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

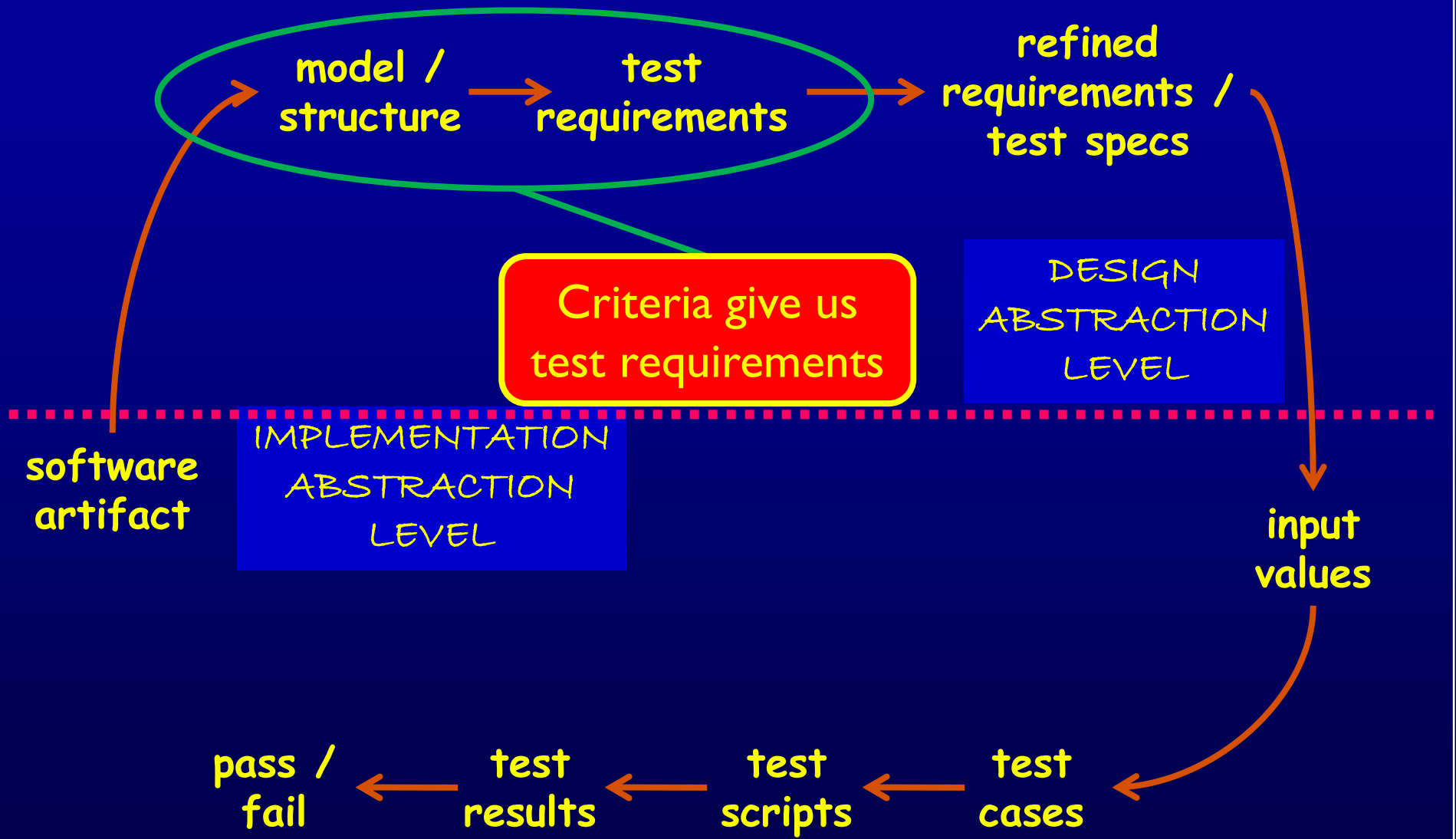
طراحی آزمون مبتنی بر معیار

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 …
- 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

-
تعریف فکر در رابطه با تستینگ:
توی روش های قدیمی فکر می کردند که توی هر فاز ایجاد نرم افزار که می خوان تست رو انجام بدن باید یه کار متفاوت انجام بدن ولی حالا فهمیدن که اینطوری نیست ینی درسته که فازهای ایجاد نرم افزار متفاوت است ولی مهم این است که توی همشون بیایم چند کار کلیدی انجام بدیم مثلا مدل استخراج بکنیم و بعد از روی اون مدل تست ها رو طراحی بکنیم و بعد ببینیم تست ها پاس میشن یا نه
فرق model driven با model based چیه؟
model based : اول تست رو بسازیم و بعد از روی تست بریم اون software artifacts که میخوایم رو بسازیم اما model driven ینی اینکه نرم افزار رو کدش رو داریم توی هر فاز حالا یا کد است یا سند ریکوارمنت ها هستش و از روی اون یک مدل در میاریم و بعد از روی مدلش تست رو طراحی میکنیم
توی روش های model driven موجود است artifacts و براساس این می تونیم تست رو انجام بدیم ولی برعکسش هم امکان پذیر است ینی اول تست رو بسازیم بعد کد رو بسازیم

