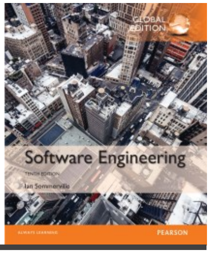
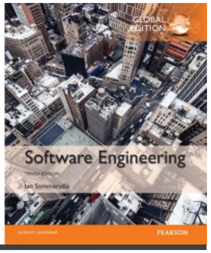


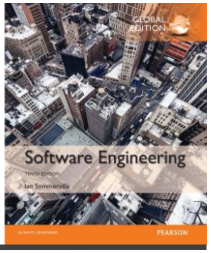
Chapter 3 – Agile Software Development

Topics covered



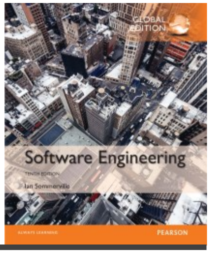
- Agile methods
- Agile development techniques
- Agile project management
- Scaling agile methods

Rapid software development



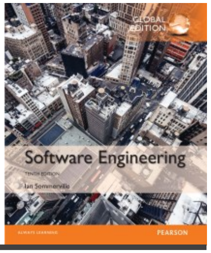
- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs.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

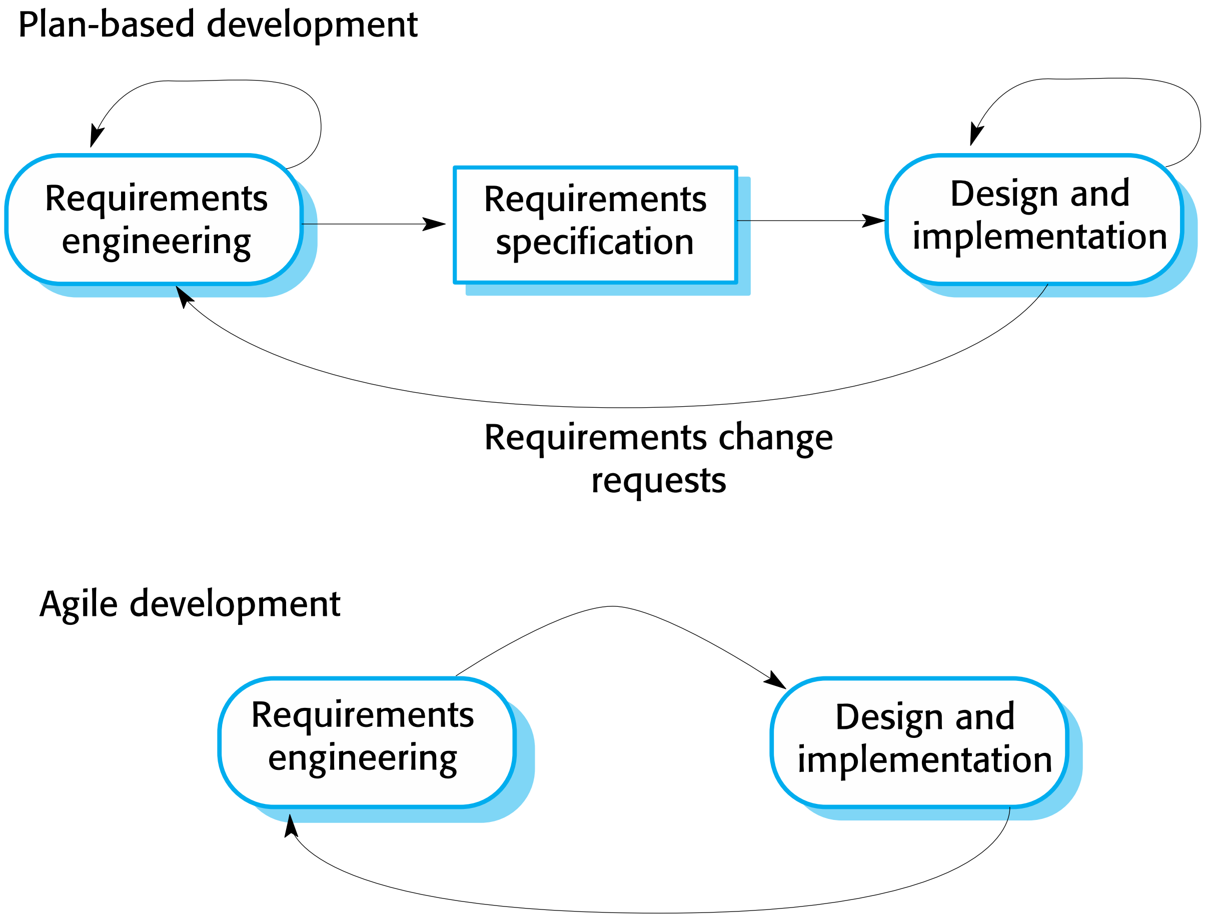
Agile development



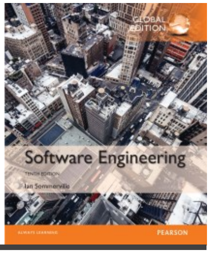
- ✧ Program specification, design and implementation are inter-leaved
- ✧ The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- ✧ Frequent delivery of new versions for evaluation
- ✧ Extensive tool support (e.g. automated testing tools) used to support development.
- ✧ Minimal documentation – focus on working code

Plan-driven and agile development




Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

Plan-driven and agile development

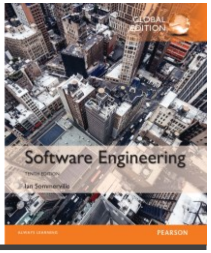


✧ Plan-driven development

- A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
- Not necessarily waterfall model – plan-driven, incremental development is possible
- Iteration occurs within activities.

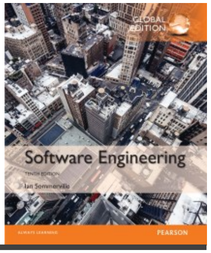
✧ Agile development

- Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.
- Iteration occurs across activities.



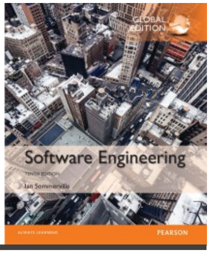
Agile methods

Agile methods



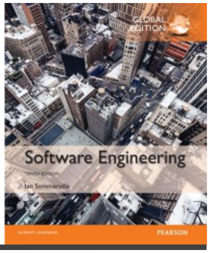
- ✧ Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software 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.

