Software Engineering – Lecture 11
Agile software development

Adapted after ©**Ian Sommerville**
Software Engineering, 2010, chapter **3**;
Engineering Software Products, 2019, chapter **2**

Topics covered

AGILE methods

Plan-driven and agile development

Extreme programming

Scaling agile methods

Scrum

Agile methods

Rapid development and delivery is now often the most important requirement for software systems

- Businesses operate in a fast–changing environment (ex. volatile Internet software industry, emerging mobile application environment).
- Software has to evolve quickly to reflect changing business needs.

- It is practically impossible to produce a set of stable software requirements.

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

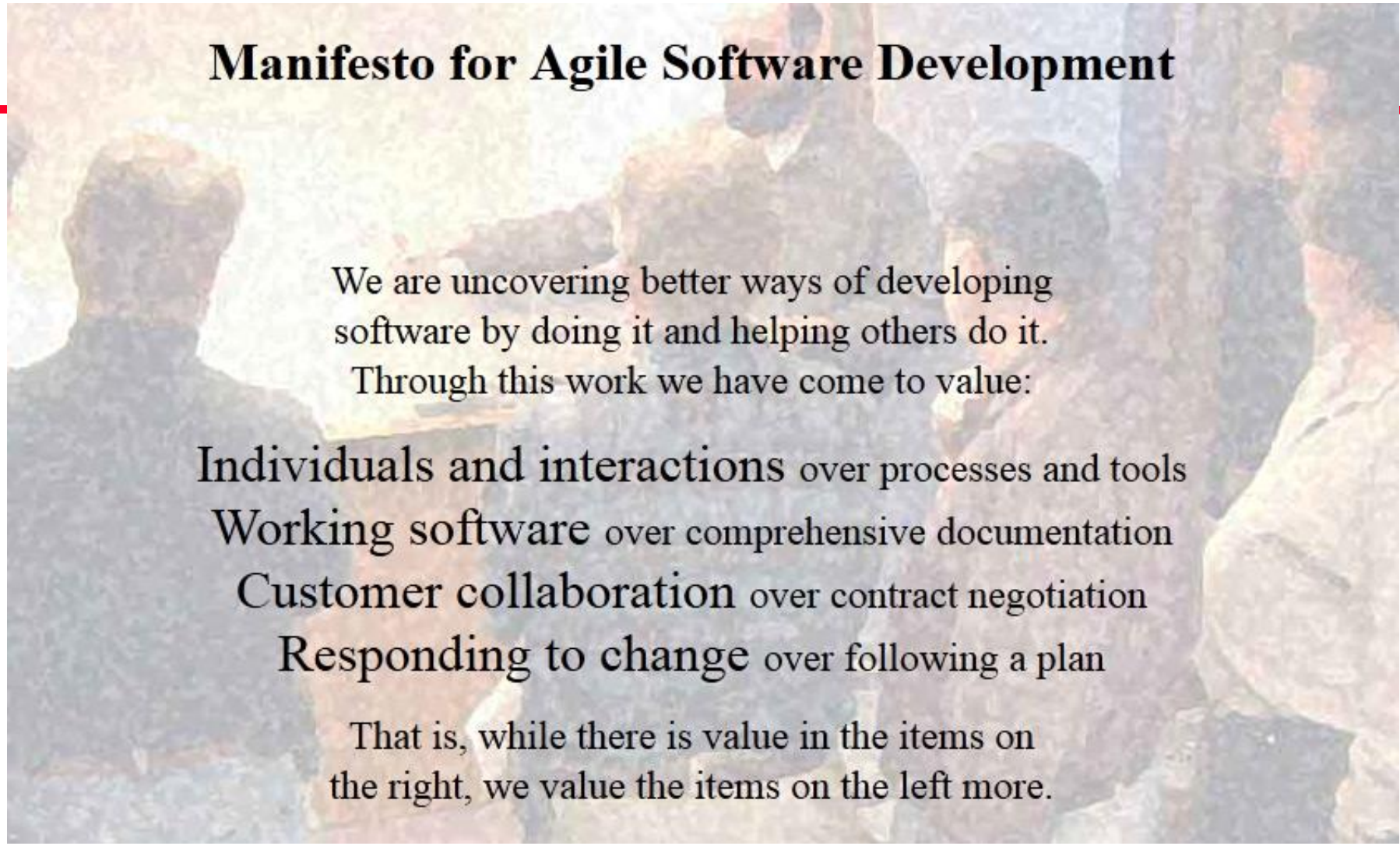
Agile methods:

Are based on an *iterative and incremental* approach of software development;

Are intended to *deliver working software quickly*, and *evolve this quickly* to meet changing requirements.

Aim 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 methods



www.agilemanifesto.org

Agile methods features

- *Short releases and iterations*
- *Incremental design*
- *User involvement*
- *Minimal documentation:* Do only the necessary amount of documentation, which is just a means to an end. The source code is a big part of the actual documentation.
- *Informal communication*
- *Change is assumed*

Obs. ***Enough documentation*** must be available if the released software needs to be maintained by a group different from the original developers.

Principles of agile methods

Principle	Description
Customer involvement	The *customer* should be closely *involved* throughout the development process. Their role is to *provide* and *prioritise* new system *requirements* and to *evaluate the iterations* of the system.
Incremental delivery	The software is developed in *increments*. *Test and evaluate* each increment *as it is developed* and feed back required changes to the development team.
People not process	The *skills* of the development team should be *recognised and exploited.* The team should be left to develop their own ways of working *without prescriptive processes*.
Embrace change	Expect the system requirements to change and design the system so that it can *accommodate* these *changes*.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process used. Wherever possible, actively work to eliminate complexity from the system.

Agile methods

Aplicability

- Developing of small or medium-sized product for sale. Virtually all software products are now developed using an agile approach.
- Developing custom system 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.

Processes and methodologies (agile or traditional) have to be adjusted for each project.

⇓

A software engineer must know about many different methodologies to be able to adopt specific techniques that may be useful to his particular project.

Problems with agile methods

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 characterizes agile methods.

Prioritizing 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.

Requirements are collected informally and incrementally, without a coherent requirements document.

Difficulties if original development team cannot be maintained.

Topics covered

AGILE methods

Plan-driven and agile development

Extreme programming

Scaling agile methods

Scrum

Plan-driven and agile development

Plan-driven development

Separate development stages; the outputs to be produced at each stage are planned in advance.

Not necessarily waterfall model – plan-driven, incremental development and delivery is possible.

Iteration within activities; formal documents to communicate between stages.

Agile development

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

Iteration occurs across activities.

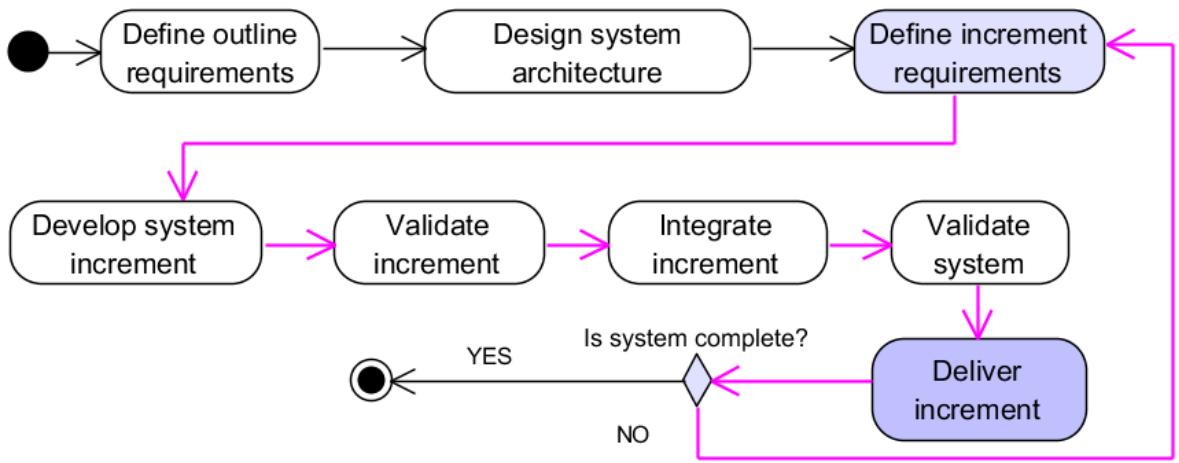
Not inevitably code-focused - may produce some design documentation.

