4 Development Teams

A typical Software Team consists of the following roles:

- Project Manager
- System Architect
- UX Designer (Software Designer)
- Programmer
- Software Tester

In addition, we have the Stakeholders or Customers that play an important role in the development.

In Figure 4-1 we see a typical Software Team.

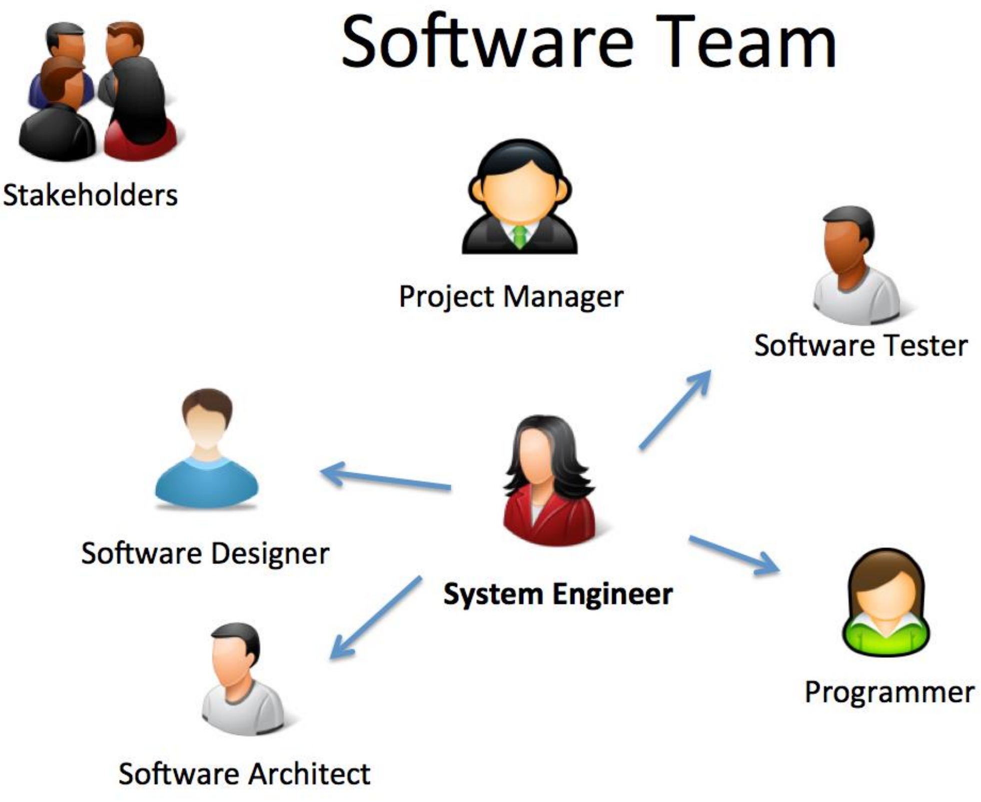


Figure 4-1: Software Team

A System Engineer is a general person that could be a Programmer, Architect, Designer, Tester in different phases in the project, or he could be a tester in one project and a programmer in another project – all in one person. That is usually the case in small companies, while in larger companies these roles (designer, tester, programmer) could be a full-time job.

4.1 Teams

To create successfully software, collaboration inside the team is essential.

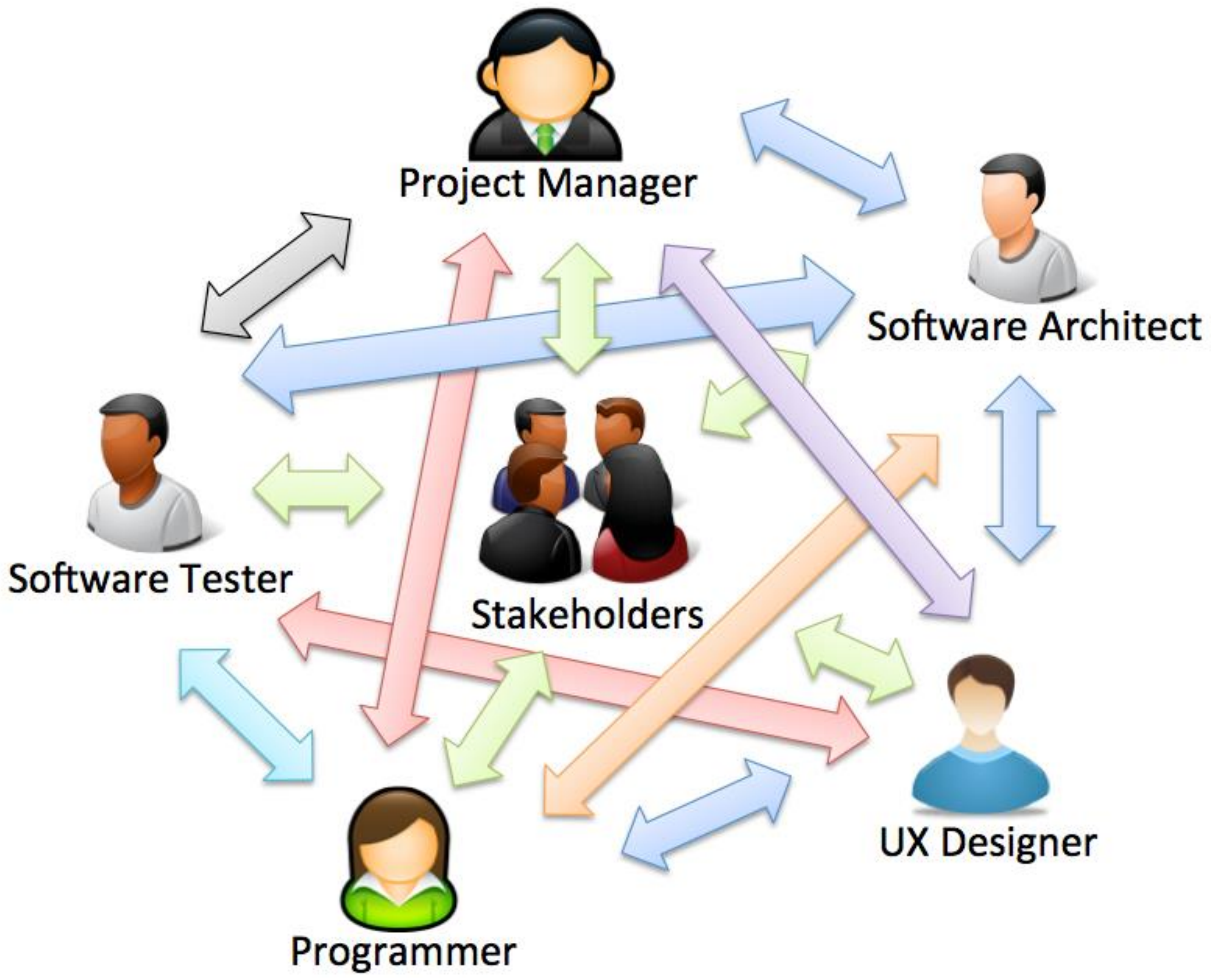


Figure 4-2: Team Collaboration

It is important that the team collaborate. Communication as well!

4.2 Roles

A typical Software Team consists of the following roles:

- Project Manager
- System Architect
- UX Designer (Software Designer)
- Programmer
- Software Tester

In addition, we have the Stakeholders or Customers that play an important role in the development.

We will discuss these roles in more detail below.

