Introduction to Software Testing
(*2nd edition*)
Chapter 4

Putting Testing First

Paul Ammann & Jeff Offutt

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

August 2014

Testing has evolved from afterthought to a central activity in certain development methods

If high-quality testing is not centrally and deeply embedded in your development process,

your project is at high risk for failure.

تست کردن از بعد فکری به یک فعالیت مرکزی در روشهای توسعه خاص تبدیل شده است
اگر تست با کیفیت بالا به طور متمرکز و عمیق در فرآیند توسعه شما تعبیه نشده باشد، پروژه شما در معرض خطر بالایی برای شکست قرار دارد.

The Increased Emphasis on Testing

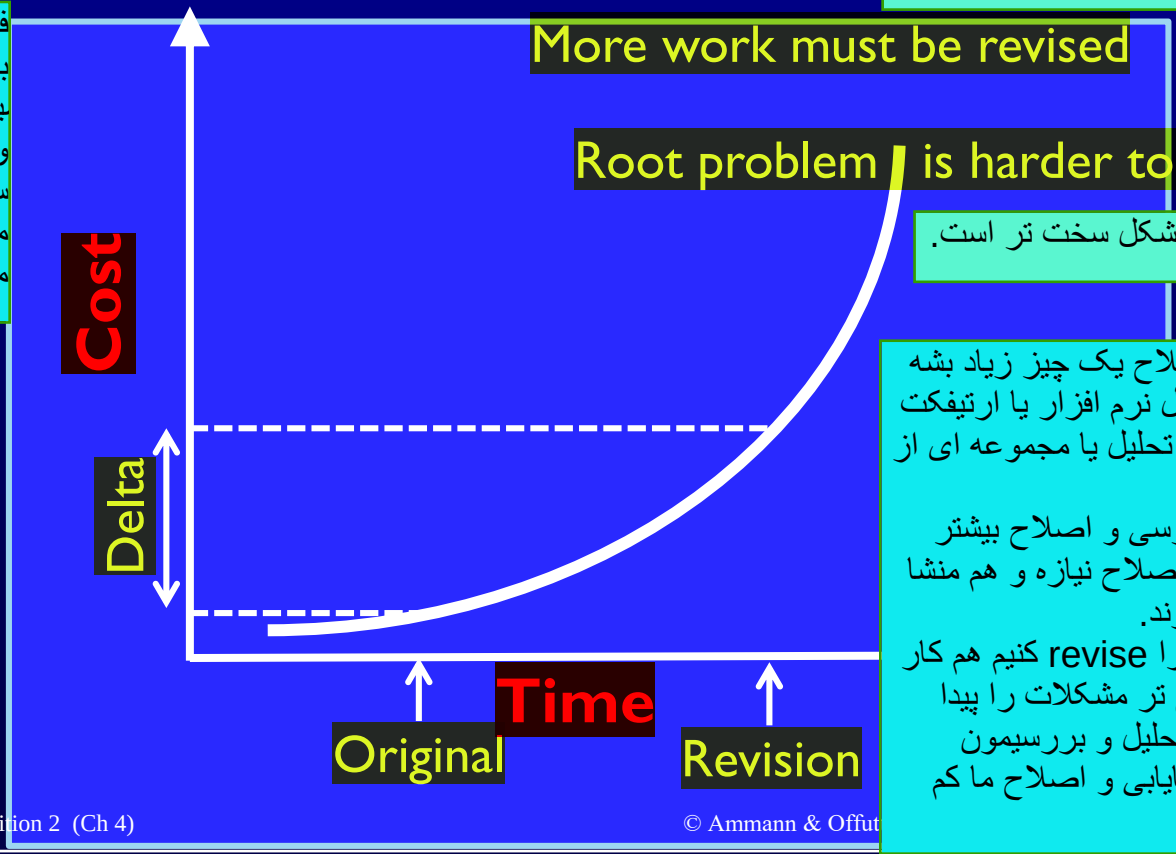
- Philosophy of traditional software development methods
  - Upfront analysis
  - Extensive modeling
  - Reveal problems as early as possible

فلسفه روش های سنتی توسعه نرم افزار
- تحلیل اولیه
- مدل سازی گسترده
- مشکلات را در اسرع وقت آشکار کنید

اما این روند در روش های سنتی نبوده چون ونها reviseکردن را اخر کار انجام میدادن س ماکس زمان بین تولید و revision بوده بین هرکدوم از ارتیفکت های نرم افزاری و اون زمانی که محصول تحویل داده میشه

کارهای بیشتری باید اصلاح شود

فلسفه ی پشت این کارشون این بوده که تحلیل ها مفصل انجام بیشه و مدلینگ ها سنگین هستند وسعی میکردن با تحلیل و مدل سازی های اول کار، در حد ممکن و هرچه سریع تر مشکلات را شناسایی کنند.



پیدا کردن ریشه مشکل سخت تر است.

