Software Engineering

Dr. Novarun Deb

Agile Software Development

Topics to be covered

Chapter 3: Software Engineering (Ian Sommerville)

- Agile methods
- Agile development techniques
- Agile project management
- Scaling agile methods

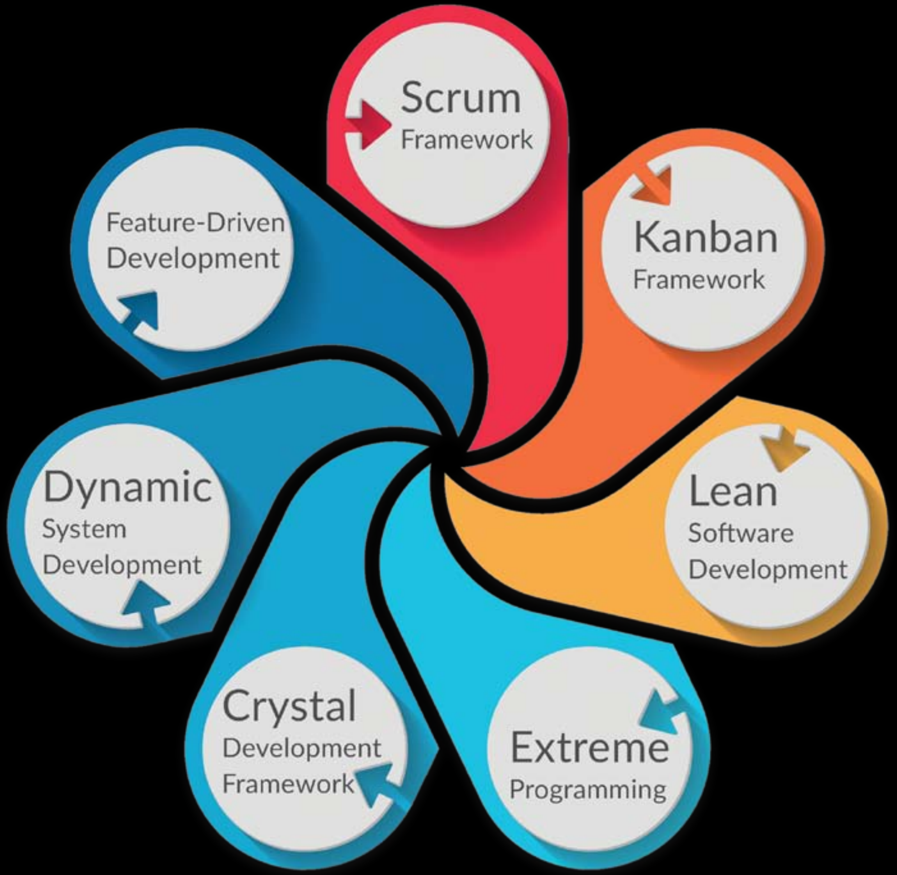
Rapid Software Development

Motivation

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast –changing environment and it is practically impossible to produce a set of stable software requirements.
  - Software has to evolve quickly to reflect changing business needs.

- Plan-driven development is essential for some types of system but does not meet these business needs.

- Agile development methods emerged in the late 1990s whose aim was to radically reduce the delivery time for working software systems.

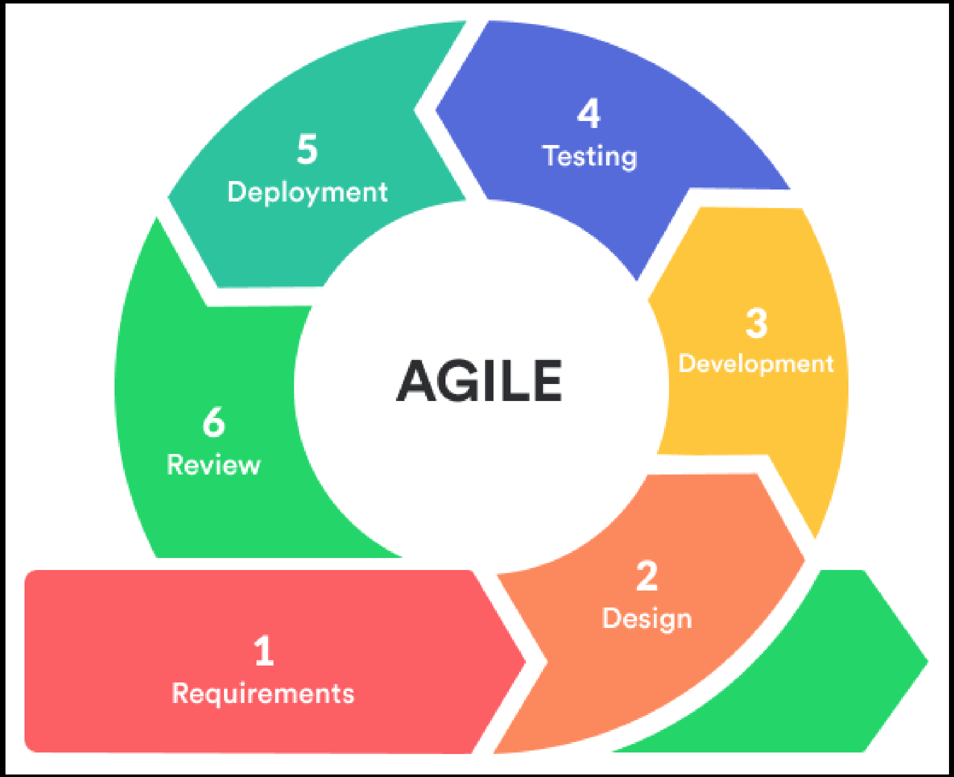


Scrum Framework
Kanban Framework
Lean Software Development
Extreme Programming
Crystal Development Framework
Dynamic System Development
Feature-Driven Development

Agile Development

Motivation

- Program specification, design and implementation are inter-leaved
- The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- Frequent delivery of new versions for evaluation
- Extensive tool support (e.g. automated testing tools) used to support development.
- Minimal documentation – focus on working code

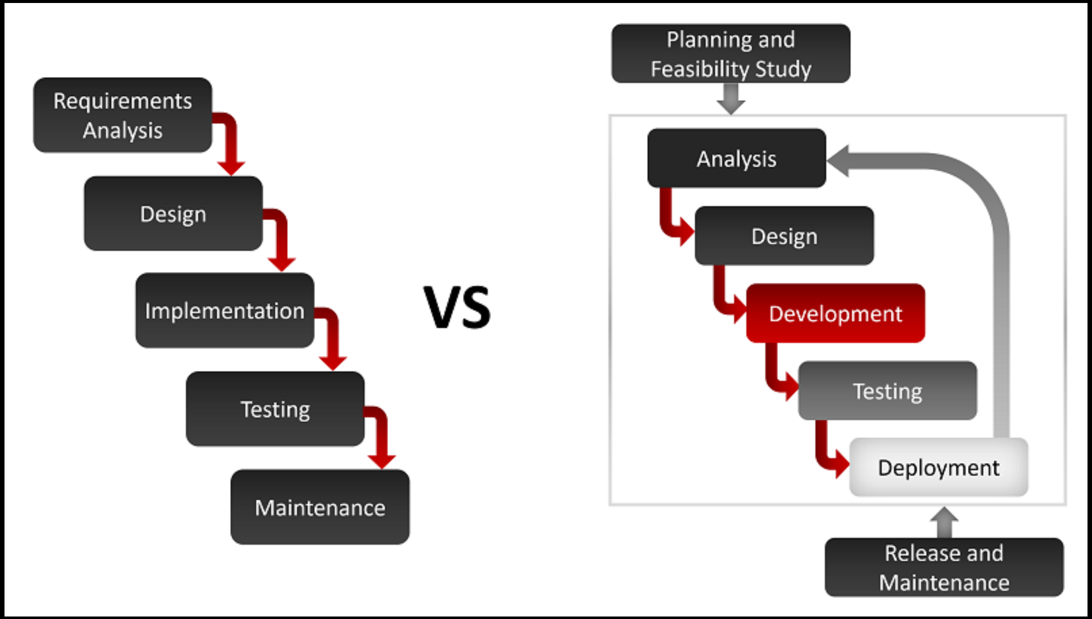


Plan-driven vs Agile development

Motivation

- Plan-driven development
  - A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
  - Not necessarily waterfall model – plan-driven, incremental development is possible
  - Iteration occurs within activities.

