# EBU5304 – Software Engineering

## Software Processes and Agile

- Topics
  1. What is software process?
  2. Traditional models
     - Waterfall
     - Evolutionary development (Incremental)
     - The Rational Unified Process
  3. Modern models
     - Agile

# Software process

- Software process: a structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - Requirement Specification – defining what the system should do;
  - Development (including analysis, design, implementation) – defining the organisation of the system and implementing the system;
  - Validation (Testing) – checking that it does what the customer wants;
  - Evolution – changing the system in response to changing customer needs.

# Software process models

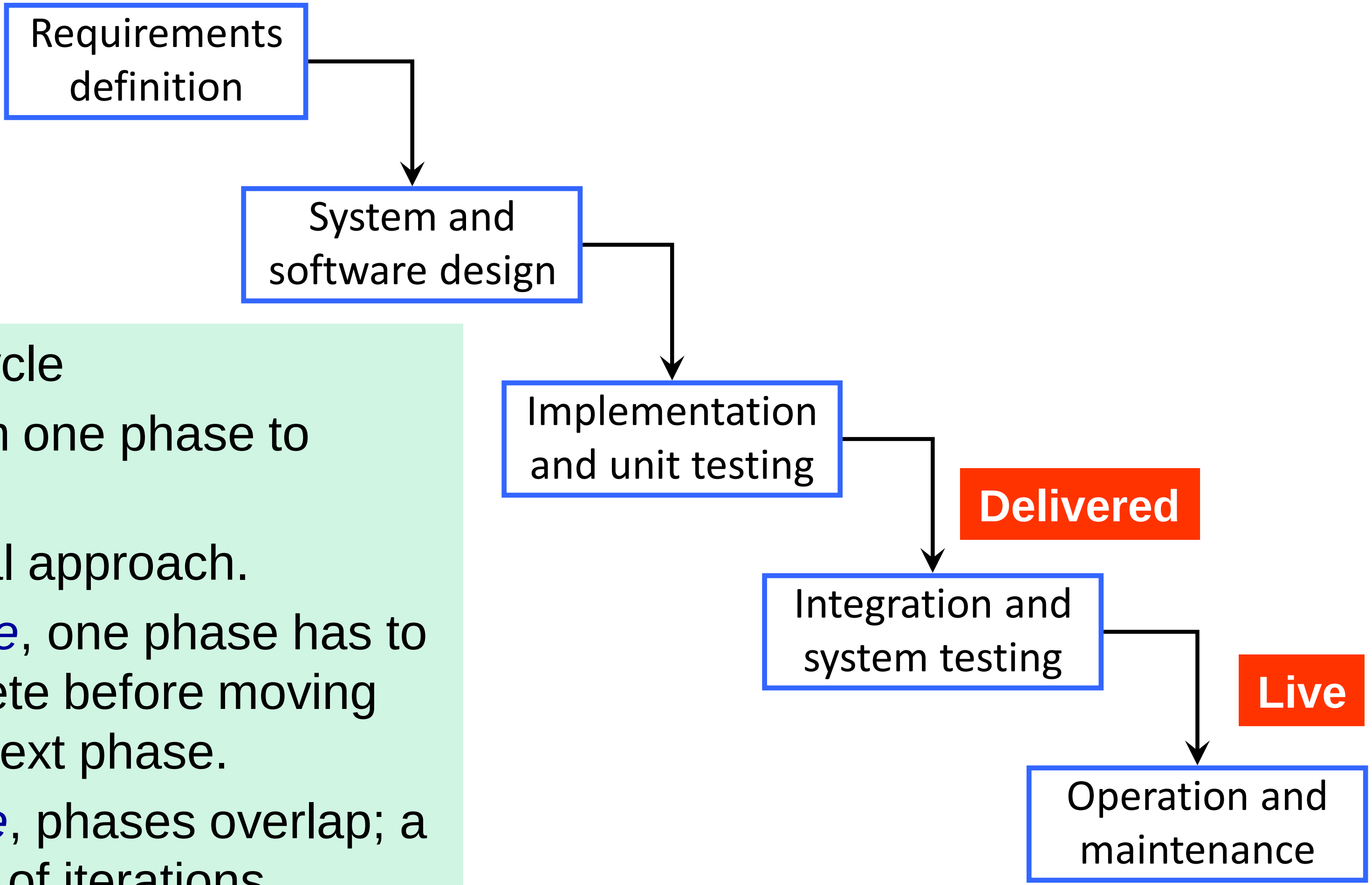
- Depends on the system; the activities can be:
  - organised in sequence
  - organised as interleaved
  - organised concurrently
- Software Process model
  - A simplified representation of a software process – an abstract representation.