4.2.1 Stakeholders

All the people that has an interest in the outcome of the software are called Stakeholders. In most cases the Stakeholders are referred to as "Customers" but others may also be referred to as stakeholders, such as management, shareholders, etc.

4.2.2 Project Manager

The Project Managers have the responsibility of the planning, execution and closing of the project.

More about Project management in a later chapter.

4.2.3 System Architect

With "Technical Design" we mean the Platform and Architecture Design, i.e., how to build the software.

This is typically done by a so-called Software/System Architect.

4.2.4 UX Designer

UX Design is the Design of the User eXperience (UX) and the Graphical User Interface (GUI), sometimes also called Human Machine Interface (HMI). This is what the end user of the software see.

This is typically done by a so-called UX Designer.

4.2.5 Programmer

The Programmer or the Developer is doing the actual implementation of the software, i.e., the coding.

4.2.6 Software Tester

Before the customer can start using the software it needs to be properly tested. The Developer/Programmer needs to test his software, but since software consists of several software modules and components created by different developers, we need dedicated software testers that can test the software on a higher level.

The Customers are/should also be involved in the testing as well.

5 Software Development Phases

In software development, we have the following phases:

- Requirements (e.g., from Customer)
- Analysis and Design
- Implementation, Coding
- Documentation
- Testing
- Deployment, Installation and Maintenance

This chapter introduces these phases. Figure 5-1 shows an overview of the different phases involved in Software Development:

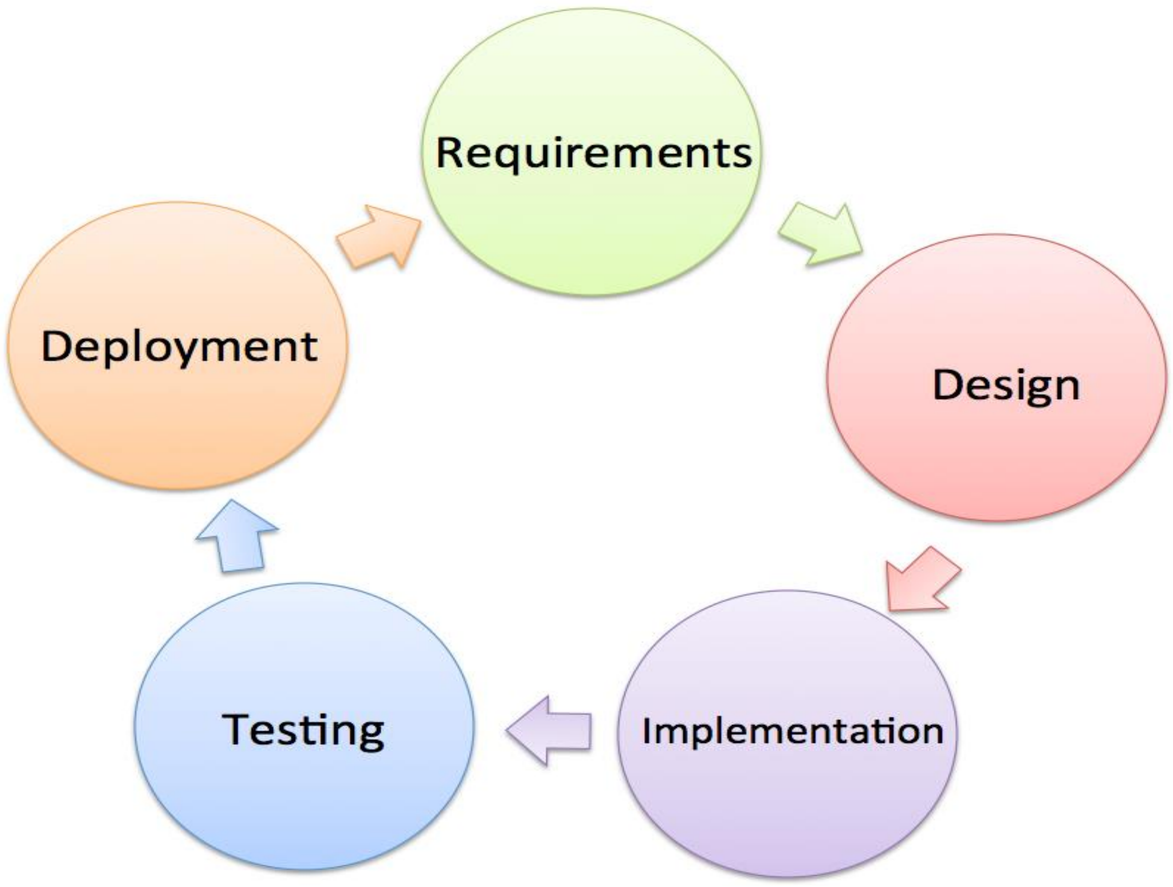


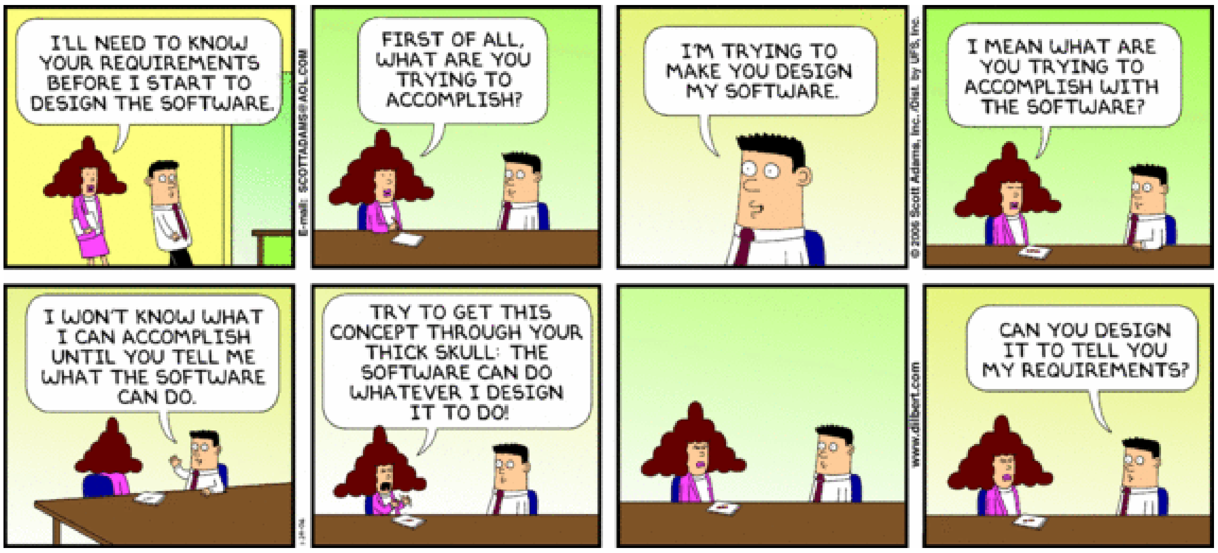
Figure 5-1: Phases in Software Development

5.1 Requirements

In the requirements, we describe what the system should do. The requirements include both functional requirements and non-functional requirements [1].

Functional Requirements: Statements of services the system should provide, how the system should react to inputs and how the system should behave in different situations. May state what the system should not do.

Non-Functional Requirements: Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc. Often apply to the system rather than individual features or services.


I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?
E-mail: SCOTTADAMS@AOL.COM
© 2006 Scott Adams, Inc./Dist. by UFS, Inc.
www.dilbert.com

The requirements are often collected in a so-called "Software Requirements Specification (SRS)" document.

