Agility in Software Development

Dr J Paul Gibson, Dept. INF, TSP, Evry, France

Ashleigh Brilliant (https://www.ashleighbrilliant.com)

http://blog.dilbert.com
Introduction To Agile

A Traditional Project
Build a bridge in time for the September conference

1. Plan

2. Build

3. Test

An Agile Project

User stories:
As an attendee, I want to access the program so that I can attend the conference.

AGILE

RISK

VALUE

What's the problem with a traditional project?

Agile projects adapt to change

https://www.youtube.com/watch?v=vRne1NiCIHQ

(Black Pepper Software Limited)
Structure of the Presentation

1. Software Process and the Software Life Cycle

2. History of Agile: More or less a process?

3. Agile Fundamentals

4. Comparing Agile Methods

5. Agile Resources
How important is a software development process?

"... the quality of the people on a project, and their organization and management, are much more important factors in the success than are the tools they use or the technical approaches they take."

ISO 9000 - Quality management
http://www.iso.org/iso/iso_9000

ISO/IEC JTC 1/SC 7 - Software and systems engineering

ISO/IEC 12207 Systems and software engineering – Software life cycle processes

CMMI maturity model for Software Process Improvement (SPI)
http://cmmiinstitute.com
Why Do Software Projects Fail (Often)?

Most often it is because of:

- A failure to properly manage the risks
- Building the wrong thing

Adopting a good *software process & life cycle* will help address these failure modes.

Adopting a good *software process & life cycle* does not guarantee success.

We can *never* have a completely rational development process.

---

Failure to Properly Manage The Risks

As projects progress, they often seem to lose their way:

• Unrealistic schedules and plans are drawn up
• No-one has the nerve to stand up and acknowledge reality
• Many problems are viewed as ‘a simple programming matter’, even when they are process or architecture concerns
• Project direction is set by the most ‘stubborn’ participants because it is easier for management to let these people have their way.
• Free fall --- No one takes responsibility and everyone waits for the impact.
• Petty empires form ... issues become political
Failure from Building the Wrong Thing

Projects can also lose their way because they go adrift in completely uncharted territory:

- There is no shared vision of the problem being solved.
- The (development) team is clueless as to the final destination.
- No-one takes time to validate what is being built with end-users or domain experts.
- Analysts understand the real requirements, but for a number of political/social reasons, this understanding never reaches the designers/implementers.
- A false air of understanding pervades the project.
- Everyone will be shocked when users reject the delivered software.
- This is known as working in a vacuum.
What is the Software Process?

A process is a systematic approach performed to achieve a specific purpose.

A software process is the set of activities, methods, practices, and transformations used to develop software and associated products that are released with it.

Software Process Capability is the range of expected results that are achievable by following the software process.

Software process performance is the actual result achieved in the development of software by following a software process.

Software Process Maturity is the extent to which a Software Process is defined, managed, controlled, measured and effective.
No process is perfect

Even the most successful projects seem to take longer, involve more effort, and require more *crisis management* than we really believe they ever should. We must never rely on the process pulling a project through. The process can never be completely rational:

- Users typically don’t know what they want
- Users typically can’t express what they want
- Requirements are incomplete and/or change
- Implementation architectures change
- We all bring intellectual/technological baggage to projects
- Systems built by humans are always subject to human error
- Fundamental limits to the amount of complexity which can be handled

What is the Software Life Cycle?

The software life cycle is the collection of phases through which a software product passes from initial conception through to retirement from service.

- Every software product has a life cycle.
- Life cycles used to be typically quite long—some software products have been “alive” for 30 years.
- Life cycles are shortening due to technological advances

Life Cycle Phases - Implicitly or explicitly, all software products go through at least the following phases:

- Requirements—determine customer needs and product constraints
- Design—determine the structure/organisation of the software system
- Coding—write the software
- Testing—exercise the system to find and remove defects
- Maintenance—correct and enhance product after customer deployment
Life Cycle Models and the Software Process

The core of any software project is the coding --- architecture, abstraction, implementation

Life cycle models revolve around this core --- how does the software evolve as the project progresses?