Model-Driven Test Design



توی هر فاز ما software artifacts رو داریم و توی روش model driven میایم از روی این artifactsمون یک مدل استخراج می کنیم مثل گراف ینی گرافش رو می سازیم بعد از روی اون گراف که می ساختیم می اومدیم تست ریکوارمنت ها رو در میاوردیم و تست ریکوارمنت ها رو براساس ریکوارمنت ها در میاوردیم و اینکه تا چه حد ریز بشیم داخلش براساس معیار پوشش است

توی این فصل میخوایم راجع به معیار پوشش ها حرف بزنیم و توضیح بدیم اصلا که چرا این لازمه

New : Test Coverage Criteria

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

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

- 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 …

-
معیارهای پوششی که داریم معیارهای مختلفی هستن
نکته: ما نمیتونیم اینو بگیم که کدوم یکی از معیارها نسبت به اون یکی بهتر هست مثلا نمیتونیم بگیم every path بهتر هستش یا every state اینو نمی تونیم بگیم --> چیزی که هست اینه که همه معیارهای پوشش توی روش model driven بودن روی 4 تا استراکچر اعمال میشن:
استراکچرها عبارتند از:
1- Input Domain : مثلا Input Domain رو به عنوان استراکچر در نظر بگیریم و روش معیارهای پوشش رو اعمال بکنیم
2- گراف
3- Logical Expressions
4- Syntactic Structures

