COSC2299/2428
Software Engineering: Process and Tools

Semester 2, 2018

Course Introduction and Overview

Who is teaching

Position	Name	Email	Office / Consultation	Consultation Time
Lecturer	**Associate Professor Lawrence Cavedon**	**lawrence.cavedon@rmit.edu.au**	14.08.09 Phone: 9925-2325	Wednesday 11:30am-12:30pm
Head Tutor	Homy Ashrafzadeh	amirhomayoon.ashrafzadeh@rmit.edu.au	14.08.7A	Tuesday 3:30pm–4:30pm
Tutor	Reza Soltanpoor	reza.soltanpoor@rmit.edu.au	14.08 AI lounge for consultation time only	Friday 3pm-4pm From Week 4
Tutor	Mohamad Ali	Mohamad.ali@rmit.edu.au	14.11.16 for consultation time only	Thursday 5:30pm-6:30pm From Week 4

Class Time/Venue

Class	Who	Day	Time	Location
Lecture	**Lawrence**	**Wednesday**	**12:30pm-2:30pm**	**080.04.06**
Tutorial	Homy	Thursday	9:30am-11:30am	057.03.10
Tutorial	Homy	Thursday	12:30pm-2:30pm	056.03.89
Tutorial	Homy	Friday	11:30pm-1:30pm	080.08.47
Tutorial	Reza	Monday	9:30am-11:30am	080.10.13
Tutorial	Reza	Tuesday	1:30pm-3:30pm	080.09.10
Tutorial	Reza	Friday	9:30am-11:30am	13.04.09
Tutorial	Mohamad	Tuesday	4:30pm-6:30pm	014.06.13
Tutorial	Mohamad	Thursday	6:30pm-8:30pm	070.03.04

What this course is about

- Software Engineering is more than the programming phase of an application arena. Phases of the software engineering life cycle include specialised processes and tools to ensure that real-world projects, both large and small, are delivered in a quality manner under financial constraints. Invariably, the software end-product is produced by a team of software engineers, stake-holders and other personnel

- This course aims to develop your knowledge of **the processes and associated tools** required to service the software development life-cycle to:
  - Manage **complexity**
  - Reduce **risk**
  - Maximise **quality** of product

Course learning outcomes

At course end, you should be able to:

- discuss various **software engineering processes and their tools** as required for the development of software systems.
- identify, analyse, compare and contrast different processes and their assistive tools for selected phases of the software engineering life cycle.
- cooperate and contribute to a team environment.
- **apply teamwork techniques** to develop team dynamics and leadership, and resolve management/conflict issues.
- work according to an agreed team protocol; run and document meetings
- plan, identify and apply processes and tools for phases of a software engineering life cycle tailored to a particular application domain.
- consider and apply relevant standards and ethical considerations to the employment of tools and processes supporting phases of the software engineering life cycle.

Learning activities

- **Lectures** will cover a weekly Software Engineering process issue and may describe some of the available tools and associated problems.
- A major **software engineering project** will be performed **in teams**. Students will be presented with guidelines to aid in the direction of weekly progress meeting records as well as documents describing how assessment for the team project peer review will be conducted.
- **Tutorials/labs** will be guided by tutors. A major focus will be a **Scrum meeting** and update on Assignment progress, as well as topic-focused exercises. Students are encouraged to perform their learning in **teams**. Teams will be responsible for installing and learning to use SE tools, under guidance from the tutor/lab assistant.
- Online tools will be used to **record individual, team, and feedback** records of all team learning activities. These will also be used for feedback on progress and contribution.
- **Experienced industry professionals will give guest presentations**, e.g., describing aspects of SE practice in different Melbourne companies

Assessment

- **Major Assignment (team)**: 40% (submission will be in multiple parts)
  - The project will commence with an existing codebase
  - Some marks will be for weekly progress demos
  - Peer assessment and communication/document reviews will be usedto award individual marks to team members
  - Major assignment will be discussed in detail in Weeks 1 and 2. For now, try and organise yourself into teams; there will be support for this in the tute/lab sessions in Weeks 1 and 2.
  - **You MUST be in a team by end of Week 2!**
- **Major assignment: 18% PartA +18% PartB + 4% Final Presentation**
- **Mid-Semester Test: 10%**
- **Final Written Exam: 50%**