هرچه زمان بین تولید و اصلاح یک چیز زیاد بشه یعنی دلتا زیاد بشه => ماژول نرم افزار یا ارتیفکت میتواند مدل یا سورس کد یا تحلیل یا مجموعه ای از تصمیم گیری ها باشه
هرچه زمان بین تولید و بررسی و اصلاح بیشتر باشه هم کار بیشتری برای اصلاح نیازه و هم منشا مشکلات سخت تر پیدا میشوند.
هرچه زودتر یک ارتیفکتی را revise کنیم هم کار کمتری انجام میدیم هم سریع تر مشکلات را پیدا میکنیم و هم اینکه اسکوپ تحلیل و بررسیمون کوچکتر است. درنتیجه خطایابی و اصلاح ما کم هزینه تر میشه

The primary reason for the ever-increasing cost

دلیل اصلی افزایش روزافزون هزینه

- Additional work is invested that depends on the original decision, and this work must also be revised if the original decision is revised.
- A secondary problem is that as the software grows it gets harder to find the root cause of failures.

کار اضافی سرمایه گذاری می شود که بستگی به تصمیم اصلی دارد و در صورت تجدید نظر در تصمیم اولیه باید این کار نیز تجدید نظر شود.
یک مشکل ثانویه این است که با رشد نرم افزار، یافتن علت اصلی خرابی ها سخت تر می شود.

شتریان را از تغییر کردن نظراتشون نمیتوانید منع کنید پس همواره ممکن است اون نیازمندی ها تغییر کنند پس اینکه بیاییم تغییر را حذف کنیم درست نیست چون شرایط طوری است که همواره امکان تغییر هست.

مدل سازی و تحلیل را خیلی سنگین انجام میدادن

Traditional Assumptions

1. مدل سازی و تجزیه و تحلیل می تواند مشکلات بالقوه را در مراحل اولیه توسعه شناسایی کند.

1. Modeling and analysis can identify potential problems early in development

2. صرفه جویی در منحنی هزینه تغییر، هزینه مدل سازی و تجزیه و تحلیل را در طول عمر پروژه توجیه می کند

2. Savings implied by the cost-of-change curve justify the cost of modeling and analysis over the life of the project

- These are true if requirements are always complete and current.

داریم هزینه ی تغییررا مینیمم میکنیم یعنی نمیپذیریم که تغییری رخ بده

به جای پذیرفتن تغییر در طول فرایند، داریم روی مدل سازی سنگین وقت میگذاریم این فرضیات درست هستند به شرطی که نیازمندی ها واضح و کامل و شفاف باشند و ابهامی نداشته باشند

- But those annoying customers keep changing their minds!

ادم ها در تقریب زدن کارها خوب هستند ولی در دقیق بیان کردن نه

ادم ها بهتر میتوانند اول کار کلیات را بگن بعد هرچه کار جلو میره وارد جزئیات هم میشن

  - Humans are naturally good at approximating
  - But pretty bad at perfecting

اگر الزامات همیشه کامل و جاری باشند، اینها درست هستند. اما آن مشتریان مزاحم مدام نظر خود را تغییر می دهند!- انسان ها به طور طبیعی در تقریب خوب هستند- اما در کمال کردن بسیار بد است

- These two assumptions have made software engineering frustrating and difficult for decades

این دو فرض، مهندسی نرم افزار را برای چندین دهه خسته کننده و دشوار کرده است

Thus, agile methods …

Key end result of Agile methods

- Working software
- Responsiveness to change
- Effective development teams
- Happy customers.

تمرکز متدهای اجایل

نتیجه نهایی کلیدی روش های چابک
نرم افزار کاری
پاسخگویی به تغییر
تیم های توسعه موثر
مشتریان خوشحال

در تعامل با مشتری هستند و مشتری ها را میتوانند جزو تیم خودشون بیارن

چرا چابک باشیم؟ روش های چابک با تشخیص اینکه هیچ یک از این فرض ها برای بسیاری از پروژه های نرم افزاری فعلی معتبر نیستند شروع می شوند.

Why Be Agile ?

- مهندسان نرم افزار در توسعه نیازمندی ها خوب نیستند
- ما تغییرات زیادی را پیش بینی نمی کنیم.

- Agile methods start by recognizing that neither assumption is valid for many current software projects
  - Software engineers are not good at developing requirements
  - We do not anticipate many changes
  - Many of the changes we do anticipate are not needed

بسیاری از تغییراتی که ما پیشبینی میکنیم لازم نیست.

- Requirements (and other "non-executable artifacts") tend to go out of date very quickly
  - We seldom take time to update them
  - Many current software projects change continuously

نیازمندیها و سایر مصنوعات غیرقابل اجرا خیلی زود قدیمی میشوند.
- ما به ندرت برای به روز رسانی آنها وقت می گذاریم

- Agile methods expect software to start small and evolve over time
  - Embraces software evolution instead of fighting it

