



3. Agile Software Development

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

Learning Outcomes

- Define agile software development.
- Understand the rationale for agile software development methods and agile manifesto.
- Compare and contrast agile and plan-driven development.
- Identify the key practices in extreme programming and how these relate to general practices of agile methods.
- Understand scrum / Kanban approach to agile software development.

Most Software Developments Fail

- Software developments projects being cancelled every now and then
- Software projects are being considered as failures by those who initiated it.
- One in every two projects exceed its budget by 200%.

Why?

- Do not meet the need of the user.
- Deadline rush.
- Less number of features delivered.
- Poor interfaces.



- Success is NOT just functions any more!!!

Figure 1

LOT more EXPECTATIONS from Users

- Speed Delivery
- Accuracy
- Features vs Usability



Figure 2

Waterfall model build-upon Assumptions

- Requirements can be entirely predicted upfront.
- Each phase of the lifecycle can be perfected before moving to the next.
- Timeframes and budgets can be predicted up front.
- The feedback from the real user is not so valuable.

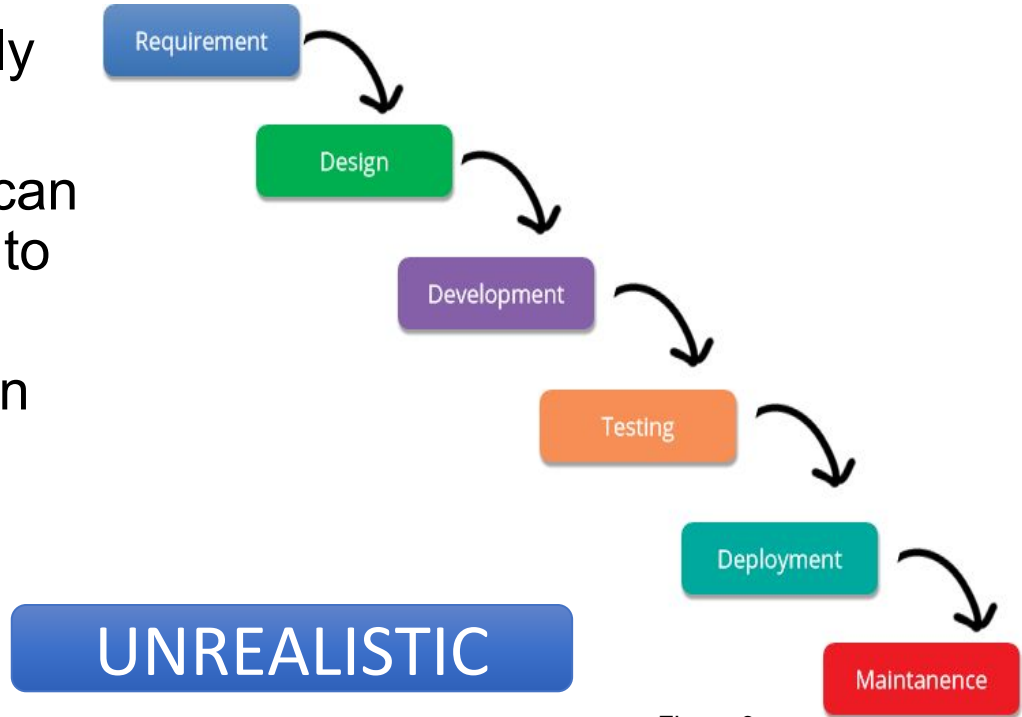


Figure 3

Causes for Failures

- Risk is bundled!

Functions	Functions	Functions	Functions
Functions	Functions	Functions	Functions
Functions	Functions	Functions	Functions
Functions	Functions	Functions	Functions



Figure 4



Figure 5

Causes for Failures Cont...

- Poor communication!
- Feedback is obtained at the end of the project

Beginning

End



Figure 6



Figure 7



Figure 8

Can you meet the client requirement at this stage????

Rapid Software Development

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



3.1 Agile Methods

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

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

- ① **CUSTOMER**
COLLABORATION
over contract negotiation
- ② **INDIVIDUALS** AND
INTERACTIONS
over processes and tools
- ③ **RESPONDING** TO
CHANGE
over following a plan
- ④ **WORKING**
SOFTWARE
over full documentation

www.softwaretestingclass.com

Figure 8

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 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.

12 Agile Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	7. Working software is the primary measure of progress.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	9. Continuous attention to technical excellence and good design enhances agility.
4. Business people and developers must work together daily throughout the project.	10. Simplicity--the art of maximizing the amount of work not done--is essential.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	11. The best architectures, requirements, and designs emerge from self-organizing teams.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

What is Agile Software Development?

- An umbrella term for a set of methods and practices based on the values and principles expressed in the Agile Manifesto.

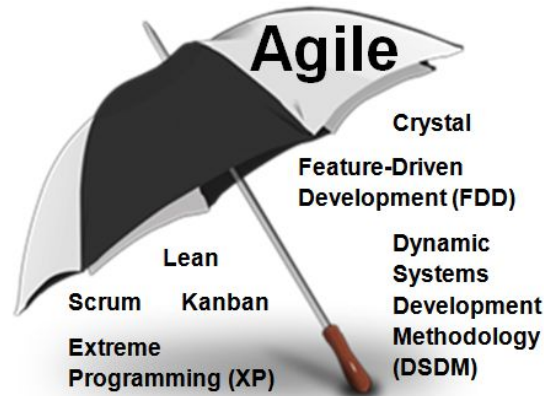


Figure 9

What is Agile Software Development? Cont...

- 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

What is Agile Software Development? Cont...

