Introduction to Software Design using Agile Methodologies

Introduction to Software Design using Agile Methodologies

- Introduction to Agile Methodologies
- Software Design and Architecture using Agile Methodologies
- Software Architecture Views?
- Extreme Programming
- Test Driven Development (TDD)
- How to Use TDD
- References

Introduction to Agile Methodologies

- Agile development is a philosophy or way of thinking about software development described in the Agile Manifesto (a collection of 4 values and 12 principles).

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to 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

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

©2001, the above authors
This declaration may be freely copied in any form, but only in its entirety through this notice.

Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

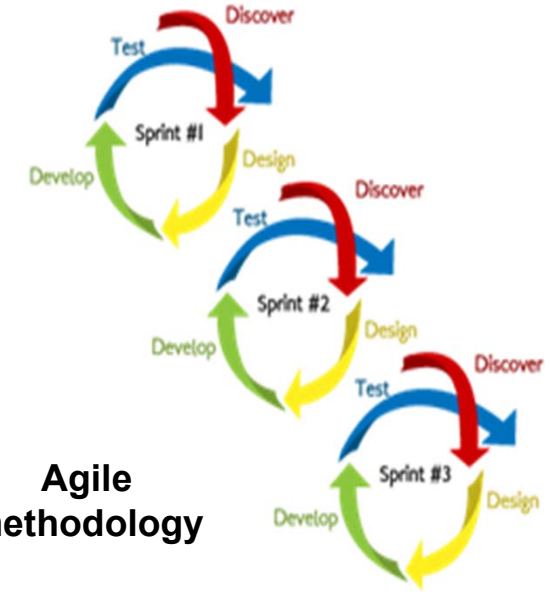
Continuous attention to technical excellence and good design enhances agility.

Simplicity, the art of maximizing the amount of work not done, is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Software Design and Architecture using Agile Methodologies

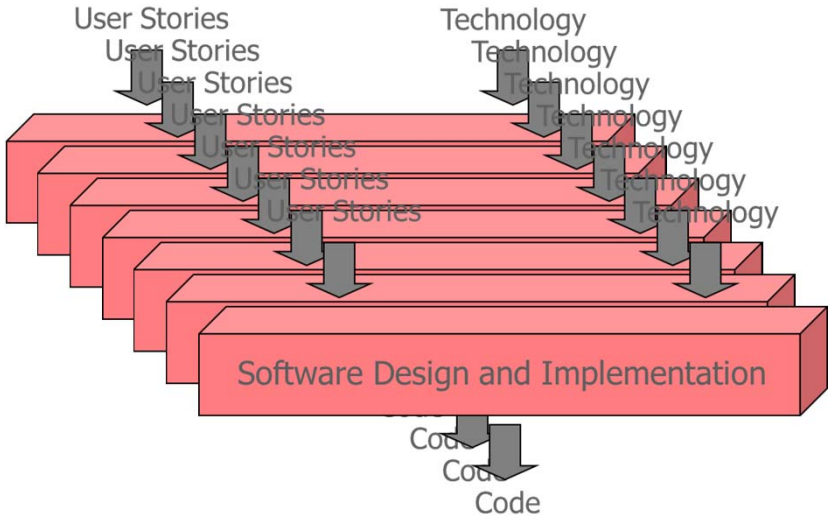


Agile methodology

- Software design and Architecture starts when the discover phase is not finished.
- Software Architecture is informal and incremental.
- Software design is focused on completing stories in short iterations.
- Software design is a light process with a light documentation.
- Software design requires architects/ developers.
- Architects/developers have frequent interactions with business people.