بسیاری از پروژه های نرم افزاری فعلی به طور مداوم تغییر می کنند

تکامل نرم افزار را به جای مبارزه با آن در آغوش می گیرد

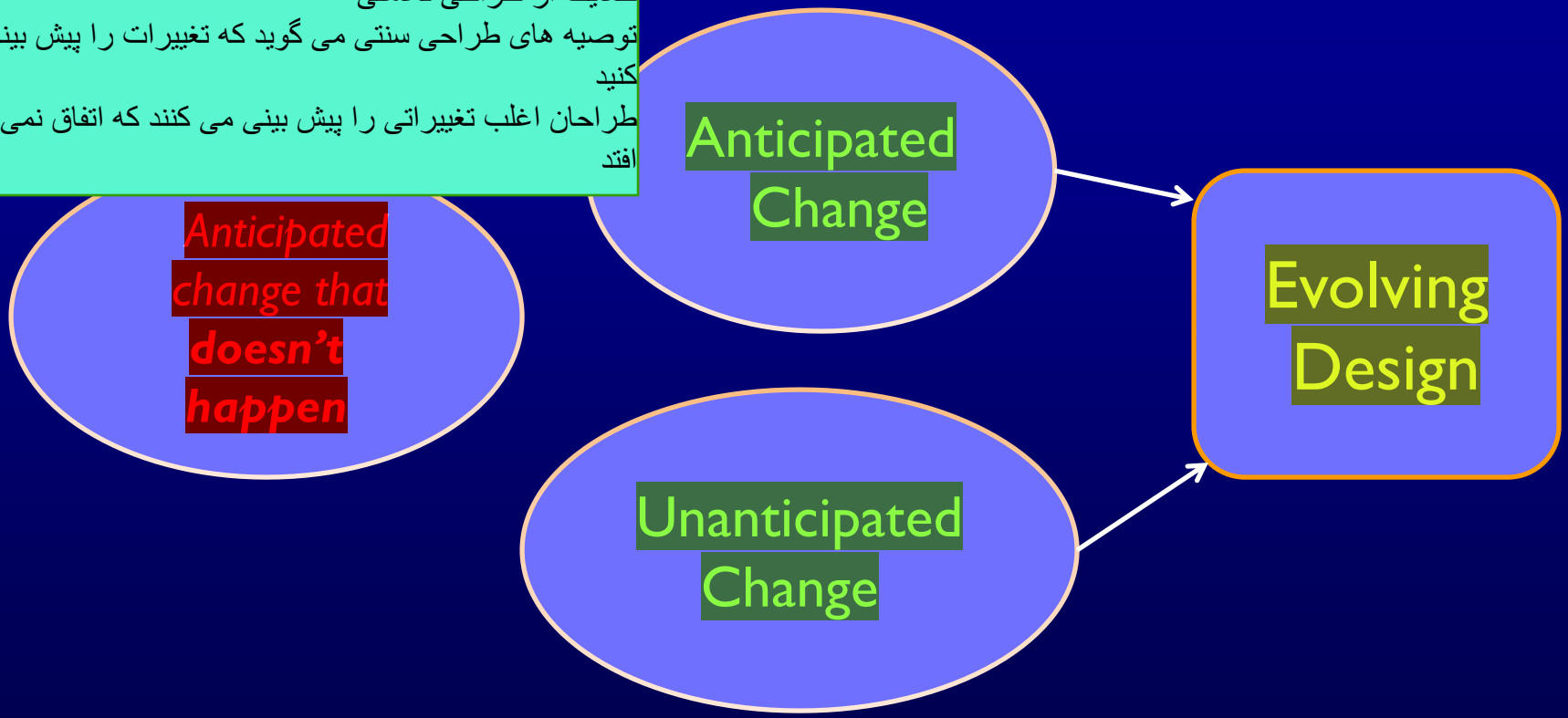
روشهای چابک انتظار دارند که نرم افزار کوچک شروع شود و در طول زمان تکامل یابد

Supporting Evolutionary Design

Traditional design advice says to anticipate changes

Designers often anticipate changes that don't happen

حمایت از طراحی تکاملی
توصیه های طراحی سنتی می گوید که تغییرات را پیش بینی کنید
طراحان اغلب تغییراتی را پیش بینی می کنند که اتفاق نمی افتد



Both anticipated and unanticipated changes affect design

تغییرات پیش بینی شده و پیش بینی نشده هر دو بر طراحی تاثیر می گذارند

Agile methods

- An agile principle that goes directly to the heart of the both assumptions is "You ain't gonna need it!", or **YAGNI**.

روش های چابک
یک اصل چابک که مستقیماً به قلب هر دو فرض می رود این است که »به آن نیاز نخواهی داشت! یا YAGNI.

- The YAGNI principle states that traditional planning is fraught precisely because predicting system evolution is fundamentally hard, and hence expected savings from the cost-of-change curve do not materialize.

تحقق یابد

- Instead, agile methods defer many design and analysis decisions and focus on creating a running system that does "something" as early as possible.

در عوض، روشهای چابک بسیاری از تصمیمگیری های طراحی و تحلیل را به تعویق میاندازند
بر ایجاد یک سیستم در حال اجرا متمرکز میشوند که کاری را در اسرع وقت انجام دهد.

اصل YAGNI بیان میکند که برنامه ریزی سنتی دقیقاً به این دلیل که پیشبینی تکامل سیستم اساساً سخت است، پرمشغله است و از این رو صرفه جوییهای مورد انتظار از منحنی هزینه تغییر محقق نمیشود.

The Test Harness as Guardian (4.2)

مهار تست به عنوان نگهبان

What is Correctness ?

