

دانشگاه آزاد اسلامی واحد تبریز

نام درس: آزمون نرم افزار

بخش: آزمون نرم افزار

نام استاد: دکتر مسعود کارگر



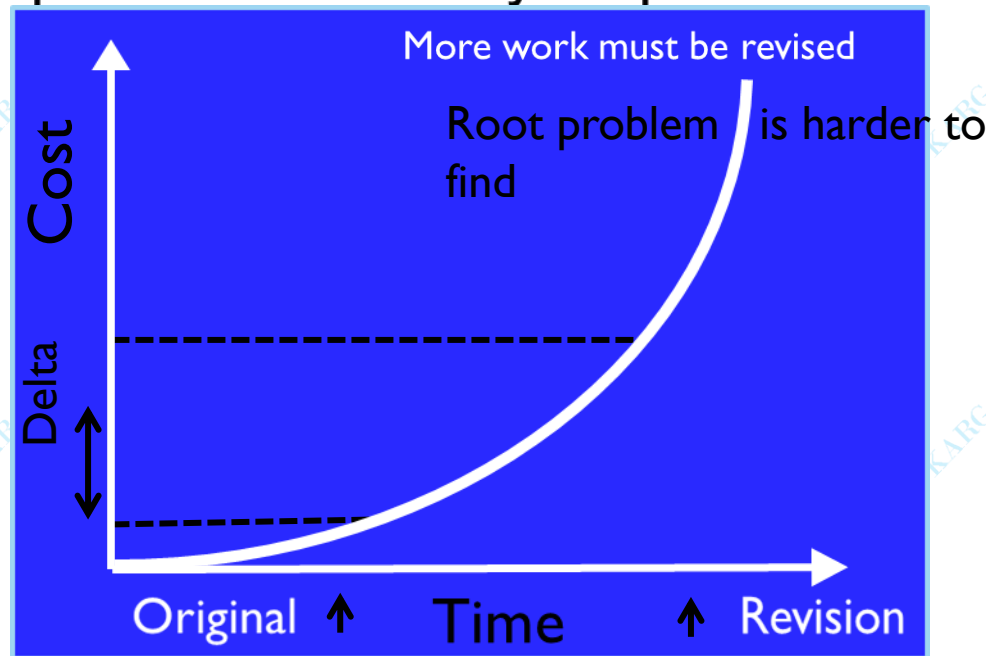
The Increased Emphasis on Testing

- Philosophy of traditional software development methods

- _Upfront analysis

- _Extensive modeling

- _Reveal problems as early as possible



Traditional Assumptions

1. Modeling and analysis can identify potential problems early in development
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
 - But those annoying customers keep changing their minds!
 - _ Humans are naturally good at approximating
 - _ But pretty bad at perfecting
 - These two assumptions have made software engineering frustrating and difficult for decades

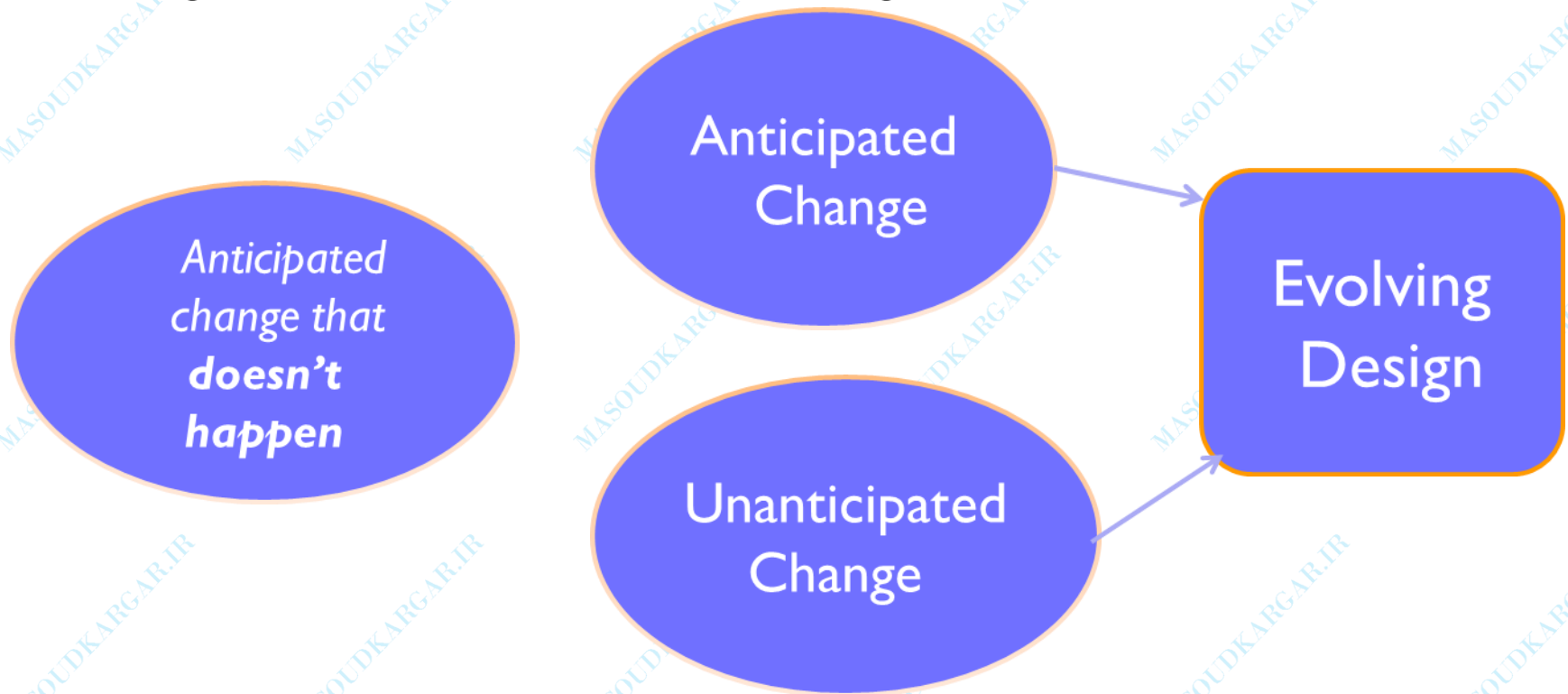
Thus, agile methods ...