The SRS could contain stuff like [2]:

- Introduction
  - Purpose
  - Definitions
  - System overview
  - References
- Overall description
  - Product perspective
    - System Interfaces
    - User Interfaces
    - Hardware interfaces
    - Software interfaces
    - Communication Interfaces
    - Memory Constraints

        - Operations
        - Site Adaptation Requirements
    - Product functions
    - User characteristics
    - Constraints, assumptions and dependencies
- Specific requirements
    - External interface requirements
    - Functional requirements
    - Performance requirements
    - Design constraints
        - Standards Compliance
    - Logical database requirement
    - Software System attributes
        - Reliability
        - Availability
        - Security
        - Maintainability
        - Portability

- Other requirements

The Requirements is normally given by the Customer if we deal with customized products. The software requirements document is the official statement of what is required of the system. It should include both a definition of user requirements and a specification of the system requirements. It is NOT a design document. As far as possible, it should include a set of WHAT the system should do rather than HOW it should do it [1].

5.2 Design

In the design phase, we use the specification and transform it into descriptions of how we should do it.

In principle, requirements should state what the system should do and the design should describe how it does this – but in practice this is not so easy! - In practice, requirements and design are inseparable.

We can divide design into 2 main groups:

- Technical Design – Platform and Architecture Design, i.e., how to build the software.
- UX Design – Design of User eXperience (UX) and the Graphical User Interface (GUI), sometimes also called Human Machine Interface (HMI). This is what the end user of the software see.

5.2.1 Technical Design

Technical Design is the Platform and Architecture Design, i.e., how to build the software.

This is typically done by a so-called Software Architect.

5.2.2 UX Design

UX Design is the Design of the User eXperience (UX) and the Graphical User Interface (GUI), sometimes also called Human Machine Interface (HMI). This is what the end user of the software see.

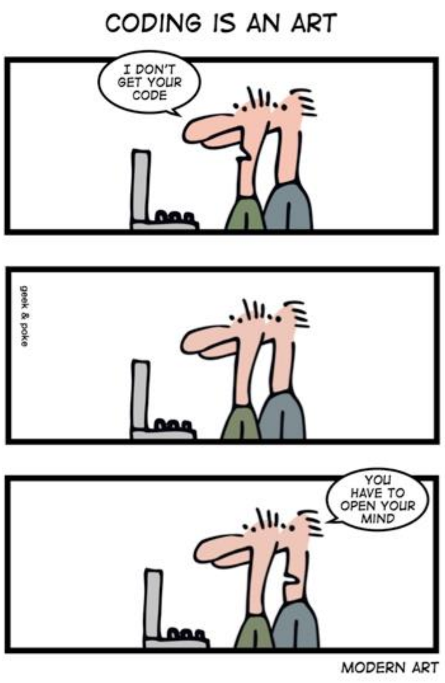
This is typically done by a so-called UX Designer.

5.3 Implementation

Implementation = Coding.

Software is usually designed and created (coded/written/programmed) in integrated development environments (IDE) like Eclipse, Xcode or Microsoft Visual Studio that can simplify the process and compile the program to an executable unit. Software is usually created on top of existing software and the application programming interface (API) that the underlying software frameworks provide, e.g. Microsoft .NET, etc.

Most of the software has a Graphical User Interface (GUI). Normally you separate the GUI design and code in different layers or files.



More about implementation later in this document.

5.4 Testing

Testing can be performed on different levels and by different persons. Testing is a very important part of software development. About 50% of the software development is about testing your software.

Creating User-friendly Software is Crucial!

More about Testing later in this document.

5.5 Deployment

What is Deployment?

Software deployment is all the activities that make a software system available for use.

Examples:

- Get the software out to the customers
- Creating Installation Packages
- Documentation, e.g., Installation Guide, etc.
- Installation
- etc.

Deployment strategies may vary depending of what kind of software we create, etc.

More about Deployment later in this document.

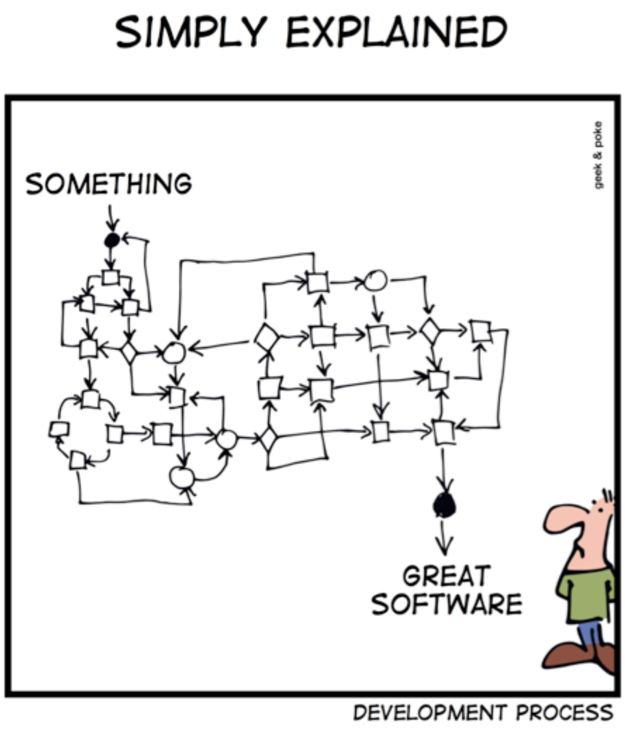
When the software is deployed, or installed, you normally go into a Maintenance phase. The maintenance of software involves bug fixes of the software after the software is released, etc. At some time, you also need to start planning new releases of the software.

6 Software Development Process

There are lots of different software development processes or methods in use today [3], e.g.:

- Waterfall model
- V-model
- Spiral model
- Unified Process (UP)/ Rational Unified Process (RUP)
- Scrum
- eXtreme Programming (XP)
- Lean Software Development
- TDD (Test Driven Development)
- Lean Software Development
- Kanban
- etc.

These processes or models may be divided in 2 main categories; **Plan-driven models** and **Agile methods**. The Waterfall model, V-model and the Spiral model is so-called plan-driven models, while Scrum and eXtreme Programming are so-called Agile methods.



Traditionally plan-driven methods where used in software development, while today Agile methods such as Scrum have become very popular, especially in smaller development teams.

Plan-driven models (e.g., Waterfall) generally produce more documentation than Agile models.

In Figure 6-1 we see an overview of some of the most used methods.