Plagiarism is an Offence

- Plagiarism:
  - Submitting an assignment that contains other people's work
  - Helping other students to plagiarise
- All submitted work must be your own
  - The only exception: Other people's work can be included if the assignment has explicit instructions to do so. All copied work (from the internet, from other students, or from staff) must be fully identified
  - All detected cases of plagiarism result in a disciplinary hearing

Learning activities – lectures and tutorials

- Course structure
  - Lectures
    - 2 hours per week
    - Supported by lecture notes, other reading (via Canvas)
  - Practical tool presentations, one hour each in various lectures
    - Typically 3/4 hour – 1 hour
    - Some will involve industry presenters
  - Tutorials/Labs/Workshops (2 hours per week)
    - Reinforce SE practice by supporting Scrum meetings, progress updates for major assignment
    - Introduce practical tools that support the SE process
  - Assessment
    - Promote teamwork on major project
    - Reinforce and assess understanding of core SE process and tools

Lecture schedule

- Week 1: Course overview; Introduction to Major Project;
  Overview of SE methodologies; Scrum 1
- Week 2: Industry lecture: overview of modern SE practice; Scrum 2 + User Stories
- Week 3: Unit Testing and TDD, BDD; Test Management, Logging, Debugging
- Week 4: Advanced Git Workflows and release management
- Week 5: Design Patterns
- Week 6: Builds: Ant, Maven, Builds for Python/PHP
- Week 7: Docker 1
- Week 8: Docker 2; Mid semester test
- Week 9: DevOps
- Week 10: Continuous Integration and Delivery
- Week 11: Performance Testing; Refactoring
- Week 12: Review

When and where?

- See timetable for tute/lab/workshop rooms
  - Choose a tute group to be part of
    - This will be both your learning team and your assignment team
    - All assignment team members must attend the same tutorial/lab session
- Notes:
  - Guest presentations will be in lecture room
  - Teams for assignment
    - Meetings to be arranged by team members

Other communication

Check these often!!!

- Communication
  - Announcements and Discussion Board in Blackboard
    - https://my.rmit.edu.au
  - Consultation
    - Please use Discussion Board to post questions
    - Via Email if not answered in Discussion Board within 2 days
    - Consultation time prior to the lecture

- Comments and feedback
  - Speak with tutor first!
  - Speak with me next
  - Use Student Staff Consultative Committees

Major Assignment and Tute

- Major Assignment:
  - significant Software Engineering project, commencing with existing codebase:
    - user stories + implementation + tests + basic documentation
  - Options using Java or Python or JavaScript or PhP – your choice
  - Part A – 18% -- due Sunday September 7 11:59 PM
  - Part B – 18% -- due start of Week 12
  - Presentations – 4% -- in Week 12

Major Assignment – Part A

- Part A
  - focus is on Process and Tools
    - use of Trello, Slack/HipChat/etc, github, testing and test management framework
  - tute/labs will support Scrum meetings and progress reports
    - 5% for weekly progress updates
  - marking criteria for Week 7 submission (13%):
    - system quality (functionality, usability)
    - code quality (layout, documentation, logging etc)
    - design etc
    - process: use of tools, constant progress, etc
    - BONUS: using Test Driven Development (talk to tutors)
    - individual marks will be adjusted based on contribution:
      - Peer Evaluation; inspection of tool logs