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

--

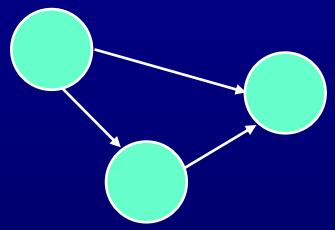
Criteria Based on Structures

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;
```

-

Input Domain: مثلا اگر قرار باشه یک ورودی x اینتجر بگیریم بگیم xهای مثبت و xهای منفی و xهای صفر ینی پارتیشن بندی بکنیم و بعد برنامه رو براساس اون تست بکنیم

Logical Expressions: این براساس شرط هایی است که توی اون software artifacts برقرارن مثلا اگر یک کد داشته باشیم و software artifacts مون سورس کد باشه تون اون فاز در این حالت میشه کل ساختارهای شرطی که دارن Logical Expressions رو بیان میکنن یا مثلا توی مرحله دیزان هستیم اون راهنماهایی که روی استیت ماشین ها نوشته میشه اون راهنماها Logical Expressions ما رو میسازن

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
     - Deciding if yellow jelly bean is Lemon or Apricot is a controllability problem
  2. Taste one jelly bean of each color

-
مثال را روی شکلات های ژله ای زده :)))
یه عالمه شکلات داریم و ما میخوایم تست بکنیم --> حالا چجوری تست رو انجام میدیم؟
براساس چی میخوایم تست بکنیم؟ ینی براساس طعم میخوایم تست بکنیم یا براساس رنگ یا براساس مزه یا... --> اینا معیارهای پوشش میشه
طعم ها: لیمو - پسته - طالبی - گلابی - نارنگی - زردالو
معیار رنگ رو انتخاب بکنیم یا طعم؟ اگر معیار پوشش رو طعم قرار بدیم معیار رنگ رو هم ناخوداگاه پوشش میدیم اما توی مثال های مختلف ممکنه همیشه اینطوری نباشه

یه مثال زد:
معیار پوششی که مد نظر است:
نکته: ایا می تونیم به معیار پوشش 100 درصد برسیم در طراحی؟ ایا می تونیم تست ستی طراحی کنیم براساس هر معیار پوششی که بگیم 100 درصد کاور کرده؟ بسته به معیار پوشش داره -->
بررسی این مسئله undesirable است
مسئله undesirable مسئله ای است که با الگوریتمی نمی تونیم به جواب براش برسیم

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*

- 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

-
بعضی از تست ریکوارمنت هایی که بخاطر معیار پوشش داریم Infeasible هستن چون هیچ وقت ما نمی تونیم تست ستی رو طراحی کنیم که باعث اجرای اون بشه --> ینی ممکنه یک تست ریکوارمنتی داشته باشیم که هر چقدر هم تلاش کنیم نتونیم هیچ تست کیسی براش طراحی کنیم مثل Dead code ینی کدی که توی سورس کد هست ولی هیچ وقت اجرا به اونجا نمی رسه
تست ریکوارمنتی که satisfied نشه توسط هیچ تست ستی بهش Infeasible test requirements میگن مثل Dead code --> و Dead code : یه تیکه کد داریم که هیچ وقت اجرا نمیشه مثلا یه سری کد بعد از ریترن باشه و مثلا اگر ما معیار پوششمون این باشه که هر استیت رو بخوایم توی سورس کد پوشش بدیم اون جملات بعد از ریترن رو نمی تونیم با تست اجرا بکنیم که ببینیم درست هست یا نه پس every state برای اون جملات بعد از ریترن میشه Infeasible