Obs. Most software projects include practices from plan-driven and agile approaches.

The balance in a specific process depends on technical, human and organizational factors.

Incremental development and delivery

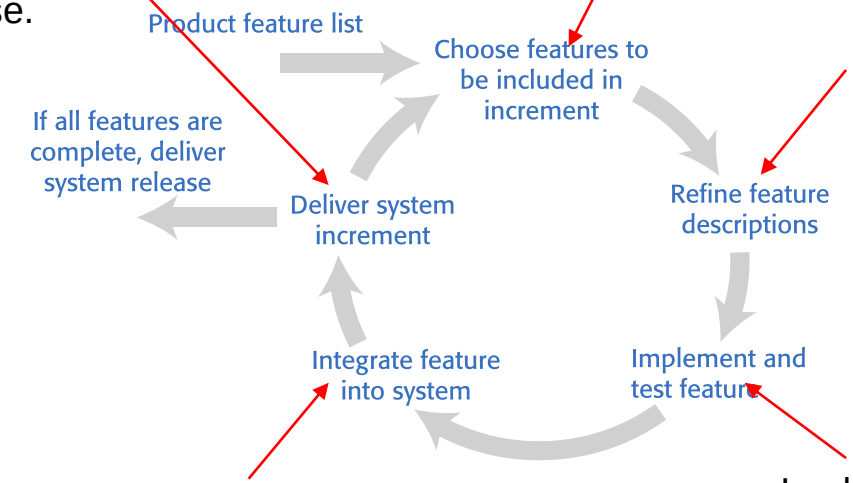
- Features are prioritized so that the most important features are implemented first.
  - Only the details of the feature being implemented in an increment are defined.
  - That feature is then implemented and delivered.
- Users or surrogate users can try it out and provide feedback to the development team. Then the next feature of the system is defined and implemented.



Incremental development and delivery

Deliver the system increment to the customer or product manager for checking and comments. If enough features have been implemented, release a version of the system for customer use.

Using the list of features in the planned product, select those features that can be implemented in the next product increment.



Add detail to the feature descriptions so that the team have a common understanding of each feature and there is sufficient detail to begin implementation.

Integrate the developed feature with the existing system and test it to check that it works in conjunction with other features.

Implement the feature and develop automated tests for that feature that show that its behaviour is consistent with its description.

Figure 2.1 Incremental development

Ian Sommerville – Engineering Software Products

Formative evaluation

1. Check the correct answers:

Agile methods in software development imply :

[ ] Incremental delivery

[ ] Periodic activities to eliminate complexity from the system

[ ] Customer involvement during the software process

[ ] Modeling the whole software before writing the code

[ ] Establishing normative processes for team working

[ ] Planning in advance all software process activities

2. Explain why agile methods in software engineering ensures rapid development and delivery of software products.
(Base your argumentation on the agile methods principles and specifics).

https://forms.gle/vdy1t251EkN6Srn4A

Topics covered

AGILE methods

Plan-driven and agile development

Extreme programming

Scaling agile methods

Scrum

Agile methods – examples (1)

Extreme programming (XP)

small and medium teams (3-20), co-located,

need for adequate technology

Crystal family of methodologies

co-located teams, max. 40

not suitable for life-critical systems

FDD (Feature Driven Development)

identification and planification of “features”,

iterative only for design and construction;

high quality, supports life-critical systems

Agile methods – examples (2)

Dynamic Systems Development Method

multiple teams of 2-6 members

small projects, large but divisible projects, bussiness applications

Adaptive Software Development

includes a learning loop

distributed teams

Open Source Software Development

distributed teams

software tools, infrastructures with large number of users

Some agile solutions

AUP (Agile Unified Process)

www.ambysoft.com/unifiedprocess/agileUP.html

Scott Ambler – Disciplined Agile Delivery : A Practitioners"s Guide to Agile Software Delivery in Enterprise - 2012

Agile solutions at IBM:

www-01.ibm.com/software/rational/info/agilityatscale/?ca=rhp

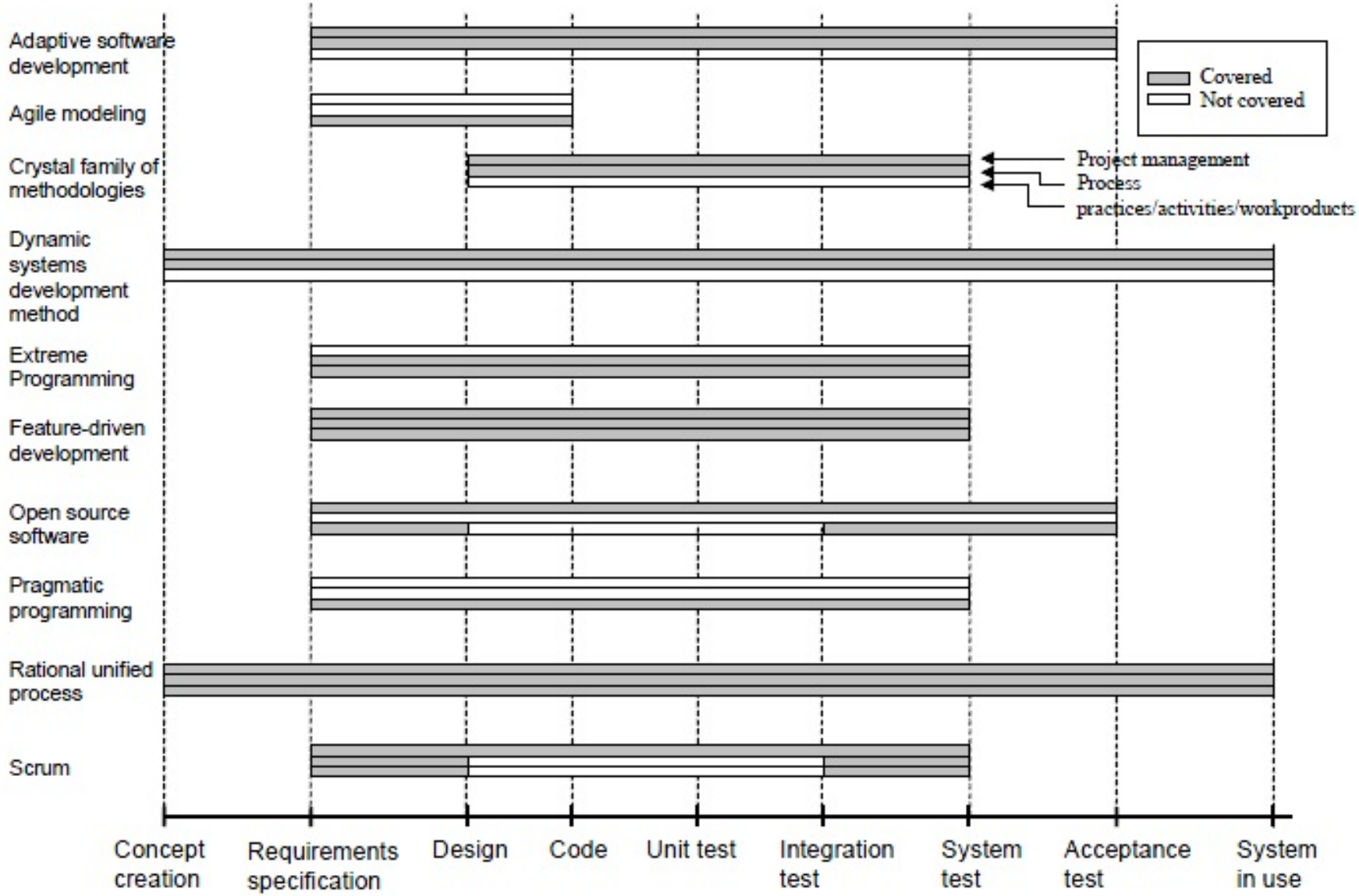
Software tool: Jazz - Managing distributed agile teams

Fundaments:

Customizable process

Extends Scrum, XP and Agile Modeling

Agile methods – suport for the software process



eXtreme Programming (XP)

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

XP = an approach to software development based on the development and delivery of very small increments of functionality.

Relies on:

- constant code improvement,
- user involvement in the development team,
- pairwise programming.

eXtreme Programming (XP) takes an “extreme” approach to iterative development.