Tute/lab progress marking

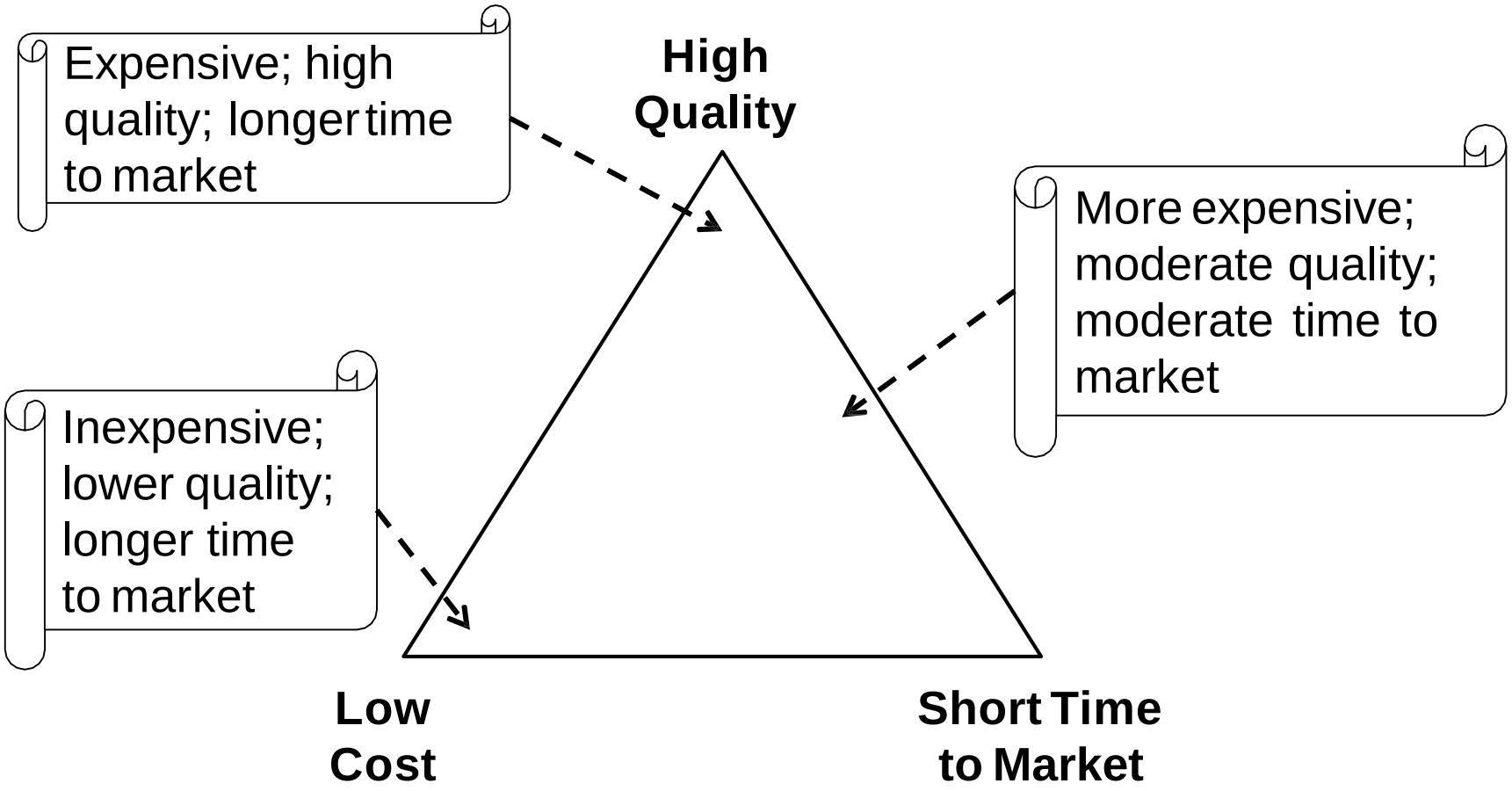
Week	Tasks	Delivery to be examined	Marks
1	Organise teams; set up tools & environment; select project		0
2	Understand code base; develop first user stories	Names of team members; roles of each/	0
3	Start development; define tests; plan next function	Mock-up of login/reg functionality; planned tasks; github set up	1
4	Review task/progress; plan and implement next function	Extended prototype; more user stories; test plan; updated task plan / github	1
5	Review task/progress; plan and implement next function	As week 4; bug tracking; updated github	1
6	Review, etc; construct build for Part A Submission	As previous	1
7	Submit; start Part B	Demo working prototype	1

Processes and Tools

Why SE Process and Tools

- This course aims to develop your knowledge of **the processes and associated tools** required to service the software development life-cycle to:
  - Manage **complexity**
  - Reduce **risk**
  - Maximise **quality** of product

Quality Triangle



Why SE Process and Tools

- What are some of the characteristics of modern software construction:
- How does Process help:
- How do Tools help:
- What else do we need:

Why SE Process and Tools

- As well as reducing Risk, we reduce Uncertainty
  - Time required to complete a project
  - Availability and cost of key resources
  - Timing of solutions to technological problems
  - Macroeconomic variables
  - The whims of clients
  - Actions taken by competitors

Why do projects fail?

