Ingegneria del Software

Corso di Laurea in Informatica per in Management

Agile software development

Davide Rossi

Dipartimento di Informatica – Scienze e Ingegneria

Università di Bologna

The problem

Efficiency: too much effort spent in overhead (all the activities that are not directly related to the construction of the software system). Improving the process in order to improve the product does not work as well as in other engineering branches.

The problem

The effort spent to guarantee adherence to the plan: contract negotiation, comprehensive (process-related) documentation, risk assessment, etc., could be wasted when the plan is not crystal clear from the beginning. Which is usually the case in software development.

We don't just need our software to be “flexible”, we need our whole development system to be able to adapt to change.

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.

Manifesto: who

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.

Agile software development

Agile software development is NOT:

- a process
- a design method

Agile software development is:

- a collection of **practices** guided by a set of **principles** inspired by **values**.

The 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.

The principles

- 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 principles

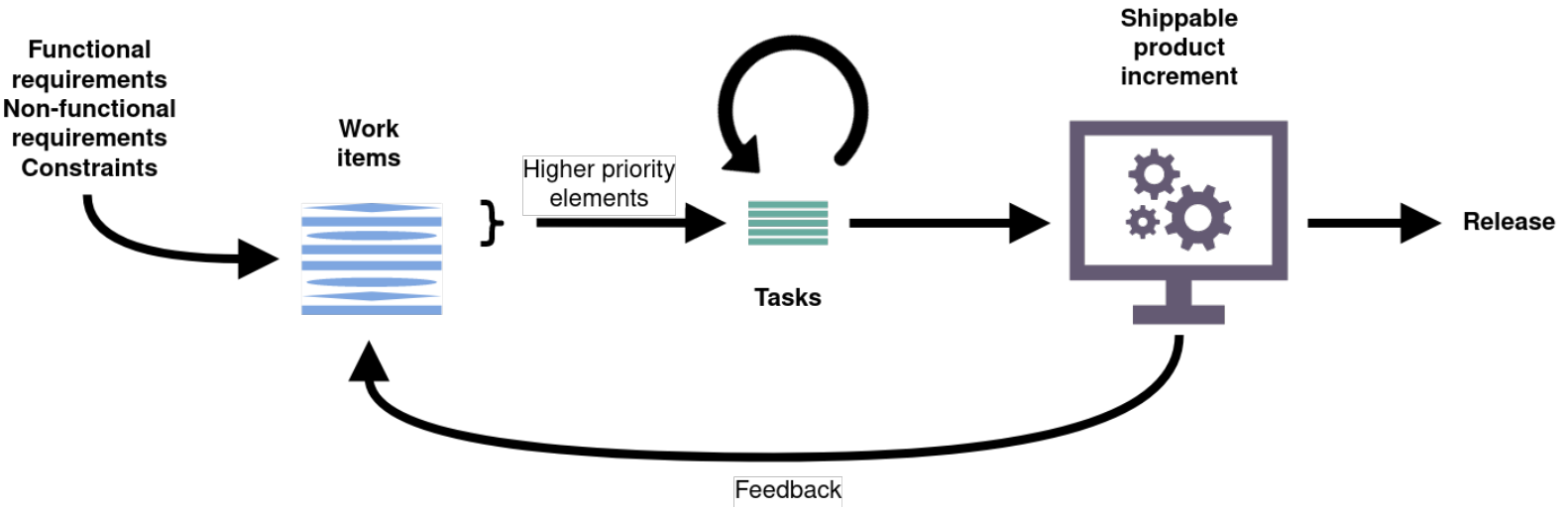
- The sponsors, developers, and users should be able to maintain a **constant pace** indefinitely.
- Continuous attention to **technical excellence and good design** enhances agility.
- **Simplicity** – the art of maximizing the amount of work not done – is essential.
- The best architectures, requirements, 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.

Agile methods

In the last few years a growing number of agile-inspired methods have been proposed:

- Agile Modeling
- Agile Unified Process
- Crystal Clear
- Extreme Programming
- Scrum
- ... and others ...

Agile lifecycle



The practices

- Refactoring
- Small release cycles
- Continuous integration
- Coding standard
- Collective ownership
- Planning game
- Whole team
- Daily meetings
- Test-Driven Design
- Code and design reviews
- Pair programming
- Document late
- Use of design patterns
- ...

See http://guide.agilealliance.org/

Code review

By code review we usually mean the practice by which new/revised code must pass a review before being committed.

Our study reveals that while finding defects remains the main motivation for review, reviews are less about defects than expected and instead provide additional benefits such as knowledge transfer, increased team awareness, and creation of alternative solutions to problems.

