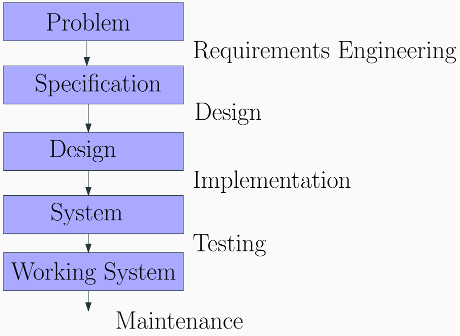
CS-230 Software Engineering

L16: Software Life Cycles 2 and Agile Methods

Dr. Liam O'Reilly

Semester 1 – 2020

Previously in CS230...



Waterfalls and Software

Previously in CS 230...

- Software life cycles
- We began to tame the software crisis with the waterfall model
- The stages of the waterfall model are:
  - Requirements Engineering
  - Design
  - Implementation
  - Testing
  - Maintenance

Previously in CS 230... (2)

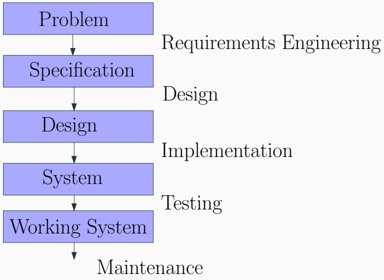
- Is this the only life cycle model?
- No there are many:
  - V-Model
  - Spiral Model
  - Agile Programming
- Current shifts in Software Engineering
  - Producing vs Using Software
  - Geographical Heterogeneity
  - Shifts to Open Source

Previously in CS 230... (3)

- Waterfall is heavyweight but thorough
- Agile is quick to adapt changing requirements
- Which should we use? How should we go about it?

Software Life Cycle Models and Agile Methods

Advantages of the Waterfall Model

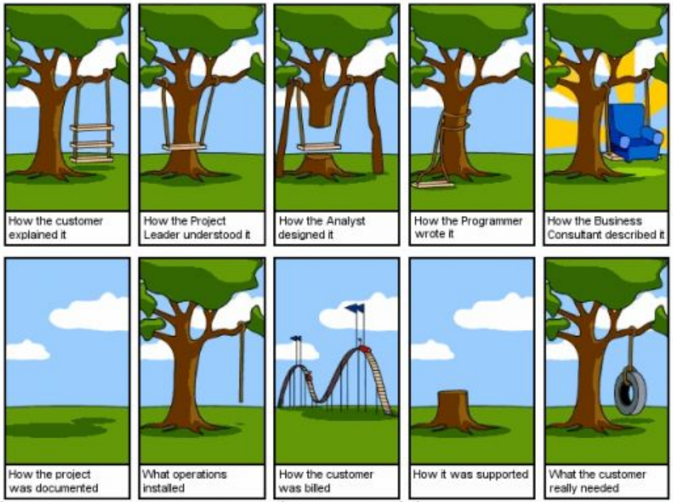


- Controlled, document driven, planning driven, heavyweight
- Stage is completed when documents are delivered
  - requirements specification, design specification, program, test document

Advantages of the Waterfall Model (2)

- The waterfall inherently enforces after each stage:
    - iteration and feedback
    - validation (are we building the right system?)
    - verification (are we building the system right?)
- User requirements are fixed early on => go and design system
- **What are the drawbacks with this development model?**

Waterfall Problems


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

- User requirements can change and be miss-communicated
- Different team members have different visions
- Contact with the client only at Req, and Testing/Maint

Agile Methods

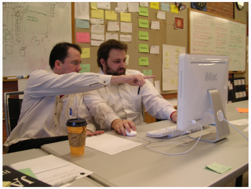
Rapid Software Development

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a global rapidly changing environment – it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs.
- Rapid software development
  - Specification, design and implementation are inter-leaved
  - System is developed as a series of versions with stakeholders involved in version evaluation
  - User interfaces are often developed using an IDE and graphical toolset.

Agile Methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
    - Focus on the code rather than the design.
    - Are based on an iterative approach to software development.
    - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

The Agile Manifesto – 2001



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. http://www.agilemanifesto.org/

Agile Method Applicability

- Product development where a software company is developing a small or medium-sized product for sale.
- Custom system development within an organisation, where there is a clear commitment from the customer to become involved in the development process and where there are **not a lot** of external rules and regulations that affect the software.
- Because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems.

Problems with Agile

- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the intense involvement that characterises agile methods.
- Prioritising changes can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work. Contracts may be a problem as with other approaches to iterative development.