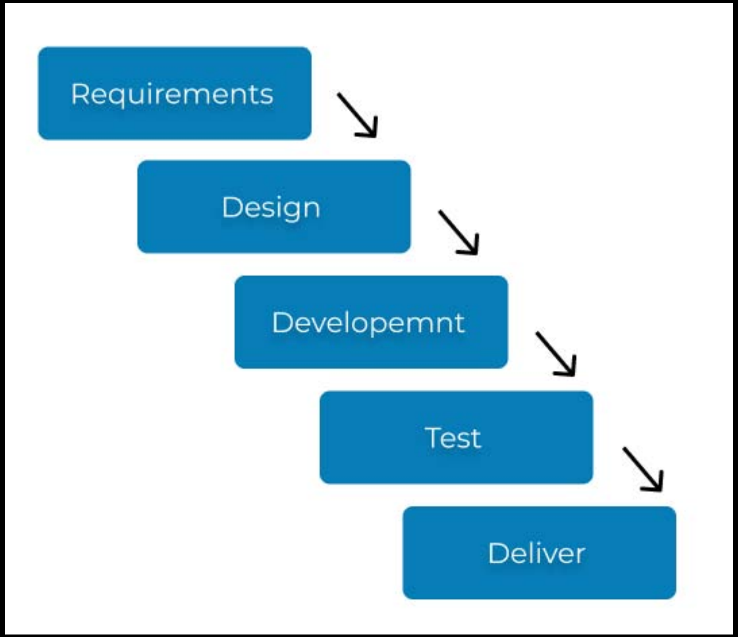
- Agile development
  - Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

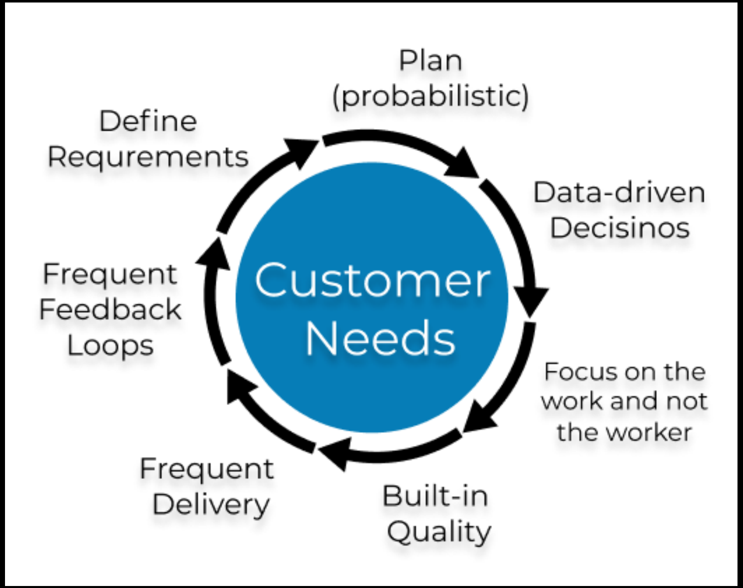


Agile Methods

Agile Methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s.
  - 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.
  - Reduce overheads in the software process (e.g. by limiting documentation)
  - Be able to respond quickly to changing requirements without excessive rework.





Agile Manifesto

- *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	*Processes and Tools*
Working software	*Comprehensive Documentation*
Customer collaboration	*Contract Negotiation*
Responding to change	*Following a Plan*

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

The Principles of Agile Methods

Principle 1

Customer Involvement

Description

Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.

The Principles of Agile Methods

Principle 2

Incremental delivery

Description

The software is developed in increments with the customer specifying the requirements to be included in each increment.

The Principles of Agile Methods

Principle 3	**Description**
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.

The Principles of Agile Methods

Principle 4	Description
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.

The Principles of Agile Methods

Principle 5

Maintain simplicity

Description

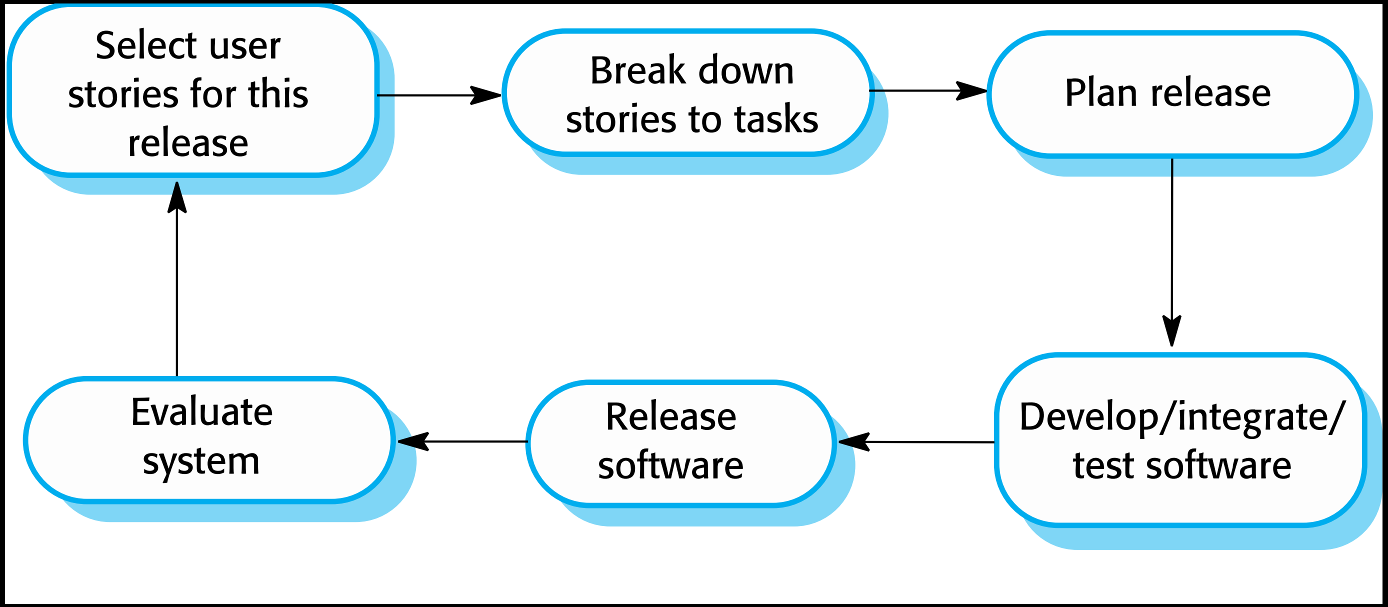
Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Agile Development Techniques

Extreme programming

Developed in the late 1990s

- 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.


Select user stories for this release
Break down stories to tasks
Plan release
Develop/integrate/ test software
Release software
Evaluate system

Extreme Programming Practices

Principle 1	**Description**
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development ‘Tasks’.

Extreme Programming Practices

Principle 2

Small releases

Description

The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.

Extreme Programming Practices

Principle 3

Simple design

Description

Enough design is carried out to meet the current requirements and no more.

Extreme Programming Practices

Principle 4	Description
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.

Extreme Programming Practices

Principle 5

Refactoring

Description

All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Extreme Programming Practices

Principle 6

Pair programming

Description

Developers work in pairs, checking each other’s work and providing the support to always do a good job.

Extreme Programming Practices

Principle 7

Collective ownership

Description

The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.

Extreme Programming Practices

Principle 8

Continuous integration

Description

As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.

