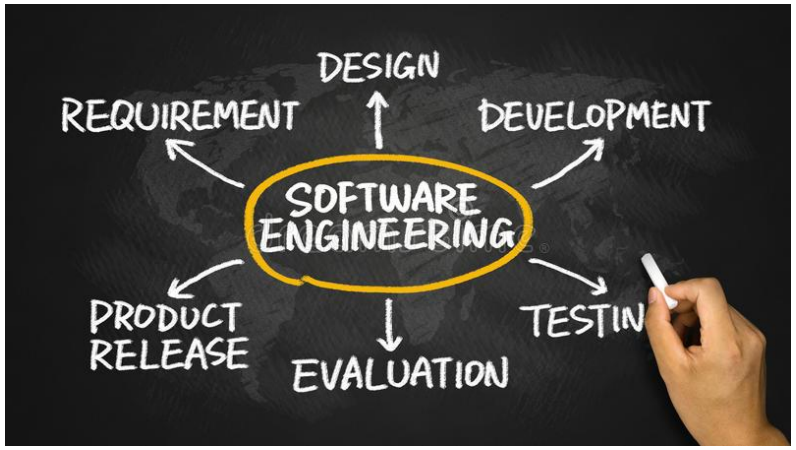
CSE241/CMM341
Foundations of Software Engineering



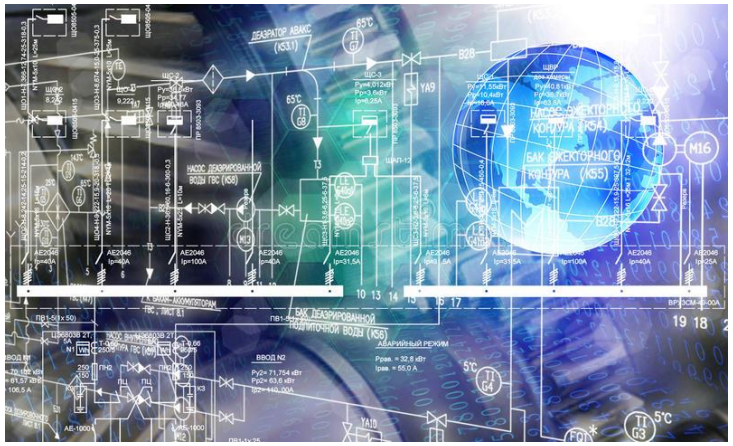


Photo credit: Dreamstime.com

Topic 3: Agile Software Development

CSE241 Course Content

We lead

Foundations of Software Engineering

1. Introduction to Software Engineering
2. Software Processes
3. **Agile Software Development**
4. Requirements Engineering
5. System Modeling
6. Software Design Strategies and Methods
7. Architectural Design & Implementation
8. Software Testing
9. Software Evolution
10. Project Management
11. Project planning
12. Quality Management
13. Configuration Management

Topic 3: Agile Software Development

Contents

AGILE METHODS

AGILE DEVELOPMENT TECHNIQUES

AGILE PROJECT MANAGEMENT

SCALING AGILE METHODS

Learning Outcomes

Understand the rationale for agile software development methods, the agile manifesto, and the differences between agile and plan-driven development

Know about important agile development practices such as user stories, refactoring, pair programming and test-first development

Understand the Scrum approach to agile project management

Understand the issues of scaling agile development methods and combining agile approaches with plan-driven approaches in the development of large software systems.

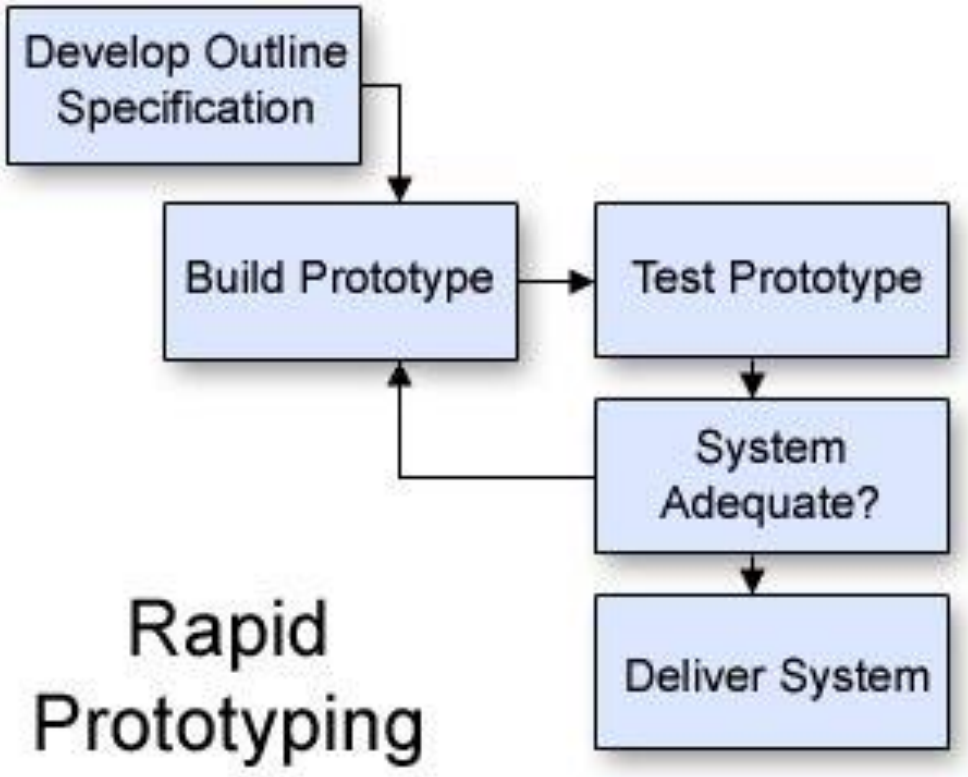
Recall Topic 2……

Categorization of Software Processes

- **Plan-driven processes**
  - all of the process activities are planned in advance
  - progress is measured against the plan
- **Agile processes**
  - planning is incremental
  - easier to change the process to reflect changing customer requirements
  - RAPID SOFTWARE DEVELOPMENT

Rapid Software Development

https://www.youtube.com/watch?v=Z9QbYZh1YXY



- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast – changing requirement 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 aim was to radically reduce the delivery time for working software systems

Agile Methods Characteristics

The processes of specification, design and implementation are interleaved.

- no detailed system specification
- design documentation is minimized or generated automatically by the programming environment.
- user requirements document : an outline definition of the most important characteristics of the system.

Agile Methods Characteristics

The system is developed in a series of increments.

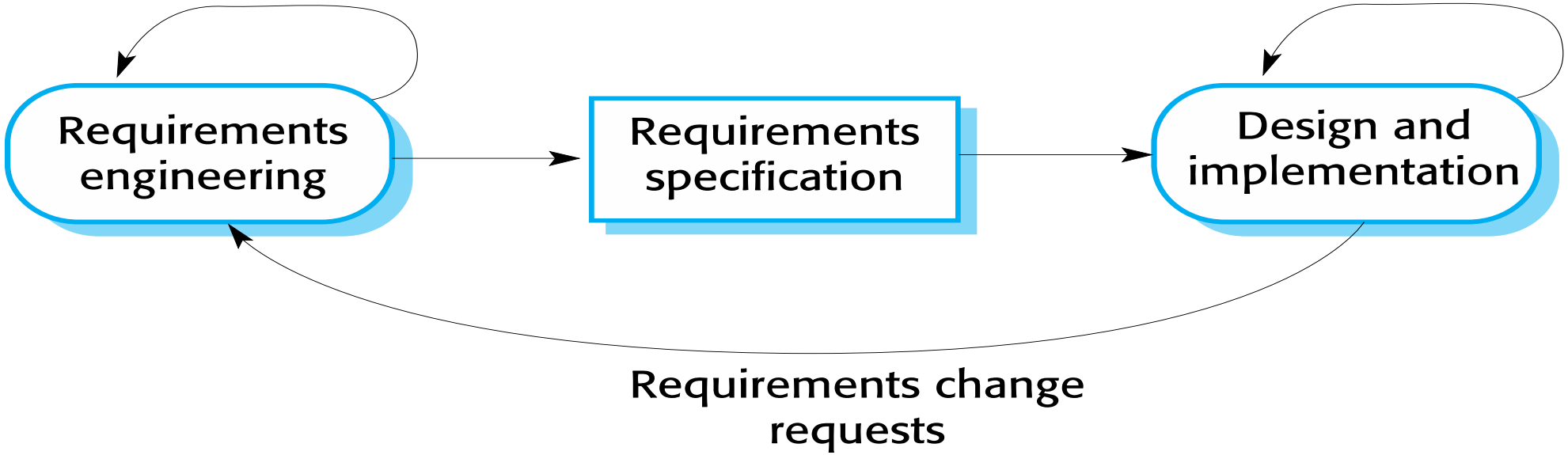
- End-users and other system stakeholders are involved in specifying and evaluating each increment
- Proposed changes to the software and new requirements should be implemented in a later version of the system.

