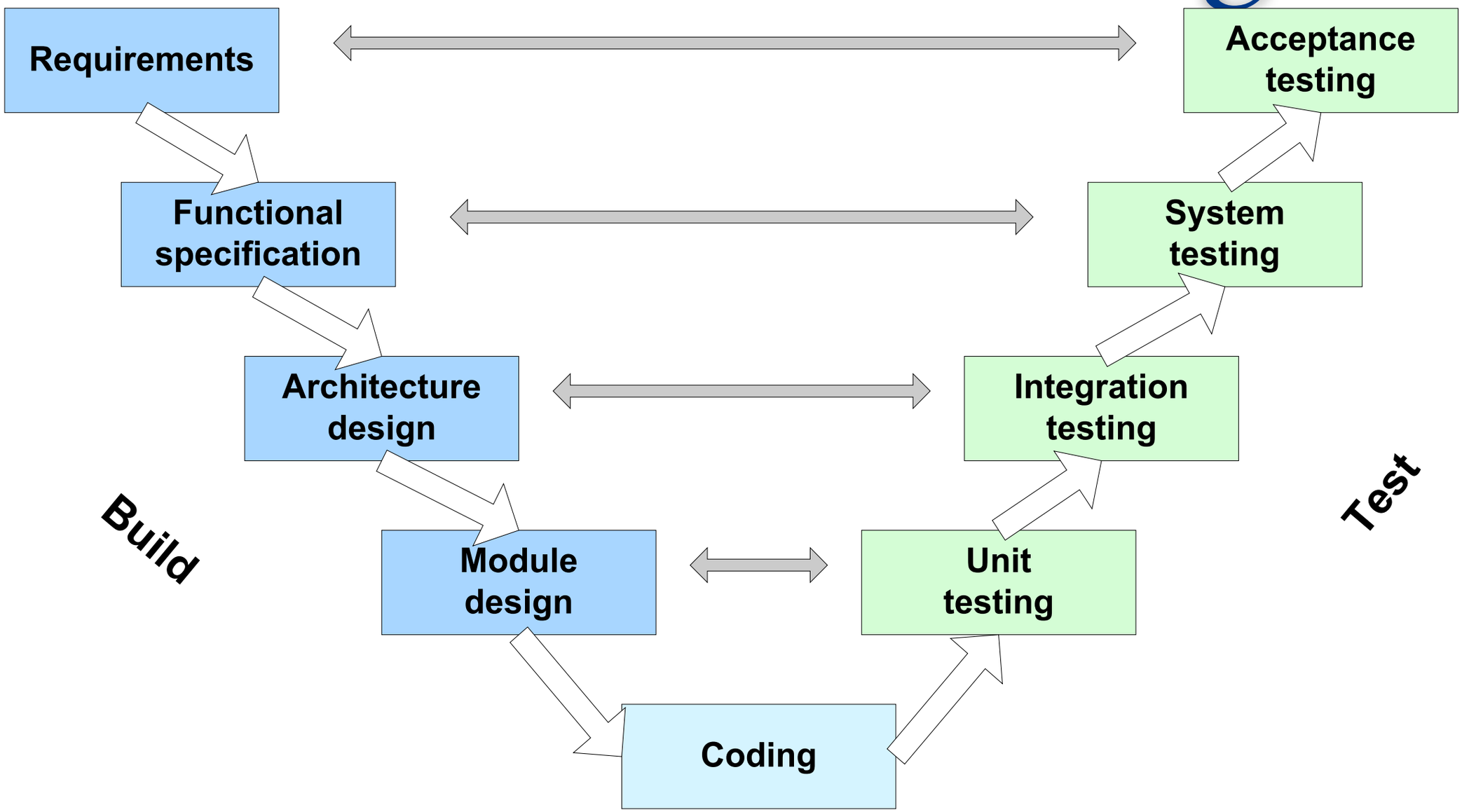
Testing in Agile Software Development

T–76.5613, Software Testing and Quality Assurance

Slides by Juha Itkonen
Lecture delivered by Casper Lassenius

4.10.2006

V-model of testing



Benefits of the V-model

- Intuitive and easy to explain
  - Even to people who have never heard of software development life-cycles
  - Matches familiar waterfall model
- Quite adaptable to various situations
  - If the team is flexible and understands the inherent limitations of the model
- Makes a good model for training people
  - Simple and easy to understand
  - Shows how testing is related to other phases of the development project
- Beats the code-and-fix approach on any larger project
  - More than a dozen or so people