Agile Methods and Software Maintenance

- Most organisations spend more on maintaining existing software than they do on new software development. So, if agile methods are to be successful, they have to support maintenance as well as original development.
- Two key issues:
  - Are systems that are developed using an agile approach maintainable, given the emphasis in the development process of minimising formal documentation?
  - Can agile methods be used effectively for evolving a system in response to customer change requests?
- Problems may arise if original development team cannot be maintained.

Lightweight Agile Methods

Agile methods describe a family of methodologies/practices:

- Incremental Development & Delivery
- eXtreme Programming (XP)
- Rapid Prototyping (not strictly Agile)
- Scrum – an industry standard!

Incremental Development & Delivery

Incremental Delivery

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements for that increment are frozen though requirements for later increments can continue to evolve.

Incremental Development and Delivery

- Incremental Development
    - Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;
    - You then do a Waterfall-style model for that increment.
    - Normal approach used in agile methods;
    - Evaluation done by user/customer proxy.
- Incremental Delivery
    - Deploy an increment for use by end-users;
    - More realistic evaluation about practical use of software;
    - Deployment of an increment can influence specifications of future increments.

Incremental Delivery Advantages

- Customer can gain value early (and start using product) as they do not have to wait for the entire system to be developed. The first increment will satisfy their most critical requirements.
- Early increments act as a prototype to help elicit requirements and feedback for later increments. (Increments are not prototypes, they are finished aspects).
- The highest priority system services tend to receive the most testing (as they are delivered first).
- Lower risk of overall project failure.
- The process maintains the benefits of incremental development: changes should be relatively easy to incorporate (at least in later increments).

Incremental Delivery Problems

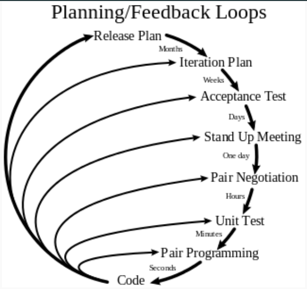
- Most systems require a set of basic facilities that are used by different parts of the system.
  - As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
- The essence of iterative processes is that the specification is developed in conjunction with the software.
  - However, this conflicts with the procurement model of many organisations, where the complete system specification is part of the system development contract. Requires new types of contract which large organisations might find difficult to accommodate.

Incremental Delivery Problems (2)

- The process is not visible.
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- System structure tends to degrade as new increments are added.
  - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.
- Difficult to implement for replacement systems as increments have less functionality than the system being replaced (and currently used).

eXtreme Programming (XP)

eXtreme Programming (XP)



Perhaps the best-known and most widely used agile method.

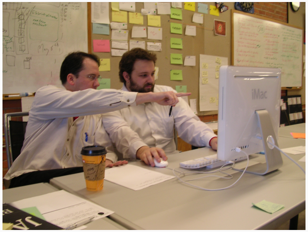
Extreme Programming (XP) takes an ‘extreme’ approach to iterative development.

- New versions may be built several times per day;
- Increments are delivered to customers every 2 weeks;
- All tests must be run for every build and the build is only accepted if tests run successfully.

XP and Agile Principles

- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People not process, through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases. Maintaining simplicity through **constant refactoring of code**.

XP and Agile Principles (2)



- Pair programming (collective ownership, code review)
- Write the tests first and then implement code to meet them
- Continuous integration: after each piece of work, build whole system and run all tests.
- Change is embraced, designing for change de-emphasized. **Programmers expected to refactor code as they work**.

Advantages of Agile Methods

- Client is at the centre of the development process.
    - Should be onsite at all times.
- Easy to adapt to changing software requirements
- Process is less rigid and fosters collaboration
- Best for smaller teams and products
- Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity.

XP and Change

- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.
- XP, however, maintains that this is not worthwhile as changes cannot be reliably anticipated.
- Rather, it proposes constant code improvement (refactoring) to make changes easier when they have to be implemented.

Refactoring

- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is well-structured and clear.
- However, some changes requires architecture refactoring and this is much more expensive.
- XP tends to need better programmers than Plan-Driven approaches. They need to foresee what will make large refactoring and try to avoid doing it in the first place.

Examples of Refactoring

- Re-organisation of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

Summary

- Agile methods are incremental development methods that focus on rapid development, frequent releases of the software, reducing process overheads and producing high-quality code. They involve the customer directly in the development process.
- The decision on whether to use an agile or a plan-driven approach to development should depend on the type of software being developed, the capabilities of the development team and the culture of the company developing the system.
- Extreme programming is a well-known agile method that integrates a range of good programming practices such as frequent releases of the software, continuous software improvement and customer participation in the development team.