Agile Methods Characteristics

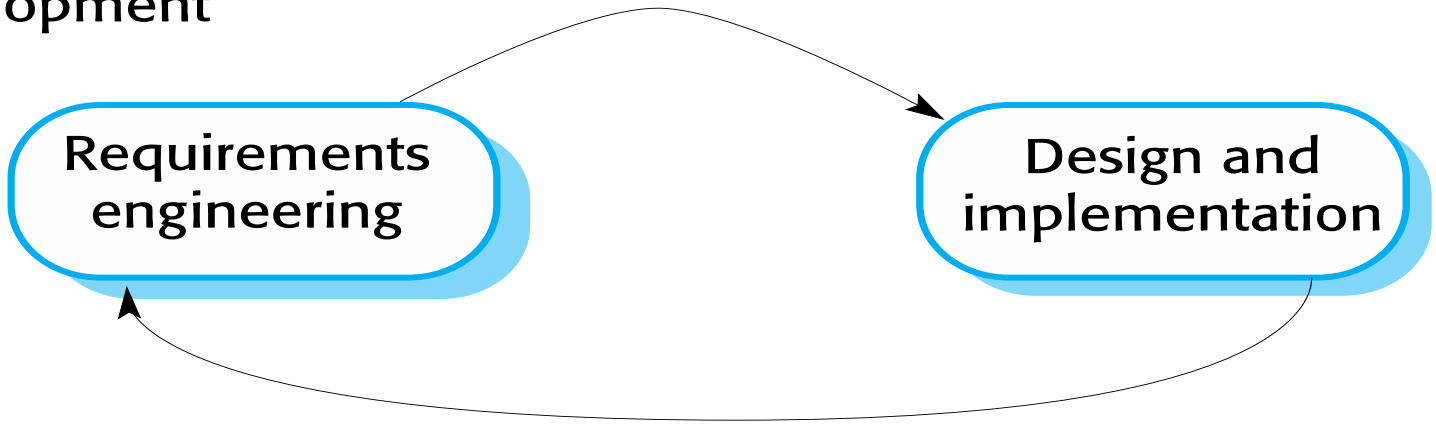
Extensive tool support is used to support the development process.

- automated testing tools
- tools to support configuration management and system integration
- tools to automate user interface production

Plan-based development



Agile development



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.

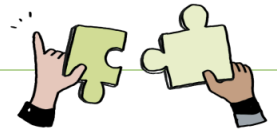
Agile Manifesto

Agile Values

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**



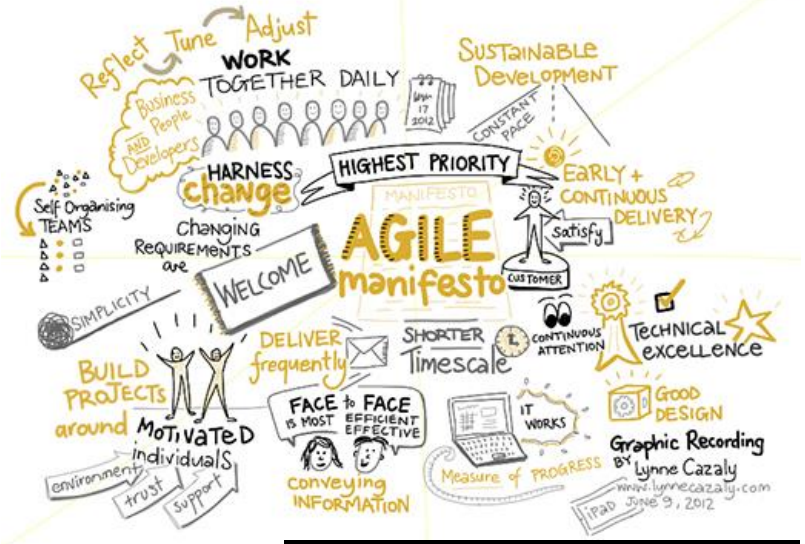
Customer collaboration over **contract negotiation**

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**.

https://agilemanifesto.org/

@SketchingSM
www.sketchingscrummaster.com



Agile Manifesto

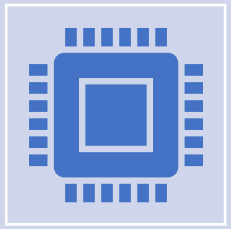
The Principles of Agile Methods

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is to provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
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

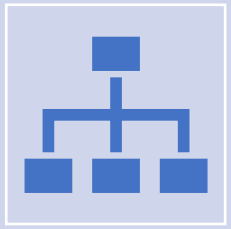
Principle	Description
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Agile method applicability



Product development where a software company is developing a small or medium-sized product for sale.

Virtually all software products and apps are now developed using an agile approach



Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are few external rules and regulations that affect the software.