Extreme Programming Practices

Principle 9

Sustainable pace

Description

Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity

Extreme Programming Practices

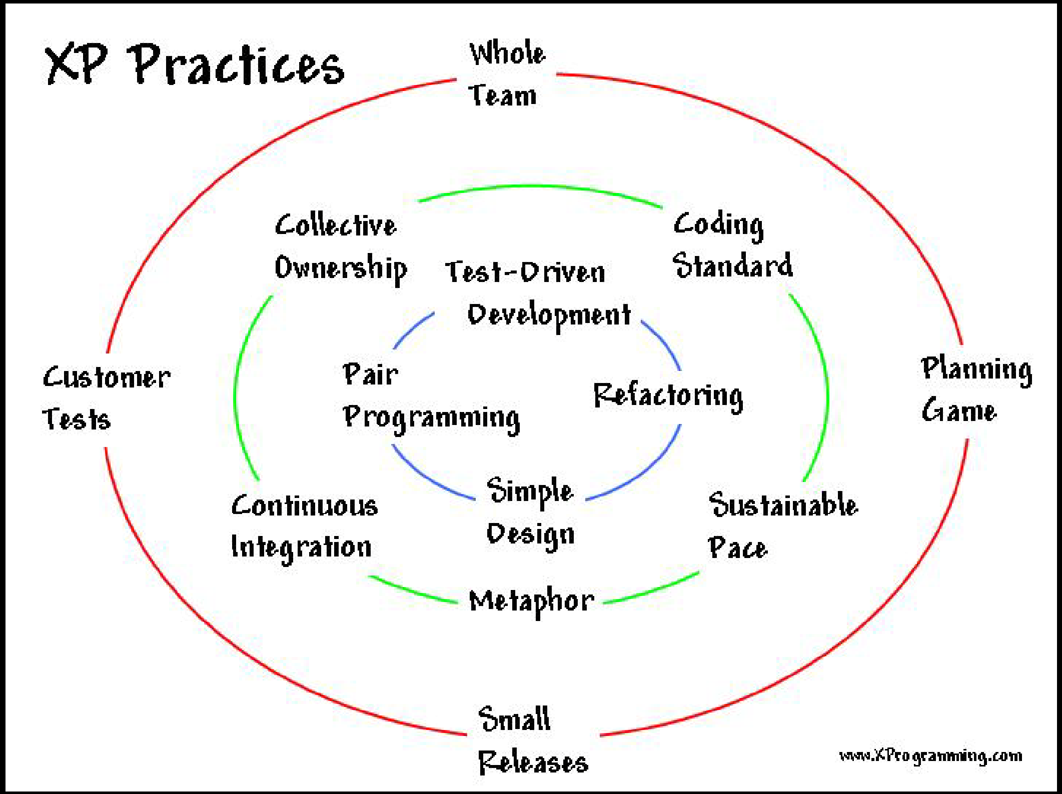
Principle 10	**Description**
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP and Agile Principles

Product characteristic	Description
Incremental development	Supported through small, frequent system releases.
Customer involvement	Full-time customer engagement with the team.
People not process	Pair programming, collective ownership and a process that avoids long working hours.
Change	Regular system releases.
Maintaining simplicity	Constant refactoring of code.

Influential XP Practices

- Extreme programming has a technical focus and is not easy to integrate with management practice in most organizations.
- Consequently, while agile development uses practices from XP, the method as originally defined is not widely used.
- Key practices
  - User stories for specification
  - Refactoring
  - Test-first development
  - Pair programming



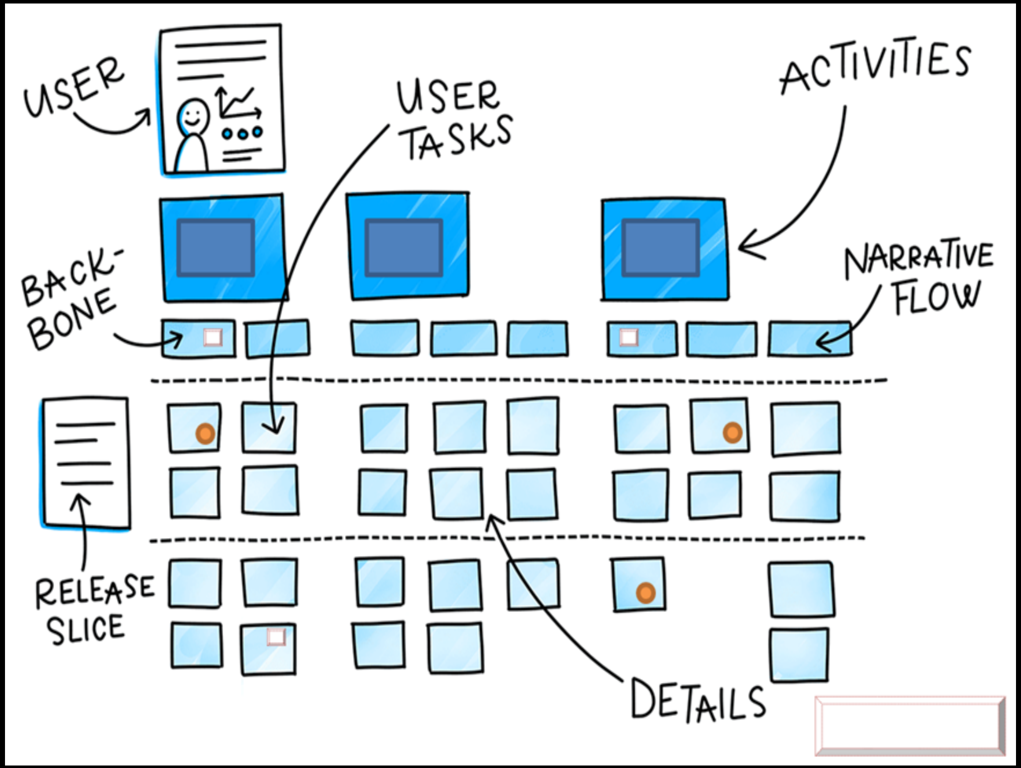
Agile Development Techniques

1. User Stories for Requirements

User Stories for Requirements

How they are created?

- In XP, a customer or user is part of the XP team and is responsible for making decisions on requirements.
- User requirements are expressed as user stories or scenarios.
- These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.



User Stories for Requirements

Example.

Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either 'current medication', 'new medication' or 'formulary'.

If you select 'current medication', you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

If you choose, 'new medication', the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose 'formulary', you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click 'OK' or 'Change'. If you click 'OK', your prescription will be recorded on the audit database. If you click 'Change', you reenter the 'Prescribing medication' process.

Task 1: Change dose of prescribed drug

Task 2: Formulary selection

Task 3: Dose checking

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.
Using the formulary id for the generic drug name, lookup the formulary and retrieve the recommended maximum and minimum dose.
Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low.
If within the range, enable the 'Confirm' button.