Traditional Correctness (Universal)

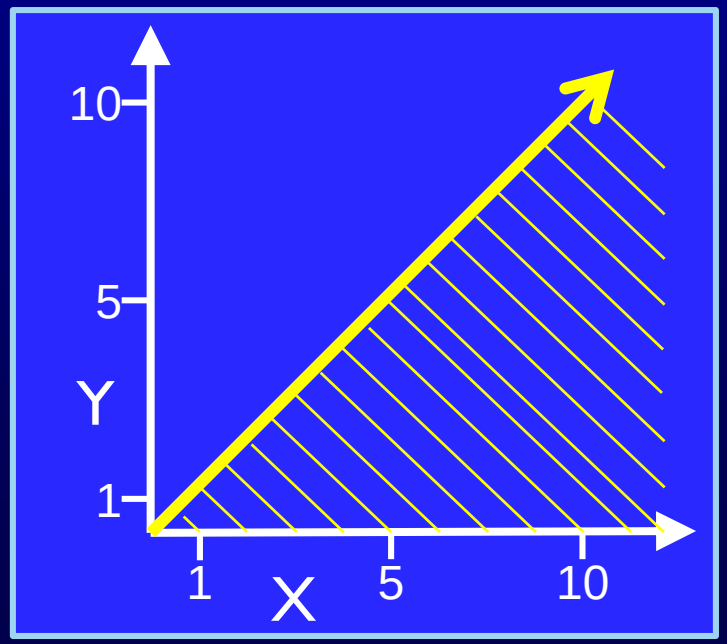
مفهوم جهانی

$\forall x,y, x \geq y$

درستی یا correctness یه مفهوم غیرقابل دستیابی است

Agile Correctness (Existential)

تعریف وجودی



{ (1, 1) → T
(1, 0) → T
(0, 1) → F
(10, 5) → T
(10, 12) → F }

Agile methods in general, and test-driven development in particular,

take a novel, and somewhat more restricted, view of correctness.

روش های چابک به طور کلی، و توسعه مبتنی بر تست طور خاص دیدگاهی بدیع و تا حدودی محدودتر از correctness دارند.

به جای درنظر گرفتن کل ورودی های ممکن یه subsetای را درنظرمیگیره بعد میگه اگه روی این زیرمجموعه داره درست جواب میده برای من کافیه.

دیدگاه محدودی از صحت

A Limited View of Correctness

- In traditional methods, we try to define all correct behavior completely, at the beginning
  - What is correctness?
  - Does "correctness" mean anything in large engineering products?
  - People are VERY BAD at completely defining correctness

در روشهای سنتی سعی میکنیم در ابتدا همه رفتارهای صحیح را به طور کامل تعریف کنیم
- صحت چیست؟

- مردم در تعریف کامل درستی بسیار بد هستند

- In agile methods, we redefine correctness to be relative to a specific set of tests
  - If the software behaves correctly on the tests, it is "correct"
  - Instead of defining all behaviors, we demonstrate some behaviors
  - Mathematicians may be disappointed at the lack of completeness

در روشهای چابک، ما صحت را نسبت به مجموعهای از آزمونها دوباره تعریف میکنیم

- به جای تعریف همه رفتارها، برخی رفتارها را نشان می دهیم

- اگر نرم افزار در تست ها به درستی رفتار کند، "درست" است.

But software engineers ain't mathematicians!

- ممکن است ریاضیدانان از عدم کامل بودن ناامید شوند
اما مهندسان نرم افزار ریاضیدان نیستند!

به صورت نسبی تعریف میکنیم.

Test Harnesses Verify Correctness

control and make use of (natural resources)

A *test harness* runs all automated tests efficiently and reports results to the developers

یک مهار تست تمام تست های خودکار را به طور موثر اجرا می کند و نتایج را به توسعه دهندگان گزارش می دهد

- Tests must be automated
  - Test automation is a prerequisite to test driven development
- Every test must include a test oracle that can evaluate whether that test executed correctly
- The tests replace the requirements
- Tests must be high quality and must run quickly
- We run tests every time we make a change to the software

آزمون ها باید خودکار باشند
-اتوماسیون تست یک پیش نیاز برای توسعه آزمایش محور است

یه ماژول نرم افزاری است

هر آزمون باید شامل یک اوراکل آزمایشی باشد که بتواند ارزیابی کند که آیا آن تست به درستی اجرا شده است یا خیر.
آزمون ها جایگزین الزامات می شوند

ما وقتی میگیم برنامه ای میخایم که یک تست را پاس کنه یعنی داریم میگیم فلان قابلیت را میخایم داشته باشه نرم افزارمون

تست ها باید کیفیت بالایی داشته باشند و باید سریع اجرا شوند.
هر بار که تغییری در نرم افزار ایجاد می کنیم، تست هایی را اجرا می کنیم

Test driven development
مفهوم محوریت داشتن تست
ست یه مفهوم مرکزی و مهم در فرایند توسعه
نرم افزار میشه

TDD

- From the developer's perspective, testing is the central activity in development.

از دیدگاه توسعهدهنده، تست فعالیت اصلی در توسعه است.

- Good design still matters in TDD, it simply occupies a different, and niche in the development cycle.

طراحی خوب هنوز در TDD اهمیت دارد، به سادگی در چرخه توسعه جایگاه متفاوت و جایگاهی را اشغال می کند.