Topic 3: Agile Software Development

Contents

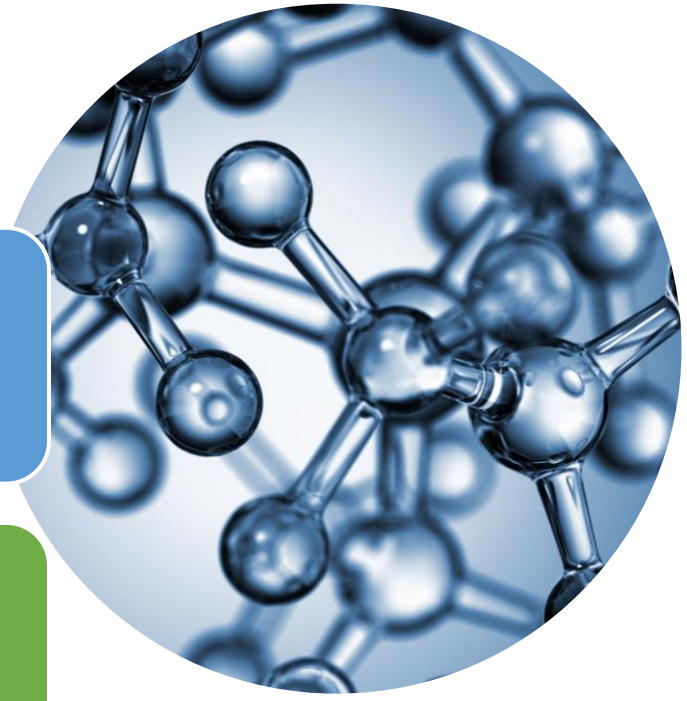
AGILE METHODS

AGILE DEVELOPMENT TECHNIQUES

AGILE PROJECT MANAGEMENT

SCALING AGILE METHODS

Extreme Programming

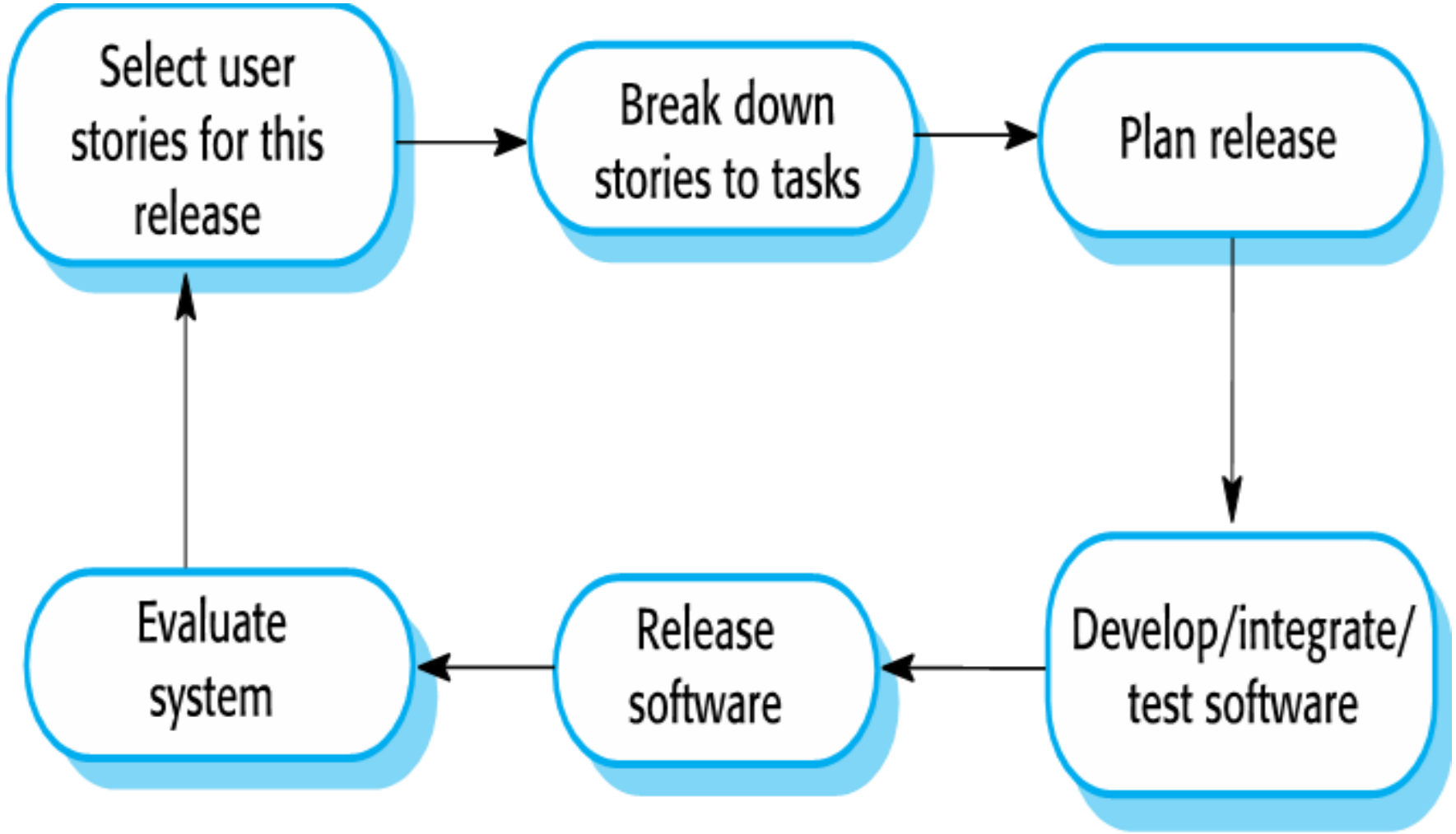


A very influential agile method, developed in the late 1990s, that introduced a range of agile development techniques.

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.

The extreme programming release cycle



Extreme programming practices

Principle or practice	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’.
Small releases	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.
Simple design	Enough design is carried out to meet the current requirements and no more.

Extreme programming practices

Principle or practice	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.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	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 or practice	Description
Continuous integration	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 pas
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
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 (recap)

Incremental development is supported through small, frequent system releases.

Customer involvement means continuous customer engagement with the team.

People, not process, are supported through pair programming, collective ownership and a process that avoids long working hours.

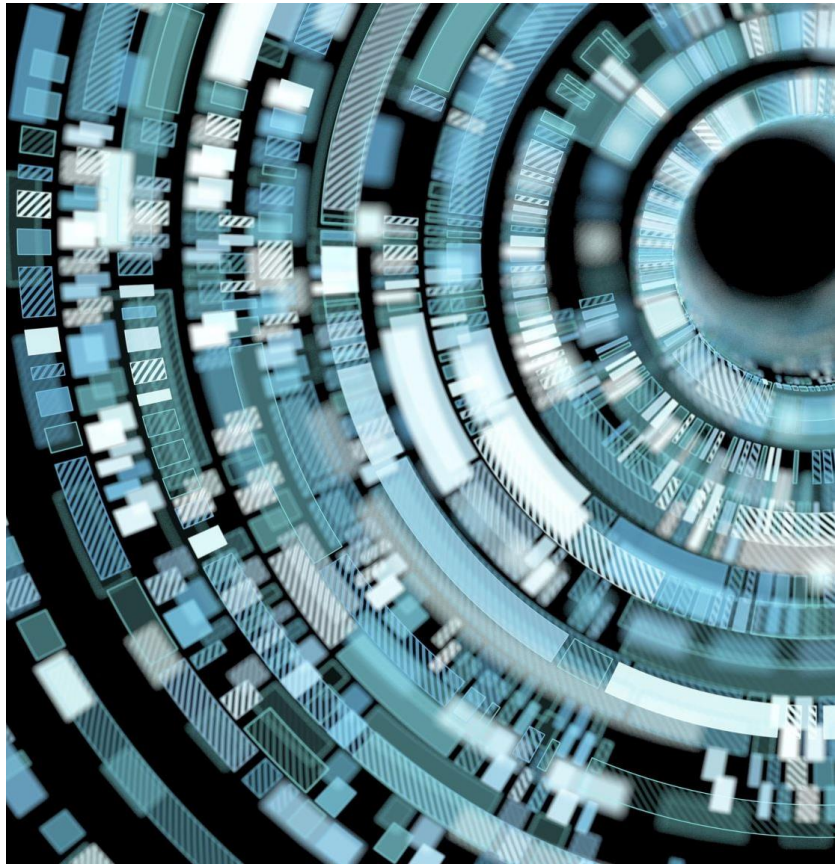
Change supported through regular system releases to customers, test-first development, refactoring to avoid code degeneration, and continuous integration

of new functionality.

Maintaining simplicity through constant refactoring of code to improves code quality and by using simple designs

Key XP practices

- User stories for specification
- Refactoring
- Test-first development (TDD)
- Pair programming



User stories for requirements

https://youtu.be/LGeDZmrWwsw

A customer or user

- part of the XP team
- responsible for making decisions on requirements.

User requirements

- expressed as user stories or scenarios.
- 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.

Refactoring

https://youtu.be/LsLniadcRTw

- 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.

Examples of Refactoring

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.

Test-first development

An approach where the program is tested after every change has been made.

XP testing features:

- Test-first development.
- 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.

https://youtu.be/uGaNkTahrIw

Test-driven development (TDD)

- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
  - Usually relies on a testing framework such as Junit.
- All previous and new tests are run automatically when new functionality is added, thus checking that the new functionality has not introduced errors.

Customer involvement in TDD

To help develop acceptance tests for the stories that are to be implemented in the next release of the system.

Writes tests together with team as development proceeds.

All new code is validated to ensure that it is what the customer needs.

However, in real life customer have limited time available and so cannot work full-time with the development team. Only provides the requirements and not involved in the testing process.

https://blog.hubspot.com/service/customer-orientation

Test Automation

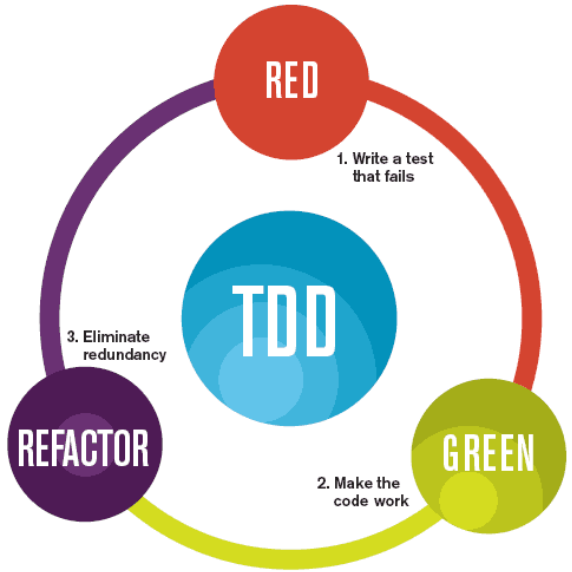
- Essential for test-first development
- Tests are written as executable components before the task is implemented
- Testing components should :
  - stand-alone
  - simulate the submission of input to be tested
  - check that the result meets the output specification
- An automated test framework is a system that makes it easy to write executable tests and submit a set of tests for execution. (e.g. Junit)
- 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.

Problems with test-first development

Programmers prefer programming to testing and sometimes they take short cuts when writing tests. For example, they may write incomplete tests that do not check for all possible exceptions that may occur.

Some tests can be very difficult to write incrementally. For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the 'display logic' and workflow between screens.

It difficult to judge the completeness of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage.



