SW Development Testing

Jari Tahvanainen

Testing Specialist

Nokia/Maemo Devices

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0}$$

$$\nabla \cdot \mathbf{B} = 0$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$$

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{J} + \mu_0 \epsilon_0 \frac{\partial \mathbf{E}}{\partial t}$$

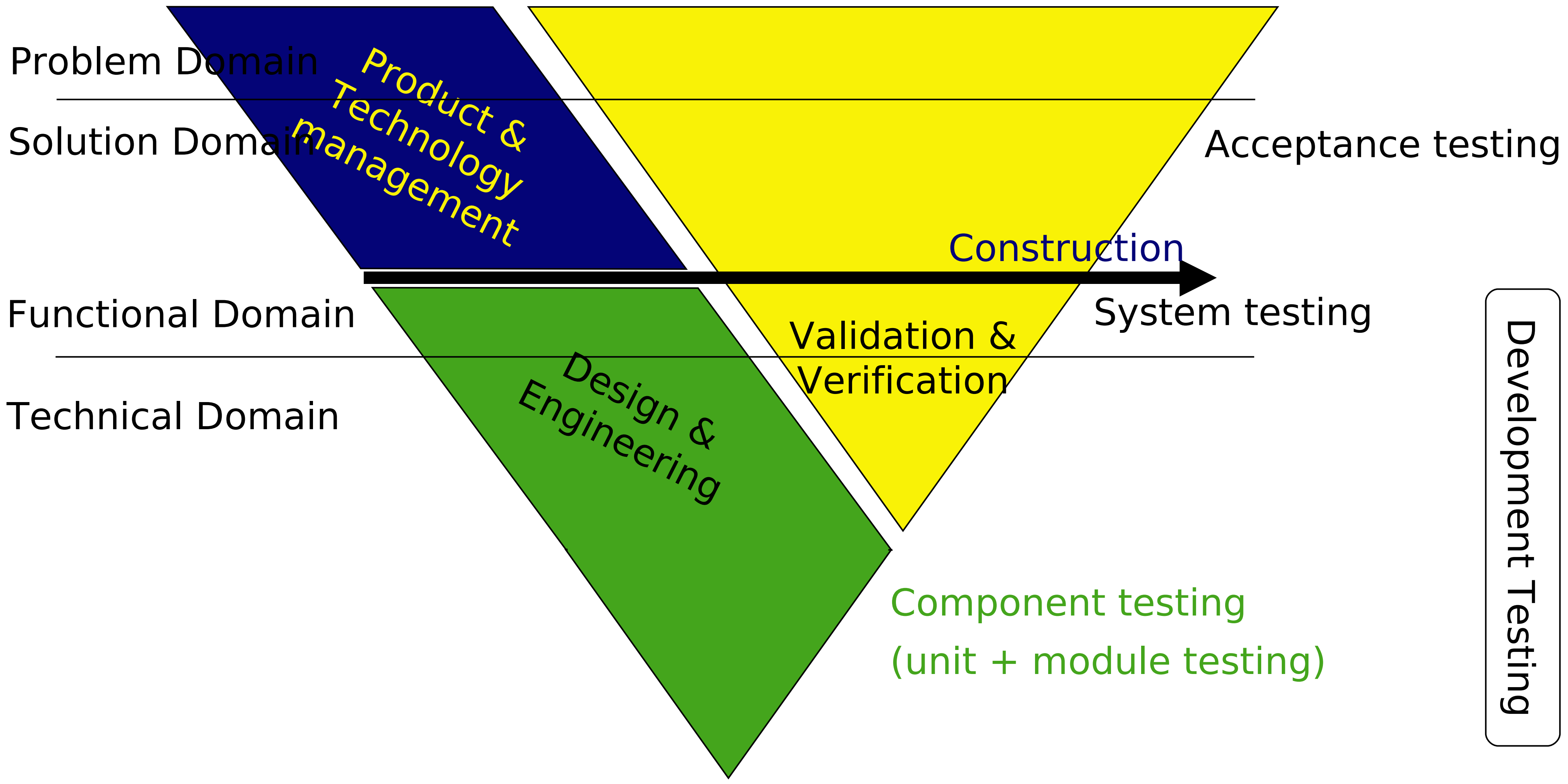
Content

- Agile stages – Fragile stages
- Agile practices

- Development testing
    - ET
    - TDD
    - ATDD
    - CI
    - Confirmation

- Evolution of Agile Testing

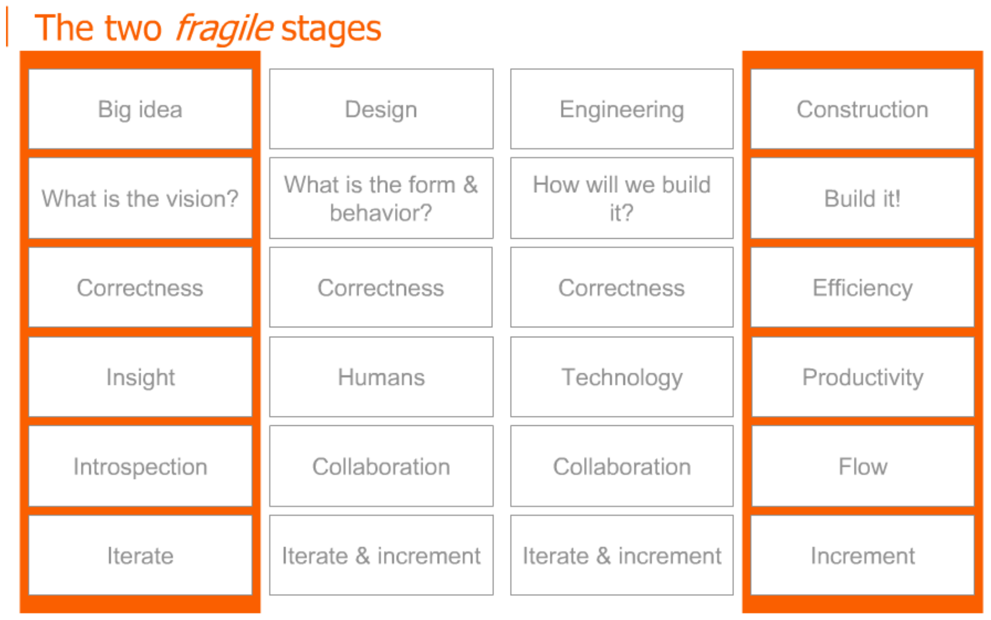
Test levels (and “engineering” domains)



Fragile Stages

http://www.cooper.com/journal/agile2008/

- “And the two outer stages are the fragile stages.”
- “The first is utterly unpredictable and magical. It can come from anyone, anywhere, at any time. You cannot really seek it out, but only cherish it when it happens.”
- “And do remember that is almost always completed before we get there, and we rarely encounter it in any project.”
- “The last stage is fragile because it is so big, so lengthy, so delicate, so difficult, and so critical to the success of the whole, that disturbing it in any way is foolishly, hellishly expensive.”
- “In these two stages, there is simply no advantage to putting lots of people in a room together to work openly and collaboratively, regardless of how intelligent or well-intentioned they are. The construction stage in particular demands quiet, uninterrupted, solitude: programming time.”
- “As you can clearly see, each stage is different, and each stage requires different skills, tools, and temperament.”