# Generic models (traditional)

1. The waterfall model
   – Separate and distinct phases.
2. Evolutionary development
   – Activities are interleaved.
3. RUP (The Rational Unified Process)
   – Four phases

**There are many other software process models**

# 1. Waterfall model and its phases


Requirements definition
System and software design
Implementation and unit testing
Delivered
Integration and system testing
Live
Operation and maintenance

- Classic life cycle
- Cascade from one phase to another:
  - Sequential approach.
  - *In principle*, one phase has to be complete before moving onto the next phase.
  - *In practice*, phases overlap; a sequence of iterations.

# Waterfall model benefits and drawbacks

- Easy to monitor the progress
  - After a small number of iterations, freeze parts of the development and continue with later phases.
- Documentation is well produced at each stage.
- Structured approach.
- Specialised teams can be used at each stage of the lifecycle.

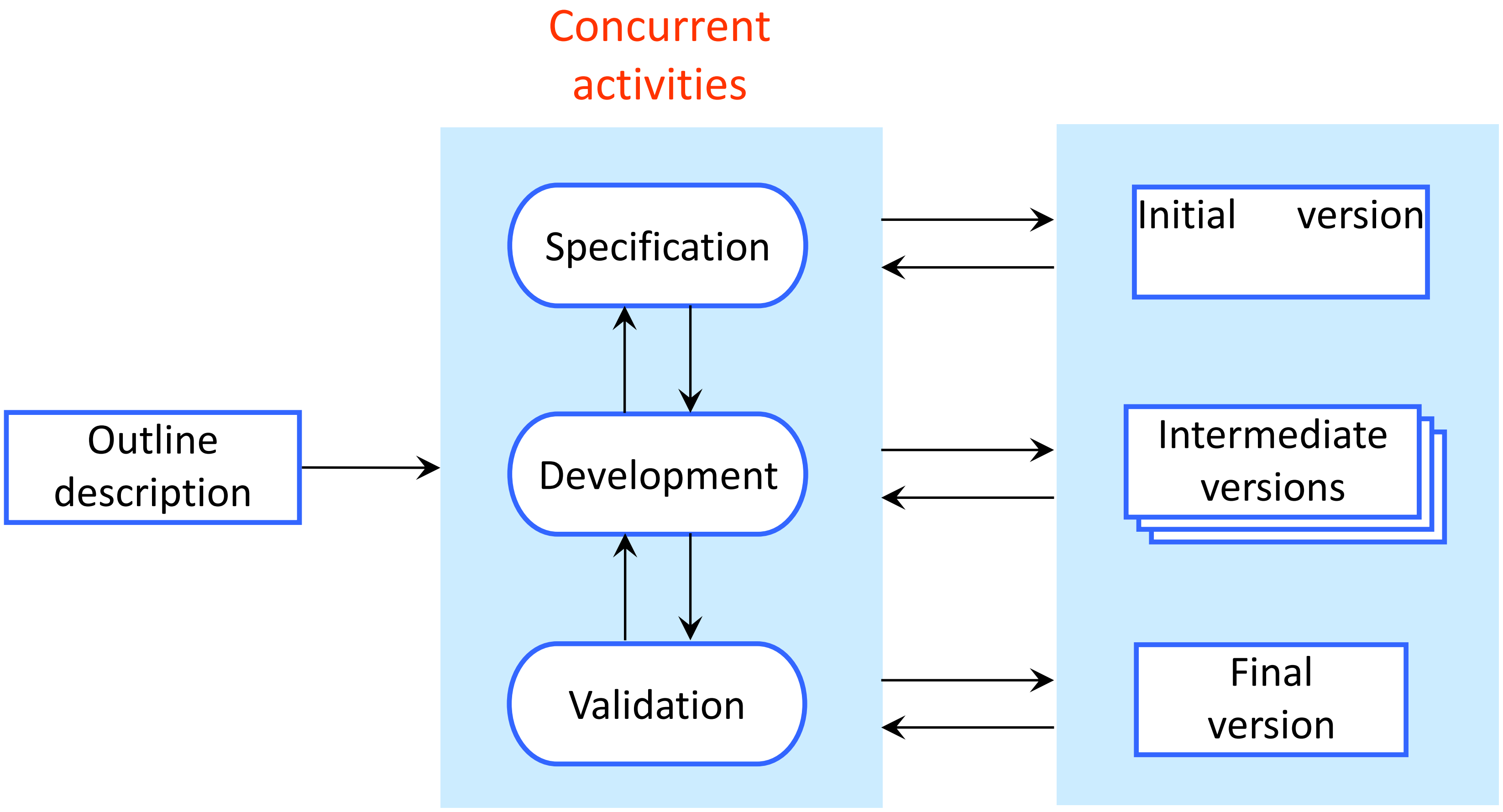
- Inflexible
  - Difficulty of accommodating change after the process is underway.
- Time consuming
  - Real projects rarely follow the sequential flow.
  - A working version of the system will not be available until late in the project time-span.
- Minimises impact of global understanding over the lifecycle of a project.
- Not realistic.