پس برای هر تست ستی که می نویسیم متناسب با اون معیار پوشش یک درصدی مشخص می کنیم که بهمون بگه تا چه حد پوشش بده
نکته: اینکه تشخیص بدیم که ایا یک مجموعه ای از تست ریکوارمنت ها پوشش 100 درصد داره یا نداره این مسئله undesirable است (ینی این که ایا این تست ریکوارمنت ها رو می تونیم 100 درصد پوشش بدیم یا نه این یک مسئله undesirable است) --> برای همین معیار پوشش رو برحسب درصد بیان می کنن ینی اندازه کل تست ریکوارمنت هامون ینی اون مجموعه تست ریکوارمنت توی مخرج قرار میگیره و اون تست ریکوارمنت هایی که تونستیم براشون تست کیس در بیاریم توی صورت میذاریم که این میشه coverage level

More Jelly Beans

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

- Does test set T1 satisfy the flavor criterion ?

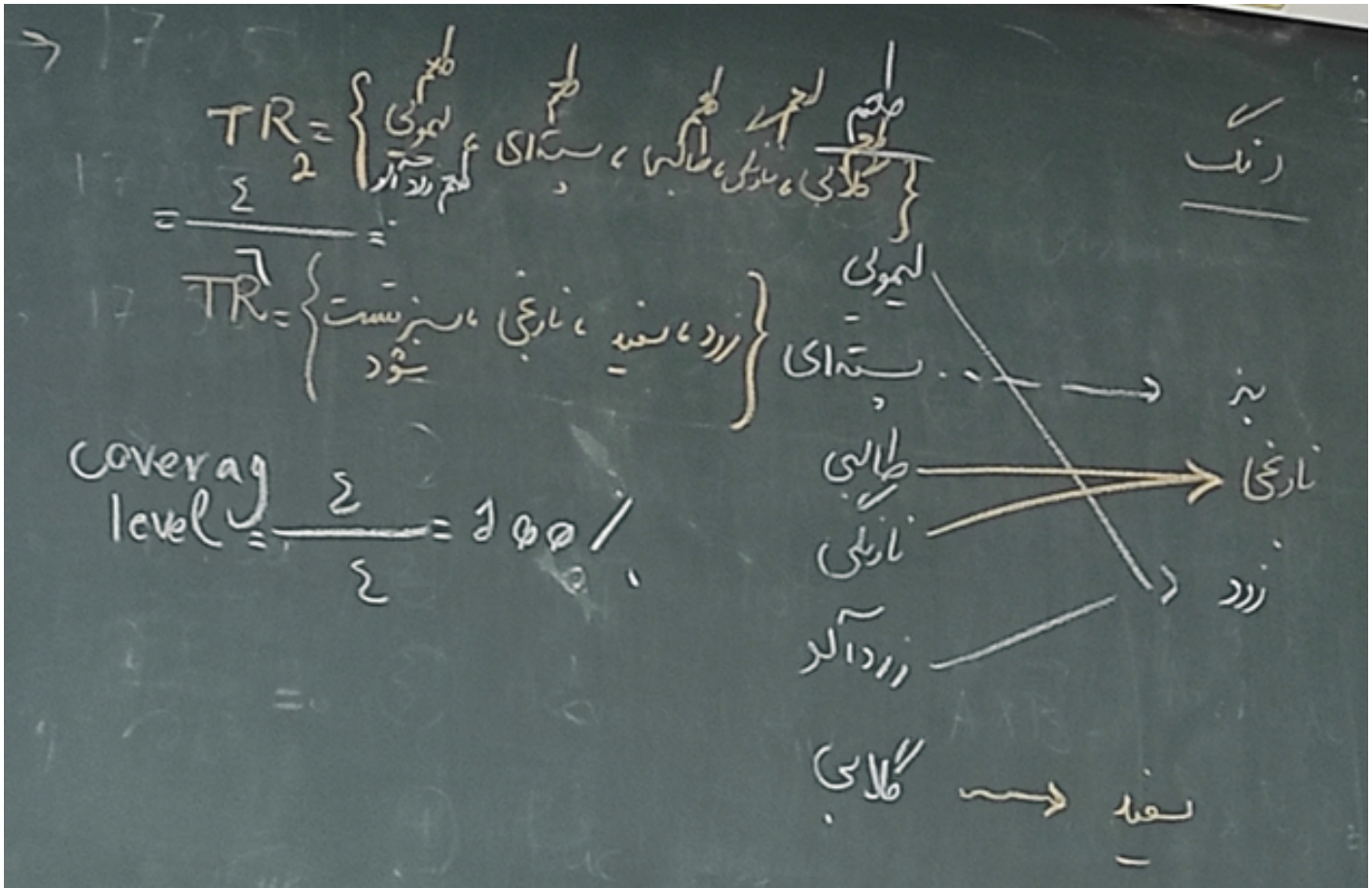
T2 = { One Lemon, two Pistachios, one Pear, three Tangerines }

- Does test set T2 satisfy the flavor criterion ?
- Does test set T2 satisfy the color criterion ?

بیشتر ژله لوبیا
T1 = { سه عدد لیمو، یک عدد پسته، دو عدد طالبی، یک عدد گلابی، یک عدد نارنگی، چهار عدد زردآلو}
آیا مجموعه تست T1 معیار طعم را برآورده می کند؟
T2 = { یک لیمو، دو عدد پسته، یک گلابی، سه نارنگی}
آیا مجموعه تست T2 معیار طعم را برآورده می کند؟
آیا مجموعه تست T2 معیار رنگ را برآورده می کند؟

مثال: یکی ممکنه بگه که معیار پوششش رنگ است --> پس مجموعه تست ریکوارمنت ها میشه این که هر رنگ تست بشه --> یک تست ست طراحی میکنیم و یکی از تست کیس ها معادل این است: 3 تا لیمو - 1 پسته - 2 طالبی - 1 گلابی - 1 نارنگی - 4 زردالو --> حالا اگر این تست ها رو انجام بدیم چه درصدی از این مجموعه تست ریکوارمنت ها پوشش داده شده ؟ 4/4 میشه --> اینجا به پوشش 100 درصد رسیدیم

حالا اگر TR2 به صورت شکل زیر باشه: اگر معیار پوشش طعم باشه --> اینجا به پوشش 100 درصد نرسیدیم چون از همه میوه ها نیست داخلش (زردالو و طالبی نیست) --> تست ست ما اینجا: 1 لیمو - 2 پسته - 1 گلابی - 3 نارنگی --> 4/6



--

Coverage Level

The ratio of the number of test requirements satisfied by *T* to the size of *TR*

- T2 on the previous slide satisfies 4 of 6 test requirements

---

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

- 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

- Coverage analysis tools are quite plentiful

(output: values) undecidable : Generator

(Out put : Yes/No) undecidable : Recognizer

-
generate : generate کردن ینی اینکه ما بتونیم برای یک مسئله مجموعه جواب هاش رو تولید بکنیم
