Scrum for Developers

Altran Education Services
Alexandre Cuva
Coach Agile, CSM, CSPO, HSPTP

Day Two Overview

- The Inception Deck
- Product Backlog
- User Stories
- Estimating
- Scrum Simulation
- Day Retrospective

- Any Hot Questions ?

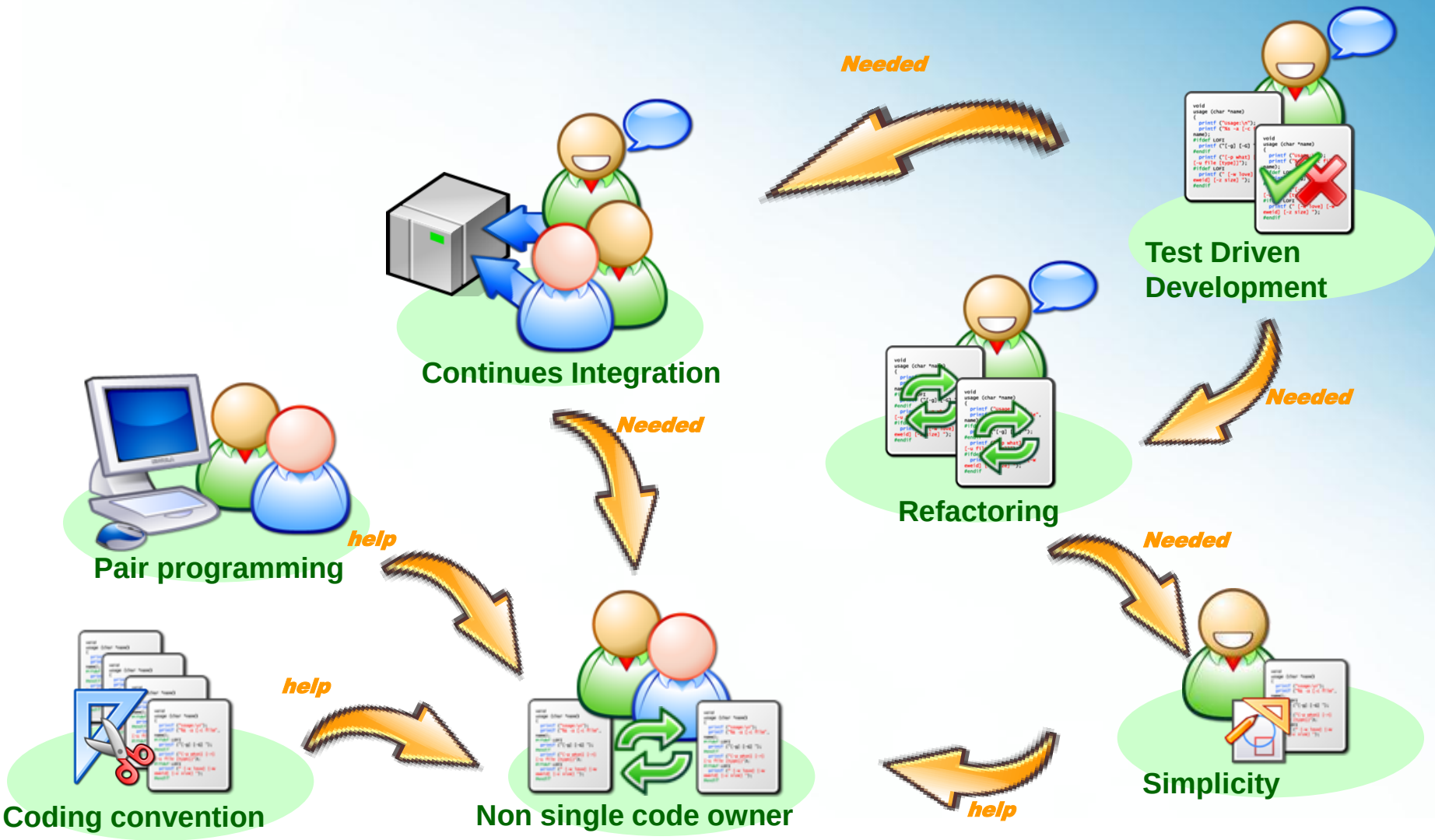
Agenda

Day Three

- eXtreme Programming
- Build Thing Right
- Unit Testing
- Day Restrospective

Practical Stuff

XP Rules


Needed
Test Driven Development
Continues Integration
Needed
Needed
Refactoring
Pair programming
help
Needed
help
Coding convention
Non single code owner
help
Simplicity

XP Best Practice

Test Driven Development
- The test is written first
- There is no functional code without testing
- The tests are performed tirelessly

Customer on site!
- To improve communication
- To respond quickly to questions
- To assess as quickly as what is developed

Pair programming
- We develop two on the same workstation
- This allows to share knowledge and skills
- This allows you to read immediately
- This will be less distracted (web, mail, ...)