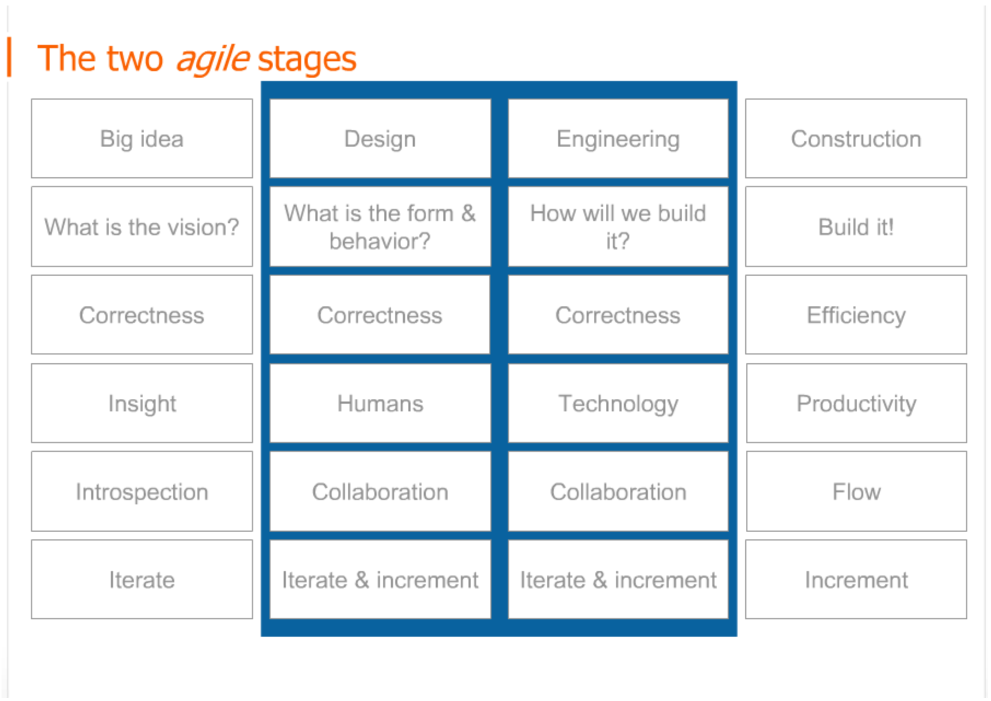

The two *fragile* stages

Big idea	Design	Engineering	Construction
What is the vision?	What is the form & behavior?	How will we build it?	Build it!
Correctness	Correctness	Correctness	Efficiency
Insight	Humans	Technology	Productivity
Introspection	Collaboration	Collaboration	Flow
Iterate	Iterate & increment	Iterate & increment	Increment

Agile Stages

http://www.cooper.com/journal/agile2008/

- “I’m sure you can see that the two middle stages, the design stages, are agile.”
- “They are about investigation, problem-solving, openness, teamwork, collaboration, iteration, incrementing, explaining, and sharing, all in an effort to achieve the right answer.”
- “These two stages can really benefit from the open, collaborative, democratic, iterative nature of the Agile process. This is true regardless of which people or what craft is involved.”