Recognizer : ینی اینکه بتونیم با توجه به یک مجموعه جواب موجود جواب yes یا no برگردونیم ینی این مجموعه جواب موجود جواب صحیح مسئله است یا نه
اگر یکسری معیارها داشته باشیم و بخوایم براشون تست کیس طراحی بکنیم این میشه Generator ینی نیاز داریم به یک پروسیجری که Generator باشه --> مسئله undesirable است --> ایا می تونیم برای چنین معیار پوششی حتما تست کیس با پوشش 100 درصد تولید بکنیم و حتی Recognizer کردن این مسئله ینی شناسایی این مسئله هم undesirable است ینی یک مجموعه از معیارهای پوشش داشته باشیم و یک مجموعه ای از test value ها و بعد بخوایم بفهمیم این test value ها این معیار پوشش رو ارضا می کنه 100 درصد یا نه اینم undesirable است ولی به مراتب اسون تر از مسئله Generate کردن است
نکته: خروجی مسئله Generator مجموعه جواب است ینی تست کیس برامون می سازه ولی خروجی مسئله Recognizer یس و no میشه

Comparing Criteria with Subsumption (5.2)

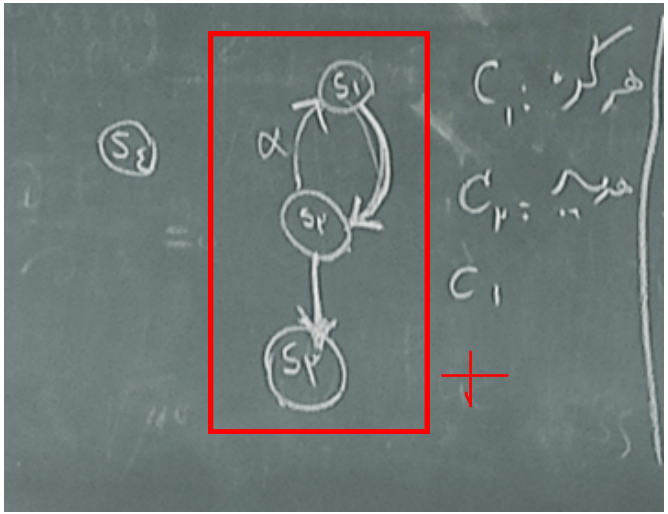
- Criteria Subsumption : A test criterion *C1* subsumes *C2* if and only if every set of test cases that satisfies criterion *C1* also satisfies *C2*
- Must be true for every set of test cases
- *Examples* :
  - The flavor criterion on jelly beans subsumes the color criterion … if we taste every flavor we taste one of every color
  - 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 پوشش مثلا c1 اینو می تونیم با معیارهای پوشش دیگه مقایسه کنیم --> مقایسه کردنشون هم Subsumption هستش --> Subsumption ینی شامل شدن یا include کردن
اگر بتونیم یک معیار پوششی رو انتخاب بکنیم و بتونیم با توجه به اون معیار پوشش دیگه ای هم پوشش بدیم در این حالت میگیم c1 اینکلود میکنه c2 ینی هر مجموعه تست کیسی که بتونه 100 درصد پوشش بکنه این معیار پوشش رو حتما معیار پوشش اینکلود شده هم 100 درصد مورد پوشش قرار میده
مثال:
معیار پوشش c1: هر گره
معیار پوشش c2: هر مسیر
ایا می تونیم بگیم c1 اینکلود می کنه c2 رو؟ نه --> اگر مسیر s1 به s2 بریم و بعد از s2 به s3 خارج بشیم هر نود رو پوشش دادیم با توجه به این تست کیس ولی یکی از مسیرها باقی می مونه که ما نمیتونیم پوشش بدیم پس نمی تونیم بگیم این رابطه بینشون برقرار هستش ولی برعکسش برقراره ینی اگر هر مسیر رو رفتیم هر نود رو هم رفتیم (برای شکل +) --> ولی اگر یک نودی داشتیم که با هیچ مسیری نمی تونستیم پوشش بدیم در این حالت برعکسش هم دیگه برقرار نیست ینی توی این حالت Dead code داریم

توی مثال ژله هم همینطوره --> اگر معیار پوشش رو هر طعم انتخاب بکنیم اگر با یک تست کیسی همه طعم ها رو بررسی کنیم ایا همه رنگ ها رو هم بررسی کردیم؟ بله پس اینجا معیار پوشش هر طعم اینکلود می کنه معیار پوشش هر رنگ رو (به شرطی که توی تست ریکوارمنت های استخراج شده از اون معیار پوشش Infeasible test requirements نداشته باشیم **) ولی برعکسش برقرار نیست