- What is failure
  - The project is not delivered on time
  - The cost is over budget
  - The system does not work as required.
- “The majority of accidents were not caused by some subtle failure of the control system, but by defects that were preventable if a systematic approach had been adopted throughout its design lifecycle.
- Failure to pay attention to detail, particularly during the specification phase of a project, and to properly manage technical issues were the root causes of these accidents.”

“Out of Control: Why control systems go wrong and how to prevent failure”, Sudbury, 1995 HSE Books - Health and Safety Executive, Great Britain. ISBN 0-7176-2192-8

Why projects fail

Findings (Chaos Report)	% of responses
Incomplete requirements	13.0%
Lack of user involvement	12.4%
Lack of resources	10.5%
Unrealistic expectations	9.9%
Lack of executive support	9.3%
Changing requirements and specs	8.7%
Lack of planning	8.1%
No longer needed	7.5%
Technology illiteracy	4.3%
Other	16.3%

Why projects fail

• Bad Communications between relevant parties	57%
• Lack of planning, scheduling	39%
• No quality control	35%
• Milestones not being met (outcome)	34%
• Inadequate co-ordination of resources	29%
• Costs getting out of hand (outcome)	26%
• Mismanagement of progress	20%
• Overall poor management	17%
• Supplier skills overstretched	13%
• Insufficient measurable outcomes	11%

The Bull Survey, UK Finance Sector

Product vs Process

- Small programs (<10,000 lines) can be implemented easily by one person
- Larger systems need a more systematic approach. Why?
- We distinguish between (Consider building a new car)
  - the product (what we are making)
  - the process (how we make it)
- What is a process?
  - It is a method/plan you follow to develop a system

Software Development Life Cycle

- SDLC has several activities.
- Different processes may change the order, the number of iterations, or the use of these phases.
- Other phases include deployment and maintenance

Requirements Gathering/Elicitation	Finding out what the user wants
System Analysis and Requirements Eng	Understanding/documenting requirements
Design	Planning a possible solution
Implementation	Building a solution
Testing	Ensuring it meets the requirements

Other parts of the life cycle

Feasibility analysis – can save millions of dollars

- Is it technically possible to build the system?
- Does the company have the ability to build it
  - People
  - Skills
  - Finance
  - Time

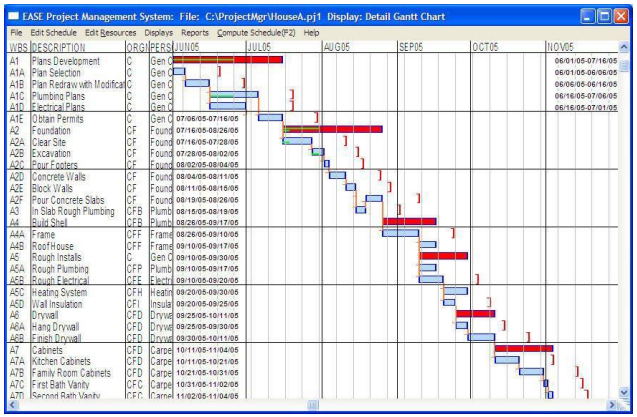
Other parts of the life cycle

Project scoping

- Deals with putting limits on how much you want to create/clearly specifying what you want to achieve.
  - You may decide to leave network play out of a game.
  - An accounting package may be limited to supporting only one database back-end (e.g. Oracle)
  - A music player may only play songs encoded in MP3 format
  - The **product** may have to be developed with a budget no greater than $400,000
  - The product may have to be created within 12 months

Other parts of the life cycle

- Project management
  - -Estimating development time
  - -Ensuring people are used effectively
  - -Ensuring good communication and a healthy work environment
  - -Managing budgets
- Training and handover
- Maintenance



Project Management Triangle

- **Time** – The actual time required to produce a deliverable - directly related to the amount of requirements and resources allocated.

- **Cost** – The amount of money required to complete the project. Why does the time allocated affect the cost?

- **Scope** – The functional elements that make up the project deliverables. The common success measure is its inherent quality upon delivery.

