درس آزمون نرم افزار

تست نرم افزار

خوشنویس

مفاهیم کلیدی

- چرا تست؟
- چه چیزی را تست کنیم؟
  - فعالیتهای تست
  - معیارهای تست
  - اصطلاحات تست
  - ساختارهای چهارگانه
  - مفهوم پوشش
- چگونه تست کنیم؟
  - سطوح بلوغ تست
  - نحوه بهبود تست

سه سؤال مهم

1. چرا تست می‌کنیم؟
2. چه کارهایی باید در طول تست انجام دهیم؟
3. چگونه میتوانیم به آینده تست برسیم؟

تست نرم افزار یک انقلاب و تحول است و ما در حال حاضر در میانه این انقلاب هستیم.

بالاخره در حال شروع به استفاده کردن از نتایج تحقیقات در عمل هستیم!

چند خرابی جالب در نرم افزارها

```
++CDatabase::_stats.mem_used_u
_params.max_unrelevance = (int
if (_params.max_unrelevance <
    _params.max_unrelevance =
_params.min_num_clause_lits_fo
if (_params.min_num_clause_lit
    _params.min_num_clause_lit
_params.max_num_clause_le
if (_params.max_conflict_claus
    _params.max_conflict_claus
CHECK(
cout << "Forced to reduce unre
cout <<"MaxUnrel: " << _params
     << "  MinLenDel: " << _pa
     << "  MaxLenCL : " << _pa
    );
```

اولین اشکالها (bug)

حشره (باگ) Hopper:
یک «بید» که در یک رله در یک ماشین قدیمی گیر افتاده بود.

«این اتفاق تقریبا در تمام اختراعات من وجود داشته است. اولین گام یک فکر ناگهانی است. سپس مشکلات خود را نشان می دهند؛ سپس خطاهای کوچک خود را به نمایش می گذارند، و ماهها کار و مطالعه زیاد برای حل این مسائل لازم است.»

Thomas Edison

«یک فرایند تحلیل باید اجرا شود تا بتواند داده های ضروری عملیاتی را برای موتور تحلیل آماده کند و ممکن است در اینجا منبعی از خطاهای ممکن وجود داشته باشد. اگر فرض کنیم مکانیزم واقعی بدون خطا است، ممکن است مشکلات پیش بینی نشده ای ایجاد شود»

– Ada, Countess Lovelace

(یادداشتی بر موتور تحلیلی Babbage)

عدم موفقیت در تولید نرم افزار

- کاوشگر مریخ ناسا، در سپتامبر ۱۹۹۹ به خاطر نقص در یکپارچه سازی واحدهای آن سقوط کرد – بیشتر از ۵۰ میلیون دلار!

- خسارات عظیمی ناشی از شکست برنامه های کاربردی وب بوده است.
  - خدمات اقتصادی: ۶.۵ میلیون دلار در هر ساعت
  - فروش کارتهای اعتباری: ۲.۴میلیون دلار در هر ساعت

- در دسامبر ۲۰۰۶ پیشنهاد BOGO سایت آمازون به تخفیف دو برابر تبدیل شد.

- در سال ۲۰۰۷ symantec اعلام کرد که اکثر آسیب پذیری های امنیتی ناشی از نرم افزارهای دارای نقص است.
- تست قویتر می تواند اغلب این مشکلات را حل کند

خسارت مالی به علت نرم افزار ضعیف، در سطح جهان، بسیار حیرت انگیز است.

چرا تست نرم افزار مهم است؟

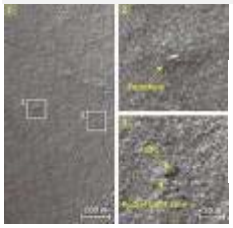
- گزارش NIST حاکی از تاثیرات اقتصادی ناشی از زیر ساختهای ناکافی تست نرم افزار است (۲۰۰۲).
  - خسارت ناشی از تست ناکافی نرم افزار به تنهایی در ایالات متحده بین ۲۲ و ۵۹ میلیارد دلار در سال است.
  - روش های بهتر می توانند این هزینه را به نصف برسانند.

- شکست های اساسی:
  - انفجار آریان ۵، مریخ پیمای Mars Polar Lander، اشکال FDIV پنیوم اینتل

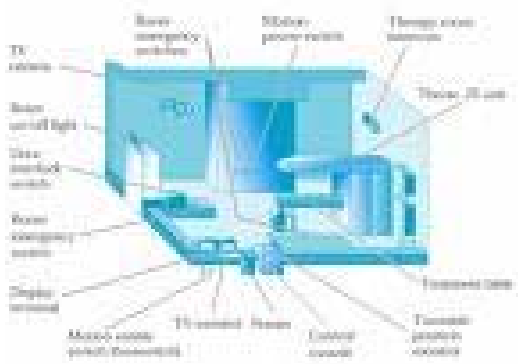
- تست ناکافی نرم افزار های safety-critical (که ایمنی در آنها بسیار مهم است) می تواند هزینه های جانی داشته باشد:
  - ماشین رادیواکتیوTHERAC-25موجب مرگ ۳ تن شد!

- ما میخواهیم برنامه هایمان قابل اطمینان باشند.
  - تست، راه حل بررسی این موضوع است.

Ariane 5:
اشکال در دستگیری خطا: موجب خودتخریبی در حین پرواز شد (تبدیل ۶۴ بیت به ۱۶ بیت. خسارت: حدود ۳۷۰ میلیون دلار)



محل سقوط Mars Polar Lander

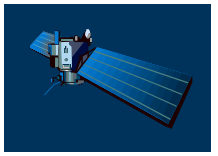


طراحی THERAC-25

نرم افزار پوسته ای است بر تمدن ما

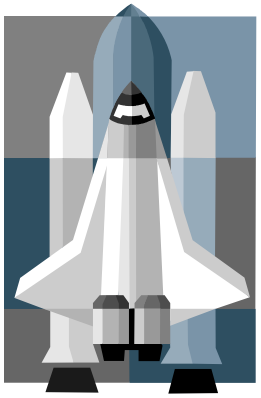
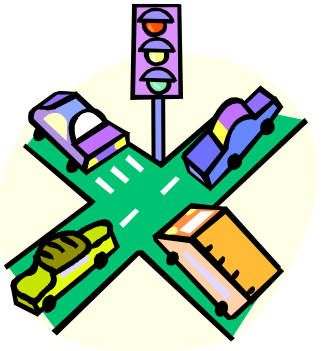










نرم افزار کنترل ایمنی ایرباس ۳۱۹

از دست دادن حالت autopilot

از دست دادن چراغ ها و نور هواپیما و سیستم مخابره داخل هواپیما

از دست دادن صفحه نمایش ناوبری و صفحه نمایش اصلی پرواز هم در سمت فرمانده و هم برای کمک خلبان

خاموشی سال ۲۰۰۳ در شمال شرقی قاره امریکا

۵۰۸ واحد تولید برق و ۲۵۶ نیروگاه تعطیل شد

۱۰ میلیون نفر را در اونتاریو کانادا تحت تأثیر قرار داد

۴۰ میلیون نفر را در ۸ ایالت مختلف ایالات متحده امریکا تحت تأثیر قرار داد

خسارت مالی
۶ میلیارد دلاری

سیستم زنگ خطر در سیستم مدیریت انرژی به علت خطای نرم افزار دچار خرابی شد و اپراتورها از اضافه بار برق در سیستم اطلاع پیدا نکردند.

تست در قرن ۲۱

اهمیت تست نرم افزار در حال افزایش است

- ما در یک دوره گذار هستیم
- نرم افزار رفتار را تعریف می کند
  - روترهای شبکه ها، امورمالی،
  - سویچ های شبکه و سایر زیرساخت ها
- امروزه بازار نرم افزار :
  - بسیار بزرگتر است
  - رقابتی تر است
  - کاربران بیشتری دارند

صنعت در حال انقلابی در کاربرد تست برای موفقیت محصولات نرم افزاری است

- فرایندهای چابک تولید نرم افزار فشار روی تست کننده ها را افزایش می دهد.
- برنامه های کنترلی تعبیه شده
  - هواپیماها،کنترل ترافیک هوایی
  - سفینه های فضایی
  - ساعت ها
  - اجاق های فر
  - ریموت کنترل ها
  - PDAs
  - صندلیهای حافظه دار
  - دستگاه های DVD
  - بازکننده های درب گاراژ
  - تلفن های همراه

تست در قرن ۲۱

- نرم افزارهای بلادرنگ و ایمن، بیشتر شده اند.
- برنامه های سازمانی به معنای برنامه های بزرگتر و کاربران بیشتر است.
- نرم افزار های تعبیه شده در همه جا موجود هستند.
- به طور تناقض آمیزی، انتظارات ما از نرم افزارهای رایگان بیشتر هم هست.
- در حال حاضر امنیت تماماً مربوط به نقص های نرم افزار است.
  - نرم افزار امن، نرم افزاری قابل اطمینان است.
- web ،یک سکوی جدید برای استقرار برنامه ها ارائه کرده است.
  - برای کاربران بسیار دسترس پذیر و نیز بسیار رقابتی است.
  - برنامه های کاربردی وب،توزیع شده هستند.
  - برنامه های کاربردی وب، باید تا حد زیادی قابل اطمینان باشند.

صنعت با نا امیدی، به اختراعات نرم افزاری ما نیازمند است!

عدم تطابق در نیازها و اهداف

- صنعت راغب به آن است که تست کردن ساده و آسان باشد.
  - آزمونگران نیازی به دانش پیش زمینه در کامپیوتر و ریاضیات نداشته باشند.

- دانشگاه ها در حال فارغ التحصیل کردن دانشمندان هستند.
  - صنعت به مهندسان نیازمند است.