Why Be Agile ?

- 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**
- Requirements (and other “non-executable artifacts”) tend to go **out of date** very quickly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**
- Agile methods expect software to **start small and evolve** over time
 - Embraces **software evolution** instead of fighting it

Supporting Evolutionary Design

Traditional design advice says to anticipate changes
Designers often anticipate changes that don't happen



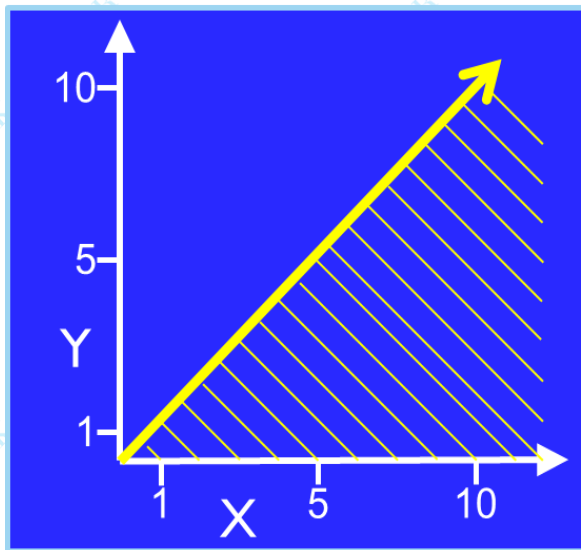
Both anticipated and unanticipated changes affect design

The Test Harness as Guardian (4.2)

What is Correctness ?

Traditional Correctness
(Universal)

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



Agile Correctness
(Existential)

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

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
- 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

But software engineers aren't mathematicians!