Consequence of the test-harness-

فلسفه اعتقادی است که در میان بسیاری از توسعه دهندگان چابک مشترک است.

اسناد غیرقابل اجرا به طور بالقوه گمراه کننده هستند.

- A philosophy is a belief shared by many agile developers.
  - Non-executable documents are potentially misleading.
  - While everyone agrees that a non-executable document that *correctly* describes a software artifact is helpful, it is also true that a non-executable document that *incorrectly* describes a software artifact is a liability.

مسئولیت
obstacle
دردسر و مساله

در حالی که همه موافق هستند که یک سند غیرقابل اجرا که به درستی یک مصنوع نرم افزاری را توصیف می کند مفید است، این نیز درست است که یک سند غیرقابل اجرا که به اشتباه یک مصنوع نرم افزار را توصیف می کند یک تعهد و مسئولیت است.

یعنی یه working software ارائه بدیم که هم قابل ارزیابی است
هم رفتار نرم افزار را نشان میده که قابل قبول است یا نه
با رویکردهای TDD قابل ارزیابی است ومیتوانیم بفهمیم که نیاز ها
را میتواند ارضا کنه یا نه؟

Agile methods
attempt to make executable artifacts to satisfy needs,
in traditional software engineering,
were satisfied by non-executable artifacts.

روش های چابک تلاش می کنند تا مصنوعات اجرایی برای برآوردن نیازها بسازند.
در مهندسی نرم افزار سنتی، با مصنوعات غیرقابل اجرا ارضا شدند.

Definition of success

- Traditional development defines success as "On time and on budget,"

تعریف موفقیت توسعه سنتی موفقیت را اینگونه تعریف می کند: به موقع و با بودجه،
ولی اینکه خروجی کارچقدر از دید مشتری مناسب و خوب بوده قابل بحث نبوده

- Agile methods aim first for having ***something*** **executable** available from the very beginning of development and second producing a different, and presumably better, product than the one originally envisioned.

هدف روشهای چابک اولاً داشتن یک چیز قابل اجرا در دسترس از همان ابتدای توسعه و
دوم تولید محصولی متفاوت و احتمالاً بهتر از آنچه در ابتدا تصور میشد.

How to make Agile work

- Test cases need to be of high quality.
- Test processes need to be efficient.
- Use of test automation is necessary, but not sufficient.

چون تست ها دارن جای requirements را میگیرن.

چگونه Agile را عملی کنیم
موارد تست باید از کیفیت بالایی برخوردار باشند.
فرآیندهای تست باید کارآمد باشند.
استفاده از اتوماسیون تست ضروری است، اما کافی نیست.

باید بتوانیم با حداقل تست ها بیشترین Failure ها را دربیاریم.

یکپارچه سازی مداوم

Continuous Integration

- Agile methods work best when the current version of the software can be run against all tests at any time

روشهای چابک زمانی بهترین کار را دارند که نسخه فعلی نرمافزار را بتوان در هر زمان در مقابل همه آزمایشها اجرا کرد

A *continuous integration server* rebuilds the system, returns, and reverifies tests whenever *any* update is checked into the repository

یک سرور یکپارچه سازی پیوسته سیستم را بازسازی می کند، هر زمان که هر به روز رسانی در مخزن بررسی شود، آزمایش ها را برمی گرداند و دوباره تأیید می کند.

- Mistakes are caught earlier
- Other developers are aware of changes early

اشتباهات زودتر تشخیص داده می شوند.

- The rebuild and reverify must happen as soon as possible
  - is an important part of the test harness.

سایر توسعه دهندگان زودتر از تغییرات آگاه هستند

- بخش مهمی از مهار تست است.

بازسازی و تایید مجدد باید در اسرع وقت اتفاق بیفتد.

A *continuous integration server* doesn't just run tests, it decides if a modified system is still correct

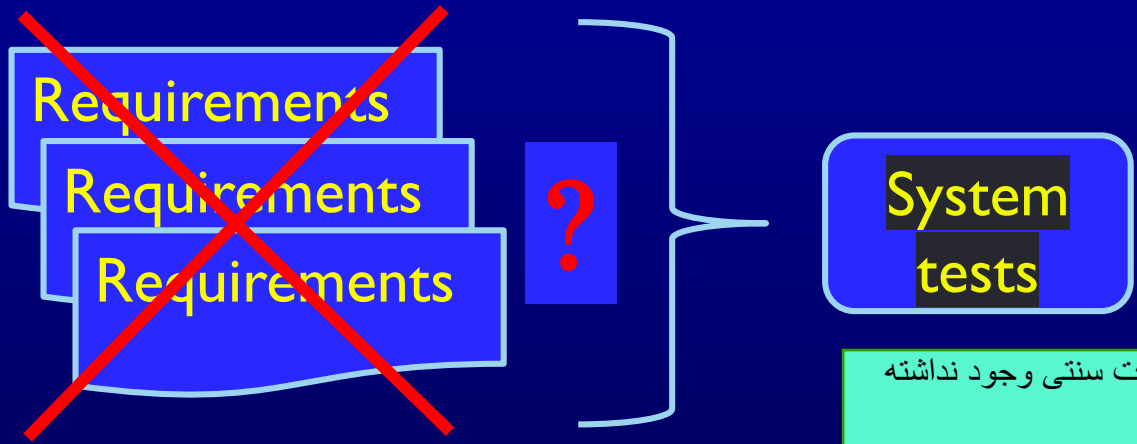
اصلاح خطاها در اسرع وقت انجام میشه

سرور یکپارچه سازی پیوسته فقط آزمایش ها را اجرا نمی کند، بلکه تصمیم می گیرد که آیا سیستم اصلاح شده هنوز درست است یا خیر