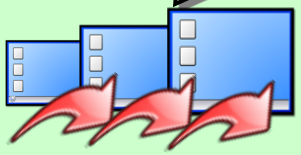
The planning game
- The client expressed needs
- The developers estimate charges
- The customer prioritizes tasks

XP Best Practice

Refactoring
- Commencer par faire le plus simple possible
- Ne pas hésiter ensuite à réécrire du code pour l'optimiser ou le clarifier

Continuous Integration
- Developers deliver their work frequently
- The work of all is consolidated and checked regularly

Short iterations
- fast Feedback
- Flexibility to change
- Visibility of Customer
- Team motivation



Simple design
- No architecture and design without obvious motivation
- Implementation directly as possible
- YAGNI: You Are not Gonna Need It!

XP Best Practice

Collective code ownership
- Any developer can work on a code
- We must regularly exchange topics
- This reduces the impact of arrivals and departures

Use of metaphor
- An invitation to be teaching
- Rely on the references of the speakers (users)
- Avoid jargon technocrat

Pace of work in progress
- Avoid the rush at the end of iteration or project
- No overtime
- Propose realistic schedules

Coding
- A shared format readability
- Use tools (IDE, continuous integration) to check and format

Exercises

- With your team select one of the XP Best practices
- Write 3 Against and 3 Answer to adopt
- Share with the others
- 10 min

BUILD THING RIGHT

Eliminate Waste

Stop doing things customers don't value!

- Value is …
  - Seen through the eyes of those who pay for, use, support, or derive value from our systems.
- Waste is …
  - Anything that depletes resources of time, effort, space, or money without adding customer value.