- نیاز است که تست ها با سختگیری و دقت بیشتری انجام شوند.
- فرایندهای چابک توقعات زیادی از تست دارند.
  - برنامه نویس ها باید تست واحد (unit) انجام دهند؛ آن هم بدون نیاز به آموزش، تحصیل یا ابزار
  - تست ها مولفه هایی کلیدی در نیازمندی های کارکردی هستند - اما چه کسی چنین تست‌هایی را ایجاد می‌کند؟

حاصل: تعداد زیادی نرم افزار مشکل دار

هزینه تست

خواه یا ناخواه، حداقل نصف بودجه توسعه روی تست کردن صرف خواهد شد!

- در دنیای واقعی ، تست کردن یک فعالیت اساسی پس از طراحی می باشد.
- محدود کردن تست زودهنگام معمولاً هزینه را افزایش می دهد
- یکپارچه سازی وسیع سخت افزار-نرم افزار، نیازمند تست بیشتری است.

سؤال اول: چرا تست؟

در صورتی که ندانید چرا در حال تهیه یک تست هستید،آن تست چندان مفید نخواهد بود!

- اهداف و نیازمندیهای مکتوب تست به ندرت ایجاد می شود.
- سطح پوشش برنامه ریزی شده شما در پروژه چیست؟
- چقدر تست کافی است؟
- هدف مشترک – صرف بودجه ...

چرا تست؟

در صورتی که قبل از هر تست، وقتی نیازمندی های کارکردی شکل گرفته اند، شروع به برنامه ریزی کردن برای آن نکرده باشید، هیچگاه نخواهید دانست که چرا تست را انجام می دهید.

- سال ۱۹۸۰: نرم افزار باید به راحتی قابل نگهداری باشد.
- آستانه نیازمندی های قابلیت اطمینان چقدر است؟
- هر تست در حال تلاش برای تصدیق چه حقیقتی است؟
- تیم های تعریف نیازمندیها، باید شامل آزمونگران باشد.

هزینه های تست نکردن

مدیران اغلب می گویند: تست بیش از حد گران است.

- تست نکردن به مراتب گران تر است.
- برنامه ریزی برای تست پس از توسعه به طرز ممانعت کننده ای گران است.
- یک ایستگاه تست برای بردهای مداری هزینه نیم میلیون دلاری دارد.
- ابزار های تست نرم افزار معمولاً کمتر از ۱۰.۰۰۰ دلار هستند.

هشدار: اثر ابزارها و تکنیک های جدید

«دانستن کافی نیست، باید عمل کنیم. خواستن کافی نیست، باید کارهارا انجام دهیم»
-Goethe

سؤال دوم: چه کاری؟

اما ... چه کاری باید انجام دهیم؟

1. انواع فعالیت های تست
2. اصطلاحات تست نرم افزاری
3. تغییر مفاهیم تست
   - معیارهای پوشش تست
   - معیارهایی که بر اساس ساختارها هستند

انواع فعالیتهای تست

- تست را می توان به چهار نوع کلی از لحاظ فعالیت ها تقسیم بندی کرد:
  1. طراحی تست
     a.1) بر اساس معیارها
     b.1) بر اساس انسان
  2. خودکار سازی تست
  3. اجرای تست
  4. ارزیابی تست
- هر نوعی از فعالیت نیازمند مهارت های مختلف، دانش پیش زمینه،تحصیلات و آموزش است.
- هیچ سازمان توسعه نرم افزار معمولی از افراد یکسان برای انواع مختلف کارها، شامل: نیازمندیها، طراحی، پیاده سازی، یکپارچه سازی و کنترل پیکر بندی استفاده نمی‌کند.

چرا سازمان های تست ، هنوز از افراد یکسان برای تمام فعالیت های تست استفاده می‌کنند؟

این کار به وضوح، هدر دادن منابع است.

۱. طراحی تست – (a) بر اساس معیارها

طراحی تست برای برآورده کردن معیارهای پوشش یا دیگر اهداف مهندسی، ارزشمند است.

- این کار تکنیکی ترین کار در تست نرم افزار است
- نیازمند دانش در این زمینه هاست
  - ریاضیات گسسته
  - برنامه نویسی
  - تست
- طراحی تست ،نیازمند مدرک علوم کامپیوتر معمولی است.
- این کار از نظر فکری فعالیتی چالش بر انگیز و پاداش دهنده است.
- طراحی تست در توسعه، با معماری نرم افزار قابل مقایسه است.
- قطعاً به کار گیری افرادی که در زمینه ی طراحی تست مهارت ندارند، می تواند موجب تولید تست هایی شود که اثربخش نخواهند بود.

۱. طراحی تست بر اساس انسان(b)

طراحی مقادیر تست بر اساس دانش درباره دامنه برنامه و دانش انسانی از تست

- این روش از آنچه که توسعه دهندگان فکر می کنند ، بسیار سخت‌تر است.
- روش های مبتنی بر معیارها گاهی برای برخی از شرایط قابل استفاده نیستند.
- نیازمند دانش در زمینه های زیر است:
  - دامنه،تست، و واسط های کاربری
- تقریباً به هیچ علم کامپیوتری سنتی نیاز ندارد.
  - پیش زمینه در دامنه نرم افزار ضروری است.
  - پیش زمینه تجربی بسیار مفید است (مثلا زیست شناسی، روانشناسی و...).
  - پیش زمینه منطقی بسیار مفید است (مثل حقوق، فلسفه، ریاضیات و..).
- طراحی تست، از نظر فکری چالش برانگیز، و پاداش دهنده است.
  - اما نه برای رشته های علوم کامپیوتر زیرا این افراد هدفشان حل مسئله و ساختن چیزها است.

۲. خودکارسازی تست

تعبیه مقادیر تست در اسکریپت های قابل اجرا

- این طراحی ، از نظر فنی و تکنیکی در سطح پایین قرار دارد
- کمی غیرفنی تر است.
- نیازمند دانش برنامه نویسی است.
  - برنامه نویسی نسبتا ساده و قابل فهم – قطعات کوچک کد و الگوریتم های ساده
- به تئوری و نظریه بسیار اندکی نیاز دارد.
- برای طراحان تست، بسیار خسته کننده و کسالت آور است.
- برنامه نویسی برای بسیاری از متخصصان دامنه، دور از دسترس است.
- چه کسی مسئول تعیین و تعبیه کردن خروجی مورد انتظار است؟
  - ممکن است طراحان تست، همیشه خروجی مورد نظر را ندانند.
  - ارزیابی کنندگان تست باید برای کمک بیشتر زودتر وارد فرآیند تست شوند.

۳. اجرای تست

اجرای تست روی نرم افزار و ثبت نتایج

- اجرای تست کاری آسان است – و اگر مرحله خودکارسازی بخوبی انجام شده باشد، ناچیز و جزیی هم هست.
- به مهارت های پایه ای کامپیوتر نیاز دارد.
  - کارآموزان
  - افراد بدون پیش زمینه ی فنی
- اگر طراحان تست، آنرا اجرا کنند، این کار حتماً آنها را متقاعد می‌کند که به دنبال یک شغل دیگر در زمینه توسعه نرم افزار باشند!! (بسیار کسالت آور)
- اگر، مثلاً، تست های GUI به خوبی خودکار سازی نشده باشند حجم کار زیادی به صورت دستی باید انجام شود.
- مجریان تست باید دقت بالایی داشته و در ثبت نتایج بسیار دقیق و باریک بین باشند.

۴. ارزیابی تست

ارزیابی نتایج حاصل از آزمایش، گزارش به توسعه دهندگان

- بسیار سخت تر از آن است که به نظر می رسد.
- نیاز به دانش:
  - دامنه
  - آزمایش کردن
  - واسط کاربری و روانشناسی
- تقریباً به هیچ علم کامپیوتری سنتی نیاز ندارد.
  - پیش زمینه در دامنه نرم افزار ضروری است.
  - پیش زمینه تجربی بسیار مفید است (مثلا زیست شناسی، روانشناسی و...).
  - پیش زمینه منطقی بسیار مفید است (مثل حقوق، فلسفه، ریاضیات و..).
- این کار از نظر فکری چالش برانگیز، و پاداش دهنده است.
  - اما نه برای رشته های علوم کامپیوتر زیرا این افراد هدفشان حل مسئله و ساختن چیزها است.

سایر فعالیتها

- مدیریت تست: تعیین خط مشی ها، سازماندهی تیم، ارتباط با تیم توسعه، انتخاب معیارها، تعیین میزان خودکار سازی مورد نیاز و ...

- نگهداری تست: تست باید برای استفاده مجدد در طی تکامل نرم افزار ذخیره شود.
  - نیاز به همکاری طراحان آزمون و خودکارسازان آزمون
  - تصمیم گیری در مورد زمان حذف مجموعه های تست تا حدودی مربوط به خط مشی ها وتا حدودی نیز فنی است - و به طور کلی، بسیار سخت است!
  - آزمون باید در فرایند کنترل پیکربندی وارد شود.
- مستندسازی تست: تمام طرفها باید مشارکت کنند.
  - برای هر آزمون باید پاسخ به سوال «چرا» مستند شود- دلیل انتخاب معیارها و نیازمندیهای آزمون که برآورده شده اند؛ یا دلیل منطقی برای آزمونهای طراحی شده توسط انسان
  - قابلیت ردیابی در سراسر فرایند باید تضمین شود.
  - مستندات باید در کنار تست های خودکار سازی شده نگهداری شود.

تعداد تقریبی پرسنل

- در یک سازمان آزمون بالغ: تنها یک طراح آزمون که با چند خودکارساز آزمون، مجری آزمون و ارزیاب آزمون کار می کند.
- بهبود خودکارسازی موجب کاهش مجریان تست می شود
  - از لحاظ تئوری به صفر (اما در عمل اینطور نیست)
- انتساب افراد اشتباه به کارهای نامتناسب با آنها منجر به ناکارآمدی، رضایت شغلی پایین و عملکرد شغلی پایین می شود.
  - یک طراح تست ماهر با کارهای دیگر خسته خواهد شد و به دنبال یک شغل دیگر در زمینه توسعه نرم افزار خواهد گشت.
  - یک ارزیاب ماهر آزمون مزایای معیارهای آزمون را درک نخواهد کرد.
- ارزیابان تست، از دانش دامنه برخوردارند، پس باید آزاد باشند تا تستهایی را اضافه کنند که در فرایندهای نرم افزاری دیده نمی شوند.