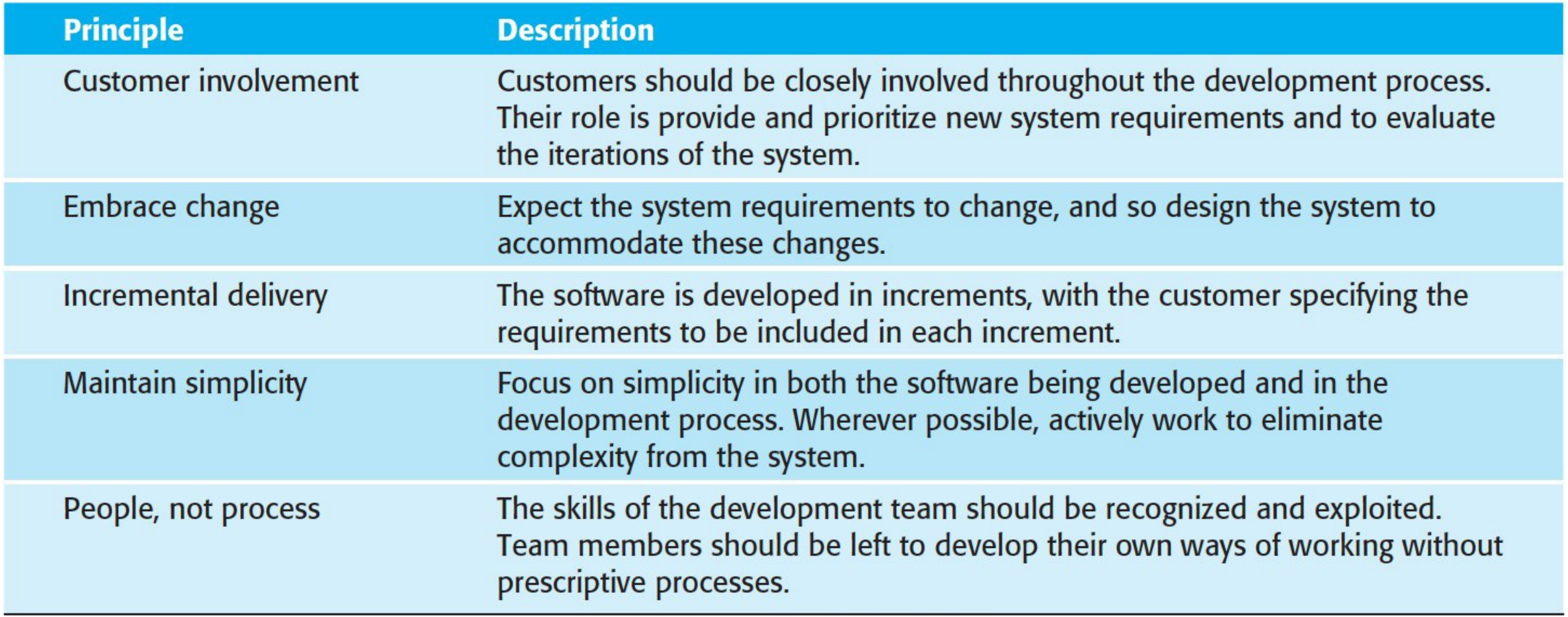
Agile manifesto



- *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.*

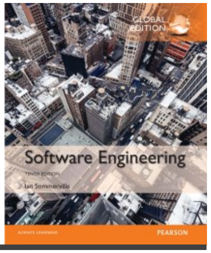
The principles of agile methods



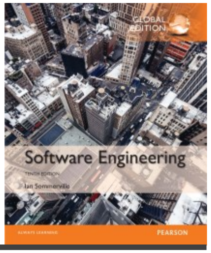


Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Embrace change	Expect the system requirements to change, and so design the system to accommodate these changes.
Incremental delivery	The software is developed in increments, with the customer specifying the requirements to be included in each increment.
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.
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.

Agile method applicability

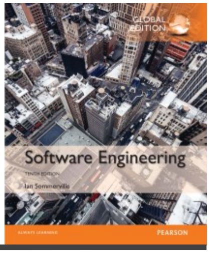


- Product development where a software company is developing a small or medium-sized product for sale.
  - Virtually all software products and apps are now developed using an agile approach
- Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are few external rules and regulations that affect the software.