Put on customer Glasses

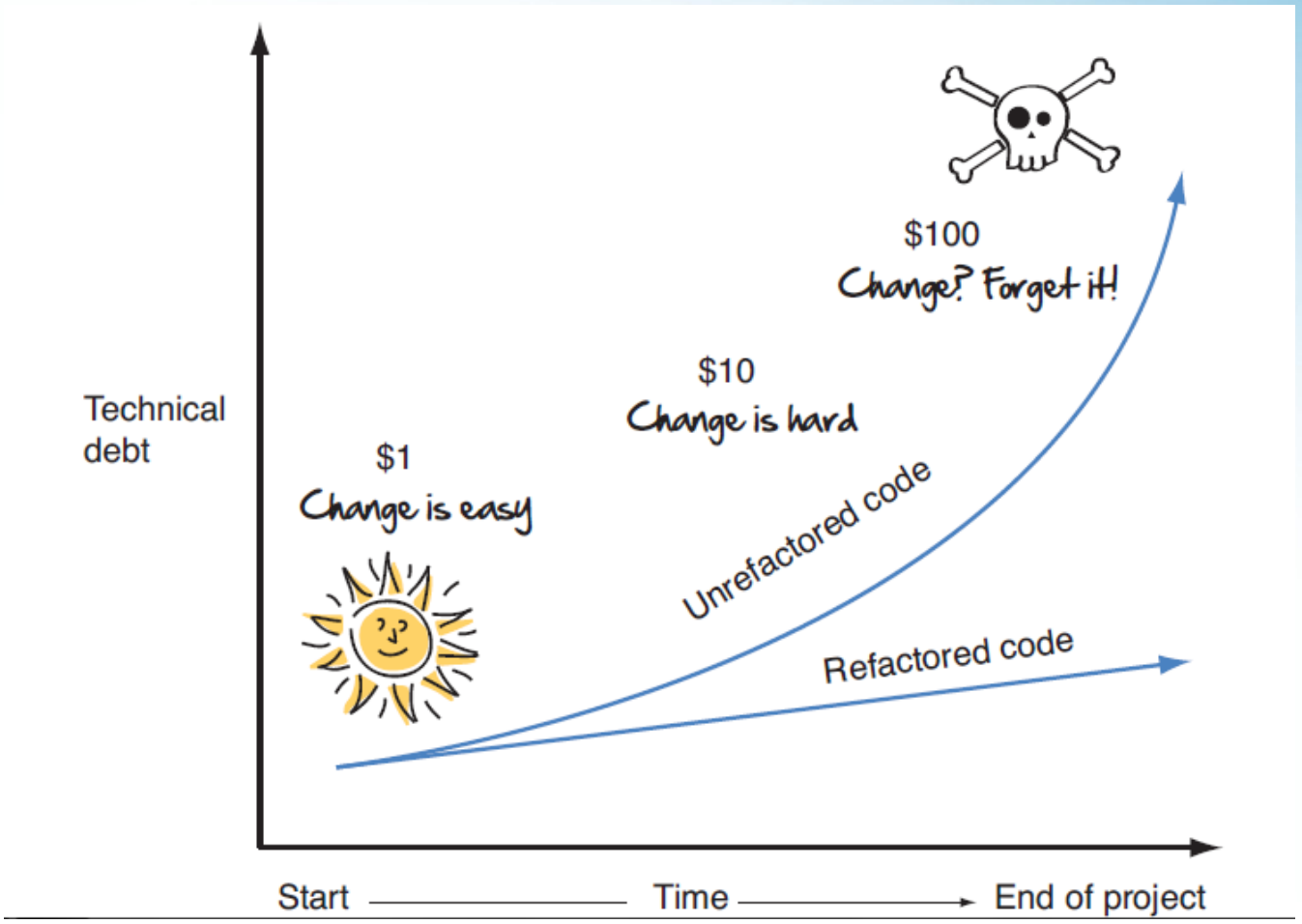
Source: Poppendieck.L.L.C

Technical Debt

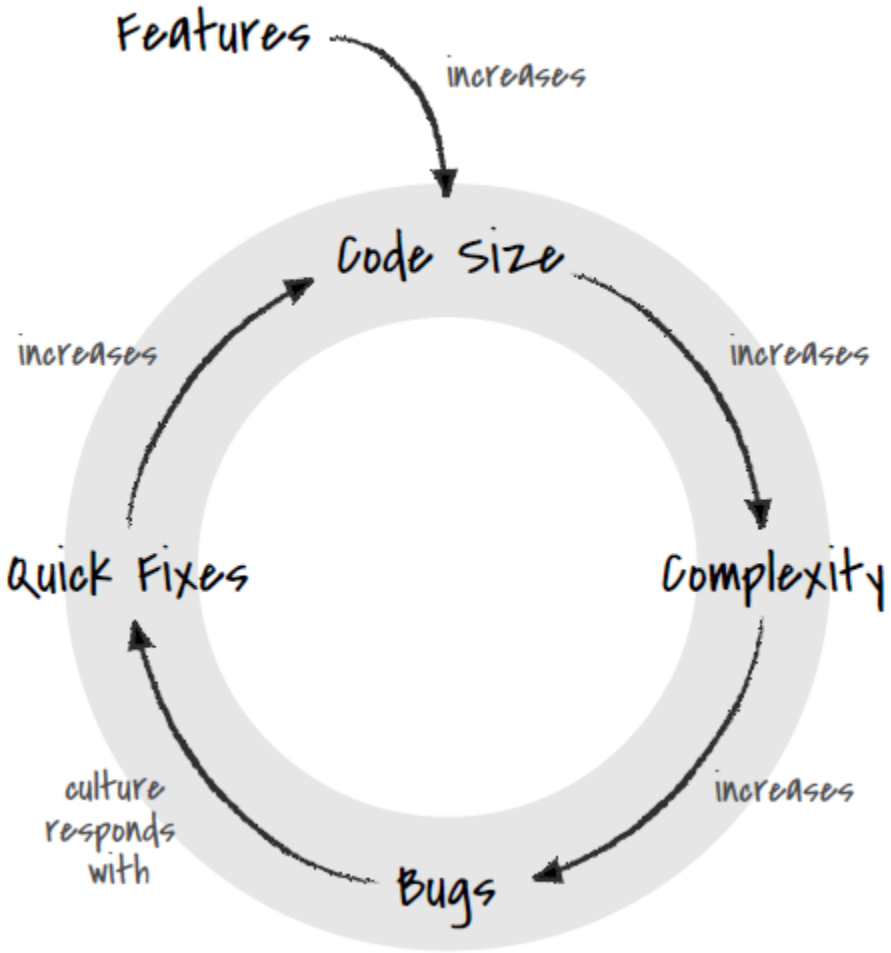
Technical Debt: Anything that make code difficult to change