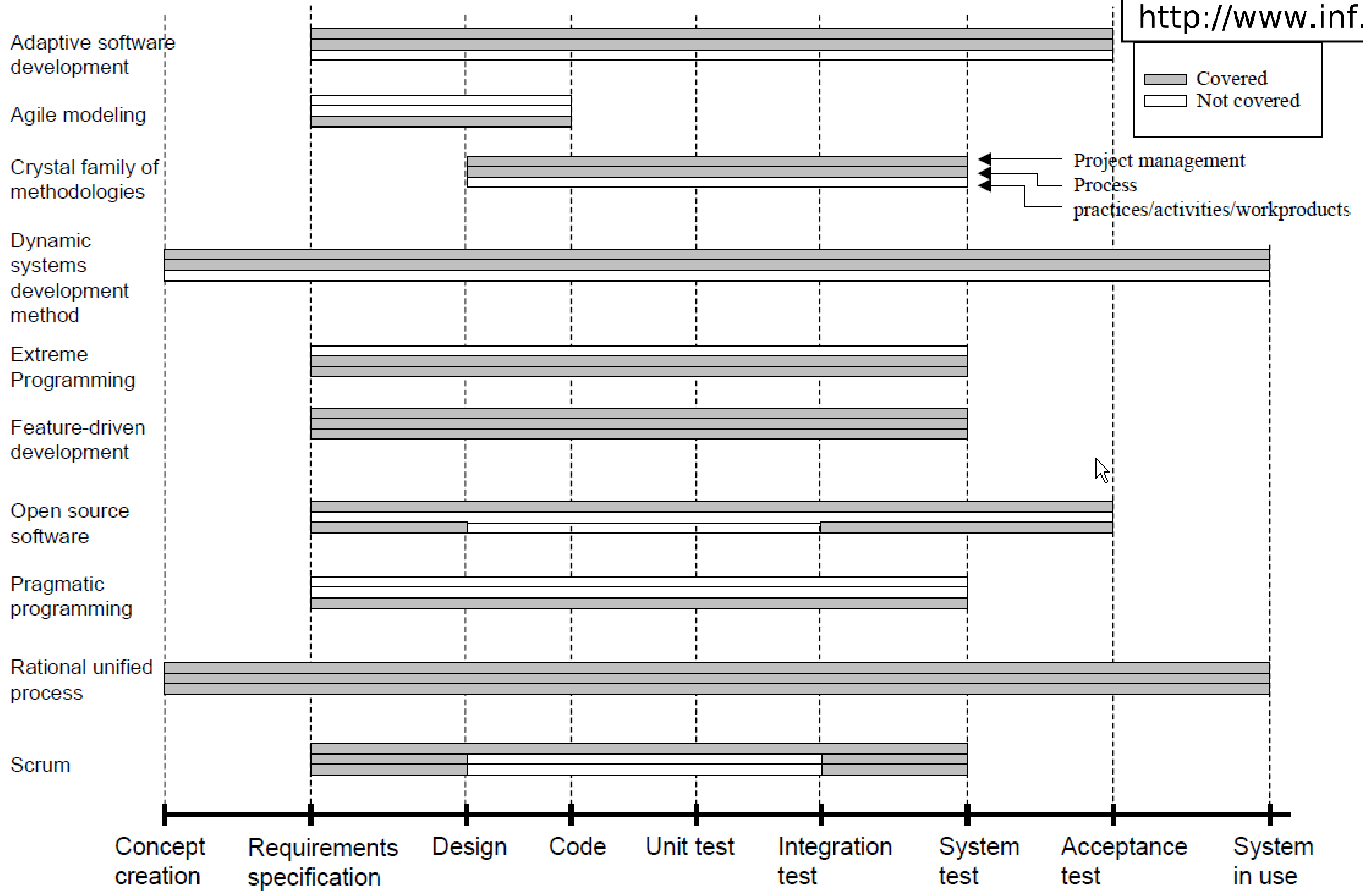


The two *agile* stages

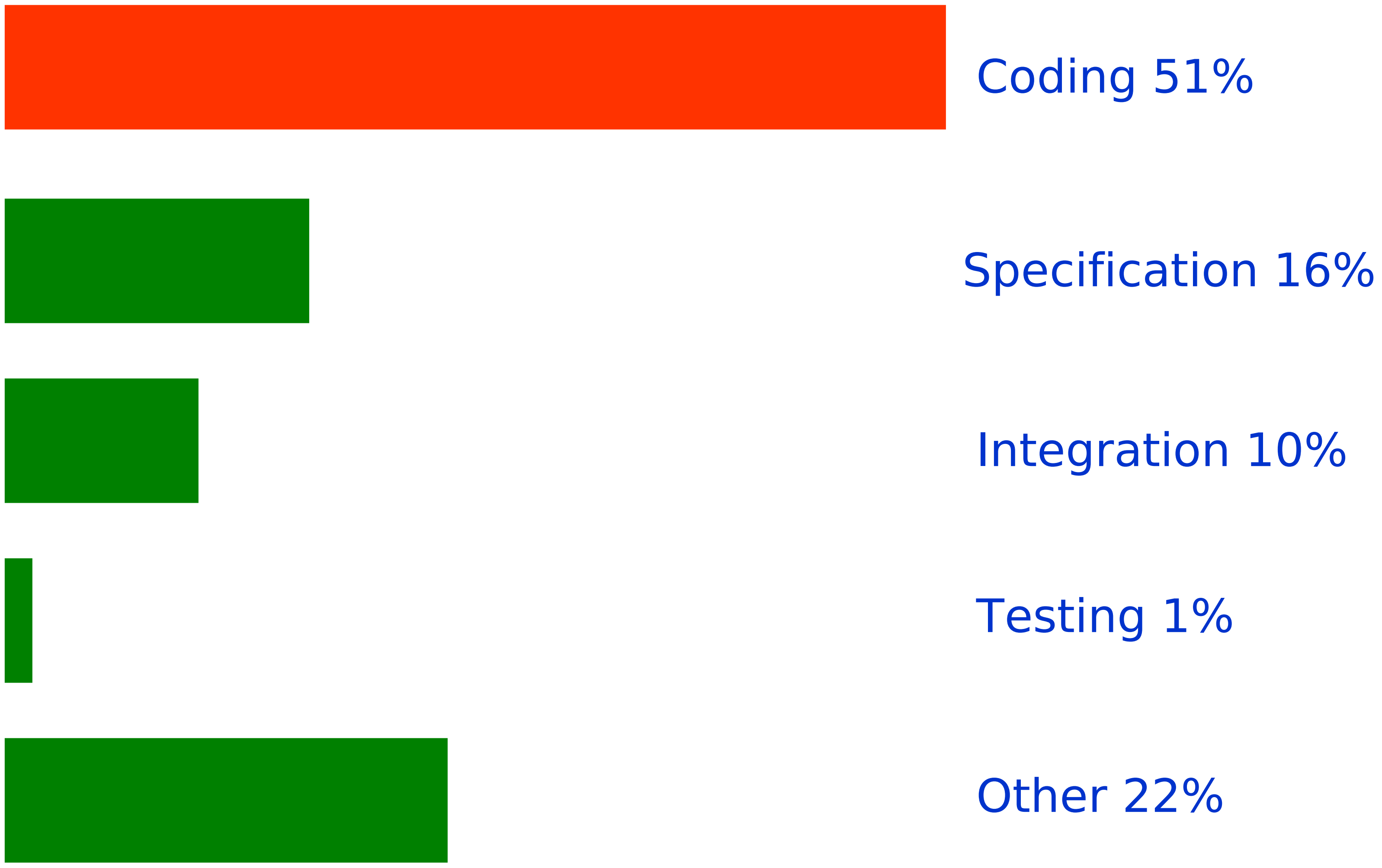
Big idea	Design	Engineering	Construction
What is the vision?	What is the form & behavior?	How will we build it?	Build it!
Correctness	Correctness	Correctness	Efficiency
Insight	Humans	Technology	Productivity
Introspection	Collaboration	Collaboration	Flow
Iterate	Iterate & increment	Iterate & increment	Increment