All life-cycle models are based on the simple idea of feedback --- synthesis and analysis are mutually defined and recursively interdependent.

The differences between the life-cycle models lie in the ways in which the feedback is organised, for example:

- Trial and error
- Exploratory Programming
- The Waterfall Model
- Iterative Feedback Model
- Prototyping
- Test-Driven
- Design/Model-Driven
- Agile
Life Cycle Chaos - A Complete Feedback Graph Between Activities

Question: Why not just reduce the number of activities to manage the complexity?
Non-strict Waterfall Model

Although the waterfall model stresses a linear sequence of phases, in fact there is in practice always an enormous amount of iteration back to earlier phases, a point made by the arrows leading back up the waterfall, in the following diagram.

Note: In this variation, feedback is only from testing phase to any previous stage.
Analysis of waterfall method

Strengths:
• Emphasises completion of one phase before moving on
• Emphasises early planning, customer input, and design
• Emphasises testing as an integral part of the life cycle
• Provides quality gates at each life cycle phase

Weaknesses:
• Depends on capturing and freezing requirements early in the life cycle
• Depends on separating requirements from design
• Not politically feasible in some organisations
• Emphasises products rather than processes
Away From Waterfall


“Any form of life cycle is a project management structure imposed on system development. To contend that any life cycle scheme, even with variations, can be applied to all system development is either to fly in the face of reality or to assume a life cycle so rudimentary as to be vacuous.”

"The life cycle concept is simply unsuited to the needs of the 1980’s in developing systems."
Prototyping Models (become popular in the 80’s)

A prototype is a working model of (part of) a final system

Prototyping is becoming more popular all the time, and people often refer to prototypes in the literature.

Unfortunately, a variety of terminology is used, so it is often difficult to tell what is meant when people discuss prototyping.

Note there are different types of prototyping model, with different characteristics, eg:

- Rapid Prototyping
- Evolutionary Prototyping
- Operational Prototyping


R. N. Burns and A. R. Dennis. 1985. Selecting the appropriate application development methodology. *SIGMIS Database* 17, 1 (September 1985), 19-23
Agile: A lightweight alternative to heavy-weight waterfall-like processes?

**MANIFESTO FOR WATERFALL SOFTWARE DEVELOPMENT**

Software development can be equated to any other engineering task. We believe software development projects can be effectively managed by:

- Understanding and **writing specifications** that define how the software will look and what it will do
- Performing in-depth **analysis and design** work before estimating development costs
- Ensuring software developers **follow the specifications**
- **Testing the software after implementation** to make sure it works as specified, and
- **Delivering the finished result** to the user

That is, if the specification is of sufficient detail, then the software will be written such that it will satisfy the customer, will be within budget, and will be delivered on time.

**Claim:**
This approach is not good for adapting to change

**Agile is lightweight.** It combines prototyping and test-based processes: where there is continual development of code and continual validation of customer requirements.
2. History of Agile: More or less a process?

History of Agile: Where Did it Come From?

History of Agile

Agile software development isn’t a set of tools or a single methodology, but a philosophy put to paper in 2001 with an initial 17 signatories.

Agile was a significant departure from the heavyweight document-driven software development methodologies—such as waterfall—in general use at the time.

While the publication of the “Manifesto for Agile Software Development” didn’t start the move to agile methods, which had been going on for some time, it did signal industry acceptance of agile philosophy.

"Many people may think that agile is just another software development process. Although that is true to a degree, there is a lot more to agile than just a process or just a set of practices. Agile (or agility) is more of a mindset - a way of thinking about software development."