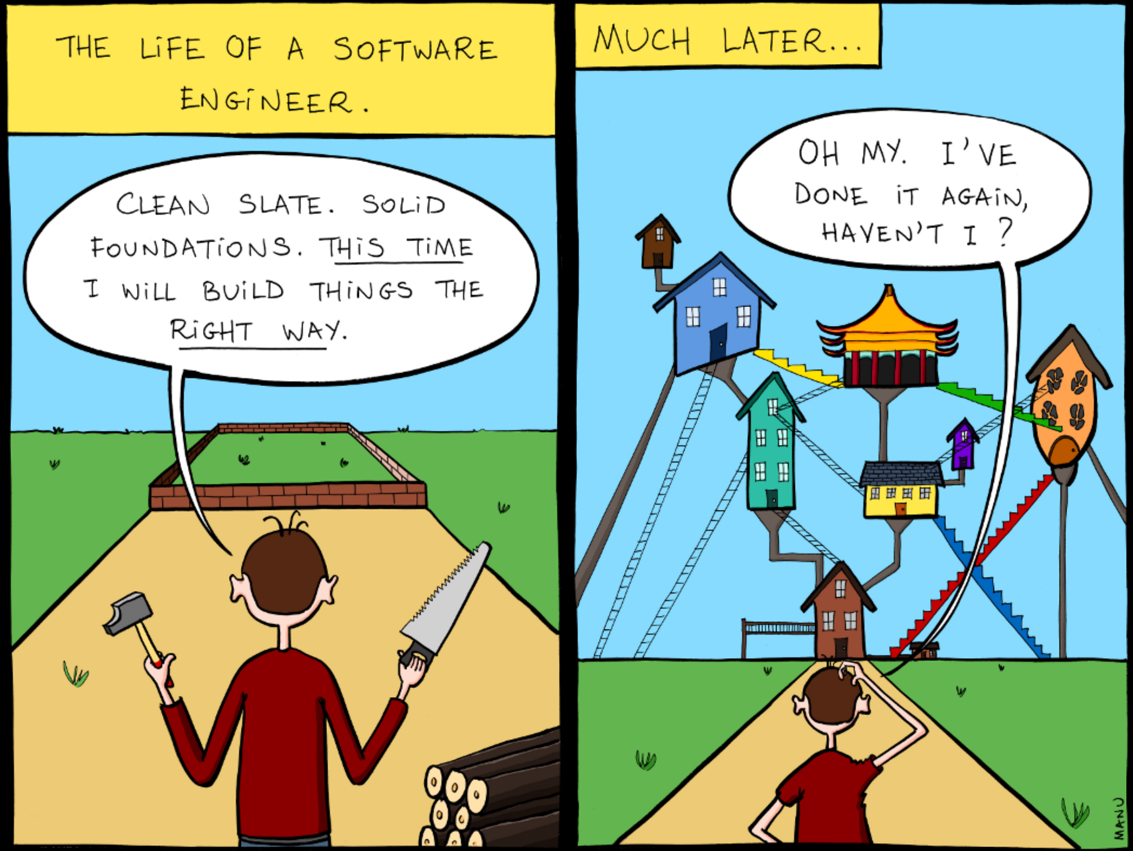
Agile Development Techniques

2. Refactoring

Refactoring

How and Why is it done?

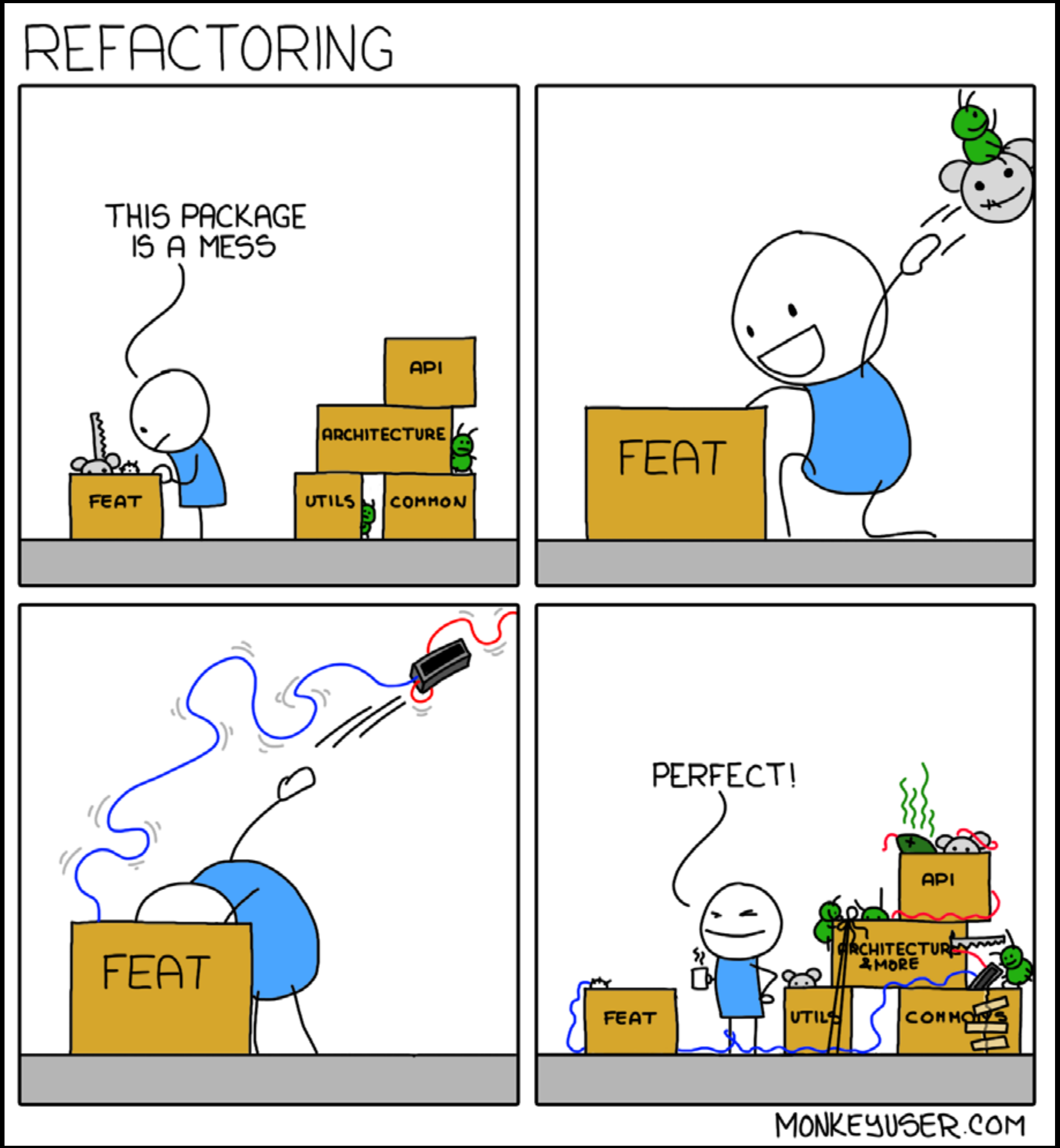
- XP maintains that it is not worthwhile spending time and effort anticipating changes as changes cannot be reliably anticipated.
    - Constant code improvement (refactoring) to make changes easier.
    - Because the code is well-structured and clear.
- Programming team look for possible software improvements
    - Improvements made even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- However, some changes requires architecture refactoring and this is much more expensive.


THE LIFE OF A SOFTWARE ENGINEER.
CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.
MUCH LATER...
OH MY. I'VE DONE IT AGAIN, HAVEN'T I ?
MANU

Refactoring

Examples.

- Re-organization 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.