انواع فعالیتهای آزمون – خلاصه

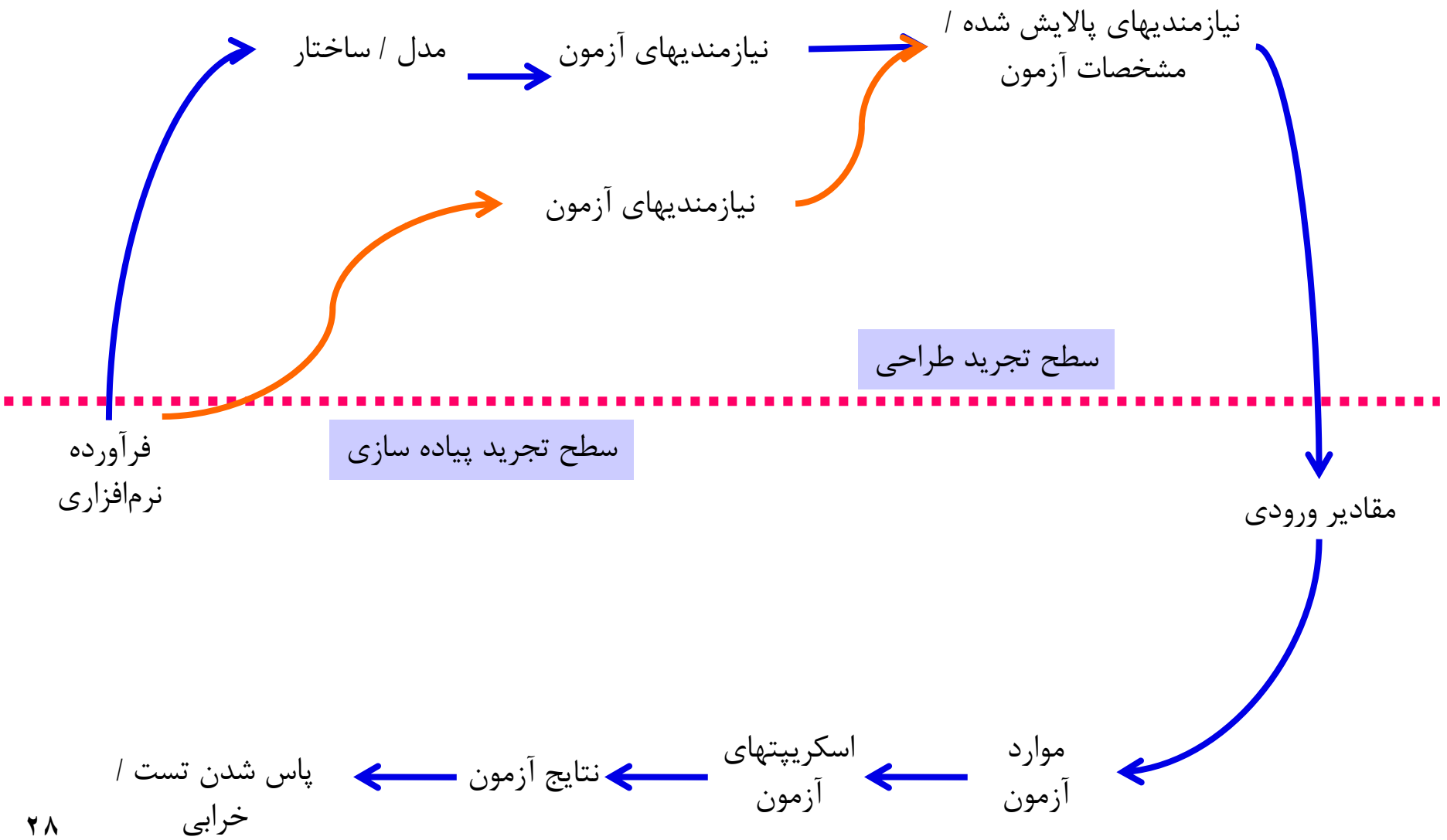
1a.	طراحی (مبتنی بر معیارها)	طراحی مقادیر تست برای برآوردن اهداف مهندسی نیاز به دانش ریاضیات گسسته، برنامه نویسی و تست
1b.	طراحی (مبتنی بر انسان)	طراحی مقادیر تست بر اساس دانش دامنه و فعالیت ذهنی نیاز به دانش دامنه، واسط کاربری و تست
2.	خودکارسازی	تعبیه مقادیر تست در اسکریپتهای قابل اجرا نیاز به دانش اسکریپت نویسی
3.	اجرا	اجرای تستها روی نرم افزار و ثبت نتایج نیاز به دانش بسیار کم
4.	ارزیابی	ارزیابی نتایج حاصل از آزمایش، و گزارش به توسعه دهندگان نیاز به دانش دامنه

اکثر تیم های آزمون از افراد یکسان برای هر چهار فعالیت استفاده می کنند، که با توجه به این که این فعالیتها کاملاً با هم متفاوت هستند موجب هدر رفتن منابع می شود

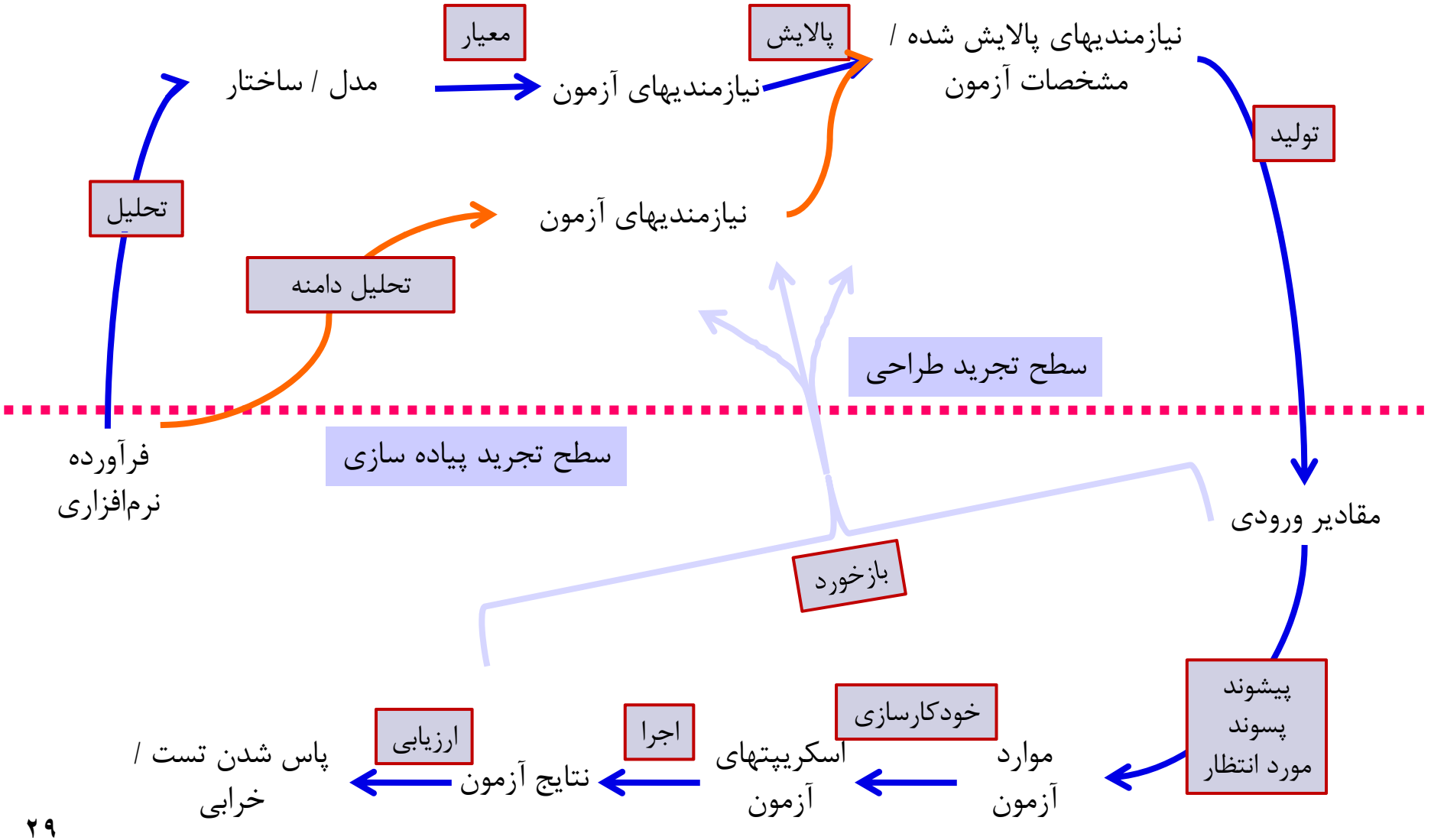
برای استفاده مؤثر از افراد، و تست کارآمد، نیاز به فرایندی داریم که به طراحان اجازه دهد که:

سطح تجرید خود را بالاتر ببرند.