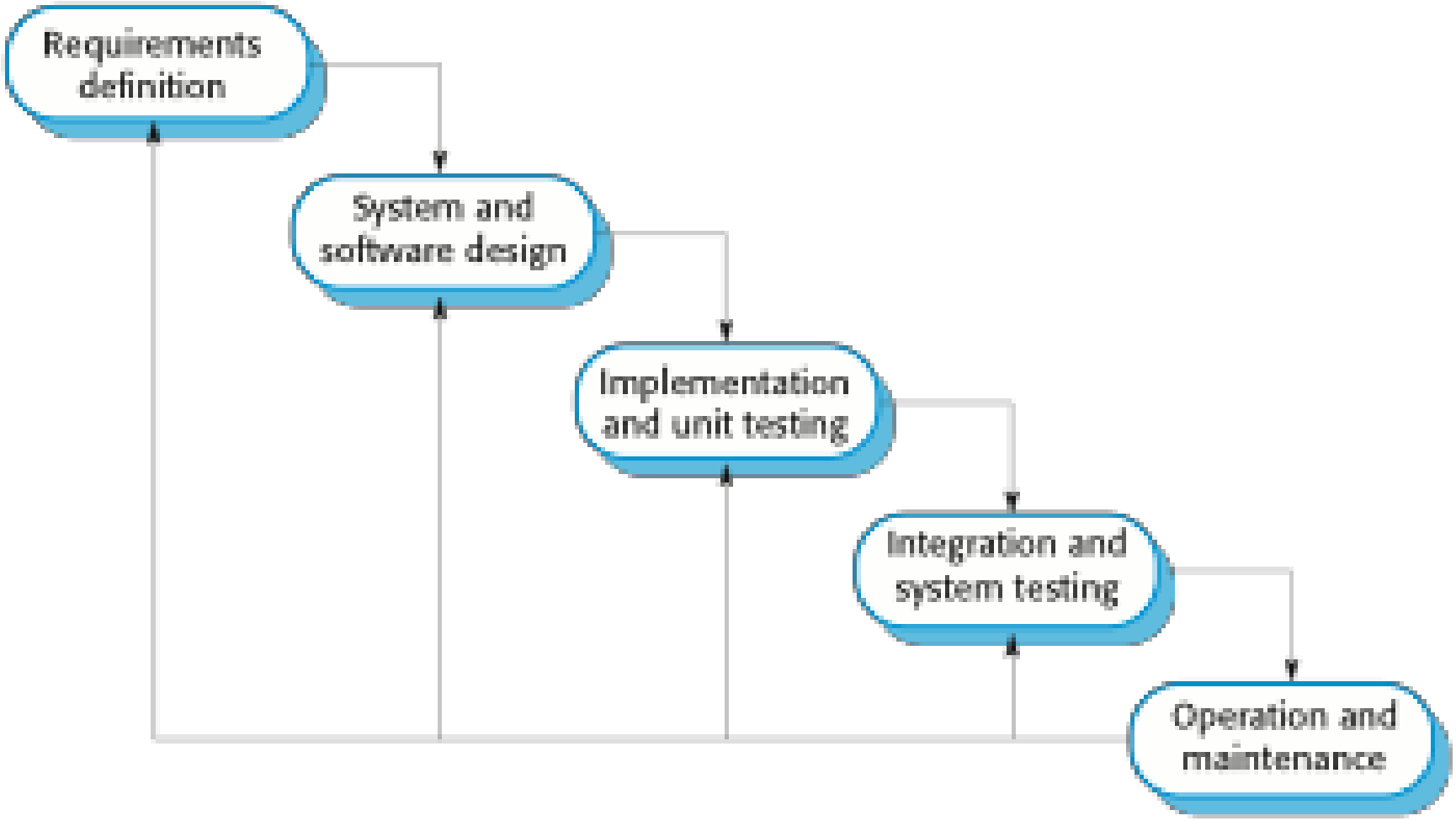
Software Process Models

Software process models

A software process model is an abstract representation of a process. It presents a description of a process from a particular perspective.

- The waterfall model is a plan-driven model with separate and distinct phases of specification and development.
- Incremental development - specification, development and validation are interleaved. May be plan-driven or agile.
- Reuse-oriented software engineering - The system is assembled from existing components. May be plan-driven or agile.
- In practice, most large systems are developed using a process that incorporates elements from all of these models.