- ✓ Sloppy Code
  Code reviews → standards, quality, knowledge transfer.
- ✓ No Test Harness
  Code without a test harness is Legacy Code.
- ✓ Dependencies
  A divisible architecture is fundamental.
- ✓ Unsynchronized Code Branches
  The longer two code branches remain apart, the more difficult they are to merge together.

Technical Debt



Technical Debt



What About Legacy Code?

Stop Digging! (Michael Feathers)

- ✓ Don't bother to retrofit unit tests
  - × Add unit tests whenever you touch the code.
- ✓ Create a regression test harness for key API's
  - × Prioritize by risk and time-to-test
  - × Add automated acceptance tests for all new features.

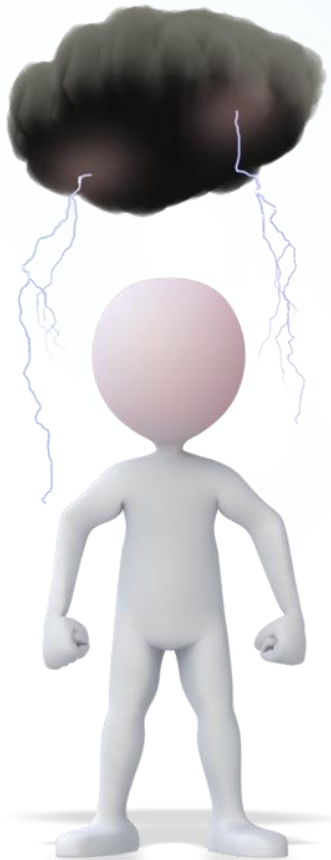
Strangler Application (Martin Fowler)

- ✓ Gradually create a new system around the edges of the old one.

Mikado Method (Ola Ellnestam & Danuel Bround)

- ✓ Objective: Create a dependency graph so you can make changes at the leafs of a dependency chain
- ✓ How: Make change, graph errors (dependencies), revert.
  - × Start with one of the errors, refactor, find & graph errors, revert.
  - × Repeat until no error occur – now your are at the leaf.
  - × Refactor back up the dependency chain