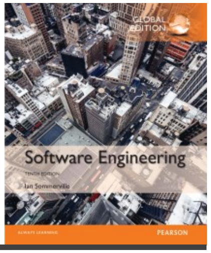
Agile development techniques

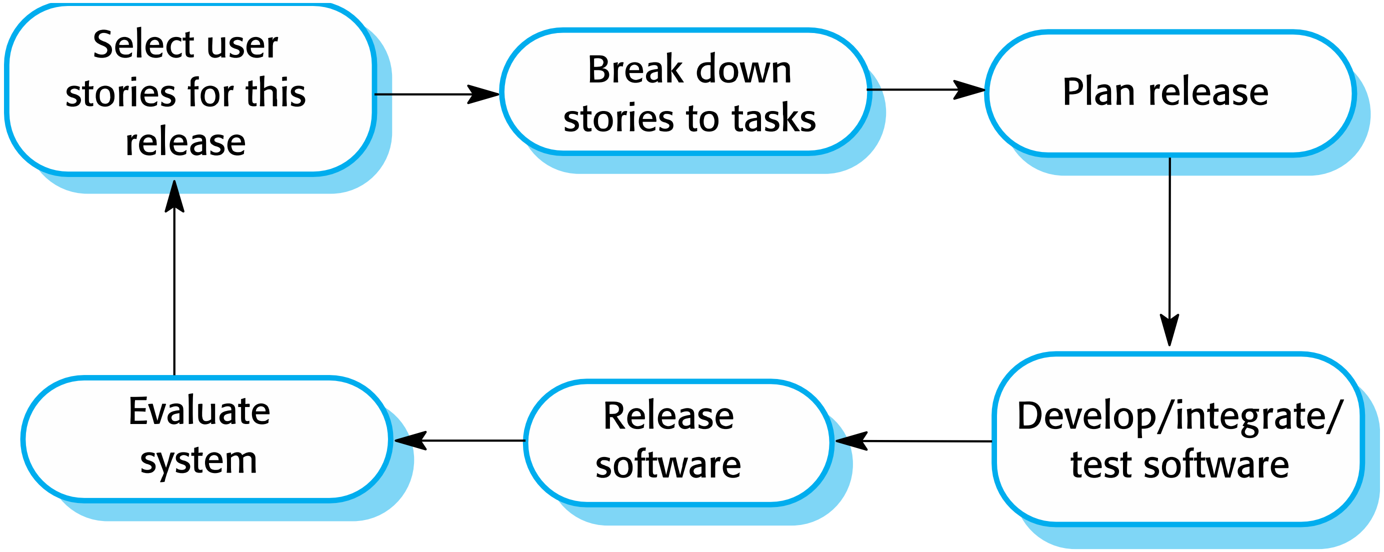
Extreme programming



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





Select user stories for this release

Break down stories to tasks

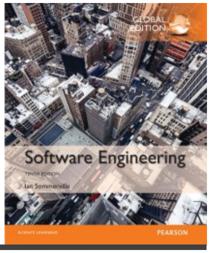
Plan release

Develop/integrate/ test software

Release software

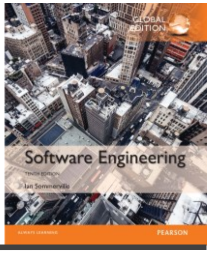
Evaluate system

Extreme programming practices (a)



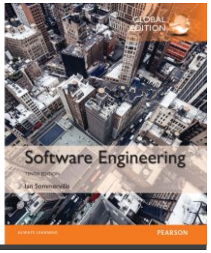
Principle or practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'. See Figures 3.5 and 3.6.
Small releases	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)



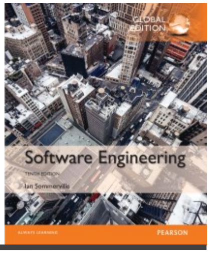
Pair programming	Developers work in pairs, checking each other’s work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. 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.

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.

Influential XP practices



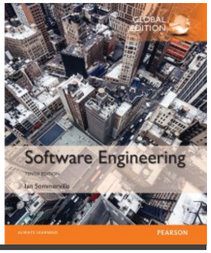
✧ Extreme programming has a technical focus and is not easy to integrate with management practice in most organizations.

✧ Consequently, while agile development uses practices from XP, the method as originally defined is not widely used.

✧ Key practices

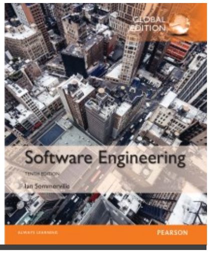
- User stories for specification
- Refactoring
- Test-first development
- Continuous integration
- Pair programming