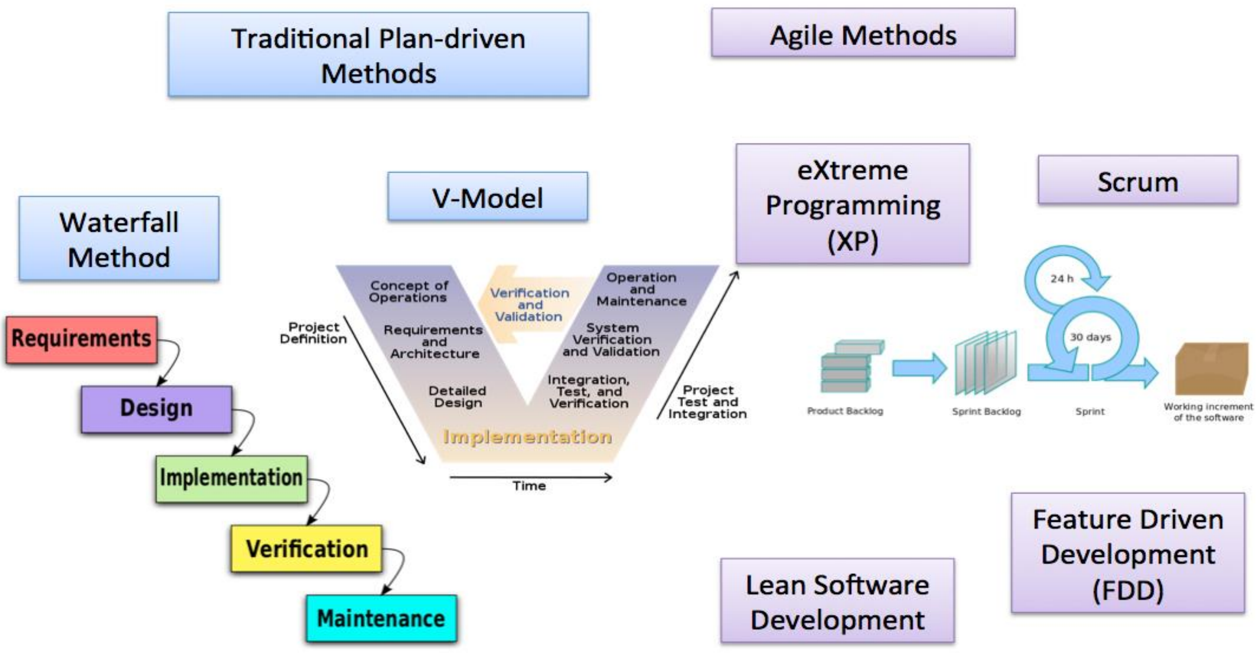

Traditional Plan-driven Methods
Agile Methods
Waterfall Method
Requirements
Design
Implementation
Verification
Maintenance
V-Model
Concept of Operations
Requirements and Architecture
Detailed Design
Implementation
Verification and Validation
Operation and Maintenance
System Verification and Validation
Integration, Test, and Verification
Project Definition
Project Test and Integration
Time
eXtreme Programming (XP)
Scrum
24 h
30 days
Product Backlog
Sprint Backlog
Sprint
Working increment of the software
Feature Driven Development (FDD)
Lean Software Development

Figure 6-1: Software Development Methods

In Figure 6-2 we see the main difference between Agile development and ordinary plan-driven development.

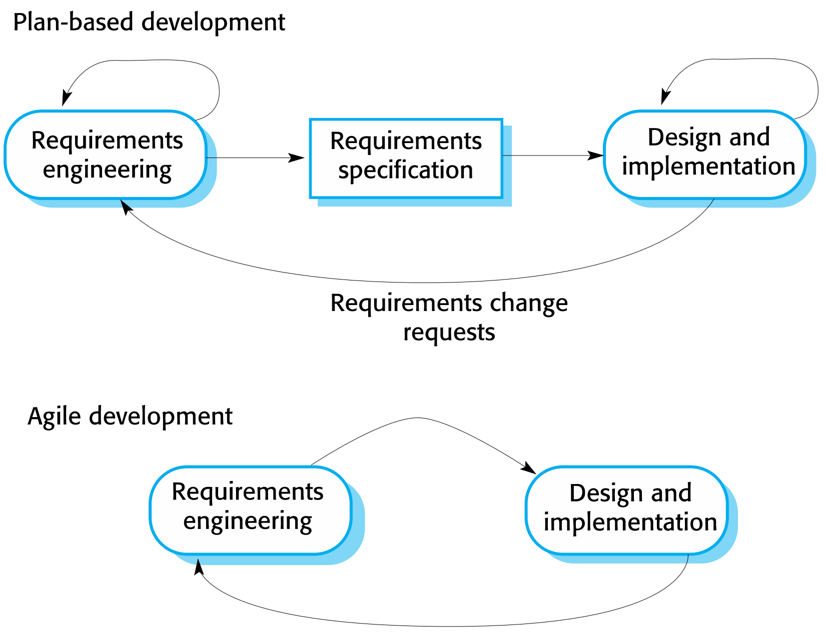
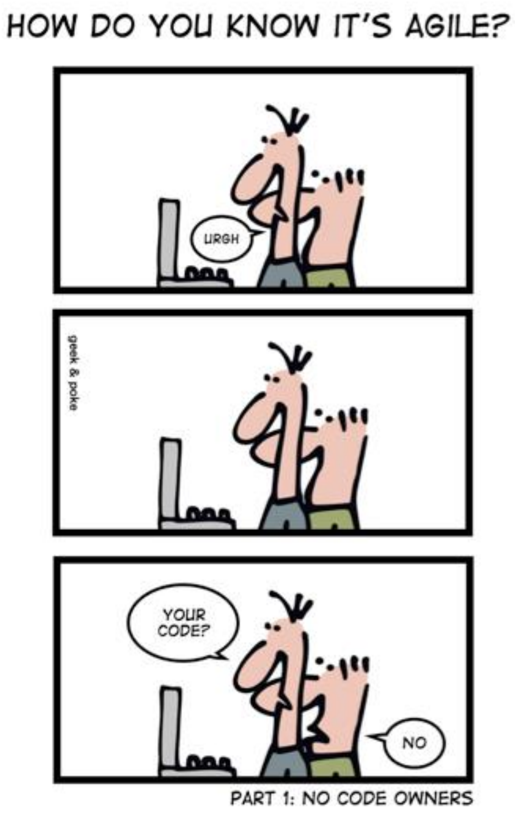

Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

Figure 6-2: Plan-driven vs. Agile Development [1]



6.1 Plan-driven models

We have different plan-driven models such as the Waterfall model, V-model, Spiral model which we will discuss in more details.

6.1.1 Waterfall model

The Waterfall model [4] consists of the following phases:

- Requirements specification (Requirements analysis)
- Software design
- Implementation and Integration
- Testing (or Validation)
- Deployment (or Installation)
- Maintenance

Traditionally with the Waterfall model, you can only start on the next phase when the previous phase is finished. Therefore, it is called the Waterfall method, see Figure 6-3.