System Tests in Agile Methods(I)

Traditional testers often design system tests from requirements

تسترهای سنتی اغلب تست های سیستم را بر اساس الزامات طراحی می کنند



اما ... اگر اسناد الزامات سنتی وجود نداشته باشد چه؟

But … what if there are no traditional requirements documents ?

User Stories

جای requirementsها را میگیره.

A *user story* is a few sentences that captures what a user will do with the software

یوزر استوری میگه انتظار کاربر از سیستم چیه؟

یک داستان کاربر چند جمله است که نشان می دهد کاربر با نرم افزار چه خواهد کرد

Withdraw money from checking account

Agent sees a list of today's interview applicants

Support technician sees customer's history on demand

- به زبان کاربر نهایی
- معمولاً در مقیاس کوچک با جزئیات کم
- بایگانی نشده است

- In the language of the end user
- Usually small in scale with few details
- Not archived

System Tests in Agile Methods(II)

- Amount of effort required to implement the functionality captured by the system test might be quite large.

مقدار تلاش مورد نیاز برای اجرای عملکرد ثبت شده توسط تست سیستم ممکن است بسیار زیاد باشد.

- How do you run a test against a system that not only isn't yet built, but cannot even be built in a short time-frame?

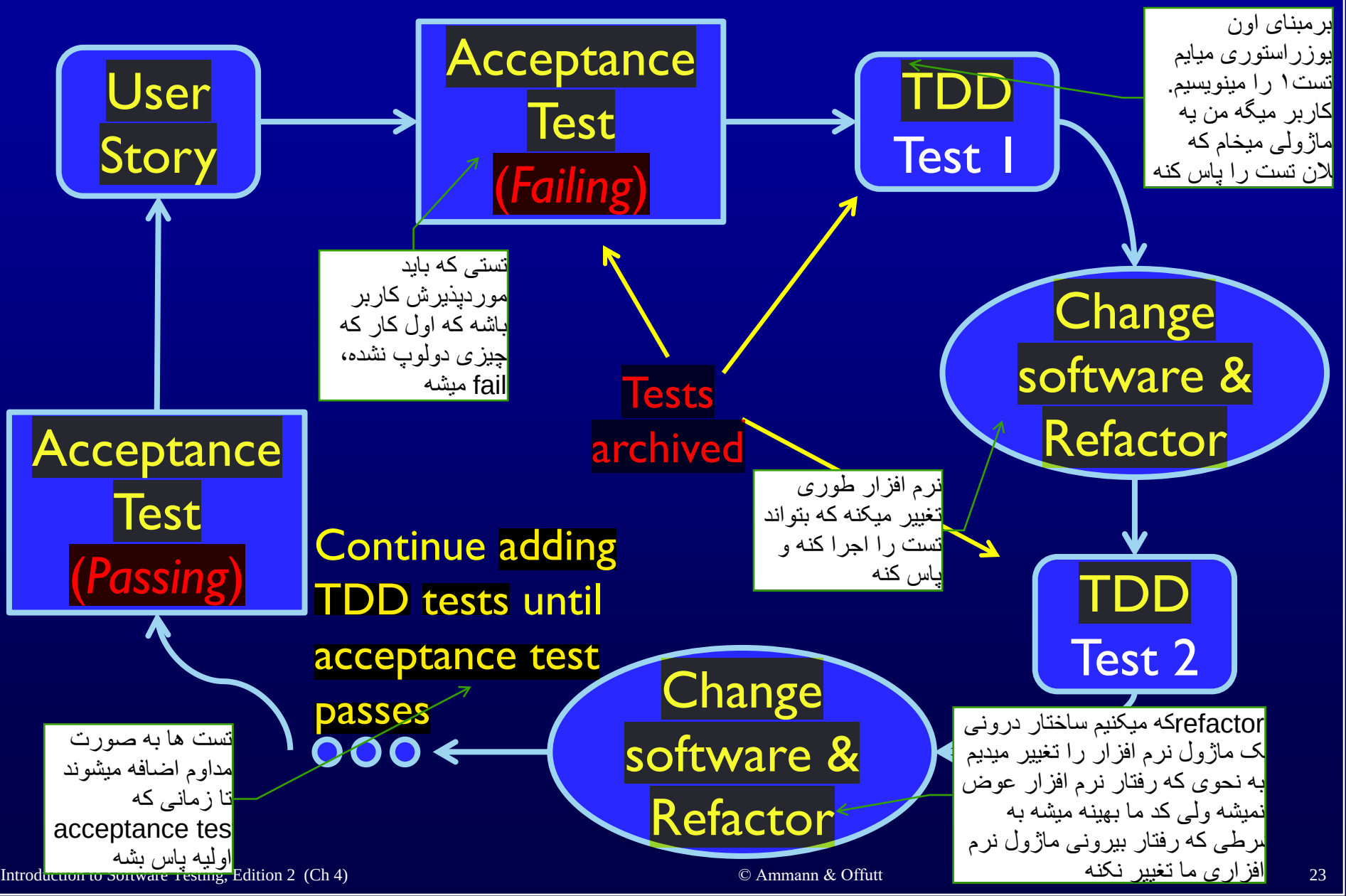
چگونه می توان آزمایشی را بر روی سیستمی اجرا کرد که نه تنها هنوز ساخته نشده است، بلکه حتی نمی تواند در یک بازه زمانی کوتاه ساخته شود؟