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

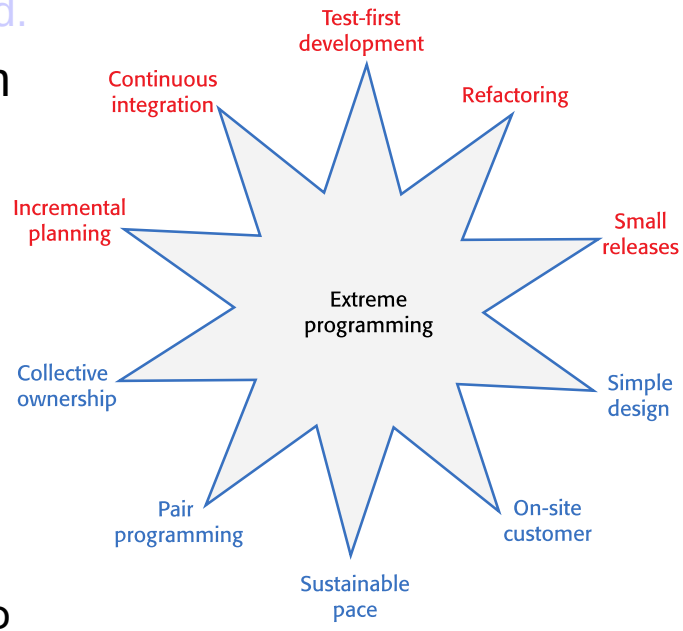
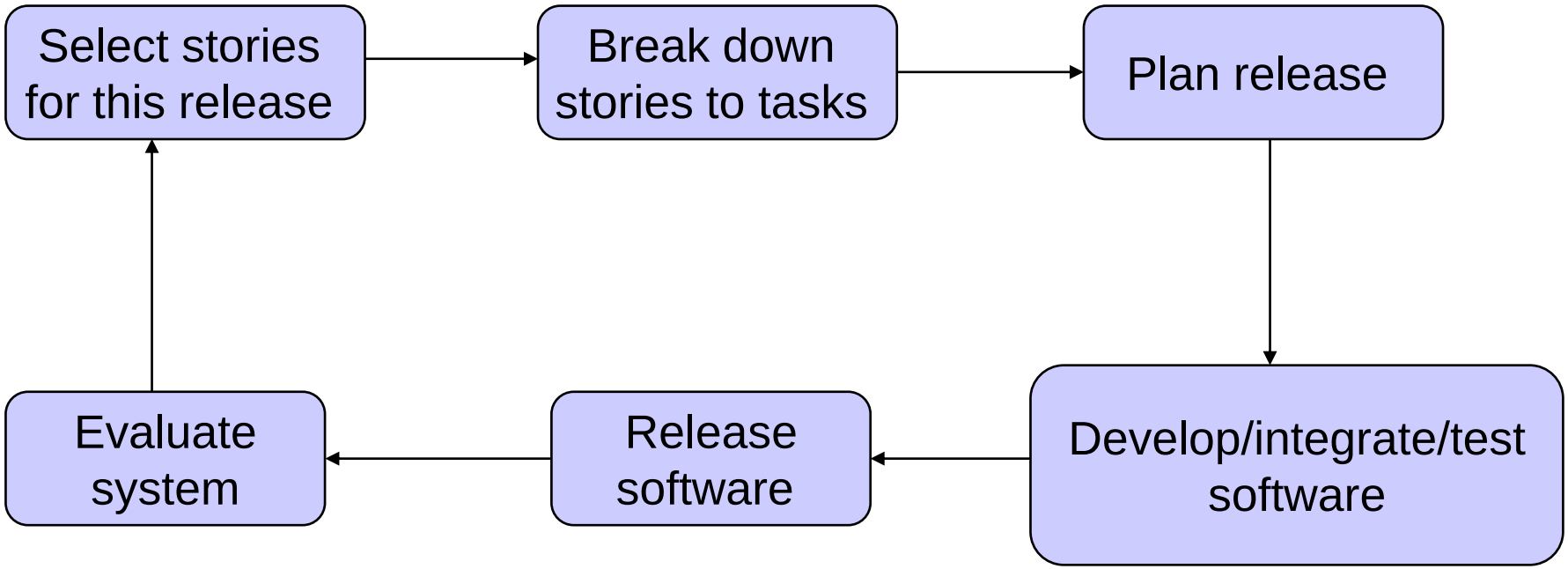


Figure 2.2 Extreme programming practices
Ian Sommerville – Engineering Software Products

The XP release cycle



Extreme programming practices

XP practice	Description
Incremental planning /user stories	There is no ‘grand plan’ for the system. Instead, what needs to be implemented (the requirements) in each increment are established in discussions with a customer representative. The requirements are written as user stories. The stories to be included in a release are determined by the time available and their relative priority
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 previous release.
Test-driven development	Instead of writing code then tests for that code, developers write the tests first. This helps clarify what the code should actually do and that there is always a ‘tested’ version of the code available. An automated unit test framework is used to run the tests after every change. New code should not ‘break’ code that has already been implemented.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system and a new version of the system is created. All unit tests from all developers are run automatically and must be successful before the new version of the system is accepted.
Refactoring	Refactoring means improving the structure, readability, efficiency and security of a program. All developers are expected to refactor the code as soon as potential code improvements are found. This keeps the code simple and maintainable

XP and agile principles

Incremental development - supported through small, frequent system releases.

Customer involvement - means full-time customer engagement with the team.

People not process through:

- pair programming,
- collective ownership
- a process that avoids long working hours.

Change - supported through regular system releases.

Maintaining simplicity - through constant refactoring of code.

Requirements scenarios

The *customer representative* is responsible for making decissions on requirements.

In XP, user requirements are expressed as *scenarios*, or *user stories*.

- 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 on the schedule estimates.

Example: Story card for document downloading

Downloading and printing an article

First, you select the article that you want from a displayed list. You then have to tell the system how you will pay for it - this can either be through a subscription, through a company account or by credit card.

After this, you get a copyright form the system to fill in and, when you have submitted this, the article you want is downloaded onto your computer

You then choose a printer and a copy of the article is printed.You tell the system if printing has been successful.

If the article is a print-only article, you can not keep the PDF version so it is automatically deleted from your computer

Example: Task cards for document downloading

Task 1: Implement principal workflow

Task 2: Implement article catalog and selection

Task 3: Implement payment collection

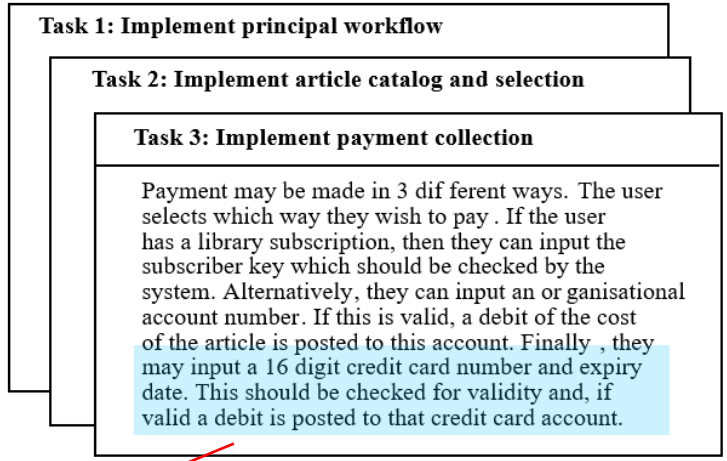
Payment may be made in 3 dif ferent ways. The user selects which way they wish to pay . If the user has a library subscription, then they can input the subscriber key which should be checked by the system. Alternatively, they can input an or ganisational account number. If this is valid, a debit of the cost of the article is posted to this account. Finally , they may input a 16 digit credit card number and expiry date. This should be checked for validity and, if valid a debit is posted to that credit card account.

Testing in XP

Test-first development and automated testing.

- Writing tests before code clarifies the requirements to be implemented.
- Tests are Incrementally developed from scenarios.
- User is involved in test development and validation.
- Tests are written as programs rather than as data, so that they can be executed automatically. The test includes a check that it has executed correctly.
- All previous and new tests are automatically run when new functionality is added. Thus is checked that the new functionality works correct and no regression appears.
- Automated test harnesses are also used to run all component tests each time a new release is built.

Example: Test case description



Test 4:	Testing the function that verifies credit card validity	

Input:
A string representing the credit card number and two integers representing the month and year when the card expires

Tests: Test that the unit executes correctly the following operations:

Check that all bytes in the string are digits

Check that the month lies between 1 and 12 and the year is greater than or equal to the current year

Using the first 4 digits of the credit card number, check that the card issuer is valid by looking up the card issuer table. Check credit card validity by submitting the card number and expiry date information to the card issuer

Output:
OK or error message indicating that the card is invalid

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.