2. History of Agile: More or less a process?

**Manifesto for Agile Software Development**

http://agilemanifesto.org/

*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
2. History of Agile: *More or less a process?*

**Kent Beck:**
- Test Driven Development: By Example. 2002 Addison-Wesley.

**Mike Beedle & Ken Schwaber:**

Arie van Bennekum

**Alistair Cockburn:**
- Writing Effective Use-Cases, 2000, Addison-Wesley.
- The Costs and Benefits of Pair Programming, In Extreme programming examined, Giancarlo Succi and Michele Marchesi (Eds.)., 2001, Addison-Wesley Longman Publishing

Ward Cunningham & KB:
* A laboratory for teaching object oriented thinking, SIGPLAN Not. 24, 10 (September 1989)

**Martin Fowler**
2. History of Agile: More or less a process?

James Grenning:
• Launching Extreme Programming at a Process-Intensive Company, 2001, IEEE Softw. 18

Jim Highsmith and Alistair Cockburn:
• Agile Software Development: The Business of Innovation, 2001, Computer 34,

Andrew Hunt

Ron Jeffries and Grigori Melnik:
• TDD--The Art of Fearless Programming, 2007, IEEE Softw. 24, 3

Jon Kern

Brian Marick:
• When Should a Test Be Automated?, 1998.
2. History of Agile: *More or less a process?*

Robert C. Martin:
- *SoapBox - eXtreme Programming Development through Dialog*, IEEE Software 17, 2000

Steve Mellor:

Jeff Sutherland:
- *Agile development: Lessons learned from the first scrum*, Cutter IT Journal, 2004

Dave Thomas:
- *Orwell: a configuration management system for team programming*. In OOPSLA’88.
- *Model driven development: the case for domain oriented programming*. In OOPLSA’03.
- *MDA: Revenge of the Modelers or UML Utopia?*, IEEE Software, 21(3), 2004
2. History of Agile: *More or less a process?*

**Agile: Values, Principles and Methods**
Agile: 12 Principles

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

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

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

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

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

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
2. History of Agile: *More or less a process?*

**Agile: 12 Principles**

7. Working software is the primary measure of progress.

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

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity—-the art of maximizing the amount of work not done--is essential.

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

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.
Fundamentals: Agile vs Waterfall

Waterfall features distinct phases with checkpoints and deliverables, while agile methods have iterations where the output of each agile iteration is working code that can be used to evaluate and respond to changing and evolving user requirements.

Waterfall assumes that it is possible to have good understanding of the requirements from the start. But in software development, stakeholders often don’t know what they want and can’t articulate their requirements. With waterfall, development rarely delivers what the customer wants even if it is what the customer asked for.

With Agile emphasis is placed on the customer and their requirements.
Fundamentals: iterations count

Agile methodologies embrace iterations.

Small teams work together with stakeholders to define quick prototypes, proof of concepts, or other visual means to describe the problem to be solved. The team defines the requirements for the iteration, develops the code, and defines and runs integrated test scripts, and the users verify the results.

Verification occurs much earlier in the development process.
Suitability of Agile Methods

There is little agreement on what types of software projects are best suited for the agile approach.

Many large organizations have difficulty moving from the traditional waterfall method to an agile one.

When Agile is risky:

- Large scale development (>20 developers)
- Distributed development (non-co-located teams)
- *Control-freak* companies
- Unreliable customer/client contact
- Forcing an agile process on a development team
- Inexperienced developers
Some of the most popular Agile methods:

- **Scrum**
- **Lean Development**
- **Extreme programming (XP)**
- Adaptive Software Development (ASD)
- Agile Modeling
- Crystal Methods
- Dynamic System Development Methodology (DSDM)
- **Feature Driven Development**

*Most relevant to CSC4102*
4. Comparing Agile Methods

Scrum

A new-style scrum machine instrumental in developing. The power of the Manawatu scrum may be mass-produced and sold to rugby unions throughout the country.

"This is our improved Mark II model - gives full scrum simulation - knees up, biting, punching, jersey pulling - the lot!"
Scrum

In rugby, ‘scrum’ (related to “scrimmage”) is the term for a huddled mass of players engaged with each other to get a job done. In software development, the job is to put out a release.