Agile - Software Development Life-Cycle Support

VTT Publications 478
http://www.inf.vtt.fi/pdf/



Faults injected in different development phases



Development Testing

- **Development Testing** (a.k.a. Developer Testing) by definition is [ISTQB Glossary] "Formal or informal testing conducted during the implementation of a component or system, usually in the development environment by developers". Depending the competences of the developer one can include to this things from component testing (unit, module and component integration testing put together) to feature tests (giving confirmation for the user story) (Test Levels).
  - **Unit tests** are so named because they each test one unit of code. These type of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. Whether a module of code has hundreds of unit tests or only five is irrelevant. One function might have multiple tests, to catch corner cases or other branches in the code. A test suite of unit tests should never cross process boundaries in a program, let alone network connections. Doing so introduces delays that make tests run slowly and discourage developers from running the whole suite.
  - **Module test** tests a module of code similar way than unit tests - the difference is that test are generated black-box style (without reference to the internal structure of the module).
  - **(Component) Integration test** - Introducing dependencies on external modules or data also turns unit tests into integration tests. If one module misbehaves in a chain of interrelated modules, it is not so immediately clear where to look for the cause of the failure. Tests are checking ready made part of the developer's product (implemented tasks and user stories) with simulators (stubs, test drivers) for missing external dependencies.
  - **Feature test** is checking whole developer's product (features and user stories done) by pretending to be a user

Development Testing Intent

- Development testing is part of agile software development
    - In agile development, a product is developed in short cycles (sprints), typically 1-4 weeks. After every cycle, the customer is provided with a potentially shippable product which provides him business value. The quality of the released product is assured with **adequate testing by the whole team**, not just testers.
    - Development testing has to adapt to the short release cycle of agile development. **Test automation has a strong role** - it is recommended that unit and module tests, component integration tests and even feature/application "acceptance“ tests are automated.
    - **Exploratory testing** with minimal documentation should be used to complement automated tests.
    - In agile environment tester has the possibility to apply the traditional testing techniques and skills in more innovative ways than before. In addition, he must possess new skills, most importantly, he must have an **agile mindset**.

Development Testing – Practices (1/3)

- **Exploratory Testing (ET)**
  - Is an approach in software testing that is concisely described as simultaneous learning, test design and test execution.
  - Seeks to find out how the software actually works, and to ask questions about how it will handle difficult and easy cases. The testing is dependent on the tester's skill of inventing test cases and finding defects. The more the tester knows about the product and different test methods, the better the testing will be.
  - When performing exploratory testing, there are no exact expected results; it is the tester that decides what will be verified, critically investigating the correctness of the result.
- **Test Driven Development (TDD)**
  - Is a development practice in which test cases are implemented before the actual program code.
  - Test cases are implemented by developers, usually with some test framework.
  - The implemented unit tests answer to the question: "Does the code function as how the programmer intended it to work?"
  - TDD is implemented in code level - each test is verifying one small unit of code in isolation
  - Cycle: Write the test, make it green, make it clean!

