Software Engineering

Week # 4

LECTURER: ABDULRAHMAN QAIM

Agile Software Development

Agile methods

Plan-driven and agile development

Extreme programming

Agile project management

Scaling agile methods

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.

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

Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods.

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

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

The principles of agile methods

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

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 not a lot of external rules and regulations that affect the software.

Because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems.

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 characterises agile methods.

Prioritising 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.

Extreme programming

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

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.

XP and agile principles

Incremental development is supported through small, frequent system releases.

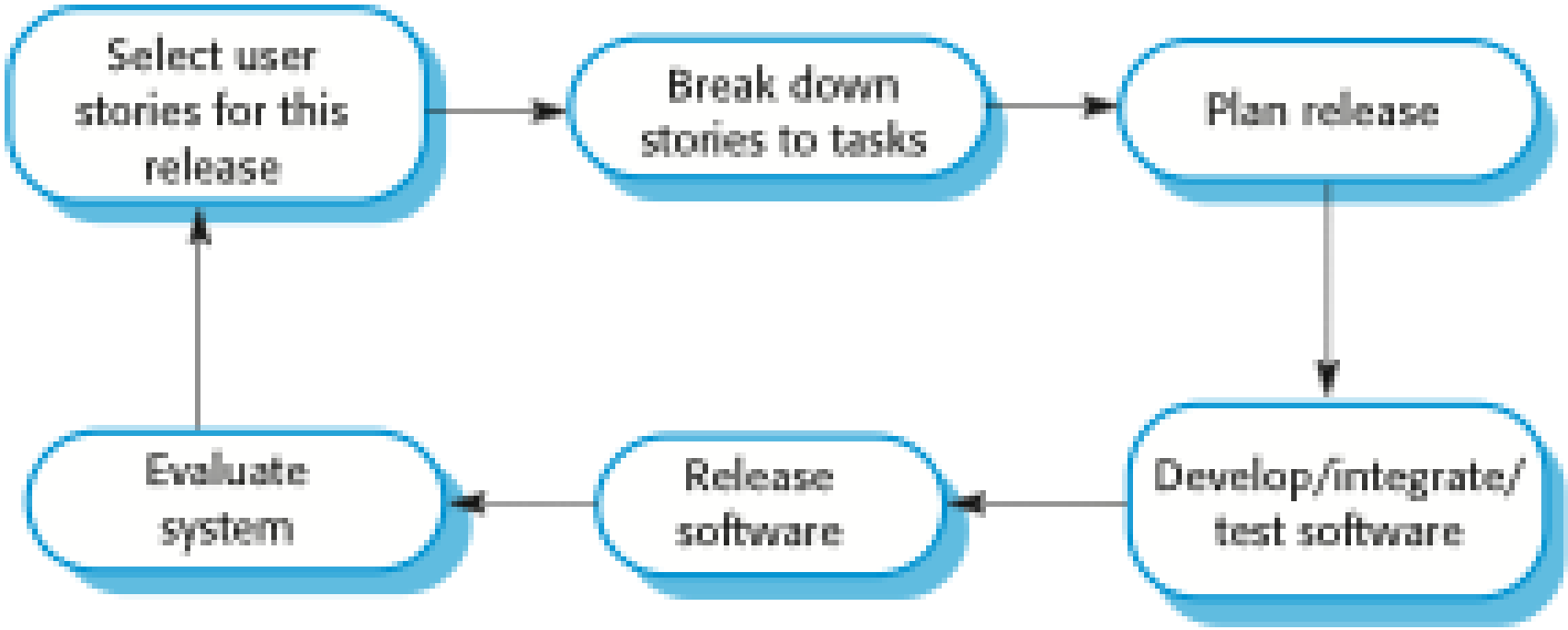
Customer involvement means full-time customer engagement with the team.

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.

The extreme programming release cycle



Extreme programming practices (a)

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'. See Figures 3.5 and 3.6.
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.
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.

Extreme programming practices (b)

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.
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
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.

Extreme programming – Story Card

STORY CARD NO: 16	Project Name E-Commerce	Estimation: 4 Hours
Story Name: User Registration		Date: 16/08/2007 1:30 PM
STORY: User needs to register with unique username and password before purchasing anything from the online store		Acceptance Test: 1. User Id must be unique 2. Try to register with duplicate user id and Password 3. Try to register user name only 4. Try to register with password only 5. Forget Password Link
Note: User Can View or Visit store as a Visitor but needs to register before purchasing anything		Risk: Low
Points to be Consider: There isn't any non-functional requirement at this stage		

Requirements scenarios

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 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 the schedule estimates.

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

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.