

Extreme Programming (XP)

- An agile method, developed in the late 1990s, that introduced a range of agile development techniques.
- 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.

2

XP principles and practices

- **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'.

3

XP principles and practices

- **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.
- **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.

4

XP principles and practices

- **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.

5

XP principles and practices

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

6

XP principles and practices

- **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 pass.
- **Sustainable pace.**
 - Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity



7

XP principles and practices

- **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.



8

Influential XP practices

- The following are the XP Key practices
 - User stories for specification
 - Refactoring
 - Test-first development
 - Pair programming



9

User stories for requirements

- In XP, a customer 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.



10

User stories

- 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.



11

Refactoring

- 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.



12

Refactoring

- Rather, it proposes constant code improvement (refactoring) to make changes easier when they have to be implemented.
- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.



13

Refactoring

- 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.



14

Test-first development

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



15

Test-first development

- Instead of writing some code and then writing tests for that code, you write the tests before you write the code.
- 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.



16

Test-first development

- 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.



17

Test automation




- Tests are written as executable components before the task is implemented
 - These testing components should be stand-alone, should simulate the submission of input to be tested and should check that the result meets the output specification.



18

Test automation




- An automated test framework (e.g. JUnit) is a system that makes it easy to write executable tests and submit a set of tests for execution.



19

Test automation




- 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.



20

Pair programming




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



21

Pair programming




- It encourages refactoring as the whole team can benefit from improving the system code.
- In pair programming, programmers sit together at the same computer to develop the software.
- Pairs are created dynamically so that all team members work with each other during the development process.






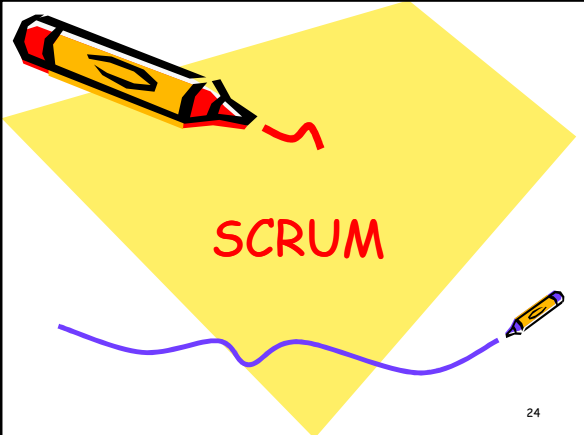
22

Pair programming

- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- Pair programming is not necessarily inefficient and there is some evidence that suggests that a pair working together is more efficient than 2 programmers working separately.



23



24

Scrum

- A software development method Originally proposed by Schwaber and Beedle (an activity occurs during a rugby match) in early 1990.
- Scrum is an agile method that focuses on managing iterative development rather than specific agile practices.



25

Scrum

- 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.



26

Scrum

- 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.



27

Scrum

- Scrum—distinguishing features
 - Development work is partitioned into "packets"
 - Testing and documentation are on-going as the product is constructed
 - Work units occurs in "sprints" and is derived from a "backlog" of existing changing prioritized requirements



28

Scrum

- Changes are not introduced in sprints (short term but stable) but in backlog.
- Meetings are very short (15 minutes daily) and sometimes conducted without chairs (what did you do since last meeting? What obstacles are you encountering? What do you plan to accomplish by next meeting?)



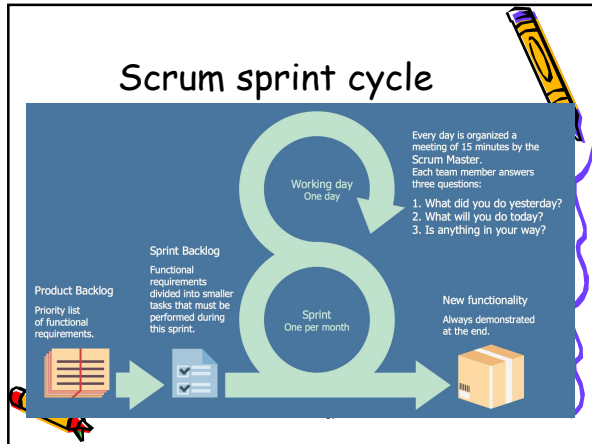
29

Scrum

- "demos" are delivered to the customer with the time-box allocated.
- May not contain all functionalities.
- So customers can evaluate and give feedbacks.



30



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 product owner to select the features and functionality from the product backlog to be developed during the sprint.

The Scrum sprint cycle

- Once these are agreed, the team organize themselves to develop the software (assignments and evaluations of the tasks).
- Production of the sprint backlog, that contains all the task that will be developed in that sprint.
- Start the sprint

The Scrum sprint cycle

- During this stage the team is isolated from the customer and the organization, with all communications channeled through the so-called 'Scrum master'.
- The role of the Scrum master is to protect the development team from external distractions.

The Scrum sprint cycle

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

Some Basic Terminology

Scrum	Agile	Definition
Sprint	Iteration	Fixed-length period of time (timebox)
Release	Small Release	Release to production
Sprint/Release Planning	Planning Game	Agile planning meetings
Product Owner	Customer	Business representative to project
Retrospective	Reflection	"Lessons learned"-style meeting
ScrumMaster	Coach	Agile project manager
Development Team	Team	Empowered Cross-Functional team
Daily Scrum	Daily Standup	Brief daily status meeting

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.



37

Teamwork in Scrum

- 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.



38

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.



39

Scrum benefits

- 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.



40

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.



41

Scaling agile methods




- The need for faster delivery of software, which is more suited to customer needs, also applies to larger systems, and therefore to larger companies.
- Scaling up agile methods involves changing these to cope with larger, longer projects where there are multiple development teams, perhaps working in different locations.



42

Scaling out and 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.



43




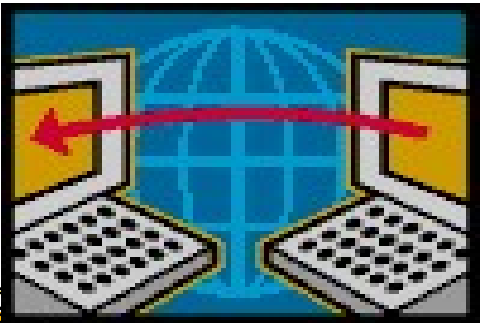
Scaling out and up

- 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.



44

Questions



45