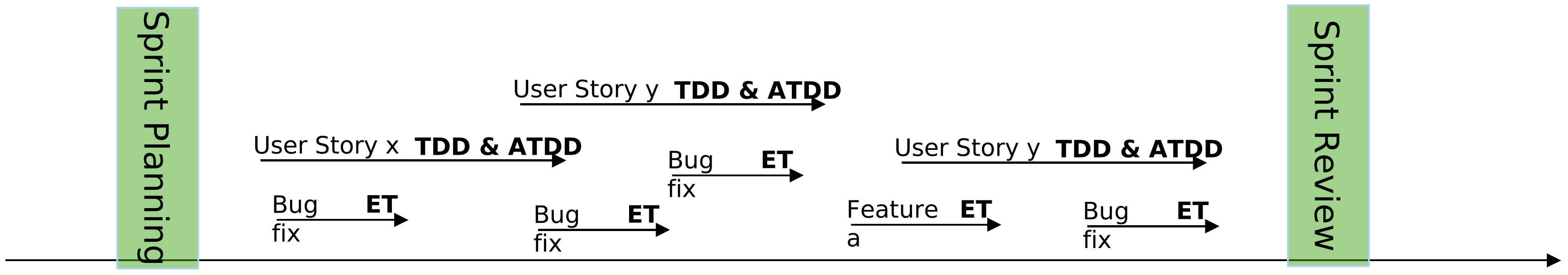
Development Testing – Practices (2/3)

- **Acceptance Test Driven Development (ATDD)**
  - Acceptance Test Driven Development is testing oriented development practice in which "acceptance" test cases are implemented before the implementation of the User Story starts
  - The main goal of the tests is to answer to the question: "Does the system do what the Customer wants?"
  - Tests are result of collaboration between developers, testers and business stakeholders
    - Test cases are implemented before the implementation of the User Story starts (similar as in TDD)
  - Tests should be end-to-end tests that drive the system through public interfaces (gui, cmd)
    - It is recommended that most test is conducted below the GUI using "Business Layer", as GUI is the most volatile layer in any application.
  - Writing acceptance tests make the team to use the application and see it from the end-users perspective. That helps the team
    - find any peculiar work flows
    - understand how the end-users sees the application (end-user experience)
  - Remember to address all applicable quality aspects (functionality, performance, reliability, security/robustness, usability)
  - Cycle:
    - Discuss: Understand what Business Stakeholder needs from any particular feature
    - Distill: Collaborate with Business Stakeholders to distill the needs into a set of acceptance tests
    - Develop: Write code to implement the requested features using TDD
    - Demonstrate: Show the Business Stakeholder the new feature and request feedback

Development Testing – Practices (3/3)

- **Continuous Integration (CI)** “is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.“ [Martin Fowler]
    - Everyone Commits Every Day
    - Make your build self-testing
    - Everyone can see what's happening
    - Automate the Build
    - Keep the Build Fast
    - Make it Easy for Anyone to Get the Latest Executable
    - Test in a Clone of the Production Environment
        - The point of testing is to flush out, under controlled conditions, any problem that the system will have in production. A significant part of this is the environment within which the production system will run. If you test in a different environment, every difference results in a risk that what happens under test won't happen in production.