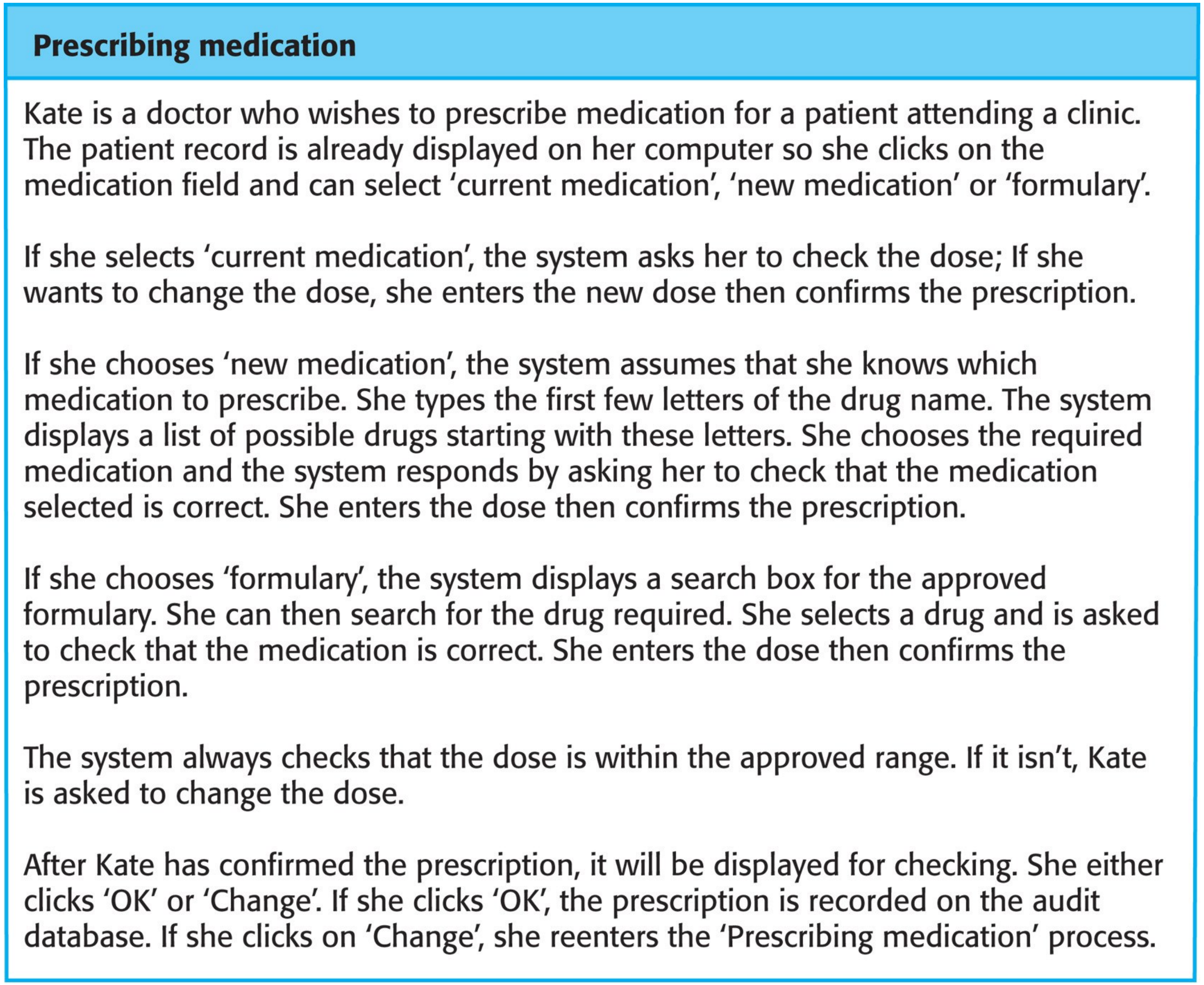
User stories for requirements



- ✧ 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.

A ‘prescribing medication’ story





Prescribing medication

Kate is a doctor who wishes to prescribe medication for a patient attending a clinic. The patient record is already displayed on her computer so she clicks on the medication field and can select ‘current medication’, ‘new medication’ or ‘formulary’.

If she selects ‘current medication’, the system asks her to check the dose; If she wants to change the dose, she enters the new dose then confirms the prescription.

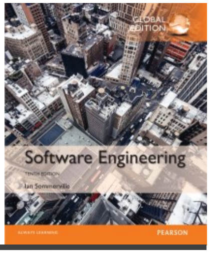
If she chooses ‘new medication’, the system assumes that she knows which medication to prescribe. She types the first few letters of the drug name. The system displays a list of possible drugs starting with these letters. She chooses the required medication and the system responds by asking her to check that the medication selected is correct. She enters the dose then confirms the prescription.