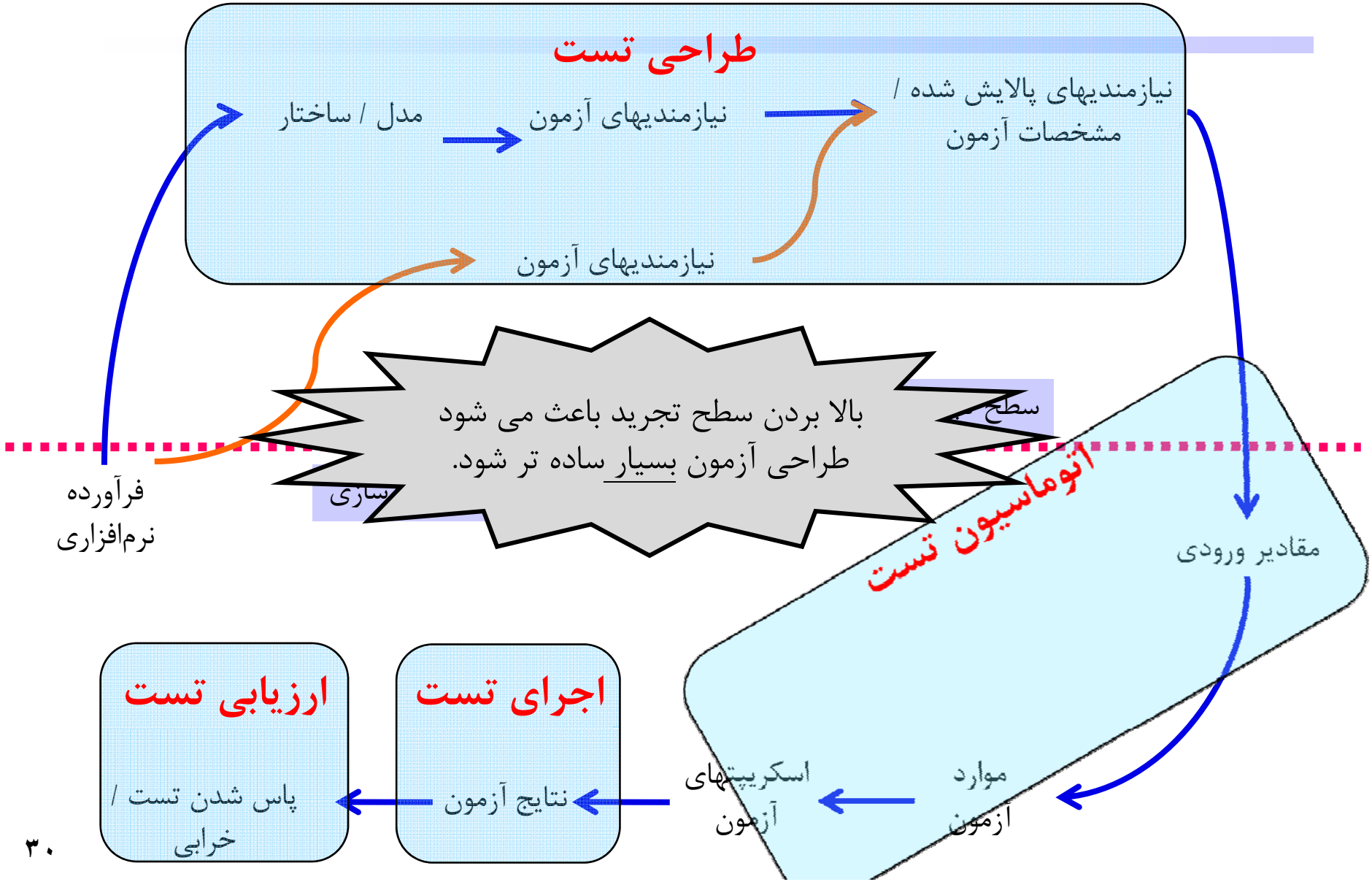
طراحی تست مدل‌رانه (Model-Driven)



طراحی تست مدل‌رانه – مراحل



طراحی تست مدل‌رانه – فعالیتها



اصطلاحات مهم اعتبار سنجی و بازبینی (V&V) (IEEE)

- اعتبارسنجی (Validation): فرایند ارزیابی نرم افزار در پایان توسعه نرم افزار برای اطمینان از انطباق با کاربری مورد نظر.

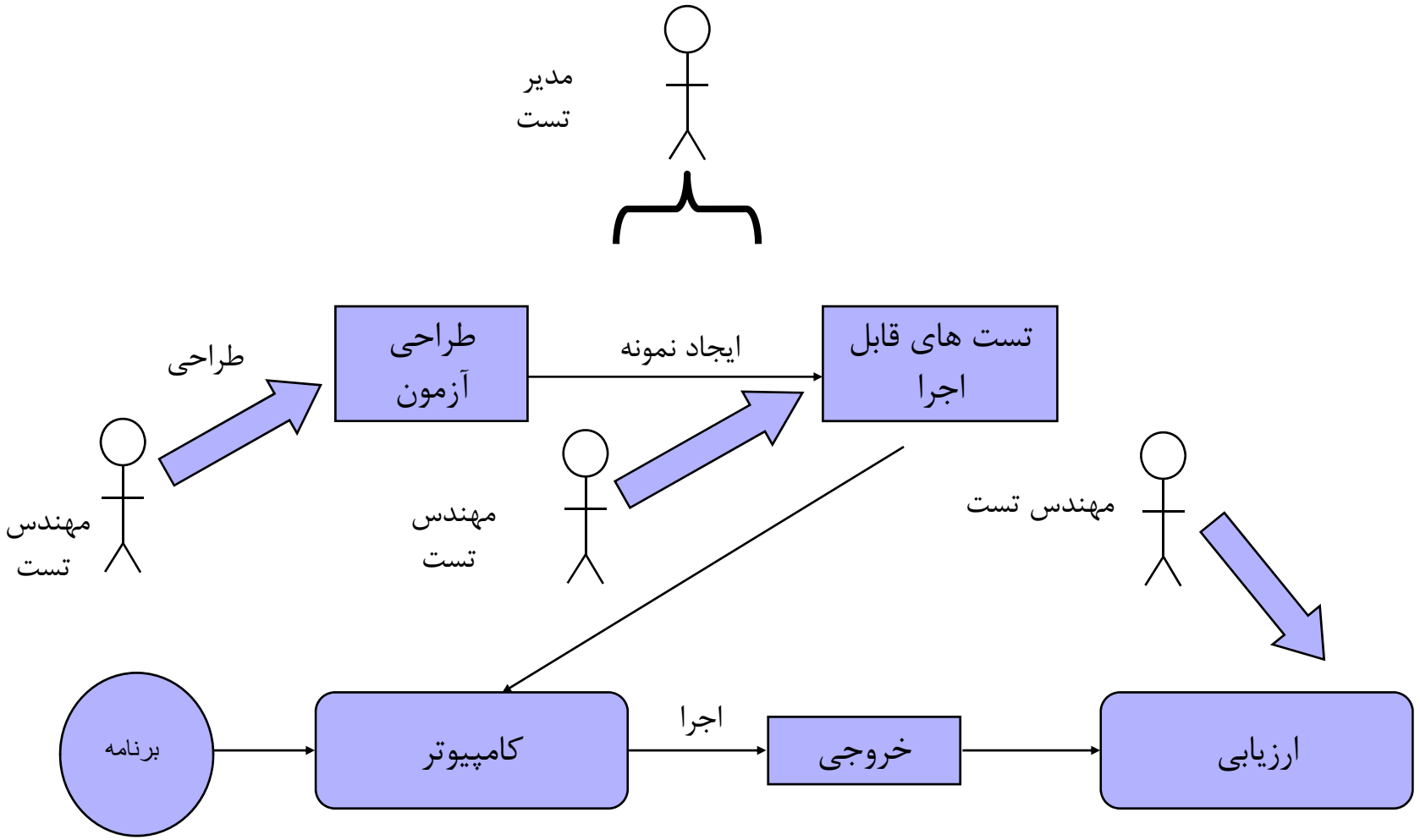
- بازبینی (Verification): فرایند تعیین این که آیا محصولات یک فاز معین از فرایند توسعه نرم افزار، نیازمندیهای تعیین شده در فاز قبلی را برآورده می کنند یا خیر.

مهندس آزمون و مدیر آزمون

- مهندس آزمون: متخصص IT که مسئول یک یا چند فعالیت فنی آزمون است
  - طراحی ورودی های آزمون
  - تولید مقادیر آزمون
  - اجرای اسکریپت های آزمون
  - تحلیل نتایج
  - گزارش نتایج به توسعه دهندگان و مدیران

- مدیر آزمون: مسئول یک یا چند مهندس آزمون
  - تعیین مجموعه خط مشی ها و فرایندهای آزمون
  - تعامل با مدیران دیگر بر روی این پروژه
  - یا کمک به مهندسان آزمون

فعالیتهای مهندس تست



تست ایستا و تست پویا

- **تست ایستا (استاتیک): تست بدون اجرای برنامه**
  - مثل بازرسی های نرم افزار و برخی از انواع تحلیل
  - بسیار مؤثر در یافتن انواع خاصی از مشکلات، بخصوص نقصهای «بالقوه». یعنی مشکلاتی که در صورت تغییر برنامه می تواند موجب بروز نقص شود.
- **تست پویا (داینامیک): تست با اجرای برنامه با ورودی های واقعی**

چند اصطلاح: نقص، خطا و خرابی – تست و اشکال‌زدایی

- نقص نرم افزار (Fault): یک ایراد ایستا در نرم افزار
- خطای نرم افزار(Error): یک حالت داخلی نادرست که در نتیجه وجود یک نقص ظاهر می‌شود
- خرابی نرم افزار (Failure): رفتار خارجی نادرست با توجه به نیازمندیها و یا سایر توصیفها از رفتار درست

نقصهای نرم افزار ناشی از اشتباه های طراحی بوده و همواره وجود خواهند داشت.

- تست: پیدا کردن ورودی که باعث می‌شود نرم افزار با خرابی مواجه شود
- اشکال زدایی (Debug): روند یافتن یک نقص با توجه به یک خرابی معین

مدل نقص و خرابی

- سه شرط لازم برای مشاهده شدن خرابی (RIP)

1. قابلیت دسترسی (Reachability): باید بتوان به مکان یا مکانهایی از برنامه که دارای نقص هستند رسید (دسترسی پیدا کرد).
2. آلوده سازی (Infection): حالت برنامه باید نادرست باشد.
3. انتشار (Propagation): حالت اشتباه برنامه باید منتشر شده و موجب غلط شدن برخی خروجی های برنامه شود.