# Waterfall model applicability

- This model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements!
- Adaptation or enhancement of existing system.
- In high risk, safety critical systems e.g. air traffic control, quality is key.

Examples:
aircraft systems, space systems, nuclear power systems, and business critical systems, e.g. power and telecommunications

# 2. Evolutionary development



# Evolutionary development

- Activities are interleaved
- Rapid feedback
- Refining through many versions, evolves over time
  - Completion of a comprehensive product is impossible.
  - Deliver core functions to meet competitive or business pressure.
  - Core requirements are well understood but not the detailed extension.

# Evolutionary development benefits and drawbacks

- Effective
  - Concurrency, several members of the team may be working on different increments or releases
- Can meet the immediate needs
  - Requirements … no longer fixed
  - Refining versions
- Specification can be developed incrementally
  - Users feedback
  - Planned feature, new feature?

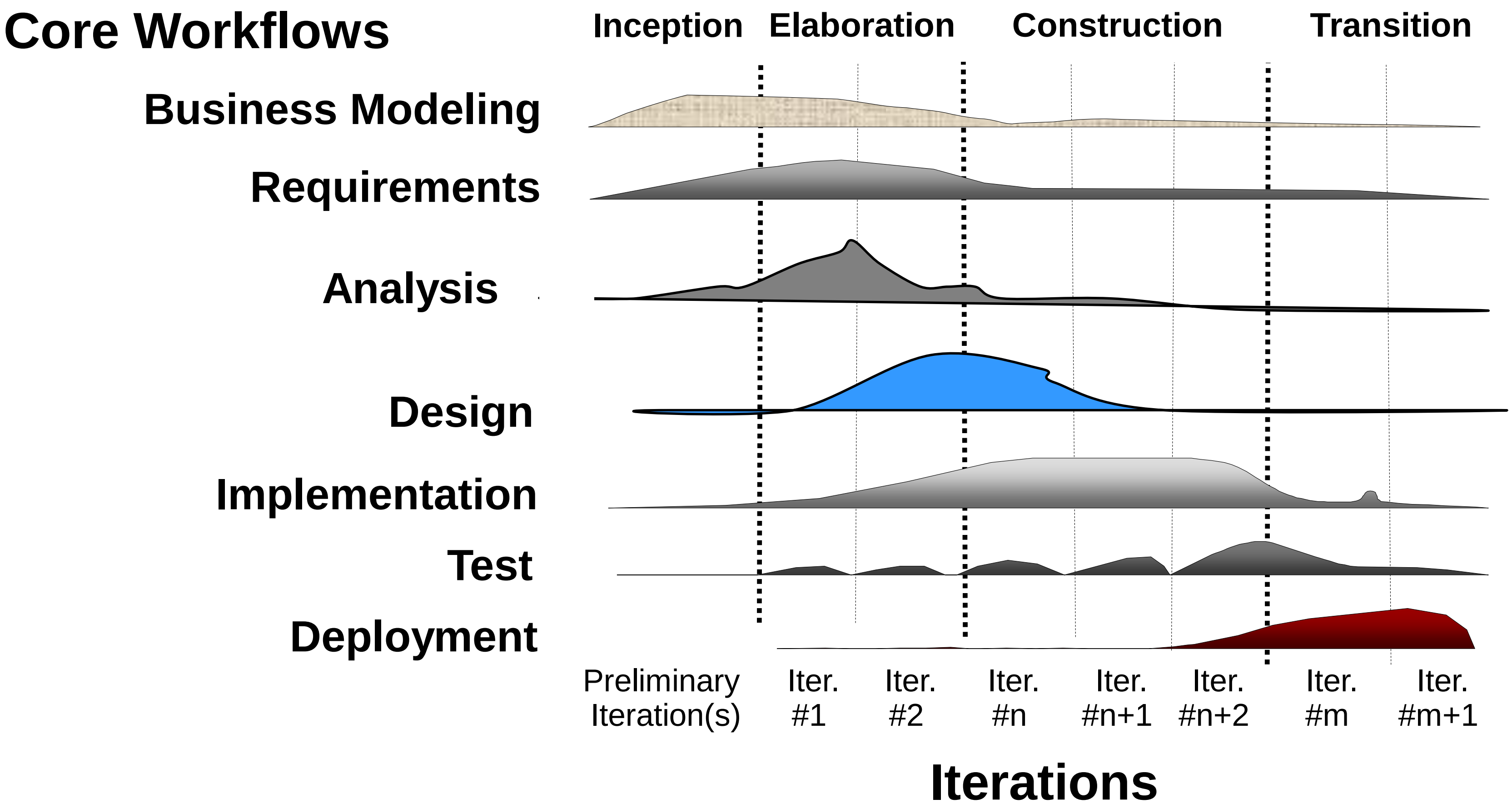
- Lack of process visibility
  - Not cost-effective to produce documents that reflect every version.
  - Lack of deliverable documents to measure progress.
- Systems are often poorly structured
  - Continual change
  - Rush work
- Special skills may be required
  - E.g. in languages for rapid prototyping.
- What is the burden on the end user/client?