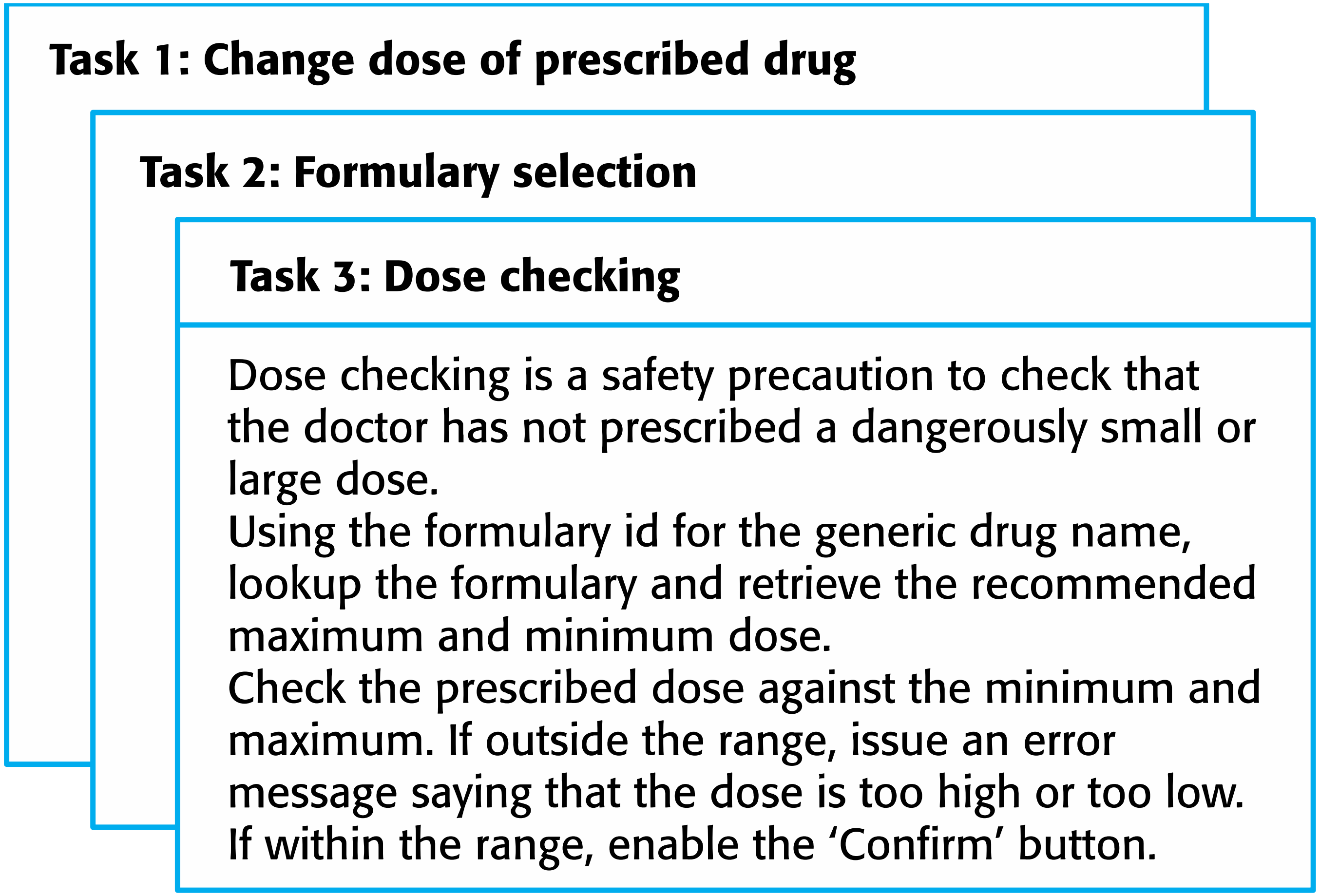
If she chooses ‘formulary’, the system displays a search box for the approved formulary. She can then search for the drug required. She selects a drug and is asked to check that the medication is correct. She enters the dose then confirms the prescription.

The system always checks that the dose is within the approved range. If it isn’t, Kate is asked to change the dose.

After Kate has confirmed the prescription, it will be displayed for checking. She either clicks ‘OK’ or ‘Change’. If she clicks ‘OK’, the prescription is recorded on the audit database. If she clicks on ‘Change’, she reenters the ‘Prescribing medication’ process.

Examples of task cards for prescribing medication





Task 1: Change dose of prescribed drug

Task 2: Formulary selection

Task 3: Dose checking

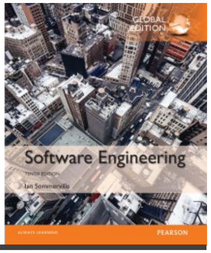
Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.

Using the formulary id for the generic drug name, lookup the formulary and retrieve the recommended maximum and minimum dose.

Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low.

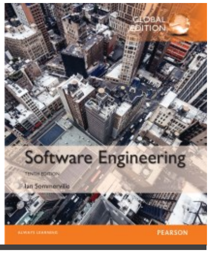
If within the range, enable the 'Confirm' button.