The waterfall model



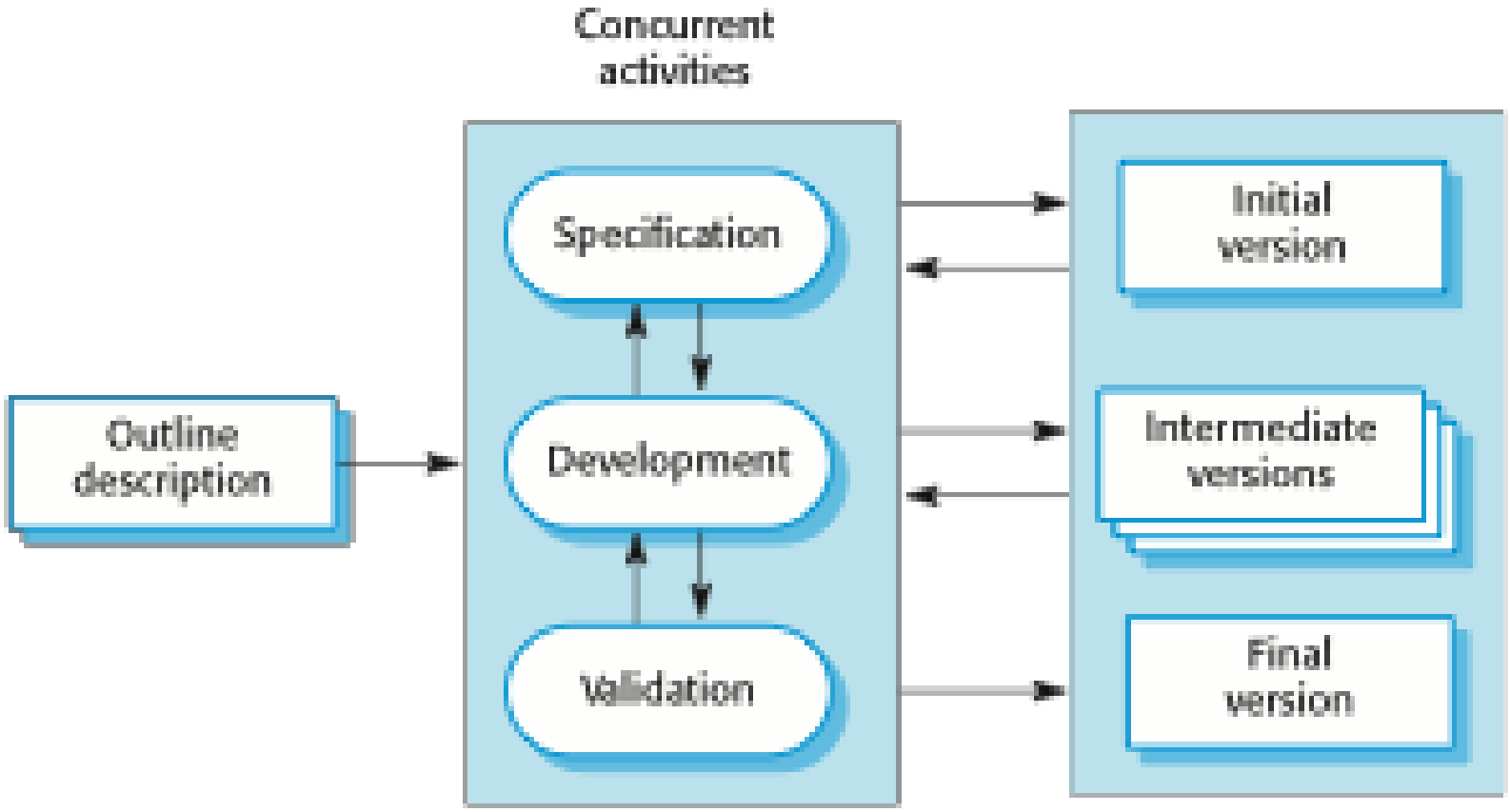
Waterfall model phases

- There are separate identified phases in the waterfall model:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance

Waterfall model problems

- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, each phase has to be complete before moving onto the next phase.
- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
- model is only appropriate when the requirements are well-understood and changes are limited during the design process. Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites. The plan-driven nature of the waterfall model helps coordinate the work.