[Bacchelli, Bird]

Pair programming: extreme code review

Pair programming consists of two programmers sharing a single workstation (one screen, keyboard and mouse among the pair). The programmer at the keyboard is usually called the "driver", the other, also actively involved in the programming task but focusing more on overall direction is the "navigator"; it is expected that the programmers swap roles every few minutes or so.

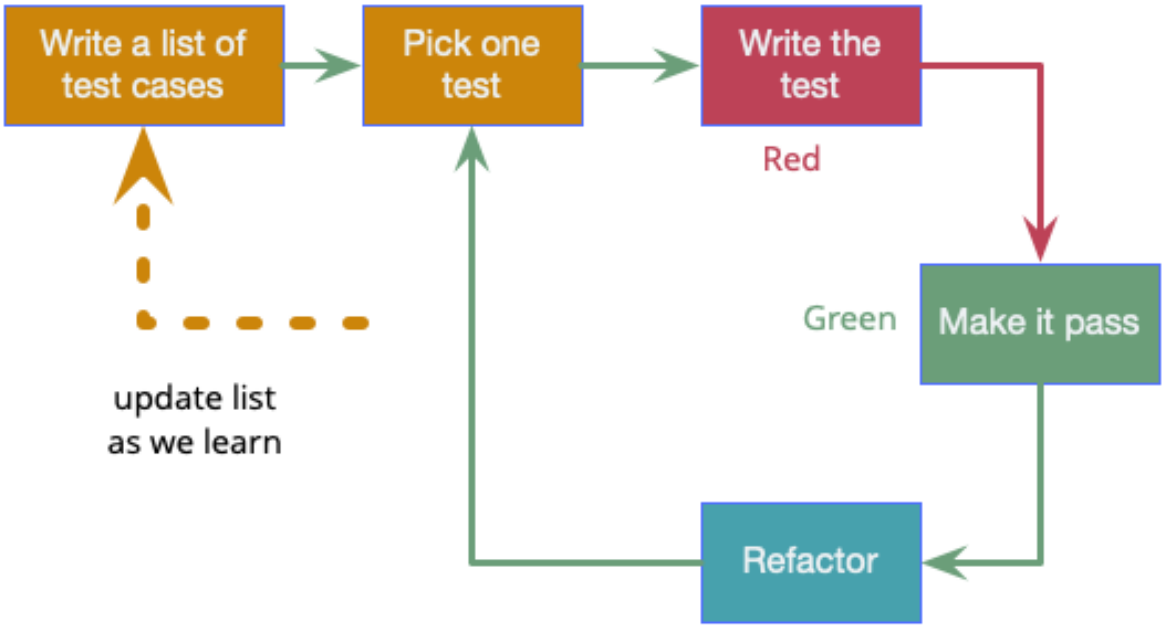
Test-Driven Design

Is a style of programming in which three activities are tightly interwoven: coding, testing (in the form of writing unit tests) and design (in the form of refactoring).

It can be described by the following set of rules:

- write a single unit test describing an aspect of the program
- run the test, which should fail because the program lacks that feature
- write just enough code, the simplest possible, to make the test pass
- refactor the code until it conforms to the simplicity criteria
- repeat, accumulating unit tests over time

TDD



https://martinfowler.com/bliki/TestDrivenDevelopment.html

User stories

- Several agile software development methods adopt user stories to represent requirements.
- User stories provide processable functional description of the system (from the user's viewpoint, as the name suggests) that **drive implementation** and are processed into **acceptance tests** that **validate implementation**.

User stories

- User stories are concise sentences, written in the **domain language**, capturing the expectations of the users.
- User stories are not use cases.