# Evolutionary development applicability

- Suitable for:
  - small or medium-size interactive systems
  - parts of large systems, e.g. the user interface
- For short-lifetime systems.
- Project with multiple features and therefore releases.

Examples:
Social networking, communication, phone apps

# 3. The Rational Unified Process



[11] “The Unified Software Development Process” by Ivar Jacobson *et al*, 1999, Addison-Wesley, pg. 11

# The Rational Unified Process (RUP)

- Inception – ends with commitment to go ahead, business case for the project and its feasibility and feature scope identified.
- Elaboration ends with
  - Basic architecture of the system in place.
  - A plan for construction agreed.
  - All significant risks identified.
  - Major risks understood enough not to be too worried.
- Construction (iterative) – ends with beta-release of the system.
- Transition – the process of introducing the system to its users.

# RUP benefits, drawbacks

- Generic process
- Separation of phases and workflows
  - Dynamic
  - With goals

- Overhead
  - Documents
  - Diagrams

# Modern software process

- Scrum (1995)
- Crystal Clear, Extreme Programming (1996)
- Adaptive Software Development, Feature Driven Development (1997)
- Dynamic Systems Development Method (DSDM)

These are now collectively referred to as

Agile Software Development

# Agile Software Development Overview

Agile will be used throughout this module

# Problems of Traditional Development

- Problems:
  - Poor quality
  - This feature can not be tested
  - Usability and User experience is bad
  - Can not meet the schedule
  - Cost too high
  - The team does not communicate and cooperate
  - Too many newcomers and lack of skills
  - Too many documents
  - Is not well maintained

So …

- How to:
  - Do the right things?
  - Do the right things right?
  - Know when you are done?

# Rapid software development

- The needs of Rapid Software Development:
  - – Rapidly changing business environments.
    - No stable, consistent set of system requirements.
    - It is essential that software is developed quickly to take advantage.
  - – Rapid development *and* delivery is Critical.
- Rapid development and delivery is now often the most important requirement for software systems
  - – Businesses may be willing to accept lower quality software if rapid delivery of essential functionality is possible.

# The Agile Process

- The processes of specification, design, implementation and testing are concurrent, referred to as an iteration:
  - – no detailed specification;
  - – design documentation is minimised.
- The system is developed in a series of increments
  - – End users evaluate each increment and make proposals for later increments.
- End users are involved
  - – System user interfaces are usually developed using an interactive development system.