** ینی اگر مثلا توی تست ریکوارمنت یه دونه Infeasible داشته باشیم مثلا طعم البالو که این اصلا موجود نیست --> مثلا معیار پوشش طعم تمام میوه های موجود در شهر باشه بعد اگر غیر از اون 6 تا میوه البالو هم جز میوه های توی شهر باشه ما هیچ وقت نمی تونیم این تست ریکوارمنت رو پوشش بدیم چون اصلا کارخانه طعم البالو رو تولید نکرده ینی هیچ راهی برای تست کردنش نیست که این میشه تست ریکوارمنتی که Infeasible هستش مثل همون dead code که قرار نیست هیچ وقت براش تست کیس داشته باشیم

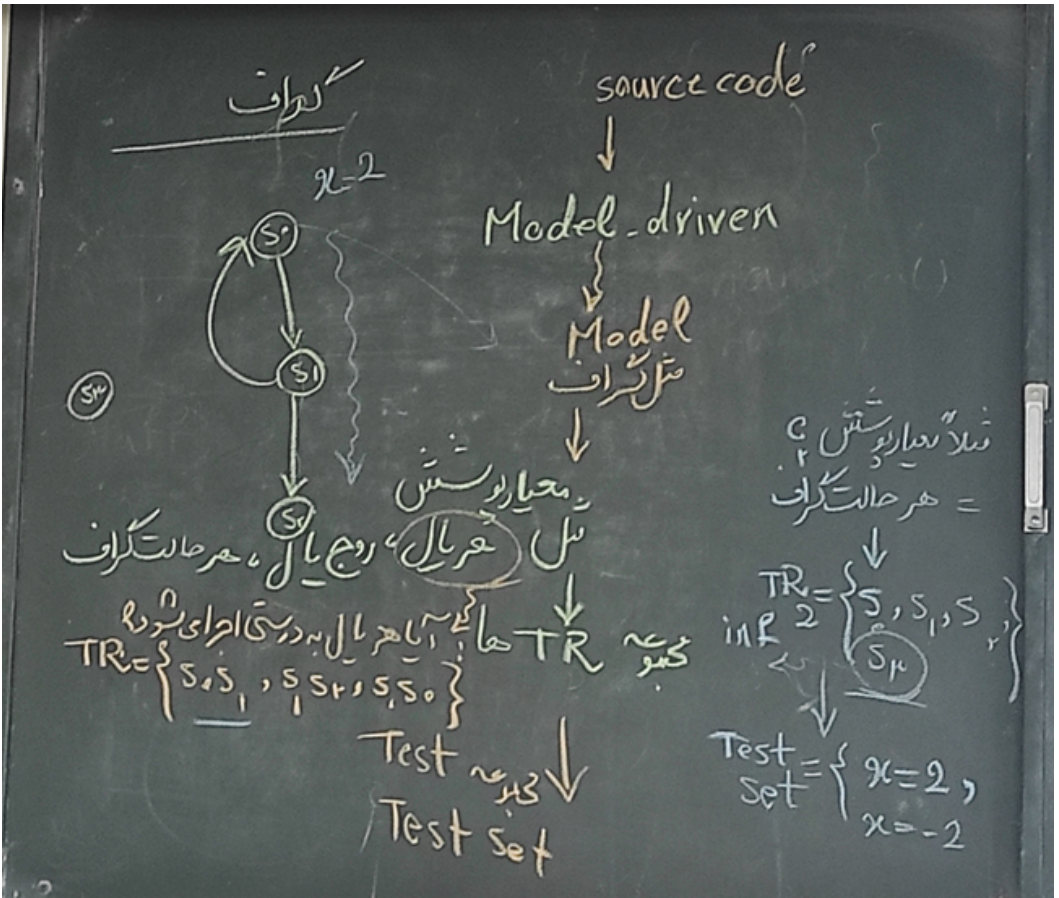
نکته: اگر معیار پوششی مثل اینجا که طعم البالو رو بهمون داد --> معیار پوششی بیاد به ما یک تست ریکوارمنت بهمون بده که Infeasible باشه دیگه نیایم رابطه Subsumption رو روی این معیار پوشش با معیارهای پوشش دیگه بررسی کنیم چون این Infeasible کار رو خراب میکنه



مثال: فرض میکنیم این گراف، گراف برنامه است
سورس کد اینجا موجود است بعد ما از روی سورس کد یک مدل در اوردیم که بتونیم تست کیس ها رو طراحی بکنیم --> روش model driven
نکته: از روی معیار پوشش، تست ریکوارمنت ها رو استخراج میکنیم--> مثلا اگر معیار پوشش هر یال است --> ایا هر یال به درستی اجرا میشه یا نه (تست ریکوارمنت میشه این) پس یال s0s1 چک بشه ایا این یال به درستی اجرا میشه و s1s2 چک بشه و s1s0 چک بشه حالا ایا با توجه به معیار پوشش هر یال ما تست ریکوارمنتی داریم که جا مونده باشه ؟ نه
حالا توی بحث Infeasible بودن --> اینجا یک تست ریکوارمنتی ایجاد بشه که نتونیم براش توی تست ست ها، تست کیسی داشته باشیم که اونو چک بکنه ینی معیار پوشش قبل از استخراج تست ریکوارمنت ها مطرح است
اگر مثلا معیار پوشش c2 برابر باشه با هر حالت گراف --> تست ریکوارمنت ها میشه:
s0,s1,s2,s3 و ایا برنامه به درستی خط s0 را اجرا می کند ایا به درستی خط s1 را اجرا می کند و...
حالا از روی این معیار پوشش c2 تست ریکوارمنت رو در اوردیم--> حالا می خوایم تست کیس ها رو تهیه بکنیم --> ینی یه تستی بنویسیم که مطمئن باشیم هر کدوم از این خواسته های ما با توجه به این مجموعه تست ها حداقل یکبار دارن چک میشن --> مثلا ورودی مسیر x است و اگر x رو بدیم 2 مسیر s0s1s2 در برنامه اجرا میشه پس در این مسیر تمام استیت های s0,s1,s2 رو چک کردیم پس توی تست x رو میدیم 2 که برنامه این مسیر رو بره و تمام اون حالت های گراف دیده بشه و مطمئن بشیم همشون چک شدن یا نه
مثلا ممکنه یک شاخه دیگه هم داشته باشیم که به ازای x مساوی 2- داره برنامه رو شبیه سازی می کنه در این حالت 2- رو به برنامه می دیم که بتونیم تستش بکنیم