Test Harnesses Verify Correctness

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

- 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

Continuous Integration

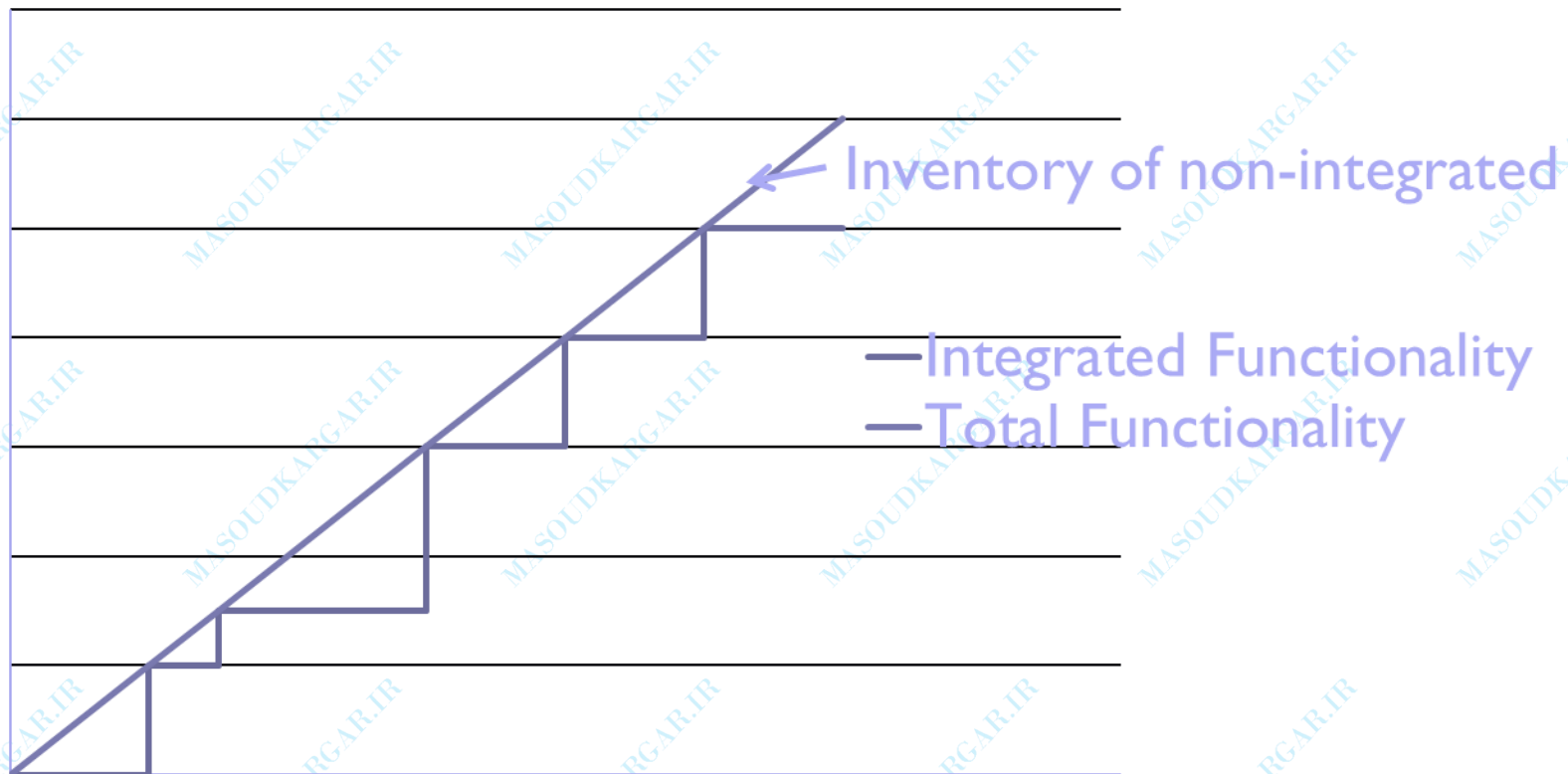
- Agile methods work best when the current version of the software can be run against all tests at any time

A continuous integration server rebuilds the system, returns, and re-verifies tests whenever any update is checked into the repository

- Mistakes are caught earlier
 - Other developers are aware of changes early
 - The rebuild and reverify must happen as soon as possible
- Thus, tests need to execute quickly

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

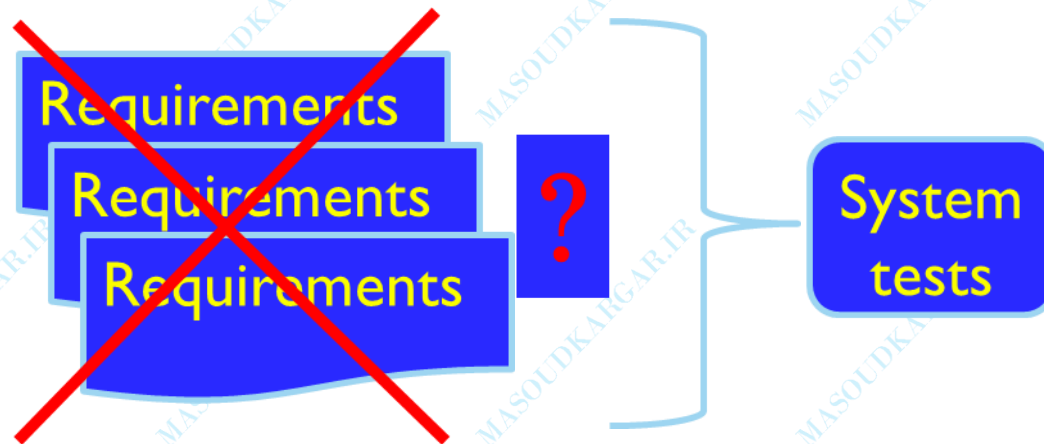
Continuous Integration Reduces Risk



Non-integrated functionality is dangerous!

System Tests in Agile Methods

Traditional testers often design system tests from requirements



But ... what if there are no traditional requirements documents ?