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Pair Programming

https://youtu.be/ET3Q6zNK3Io

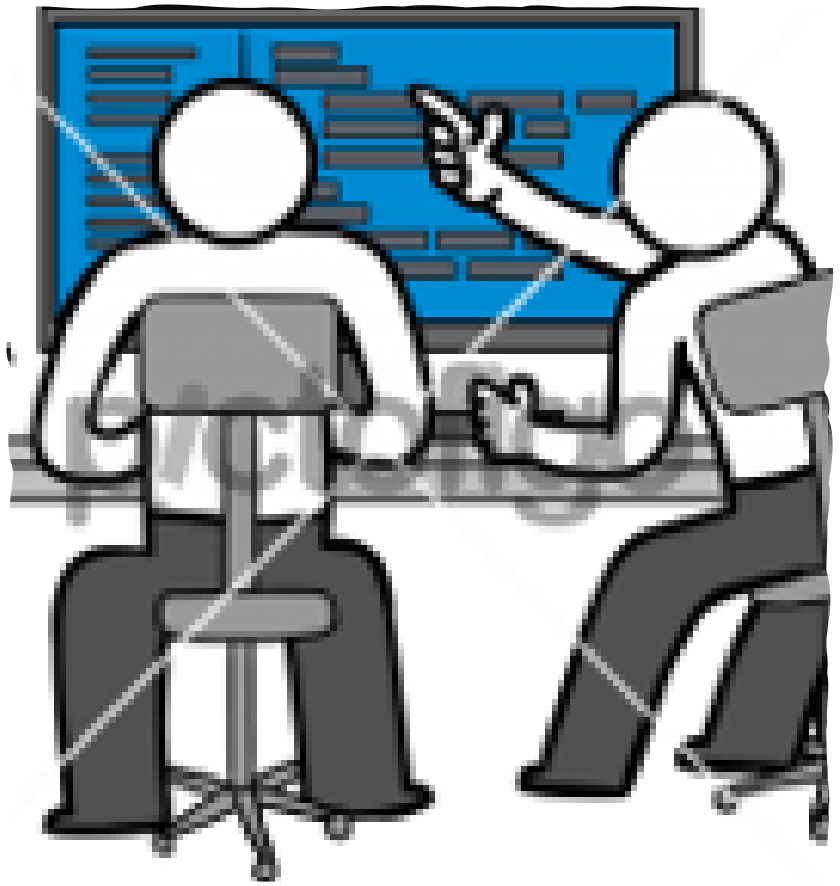
programmers working in pairs, developing code together.

develop common ownership of code and spreads knowledge across the team.

serves as an informal review process as each line of code is looked at by more than 1 person.

encourages refactoring as the whole team can benefit from improving the system code.

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.

Topic 3: Agile Software Development

Contents

AGILE METHODS

AGILE DEVELOPMENT TECHNIQUES

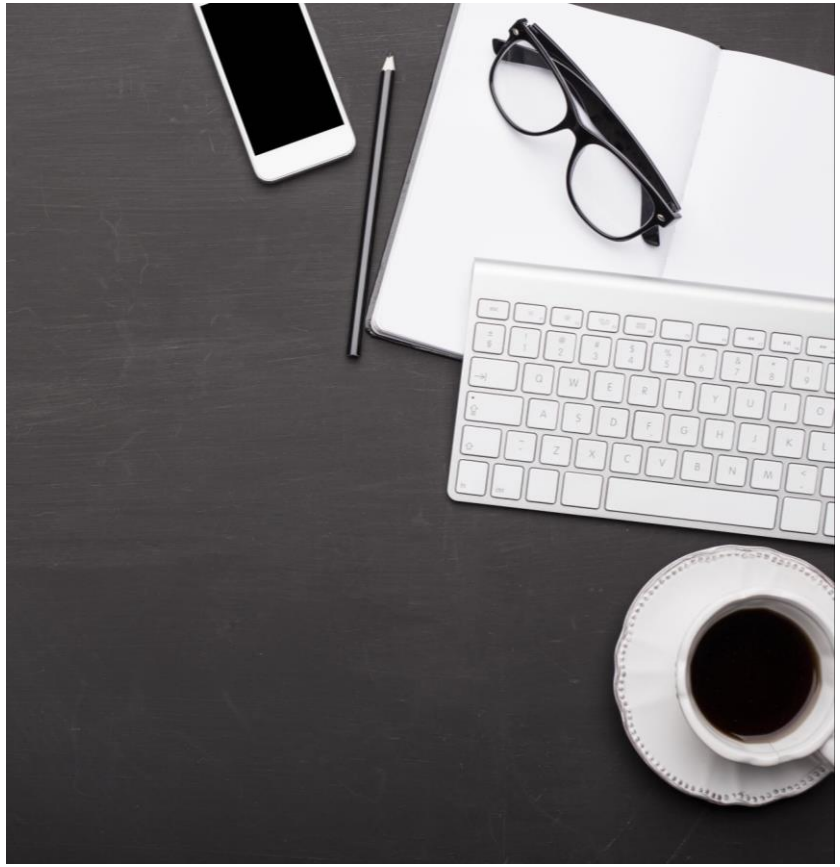
AGILE PROJECT MANAGEMENT

SCALING AGILE METHODS

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.

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



Scrum

https://youtu.be/9TycLR0TqFA

- An agile method that focuses on managing iterative development rather than specific agile practices.

- There are three phases in Scrum.

An outline planning phase where you establish the general objectives for the project and design the software architecture.

A series of sprint cycles, where each cycle develops an increment of the system.

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.

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.

Scrum Terminology

Scrum term	Definition
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.

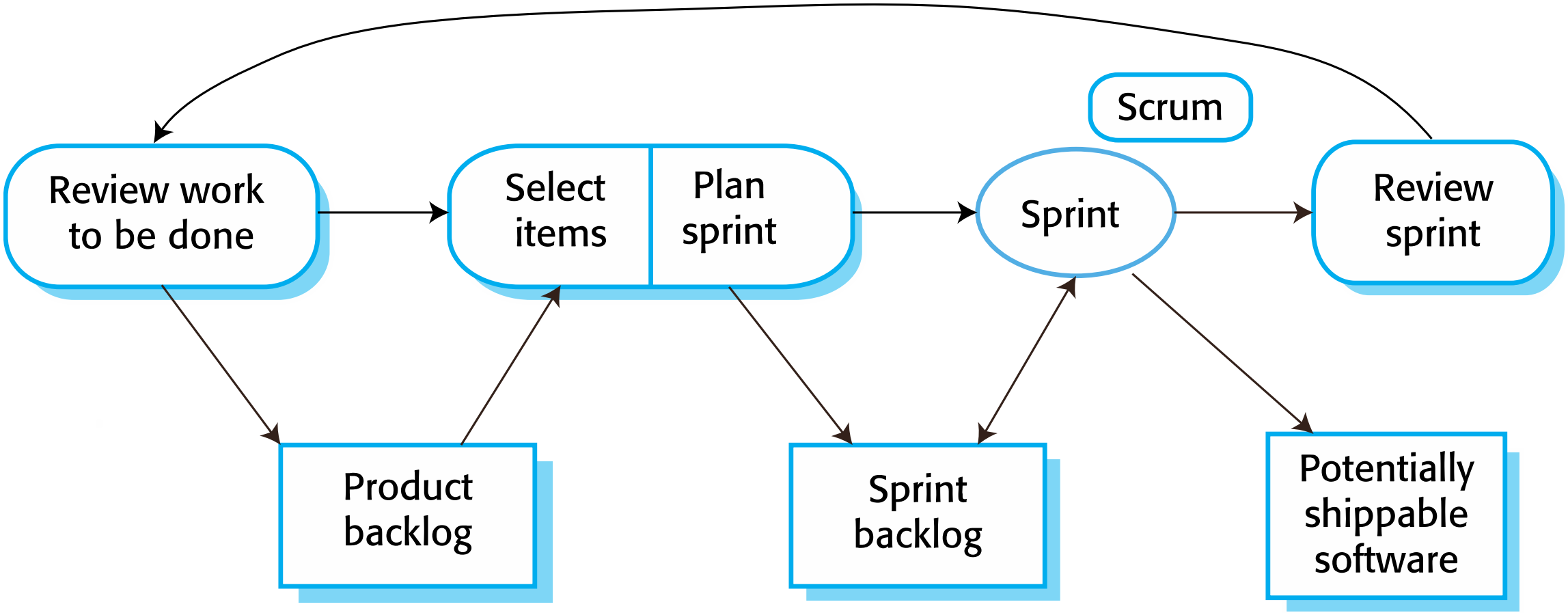
Scrum terminology

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.

Scrum terminology

Scrum term	Definition
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 sprint cycle



The Scrum sprint cycle

Sprints are fixed length, normally 2–4 weeks.

The starting point for planning is the product backlog, which is the list of work to be done on the project.

The selection phase involves all of the project team who work with the customer to select the features and functionality from the product backlog to be developed during the sprint.

The Sprint cycle

Once these are agreed, the team organize themselves to develop the software.

During this stage the team is isolated from the customer and the organization, with all communications channelled through the so-called ‘Scrum master’.

The role of the Scrum master is to protect the development team from external distractions.

At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.

Teamwork in Scrum

The ‘Scrum master’ is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.

The whole team attends short daily meetings (Scrums) where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.