Flaws in the V-model

- Document driven
  - Relies on the existence, accuracy, completeness, and timeliness of development documentation
  - Asserts a test is designed from a single document, without being modified by later or earlier documents
- Communicates change poorly
  - Ignores the fact that, in practice, software is developed in a series of handoffs
  - Asserts that tests derived from a single document are all executed together
- Does not fit into modern iterative development processes

(Brian Marick 2000)

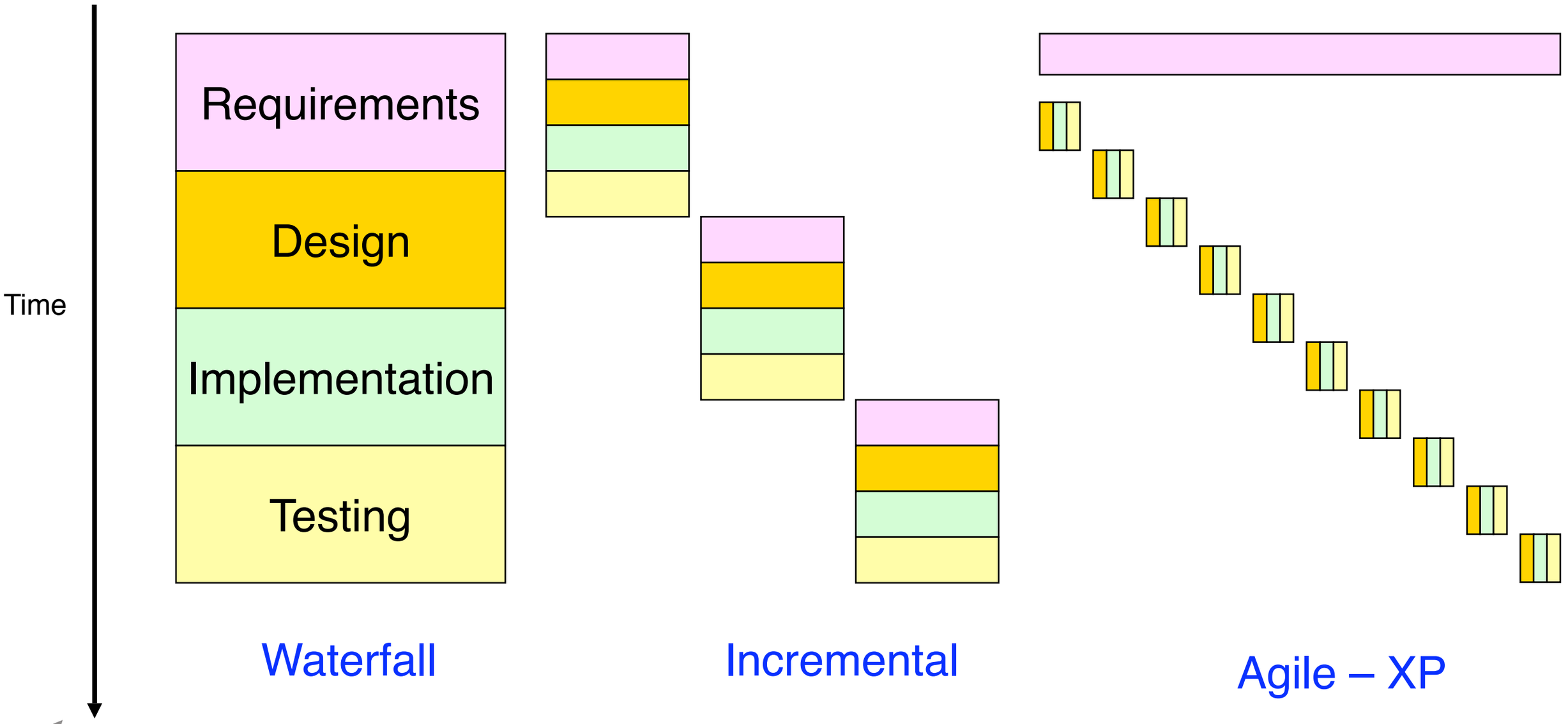
Agile Software Development Manifesto

- In agile development we value:
  - **Individuals and interactions** over processes and tools
  - **Working software** over comprehensive documentation
  - **Customer collaboration** over contract negotiation
  - **Responding to change** over following a plan
- + 12 more detailed principles