Comes from Japan and based from industrial process control theory:

4. Comparing Agile Methods

Scrums for wicked problems

“the best adaptive project management approach for wicked software development projects”

Peter DeGrace and Leslie Hulet Stahl.
Wicked Problems, Righteous Solutions.
1990. Yourdon Press

Many of the systems problems facing software developers have all the characteristics of wicked problems:

1. There is no definitive formulation of a wicked problem.
2. Wicked problems have no stopping rule.
3. Solutions to wicked problems are not true-or-false but good-or-bad.
4. There is no immediate and no ultimate test of a solution to a wicked problem.
5. Every implemented solution to a wicked problem has consequences.
6. Wicked problems do not have a well-described set of potential solutions.
7. Every wicked problem is essentially unique.
8. Every wicked problem can be considered a symptom of another problem.
9. The causes of a wicked problem can be explained in numerous ways.
10. The planner (designer) has no right to be wrong.
Scrum: early research


M. Beedle et al., *SCRUM: An Extension Pattern Language for Hyperproductive Software Development*, Pattern Languages of Program Design 4, N. Harrison, B. Foote, and H. Rohnert, eds., Addison-Wesley, 2000,


### 4. Comparing Agile Methods

#### Scrum Phases

![Scrum Phases Diagram]

**Pre-Game**
- Planning and High-Level Design

**Mid-Game**
- **Sprint**
  - Plan
  - Review
  - Develop
  - Adjust
  - Wrap
  - Review

**Post-Game**
- Close
Main scrum concepts:

**Burndown chart.** This chart, updated every day, shows the work remaining within the sprint. The burndown chart is used both to track sprint progress and to decide when items must be removed from the sprint backlog and deferred to the next sprint.

**Product backlog.** Product backlog is the complete list of requirements—including bugs, enhancement requests, and usability and performance improvements—that are not currently in the product release.

**ScrumMaster.** The ScrumMaster is the person responsible for managing the Scrum project. Sometimes it refers to a person who has become certified as a ScrumMaster by taking ScrumMaster training.

**Sprint backlog.** Sprint backlog is the list of backlog items assigned to a sprint, but not yet completed. In common practice, no sprint backlog item should take more than two days to complete. The sprint backlog helps the team predict the level of effort required to complete a sprint.
Scrum Meetings

The backlog is key: it is populated during the planning phase of a release and defines the scope of the release.
During a sprint, the team has a daily meeting called a scrum. Each team member describes the work to be done that day, progress from the day before, and any blocks that must be cleared. To keep the meetings short, the scrum is supposed to be conducted with everyone in the same room—standing up for the whole meeting.

When enough of the backlog has been implemented so that the end users believe the release is worth putting into production, management closes development. The team then performs integration testing, training, and documentation as necessary for product release.
Scrum Meetings

We stand up to keep the meeting short

The daily stand-up meeting (also known as a "daily scrum", a "daily huddle", "morning roll-call", etc.) is simple to describe:

The whole team meets every day for a quick status update. We stand up to keep the meeting short.

But this short definition does not really tell you the subtle details that distinguish an effective stand-up from a waste of time. So how can you tell?

For experienced practitioners, when things go wrong with the stand-up, they will instinctively know what to adjust to fix the situation.

For novices, when things go wrong, it is much less likely that they'll figure out what to do... and it’s much more likely that, given no assistance, they will simply abandon the practice altogether.

This would be unfortunate since well-run stand-ups add significant value to teams.
Scrum Meetings

The goals of the daily stand-up are GIFTS

There are several goals for a daily stand-up meeting:

- To help start the day well
- To support improvement
- To reinforce focus on the right things: the baton not the runners
- To reinforce the sense of team
- To communicate what is going on

As a mnemonic device, think of GIFTS:
Good Start, Improvement, Focus, Team, Status

*The purpose is not to meet... it is to improve.*
-- Joe Ely, "More on Daily Start-Up Meetings"
Scrum Meetings

Who attends?.... All Hands

People and representatives from various areas wish to know about and/or contribute to the status and progress of the project. Communicating status in multiple meetings and reports requires a lot of duplicate effort. 