نکته: وقتی که معیار پوشش داریم و تست ریکوارمنت ها رو هم داریم نگاه میکنیم که برای یکی از تست ریکوارمنت ها که حالت s3 است نتونستیم هیچ کاری انجام بدیم ینی نتونستیم هیچ تستی رو طراحی بکنیم که اجرای خط s3 رو چک بکنه در این حالت این میشه Infeasible test requirements چرا؟ چون هرچقدر فکر کردیم نتونستیم براش تست کیس طراحی بکنیم براش که بذاریم توی مجموعه تست ست ها

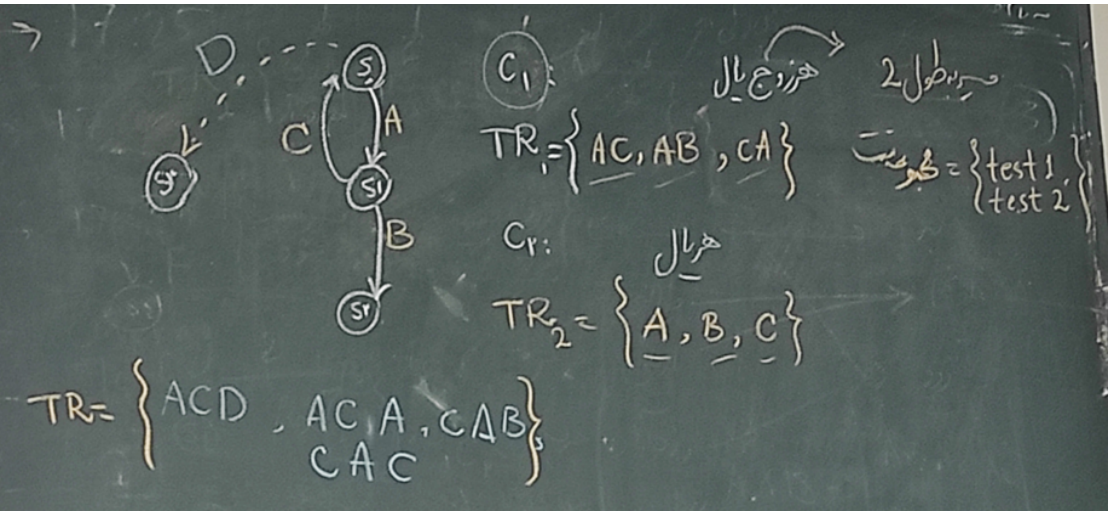
حالا چاره چیست؟ هیچ وقت پوشش 100 درصد حاصل نمیشه و مسئله هم undesirable است و ما به عنوان مدیر تست می تونیم بگیم که اگر تونستی یک مجموعه ای از تست کیس هایی طراحی بکنی که با 90 درصد تست ریکوارمنت ها رو پوشش میداد کافی است یا یه مدیر دیگه بگه 70 درصد پوشش بدی کافیه



معیارهای پوشش یکی جامع تر از دیگری ممکنه باشه این همون Subsumption است مثلا اگر معیار پوشش c1 هر یال باشه و c2 هر حالت گراف ایا می تونیم بگیم که c1 می تونه c2 رو Subsumption بکنه؟ ینی جامع تر است؟ نمی تونیم راجع بهش بحرفیم --> اینا رو باید به ازای همه حالات در نظر بگیریم نه اینکه یک مثال بزنیم و ببینیم اینکلود کرده و بگیم رابطه Subsumption بینش برقرار هست در صورتی که اگر توی تمام مدل ها این اتفاق بیوفته در این حالت ما می تونیم بگیم که Subsumption است

مثلا اگر معیار پوشش هر زوج یال باشه ینی c1 ایا می تونیم بگیم این Subsumption می کند معیار هر یال رو ینی c2 ؟ ینی ایا می تونیم بگیم c1 می تونه c2 رو Subsumption بکنه؟ بله --> c1 می تونه c2 رو Subsumption بکنه ینی ما به ازای تست کیس هایی که طراحی میکنیم که تمام تست ریکوامنت های یک رو ارضا کنیم اون تست کیس ها حتما تست ریکوارمنت های c2 هم پوشش میدن مثلا اگر یک مجموعه تست داشتیم ینی test2 , test1 اینا حداقل یکبار هر زوج یال رو چک می کنن پس می تونیم مطمئن باشیم که هر یالی رو هم حتما چک می کنن

نکته: اگر در کل گراف از مبدا تا مقصد فقط مسیری به طول 1 داشتیم ینی نتونستیم برسونیمش به مسیری به طول 2 اونوقت توی چک کردن زوج یال ها اون ها هم باید اضافه بکنیم --> تک یال هایی رو اضافه میکنیم که مسیری به طول 2 دیگه نتونن بهمون بدن



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

چرا معیار پوشش باید داشته باشیم؟ چون اگر نداشته باشیم ما از کجا بفهمیم این تست ستی که طراحی کردیم دوتا دونه توشه یا 100 تا توشه یا .. کفایت میکنه اخرش یا نه پس یک قانونی برای متوقف کردن تیم طراح تست باید داشته باشیم --> توی معیارهای پوشش باید تست ریکوارمنت رو مثلا پوشش بدیم با 80 درصد پوشش
وقتی که معیار پوشش داشته باشیم regression testing راحت تر میشه

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

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

---