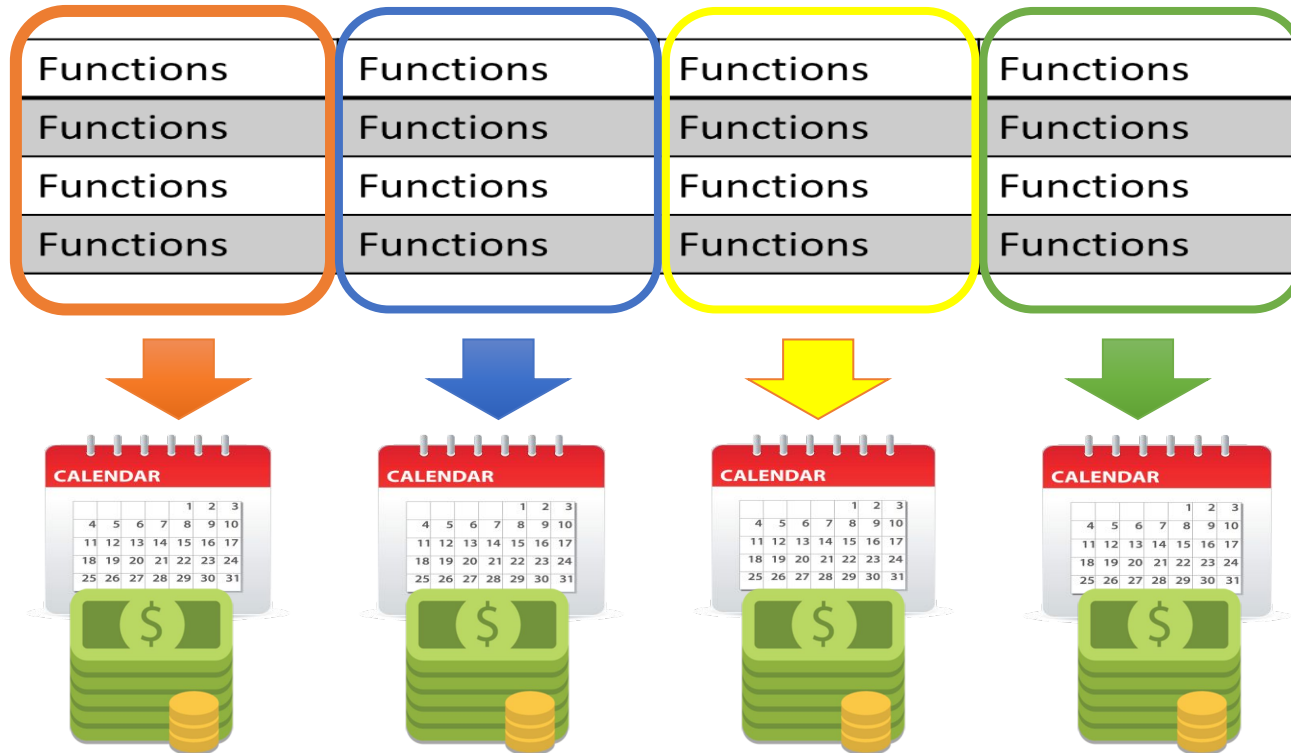
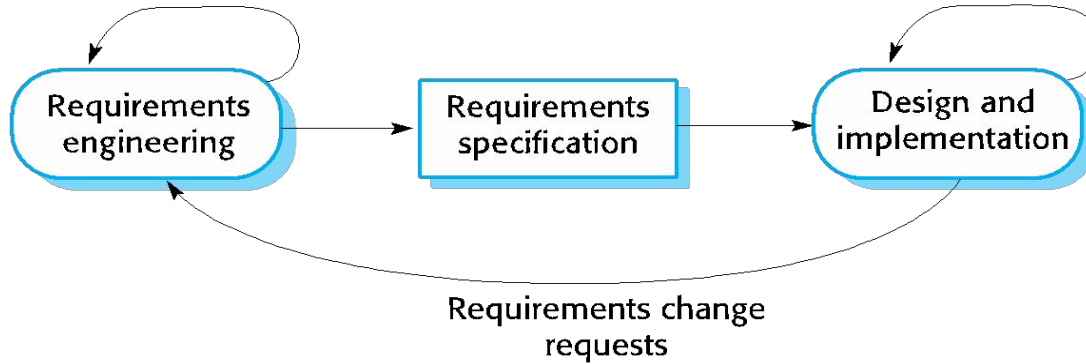


Figure 10

Plan-Driven vs Agile Development

Plan-based development



Agile development

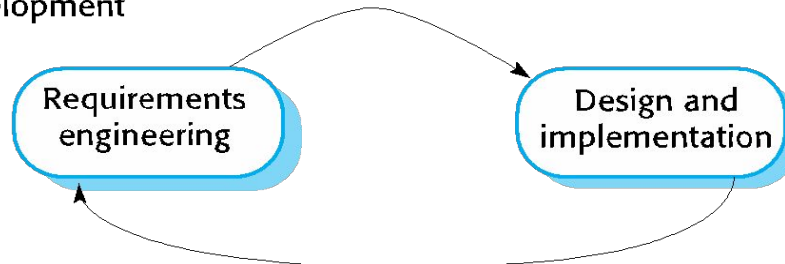
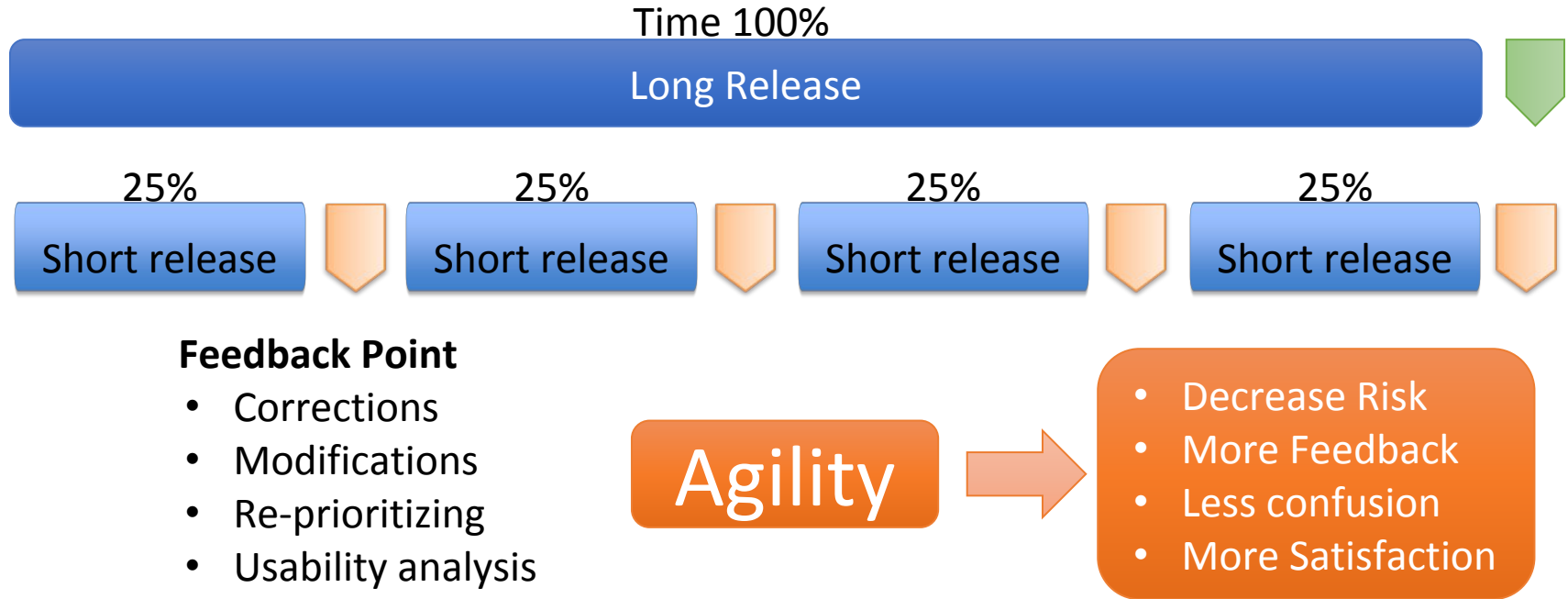