مورد آزمون (test case)

- شامل دو بخش است:

- **مقادیر مورد آزمون (values)**: مقادیری که مستقیماً یک نیازمندی آزمون را برآورده می‌کند.
  - به عبارت دیگر: ورودیهای طراحی شده برای یک نیازمندی آزمون

- **نتایج مورد انتظار(expected)**: نتیجه ای که هنگام اجرای آزمون در صورت برآورده کردن رفتار در نظر گرفته شده توسط برنامه، تولید خواهد شد
  - به عبارت دیگر: خروجی‌هایی (صحیح) که انتظار می رود در اثر اجرای برنامه با ورودیهای بالا تولید شود

مشاهده پذیری و کنترل پذیری

مشاهده پذیری (Observability) نرم افزار:

- **مشاهده رفتار برنامه در قالب خروجی های آن، آثار آن بر محیط، و سایر مؤلفه های سخت افزاری و نرم افزاری چقدر آسان است**
  - نرم افزارهایی که بر دستگاه های سخت افزاری، پایگاه داده ها، و یا فایل های راه دور تاثیر دارند از مشاهده پذیری پایینی برخوردار هستند.

کنترل پذیری (Controllability) نرم افزار:

- **فراهم کردن ورودی های مورد نیاز برای نرم افزار در قالب مقادیر آنها، عملیات و رفتارهای مورد نظر چقدر آسان است**
  - نرم افزاری که ورودیهای خود را از کیبورد بگیرد کنترل پذیری بالایی دارد
  - نرم افزاری که ورودی خود را از سنسورهای سخت افزاری و یا نرم افزارهای توزیع شده دریافت می کند کنترل آن سخت تر است
  - تجرید داده کنترل پذیری و مشاهده پذیری را کاهش می دهد

ورودی برای تأثیر بر کنترل پذیری و مشاهده پذیری

- **مقادیر پیشوند:**
  - هر ورودی لازم برای قرار دادن نرم افزار در حالت مناسب برای دریافت مقادیر مورد آزمون

- **مقادیر پسوند:**
  - هر ورودی که نیاز است پس از مقادیر مورد آزمون به نرم افزار ارسال شود
  - دو نوع مقدار پسوند
    - ۱. مقادیر بازبینی (verification values): مقادیر لازم برای مشاهده نتایج مقادیر موارد آزمون
    - ۲. فرمانهای خروج (exit commands): مقادیر لازم برای پایان دادن به اجرای برنامه و یا در غیر این صورت برگرداندن آن به یک حالت پایدار

- **اسکریپت های قابل اجرای آزمون:**
  - یک مورد آزمون که به شکلی آماده شده است که به طور خودکار روی نرم افزار اجرا شده و گزارشی را تولید نماید

تست بالا به پایین و پایین به بالا

- **تست بالا به پایین:**
  - ابتدا پروسیجر اصلی تست می شود، سپس با حرکت رو به پایین پروسیجرهایی را که پروسیجر اصلی آنها را فراخوانی می‌کند تست می‌شوند؛ و به همین ترتیب.

- **تست پایین به بالا :**
  - تست برگ ها در درخت فراخوانی (پروسیجر هایی که هیچ فراخوانی را انجام نمی‌دهند)، و حرکت به بالا به سمت ریشه
  - هر پروسیجر تست نمی‌شود؛ تا وقتی که همه فرزندان آن تست شده باشند.

تست جعبه سیاه و تست جعبه سفید

- **جعبه سیاه تست:**
  - استخراج تست بر اساس توصیف خارجی نرم افزار، شامل مشخصات، نیازمندیها و طراحی

- **تست جعبه سفید:**
  - استخراج تست بر اساس جزئیات داخلی کد منبع، مشخصاً: انشعاب ها، شرطهای منفرد، و عبارتهای دستوری

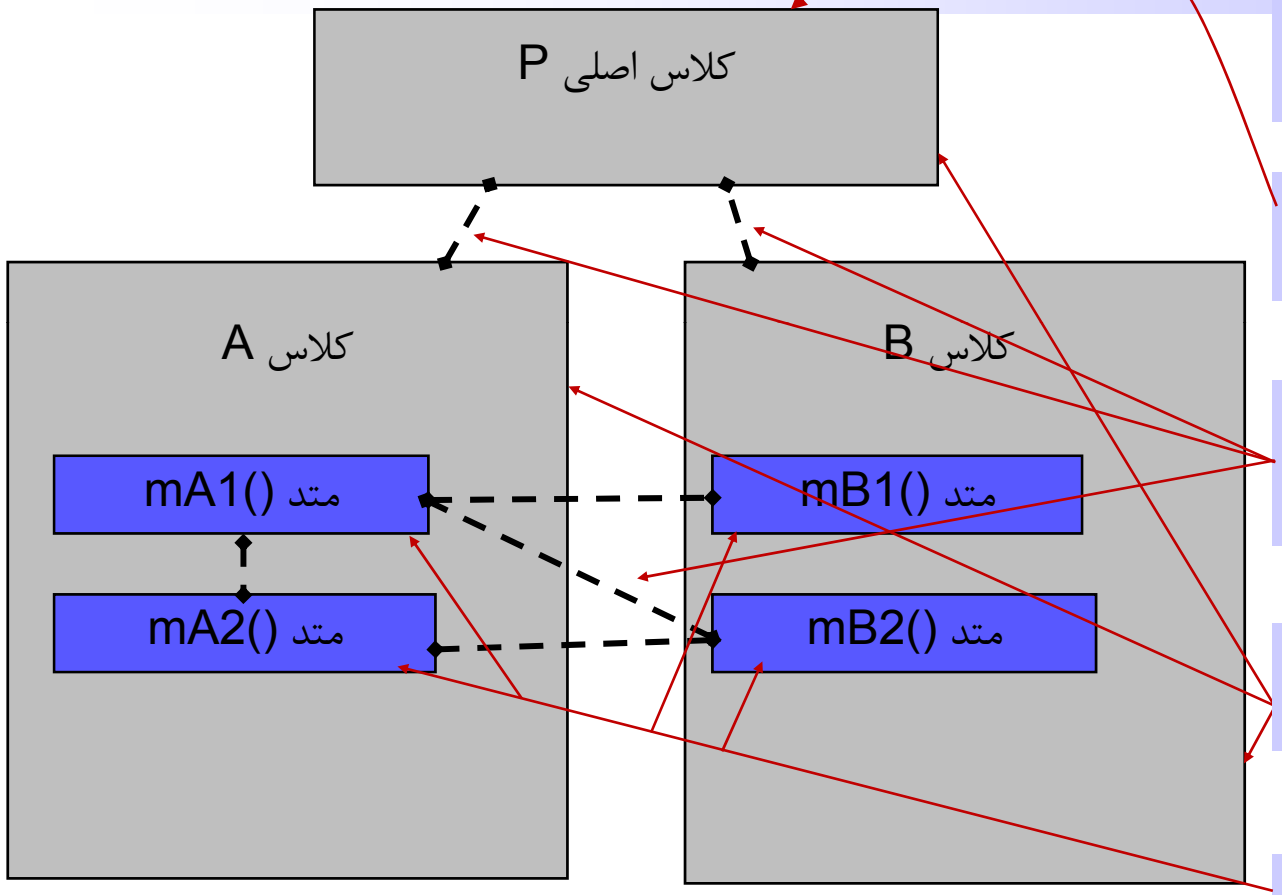
این دیدگاه قدیمی و از رده خارج است .

سوال کلی تر این است: از کدام سطح تجرید باید تست ها را استخراج کنیم؟

تغییر مفاهیم تست

- دیدگاه قدیمی تست، آزمون را در فازهای معین توسعه در نظر می گیرد
  - آزمون واحد، آزمون ماژول، آزمون یکپارچه سازی، آزمون سیستم و ...
- دیدگاه جدید بر حسب ساختارها و معیارها است
  - گرافها ، عبارات منطقی، نحو، فضای ورودی

قدیمی: تست در سطوح مختلف



- آزمون پذیرش: آیا نرم افزار برای کاربر قابل قبول است؟
- تست سیستم: تست عملکرد کلی سیستم
- تست یکپارچه سازی: تست چگونگی تعامل ماژول ها با یکدیگر
- تست ماژول: تست هر کلاس، فایل، ماژول و یا مؤلفه
- تست واحد: تست هر واحد (متد) به طور منفرد

قدیمی: یک گراف پیدا کنید و آن را پوشش دهید

- متناسب با:
  - فرآورده نرم افزاری خاص
    - کد، طراحی، مشخصات
  - یک فاز خاص از چرخه حیات
    - نیازمندیها ، مشخصات، طراحی، پیاده سازی

- گراف ها تمام تکنیک های تست را به خوبی توصیف نمی کنند.

- چهار مدل تجریدی کافی است (بعداً اشاره می شود).

جدید: معیارهای پوشش آزمون

- کار تست کننده ساده است: تعریف مدلی از نرم افزار، سپس، یافتن راهی برای پوشش آن

- نیازمندیهای آزمون : نیاز خاصی که باید در طی تست برآورده شده یا پوشش داده شود

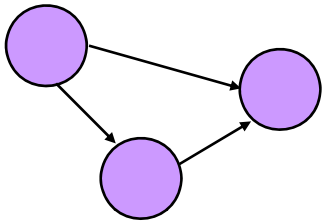
- معیار آزمون :مجموعه ای از قواعد و یک فرایند که نیازمندیهای آزمون را تعریف می کنند

پژوهشگران آزمون، ده ها معیار را تعریف کرده اند، اما اغلب آنها در چهار دسته بندی ساختاری مورد نظر ما می‌گنجند.