Pair programming

In XP, programmers work in pairs, sitting together to develop code.

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 this.

Measurements suggest that development productivity with pair programming is similar to that of two people working independently.

Formative evaluation

1. Explain how XP aligns to the the principles of agile software development methods.

2. What is the specific of acceptance testing in XP context ?

https://forms.gle/i1GyuwAgySjVyhP69

Topics covered

AGILE methods

Plan-driven and agile development

Extreme programming

Scaling agile methods

Scrum

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 out and scaling up

- ‘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 essential to maintain agile fundamentals
  - Flexible planning, frequent system releases, continuous integration, test-driven development and good team communications.

Scaling up to large systems

For large systems development, it is not possible to focus only on the code of the system. You need to do more up-front *design and system documentation*.

Cross-team communication mechanisms have to be designed and used. This should involve regular phone and video conferences between team members and frequent, short electronic meetings where teams update each other on progress.

Continuous integration, where the whole system is built every time any developer checks in a change, *is practically impossible*. However, it is essential to maintain *frequent system builds* and *regular releases* of the system.

Scaling out to large companies

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.

Topics covered

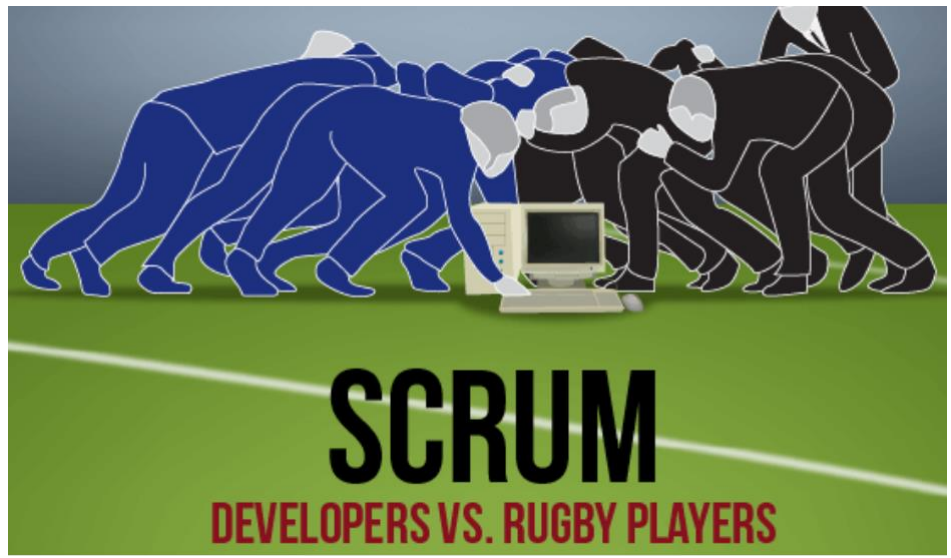
AGILE methods

Plan-driven and agile development

Extreme programming

Scaling agile methods

Scrum



Scrum

Software company managers need information about the *costs* to develop a software product, the *development time* and the *release date*.

Plan-driven development provides this information through long-term *development plans* that identify *deliverable*s - items the team will deliver and when these will be delivered.

Plans always change so anything apart from short-term plans are unreliable.

Scrum is an agile method that provides a *framework for agile project organization and planning*. It does not mandate any specific technical practices.

Scrum terminology

Product

The software product that is being developed by the Scrum team.

Product owner

A team member who is responsible *for identifying product features and attributes*. They *review work* done and *help to test* the product.

Product backlog

A *to-do list* of items such as bugs, features and product improvements that the Scrum team have not yet completed.

Development team

A *small self-organizing team* of five to eight people who are responsible for developing the product.

Sprint

A short *period*, typically two to four weeks, when a *product increment is develope*d.

Scrum terminology

Scrum

A *daily team meeting* where progress is reviewed and work to be done that day as discussed and agreed.

ScrumMaster

A *team coach* who guides the team in the effective use of Scrum.

Potentially shippable product increment

The *output of a sprint* which should be of high enough quality to be deployed for customer use.

Velocity

An estimate of *how much work* a team can do *in a single sprint*.

Key roles in Scrum

The Product Owner is responsible for ensuring that the development team are always focused on the product they are building rather than diverted into technically interesting but less relevant work.

- In product development, the product manager should normally take on the Product Owner role.

The ScrumMaster is a Scrum expert whose job is to guide the team in the effective use of the Scrum method.

The ScrumMaster is not a conventional project manager but is a coach for the team. He has authority within the team on how Scrum is used.

- In many companies that use Scrum, the ScrumMaster also has some project management responsibilities.

Scrum and sprints

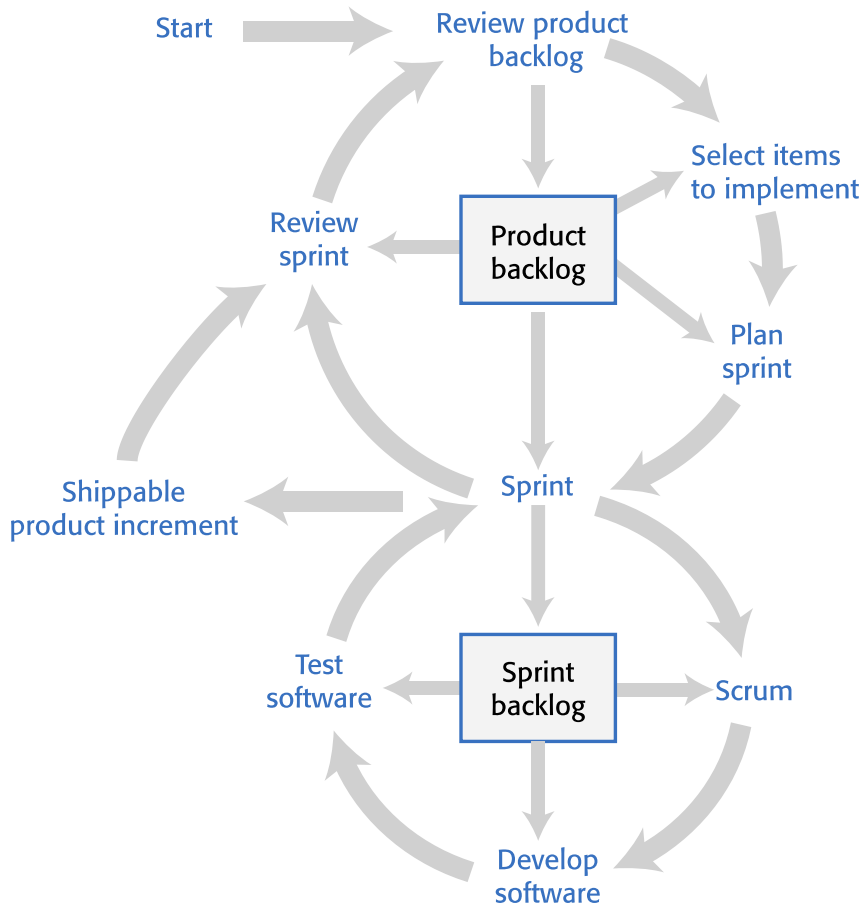
In Scrum, software is developed in *sprints*, which are *fixed-length periods* (2 - 4 weeks) in which software features are developed and delivered.

During a sprint, the team has daily meetings (Scrums) to *review progress* and to *update the list* of work items that are incomplete.

Sprints should produce a ‘shippable product increment’. This means that the developed software should be *complete and ready to deploy*.

.