Software Design and Architecture using Agile Methodologies

- Software design in Agile Methodologies (inputs and outputs)

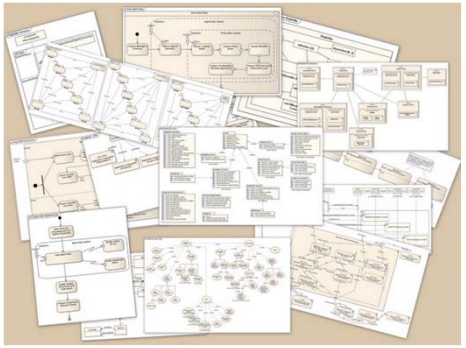


Software Design and Architecture using Agile Methodologies

- There is not an agreement about how to define the Architecture in Agile Methodologies. Two proposals:
  - Initial version of the architecture before designing/developing
    - Big teams
    - Big projects
    - Critical software
  - Incremental Architecture: Architecture emerging throughout the course of design/development
    - Small teams (5 or 6 developers)
    - Small projects
    - Not critical software

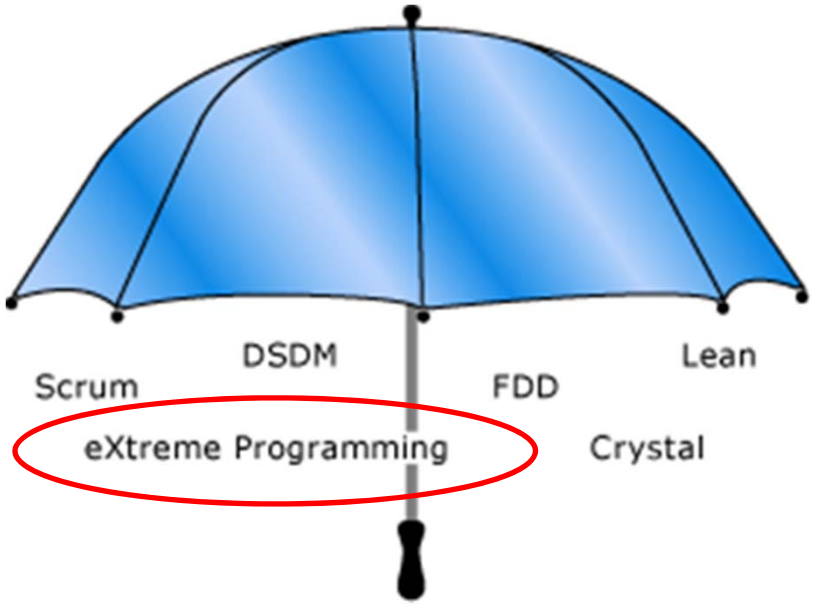
Software Architecture Views?

- Again, there is not an agreement about what documentation and diagrams have to be produced. Two proposals:
  - Don't do diagrams
  - Use diagrams (selecting important things and neglecting the less important) to gain an effective communication and for documentation.
    - Don't draw every class (only the important ones)
    - Don't draw sequence diagrams for all scenarios
    - Put diagrams where everyone can easily see them (for instance, posted in a wall). Encourage people to edit the wall copy with a pen for single changes. If people don't use them, throw them away.



Extreme Programming

- XP is an agile method that supports the agile philosophy.
- Unlike other methodologies, like Scrum, XP prescribes engineering practices.
- Other methodologies may use XP engineering practices.



Extreme Programming

- XP identify 12 rules (or practices) to follow during the development process:

  1. User stories (planning): User stories (individual features) are brief and are used for cost estimating and project management.

  2. Small releases (building blocks): Deliver the application in a series of small, frequently updated versions. As each new requirement is added, complete the system and re-release.

  3. Metaphor (standardized naming schemes): Adhering to a set of standards for items (variable names, class names, …).

  4. Collective ownership: Code is reviewed and updated by everyone on the team, allowing for a more collaborative effort.

  5. Coding standard: Same styles and formats for codification. Rapid code sharing.

  6. Simple design: Easy design that works. A correct design is one that runs all unit and functional tests, meets the business value, and does it only once for each function.

Extreme Programming

7. Refactoring: Continually work to adjust and improve the code. Code constantly revised without duplicating code.

8. Testing: Each building block (small release) must be thoroughly tested prior to release. Write the tests first and develop the code to meet the requirements of the test.

9. Pair programming: Code developed by two programmers who work together at a single machine. Higher quality code at the same or less cost.