Figure 11

Is Agile better?



Agile Software Development: Successes

- Small or medium sized product development.
- Custom system development within organization
- Experiments in using agile approaches for critical systems engineering.
 - Security, safety, and dependability analysis
 - Need significant modifications before applying.

Drawbacks / Difficulties

- Custom representatives are subject to other pressures and cannot take full part in the software development.
- Individual team members may not have suitable personalities for the intense involvement / team work.
- Prioritizing changes can be extremely difficult when there are lot of stakeholders.
- Under pressure from delivery schedules, the team may not have time to carry system simplifications.
- Cultural change – team members may find it hard to change the practice (informal / defined by team itself)



3.2 Agile Development Techniques

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

eXtreme Programming(XP)

- XP is an Agile Software Development Process created by Kent Beck in the mid 1990's
- A set of 12 key practices taken to their “extremes”
- A mindset for developers and customers
- 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.
 - Deliver most needed when it's needed.



Figure 12

The 12 Practices

1. The Planning Game
2. Small Releases
3. Metaphor
4. Simple Design
5. Testing
6. Refactoring
7. Pair Programming
8. Collective Ownership
9. Continuous Integration
10. 40-Hour Workweek
11. On-site Customer
12. Coding Standards

XP Practices 1. The Planning Game

- Planning for the upcoming iteration
- **User stories** provided by the customer
- Technical persons determine schedules, estimates, costs, etc
- A result of collaboration between the customer and the developers
- Advantages / Disadvantage?

What is a User Story?

- a reminder to have a conversation with your customer (in XP, project stakeholders are called customers),
- it's a reminder to do some just-in-time analysis.
- user stories are very slim and high-level requirements artifacts.



Figure 13

What is a User Story? Cont...

- Small, much smaller than other user requirements such as use cases or scenarios
- Represent a single story


- Students can purchase monthly parking passes online.
- Parking passes can be paid via credit cards.
- Parking passes can be paid via PayPal.
- Professors can input student marks.
- Students can obtain their current seminar schedule.
- Students can order official transcripts.
- Students can only enroll in seminars for which they have prerequisites.
- Transcripts will be available online via a standard browser.

User Stories

- Stakeholders write user stories.
- Use the simplest tool.
- Remember non-functional requirements.
- Indicate the estimated size.
- Indicate the priority.
- Optionally include a unique identifier.

User Stories- Card (Informal)

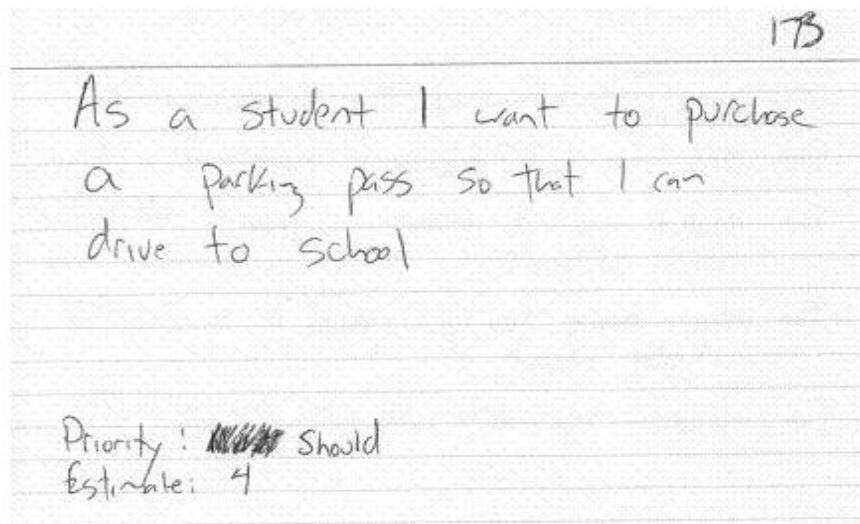
173. Students can purchase parking passes.

Priority:  8

Estimate: 4

Bit formal User Story!

As a (role), I want (feature), so that (benefit).