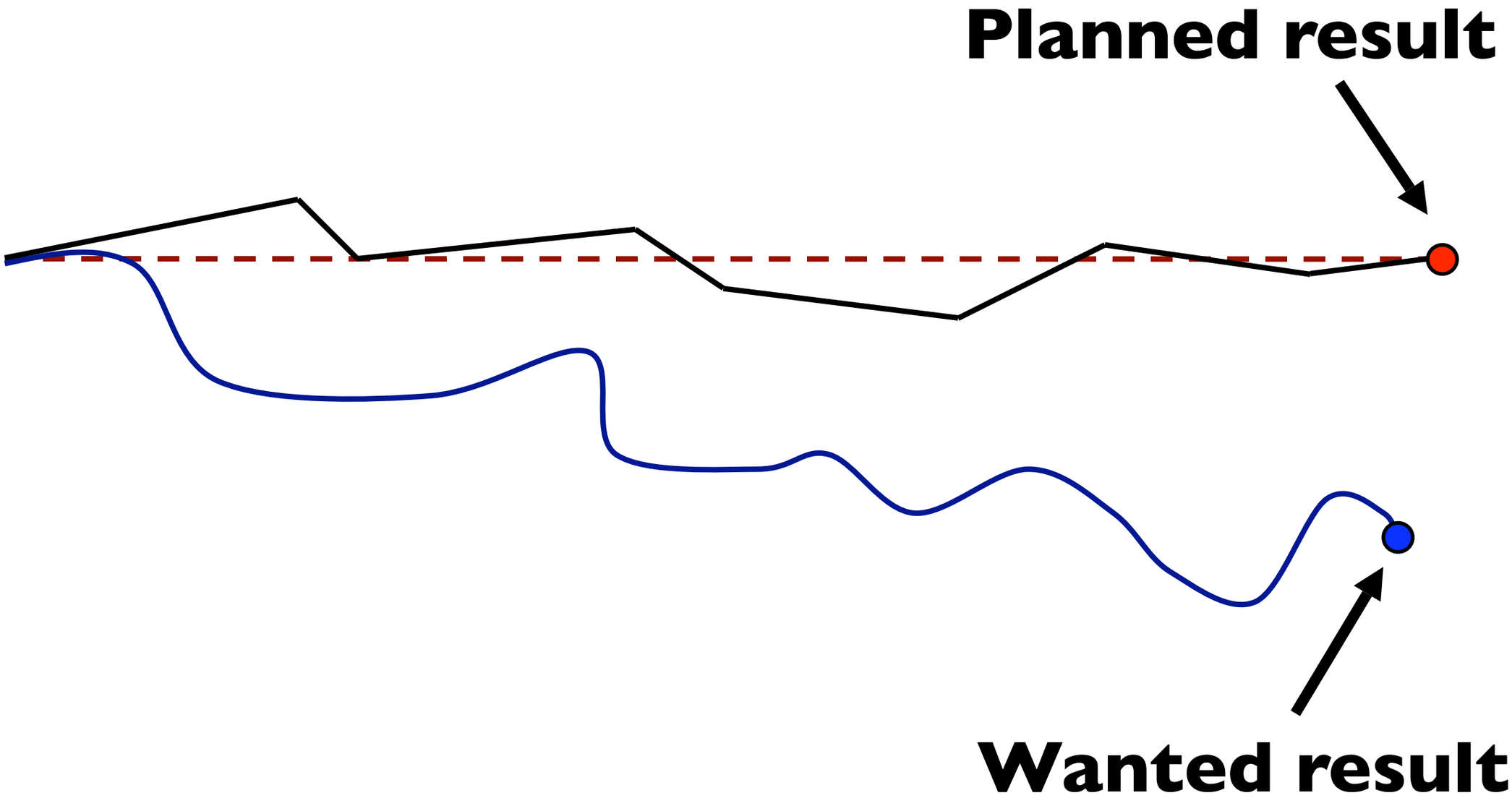
How do these values affect testing?