Figure 2.3 Scrum cycles
Ian Sommerville – Engineering Software Products



Scrum benefits

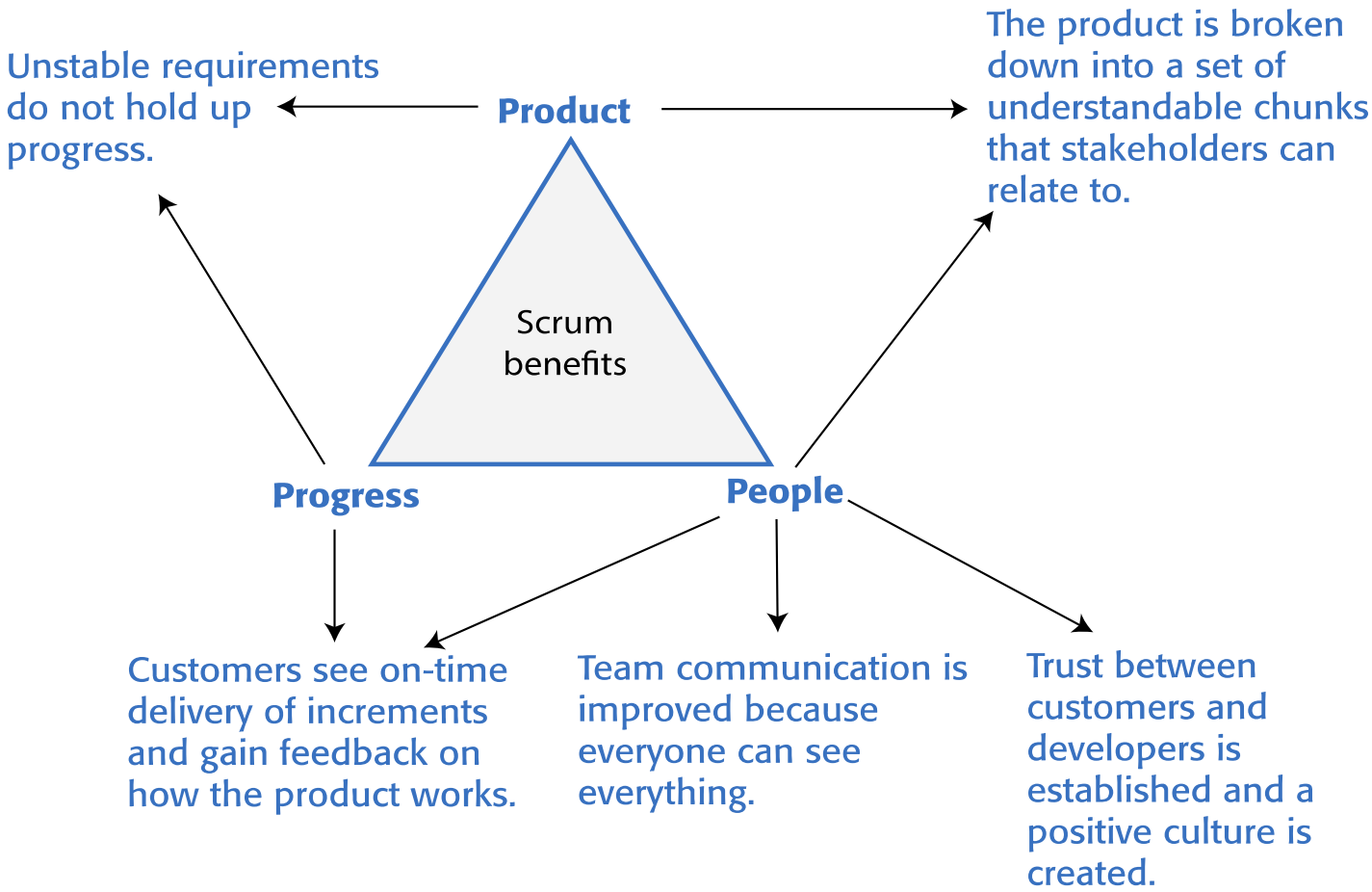


Figure 2.4 The top five benefits of using Scrum

Ian Sommerville – Engineering Software Products

Key Scrum practices

Product backlog

This is a *to-do list* of items to be implemented that is reviewed and updated before each sprint.

Timeboxed sprints

Fixed-time (2-4 week) periods in which items from the product backlog are implemented,

Self-organizing teams

Self-organizing teams make their own decisions and work by discussing issues and making decisions by consensus.

.

Product backlogs

The product backlog is *a list* of what needs to be done *to complete the development* of the product.

The items on this list are called *product backlog items (PBIs).*

The product backlog may include a variety of different items such as *product features* to be implemented, *user requests*, essential *development activities* and desirable *engineering improvements*.

The product backlog should always be *prioritized* so that the items that be implemented first are at the top of the list.

Use cases can help teams understand the bigger picture and how product backlog items are related.

.

Examples of product backlog items

1. As a teacher, I want to be able to configure the group of tools that are available to individual classes. *(feature)*

2. As a parent, I want to be able to view my childrens' work and the assessments made by their teachers. (*feature)*

3. As a teacher of young children, I want a pictorial interface for children with limited reading ability. *(user request)*

4. Establish criteria for the assessment of open source software that might be used as a basis for parts of this system. *(development activity)*

5. Refactor user interface code to improve understandability and performance. *(engineering improvement)*

6. Implement encryption for all personal user data. *(engineering improvement)*

.

Product backlog item states

Ready for consideration

High-level ideas and feature descriptions that will be considered for inclusion in the product. They are *tentative* so may radically change or may not be included in the final product.

Ready for refinement

An item on which the team has agreed to be important, so it should be implemented as part of the current development. There is a reasonably clear definition of what is required. However, work is needed to understand and refine the item.

Ready for implementation

The PBI has enough detail for the team to estimate the effort involved and to implement the item. Dependencies on other items have been identified.

.

Product backlog management

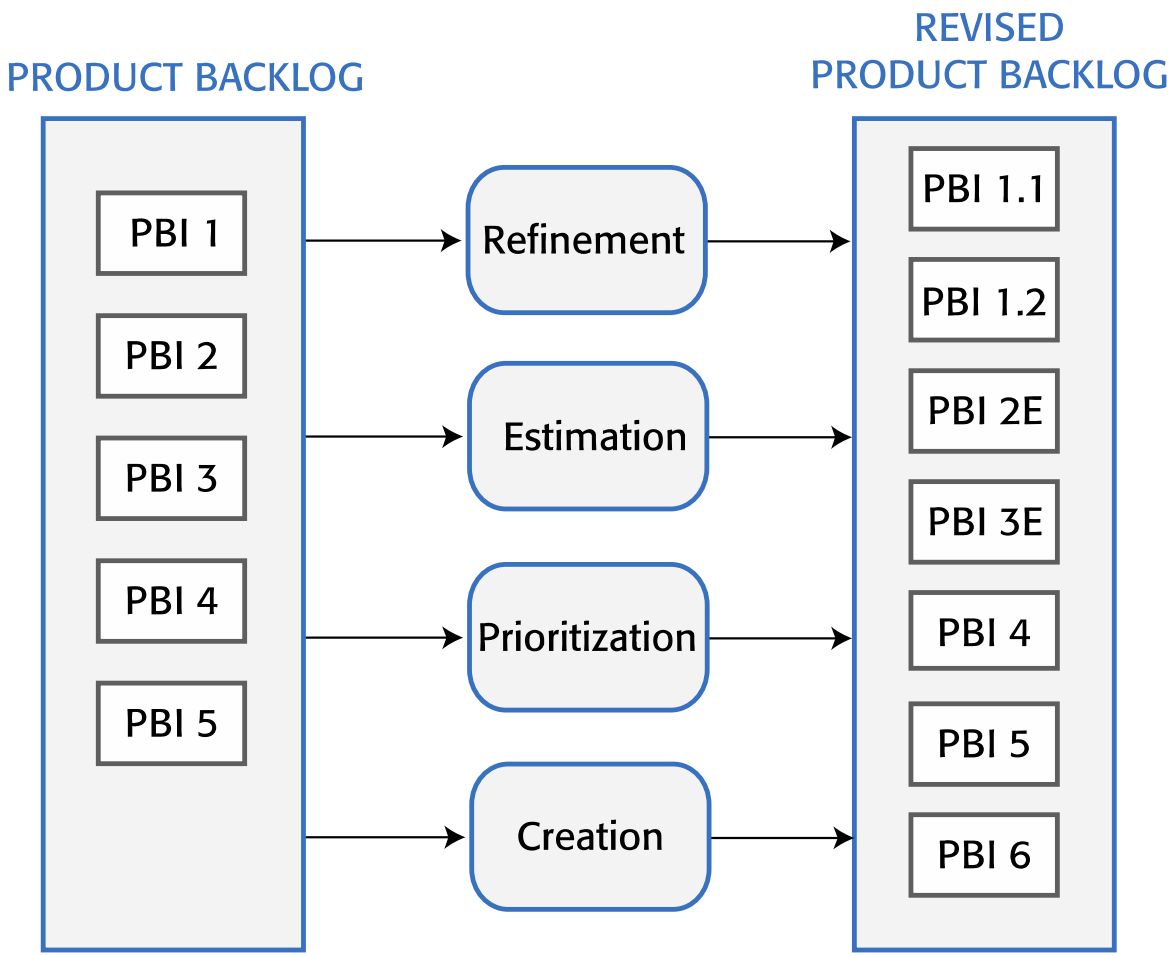


Figure 2.5 Product backlog activities
Ian Sommerville – Engineering Software Products