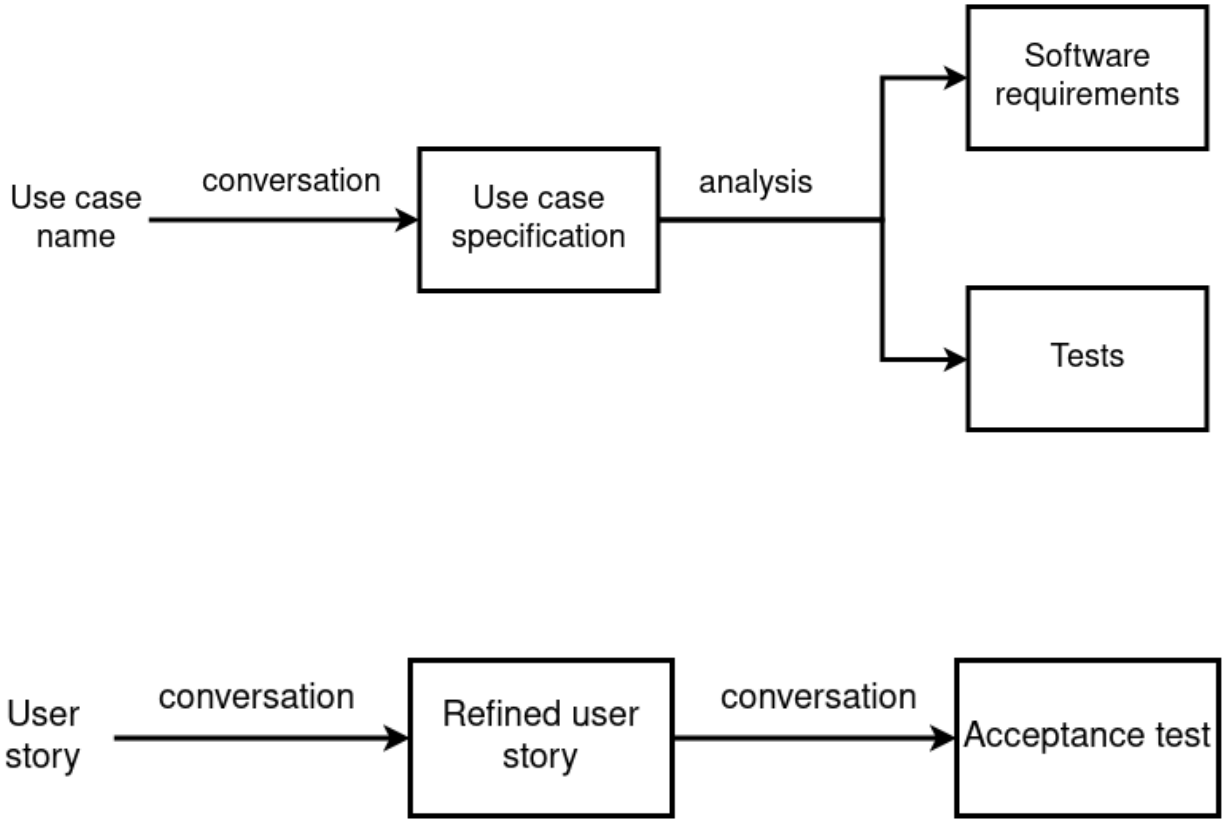
User stories templates

- Most usual format:
  as a <role>, I want <goal> so that <benefit>.
  As an employee I want to purchase a parking pass so that I can drive to work.
- "Jobs stories", from jobs to be done (JTBD):

  when <event>, I want to <goal> so that <benefit>.

  When employees drive to work, I want them to be able to purchase a parking pass to access the company's parking so that they can be at their desk on time.

Processing: stories vs UCs



Acceptance tests

- Are integral parts of user stories.
- Often follow the Given-When-Then template.

Given my bank account is in credit, and I made no withdrawals recently,
When I attempt to withdraw an amount less than my card’s limit,
Then the withdrawal should complete without errors or warnings

Example

- **Story**: **As a** cyclist, **I want to** follow my friends' cycling routes **so that** I can join them on rides
- **Test**: **Given** that the cyclist wants to ride with friends, **when** they check the map view, **then** the routes of other cyclists in their social network should be visible.

Stories are volatile

Stories are not meant to survive their processing.

What persists are the associated tests.

Stories, epics, themes

- Stories describing high level features are usually collected early but are underspecified and are refined as the project progresses
- **Epics** are large user stories; they usually need more than one iteration to be fully developed
- Epics are split in smaller, more detailed stories when they approach the development stage
- **Themes** are collection of related stories

INVEST

An accepted set of criteria, or checklist, to assess the quality of a user story:

- Independent
- Negotiable
- Valuable
- Estimable
- Small
- Testable

Independent

- Stories should not depend on each other as much as possible.
  - As a customer I want to buy a good with my AmEx credit card
  - As a customer I want to buy a good with my other credit card

Negotiable

- User stories are not contracts.
- User stories are the result of a negotiation and are prone to be re-negotiated at any point in time.
- Often a negotiation phase takes place at estimation time.

Valuable

- Stories must provide a value.
- Note: not always for the end user. A perspective should be taken, and the story written from that perspective.