http://www.agilemanifesto.org/

Agile is Tight Cycles and Small Increments



Plan-Driven vs. Agile



Agile Methods

- Many agile methodologies
  - eXtreme Programming, Scrum, Crystal, FDD, DSDM, ”pragmatic programming”, ...
- Package existing good software engineering practices
- Uses feedback to make constant adjustments in a highly collaborative environment
- Focus on delivering business value
  - Customer/user involvement paramount
- Are not ad-hoc, code-and-fix processes
- Can’t be used in all projects!
  - Suitable for extremely turbulent environments
  - See: Boehm, B. & R. Turner, ”Using Risk to Balance agile and Plan-Driven Methods”, IEEE Computer, Vol. 36, No. 6, June 2003.

Agile Software Development – What About Testing?

- Agility in software development
  - Working software is most important deliverable
  - Responding to change – not resisting change
- Tight development and release cycle
  - Iterative and incremental process
  - Fast feedback, frequent adjustments
- Efficient collaboration and communication
  - Within the team and with the customer
- Agility from the testing viewpoint
  - Places many challenges
  - Testing must also be agile in agile development
  - You can learn a lot from the agile methods

Organizing testing (waterfall vs. agile)

Waterfall model

Agile models (XP)

Idea:
Testing in collaboration

Some Agile Principles

- Satisfy the customer through early and continuous delivery of valuable software.
- Working software is the primary measure of progress.
- Deliver working software frequently, from a couple of weeks to a couple of months.
- Welcome changing requirements, even late in development. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Business people and developers must work together daily throughout the project.
- Simplicity—the art of maximizing the amount of work not done--is essential.
- Build projects around motivated individuals.
  Give them the environment and support they need,
  and trust them to get the job done.

Challenges that agile principles place on traditional testing

Agile Principle	Challenge
Frequent deliveries of valuable software	• Short time for testing in each cycle • Testing cannot exceed the deadline
Responding to change even late in the development	• Testing cannot be based on completed specifications
Relying on face-to-face communication	• Getting developers and business people actively involved in testing
Working software is the primary measure of progress	• Quality information is required early and frequently throughout development
Simplicity is essential	• Testing practices get easily dropped for simplicity's sake

J. Itkonen, K. Rautiainen and C. Lassenius. "Towards Understanding QA in Agile Software Development", ICAM2005.

Contradictions with traditional testing principles

Testing principle	Contradicting practices in agile methods
Independency of testing	• Developers write tests for their own code • The tester is one of the developers or a rotating role in the development team
Testing requires specific skills	• Developers do the testing as part of the development • The customer has a very important and collaborative role and a lot of responsibility for the resulting quality
Oracle problem	• Relying on automated tests to reveal defects
Destructive attitude	• Developers concentrate on constructive QA practices, i.e., building quality into the product and showing that features work
Evaluating achieved quality	• Confidence in quality through tracking conformance to a set of good practices

J. Itkonen, K. Rautiainen and C. Lassenius. "Towards Understanding QA in Agile Software Development", ICAM2005.

Agile methodologies and testing

- Some define strict disciplined testing practices
  - XP, Synch-and-stabilize, ...
- Many methodologies have an idea of continuous testing and automated unit testing approach.
  - XP, Crystal, DSDM, ...
- Some do not say much about testing approach
  - Scrum
  - E.g. FDD: "... processes used for testing are not the main process issues with which the organisations are struggling ... and most organizations already have reasonable testing processes in place"

"Agile Testing" seems to be is a popular buzzword,
but it is not clearly defined what does it actually mean…

Two views of agile testing

- **eXtreme Testing**
  - Automated unit testing
    - Developers write tests
    - Test first development
    - Daily builds with unit tests always 100% pass
  - Functional testing
    - Customer-owned
    - Comprehensive
    - Repeatable
    - Automatic
    - Timely
    - Public

Focus on automated verification – enabling agile software development

- **Exploratory Testing**
  - Exploratory testing without pre-specified test cases
  - Utilizes professional testers' skills and experience
  - Optimized to find bugs
  - Minimizing time spent on documentation
  - Continually adjusting plans, re-focusing on the most promising risk areas
  - Following hunches
  - Freedom, flexibility and fun for testers

Focus on manual validation – making testing activities agile

How Do Agile Quality Practices Work?