Scrum benefits

The product is broken down into a set of manageable and understandable chunks.

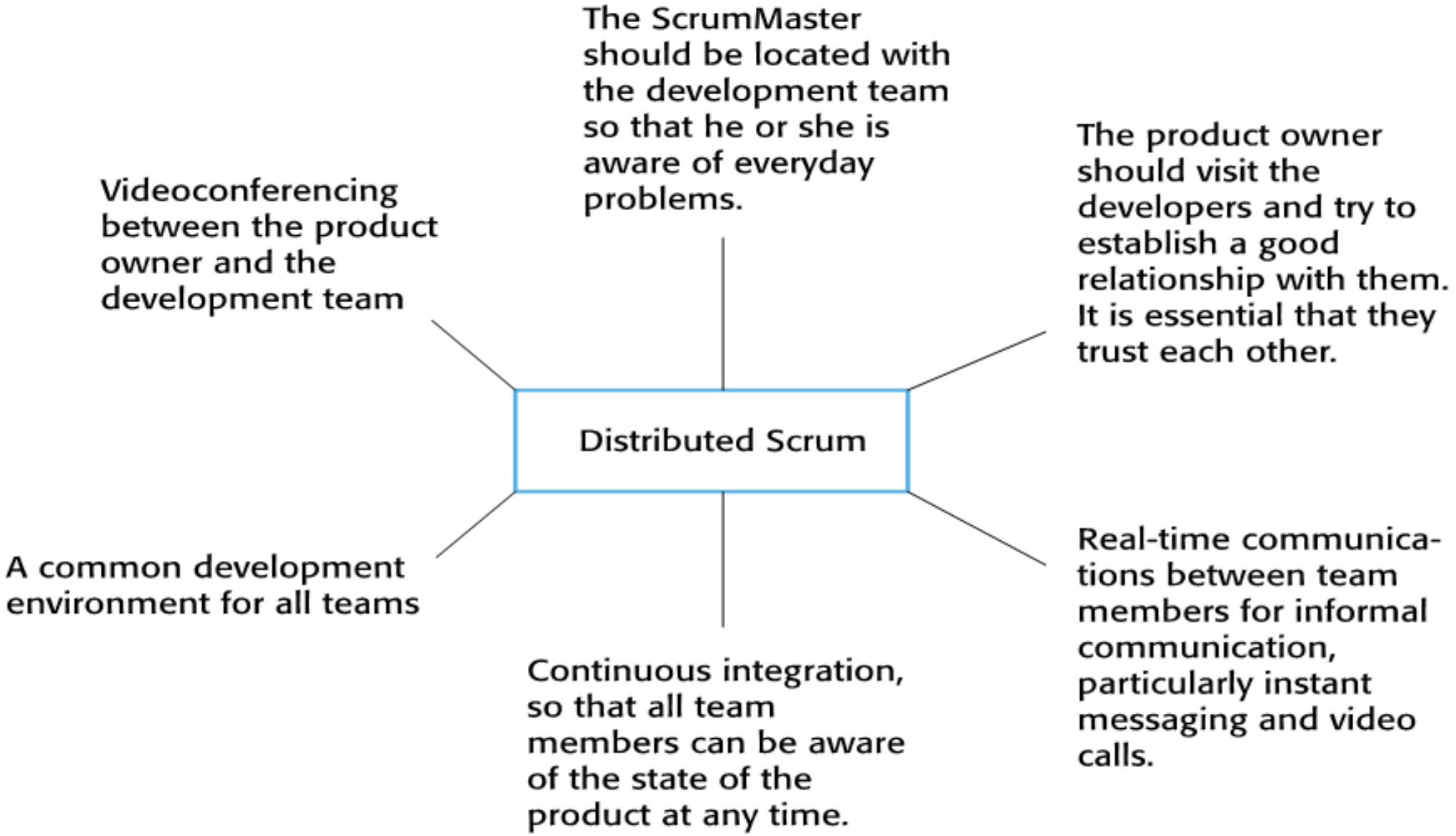
Unstable requirements do not hold up progress.

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.

Distributed Scrum



Summary : Scrum

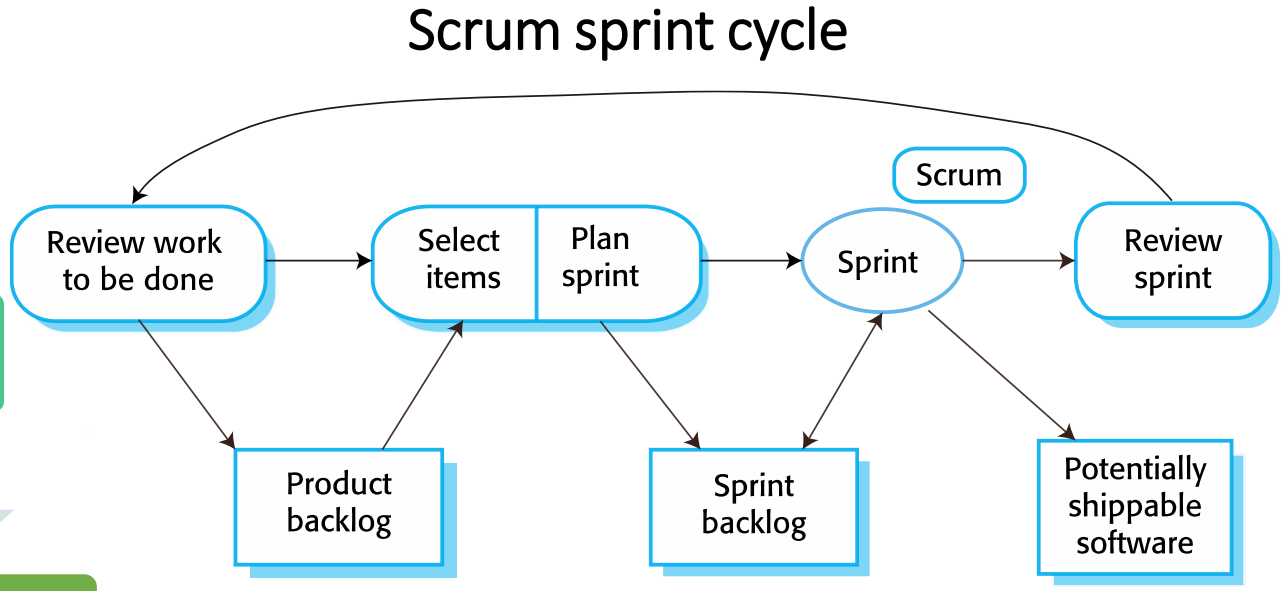
- An agile method that focuses on managing iterative development rather than specific agile practices.

- There are three phases in Scrum.

An outline planning phase.