جدید: معیارهای مبتنی بر ساختارها

ساختار : ۴ روش برای مدل کردن نرم افزار

1. گراف ها



2. عبارات منطقی

```
(not X or not Y) and A and B
```

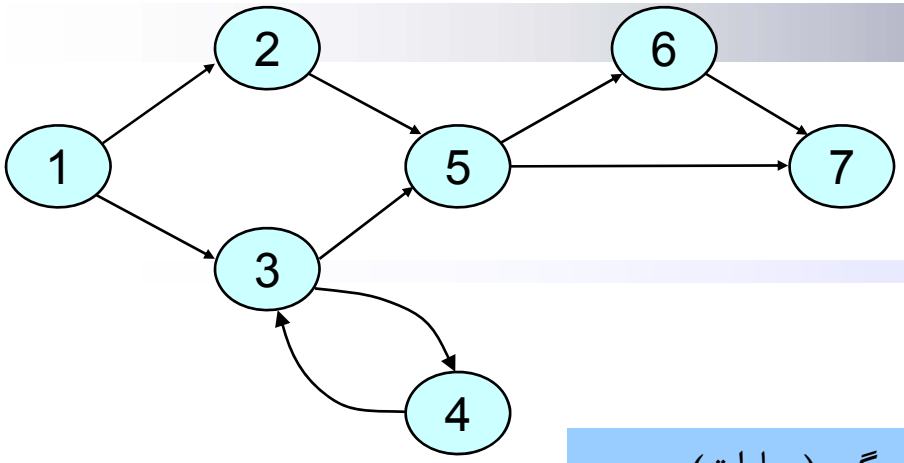
3. تعیین خصوصیات ورودی دامنه

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

4. ساختارهای نحوی

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

۱. پوشش گراف – ساختاری



این گراف می‌تواند نشان‌دهنده موارد زیر باشد:

- عبارتها و انشعابها
- متدها و فراخوانی ها
- مؤلفه ها و سیگنالها
- حالتها و گذارها
- و ...

پوشش گره (عبارات)

هر گره را پوشش دهید:

- 12567
- 1343567

پوشش یال (انشعابها)

هر یال را پوشش دهید:

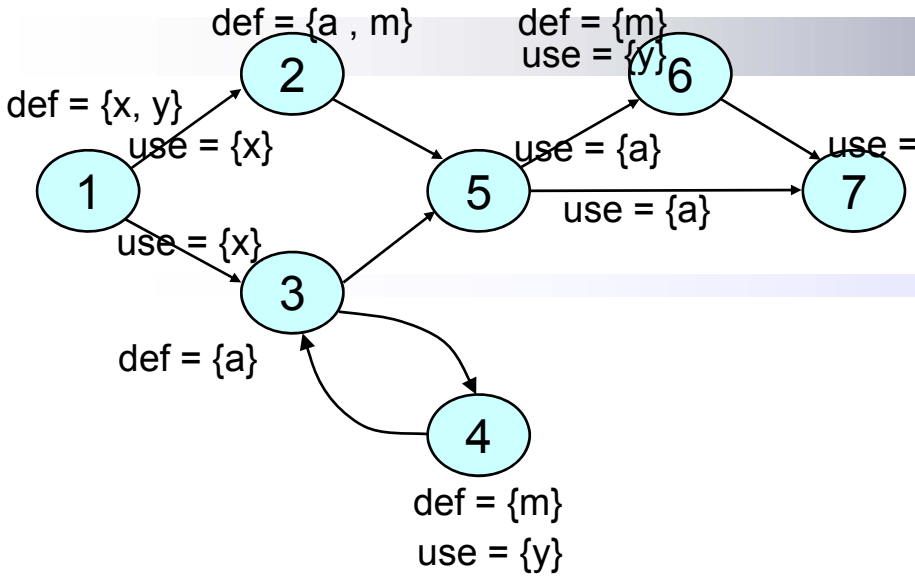
- 12567
- 1343567
- 1357

پوشش مسیر

هر مسیر را پوشش دهید:

- 12567
- 1257
- 13567
- 1357
- 1343567
- 134357 …

۱– گراف پوشش – جریان داده



این گراف حاوی دو فاکتور زیر است :

تعاریف (defها): گره‌ها و یال هایی که در آنها به متغیرها مقادیری نسبت می‌دهیم

کاربردها (useها): گره‌ها و یال هایی که در آنها مقادیر متغیرها مورد دسترسی قرار می‌گیرند.

زوج های تعریف-کاربرد (Def-Use)

- (x, 1, (1,2)), (x, 1, (1,3))
- (y, 1, 4), (y, 1, 6)
- (a, 2, (5,6)), (a, 2, (5,7)), (a, 3, (5,6)), (a, 3, (5,7))
- (m, 2, 7), (m, 4, 7), (m, 6, 7)

پوشش همه تعاریف

هر تعریف به یک کاربرد دسترسی پیدا کند:

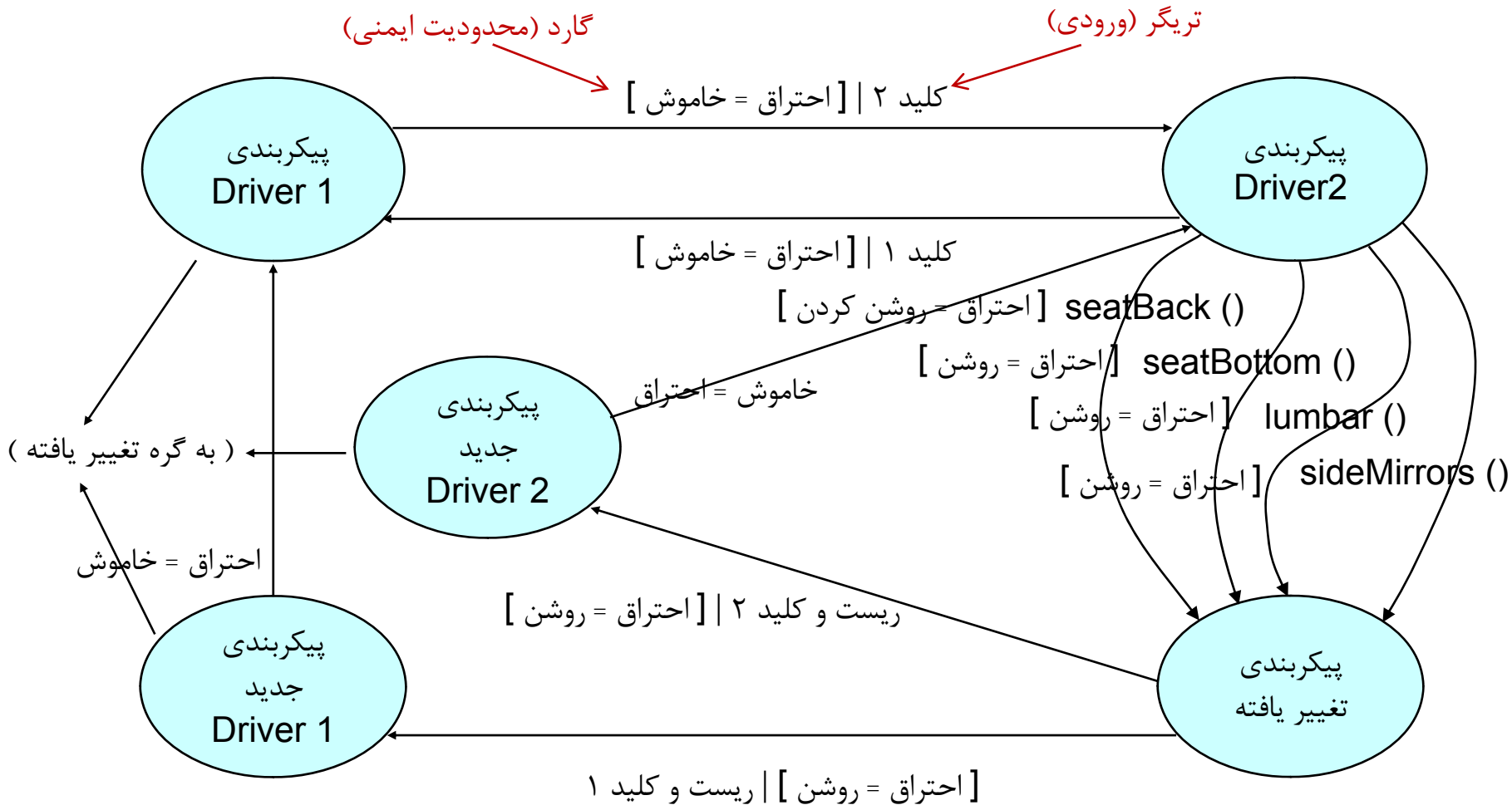
- 1, 2, 5, 6, 7
- 1, 2, 5, 7
- 1, 3, 4, 3, 5, 7

پوشش همه کاربردها

هر تعریف به هر کاربرد دسترسی پیدا کند:

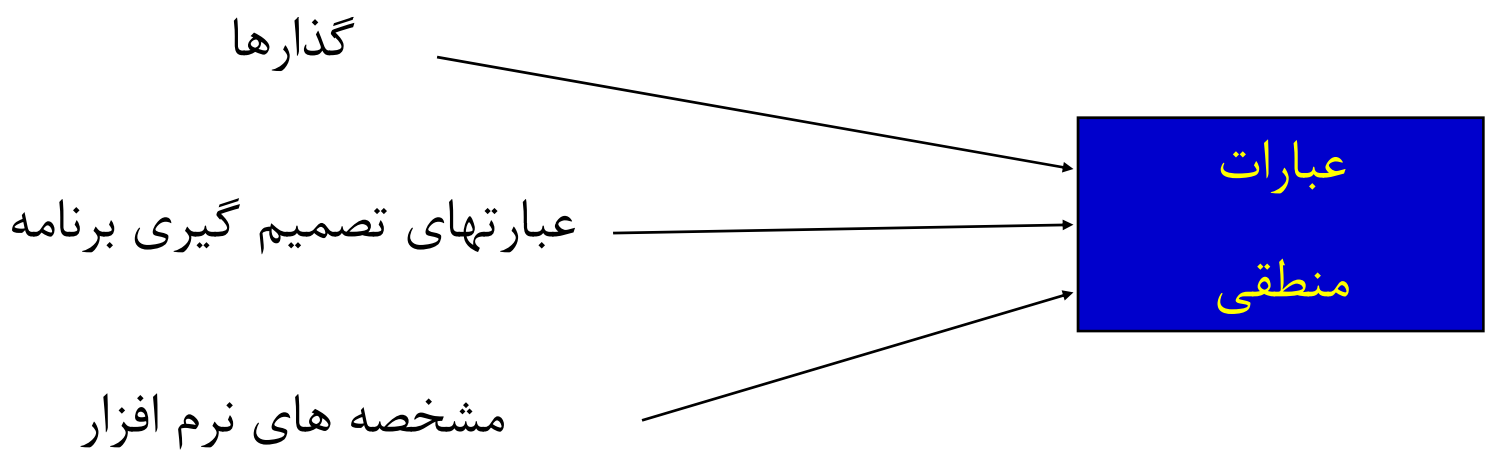
- 1, 2, 5, 6, 7
- 1, 2, 5, 7
- 1, 3, 5, 6, 7
- 1, 3, 5, 7
- 1, 3, 4, 3, 5,7