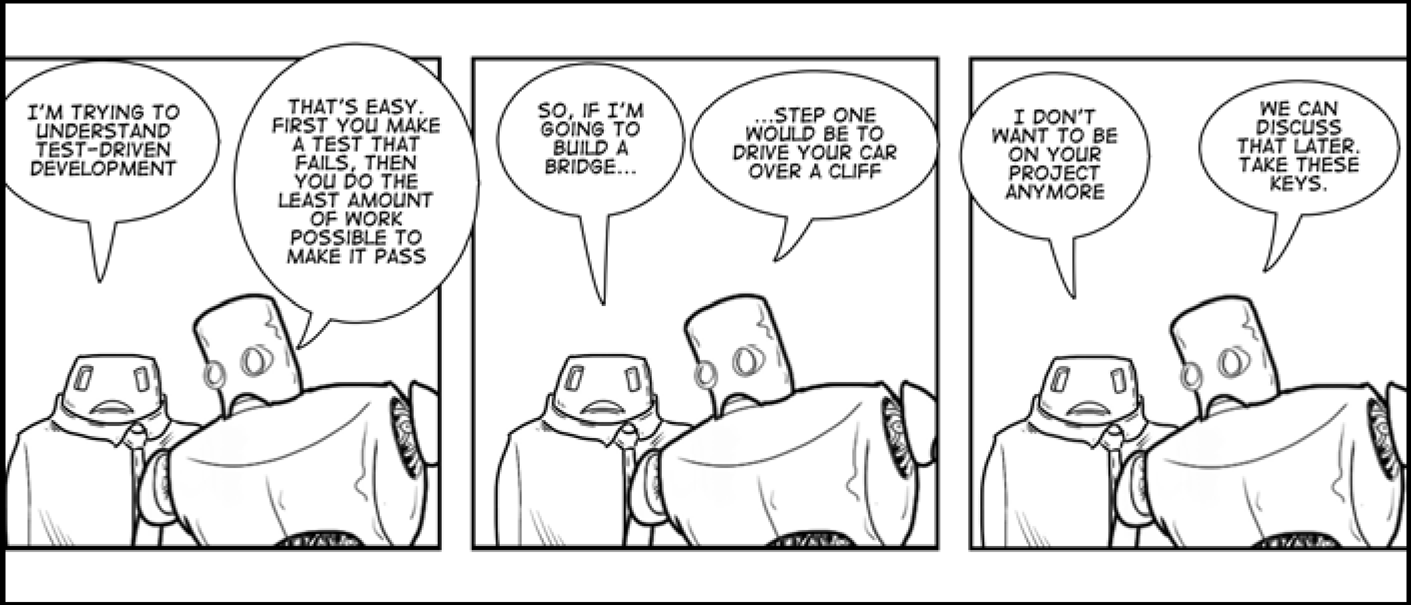
Agile Development Techniques

3. Test-first Development

Test-first development

What is it?

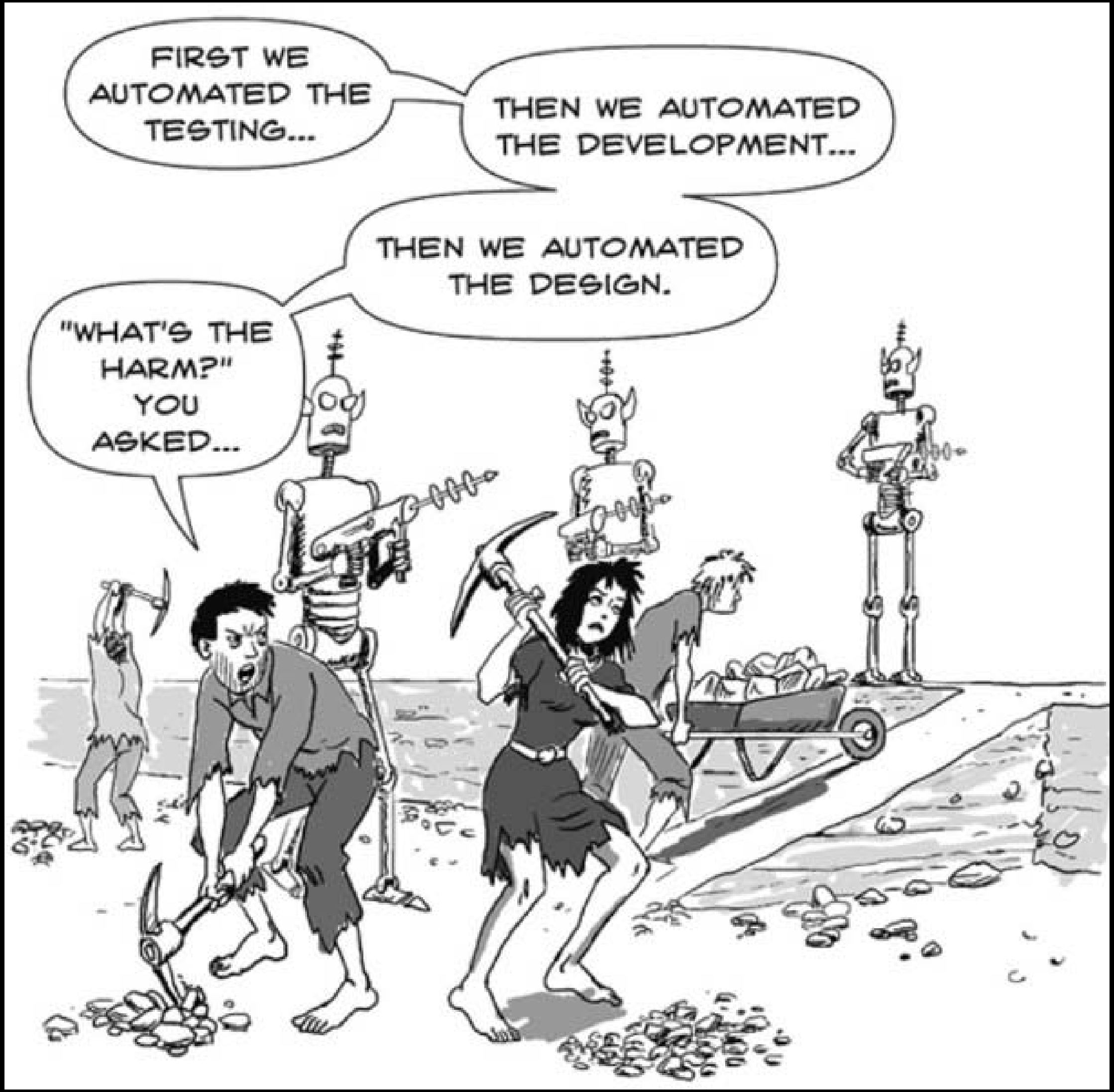
- Testing is central to XP and XP has developed an approach where the program is tested after every change has been made.
- XP testing features:
  - Incremental test development from scenarios.
  - User involvement in test development and validation.
  - Automated test harnesses are used to run all component tests each time that a new release is built.


I'M TRYING TO UNDERSTAND TEST-DRIVEN DEVELOPMENT
THAT'S EASY. FIRST YOU MAKE A TEST THAT FAILS, THEN YOU DO THE LEAST AMOUNT OF WORK POSSIBLE TO MAKE IT PASS
SO, IF I'M GOING TO BUILD A BRIDGE...
...STEP ONE WOULD BE TO DRIVE YOUR CAR OVER A CLIFF
I DON'T WANT TO BE ON YOUR PROJECT ANYMORE
WE CAN DISCUSS THAT LATER. TAKE THESE KEYS.

Test-first development

What is it?

- Writing tests before code clarifies the requirements to be implemented.
- Test automation means that tests are written as executable components before the task is implemented
  - Should be stand-alone, should simulate the submission of input to be tested and should check that the result meets the output specification.
  - An automated test framework (e.g. Junit) is a system that makes it easy to write executable tests and submit a set of tests for execution.
- As testing is automated, there is always a set of tests that can be quickly and easily executed
  - Whenever any functionality is added to the system, the tests can be run and problems that the new code has introduced can be caught immediately.


FIRST WE AUTOMATED THE TESTING...
THEN WE AUTOMATED THE DEVELOPMENT...
THEN WE AUTOMATED THE DESIGN.
"WHAT'S THE HARM?" YOU ASKED...