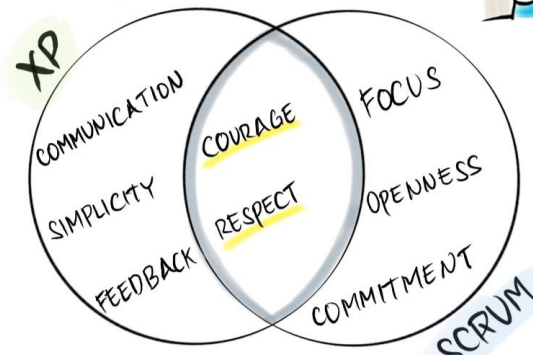
A series of sprint cycles

Project closure phase wraps up the project

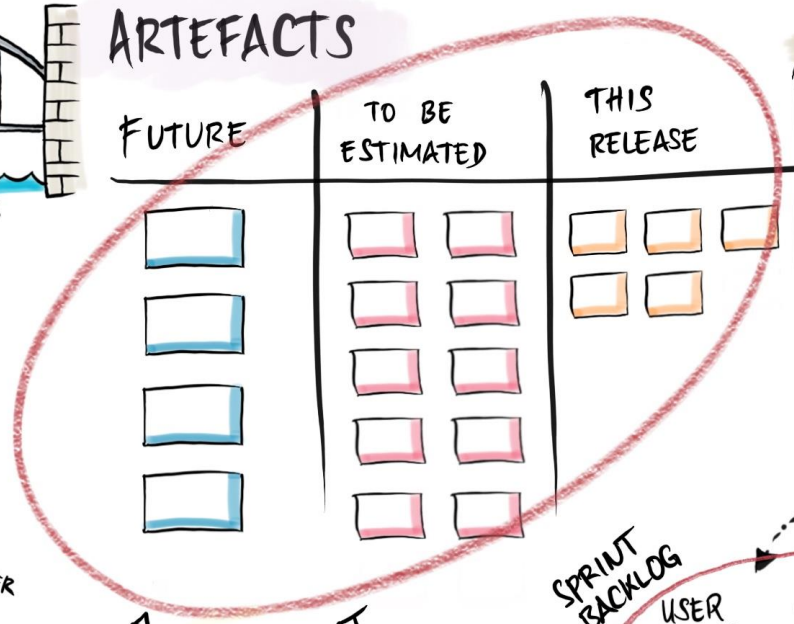


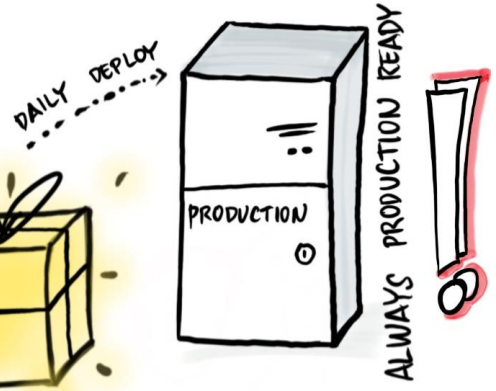
SCRUM & XP

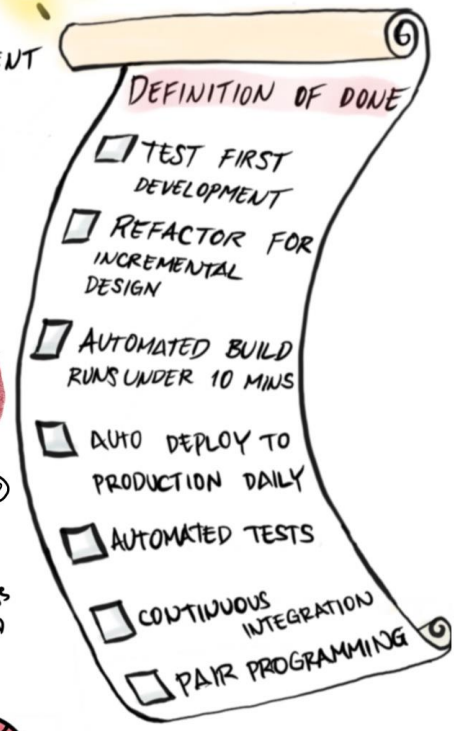
VALUES

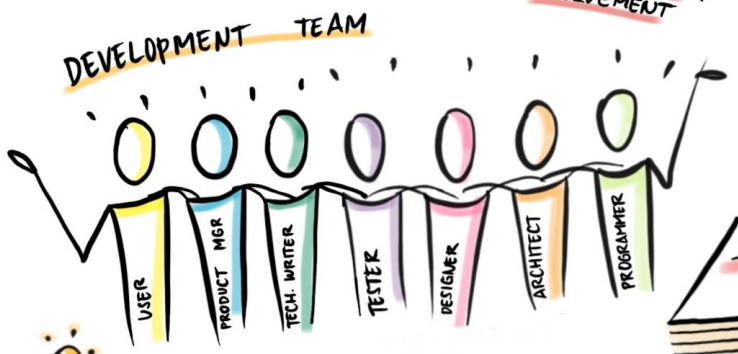


ARTEFACTS



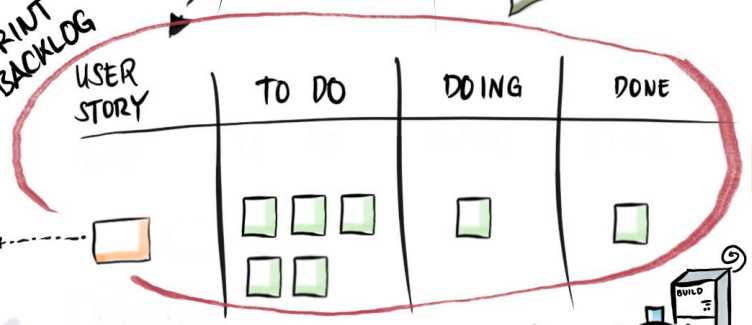




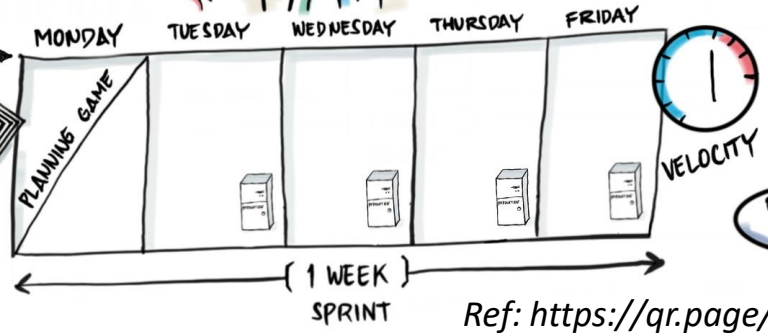


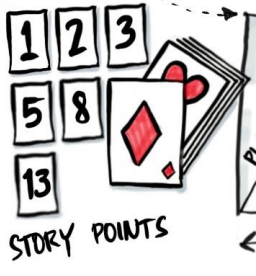
PRODUCT BACKLOG

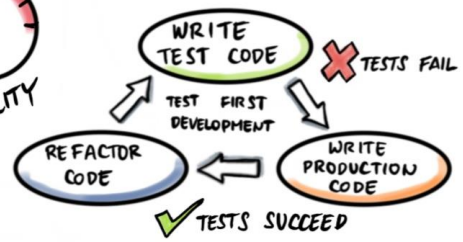
SPRINT BACKLOG



EVENTS







ROLES

@jpartogi
www.agilitypath.com.au

Ref: https://qr.page/g/UGeFko1hq8

XP and Scrum discussions

- XP- incorporated in their own development processes but commonly, used in conjunction wih management-focused agile method i.e Scrum
- Scrum→ framework to organize agile projects

- difference between Scrum and XP is subtle
- Scrum is just a framework for product development, Scrum is a container where you can add other practices.
- XP is one of those practices that you can do within Scrum framework.
- XP is one of the missing piece Scrum team need to deliver great quality products.

Ref: https://qr.page/g/UGeFko1hq8

Topic 3: Agile Software Development

Contents

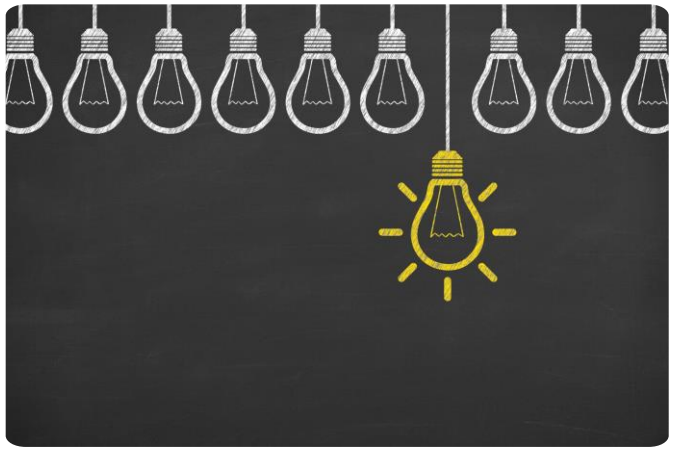
AGILE METHODS

AGILE DEVELOPMENT TECHNIQUES

AGILE PROJECT MANAGEMENT

SCALING AGILE METHODS

Scaling agile methods

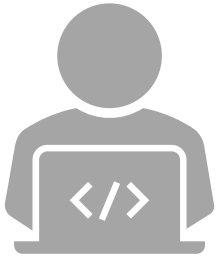


- Agile methods have proved to be successful for small and medium sized projects that can be developed by **a small co-located team**.
- It is sometimes argued that the success of these methods comes because of improved communications which is possible when everyone is working together.

Scaling out and scaling up

‘Scaling up’ :

using agile methods for **developing large software systems** that cannot be developed by a small team.

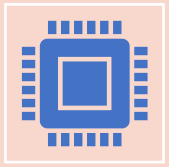


‘Scaling out’

how agile methods can be **introduced across a large organization** with many years of software development experience.

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.

Contractual issues

- Most software contracts for custom systems are based around a specification, which sets out what has to be implemented by the system developer for the system customer.
- However, this precludes interleaving specification and development as is the norm in agile development.
- A contract that pays for developer time rather than functionality is required.
  - However, this is seen as a high risk on many legal departments because what has to be delivered cannot be guaranteed.

Agile methods and software maintenance

Most organizations 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 minimizing 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.

Agile maintenance

Key problems are:

Lack of product documentation	Keeping customers involved in the development process	Maintaining the continuity of the development team

Agile development relies on the development team knowing and understanding what has to be done.

For long-lifetime systems, this is a real problem as the original developers will not always work on the system.