User Story – More Examples

As...	Conditional	I want...	So that...
As an HR Rep	who is authorized to initiate reviews for new employees	I want to be notified when new hires have reached their 90 day mark	so that I can initiate a 90-day review.
As an HR Rep	who has initiated a 90-day review	I want to notify the new hire of all of the requirements of the 90-day review	so they can begin to submit their evaluation in the system.
As an HR Rep	who has initiated a 90-day review	I want to notify the new hire of all of the requirements of the 90-day review	so they can begin to submit their evaluation in the system.
As an employee	who is under 90-day review	I want to create a log in for the HR review system	so that I can log in to submit my 90-day evaluation.
As an employee	who is under 90-day review	I want to log in to the system	so that I can view the requirements for my 90-day evaluation.
As an employee	who is under 90-day review	I want to submit the names of two peers I have worked with since being hired	so that they can contribute to my 90-day review.
As an employee	selected to peer review a new hire	I want to be notified when I have been selected to review a new hire after 90 days	so that I can log in to the system and submit my evaluation.

User Stories for Requirements

- 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.

Activity

- Write user stories about your dream vehicle.

XP Practices 2. Small Releases

- Small in terms of functionality
- Less functionality means releases happen more frequently
- Support the planning game
- The minimal useful set of functionality that provides business value is developed first.
- Advantages / Disadvantages?

XP Practices 3. Metaphor

- The oral architecture of the system
- A common set of terminology
- Advantages / Disadvantages?

XP Practices 4. Simple Design

- Enough design is carried out to meet the current requirements and no more.
- Do as little as needed, nothing more
- Advantages / Disadvantages?

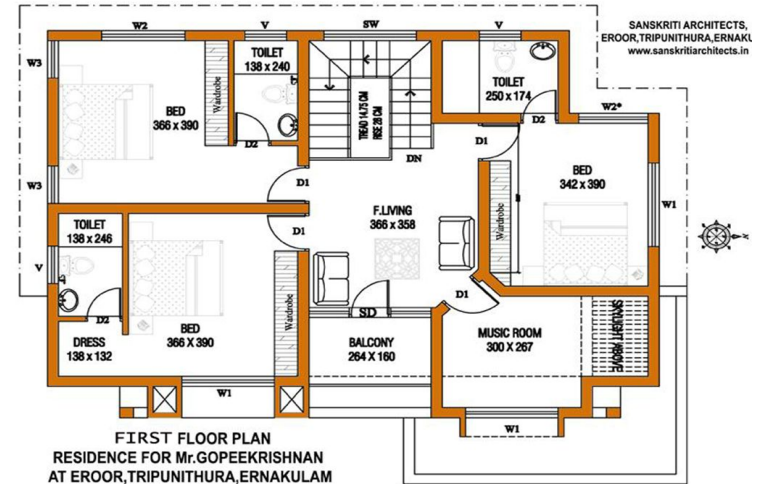


Figure 14

XP Practices 5. Testing: Key Features

The program is tested after every change has been made.

- Test-First development
 - Writing tests before code.
- Incremental test development from scenarios.
- User involvement in test development and validation.
- Use of automated testing frameworks
 - 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: Testing in XP

- Help develop acceptance tests for the stories that are to be implemented in the next release of the system.
- Writes tests as development proceeds. All new code is therefore validated to ensure that it is what the customer needs.
- However, people adopting the customer role have limited time available and so cannot work full-time with the development team. They may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process.

XP Practices 6. Refactoring

- Changing how the system does something but not what is done
 - Re-organization of a class hierarchy to remove duplicate code.
 - Tidying up and renaming attributes and methods to make them easier to understand
- All developers are expected to refactor the code continuously as soon as possible code improvements are found.
- Improves the quality of the system in some way
- This keeps the code simple and maintainable.

Refactoring: Pros & Cons

- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is well-structured and clear.
- However, some changes requires architecture refactoring and this is much more expensive.

XP Practices 7. Pair Programming

- Two Developers, One monitor, One Keyboard
- Developing the code together
- One “drives” and the other thinks
- Switch roles as needed

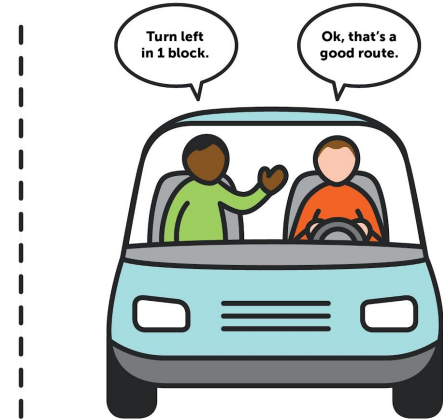
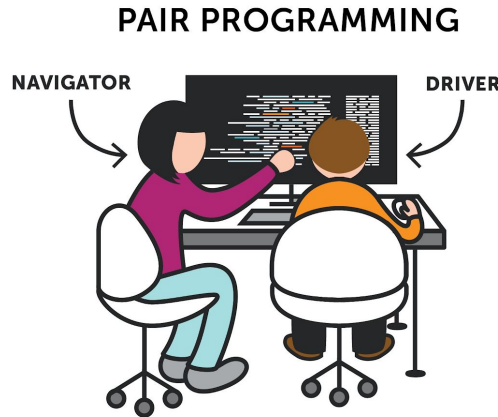


Figure 15

Pair Programming: Pros & Cons

- 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.
- Pairs are created dynamically so that all team members work with each other during the development process.
- The sharing of knowledge reduces the overall risks to a project when team members leave.
- A pair working together is more efficient than 2 programmers working separately.

XP Practices 8. 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.
- The idea that all developers own all of the code
- Enables refactoring
- Advantages / Disadvantages?

XP Practices 9. Continuous Integration

- As soon as the work on a task is complete, it is integrated into the whole system.
- New features and changes are worked into the system immediately
- After any such integration, all the unit tests in the system must pass.
- Code is not worked on without being integrated for more than a day
- Pros / Cons?

XP Practices 10. 40-Hour Week/Sustainable Pace

- The work week should be limited to 40 hours
- Regular overtime is a symptom of a problem and not a long term solution
- Pros / Cons?