Agile Development Techniques

4. Pair Programming

Pair programming

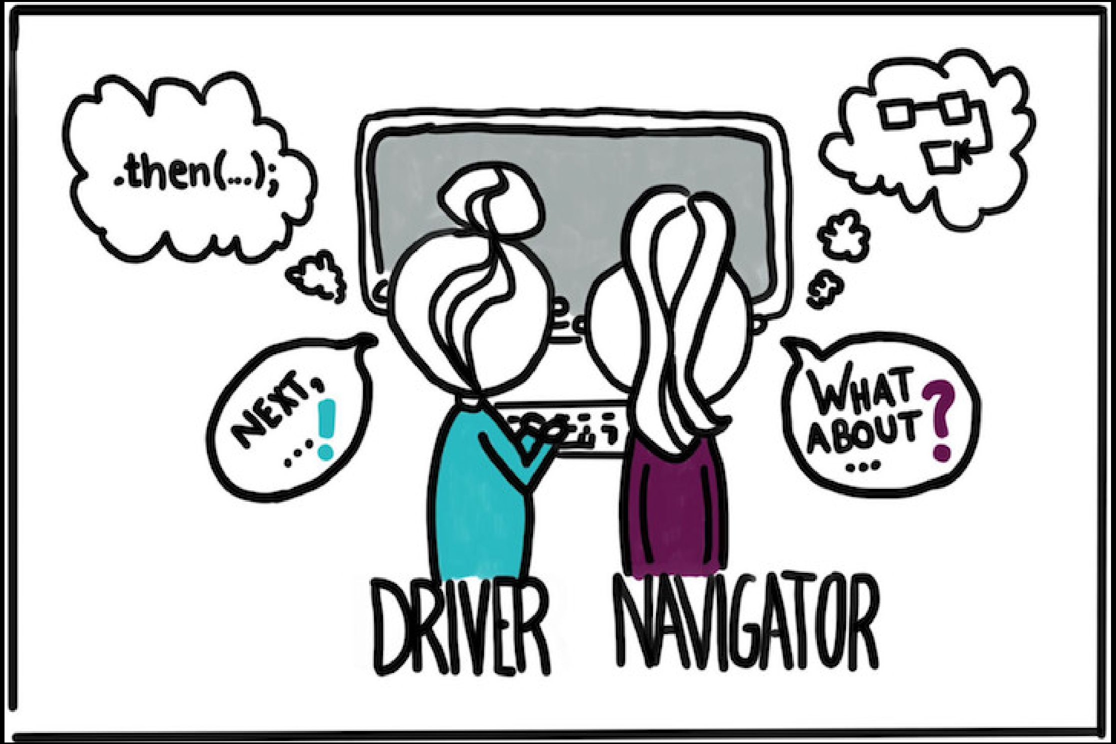
What is it?

- Pair programming involves programmers working in pairs, developing code together.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from improving the system code.

Pair programming

What is it?

- In pair programming, programmers sit together at the same computer to develop the software.
- Pairs are created dynamically so that all team members work with each other during the development process.
- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- Pair programming is not necessarily inefficient and there is some evidence that suggests that a pair working together is more efficient than 2 programmers working separately.



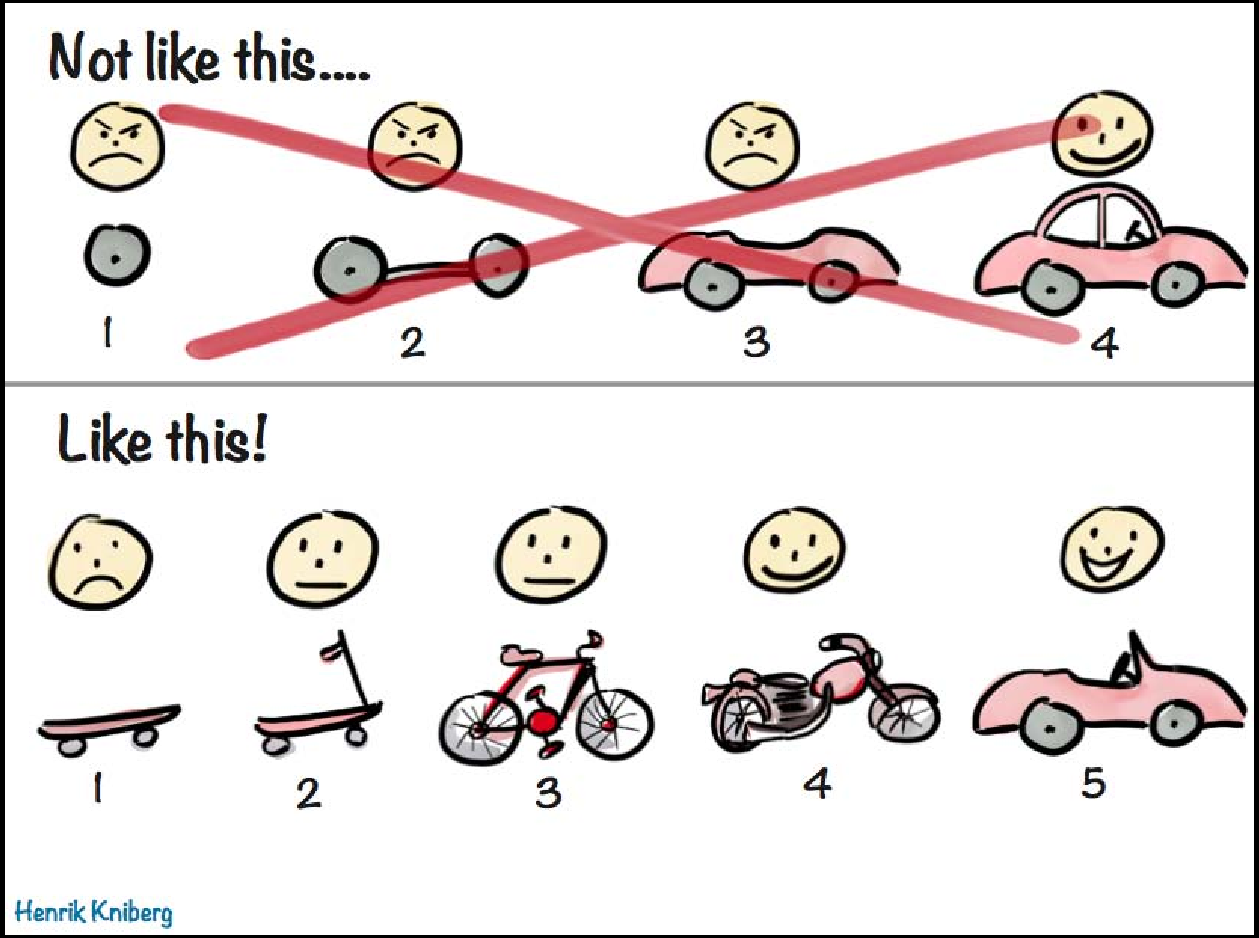
Agile Project Management

Agile Project Management

- The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.

- The standard approach to project management is plan-driven.