Agile vs plan-driven methods

Is it important to have a very detailed specification and design before moving to implementation?

- If so, you probably need to use a plan-driven approach.

Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic?

- If so, consider using agile methods.

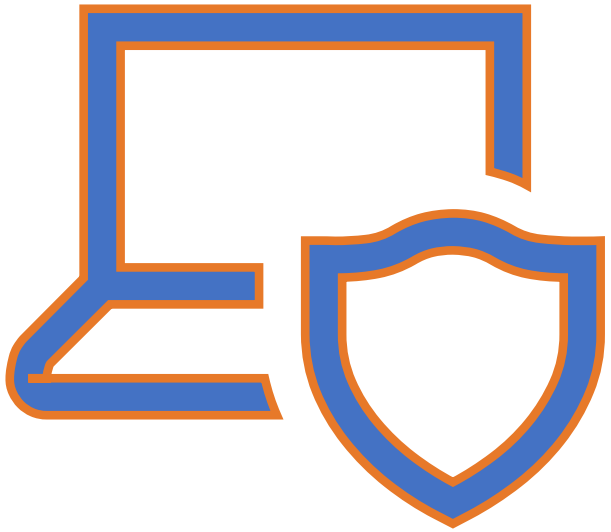
How large is the system that is being developed?

- Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally. This may not be possible for large systems that require larger development teams so a plan-driven approach may have to be used.

Agile principles and organizational practice

Principle	Practice
Customer involvement	This depends on having a customer who is willing and able to spend time with the development team and who can represent all system stakeholders. Often, customer representatives have other demands on their time and cannot play a full part in the software development. Where there are external stakeholders, such as regulators, it is difficult to represent their views to the agile team.
Embrace change	Prioritizing changes can be extremely difficult, especially in systems for which there are many stakeholders. Typically, each stakeholder gives different priorities to different changes.
Incremental delivery	Rapid iterations and short-term planning for development does not always fit in with the longer-term planning cycles of business planning and marketing. Marketing managers may need to know product features several months in advance to prepare an effective marketing campaign.
Maintain simplicity	Under pressure from delivery schedules, team members may not have time to carry out desirable system simplifications.
People, not process	Individual team members may not have suitable personalities for the intense involvement that is typical of agile methods and therefore may not interact well with other team members.

System issues



- How large is the system being developed?
  - Agile methods are most effective a relatively small co-located team who can communicate informally.
- What type of system is being developed?
  - Systems that require a lot of analysis before implementation need a fairly detailed design to carry out this analysis.
- What is the expected system lifetime?
  - Long-lifetime systems require documentation to communicate the intentions of the system developers to the support team.
- Is the system subject to external regulation?
  - If a system is regulated you will probably be required to produce detailed documentation as part of the system safety case.

People and teams

How good are the designers and programmers in the development team?

- It is sometimes argued that agile methods require higher skill levels than plan-based approaches in which programmers simply translate a detailed design into code.

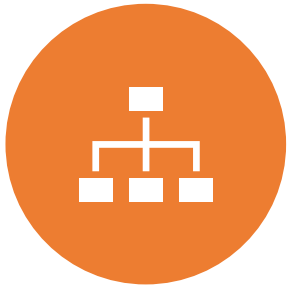
How is the development team organized?

- Design documents may be required if the team is dsitributed.

What support technologies are available?

- IDE support for visualisation and program analysis is essential if design documentation is not available.

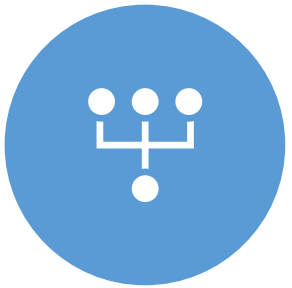
Organizational issues



Traditional engineering organizations have a culture of plan-based development, as this is the norm in engineering.

Is it standard organizational practice to develop a detailed system specification?

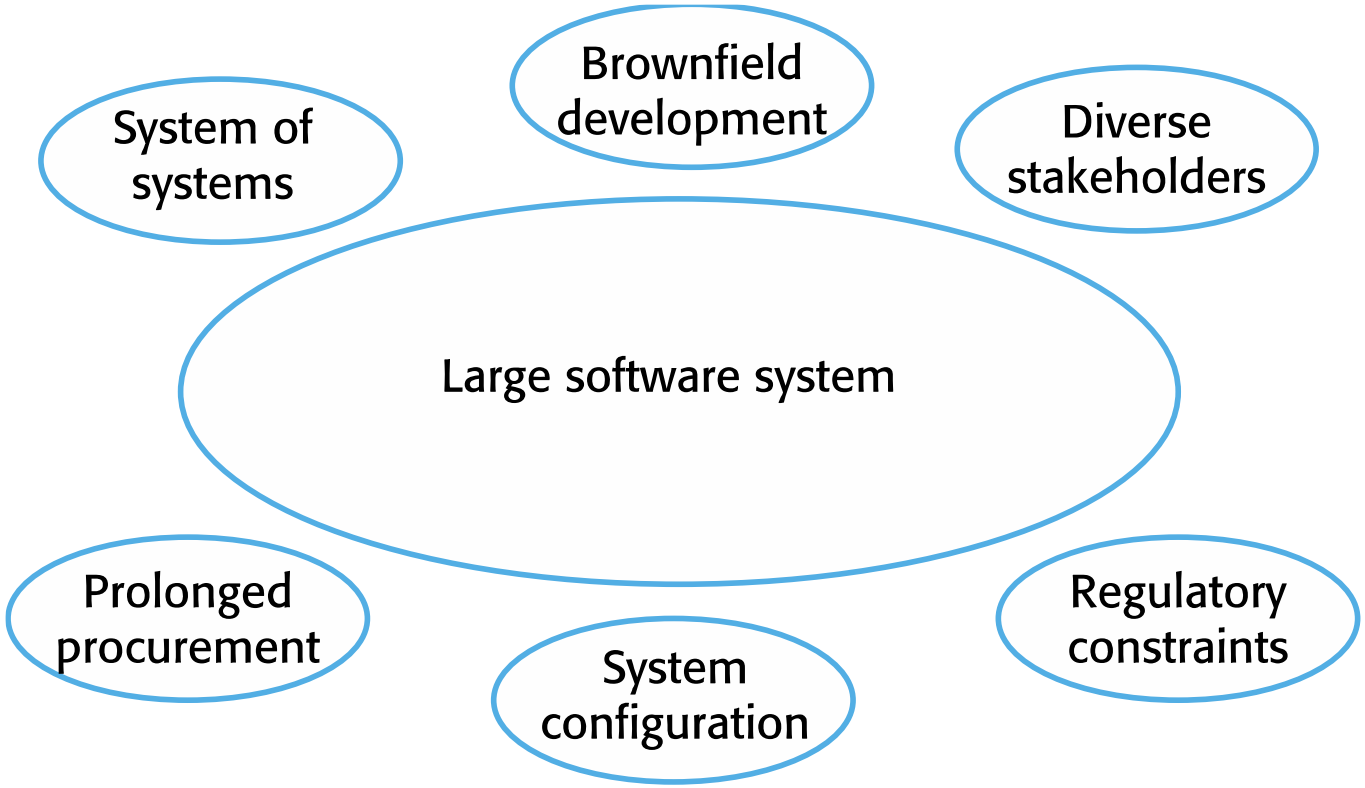
Will customer representatives be available to provide feedback of system increments?



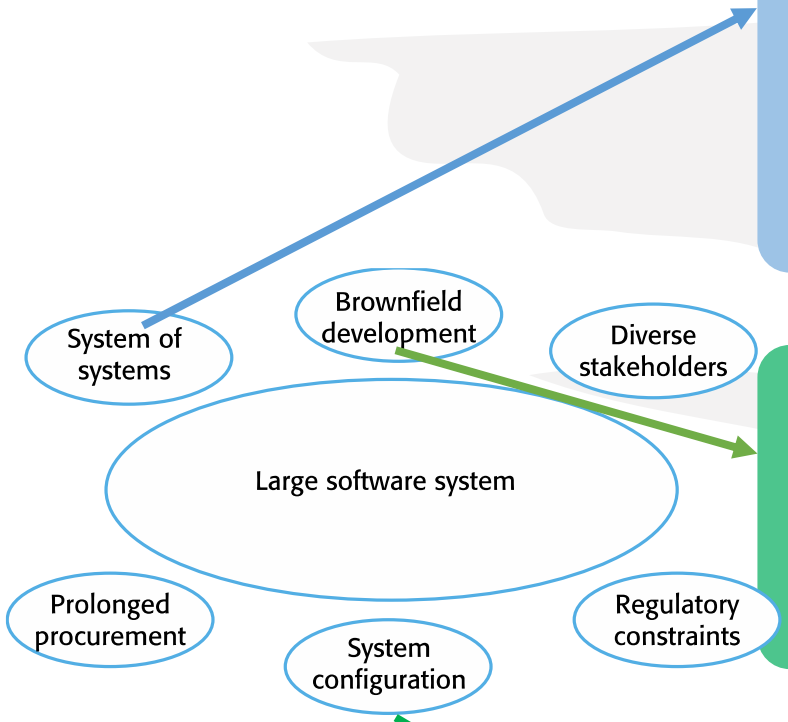
Can informal agile development fit into the organizational culture of detailed documentation?