Product backlog activities

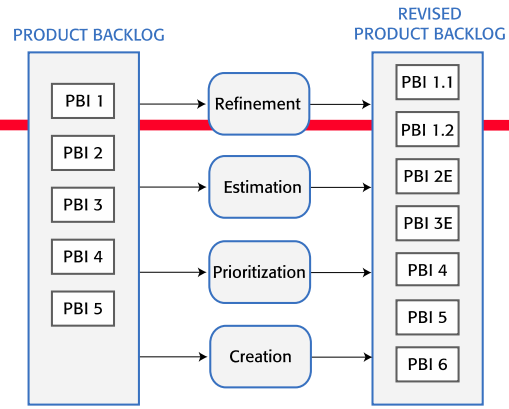


Figure 2.5 Product backlog activities
Ian Sommerville – Engineering Software Products

Refinement

Existing PBIs are analyzed and refined to create more detailed PBIs. This may lead to the creation of new product backlog items.

Estimation

The team estimate the amount of work required to implement a PBI and add this assessment to each analyzed PBI.

Creation

. New items are added to the backlog. These may be *new features* suggested by the product manager, *required feature changes*, *engineering improvements*, or *process activities* such as the assessment of development tools that might be used.

Prioritization

The product backlog items are reordered to take new information and changed circumstances into account.

PBI estimation metrics

Effort required

Expressed in person-hours or person-days i.e. the number of hours or days it would take one person to implement that PBI. Several people may work on an item, which may shorten the calendar time required.

Story points

Arbitrary estimate of the effort involved in implementing a PBI, taking into account the size of the task, its complexity, the technology that may be required and the ‘unknown’ characteristics of the work.

They were derived originally by comparing user stories, but they can be used for estimating any kind of PBI.

Story points are estimated relatively. The team agree on the story points for a baseline task and other tasks are estimated by comparison with this e.g. more/less complex, larger/smaller etc.

.

Timeboxed sprints

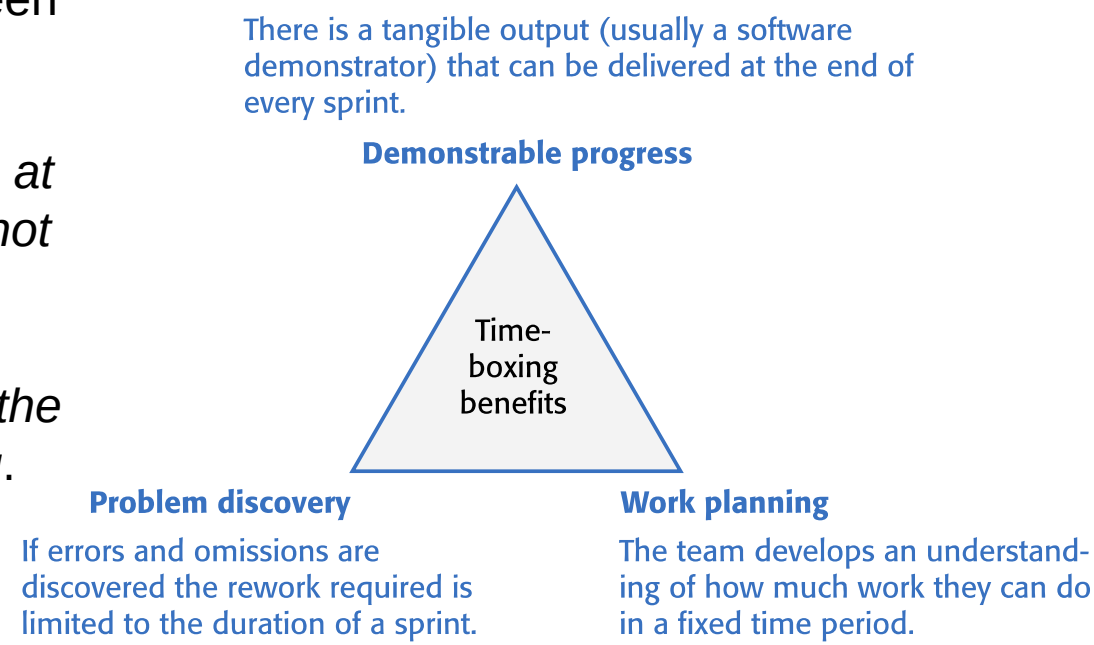
Products are developed in a *series of sprints*, each of which delivers an increment of the product or supporting software.

Sprints are *short duration activities* (1-4 weeks) and take place between a defined start and end date. Sprints are timeboxed, which means that development *stops at the end of a sprint whether or not the work has been completed*.

During a sprint, the team *work on the items from the product backlog*.

.

Figure 2.6 Benefits of using timeboxed sprints
Ian Sommerville – Engineering Software Products



Sprint activities

Sprint planning

Work items to be completed in that sprint are selected and, if necessary, refined to create a *sprint backlog*. This should not last more than *a day at the beginning of the sprint*.

Sprint execution

The *team work to implement* the sprint backlog items that have been chosen for that sprint. If it is impossible to complete all of the sprint backlog items, the *sprint is not extended*. The *unfinished items* are returned to the product backlog and *queued for a future sprint*.

Sprint reviewing

The work done in the sprint is reviewed by the team and (possibly) external stakeholders. The *team reflect* on what went well and what went wrong during the sprint with a view *to improving their work process.*

Sprint activities

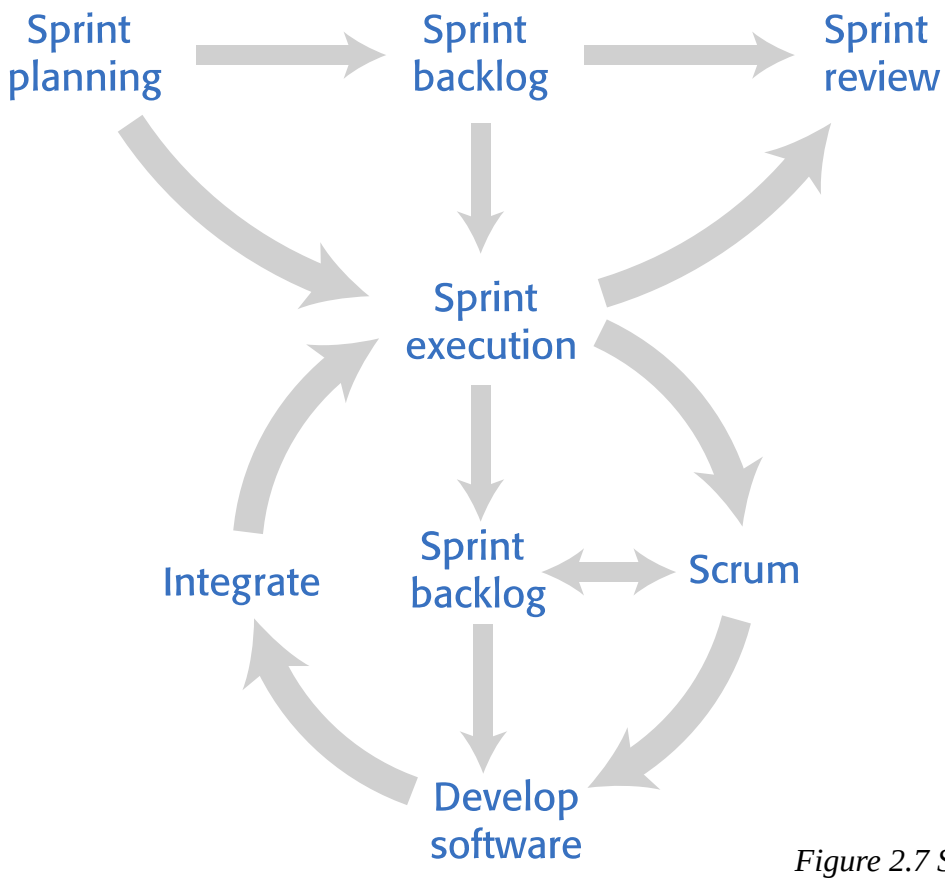
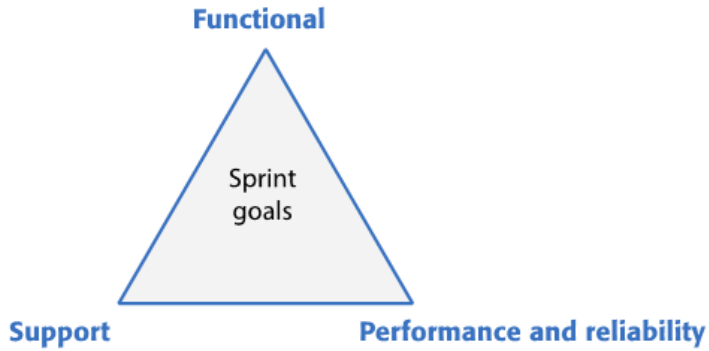


Figure 2.7 Sprint activities
Ian Sommerville – Engineering Software Products

Sprint planning

Establish an agreed sprint goal

Sprint goals may be focused on software functionality, support or performance and reliability.