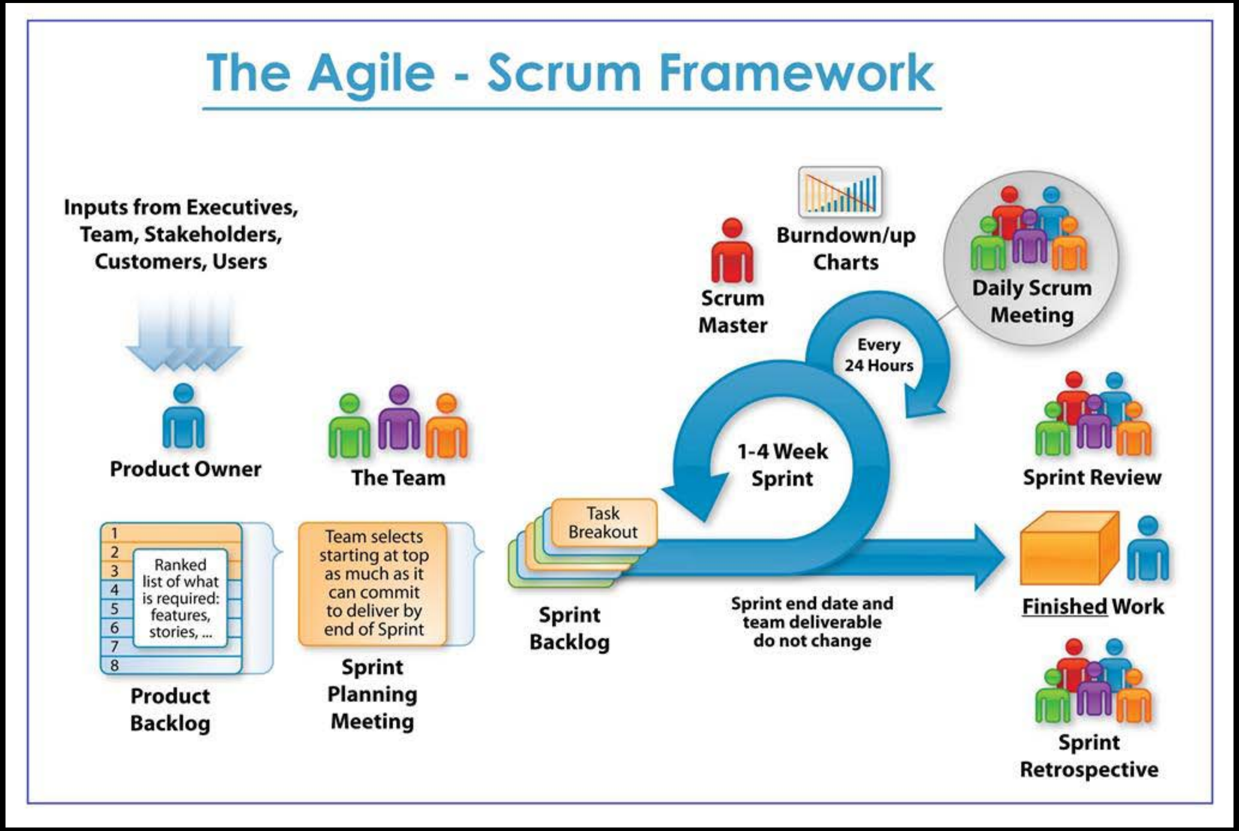
- Agile project management requires a different approach, which is adapted to incremental development and the practices used in agile methods.



Scrum

- Scrum is an agile method that focuses on managing iterative development rather than specific agile practices.

- There are three phases in Scrum.
  - The *initial phase* is an outline planning phase where you establish the general objectives for the project and design the software architecture.
  - This is followed by *a series of sprint cycles*, where each cycle develops an increment of the system.
  - The project *closure phase* wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.


The Agile - Scrum Framework
Inputs from Executives, Team, Stakeholders, Customers, Users
Product Owner
The Team
1
2
3
4
5
6
7
8
Ranked list of what is required: features, stories, ...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Scrum Master
Burndown/up Charts
Daily Scrum Meeting
Every 24 Hours
1-4 Week Sprint
Sprint end date and team deliverable do not change
Sprint Review
Finished Work
Sprint Retrospective

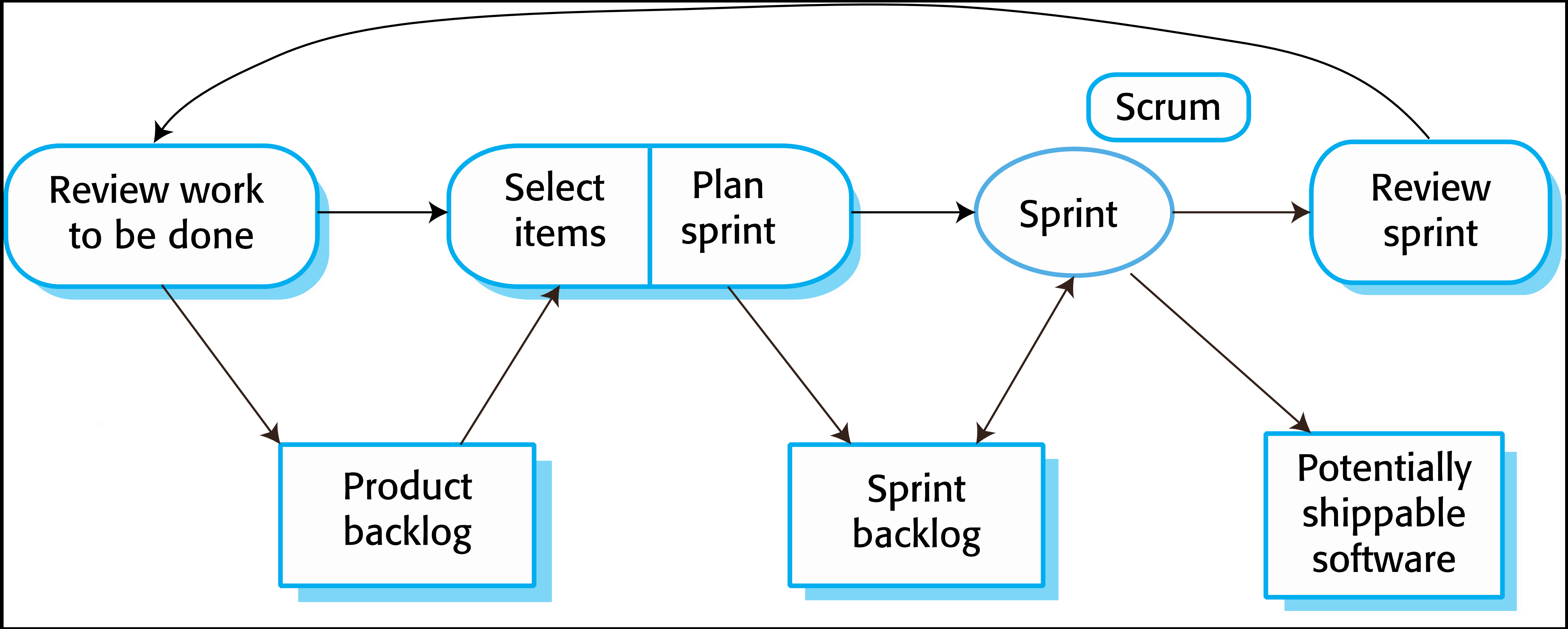
Scrum Terminology

Scrum term	Definition
Development team	A self-organizing group of software developers, which should be no more than 7 people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. The idea is that this should be 'potentially shippable' which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable.
Product backlog	This is a list of 'to do' items which the Scrum team must tackle. They may be feature definitions for the software, software requirements, user stories or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.
Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical business needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative.