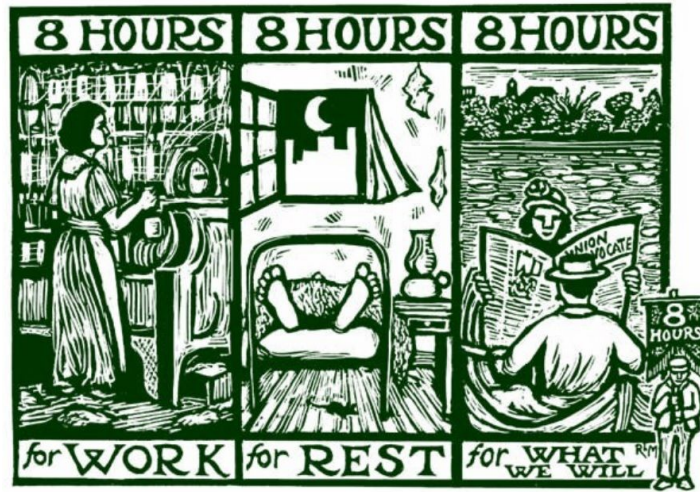


Figure 16

XP Practices 11. 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.
- Gives quick and continuous feedback to the development team

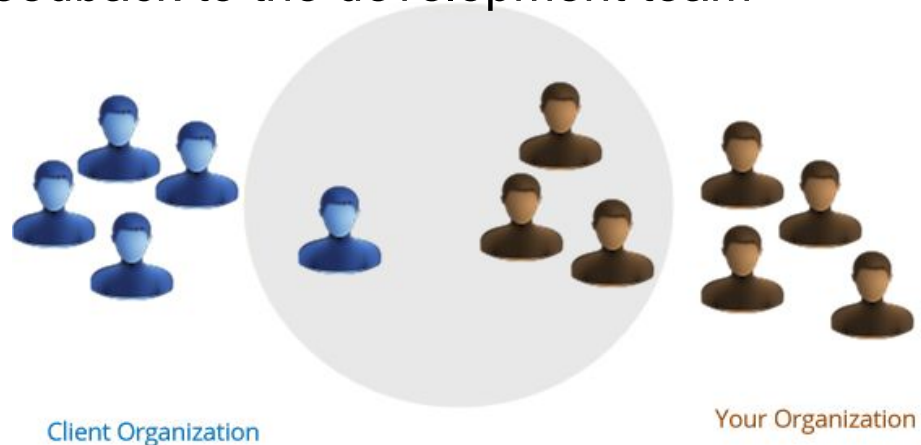


Figure 17

XP Practices 12. Coding Standards

- All code should look the same
- It should not possible to determine who coded what based on the code itself

The extreme programming release cycle

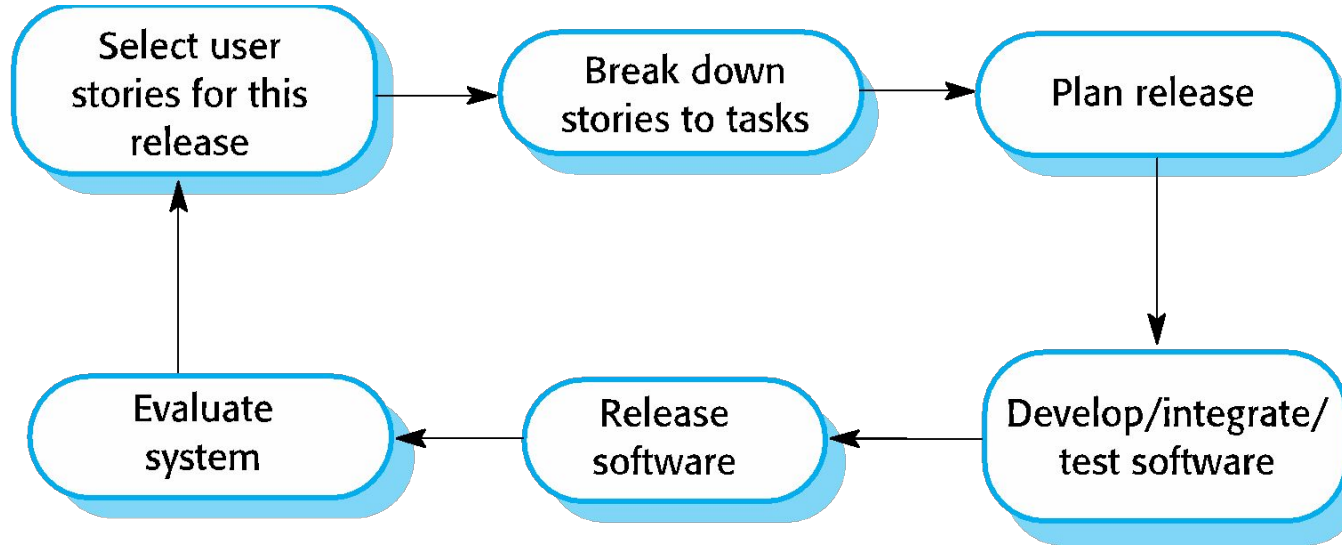


Figure 18

XP and Agile Principles

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

XP: Advantages & Disadvantages

- Advantages

- Built-In Quality
- Overall Simplicity
- Programmer Power
- Customer Power
- Synergy Between Practices

- Disadvantages

- Informal, little, or no documentation
- Contract Issues



3.3 Agile Project Management

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

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. Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables.
- 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.

What is SCRUM?