Decide on the list of items from the product backlog that should be implemented

Create a sprint backlog.

This is a more detailed version of the product backlog that records the work to be done during the sprint.

Sprint goals examples

Implement user roles so that a user can select their role when they login to the system

Functional

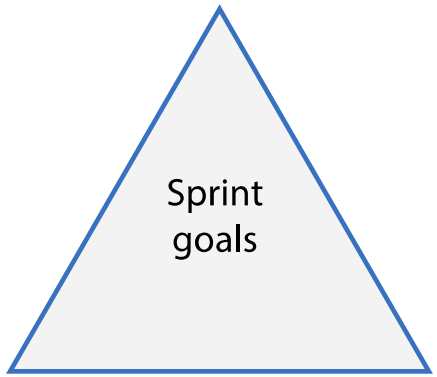


Figure 2.8 Sprint goals
Ian Sommerville – Engineering Software Products

Support

Develop analytics that maintain information about the time users spend using each feature of the system.

Performance and reliability

Ensure that the login response time is less than 10 seconds for all users where there are up to 2000 simultaneous login connections.

Scrums

Scrum = short, daily meeting that is usually held at the beginning of the day.

During a scrum, all team members *share information*, describe their *progress* since the previous day's scrum, *problems* that have arisen and plans for the coming day. This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.

Scrum meetings should be *short and focused*. To dissuade team members from getting involved in long discussions, they are sometimes organized as 'stand-up' meetings where there are no chairs in the meeting room.

During a scrum, the *sprint backlog is reviewed*. *Completed items* are *removed* from it. New items may be *added* to the backlog as new information emerges. The team then *decide who should work* on sprint backlog items that day.

Agile activities

Scrum does not suggest the technical agile activities that should be used. However, two practices should always be used in a sprint.

Test automation

As far as possible, product testing should be automated. A suite of executable tests, that can be run at any time, should be developed.

Continuous integration

Whenever anyone makes changes to the software components they are developing, these components should be immediately integrated with other components to create a system. This system should then be tested to check for unanticipated component interaction problems.

Code completeness checklist

Reviewed

The code has been reviewed *by another team member* who has checked that it *meets* agreed coding standards, is *understandable*, includes *appropriate comments*, and has been *refactored* if necessary.

Unit tested

All unit tests have been *run* automatically and all tests have executed *successfully.*

Integrated

The code has been integrated with the project codebase and *no integration errors* have been reported.

Integration tested

All integration tests have been run automatically and all tests have *executed success*fully.

Accepted

Acceptance tests have been run if appropriate and the product owner or the development team have confirmed that the product backlog item has been completed.

Sprint reviews

At the *end of each sprint*, there is a *review meeting*, which involves the whole team:

- reviews whether or not the *sprint has met its goal*.
- sets out any *new problems and issues that have emerged* during the sprint.
- is a way for a team to *reflect on how they can improve* the way they work.

The product owner (as ultimate authority to decide whether or not the goal of the print has been achieved) should confirm that the implementation of the selected product backlog items is complete.

The sprint review should include a *process review*, in which the team reflects on its own way of working and how Scrum has been used.

- The aim is to identify ways to improve and to discuss how to use Scrum more productively.

Self-organizing teams

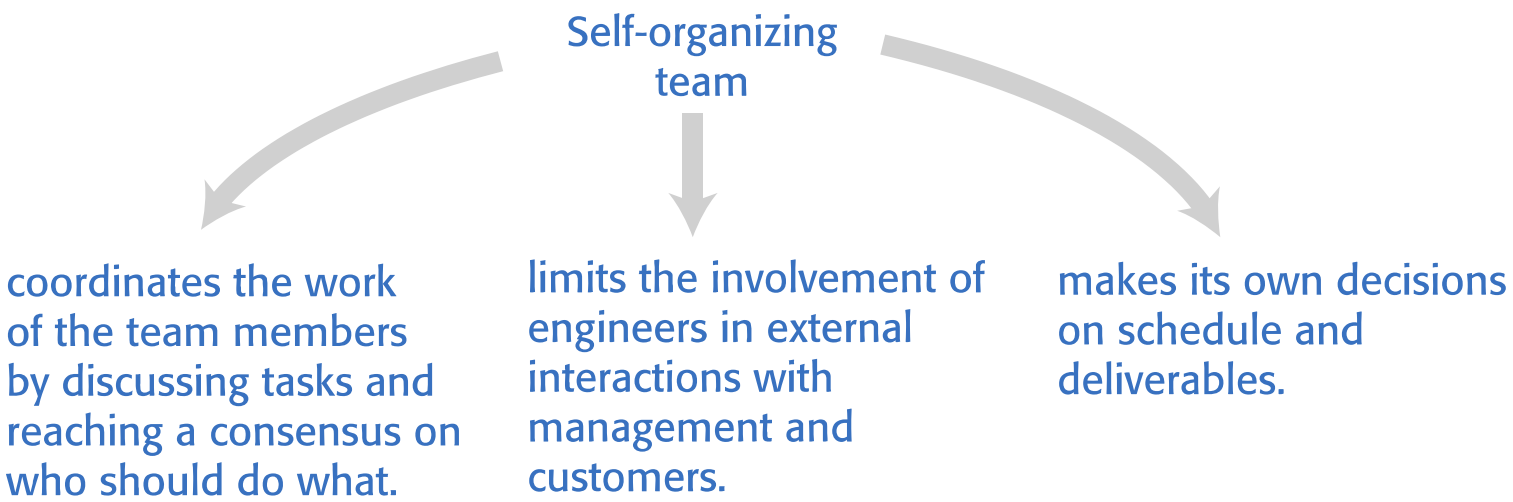


Figure 2.9 Self-organizing teams
Ian Sommerville – Engineering Software Products

Team size and composition

The ideal Scrum team size is between 5 and 8 people.

- Teams have to tackle diverse tasks and so usually require people with different skills, such as networking, user experience, database design and so on.
- They usually involve people with different levels of experience.
- A team of 5-8 people is large enough to be diverse yet small enough to communicate informally and effectively and to agree on the priorities of the team.

The advantage of a self-organizing team is that it can be a cohesive team that can adapt to change.

- Because the team rather than individuals take responsibility for the work, they can cope with people leaving and joining the team.
- Good team communication means that team members inevitably learn something about each other’s areas

Team coordination

Ideally :

- Teams would be *co-located* (in the same room) and could *communicate informally*.
- *Daily scrums* mean that the team members know *what is been* done and *what others are doing*.

Assumptions that are not always correct:

- Scrum assumes that the team will be made up of full-time workers who share a workspace. In reality, team members may be part-time and may work in different places. For a student project team, the team members may take different classes at different times.
- Scrum assumes that all team members can attend a morning meeting to coordinate the work for the day. However, some team members may work flexible hours (e.g. because of childcare responsibilities) or may work on several projects at the same time.

External interactions

In Scrum, the idea is that *developers should focus on development* and only the *ScrumMaster* and *Product Owner* should be *involved in external interactions*.

The intention is that the *team* should be able to work on software development *without external interference or distractions*.