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?

Four Roadblocks to Adoption

1. Lack of test education
   - Microsoft and Google say half their engineers are testers, programmers test half the time
   - Number of UG CS programs in US that require testing ? **0**
   - Number of MS CS programs in US that require testing ? **0**
   - Number of UG testing classes in the US ? **~50**
2. Necessity to change process
   - Adoption of many test techniques and tools require changes in development process
   - This is expensive for most software companies
3. Usability of tools
   - Many testing tools require the user to know the underlying theory to use them
   - Do we need to know how an internal combustion engine works to drive ?
   - Do we need to understand parsing and code generation to use a compiler ?
4. Weak and ineffective tools
   - Most test tools don't do much – but most users do not realize they could be better
   - Few tools solve the key technical problem – generating test values automatically

از اینجا به بعد نمیاد!!!

Needs From Researchers

1. Isolate : Invent processes and techniques that isolate the theory from most test practitioners
2. Disguise : Discover engineering techniques, standards and frameworks that disguise the theory
3. Embed : Theoretical ideas in tools
4. Experiment : Demonstrate economic value of criteria-based testing and ATDG (*ROI*)
   - Which criteria should be used and when ?
   - When does the extra effort pay off ?
5. Integrate high-end testing with development

Needs From Educators

1. Disguise theory from engineers in classes

2. Omit theory when it is not needed

3. Restructure curricula to teach more than test design and theory
   - Test automation
   - Test evaluation
   - Human-based testing
   - Test-driven development

Criteria Summary

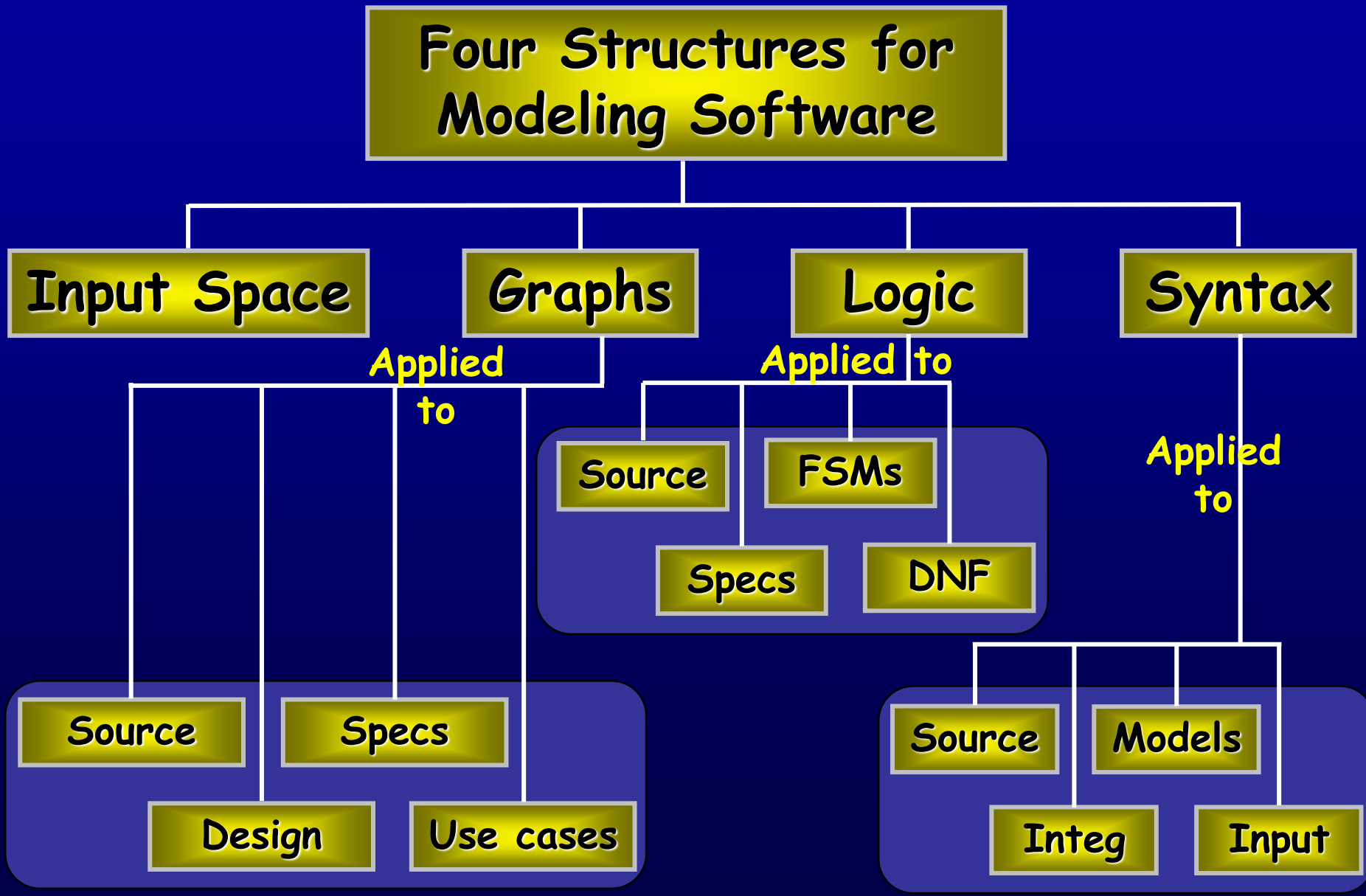
- Many companies still use "monkey testing"
  - A human sits at the keyboard, wiggles the mouse and bangs the keyboard
  - No automation
  - Minimal training required
- Some companies automate human-designed tests
- But companies that use both automation and criteria-based testing

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, test automation frameworks
4. Test Driven Development
5. Criteria-based test design
   - Four structures – test requirements and criteria

Earlier and better testing empowers test managers

--