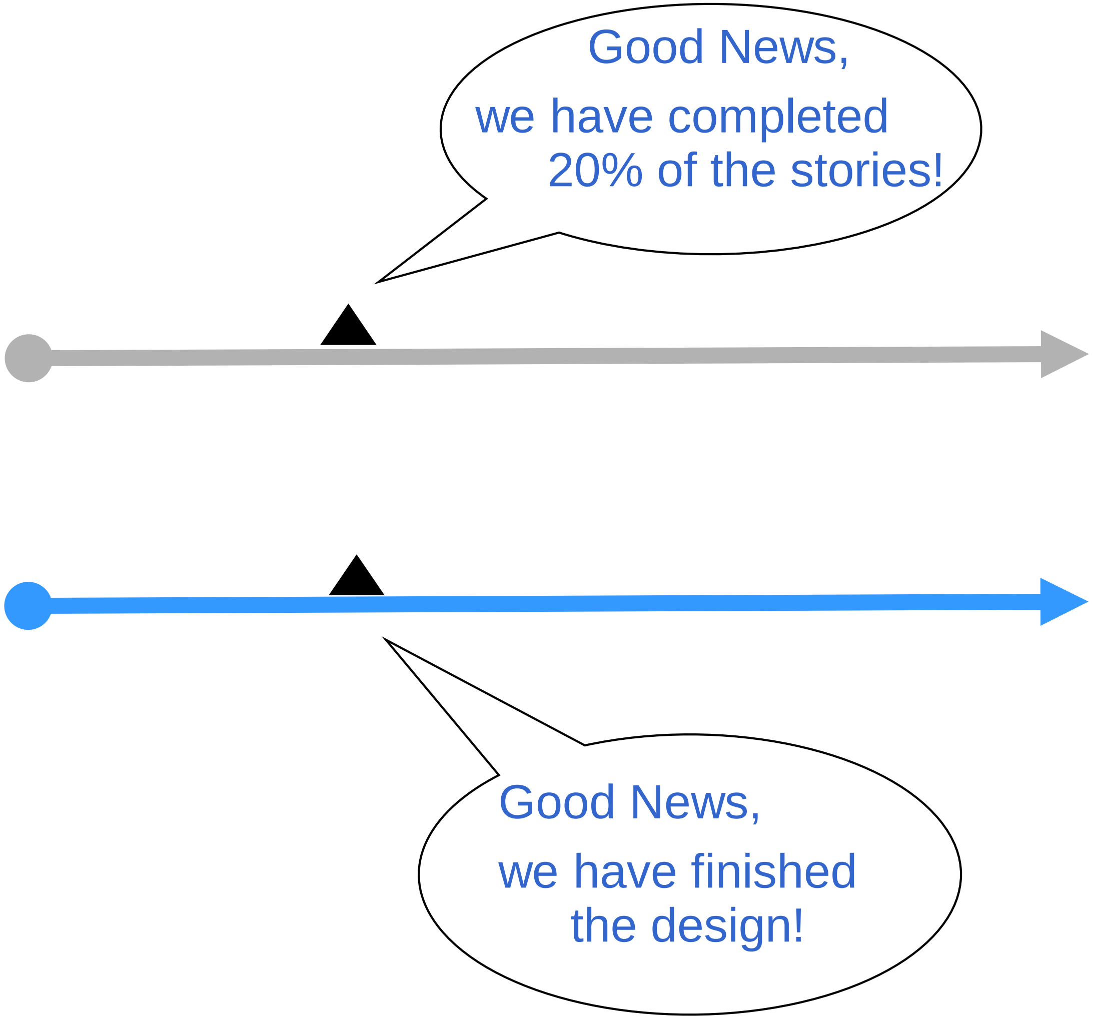
# What is Agile?

- To address the dissatisfaction with the overheads involved.
- Agile is a set of best practices in software development based on *Scrum*, *Extreme Programming* and *Lean*.
- The set includes:
    - Iteration, TDD, continuous integration, refactoring, pair programming, story card/wall, automation test, feedback, stand up, retrospective and showcase.

# Focus of Agile

- The focus of Agile:
  - Focus on the code rather than the analysis/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.

# Which do you trust?



Good News,

we have completed 20% of the stories!

Good News,

we have finished the design!