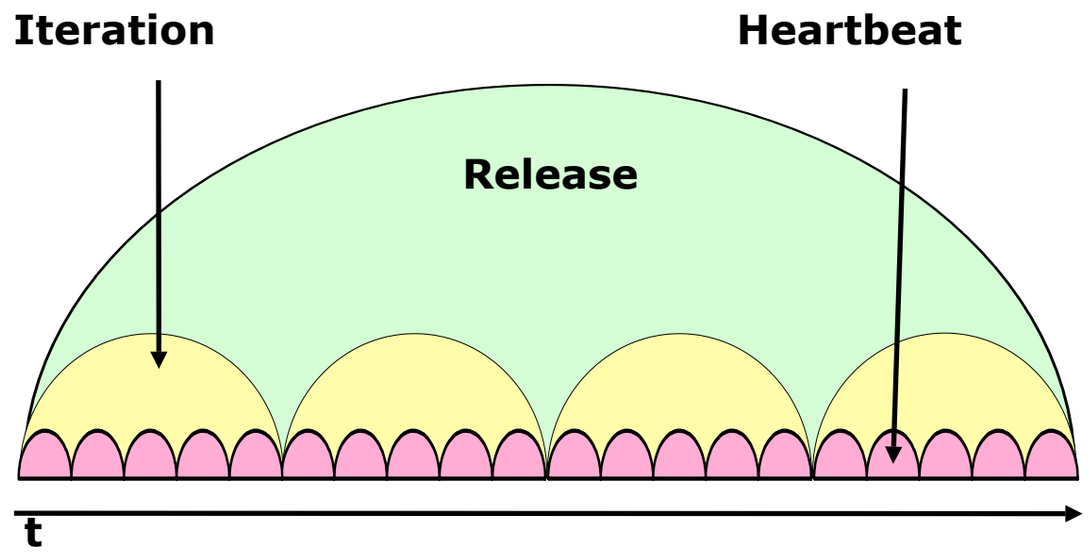
Quality Assurance in Agile Development

- **Plan-driven** quality assurance **does not work** in an agile context
  - There is no V-model and waterfall
  - The role of documentation and specifications is secondary
  - The rhythm of development is fast and tight
  - Roles and responsibilities are assigned differently

- **Cornerstones** of agile quality assurance
  - Constant and tight rhythm
  - Collaboration and communication
  - Rigorous low level (quality) practices
  - Test-Driven Software Engineering

Rhythm

- Continuous unit-level integration and test cycle
- Short incremental release cycles
- Completing features in short cycles and small increments
  - Complete, integrated, tested
- Building quality and tracking the quality level in short cycles



Challenges
- Short time for testing in each cycle
- Testing must be time-boxed, too
- Testing cannot be designed beforehand
- Quality information is required early and frequently
- Testing practices get easily dropped

Rhythm – viewpoint to testing

- Feedback between development and testing is faster
- Feedback from customers and users is received early and often
- Defects and issues are found earlier
- Feel the rhythm—tackle tasks before they bunch up
  - Testing features in small increments is easier
  - Less new defects to hinder testing
  - Less open defects to distract development
- Real quality level and real progress better visible
- Easier to estimate and plan testing phase (or tasks)
- Risks of testing and deployment are smaller

Communication

- Communication in agile development does not work in the traditional way
  - Traditionally others communicate to testers with requirements and specification documents and testers respond with defect reports and test reports
- *"So we can let free of the illusion that documents will save us. We can view them as they are: interesting texts, partly fictional, often useful."*
  - Brian Marick

Communication and collaboration – viewpoint to testing

- The tester's role is to help the development team to create software of good enough quality
- Testers bring expertise and skills as well as the testers destructive viewpoint to the team
- By working together with developers and the customer the reliance on comprehensive documentation is not needed