Development Testing – Flow and Levels Example

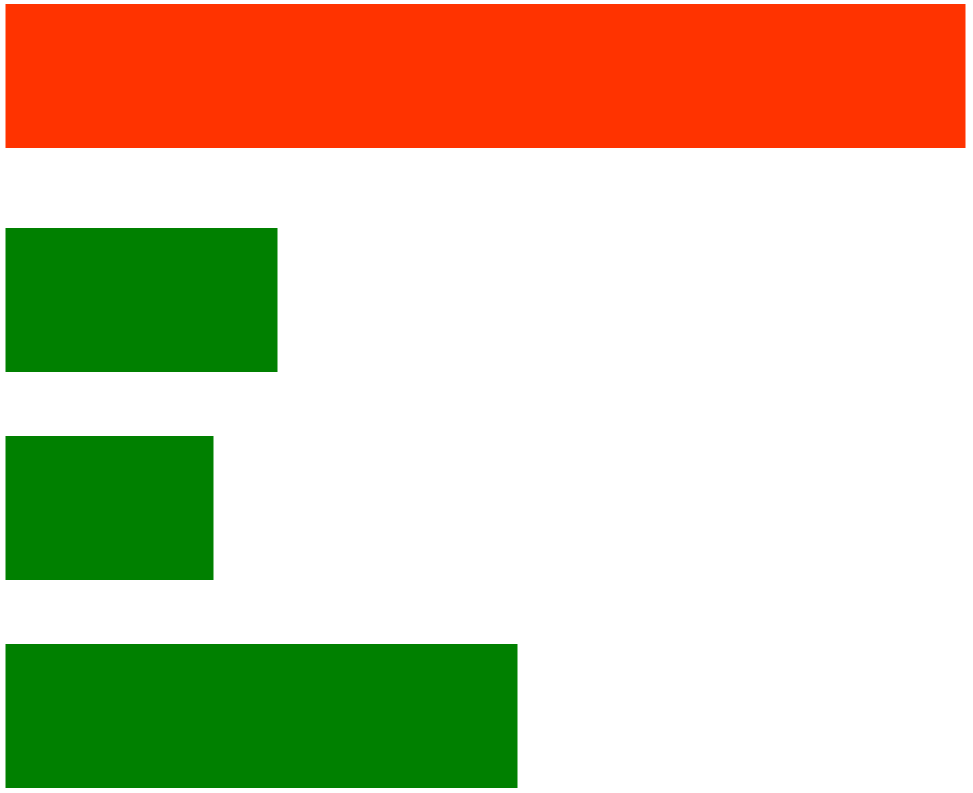


- Unit tests - test individual classes in isolation
  - Focus on component logic – TDD approach recommended
- Module tests - test user story with simulations (stubs, test drivers) for external dependencies
  - Focus on class interaction – TDD & ATDD approaches recommended
- Component integration tests - test ready made part of the team's product (implemented tasks and user stories) with simulators (stubs, test drivers) for missing external dependencies
  - Focus on feature interaction - TDD & ATDD & ET approaches recommended
- Application/Feature tests - Test whole team's product (features and user stories done) by pretending to be a user
  - Focus on mostly happy paths – ATDD & ET approaches recommended

Confirmation Testing

- **Confirmation Testing** – “Acceptance criteria” (a.k.a. conditions of satisfaction) – user story test objective:
  - By default user story is having three Cs - Card defining the story itself "As a <user role>, I want to <goal> so that <benefit>",and promise about the Conversation and Confirmation (these are essentially acceptance tests confirming that the story was coded correctly).
  - Confirmation - Acceptance criteria - should include quality aspects (or characteristics e.g. functionality, efficiency, reliability, usability, portability and maintainability) important for this particular development item. Aspects could have relative importance defined from which one can define test objective (or goal) telling what is tested and how well it is tested, e.g.:
    - Functionality - thorough testing (including security, localization, pre-certification, etc.)
    - Performance - light testing (e.g. Application opening and closing times, number of data queries the database per minute, etc.)
    - Reliability - average testing (e.g. Robustness testing, valgrind in use while executing functionality tests)
    - Usability – light testing
    - Portability - light testing (e.g. testing of the component in two different configurations, etc.)
    - Maintainability –light testing
  - Definition of these aims from testing point of view to have a test basis, agreed with the client of the test, of adequate quality to design the test cases.

Fault could have been avoided in “Coding”



Development testing 45%

Careful coding 13%

Tester within team 10%

Other 30%

Evolution of agile testing

- **Evolution of agile testing**
  - Unit testing for all new code
  - Before re-factoring legacy code write unit tests first. To enable re-factoring, improvements etc.
  - As side effect of re-factoring you start to have better unit test coverage
  - Start to measure code coverage on unit testing. Try to improve: complex areas, low coverage areas
  - Go for CI and keep it green
  - Start evaluating test tools for automated functional tests
  - Try to keep up with current sprint goals. Write automated functional test, API tests etc.
  - Go for CI and keep it green
  - While dealing with reported bugs:
    - Ensure that there are automated unit tests in place which reproduce the bug.
    - Ensure that there is an automated functional acceptance test case in place for verifying the bug.
- **What else? Can we do more?**
  - Do exploratory testing meanwhile you run your automation to find more bugs. So far we talked about regression only...
  - Learn test design techniques (Equivalence Partitioning, Boundary Value Analysis, All-Pairs, Cause Effect Graphing, State Machine etc.)
  - Model based testing