*Therefore*
Replace some or all of the meetings and reports with the daily stand-up. Anyone who is directly involved in or wants to know about the day-to-day operation of the project should attend the single daily stand-up meeting. 

*But*
People not directly involved can disrupt the stand-up if they are unclear about what is expected behaviour. This may be addressed by simply informing new participants and observers of expected norms beforehand.

In The Perfect World, *Work Items Attend* 
*Also Known As:* Story-focused stand-up - if the stories are so important to the project, *they* ought to be the ones speaking in the standup
4. Comparing Agile Methods

Scrum Meetings

What do we talk about?

Yesterday Today Obstacles

Also Known As: Three Questions

Some people are talkative and tend to wander off into **Story Telling**. Some people want to engage in **Problem Solving** immediately after hearing a problem. Meetings that take too long tend to have low energy and participants not directly related to a long discussion will tend to be distracted.

**Therefore**

Structure the contributions using the following format:

- What did I accomplish yesterday?
- What will I do today?
- What obstacles are impeding my progress?
Scrum Meetings

Where and when and how?

• Meet Where the Work Happens

• Same Place, Same Time

• At the start of the day? ... or not?

• Stand Up close to each other

• Prepare in advance

• Fifteen minutes or less ... Take problem solving off-line

• Encourage Autonomy (rotate the facilitator?)
## Scrum

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scrum uses small Sprints to effectively break the system down into smaller components and divided among teams.</td>
<td>Scrum only works when management “trust the developers to use own judgment “to accomplish task. Key attributes of scrum is light and subtle control. So if development team is young and immature, Scrum is risky.</td>
</tr>
<tr>
<td>A key activity in scrum is the “daily scrum meetings”, which help team members to show evidence of task completion and allows for continuous improvement, thereby enabling rapid, bottom-up engineering.</td>
<td>Scrum is ideally designed for company with “currently existing agile methods.” Therefore a company must already have some working knowledge of agile methods before using Scrum.</td>
</tr>
</tbody>
</table>
Lean development
Lean software development is a translation of lean manufacturing principles and practices to the software development domain. Adapted from the Toyota Production System, a pro-lean subculture is emerging from within the Agile community:


Mary Poppendieck and Tom Poppendieck. 2006. *Implementing Lean Software Development: From Concept to Cash (The Addison-Wesley Signature Series)*. Addison-Wesley Professional.

Lean Software Development - key ideas

• Life would be so much easier if customers would just stop changing their minds.

• Let customers delay their decisions about exactly what they want as long as possible, and when they ask for something, give it to them so fast they don’t have time to change their minds.

• Great designs come from great designers, and great designers understand that designs emerge as they develop a growing understanding of the problem, not collecting mass amounts of requirements.

• Deliver working system as fast as possible.

QUESTION: Do you agree with this?
Lean Software Development

1. **Eliminate waste.** Any activity that does not “pay for itself” in reduced effort elsewhere in the system and should be removed.

2. **Amplify learning.** Developers always need to learn new methods to produce the most robust system.

3. **Decide as late as possible.** The benefits of making a decision at the last minute is it avoids making the wrong decision early and then having to fix it later.

4. **Deliver as fast as possible.** This principle is that if you deliver very quickly, it reduces the chance of requirement changing. Which can cost a great deal if the change is late in the development.

5. **Empower the team.** In order to get people to take responsibility, get motivated, and gel as a team, they need to be responsible for the outcome and authorized to make it happen.

6. **Build integrity in:** It is key that the system maintains integrity throughout the development cycle. That means integration test, unit testing and general testing is a must, particularly from the customer.