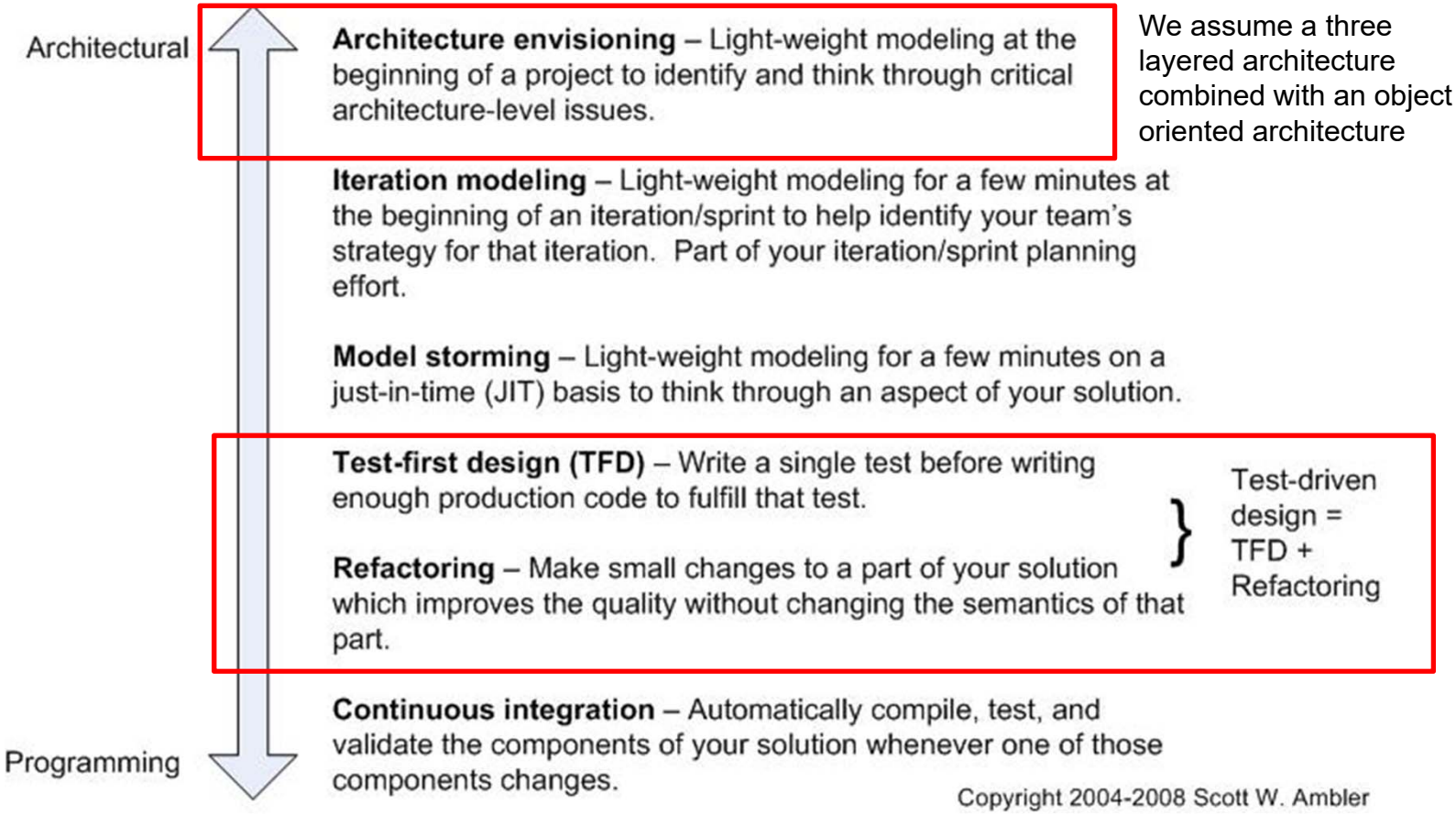
10. Continuous integration: Software builds are completed several times a day keeping the application up to date.

11. 40-hour workweek: Tired or sleep-deprived developers make more mistakes resulting in lower-quality code.