References

- TDD – Test Driven Development - http://en.wikipedia.org/wiki/Test-driven_development
- ATDD – Acceptance Test Driven Development - http://testobsessed.com/2008/12/08/acceptance-test-driven-developme
- BDD – Behavior Driven Development http://en.wikipedia.org/wiki/Behavior_Driven_Development and http://behaviour-driven.org/
- Terminology
    - http://www.istqb.org/downloads/glossary-current.pdf
    - http://eng.tmap.net/Home/TMap/Glossary.jsp
    - http://en.wikipedia.org/wiki/Software_testing

Agile Software development methods (1/4)

From "Agile software development methods, review and analysis" by Pekka Abrahamsson et.al. (VTT Publications 2002)

Gives procedure frame

- Scrum
  - Adaptive, quick, self-organizing product development process
  - Each sprint includes analysis, design, evolution, testing, delivery
    - 1st sprint Product Backlog (Business functionality/features and technology requirements) -> goal "demonstrate a key peace of user functionality on the selected technology".
  - Suitable for small teams of less than 10 engineers
  - XP and scrum can be implemented together

Gives process and "activities"

- eXtreme Programming
  - Iterations to release phase consists continuous integration and test
  - Testing
    - Tester help customer write functional tests.
    - Tester run functional tests regularly, broadcast test results and maintain testing tools
    - Software development is test driven. Unit tests are implemented before the code and are run continuously. Customer write the functional tests (with help from tester).
  - Idea: Schedule projects based on customer stories (use cases), evolutional delivery (adapted from Gilb)

Agile Software development methods (2/4)

From "Agile software development methods, review and analysis" by Pekka Abrahamsson et.al. (VTT Publications 2002)

Very short iterations: from hours to 2 weeks

- Feature driven development
  - Focuses on design and building phases
  - Testing
    - Toolsmith builds small tools for development, test and data conversion
    - Tester verify that the system being produced will meet the requirements of the customer. May be an independent of a part of the project team.
    - Inspections, regular builds, configuration management, progress reporting …
    - "worthy of serious consideration by any software development organization that needs to deliver quality, business critical software systems on time"

Not agile as such

- Rational Unified Process
  - Iterative approach for object orienting systems, and it strongly embraces use cases for modeling requirements
  - Develop software iteratively, Manage requirements, use component-based architectures, visually model software, verify software quality, control changes to software
  - Not agile as such but can be used in an agile manner (Agile modeling, etc.)

Agile Software development methods (3/4)

From "Agile software development methods, review and analysis" by Pekka Abrahamsson et.al. (VTT Publications 2002)

- Dynamic Systems Development Method

1st truly agile software development method

  - Framework for Rapid Application Development
  - Idea: Fix time and resources, and then adjust the amount of the functionality accordingly
  - …
  - The focus is frequent delivery of the products
  - Testing is integrated throughout the lifecycle
    - Every system component should be tested by the developers and users as they are developed. Testing is also incremental, and the tests builds more comprehensive throughout the project. Regression testing is particularly emphasized because of the evolutionary development style
- Adaptive Software Development

Mission and risk driven, adaptive

  - Focuses mainly on problems developing complex, large systems
  - Project postmortems are seen as critically in high-speed and high-change projects, where agile development takes place
  - Mission-driven, component-based, iterative, time-boxed, change-tolerant, risk-driven
  - Does not enforce the use of co-located teams

Agile Software development methods (4/4)

From "Agile software development methods, review and analysis" by Pekka Abrahamsson et.al. (VTT Publications 2002)

Volunteer based

- Open Source Software development
  - Is collaborative way of widely dispersed individuals to produce software with small and frequent increments.
  - Is not a compilation of well defined and published software development practices constituting a written eloquent method.
  - There is no explicit system-level design, or even detailed design
  - There is no project plan, schedule, or list of deliveries
  - Tester
    - Other individual who, in course of using software, perform testing, identify bugs and deliver software problem reports.

Newest comer

- Agile Modeling
  - Idea is to encourage developers to produce advanced enough models to support accurate design problems and documentation purposes, while still keeping the amount of models and documentation as low as possible