Challenges
- Tests must not prevent change
- Getting developers and business people actively involved in testing
- Developers write tests for their own code as part of the development
  - Developers concentrate on constructive QA practices
  - The customer has a lot of responsibility for the resulting quality
- Relying on automated tests to reveal defects

Low Level Quality Practices

- Quality practices are tightly integrated into the daily heartbeat rhythm of the development
  - Testing each individual feature as part of its development
  - Building quality using constructive practices
  - Quality is part of development—not delayed any further
  - E.g. automated unit tests, pair-programming, continuous integrations, refactoring, ...
- The goal is to achieve a high quality level before the code leaves the developer's desk

Some Practices of an Agile Developer

- Integrate early integrate often
- Keep your project releasable at all times
- Automate acceptance testing
- Use automated unit tests
- Use it before you build it – TDD
- Write code in short edit/build/test cycles
- Emphasize collective ownership of code
- Keep a solutions log
- Keep it simple
- Don't fall for the quick hack
- Write code to be clear not clever
- Warnings are really errors
- Provide useful error messages
- Be a mentor
- Share code only when it's ready
- Review code
- Keep others informed of the status of your work
- ...

V. Subramaniam & A. Hunt, "*Practices of an Agile Developer*", The Pragmatic Bookshelf, 2006.

Challenges
- Short time for testing in each cycle
- Testing must be time-boxed, too
- Tests must not prevent change
- Testing practices get easily dropped

Low level quality practices – viewpoint to testing

- Technical quality of the software is high
- Less trivial defects—easier to test hard issues
- Boring, routine, checking procedures are automated
- Testers and developers have a common goal of creating high quality software
  - Testers help developers ↔ developers help testers
- Testers can help developers
  - Pair tester, who helps a developer to test each functionality
  - Tester writes tests for the current functionality under development
    - Tests help the developer in implementation and provide fast feedback already during implementation

Test-Driven Software Engineering

- The agile test-driven approach can be applied also on the level of customer requirements and acceptance testing
- Tests can help programmers, testers, and business people (customer) communicate
- Tests can by used to guide development work
- Tests act as the true measure of real progress

The Usual Approach

- Programmers implement brilliant and fancy features and produce code based on the requirement and specification documentation

- Testers in their independent team design clever and tricky tests to reveal defects in the code and errors that programmers might have made

- When the programmers have the code completed, testers use their ingenious tests to catch the defects and reveal the errors programmers have done

Test-Driven Approach

- Testers, programmers, and the customer together design good tests based on the customer requirements
  - Tests that describe how, exactly, the features and core business logic should work
  - Tests that will reveal any major problems in the implementation (risks)
  - Tests that document the important details and potential problem areas and complicated features beforehand
- Tests guide the programmers in implementation work
  - What exactly must be done; how special cases should behave; when the feature is done
  - What are the important aspects of this feature; which issues to pay attention to
- Tracking the amount of passing tests communicates the progress of development
  - As agile values and principles state
  - http://www.xprogramming.com/xpmag/jatRtsMetric.htm
- Test-Driven Software Engineering requires test automation

Collaboration in Testing and Automation

- Business (customer) provides requirements in the form of tests
  - Examples that describe the customer requirements and core business logic
  - Concrete and detailed specificaitons
- Testers help creating tests
  - Create the actual tests based on the customer's examples
  - Extend tests by designing more tests from the tester's viewpoint based on their expertise
- Developers create the technical framework
  - Test automation framework
  - Way of defining automated tests on the level of customer requirements
  - Way of integrating tests into the implementation
  - See, for example, Framework for Integrated Test (FIT)
    - http://fit.c2.com/

New Roles for Tests

- In Test-Driven Software Engineering tests have a clear role in the **early** requirements and design **phases** of development
- **Tests help** developers, testers, and customer to **communicate**
  - Common vocabulary & understanding
- **Tests document** and concretize the **details** of requirements
  - Concrete examples of correct functions and logic
  - Easy to document special cases
  - Easy to extend and update during the development
- **Tests guide** the **implementation** work
  - Fast feedback to developers when automated
- **Tests provide** an honest and agile **metric** of true **progress** in terms of working software

Agile Software Testing in a Large-Scale Project (1/4)

- Experiences from a real large scale agile project to develop business critical enterprise information system for Israeli Air Force
- Using an XP-based method
- Two week iterations
  - Fully tested features
  - Plus regression testing and fixing all known defects

Talby D., Keren A., Hazzan O. and Dubinsky Y. "Agile Software Testing in a Large-Scale Project", IEEE Software, July/August, 2006.