- Scrum is an Agile framework for completing complex projects.
- Achieve highest business value within the shortest time.
- Scrum is:
 - Lightweight
 - Simple to understand
 - Difficult to master



Figure 19

SCRUM

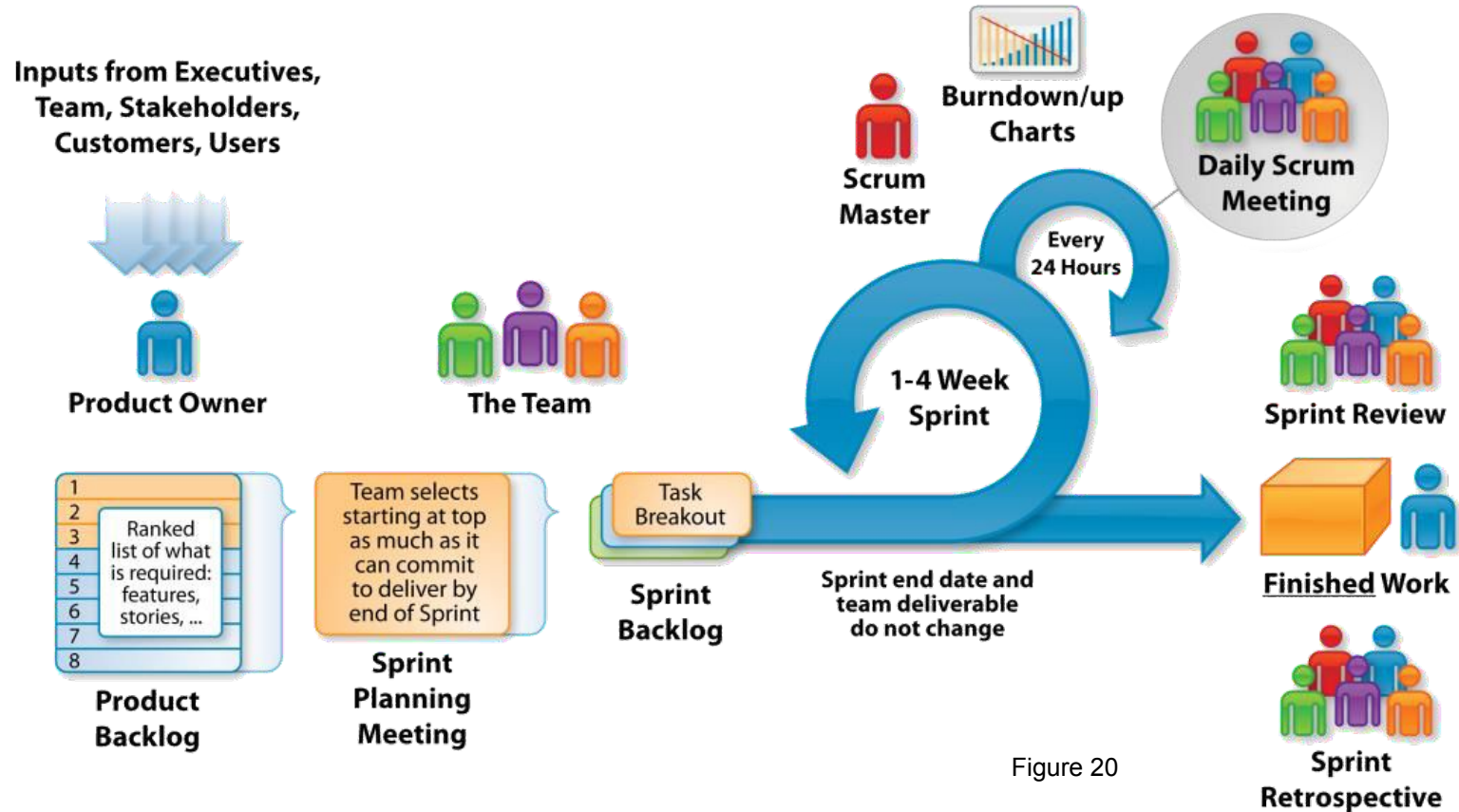


Figure 20

The SCRUM Roles

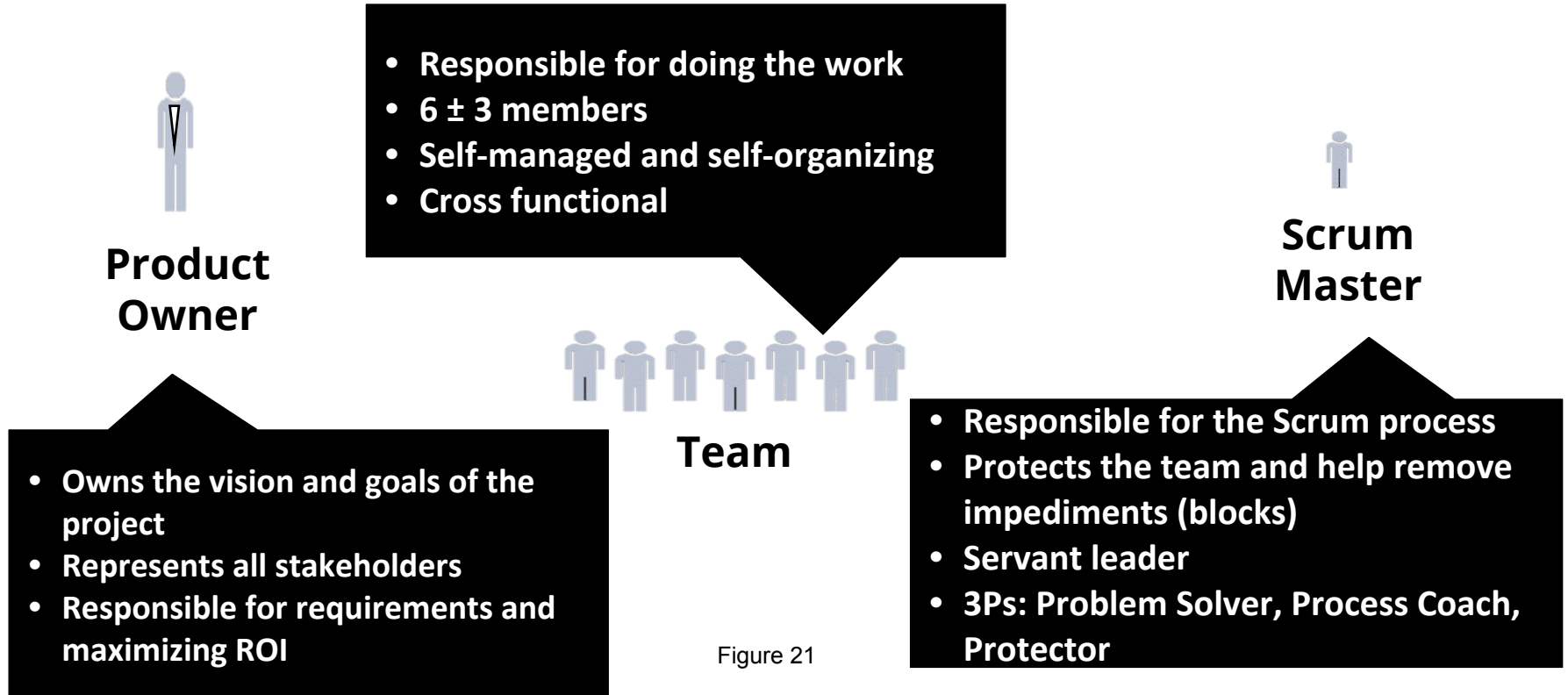


Figure 21

SCRUM Three Pillars

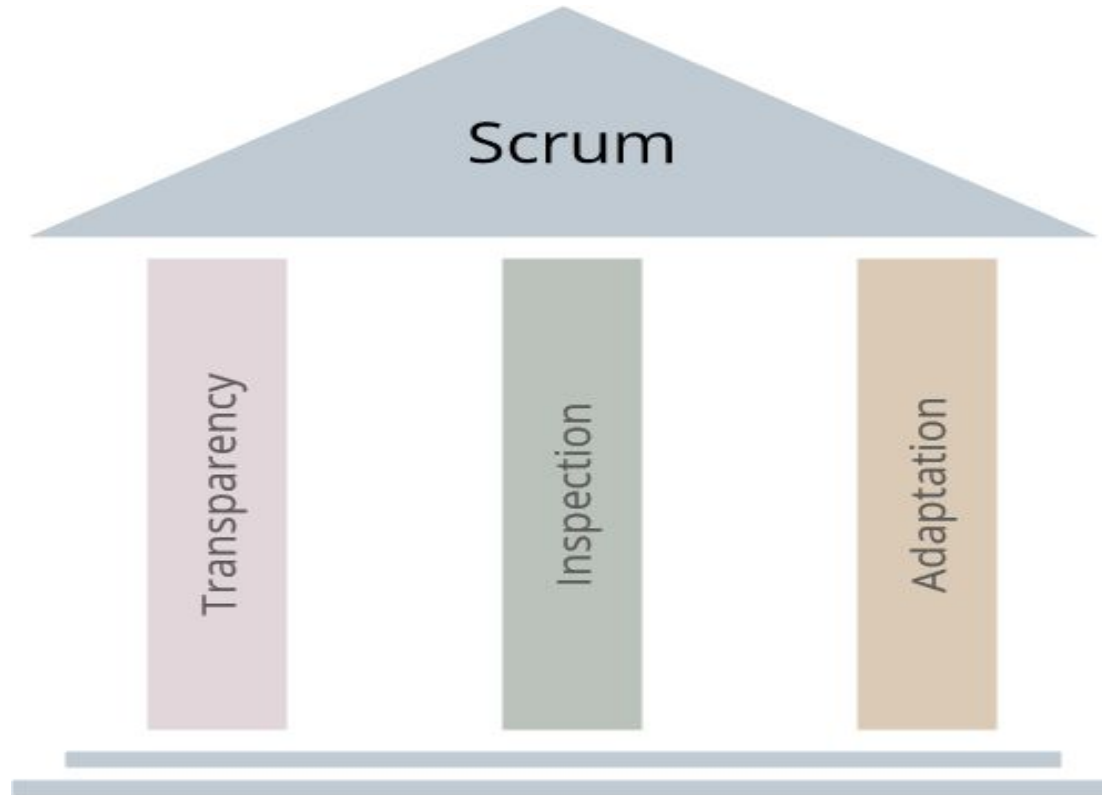


Figure 22

© 2020 e-Learning Centre, UCSC

SCRUM Three Phases

- 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.