۱ . گراف – مثالی از FSM
صندلی های حافظه دار در Lexus Es 300



۲. عبارات منطقی

$$((a > b) \text{ or } G) \text{ and } (x < y)$$



۲. عبارات منطقی

$$((a > b) \text{ or } G) \text{ and } (x < y)$$

- پوشش گزاره (predicate): هر گزاره باید یک بار درست (T) و یک بار نادرست (F) باشد.

$$((a>b) \text{ or } G) \text{ and } (x < y) = \text{True, False}$$

- پوشش جزء (clause): هر جزء باید یک بار درست(T) و یک بار نادرست (F) باشد.

$$(a > b) = \text{True, False}$$
$$G = \text{True, False}$$
$$(x < y) = \text{True, False}$$

- پوشش ترکیبیاتی (combinatorial): ترکیبات متعددی از جزءها
  - *پوشش جزء فعال: هر جزء باید تعیین کننده نتیجه گزاره باشد.*

۲.پوشش شرطی منطقی فعال

$$((a > b) \text{ or } G) \text{ and } (x < y)$$

1	T	F	T
2	F	F	T
3	F	T	T
4	F	F	T
5	T	T	T
6	T	T	F

با این مقادیر برای G و (a>b)، (x<y) تعیین کننده مقدار گزاره است.

تکراری

۳. توصیف دامنه ورودی

- توصیف دامنه ورودی نرم افزار
  - شناسایی ورودی‌ها، پارامترها، و ...
  - افراز هر ورودی به مجموعه های متناهی از مقادیر
  - انتخاب ترکیبی از مقادیر
- سطح سیستم
  - تعداد دانشجویان { *0, 1, >1* }
  - مقطع تحصیلی { *600, 700, 800* }
  - رشته { *swe, cs, isa, infs* }
- سطح واحد
  - پارامترها *F (int X, int Y)*
  - مقادیر ممکن *X: { <0, 0, 1, 2, >2 }, Y : { 10, 20, 30 }*
  - تست‌ها
    - *F (-5, 10), F (0, 20), F (1, 30), F (2, 10), F (5, 20)*

۴. ساختارهای نحوی

- مبتنی بر یک گرامر، یا دیگر تعریف های نحوی (syntactic) است.
- نمونه اصلی آن، تست جهش (mutation) است.

1. ایجاد تغییرات کوچک (جهش) در برنامه: تولید جهشگرها (mutants)
2. یافتن تستهایی که موجب خرابی هر جهشگر شوند: کشتن جهشگرها
3. خرابی در اینجا = تفاوت خروجی جهشگر با خروجی برنامه اصلی.
4. بررسی خروجی تست های مفید روی برنامه ی اصلی

- یک برنامه نمونه و جهشگرهای آن

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

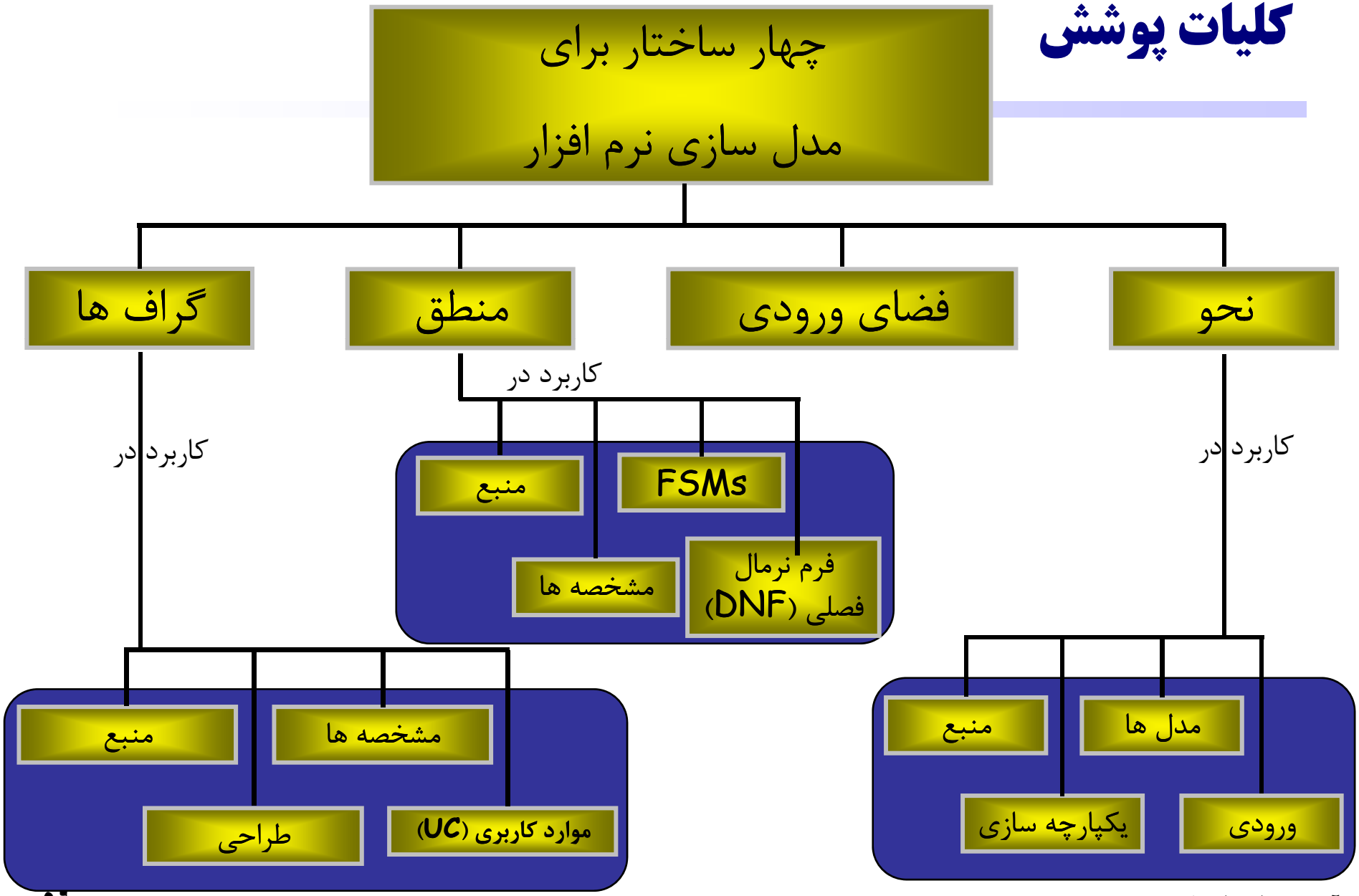
```
if (x > y)
Δif (x >= y)
    z = x - y;
    Δ z = x + y;
    Δ z = x – m;
else
    z = 2 * x;
```

منبع ساختارها

- این ساختارها می‌توانند از تعداد زیادی از فرآورده‌های نرم‌افزاری استخراج شوند.
  - گراف‌ها می‌توانند از موارد کاربردی (use case) UML، ماشین های حالت متناهی، کد منبع و ... استخراج شوند.
  - عبارات منطقی می‌توانند از تصمیم‌های داخل کد منبع برنامه، گاردهای روی گذارها (شرطهای نگهبان روی انتقال بین حالتها)، شروط موجود در use case و ... استخراج شوند .

- تست مبتنی بر مدل:
  - تستها را از مدلی استخراج می‌کند که جنبه ای از سیستم تحت تست را توصیف می‌نماید.
  - معمولاً مدل بخشی از رفتار را توصیف می کند .
  - کد منبع معمولاً مدل محسوب نمی‌شود .

کلیات پوشش


چهار ساختار برای
مدل سازی نرم افزار
نحو
فضای ورودی
منطق
گراف ها
کاربرد در
ورودی
یکپارچه سازی
مدل ها
منبع
کاربرد در
فرم نرمال
فصلی (DNF)
مشخصه ها
FSMs
منبع
کاربرد در
موارد کاربری (UC)
طراحی
مشخصه ها
منبع

پوشش

با در اختیار داشتن مجموعه ای از نیازمندیهای آزمون (TR) برای معیار پوشش C : مجموعه آزمون T ، پوشش C را برآورده می کند اگر و فقط اگر برای هر نیازمندی آزمون tr در TR، حداقل یک آزمون t در T وجود داشته باشد به طوری که t، tr را برآورده کند.

- نیازمندی های آزمون ناممکن (Infeasible):
  - نیازمندیهای آزمونی هستند که نمی‌توانند برآورده شوند.
  - هیچ مقدار مورد آزمونی وجود ندارد که چنین نیازمندی های آزمون را برآورده کند.
  - کدهای مرده
  - تشخیص نیازمندیهای آزمون ناممکن برای اغلب معیارهای آزمون مسأله‌ای تصمیم ناپذیر است (یعنی نمی توان برای آن الگوریتم ثابتی نوشت که هر بار بتواند به یک جواب برسد).
- بنابراین ، عملاً پوشش ۱۰۰ درصدی، غیرممکن است.

دو راه برای استفاده از معیارهای تست

1. تولید مستقیم مقادیر تست برای برآورده کردن معیار (محققان این روش را واضح ترین روش برای استفاده از معیارها و سخت ترین روش بدون استفاده از ابزارهای خودکار می دانند.)