User Stories

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

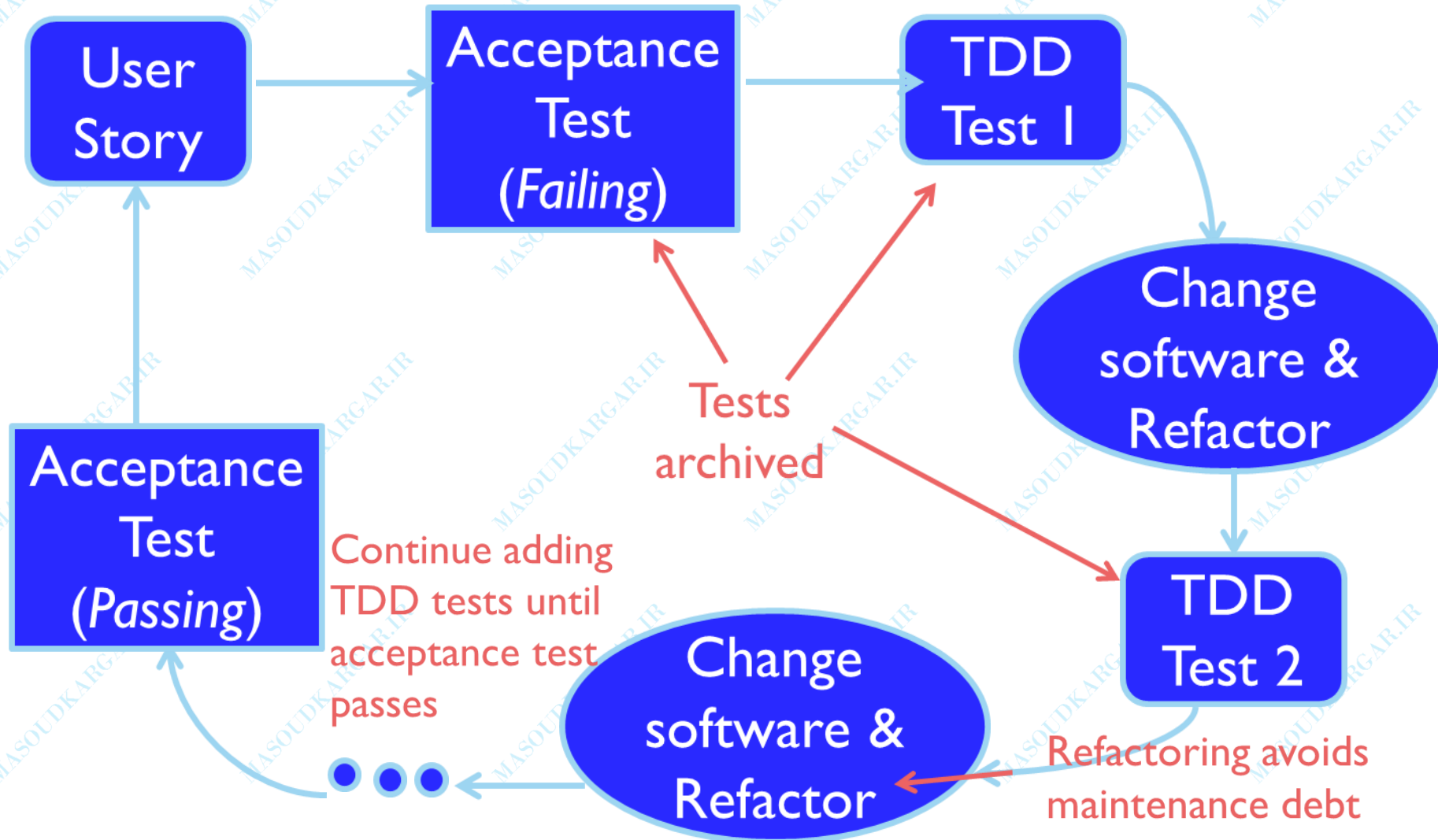
Withdraw money from
checking account

Agent sees a list of today's
interview applicants

Support technician sees
customer's history on
demand

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

Acceptance Tests in Agile Methods



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

How to apply TDD to legacy software with no tests?

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

Incremental TDD

Test-Driven Development

- When a change is made, add TDD tests for **just that change**
_Refactor
- As the project proceeds, the collection of TDD tests continues to **grow**
- Eventually, the software will have **strong TDD tests**

The Testing Shortfall

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

NO!



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

The agile methods literature does not give much guidance

What Should Testers Do?

Ummm ... Excuse me, Professor ...



What do I DO?

Design Good Tests

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



Part 2 of
book ...

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

Summary

- More companies are putting **testing first**
- This can dramatically **decrease costs** and **increase the quality**
- A different view of “**correctness**”
 - _Restricted but practical
- Embraces **evolutionary design**
- TDD is definitely **not** tested automation
 - _Test automation is a **prerequisite** to TDD
- **Agile tests** aren't enough