه صنعت است
- گاهی اوقات گمراه کننده

Test criteria are sometimes called metrics

- اگر تست ها به پوشش %100 نرسند، به چه معناست؟

Generators and Recognizers

- Generator : A procedure that automatically generates values to satisfy a criterion
- Recognizer : A procedure that decides whether a given set of test values satisfies a criterion

Generator: رویه ای که به طور خودکار مقادیری را برای برآورده کردن یک معیار تولید می کند
شناساگر: رویهای که تصمیم میگیرد آیا مجموعه دادهای از مقادیر آزمون یک معیار را برآورده میکند یا خیر.

- Both problems are provably undecidable for most criteria
- It is possible to recognize whether test cases satisfy a criterion far more often than it is possible to generate tests that satisfy the criterion

تشخیص اینکه آیا موارد آزمایشی یک معیار را برآورده میکنند بسیار بیشتر از تولید تست هایی که معیار را برآورده میکنند ممکن است.

- Coverage analysis tools are quite plentiful

ابزارهای تحلیل پوشش بسیار زیاد هستند

Comparing Criteria with Subsumption (5.2)

- Criteria Subsumption : A test criterion *C1* subsumes *C2* if and only if every set of test cases that satisfies criterion *C1* also satisfies *C2*

باید برای هر مجموعه ای از موارد آزمایش درست باشد

- Must be true for every set of test cases
- *Examples* :

زیرمجموعه معیارها: یک معیار آزمایشی C1، C2 را زیرمجموعه می گیرد اگر و فقط اگر هر مجموعه ای از موارد آزمایشی که معیار C1 را برآورده می کند C2 را نیز برآورده کند.

  - The flavor criterion on jelly beans subsumes the color criterion … if we taste every flavor we taste one of every color.
  - A many-to-one mapping exists between the requirements for the flavor criterion and the requirements for the color criterion.
  - If a test set has covered every branch in a program (satisfied the branch criterion), then the test set is guaranteed to also have covered every statement.

معیار طعم روی لوبیاهای ژله ای، معیار رنگ را در بر می گیرد ... اگر هر طعمی را بچشیم، یکی از هر رنگی را می چشیم.

Advantages of Criteria-Based Test Design (5.3)

- Criteria maximize the "bang for the buck"
  - Fewer tests that are more effective at finding faults
- Comprehensive test set with minimal overlap
- Traceability from software artifacts to tests
  - The "why" for each test is answered
  - Built-in support for regression testing
- A "stopping rule" for testing—advance knowledge of how many tests are needed
- Natural to automate

ضوابط باعث به حداکثر رساندن "بنگ برای دلار" می شوند
- تست های کمتری که در یافتن عیوب موثرتر هستند
مجموعه تست جامع با حداقل همپوشانی
قابلیت ردیابی از مصنوعات نرم افزاری تا تست ها
- "چرا" برای هر آزمون پاسخ داده می شود
- پشتیبانی داخلی برای تست رگرسیون
یک "قانون توقف" برای آزمایش - آگاهی از تعداد آزمایشات مورد نیاز
طبیعی برای خودکار

Characteristics of a Good Coverage Criterion

1. It should be fairly easy to compute test requirements automatically
2. It should be efficient to generate test values
3. The resulting tests should reveal as many faults as possible

محاسبه خودکار الزامات آزمون باید نسبتاً آسان باشد
2. برای تولید مقادیر تست باید کارآمد باشد
3. آزمایشهای بهدستآمده باید تا حد امکان عیوب را آشکار کنند
Subsumption تنها یک تقریب تقریبی از قابلیت آشکارسازی خطا است

- Subsumption is only a rough approximation of fault revealing capability
- Researchers still need to gives us more data on how to compare coverage criteria

محققان هنوز باید اطلاعات بیشتری در مورد نحوه مقایسه معیارهای پوشش به ما بدهند

Test Coverage Criteria

- Traditional software testing is expensive and labor-intensive
- Formal coverage criteria are used to decide which test inputs to use
- More likely that the tester will find problems
- Greater assurance that the software is of high quality and reliability
- A goal or stopping rule for testing
- Criteria makes testing more efficient and effective

How do we start applying these ideas in practice?

How to Improve Testing ?