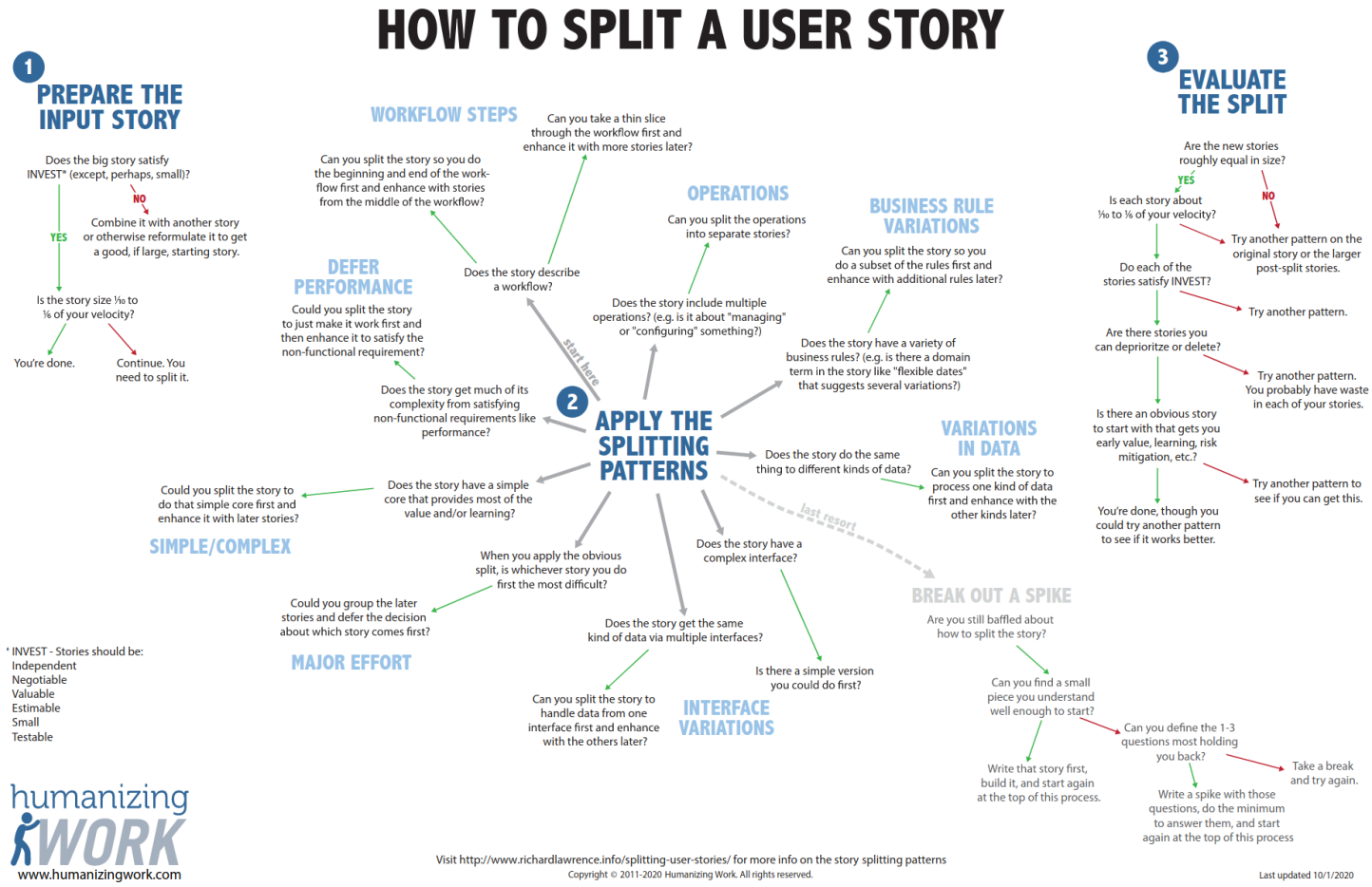
Estimable

- The team should be able to estimate the level of complexity and the amount of work need for its processing.
- A high level of uncertainty is a good indicator that the scope of the story is too wide or not focused enough.

Small

- A user story should be processed in an iteration. By the end of the iteration, they have to be considered **done**.
- Iterations in agile methods are usually short (1-2 weeks) and we have more than one developer to be kept busy.

What if the story is not short?



Testable

- In most agile methods, a story is done only when the corresponding features pass the acceptance tests.
- No tests → no stories done.

Agile and evolution

- Development is just a part of software lifecycle, evolution is just as important
- In agile approaches, knowledge is shared implicitly and very little documentation is produced: what happens when maintenance is handed over to a different team?
- That should be *document later* but often times it becomes *document never*.