Agile methods for large systems

6 Factors in large systems



Agile methods for large systems

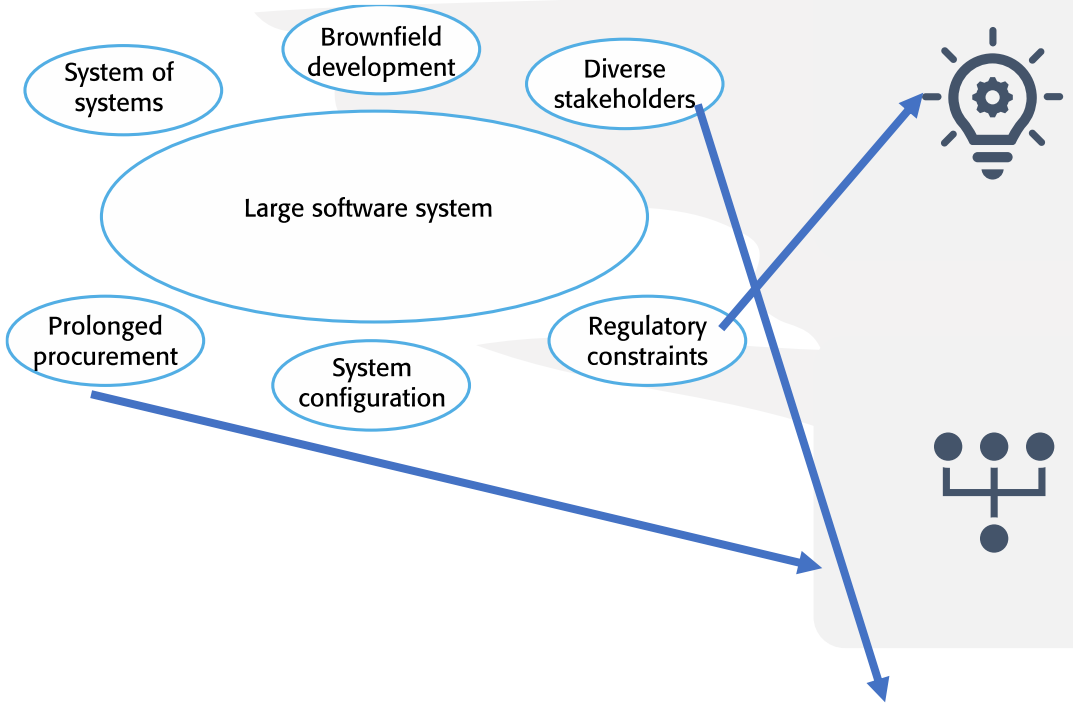


Large systems are usually collections of separate, communicating systems, where separate teams develop each system. Frequently, these teams are working in different places, sometimes in different time zones.

Large systems are 'brownfield systems', that is they include and interact with a number of existing systems. Many of the system requirements are concerned with this interaction and so don't really lend themselves to flexibility and incremental development.

Where several systems are integrated to create a system, a significant fraction of the development is concerned with system configuration rather than original code development.

Agile methods for large systems



Large systems and their development processes are often constrained by external rules and regulations limiting the way that they can be developed.

Large systems have a long procurement and development time. It is difficult to maintain coherent teams who know about the system over that period as, inevitably, people move on to other jobs and projects.

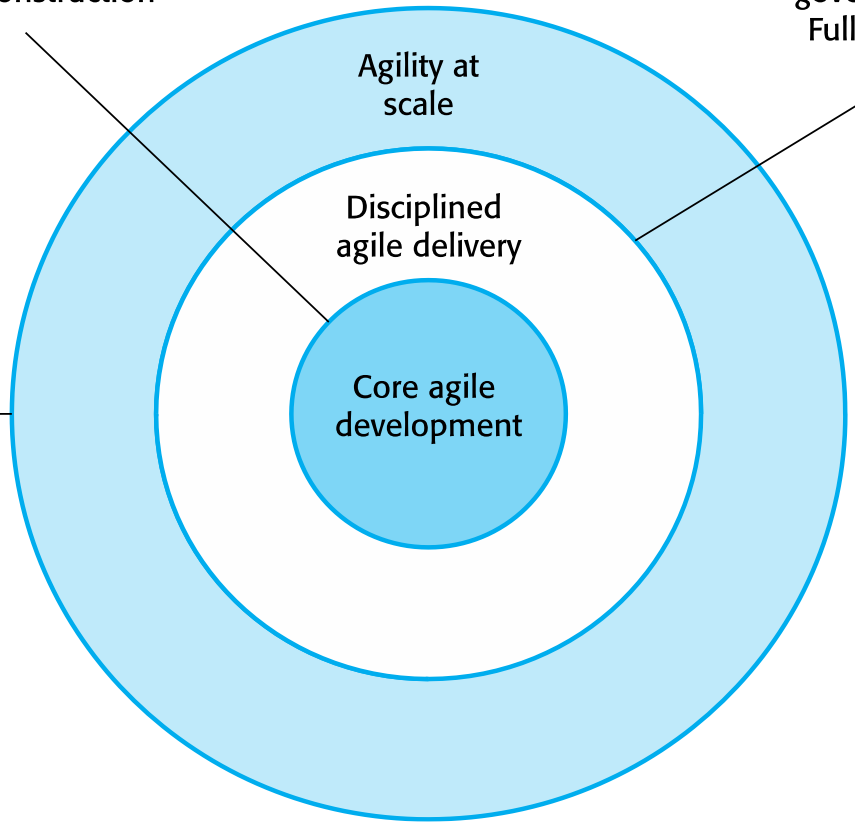
Large systems usually have a diverse set of stakeholders. It is practically impossible to involve all of these different stakeholders in the development process.

IBM's agility at scale model

Core agile development
Value-driven life-cycle
Self-organizing teams
Focus on construction

Disciplined agile delivery
Risk+value driven lifecycle
Self-organizing with appropriate governance framework
Full delivery life-cycle

Agility at scale
Disciplined agile delivery where scaling factors apply:
Large team size
Geographic distribution
Regulatory compliance
Domain complexity
Organization distribution
Technical complexity
Organizational complexity
Enterprise discipline



Scaling up to large systems

A completely incremental approach to requirements engineering is impossible.

There cannot be a single product owner or customer representative.

For large systems development, it is not possible to focus only on the code of the system.

Cross-team communication mechanisms have to be designed and used.

Continuous integration is practically impossible. However, it is essential to maintain frequent system builds and regular releases of the system.

Multi-team Scrum

Role replication

- Each team has a Product Owner for their work component and ScrumMaster.

Product architects

- Each team chooses a product architect and these architects collaborate to design and evolve the overall system architecture.

Release alignment

- The dates of product releases from each team are aligned so that a demonstrable and complete system is produced.

Scrum of Scrums

- There is a daily Scrum of Scrums where representatives from each team meet to discuss progress and plan work to be done.

Agile methods across organizations

- Project managers who do not have experience of agile methods may be reluctant to accept the risk of a new approach.
- Large organizations often have quality procedures and standards that all projects are expected to follow and, because of their bureaucratic nature, these are likely to be incompatible with agile methods.
- Agile methods seem to work best when team members have a relatively high skill level. However, within large organizations, there are likely to be a wide range of skills and abilities.
- There may be cultural resistance to agile methods, especially in those organizations that have a long history of using conventional systems engineering processes.

Key points (1/2)

- Agile methods are incremental development methods that focus on rapid software development, frequent releases of the software, reducing process overheads by minimizing documentation and producing high-quality code.

- Agile development practices include
  - User stories for system specification
  - Frequent releases of the software,
  - Continuous software improvement
  - Test-first development
  - Customer participation in the development team.

Key points (2/2)

- Scrum is an agile method that provides a project management framework.
  - It is centred round a set of sprints, which are fixed time periods when a system increment is developed.
- Many practical development methods are a mixture of plan-based and agile development.
- Scaling agile methods for large systems is difficult.
  - Large systems need up-front design and some documentation and organizational practice may conflict with the informality of agile approaches.