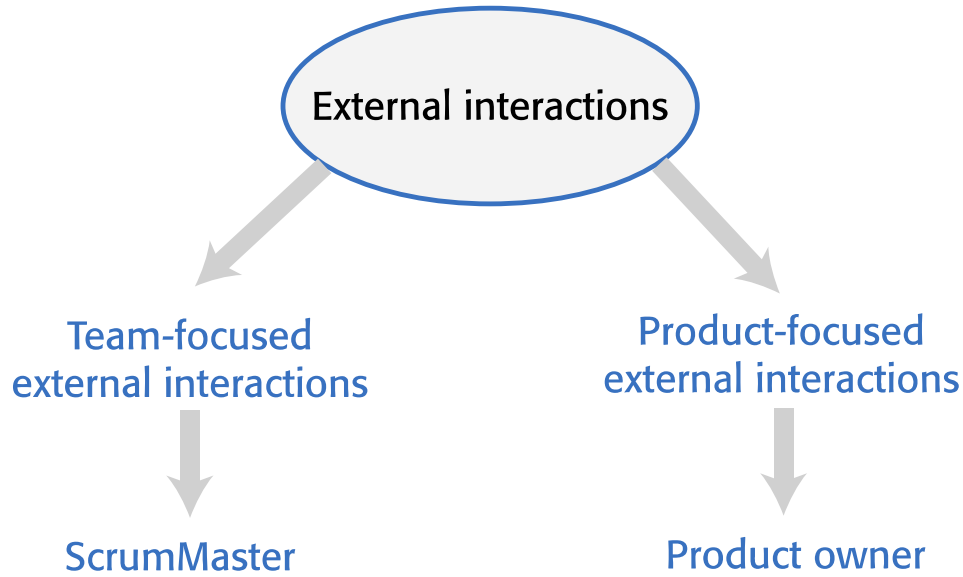


Figure 2.10 Managing external interactions
Ian Sommerville – Engineering Software Products

Project management

In many companies, the ScrumMaster has to take on project management responsibilities.

He knows the work going on and is in the best position to provide accurate information and project plans and progress.

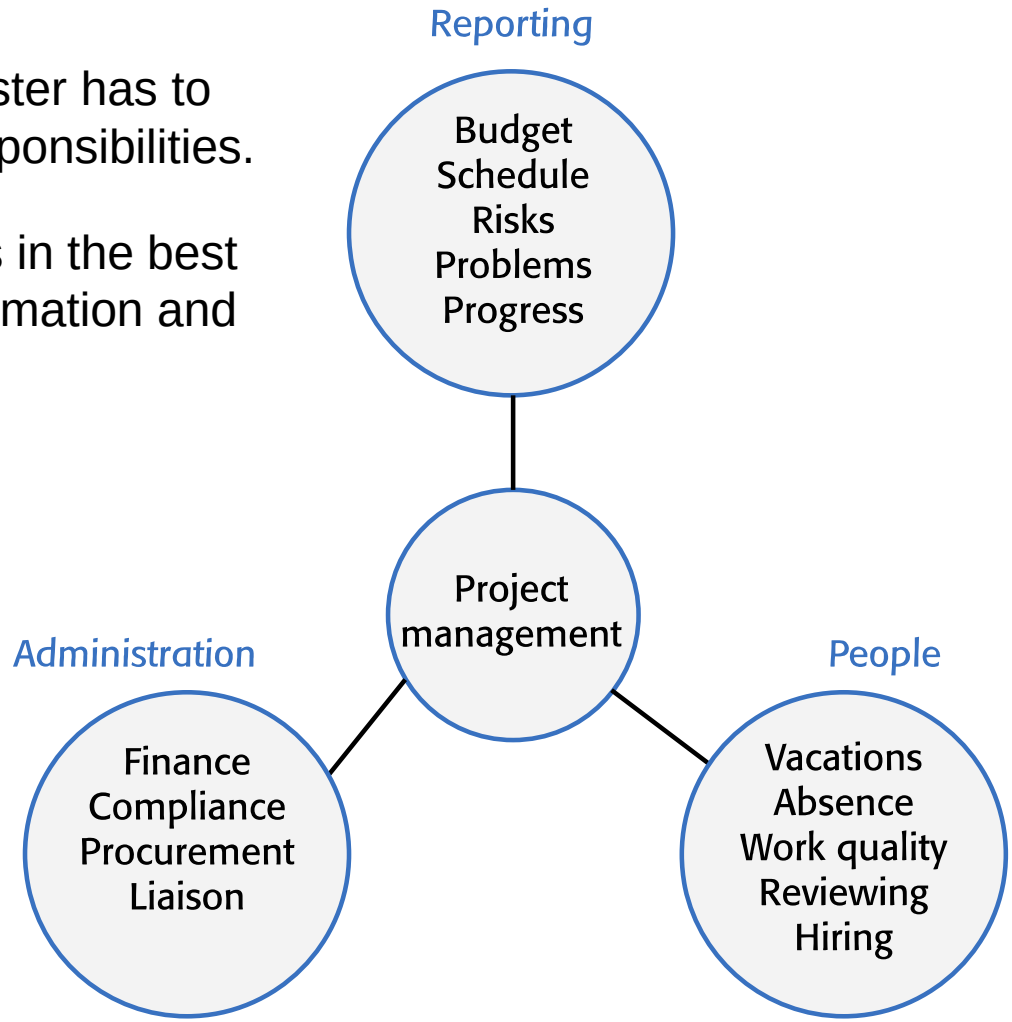


Figure 2.11 Project management responsibilities

Ian Sommerville – Engineering Software Products

Formative evaluation

1. In what relation is 'product backlog' with 'spring backlog' ?

2. Why is it important that each sprint should normally produce a ‘potentially shippable’ product increment? When might the team relax this rule and produce something that is not ‘ready to ship’?

3. Scrum has been designed for use by a team of 5-8 people working together to develop a software product. What problems might arise if you try to use Scrum for student team projects where a group work together to develop a program. What parts of Scrum could be used in this situation?

https://forms.gle/ESDYTtC2z8dfa9zWA

Key points

The best way to develop software products is to use agile software engineering methods that are geared to rapid product development and delivery.

Agile methods are based around iterative development and the minimization of overheads during the development process.

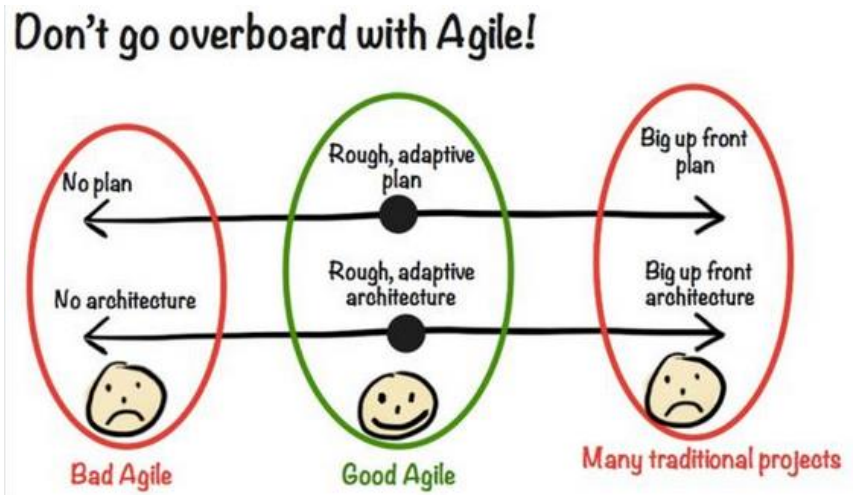
Extreme programming (XP) is an influential agile method that introduced agile development practices such as user stories, test-first development and continuous integration. These are now mainstream software development activities.

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.

Key points

Agile methods may be scaled up to large systems and scaled out to large companies. Scalling agile methods up to large systems is difficult. Large systems need models and documentation to be realized in advance.

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.



https://twitter.com/michelnadeau7/status/527798932269973504

Key points

Scrum is an agile method that focuses on agile planning and management. Unlike XP, it does not define the engineering practices to be used. The development team may use any technical practices that they believe are appropriate for the product being developed.

In Scrum, work to be done is maintained in a product backlog – a list of work items to be completed. Each increment of the software implements some of the work items from the product backlog.

Sprints are fixed-time activities (usually 2–4 weeks) where a product increment is developed. Increments should be 'potentially shippable' i.e. they should not need further work before they are delivered.

A self-organizing team is a development team that organizes the work to be done by discussion and agreement amongst team members.

Scrum practices such as the product backlog, sprints and self-organizing teams can be used in any agile development process, even if other aspects of Scrum are not used.