Defects



The longer Defects are Undetected, the harder they are to find

Source: Poppendieck.L.L.C

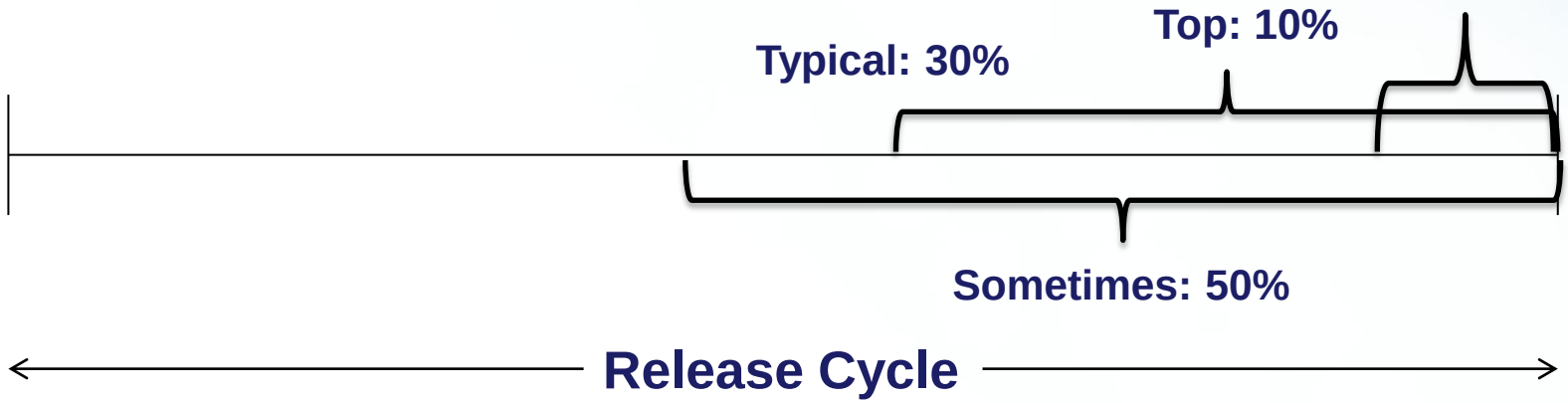
Build Quality In

Every software development process ever invented has had the same primary goal – find and fix defects are early in the development process as possible. If you are finding defects at the end of the development process – your process is not working for you.

How good are you?

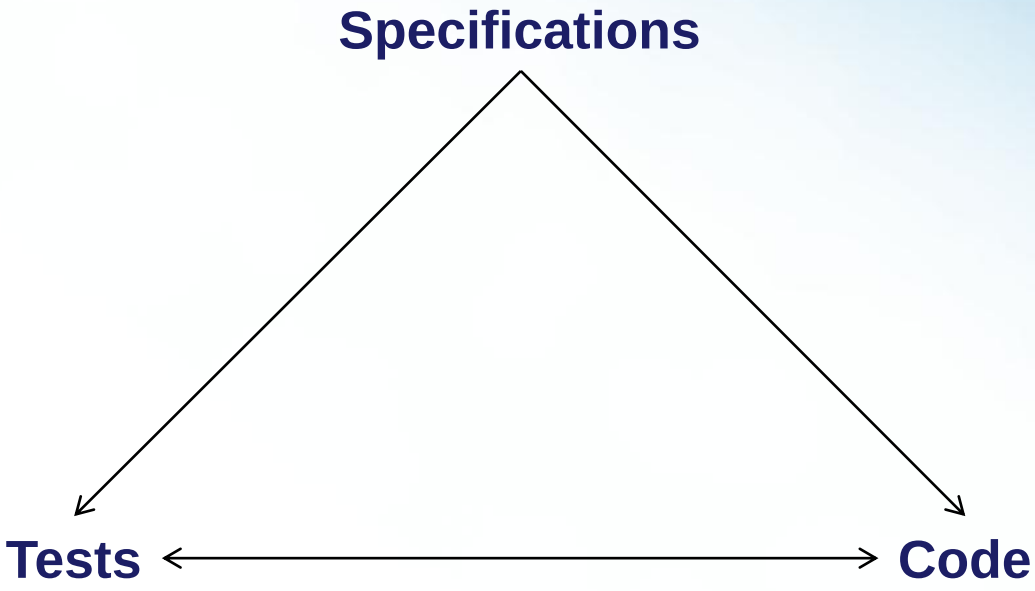
When in your release cycle do you try to freeze code and test the system?

What percent of the release cycle remains for this “hardening”?