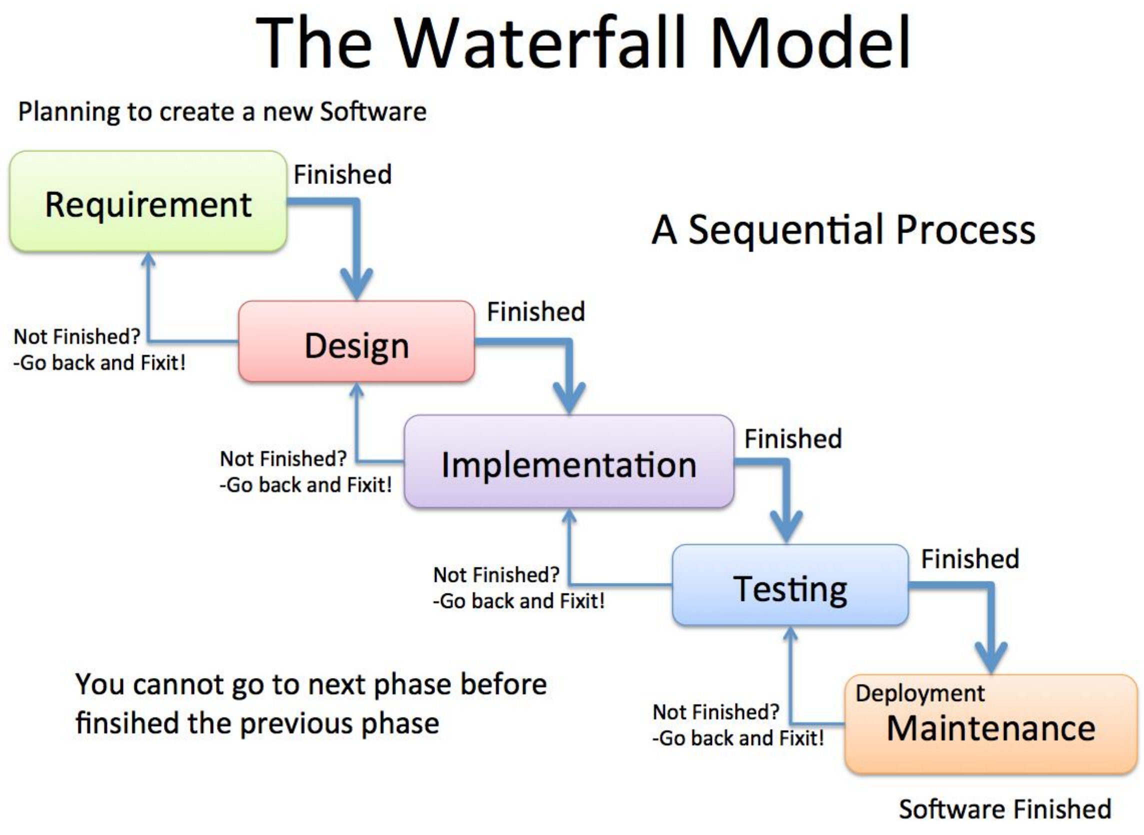

The Waterfall Model
Planning to create a new Software
Requirement
Finished
A Sequential Process
Design
Finished
Not Finished? -Go back and Fixit!
Implementation
Finished
Not Finished? -Go back and Fixit!
Testing
Finished
Not Finished? -Go back and Fixit!
You cannot go to next phase before finsihed the previous phase
Deployment
Maintenance
Not Finished? -Go back and Fixit!
Software Finished

Figure 6-3: Waterfall model [4]

In practice, there is impossible to create perfect requirements and design before you start implementing the code, so it is common to go back and update these phases iteratively.

6.1.2 V-model

The V-model [5] is derived from the more traditional Waterfall model.

The V-model is an extension of the waterfall model, but its using a more flexible approach.

“The V-Model reflects a project management view of software development and fits the needs of project managers, accountants and lawyers rather than software developers or users.”

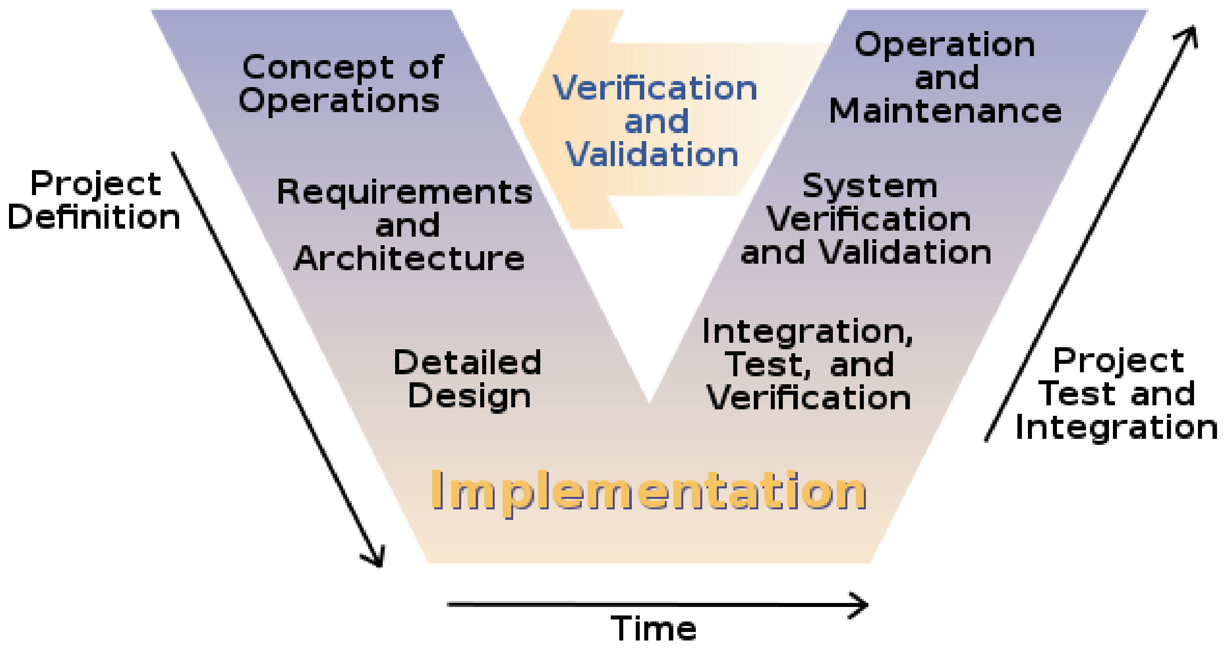
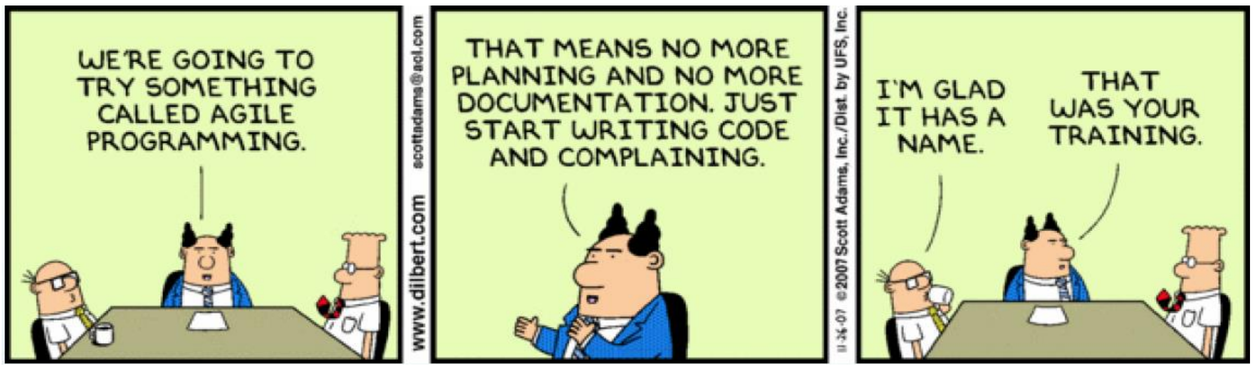

Concept of Operations
Requirements and Architecture
Detailed Design
Implementation
Integration, Test, and Verification
System Verification and Validation
Operation and Maintenance
Verification and Validation
Project Definition
Project Test and Integration
Time

Figure 6-4: V-model [5]

As we see in Figure 6-4, the left side is about requirements and design, while the right-side of the model is about testing and validating.

6.2 Agile Software Development

Agile software development is a group of software development methods based on iterative and incremental development.


WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.
www.dilbert.com scottadams@aol.com
11-26-07 ©2007 Scott Adams, Inc./Dist. by UFS, Inc.

So what is Agile development? – Here is a short summary:

- A group of software development methods
- Iterative approach
- Incremental: Software available to Customers every 2-4 weeks
- Self-organizing and cross-functional Teams
- Refactoring

In Figure 6-5 we see some important Agile features and principles.