Incremental development



Incremental development benefits

- The cost of accommodating changing customer requirements is reduced. The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- It is easier to get customer feedback on the development work that has been done. Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Incremental development problems

- The process is not visible.
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- System structure tends to degrade as new increments are added.
  - Unless time and money are spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

Rapid software development --- Agile methods

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast –changing requirement and it is difficult to produce a set of stable software requirements
  - Software must evolve quickly to reflect changing business needs.
  - 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 which:
  - 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.
  - 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 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.*

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.

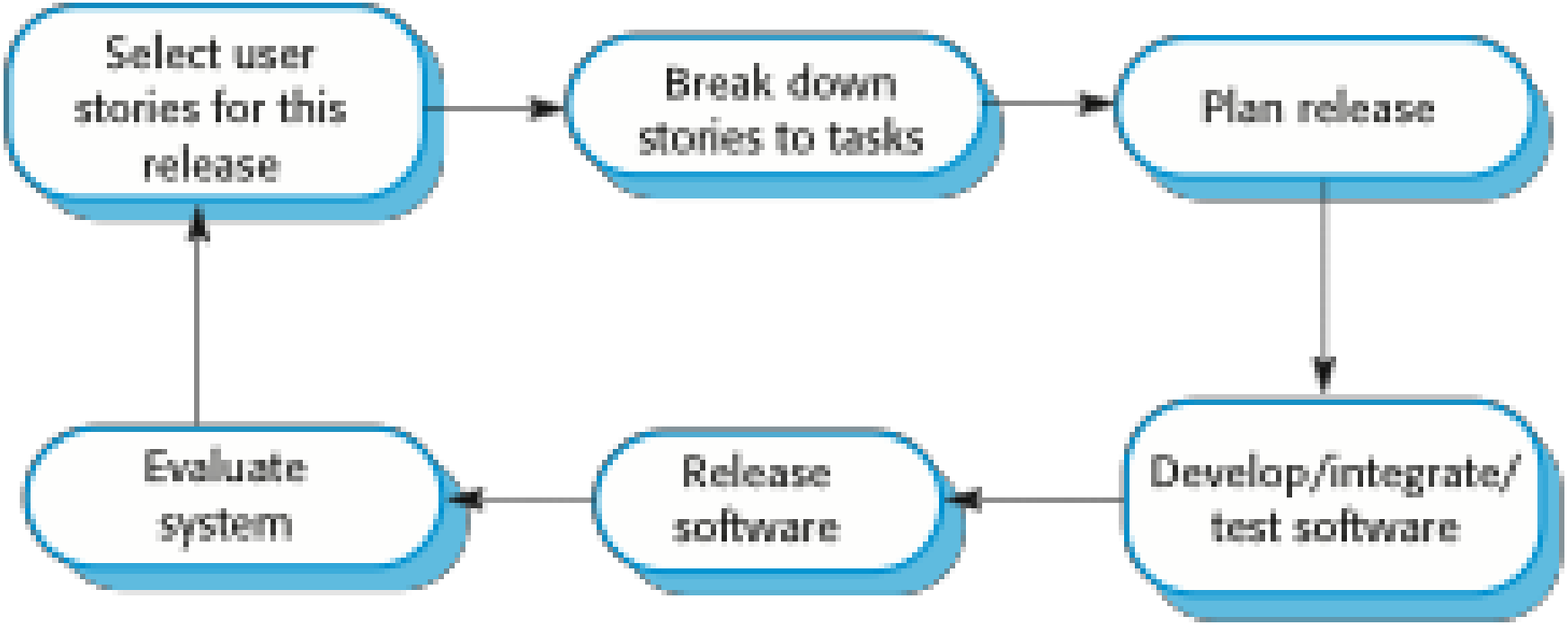
Agile method applicability

- 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.

Extreme programming - XP

- Early 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.

The extreme programming release cycle



Extreme programming practices

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'. (Figs 3.5, 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

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.