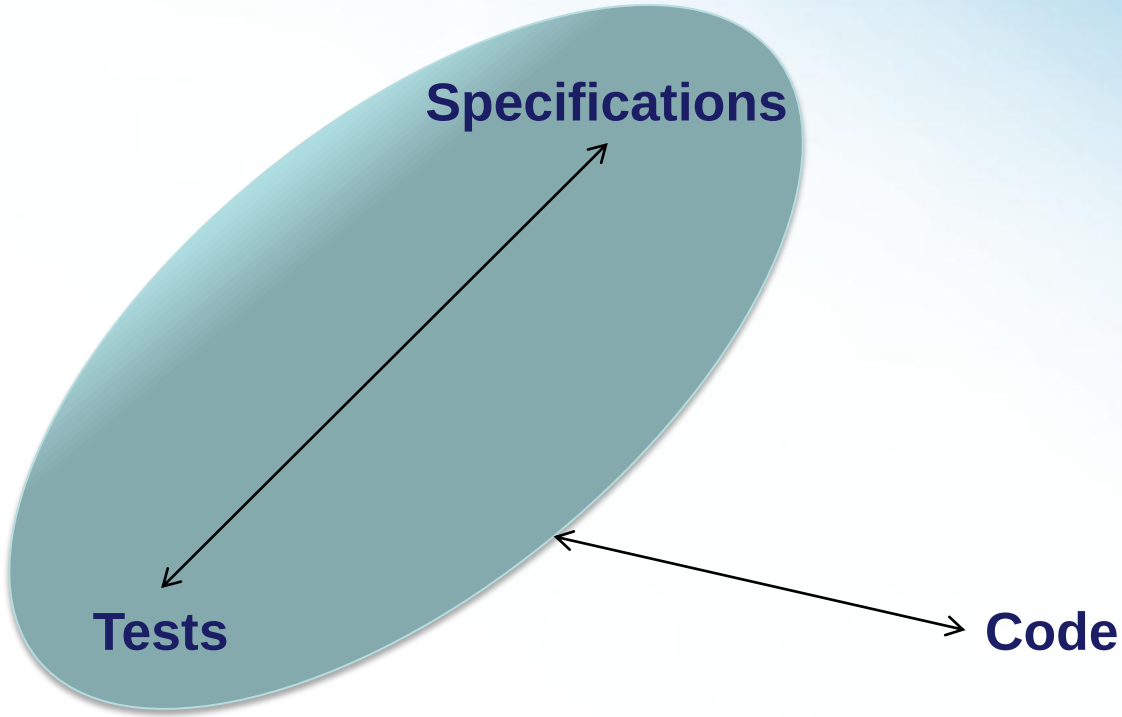


Source: Poppendieck.L.L.C

Defect Injection Process

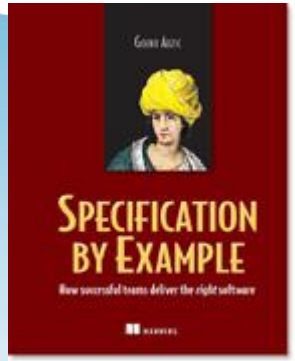


Defect Prevention Process



Source: Poppendieck.L.L.C

Specification by Example



- For each feature:

1. Design
   a. Specify: Discuss and agree on examples of intended behavior.
   b. Automate: Put the examples in a framework such as cucumber.
2. Implement
   1. Develop: Write feature code using TDD
   2. Refactor: Clean up the code to keep it simple.
   3. Regression: Check that all completed examples work.
3. Deploy
   1. Validation: Determine value ASAP

Sources: Specification By Example

Application Testing

	Continuous Integration	Test to specification	Test to failure	
Automated if Practical: Daily →	Product Design	Acceptance Tests	Exploratory Tests	← Manual: As early as possible
Automated Every build →	Technical Design	xUnit Tests	Stress Tests	← Tool-Based: As early as practical
Test Interaction Layer Separately →	Interaction Design	Presentation Layer Tests	Usability Tests	← Manual: As early as possible

Source: Agile Testing

Assessment: Basic Disciplines

1. Low Dependency Architecture
2. Coding Standards
3. Source Control / Configuration Mgt
4. Automated Specification Examples
5. Automated Unit Tests
6. STOP if tests don't pass
7. Design/Code Reviews
8. Refactoring is a Habit
9. Continuous Integration
10. System Testing / UAT – early & often
11. Customer Validation
12. Escaped Defect Root Cause Analysis

Choose an organization

Rate the organization on a scale of 0-5 (high).

Total the score.

Plot the totals (front of room)

Report

Source: Poppendieck.L.L.C

Unit testing is a method by which individual units of source code are tested to determine if they are fit for use. A unit is the smallest testable part of an application.” wikipedia.org

UNIT TESTING

Exercises

- With your team write 3 Cons and 3 For, about using Unit Test
- Share with the others
- 5 min

Why Unit Test

- Debugging is a time consuming process
- When new functionality is added, how do we make sure the old one doesn't break
- By looking at a Unit Test, you can see the class in action, which lets you easily understand its intent and proper use
- Unit tests are the only real measure of project health and code quality