Agile methods place a premium on having a test harness continuously verify the system.

در اجایل به جای اینکه بگذاریم سیستم کامل دولوپ بشه و بعد تست را انجام بدیم، میایم گام به گام تست را انجام میدیم
چون هرچه که سیستم بزرگتر بشه، تست کردنش هم سخت تر و هزینه برتر میشه.
پس به صورت مداوم و continuously سیستم را Verify میکنیم.

تست های پذیرش

Acceptance Tests in Agile Methods



An Example

سناریو نرمال یک یوزراستوری یعنی کاری که باید انجام بشه به صورت طبیعی

- User story: "Support technician sees customer's history on demand."

داستان کاربر: "تکنسین پشتیبانی تاریخچه مشتری را در صورت تقاضا می بیند."

- The story does not include implementation details, and is not specific enough to run as a test case.

داستان شامل جزئیات پیاده سازی نمی شود و به اندازه کافی مشخص نیست که به عنوان یک مورد آزمایشی اجرا شود.

- This user story might have a happy path test where a technician fields a call from a specific, existing user. The test passes if that specific user's history is displayed on demand.

این یوزر استوری ممکن است یک تست مسیر خوشحال کننده داشته باشد که در آن یک تکنسین یک تماس از یک کاربر خاص و موجود را ارسال می کند.

در صورتی که تاریخچه آن کاربر خاص در صورت تقاضا نمایش داده شود، آزمایش انجام می شود و تست pass میشه.

- A different test might involve a new user; this test passes if the technician is informed that the user does not have a history.

یک آزمایش متفاوت ممکن است شامل یک کاربر جدید باشد. در صورتی که به تکنسین اطلاع داده شود که کاربر سابقه ندارد، این تست انجام می شود.

- These tests provide specific, concrete guidance to developers as to exactly what functionality needs to be implemented.

این تستها راهنمایی هایی مشخصی را برای توسعه دهندگان ارائه میکنند که دقیقاً چه عملکردی باید پیاده سازی شود.

تستهای با کیفیت بالا برای موفقیت پروژههای چابک اهمیت زیادی دارند.

High-quality tests are of central importance to whether agile projects succeed.

موفقیت پروژه ی اجایل به کیفیت تست ها بستگی داره چون انتظارات یا همان requirenmentها در قالب تست ها تعریف میشوند.
پس هرچه کیفیت تست ها بره بالا پروژه موفق تر میشه.

اضافه کردن تست ها به سیستم های موجود

Adding Tests to Existing Systems

- Most of today's software is legacy
  - No legacy tests
  - Legacy requirements hopelessly outdated
  - Designs, if they were ever written down, lost
- Companies sometimes choose not to change software out of fear of failure

بیشتر نرم افزارهای امروزی قدیمی هستند
- بدون تست میراث
- الزامات میراث به طرز ناامیدکننده ای منسوخ شده است
- طرح ها، اگر زمانی نوشته شده باشند، گم شده اند

شرکت ها گاهی اوقات از ترس شکست نرم افزار را تغییر نمی دهند

How to apply TDD to legacy software with no tests?

چگونه می توان TDD را در نرم افزارهای قدیمی بدون تست اعمال کرد؟

- Create an entire new test set? — too expensive!
- Give up? — a mixed project is unmanageable

نه میشه پروژه جدیدی تعریف کرد و نه میشه کل سیستم را تست کنیم و نه میشه از اون پروژه legacy منصرف شد چون ممکنه یه پروژه ای داشته باشیم که داره از کدهای legacy استفاده میکنه.

یک مجموعه آزمایشی جدید ایجاد کنید؟
- بیش از حد گران!
تسلیم شدن؟
- یک پروژه مختلط غیر قابل مدیریت است

Incremental TDD

Needs to incrementally introduce test cases,
so that over time a system safely moves towards
both new functionality
and new test cases that verify that functionality.

TDD افزایشی
نیاز به معرفی تدریجی موارد آزمایشی یا همان تست کیس ها را دارد، به طوری که در طول زمان یک سیستم با خیال راحت به سمت عملکرد جدید و تست کیس های جدید که آن عملکرد را تأیید می کند حرکت کند.

تست کیس ها را مرحله به مرحله و کوچیک کوچیک تولید میکنیم.
و همین طور که جلو میریم و فانکشنالیتی های جدید به سیستم اضافه میکنیم،برای اونها تست کیس های جدید تعریف میکنیم

پس درحین بهبود سیستم براش تست مینویسیم.