2. تولید مقادیر تست به صورت خارجی، و اندازه گیری با توجه به معیار (این روش بیشتر در صنعت مورد توجه و علاقه است).
   - گاهی اوقات منحرف کننده است .
   - اگر تست ها پوشش دهی ۱۰۰ درصدی را ایجاد نکنند، این چه معنایی دارد؟

معیارهای تست، گاهی سنجه (metric) نامیده می‌شوند.

مولدها و شناساگرها

- مولد (Generator): پروسیجری است که به صورت خودکار، مقادیری را برای برآورده کردن یک معیار تولید می‌کند.

- شناساگر (Recognizer): پروسیجری است که تصمیم می‌گیرد که آیا مجموعه‌ای معین از مقادیر تست، یک معیار آزمون را برآورد می‌کنند یا خیر.

- می توان ثابت کرد که هر دو مسئله برای اکثر معیارها تصمیم ناپذیر هستند.

- امکان شناسایی اینکه آیا یک **test case** می‌تواند معیاری را برآورده کند یا خیر بسیار بیشتر است از امکان تولید تست هایی که آن معیار را برآورده می کنند.

- ابزارهای تحلیل پوشش به میزان فراوانی وجود دارند.

مقایسه معیارها با استفاده از مفهوم «در بر داشتن» (subsumption)

- رابطه در برداشتن میان معیارها: یک معیار آزمون c1 ، c2 را در بر دارد، اگر و فقط اگر هر مجموعه از موارد آزمونی که معیار c1 را برآورده می کند، معیار c2 را هم برآورده کند.

- این مطلب باید برای هر مجموعه از موارد آزمون برقرار باشد.

- مثال: اگر یک مجموعه آزمون همه انشعابهای برنامه ای را پوشش داده باشد (معیار انشعاب را برآورده کرده باشد) آنگاه این مجموعه آزمون به طور تضمین شده همه عبارات را نیز پوشش داده است.

معیارهای پوشش آزمون

- تست نرم افزار به طور سنتی، هزینه بر بوده و زحمت زیادی می‌طلبد .
- معیارهای صوری (formal) پوشش برای تصمیم جهت انتخاب ورودی های تست، به کارگرفته می شوند .
- احتمال آنکه آزمونگر، مشکلاتی را پیدا نماید بسیار بالاست.
- اطمینان بیشتر درباره این که نرم افزار از کیفیت بالا و قابلیت اطمینان برخوردار باشد.
- هدف یا قاعده توقف برای تست
- معیارها، تست را کارآمدتر و اثربخش تر می کنند.

اما در عمل باید چگونه شروع به بکارگیری این ایده ها نمود ؟

بخش سوم: چگونه

حال می‌دانیم که چرا و چه چیزی ...

اما چگونه باید به آن برسیم ؟

سطوح تست مبتنی بر بلوغ فرایند تست

- سطح ۰ : تفاوتی بین تست و اشکال زدایی وجود ندارد .
- سطح ۱ : هدف تست ، نشان دادن صحت و درستی است .
- سطح ۲ : هدف تست کردن این است که نشان دهیم نرم افزار کار نمی کند .
- سطح ۳ : هدف تست کردن اثبات چیز مشخصی نیست، بلکه کاهش ریسک استفاده از نرم افزار است.
- سطح ۴ : تست کردن ، یک نظم ذهنی است که به تمامی متخصصین IT کمک می‌کند که نرم افزار با کیفیت بالاتری را توسعه دهند.

تفکر سطح صفر

- تست کردن همان اشکال زدایی است.
- تمایزی بین رفتار نادرست و اشتباه در برنامه قائل نمی شود .
- به توسعه ی نرم افزار قابل اطمینان یا ایمن کمکی نمی کند.

این چیزی است که به دانشجویان دوره ی کارشناسی کامپیوتر آموزش داده می‌شود.

تفکر سطح یک

- هدف نشان دادن درستی است .
- دستیابی به درستی غیرممکن است .
- در صورتی که هیچ خرابی رخ ندهد، چه نتیجه ای می توان گرفت ؟
  - آیا نرم افزار خوب است یا بد؟

- مهندسان تست فاقد موارد زیر هستند :
  - هدف دقیق
  - قاعده واقعی توقف تست
  - تکنیک صوری تست
- مدیران تست قدرتی ندارند.

این چیزی است که مهندسان سخت افزار اغلب انتظار دارند.

تفکر سطح دوم

- هدف، نشان دادن خرابی است .
- جستجوی خرابی، فعالیتی منفی است.
- آزمونگران و توسعه دهندگان را در رابطه ای خصمانه قرار می‌دهد.
- در صورتی که هیچ خرابی رخ ندهد، چه می‌توان گفت ؟

این موارد، توصیف کننده اغلب شرکت های نرم افزاری است.

حال چگونه می توانیم به سوی یک رویکرد تیمی پیش برویم ؟

تفکر سطح سوم

- تست کردن تنها می تواند وجود خرابی را نشان دهد.
- هنگام استفاده از نرم افزار ، متحمل ریسک هایی می شویم.
- ممکن است که ریسک کوچک باشد و نتیجه آن اهمیت چندانی نداشته باشد.
- ممکن است ریسک بزرگ بوده و نتیجه آن فاجعه آمیز باشد.
- تست کننده و توسعه دهندگان برنامه ، با هم و برای کاهش ریسک ها، کار می کنند.

این موارد، شمار اندکی از شرکت های نرم افزاری «آگاه» را توصیف می کند .

تفکر سطح چهارم

یک نظم ذهنی که کیفیت را افزایش می دهد

- تست کردن تنها یک راه برای افزایش کیفیت است .
- مهندسان تست، می توانند به راهبران فنی پروژه تبدیل شوند .
- مهمترین مسئولیت، اندازه گیری و بهبود کیفیت نرم افزار است .
- تجربه مهندسان تست باید به توسعه دهندگان برنامه کمک کند .

چگونه می توان تست را بهبود بخشید؟

- آزمونگران به ابزارهای نرم افزاری بیشتر و بهتر نیازمندند.

- آزمونگران نیاز دارند که فعالیتها و تکنیک هایی را انتخاب کنند که منجربه تست کارآتر و اثربخش تر می شود .
  - آموزش بیشتر
  - راهبردهای سازمانی مدیریتی گوناگون

- تیم های تست / تضمین کیفیت (QA) به تخصص های فنی بیشتری نیازمندند.
  - تخصص توسعه دهندگان، به میزان زیادی افزایش پیدا کرده است .

- تیم های تست / QA نیازمند تخصصی تر شدن بیشتر هستند .
  - همین گرایش در دهه ۱۹۹۰ برای توسعه اتفاق افتاد.

چهار مانع برای پذیرش عوامل مؤثر بر بهبود تست

- ۱- کمبود آموزش تست
  - بیل گیتس: «نیمی از مهندسان MS آزمونگر هستند؛ برنامه نویسان نیمی از وقت خود را صرف تست می کنند»
  - تست نرم افزار در دوره های کارشناسی و کارشناسی ارشد علوم کامپیوتر در امریکا الزامی نیست (تعداد کل کلاسهای تست در این دوره ها حدود ۲۵ کلاس است).
- ۲- لزوم تغییر فرایند
  - پذیرش بسیاری از تکنیکها و ابزارهای تست نیازمند تغییراتی در فرایند توسعه است.
  - این کار برای بسیاری از شرکتهای نرم افزاری بسیار گران است.
- ۳-قابلیت استفاده ابزارها
  - استفاده از بسیاری از ابزارها نیازمند آن است که کاربر از تئوری مربوطه آگاه باشد
  - اما نباید اینطور باشد (یک راننده نباید برای راندن یک وسیله نقلیه نیازی به آگاهی از نحوه احتراق داخلی آن داشته باشد؛ برای استفاده از یک کامپایلر نیازی به دانستن نحوه کار پارسر و تولید خودکار کد نداریم)
- ۴- ابزارهای ضعیف و غیراثربخش
  - اکثر ابزارهای تست کار زیادی انجام نمی دهند – ولی اغلب کاربران نمی دانند که این ابزار می تواند بهتر از این باشد
  - ابزارهای کمی هستند که می توانند مساله فنی کلیدی (یعنی تولید خودکار مقادیر تست) را حل کنند.

خلاصه

- تست بیشتر موجب صرفه جویی منابع مالی می شود.
  - برنامه ریزی برای تست، موجب صرفه جویی مقدار زیادی از منابع مالی می شود.

- تست کردن، دیگر شکل یک هنر را ندارد.
  - مهندسان یک جعبه ابزار از معیارهای تست دارند.

- هنگامی که آزمونگران، مهندس باشند، کیفیت محصول بهتر خواهد شد.
  - توسعه دهندگان نیز بهتر خواهند شد.

پرسشهای باز

- کدامیک از معیارها برای نرم افزارهای تعبیه شده، و قابل اطمینان مناسبتر است؟
  - از کدام ساختار نرم افزاری استفاده کنیم ؟
- چگونه می توانیم به بهترین شکل تست کردن را با ابزارهای قوی خودکارسازی کنیم؟
  - استخراج ساختار نرم افزار
  - ساخت نیازمندی های تست
  - ایجاد مقادیری از نیازمندی های تست
  - ایجاد اسکریپتهای کامل تست
  - یافتن راه حل برای مسئله «نگاشت»
- اعتبارسنجی تجربی
- انتقال تکنولوژی
- کاربردی کردن تست برای دامنه های جدید

خلاصه مفاهیم جدید این جلسه

- چرا تست می کنیم ؟ برای کاهش ریسک استفاده از نرم افزار
- چهار نوع از فعالیت های تست: طراحی تست، خودکار سازی، اجرا، و ارزیابی
- اصطلاحات نقص، خرابی، مدل RIP، مشاهده پذیری و کنترل پذیری
- چهار ساختار – نیازمندی های تست و معیارها
- سطوح بلوغ فرایند تست – سطح چهارم یک نظم ذهنی است که کیفیت نرم افزار را بهبود می بخشد .

تست بهتر و زودهنگام تر می تواند مدیر تست را تواناتر نماید.

پایان