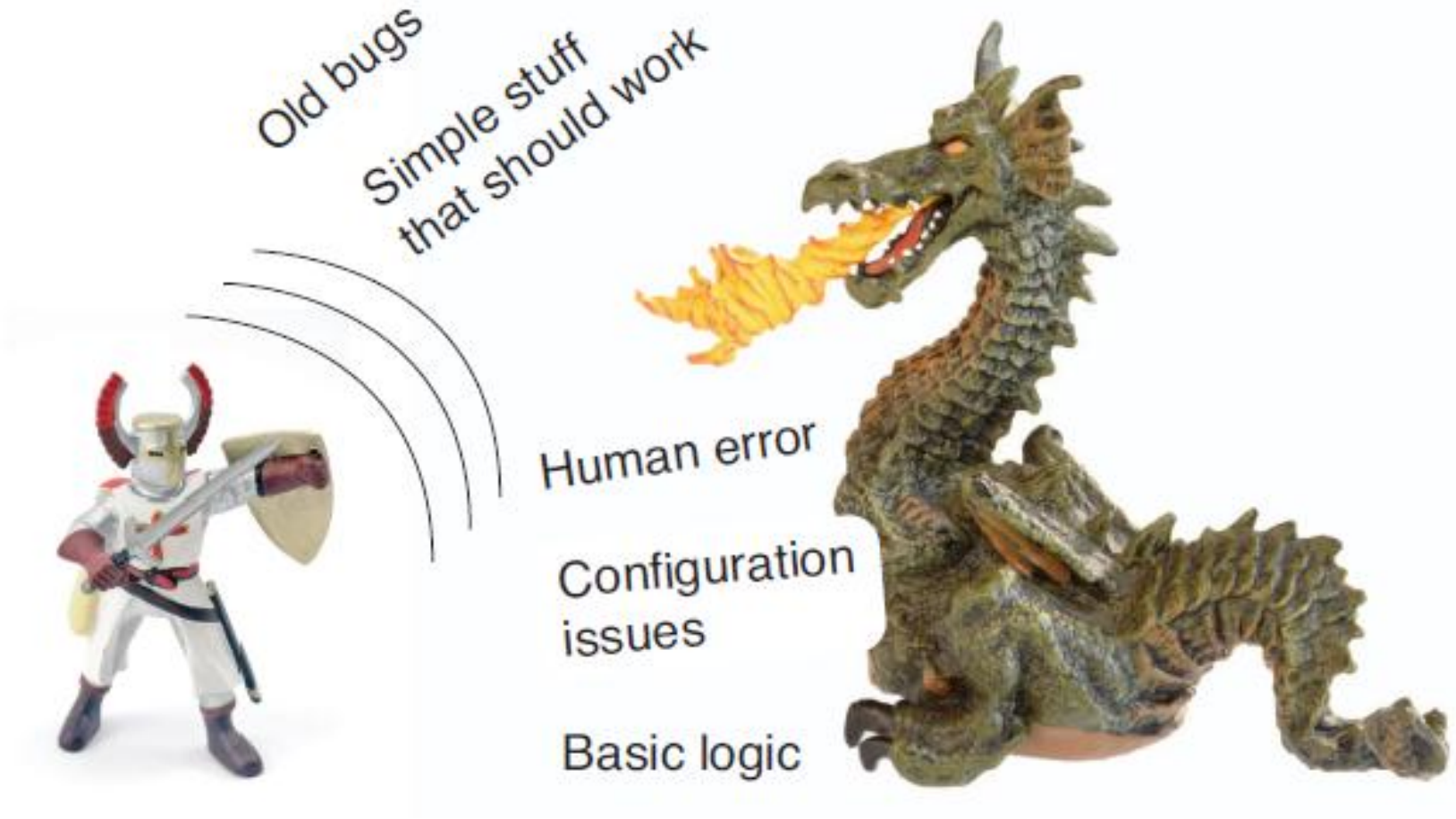
Why not Unit test

- I never make mistakes
- The functionality is trivial
- Tests slow me down
- Management wont let me
- I don't know how to test

Good Tests

Automatic	•**Unit tests need to run automatically**. We mean “automatically” in two ways: invoking the tests and checking the results.
Thorough	•**Good unit tests are thorough**; they test everything that’s likely to break. But just how thorough?
Repeatable	•Just as every test should be independent from every other test, the tests must be independent of the environment as well. The goal remains that every test should be able to run over and over again, in any order, and produce the same results.
Independent	•**Tests need to be kept neat and tidy**, which means keeping them tightly focused and independent from the environment and each other
Professional	•**The code we write for a unit test is real**; this means it must be written and maintained to the same professional standards as production code

Test Protect you



Source: The Agile Samnurai by Jonathan Rasmusson

Unit Test Framework

- Microsoft.VisualStudio.TestTools.UnitTesting
- NUnit
- XUnit

Day one Retrospective

- What went well
- What the two thing you will change for the day 2