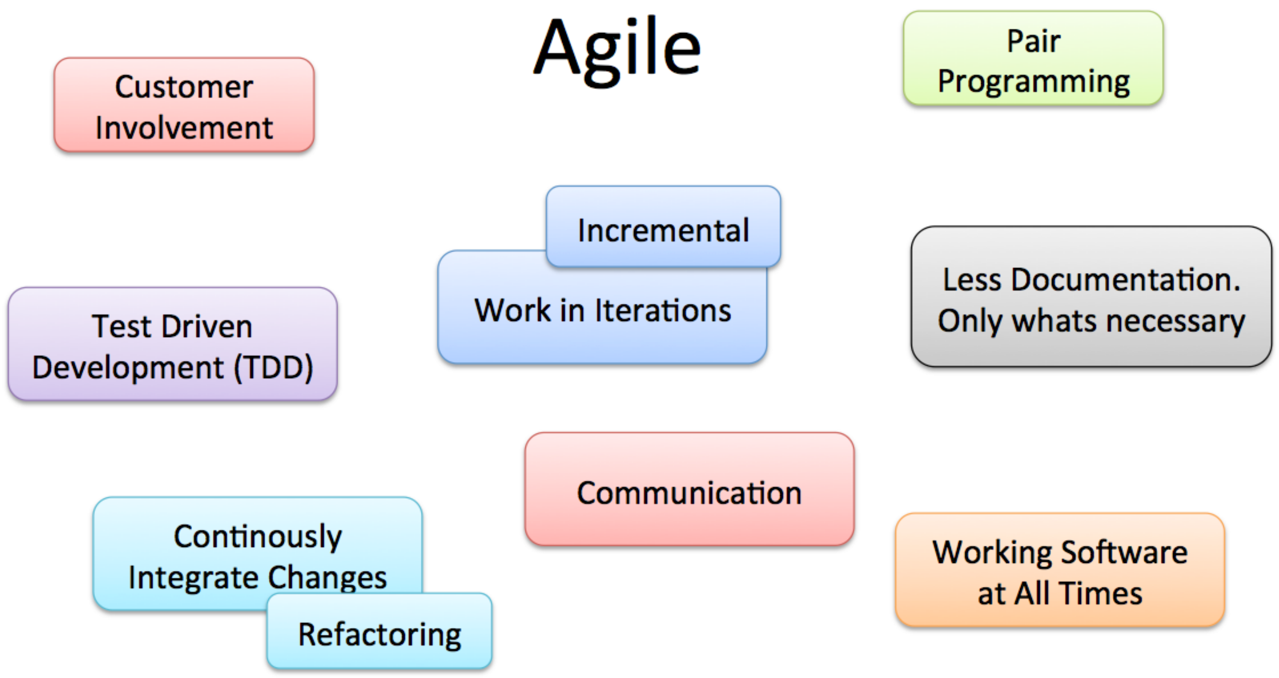

Agile
Customer Involvement
Pair Programming
Incremental
Work in Iterations
Test Driven Development (TDD)
Less Documentation. Only whats necessary
Communication
Continously Integrate Changes
Refactoring
Working Software at All Times

Figure 6-5: Agile Features and Principles

Examples of popular Agile methods:

- Scrum
- eXtreme Programming (XP)

In Figure 6-6 we see the key features with Agile Software Development.

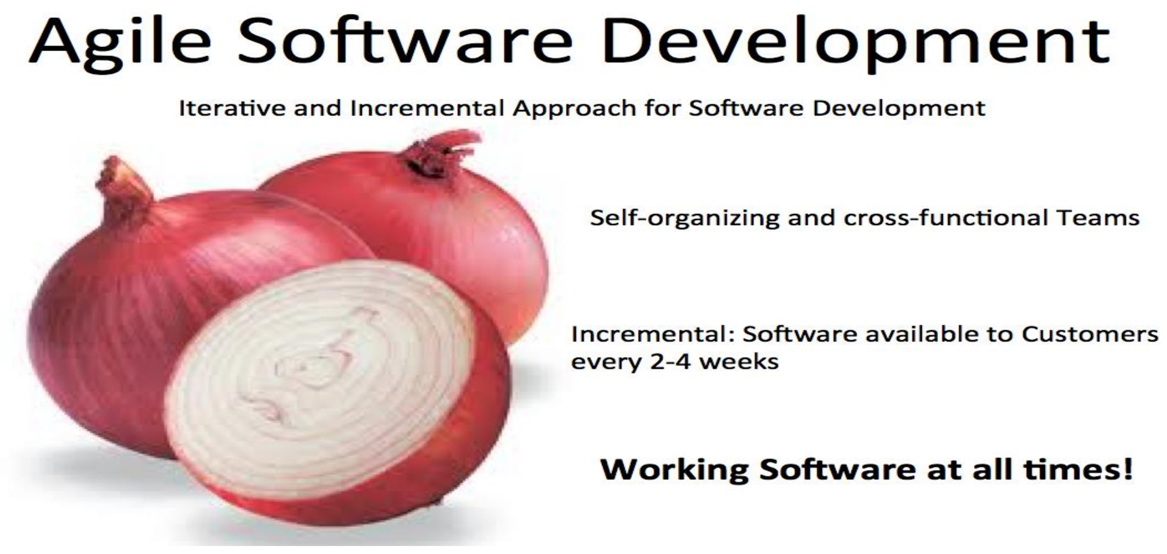

Agile Software Development
Iterative and Incremental Approach for Software Development
Self-organizing and cross-functional Teams
Incremental: Software available to Customers every 2-4 weeks
Working Software at all times!

Figure 6-6: Agile Software Development

Figure 6-7 shows some main differences between Agile development and more traditional development methods, such as, e.g., the Waterfall method.

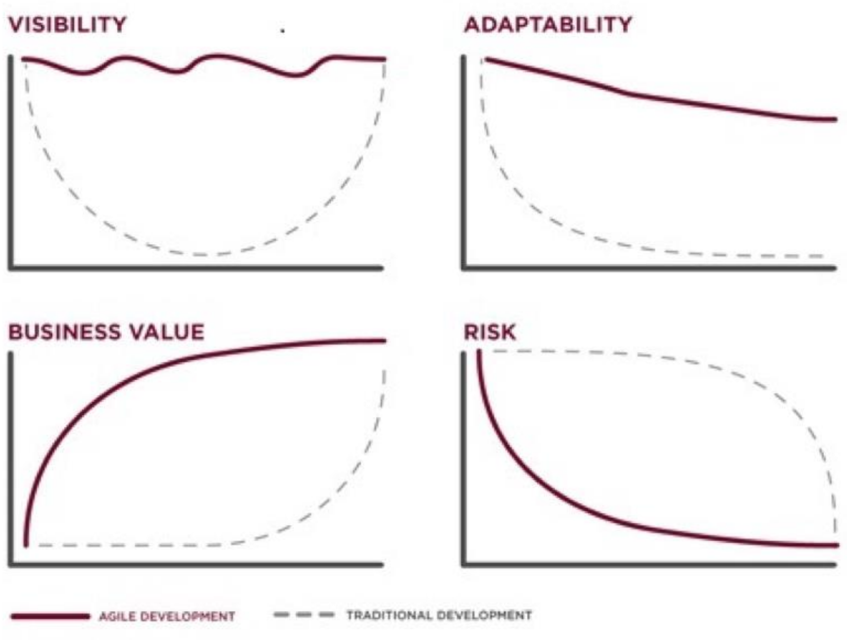


Figure 6-7: Agile vs. Traditional Development [6]

The Manifesto for Agile Software Development:

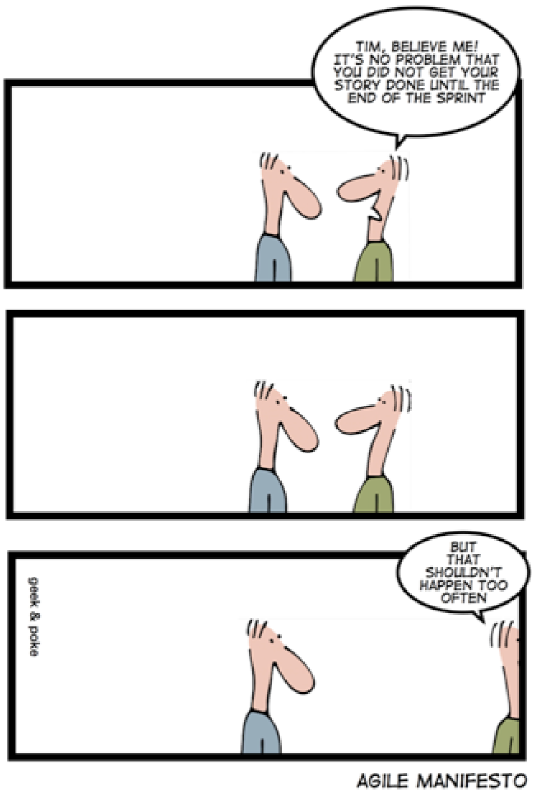
In 2001, some software developers met to discuss development methods. They published the Manifesto for Agile Software Development to define the approach now known as agile software development.

The Manifesto for Agile Software Development is as follows [7]:

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.


TIM, BELIEVE ME! IT'S NO PROBLEM THAT YOU DID NOT GET YOUR STORY DONE UNTIL THE END OF THE SPRINT
BUT THAT SHOULDN'T HAPPEN TOO OFTEN
geek & poke
AGILE MANIFESTO

Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. The outstanding work (or backlog) is often on the vertical axis, with time along the horizontal. That is, it is a run chart of outstanding work. It is useful for predicting when all the work will be completed.

It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.

In Figure 6-8 we see a typical Burndown chart.

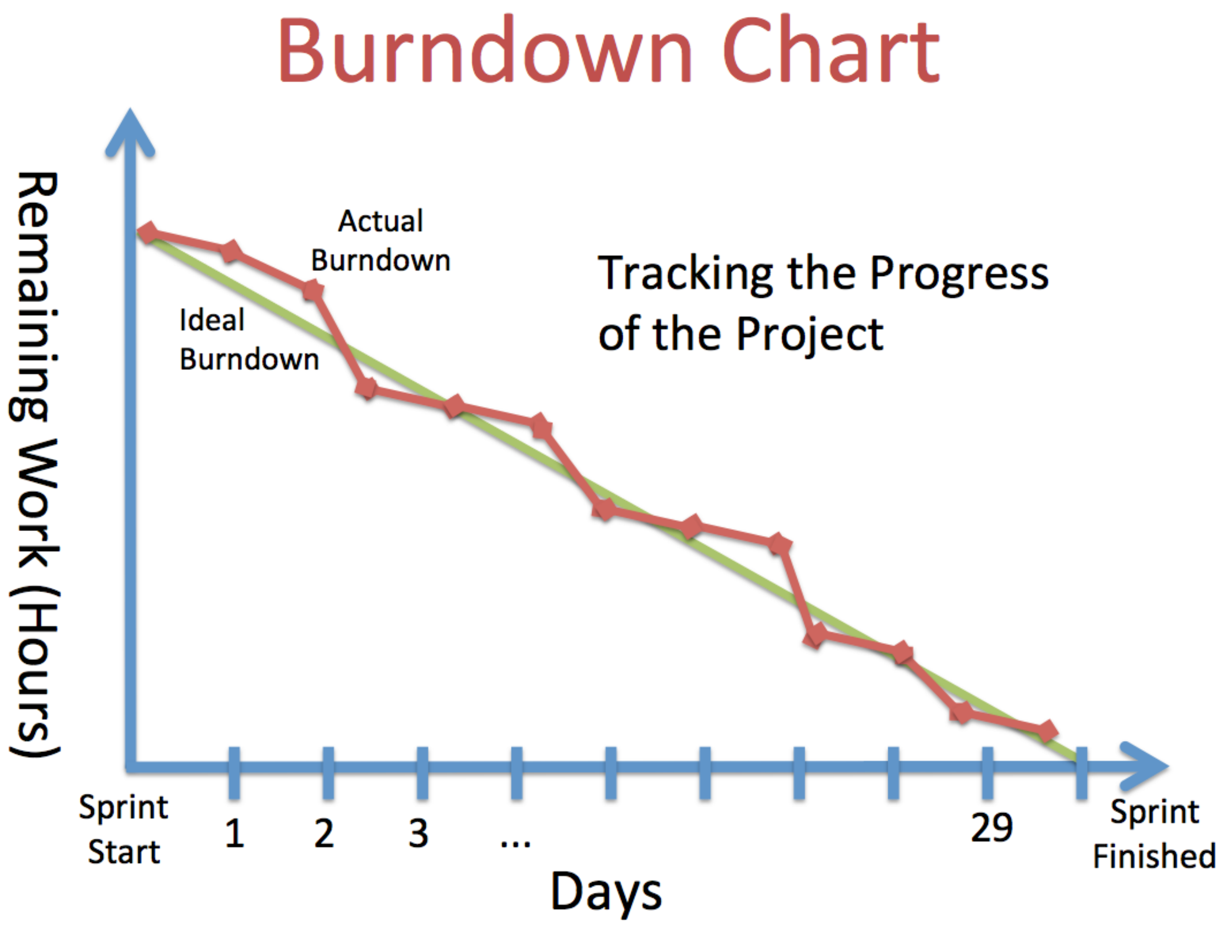


Figure 6-8: Burndown Chart

6.2.1 Waterfall vs. Agile

Agile is more flexible than traditional methods (like the waterfall).

Here are some key factors that separates the traditional waterfall method versus the more flexible Agile methods, such as Scrum:

- Agile and Scrum is based on Iterations while Waterfall is Sequential
- Agile and Scrum focus on less documentation
- Agile is good for small projects – not so good for larger projects?
- If the Customer don't know what he wants in detail – Scrum is a good approach

In Figure 6-9 we see some important differences between the traditional waterfall method and the Agile Development approach. We see that Agile delivers value in each iteration of the development.