Incremental TDD

- When a change is made, add TDD tests for just that change

وقتی تغییری ایجاد شد، تست های TDD را فقط برای آن تغییر اضافه کنید

  - Refactor: is a way to modify (hopefully improving) the structure of existing code without changing its behavior.

Refactor: راهی برای اصلاح (امیدوارانه بهبود) ساختار کد موجود بدون تغییر رفتار آن است.

- As the project proceeds, the collection of TDD tests continues to grow

با ادامه پروژه، مجموعه تست های TDD به رشد خود ادامه می دهد

- Eventually the software will have strong TDD tests

در نهایت یه مجموعه تست خوبی داریم

The Testing Shortfall

کمبود تست

- Do TDD tests (acceptance or otherwise) test the software well?

آیا تست های TDD (قبولی یا غیر آن) نرم افزار را به خوبی تست می کنند؟

  - Do the tests achieve good coverage on the code?
  - Do the tests find most of the faults?
  - If the software passes, should management feel confident the software is reliable?

ایا میتوانیم بگیم این تست های TDD از همه جنبه ای کدهامون را بررسی کردن؟ ایا میتوانیم بگیم بیشترین فالت ها را برامون در اورده؟ اگه نرم افزارمون با اون تست هایی که جایگزین نیازمندی ها بودن و برمبنای اونها ما پیاده سازی را انجام دادیم، پاس شد ایا میتوانیم بگیم نرم افزارمون reliable است؟
نههه

NO!

- آیا تست ها پوشش خوبی روی کد دارند؟
- آیا تست ها بیشتر ایرادات را پیدا می کنند؟
- اگر نرم افزار قبول شود یعنی تست هاش پاس شود، آیا مدیریت باید از قابل اعتماد بودن نرم افزار اطمینان داشته باشد؟
نههههه!!

Why Not?

- Most agile tests focus on "*happy paths*"
  - What should happen under normal use
- They often miss things like
  - Confused-user paths
  - Creative-user paths
  - Malicious-user paths

چرا که نه؟
اکثر تست های چابک بر روی »مسیرهای شاد« تمرکز دارند.
- در استفاده معمولی چه اتفاقی باید بیفتد

مسیرهای نامشخص

آنها اغلب چیزهایی مانند
- مسیرهای کاربر سردرگم
- مسیرهای کاربر خلاق
- مسیرهای کاربر مخرب
را از دست میدهند.

سناریو اصلی و موفقیت امیز نرم افزار

سناریوهای شکست هم طبق انتظارات مشتری میریم جلو

مسیرهایی که ما قبلا پیشبینی نکردیم

The agile methods literature does not give much guidance

ادبیات روشهای چابک راهنمایی زیادی نمیکند

ایا پس بررسی happy path ها کافی است یا نه؟

Design Good Tests

1. Use a human-based approach

- Create additional user stories that describe non-happy paths
- How do you know when you're finished?
- Some people are very good at this, some are bad, and it's hard to teach

از رویکرد انسان محور استفاده کنید
- داستان های کاربر اضافی ایجاد کنید که مسیرهای غیر شاد را توصیف می کند
- چگونه متوجه می شوید که کارتان تمام شده است؟
- برخی از مردم در این کار بسیار خوب هستند، برخی بد هستند، و آموزش دادن آن دشوار است

از مدل سازی و معیارها استفاده کنید
- دامنه ورودی را برای طراحی تست ها مدل کنید
رفتار نرم افزار را با نمودارها، منطق یا گرامرها مدل کنید
- حس کامل بودن
- آموزش بسیار ساده تر-مهندسی
- به دانش ریاضی گسسته نیاز دارد

رویکرد تجربی است و اموزشی نیست.

Part 2 of book …

2. Use modeling and criteria

- Model the input domain to design tests
- Model software behavior with graphs, logic, or grammars
- A built-in sense of completion
- Much easier to teach—engineering
- Requires discrete math knowledge

شرط توقف معنا پیدا میکنه یعنی معلومه که کامل تست کردن به چه معناست.

مدل سازی انجام میدیم.

Summary

- More companies are putting testing first
- This can dramatically decrease cost and increase quality
- A different view of "*correctness*"
  - Restricted but practical
- Embraces evolutionary design
- TDD is definitely not test automation
  - Test automation is a prerequisite to TDD
- Agile tests aren't enough

در اجایل، نوع نگاه به درستی یا correctness را عوض میکنیم.

طراحی به صورت تکاملی جلو میره.

چون میتوانند happy pathها و برخی از سناریوهای شکست راتعریف کنند و نمیتوانیم مطمئن شیم که نرم افزار ما به طور کامل reliable است.

شرکت های بیشتری آزمایش را در اولویت قرار می دهند
این می تواند به طور چشمگیری هزینه را کاهش دهد و کیفیت را افزایش دهد
دیدگاه متفاوت از صحت
- محدود اما کاربردی
از طراحی تکاملی استقبال می کند
TDD قطعا اتوماسیون تست نیست
- اتوماسیون تست پیش نیاز TDD است
تست های چابک کافی نیستند