Process activities

- Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
- The four basic process activities of
  - Specification,
  - Development,
  - Validation,
  - Evolution

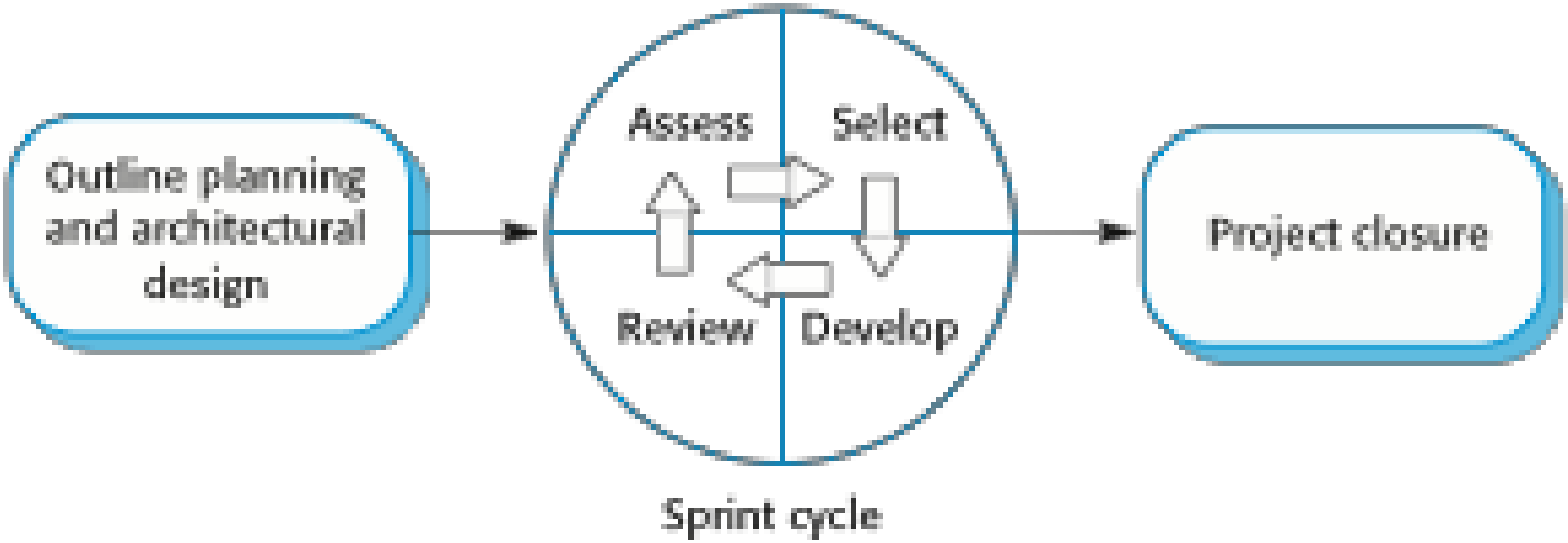
  are organized differently in different development processes. In the waterfall model, they are organized in sequence, whereas in incremental development they are inter-leaved.

Scrum

The Scrum approach is a general agile method but its focus is on managing iterative development rather than specific agile practices.

1. The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture.
2. This is followed by a series of sprint cycles, where each cycle develops an increment of the system.
3. 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.

The Scrum process



The Sprint cycle

- Sprints are fixed length, normally 2–4 weeks. They correspond to the development of a release of the system in XP.
- 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 to be developed during the sprint.

The Sprint cycle

- 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.

Scrum Team

- 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 Product Owner, typically a key stakeholder (e.g., client), is responsible for vision of what is to be built, and conveys that vision to the scrum team. Product Owner is often responsible for managing the Product Backlog

- The whole team attends short daily meetings (“stand ups”) 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 has 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.

Teamwork

- **Team formation**:
  - It is recommended you appoint a nominal lead / Scrum master
  - You may want to consider separate roles/responsibilities, but all team member must do some significant coding
  - Individual marks will be adjusted based on:
    - evidence / quality of contribution; participation; peer review
- **EXPECTATIONS on team members**:
  - You MUST be responsive (via email, phone, etc)
  - You MUST attend scheduled meetings or notify team
  - You MUST contribute to discussion of material, work during tutorials, and producing submission
  - Teams may assign a leader to distribute work but prefer this to be negotiated within team:
    - Requires all team members to actively contribute to negotiation

What now?

- **Major Assignment Part A is released and due in 7 weeks**:
  - organise yourself into teams of 3 or 4
    - team members must be in same tute/lab class
    - tute/lab classes will help with team formation
    - decide which project alternative you will work on
  - as a team, **start exploring the tools** you want to use
    - tools for Tracking (Trello) and Communication (Slack, HipChat, etc) must be in place by Week 3 tute/lab

- First **Progress Mark during Week 3 tute/lab** class:
  - see assignment spec for expectations