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.
ScrumMaster	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 1-4 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

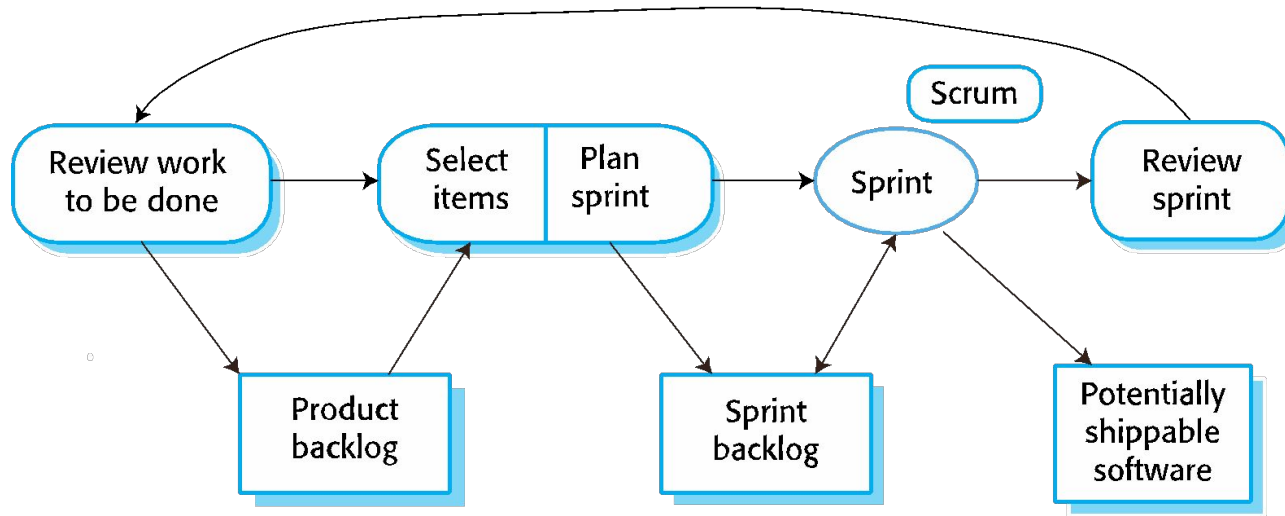


Figure 23

SCRUM Sprint Cycle

- Sprints are fixed length, normally 1–4 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.
- 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.
 - 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 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.