7. **See the whole.** Don’t break the system down into parts, but keep it as a whole.
Lean software development: focus on testing

✓ Every few minutes
  ✗ Build & run unit tests
  ✗ **STOP** if the tests don’t pass
✓ Every day
  ✗ Run acceptance tests
  ✗ **STOP** if the tests don’t pass
✓ Every week
  ✗ Run production tests
  ✗ **STOP** if the tests don’t pass
✓ Every iteration
  ✗ Deployment-ready code
✓ Every Release
  ✗ Deploy and run in production
Lean software development: focus on testing
### Lean Software Development

#### Pros

- Thinking the system as a whole, though difficult for complex system, helps to guarantee consistency and integrity of the system. Reduces integration time since since is developed as singular unit.
- The work team designs its own processes, makes its own commitments, gathers the information needed to reach its goals, and polices itself to meet its milestones.

#### Cons

- With large system or complex system, the only way for developers to visualize its construction is through partitioning the system. But LSD suggest the opposite, which can be difficult to accomplish.
- Deciding as late as possible can have adverse affect to the schedule. This can hurt parallel development and increase implementation time.
Extreme Programming (XP)

XP is not this extreme!
4. Comparing Agile Methods

Extreme Programming

From: *Embracing Change with Extreme Programming*, Beck, 1999
4. Comparing Agile Methods

Extreme Programming (XP)

*Embracing Change with Extreme Programming*, Beck, 1999

You can’t program until you know what you’re programming.

**XP considers the period before a system first goes into production to be a dangerous anomaly in the life of the project and to be gotten over as quickly as possible. However, every project has to start somewhere.**

You put the overall analysis together in terms of stories, which you can think of as the amount of a use case that will fit on an index card. Each story must be business-oriented, testable, and estimable.

A month is a good long time to come up with the stories for a 10 person-year project.
Extreme Programming (XP)

*Embracing Change with Extreme Programming,* Beck, 1999

How do you know what you should be programming at any particular time?

The customer picks the next release by choosing the most valuable features (called stories in XP) from among all the possible stories, as informed by the costs of the stories and the measured speed of the team in implementing stories.

The customer picks the next iteration’s stories by choosing the most valuable stories remaining in the release, again informed by the costs of the stories and the team’s speed.

The programmers turn the stories into smaller-grained tasks, which they individually accept responsibility for.

Then the programmer turns a task into a set of test cases that will demonstrate that the task is finished.

Working with a partner, the programmer makes the test cases run, evolving the design in the meantime to maintain the simplest possible design for the system as a whole.
4. Comparing Agile Methods

**XP Practices I  (Usually associated with Agile)**

**Planning game.** Customers decide the scope and timing of releases based on estimates provided by programmers. Programmers implement only the functionality demanded by the stories in this iteration.

**Small releases.** The system is put into production in a few months, before solving the whole problem. New releases are made often—anywhere from daily to monthly.

**Metaphor.** The shape of the system is defined by a metaphor or set of metaphors shared between the customer and programmers.

**Simple design.** At every moment, the design runs all the tests, communicates everything the programmers want to communicate, contains no duplicate code, and has the fewest possible classes and methods. This rule can be summarized as, “Say everything once and only once.”

**Tests.** Programmers write unit tests minute by minute. These tests are collected and they must all run correctly. Customers write functional tests for the stories in an iteration. These tests should also all run, although practically speaking, sometimes a business decision must be made comparing the cost of shipping a known defect and the cost of delay.

**Refactoring.** The design of the system is evolved through transformations of the existing design that keep all the tests running.

**Continuous integration.** New code is integrated with the current system after no more than a few hours. When integrating, the system is built from scratch and all tests must pass or the changes are discarded.
4. Comparing Agile Methods

XP Practices II (also found outside Agile)

Pair programming. All production code is written by two people at one screen/keyboard/mouse.

Collective ownership. Every programmer improves any code anywhere in the system at any time if they see the opportunity.

On-site customer. A customer sits with the team full-time.

40-hour weeks. No one can work a second consecutive week of overtime. Even isolated overtime used too frequently is a sign of deeper problems that must be addressed.