Scrum Terminology

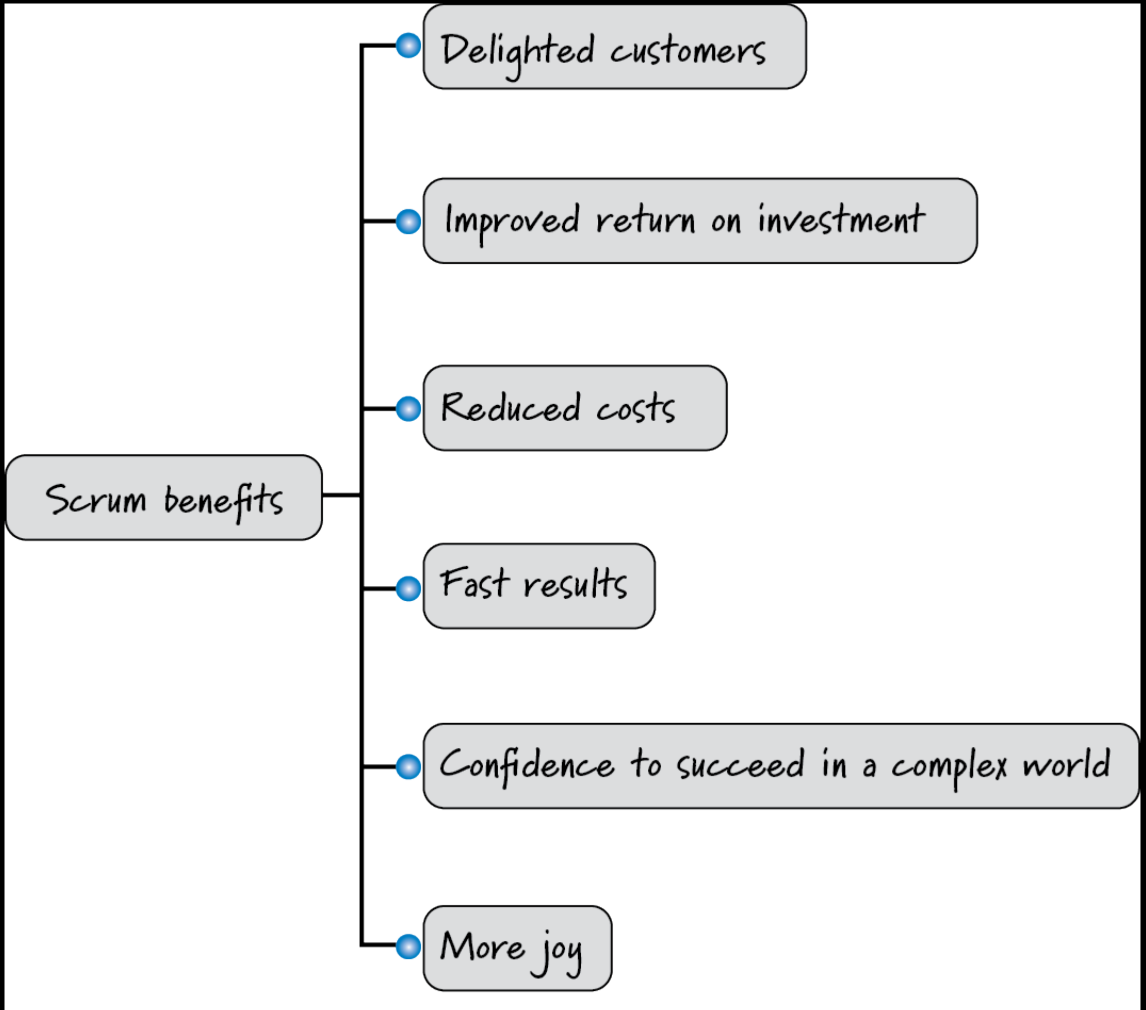
Scrum term	Definition
Scrum	A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.
Scrum Master	The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The Scrum developers are adamant that the ScrumMaster should not be thought of as a project manager. Others, however, may not always find it easy to see the difference.
Sprint	A development iteration. Sprints are usually 2-4 weeks long.
Velocity	An estimate of how much product backlog effort that a team can cover in a single sprint. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.

Scrum Terminology



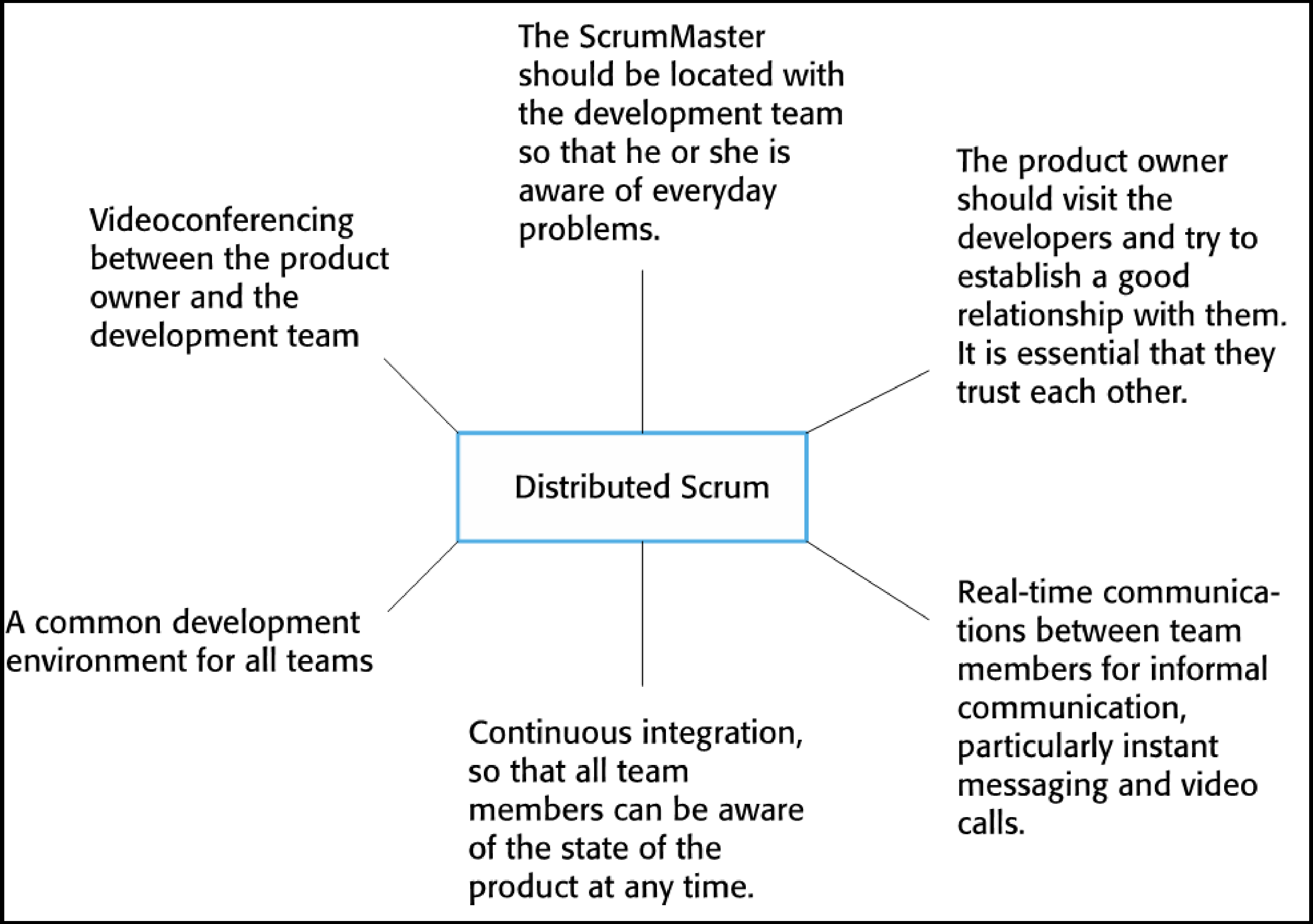
Scrum Benefits

- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.


Scrum benefits
Delighted customers
Improved return on investment
Reduced costs
Fast results
Confidence to succeed in a complex world
More joy

Copyright © 2012, Kenneth S. Rubin and Innolution, LLC. All Rights Reserved.

Distributed Scrum


The ScrumMaster should be located with the development team so that he or she is aware of everyday problems.
Videoconferencing between the product owner and the development team
The product owner should visit the developers and try to establish a good relationship with them. It is essential that they trust each other.
Distributed Scrum
A common development environment for all teams
Continuous integration, so that all team members can be aware of the state of the product at any time.
Real-time communica-tions between team members for informal communication, particularly instant messaging and video calls.

Scaling Agile Methods

Scaling Agile Methods

- Agile methods have proved to be successful for small and medium sized projects that can be developed by a small co-located team.
- It is sometimes argued that the success of these methods comes because of improved communications which is possible when everyone is working together.
- Scaling up agile methods involves changing these to cope with larger, longer projects where there are multiple development teams, perhaps working in different locations.

- 'Scaling up' is concerned with using agile methods for developing large software systems that cannot be developed by a small team.
- 'Scaling out' is concerned with how agile methods can be introduced across a large organization with many years of software development experience.
- When scaling agile methods it is important to maintain agile fundamentals:
  - Flexible planning, frequent system releases, continuous integration, test-driven development and good team communications.

Practical problems with Agile methods

- The informality of agile development is incompatible with the legal approach to contract definition that is commonly used in large companies.
- Agile methods are most appropriate for new software development rather than software maintenance. Yet the majority of software costs in large companies come from maintaining their existing software systems.
- Agile methods are designed for small co-located teams yet much software development now involves worldwide distributed teams.

Question: How do you scale up the Agile method of Software development?