Agile Software Testing in a Large-Scale Project (2/4)

- In agile projects everyone writes tests
  - Make testing
    - part of each team member's work
    - a key measure of both team and personal productivity
  - Testers worked better in collaboration with the development team
    - Working independently led to many minor deviations from specifications, but few actual bugs
    - Working together reduced number of false defect reports
- Having everyone testing eliminates the single tester bottleneck
- Developers awareness improved when they were involved in writing tests
  - Less defects, better code
  - Design for testability
- Measuring progress
  - Product size metric: Number of regression test steps run in each iteration
- Untested work equals no work

Talby D., Keren A., Hazzan O. and Dubinsky Y. "Agile Software Testing in a Large-Scale Project", IEEE Software, July/August, 2006.

Agile Software Testing in a Large-Scale Project (3/4)

- Agile defect management is simple
  - Anyone can open a defect
  - Anyone can close a defect after fixing and running tests
  - Anyone who finds a defect assigns it to someone for fixing
  - Developers can reassign defects by themselves
    - Reduces overhead from managers
- Fix every defect as soon as possible
  - Defects require less time to fix
  - Working on clean and stable code makes development faster
  - Avoids overhead of prioritising and planning defect fixes
  - Avoids unpleasant customer negotiations over which defects to fix
- No concept of defect severity; the only consideration is whether to fix it or not
  - If not, the defect is not opened
  - If yes, the defect is fixed when it's cheapest to do – usually right a way

Talby D., Keren A., Hazzan O. and Dubinsky Y. "Agile Software Testing in a Large-Scale Project", IEEE Software, July/August, 2006.

Agile Software Testing in a Large-Scale Project (4/4)

- Agile development completely redefines quality assurance work- from formal roles to day-to-day activities
  - Agile practices require full integration of testing and development
  - Fully adopting agile quality practices is harder than adopting agile programmer oriented practices, such as pp and test-first programming
- The project team cut by an order of magnitude
  - The time required to fix defects
  - Defect longevity
  - Defect management overhead
- Even on such a large scale project the team achieved full regression testing at each iteration and developer testing
- It also resolved all defects over a significant time period that included both personnel changes and team

Talby D., Keren A., Hazzan O. and Dubinsky Y. "Agile Software Testing in a Large-Scale Project", IEEE Software, July/August, 2006.

References

Agile software development in general

http://www.agilealliance.org/

http://www.martinfowler.com/articles/newMethodology.html

Beck, K., Extreme Programming Explained, Boston, Addison-Wesley, 2000.

Beck, K., Test-Driven Development by Example, Addison-Wesley, 2003.

Boehm, B. & R. Turner, "Using Risk to Balance agile and Plan-Driven Methods", IEEE Computer, Vol. 36, No. 6, June 2003.

Agile testing

http://www.testing.com/agile/index.html

Crispin L. and House T., Testing Extreme Programming, Addison-Wesley, 2003.

Ron Jeffries: A Metric Leading to Agility

http://www.xprogramming.com/xpmag/jatRtsMetric.htm

Framework for Integrated Test (FIT)

http://fit.c2.com/

Subramaniam V. and Hunt A., Practices of an Agile Developer, The Pragmatic Bookshelf, 2006.

Talby D., Keren A., Hazzan O. and Dubinsky Y., "Agile Software Testing in a Large-Scale Project", IEEE Software, July/August, 2006.

Itkonen J., Rautiainen K. and Lassenius C., "Towards Understanding Quality Assurance in Agile Software Development", in Proceedings of the International Conference on Agility, pp. 201-207, 2005.

http://www.soberit.hut.fi/jitkonen/Publications/Itkonen_Rautiainen_Lassenius_2005_ICAM.pdf