# The Agile Manifesto

Individuals and interactions over processes and tools

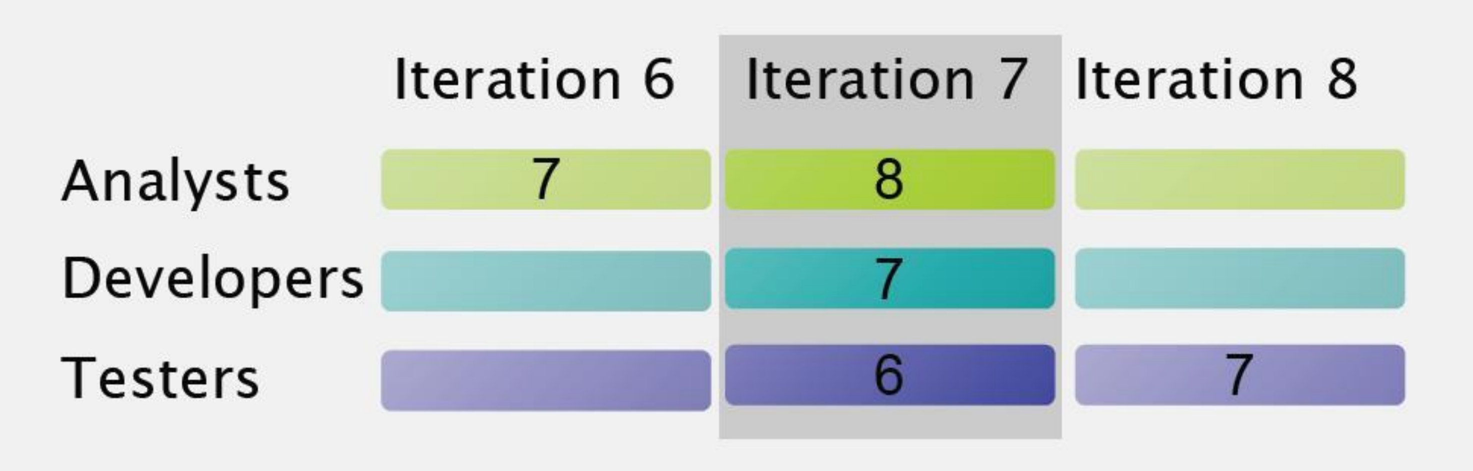
Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

# Agile team

- Agile focuses on developers (programmers) but need other roles…
  - tester, business analyst, coach, project manager…



| | Iteration 6 | Iteration 7 | Iteration 8 |
|---|---|---|---|
| Analysts | 7 | 8 | |
| Developers | | 7 | |
| Testers | | 6 | 7 |

# Agile team

- Small, co-located, multi-disciplinary team, members are usually working around a table
  - Easy communication
- Collective code ownership
- Common vision of system (‘metaphor’)
- Sustainable pace and common coding standard

# Dos and Do NOTs

- Agile needs necessary documentation
  - BUT need to ensure that every document has an audience.
- Agile encourages good practices
  - BUT need to ensure that every practice solves a problem.
- Drop anything without value.
- Don’t over design the system.