- Testers need more and better software tools
- Testers need to adopt practices and techniques that lead to more efficient and effective testing
  - More education
  - Different management organizational strategies
- Testing & QA teams need more technical expertise
  - Developer expertise has been increasing dramatically
- Testing & QA teams need to specialize more

Criteria Summary

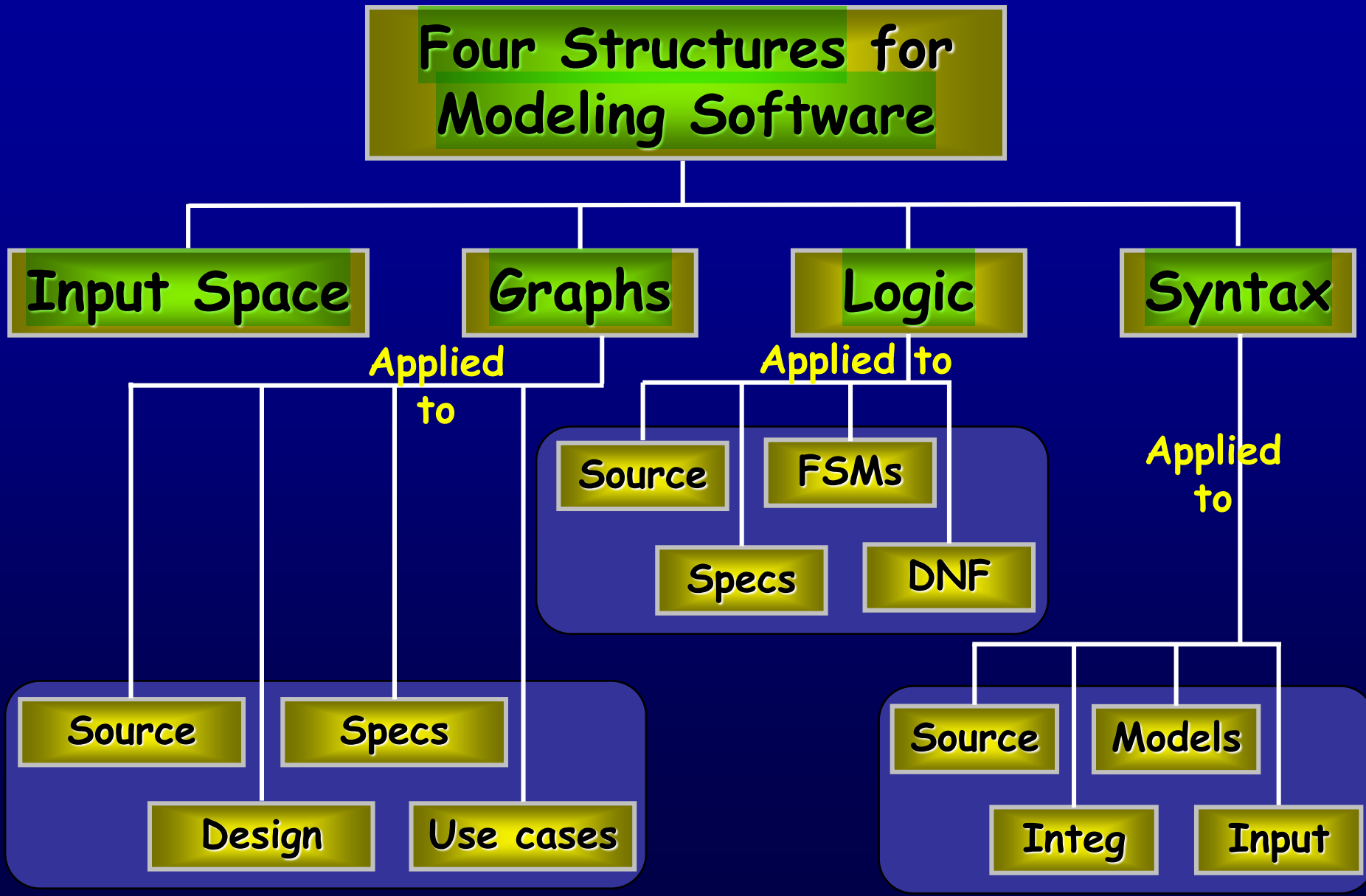
- Many companies still use "monkey testing"
  - A human sits at the keyboard, wiggles the mouse and bangs the keyboard
  - No automation
  - Minimal training required
- Some companies automate human-designed tests
- But companies that use both automation and criteria-based testing

Save money

Find more faults

Build better software

Structures for Criteria-Based Testing



Summary of Part 1's New Ideas

1. Why do we test – to reduce the risk of using software
   - Faults, failures, the RIPR model
   - Test process maturity levels – level 4 is a mental discipline that improves the quality of the software
2. Model-Driven Test Design
   - Four types of test activities – test design, automation, execution and evaluation
3. Test Automation
   - Testability, observability and controllability
4. Test Driven Development
5. Criteria-based test design
   - Four structures – test requirements and criteria

Earlier and better testing empowers test managers