12. On-site customer: Customer, available at all times, as an integral part of the development effort.

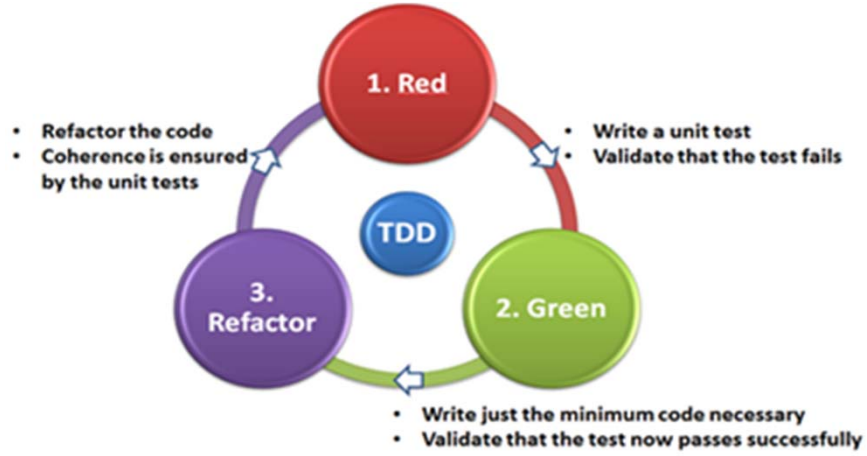
Extreme Programming

- There is a range of agile design practices from high-level architectural practices to low-level programming practices.



Test Driven Development

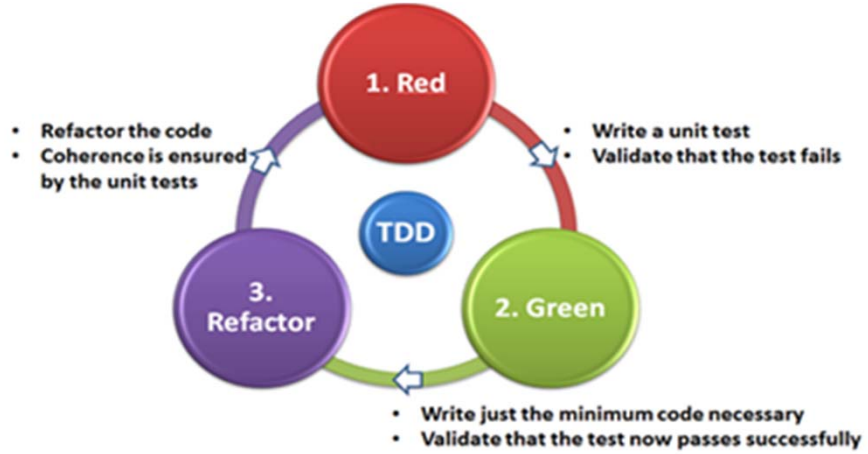
- TDD embraces the rules identified by the XP.
- TDD is an approach to design and development in which three activities are tightly interwoven: coding, testing (in the form of writing unit tests) and design (in the form of refactoring).
- It takes a few months of steady use to overcome the learning curve. Be patient!!!



How to Use TDD

- TDD operates in a very short cycle that repeats over and over again. Every few minutes the code has been tested, designed and coded.

- With experience, each cycle will take fewer than five minutes.



References

- *The art of Agile Development*
  J. Shore and S. Warden
  O' Reilly, 2007

- *Extreme Programming: Do these 12 practices make perfect?*
  http://www.techrepublic.com/article/extreme-programming-do-these-12-practices-make-perfect/

- *Agile Development & Software Architecture*
  Nanette Brown
  http://www.sei.cmu.edu/webinars/view_webinar.cfm?webinarid=18708

- *Agile Modeling. Effective Practices for Modeling and Documentation*
  http://agilemodeling.com/

- *The Rules of Extreme Programming*
  http://www.extremeprogramming.org/rules.html

- *Test Driven Development: By Example*
  Kent Beck
  Addison-Wesley