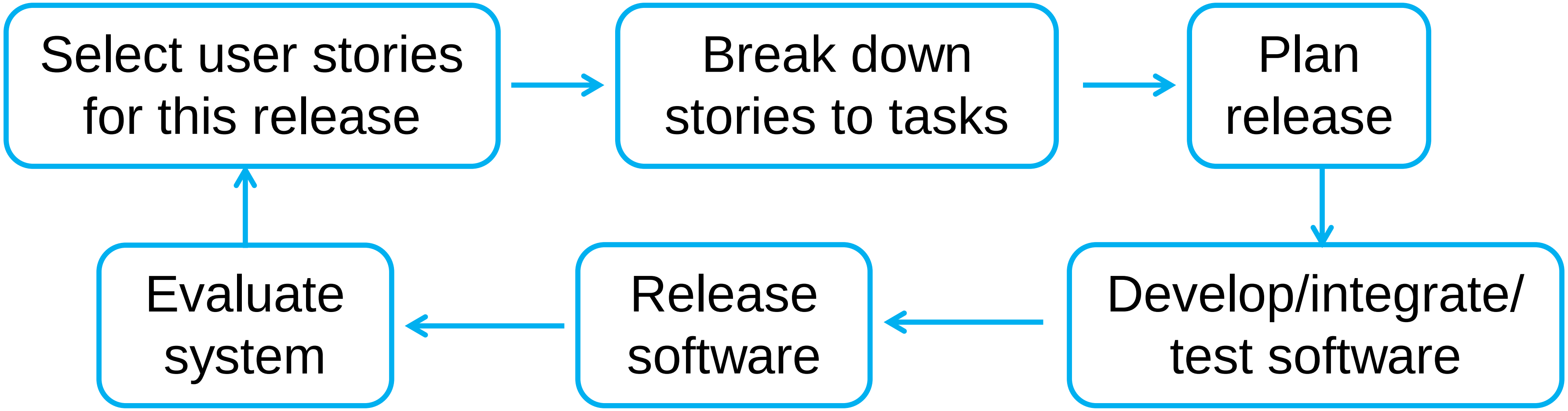
# Principles of Agile

- Customer involvement
  - Closely involved throughout the development
- Incremental delivery
  - Customer specifies each increment
- People, not process
  - Skills, the own way
- Embrace change
  - Expect change
- Maintain simplicity
  - Both the system and the process

# 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 requirements are expressed as user stories.
  - Programmers work in pairs.
  - Develop tests before writing code.
  - All tests must be run for every build and the build is only accepted if tests run successfully.

# The XP Release Cycle



Select user stories for this release
Break down stories to tasks
Plan release
Develop/integrate/ test software
Release software
Evaluate system