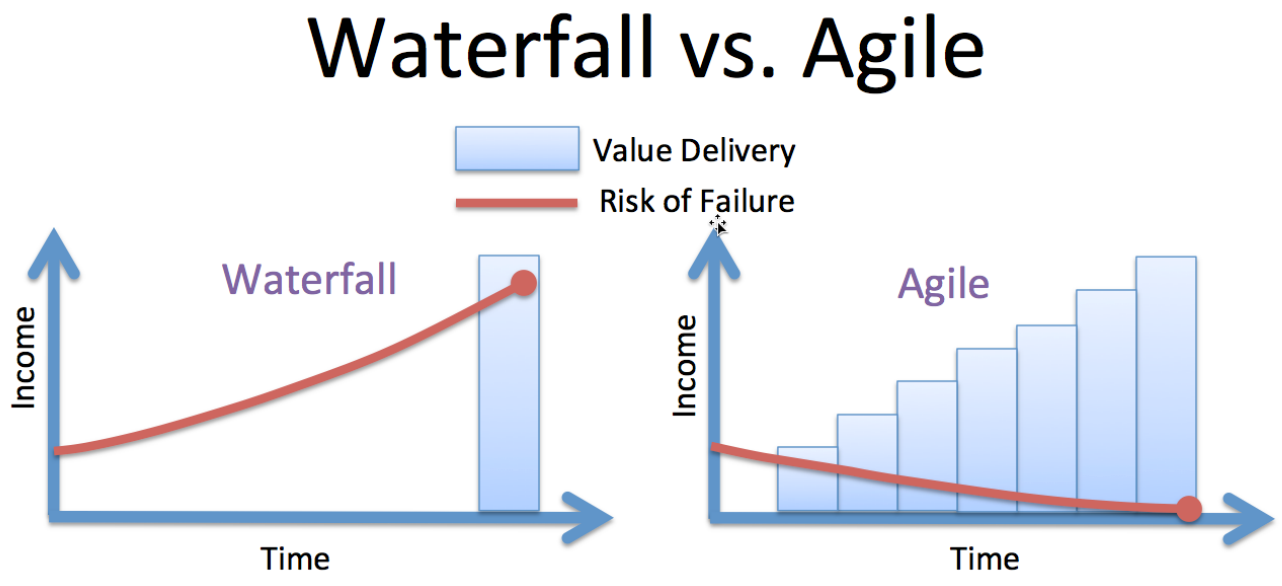


Figure 6-9: Waterfall vs. Agile Development

6.2.2 eXtreme Programming (XP)

eXtreme Programming or shorted XP is a popular Agile method. Typical features in XP are as follows:

- Pair Programming
- Code Reviews
- Refactoring
- Unit Testing - In XP you start by writing Unit Tests before you start coding
- Standup Meetings

EXTREME PROGRAMMING
I CAN'T GIVE YOU ALL OF THESE FEATURES IN THE FIRST VERSION.
AND EACH FEATURE NEEDS TO HAVE WHAT WE CALL A "USER STORY."
OKAY, HERE'S A STORY: YOU GIVE ME ALL OF MY FEATURES OR I'LL RUIN YOUR LIFE.
www.dilbert.com scottadams@aol.com
© 2002 United Feature Syndicate, Inc.

In Figure 6-10 we see how XP works in practice.

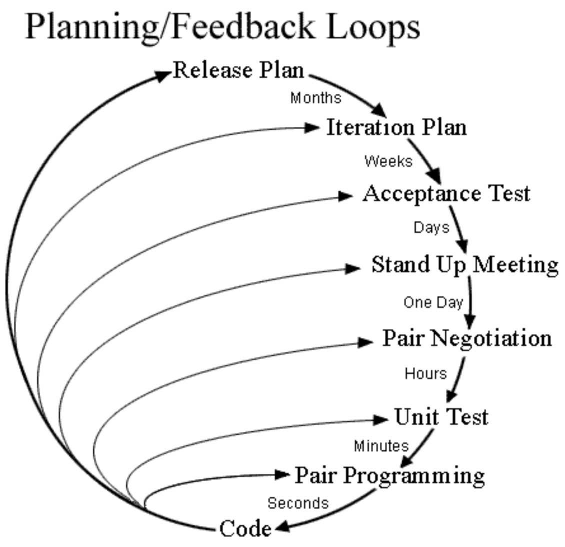


Figure 6-10: eXtreme Programming

In XP, they practice so-called "Pair Programming" (Figure 6-11), meaning 2 developers working together.

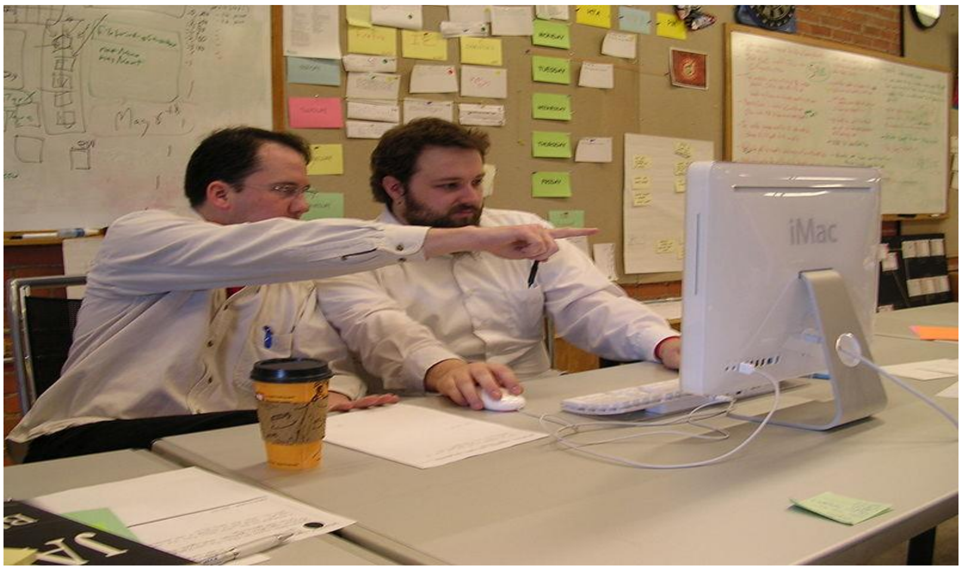


Figure 6-11: Pair Programming [8]

So, is Pair Programming Good or Bad? There exists various studies of the productivity of Pair Programming [1]:

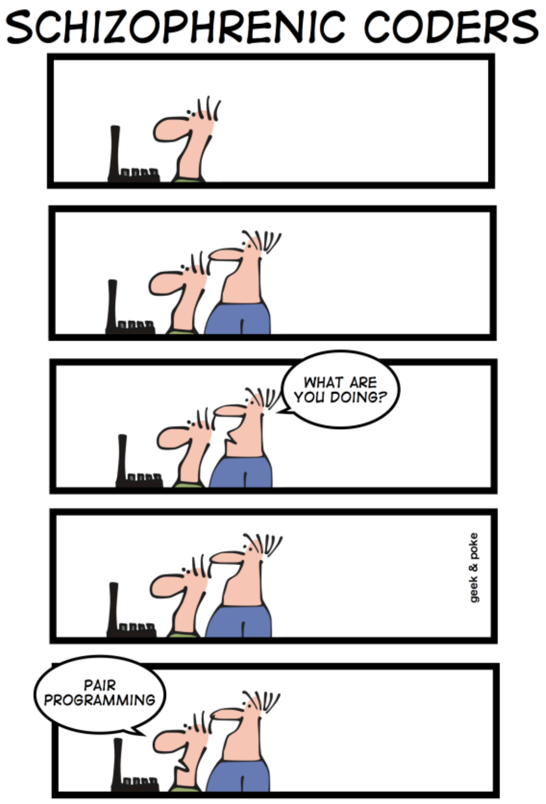
- Study 1: Comparable with that of 2 developers work independently
- Study 2: A significant loss in productivity compared with 2 developers working alone

A reasonable question is: Should the 2 developers have the same skills or not?

Newer less, there are benefits with XP:

- Collective Ownership for the code created and the results of the project.

- Continuous informal Review process because each code line is looked at by at least 2 people
- It supports Refactoring, which is a continuous process of software improvement
- Less time is spent on repairing bugs.
- Improved Code Quality
- It reduces the overall risk



6.2.3 Scrum

Scrum [9] is a so-called Agile method, and it has become very popular today. In Figure 6-12 we see an overview of the Scrum method.

Scrum is simple and easy to understand. The method is more flexible and more informal than plan-driven methods.