Open workspace. The team works in a large room with small cubicles around the periphery. Pair programmers work on computers set up in the center.

Just rules. By being part of an Extreme team, you sign up to follow the rules. But they’re just the rules. The team can change the rules at any time as long as they agree on how they will assess the effects of the change.
4. Comparing Agile Methods

XP - Daily Communication is Key
Extreme Programming (XP)

For XP to be a success, critical expertise is required in:

Building User Stories - A user story describes problems to be solved by the system being built. These stories must be written by the user and should be about three sentences long. User stories do not describe a solution, use technical language, or contain traditional requirements-speak.

Turning stories into code - Because user stories are short and somewhat vague, XP will only work if the customer representative is on hand to review and approve user story implementations.

Turning stories into test code - Unit testing is central to XP, where two twists on conventional testing strategies make tests far more effective: Programmers write their own tests and they write these tests before they code.

Evolving design - Must not break existing tests and must also support scaleable incremental development
Extreme Programming (XP)

Beck acknowledges a wide number of influences that led to the development of XP. A selection of the most accessible are:


The pros of XP

• Done well, XP improves teamwork.
  • It builds true competency in all team members.
  • It makes for an enjoyable and honest work day.
  • It gets people out of their cubes and talking to one another.

• TDD teaches developers about how to write quality code and how to improve their notions of design; it helps them to improve estimates. It improves the resumes of developers.

• It gives management many tools, including predictability, flexibility of resources, consistency, and visibility into what's really going on.

• It gives customers the ability to see whether or not a company can deliver on its promises.

• You don't spend a lot of time in stupid, wasteful meetings, and you don't produce a lot of useless documents.
The cons of XP

• Design becomes implicit rather than explicit

• Relying on emergent design is risky

• It is very hard to write good tests

• Too frequent iterations can compromise quality

• To do it well you need to do it often - so its hard to introduce successfully
4. Comparing Agile Methods

Feature Driven Development

Evolved from the Coad Method, in the late 1990s

Peter Coad and Jeff De Luca later published a brief outline of FDD in their 1999 book “Java Modeling in Color with UML”.

Stephen Palmer with Mac Felsing wrote the definitive textbook, “A Practical Guide to Feature Driven Development” two years later.

Features:

Are tiny
Map directly onto an object domain model
Can be coded directly
Can be assembled in component sets.
Feature Driven Development: looks pretty agile!

I would not recommend this for beginners to Agile development as it is hard to manage interactions between features.
Comparing Agile Methods

### 4. Comparing Agile Methods

#### Agile techniques which motivated aspects of CSC4102

<table>
<thead>
<tr>
<th>Feature</th>
<th>Scrum</th>
<th>Lean Development</th>
<th>Extreme programming (XP)</th>
<th>Feature Driven Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short “Sprints”</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Short, high frequency meetings</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Discourage changing requirements</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Working code ASAP</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Build integrity through tests</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>See the whole</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Increments - as stories or use cases</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Refactoring design</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Continuous integration</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Pair programming</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Prioritizing Increments</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Minimizing Features</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

**cf. “Pour aller plus loin” for details**
4. Comparing Agile Methods

Differences between CSC4102 and *Agile in Industry*

CSC4102 focuses on technical aspects; Agile in Industry also considers non-technical issues such as management, costs, resources, business, marketing, etc… in an agile framework.

CS4102 had to compromise a purely agile approach in order to meet its teaching objectives (because the software engineers participating in the development had to learn about things that are normally taken for granted in industrial application of agile methods).
5. Agile Resources

http://agilemanifesto.org/

http://www.agilealliance.org/

http://en.wikipedia.org/wiki/Agile_software_development

http://scrummethodology.com/

http://www.extremeprogramming.org/

http://www.leansoftwareinstitute.com/

An Overview of Feature-Oriented Software Development

An extended version of this presentation - containing links to many additional resources - can be found in: “Pour aller plus loin”
QUESTIONS ....?

Feedback helps us improve our teaching process!