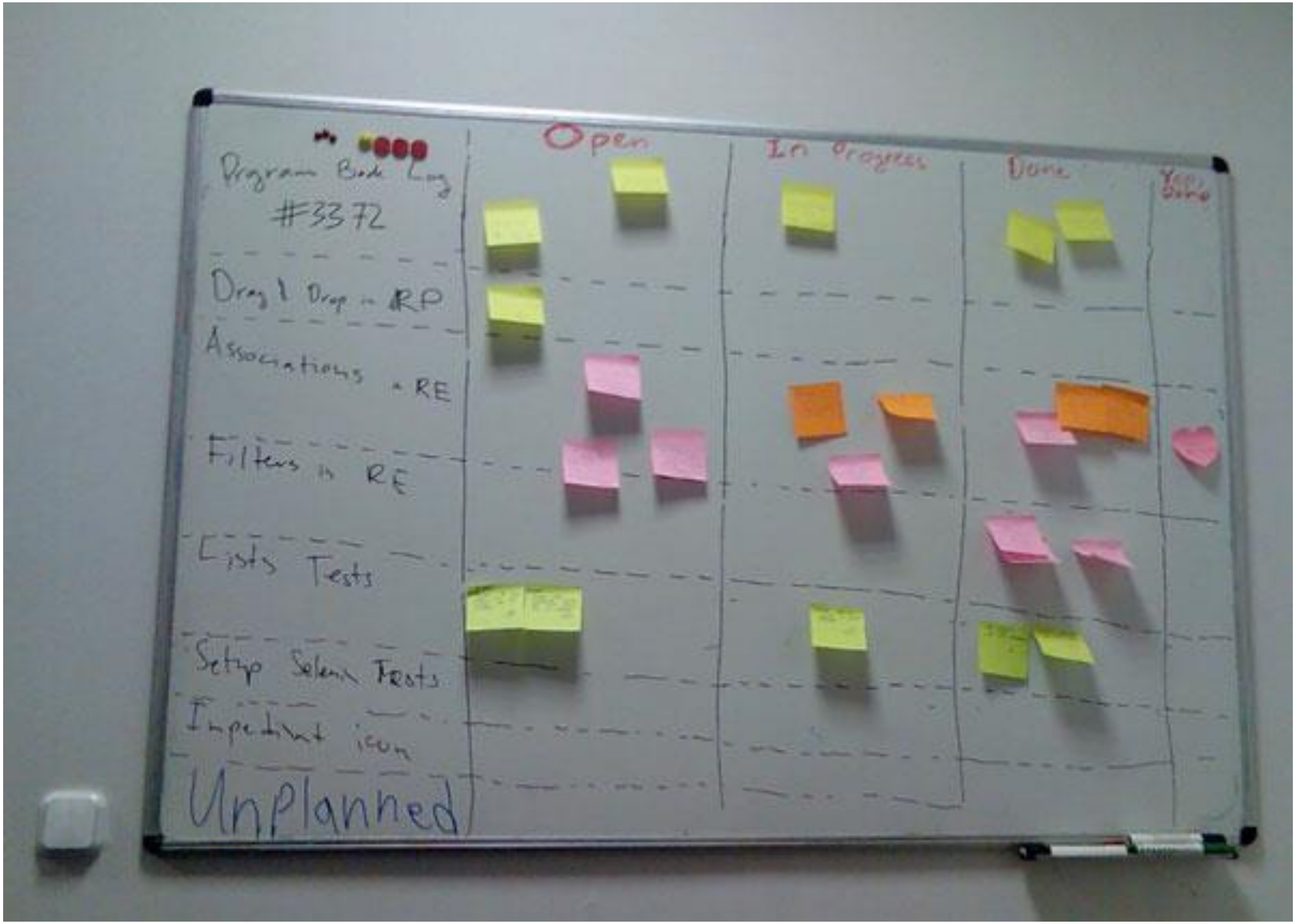
# Planning

- Emphasis on steer, rather than precise prediction.
- Release planning
  - “Customer priorities” and “programmer estimates of feature difficulty” together determine release content.
- Iteration planning
  - Two week delivery cycles
- Goal: visible progress.

# Requirements

- In XP, user requirements are expressed as 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.

# Story Wall


Program Back Log
#3372
Open
In Progress
Done
Done Done
Drag & Drop in RP
Associations in RE
Filters in RE
Lists Tests
Setup Selenium Tests
Impediment icon
Unplanned

# Pair programming

- 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 (or more efficient than) that of two people working independently.

# Design Improvement

- Emphasis on simple design and *refactoring*, i.e. improving existing code.
- Removing duplication:
  - this will inevitably creep in with incremental development.
- Increasing cohesion.
- Reducing coupling.

# Integration and release

- Frequent integration:
  - – multiple builds per day;
  - – everyone involved;
  - – automated tool support.
- Small & Frequent Releases:
  - – Team releases running, tested software delivering business value, as determined by the customer, at every iteration.

# Test Driven Development (TDD)

- Define both an interface and a specification.
- Writing tests before code clarifies the requirements to be implemented.
- Incremental test development from scenarios
  - short cycles of adding tests then making them work.
- Automated test harnesses are used to run all component tests each time that a new release is built
- User involvement in test development and validation
  - both programmer (unit) tests and customer (acceptance) tests.

# Agile Problems (1/2)

- Problems
  - 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.

WE'RE GOING TO TRY SOMETHING CALLED EXTREME PROGRAMMING.
www.dilbert.com scottadams@aol.com
FIRST, PICK A PARTNER. THE TWO OF YOU WILL WORK AT ONE COMPUTER FOR FORTY HOURS A WEEK.
1/9/03 © 2002 United Feature Syndicate, Inc.
THE NEW SYSTEM IS A MINUTE OLD AND I ALREADY HATE EVERYONE.

# Agile Problems (2/2)

- Problems
  - 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.

# Agile Strengths and Weaknesses

- Agile is strong
  - Lots of change
  - Motivated stable teams
  - Supportive management
  - Empowered people

- Agile issues
  - Stable business
  - Stable technology
  - Command and control management
  - Passive people

# Agile method applicability

- Small or medium-sized product.
- Custom system development within an organization,
  - A clear commitment from the customer to become involved in the development process
  - Not a lot of external rules and regulations that affect the software

# Agile requires … / Keys of Agile

- Requires experience
  - Has experienced leadership
  - Hire an experienced team
  - Listen to them
- Working requirement
  - Environment is critical
  - Support is critical
  - Team dynamics is critical

- Value thinking
- Effective and Efficient communication
- Information sharing
- Tools and Automation

# Principles of Agile (*more detail*)

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

# Summary

- Traditional Software Processes Models
  1. Waterfall
  2. Evolutionary development (Incremental)
  3. The Rational Unified Process
- Modern Software Processes Models
  – Agile

# References

- Chapter 3 – “Software Engineering” textbook by Ian Sommerville
- Introduction to Agile by Sondra Ashmore
- http://en.wikipedia.org/wiki/Agile_software_development

- Chapter 2 – “Software Engineering” textbook by Ian Sommerville
- Chapter 12 – “Head First Software Development textbook by Dan Pilone *et al*