3.4 Scaling Agile Methods

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

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 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.

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 by 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 and Plan-Driven Methods

- Most projects include elements of plan-driven and agile processes. Deciding on the balance depends on:
 - 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	<p>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.</p> <p>Where there are external stakeholders, such as regulators, it is difficult to represent their views to the agile team.</p>
Embrace change	<p>Prioritizing changes can be extremely difficult, especially in systems for which there are many stakeholders. Typically, each stakeholder gives different priorities to different changes.</p>
Incremental delivery	<p>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 what product features several months in advance to prepare an effective marketing campaign.</p>
Maintain simplicity	<p>Under pressure from delivery schedules, team members may not have time to carry out desirable system simplifications.</p>
People not process	<p>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.</p>

Agile and Plan-Based Factors

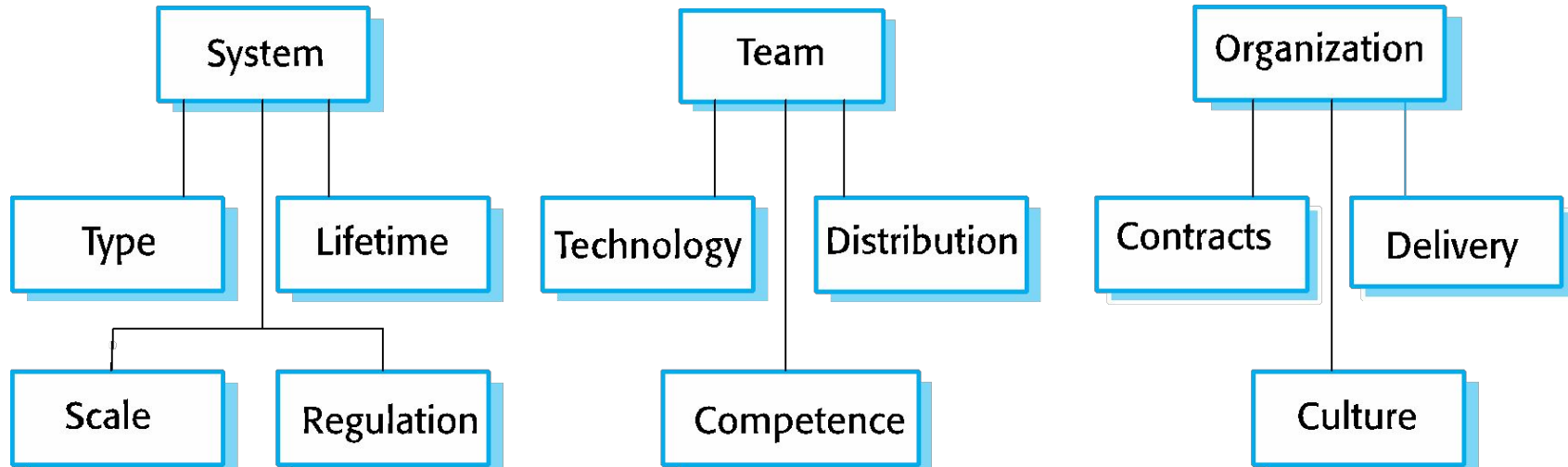


Figure 24

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 distributed.
- 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

- 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.

Large System Development

- 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.

Factors in Large Systems

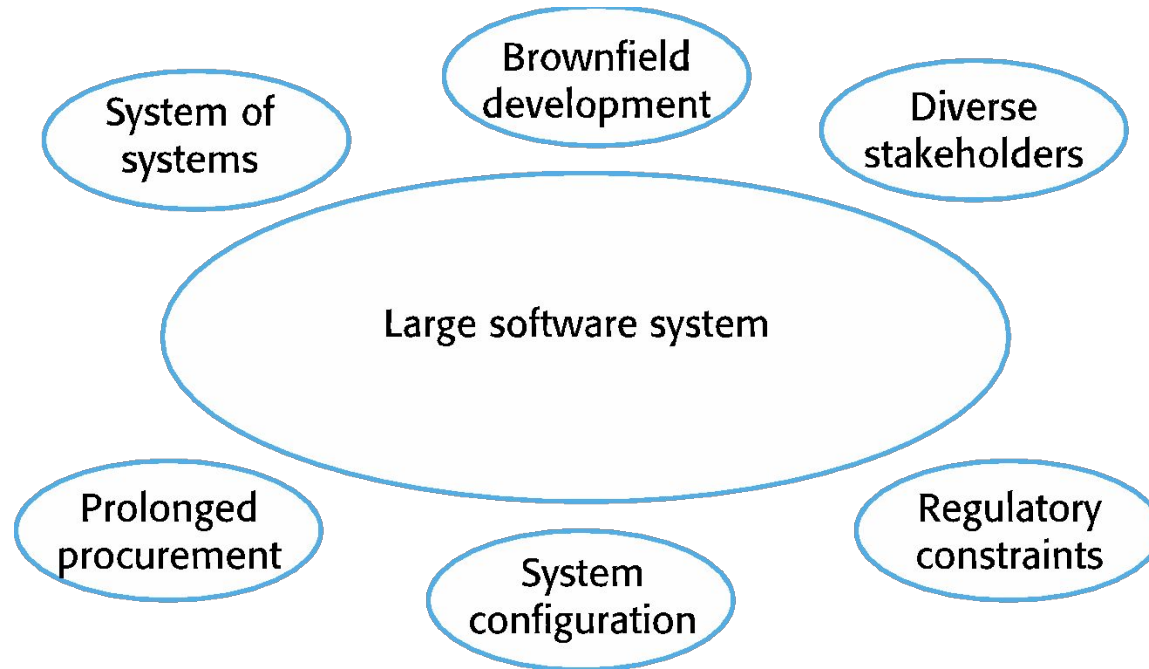


Figure 25

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.

Activity

- Explain how the principles underlying agile methods lead to the accelerated development and deployment of software.
- Explain why is it necessary to introduce some methods and documentation from plan-based approaches when scaling agile methods to larger projects that are developed by distributed development teams?

Summary

- 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.

Summary

- 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.