Refactoring

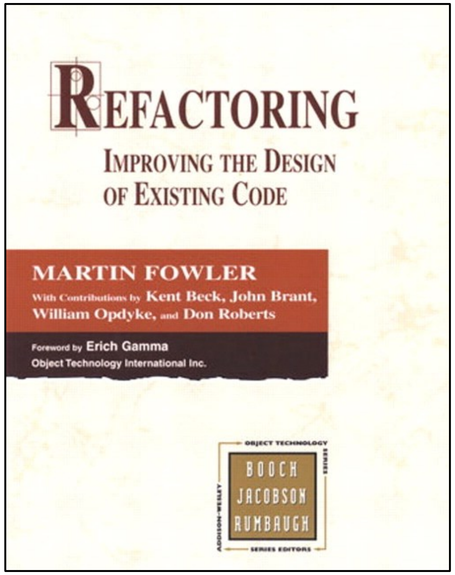


- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort *anticipating changes* as this reduces costs later in the life cycle.
- XP, however, maintains that this is not worthwhile as changes **cannot** be *reliably* anticipated.
- Rather, it proposes constant code improvement (**refactoring**) to make changes easier when they have to be implemented.

Refactoring



- is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior
- Its heart is a series of small *behavior preserving transformations*. Each transformation (called a "refactoring") does little, but a sequence of transformations can produce a significant restructuring.
- The system is kept fully working after each small refactoring, reducing the chances that a system can get seriously broken during the restructuring.



REFACTORING

IMPROVING THE DESIGN OF EXISTING CODE

MARTIN FOWLER

With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts

Foreword by Erich Gamma
Object Technology International Inc.

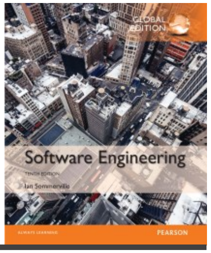
OBJECT TECHNOLOGY SERIES

BOOCH
JACOBSON
RUMBAUGH

SERIES EDITORS

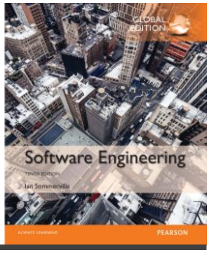
ADDISON-WESLEY

Refactoring



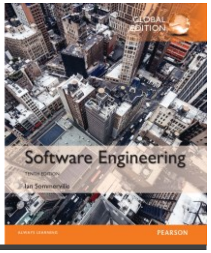
- ✧ Programming team look for possible software improvements (i.e., **code smells**) and make these improvements even where there is no immediate need for them.
- ✧ This improves the understandability of the software and so **reduces the need for documentation**.
- ✧ Changes are easier to make because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

Examples of refactoring



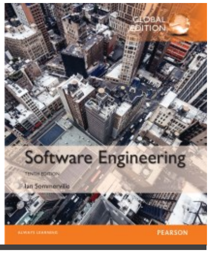
- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.
- Extract and move, …
- Eclipse IDE has supported several types of refactoring to improve the structures of Java elements (class or method)

Test-first development (TF, TFD)

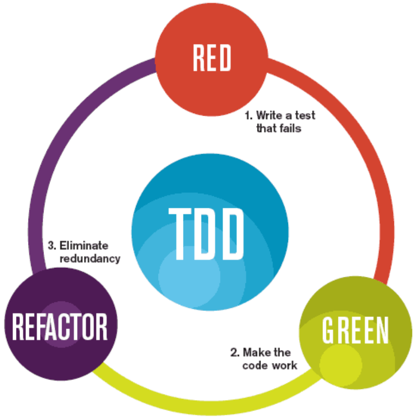


- ✧ Testing is central to XP and XP has developed an approach where the program is tested after every change has been made.
  - Write a failing test first and make it green by writing exactly enough production code to do so
- ✧ XP testing features:
  - Test-first development.
  - Incremental test development from **scenarios**.
  - User involvement in test development and validation.
  - Automated test harnesses are used to run all component tests each time that a new release is built.

Test-driven development (TDD)

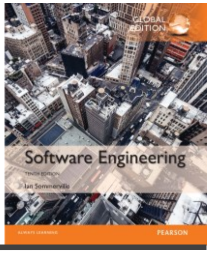


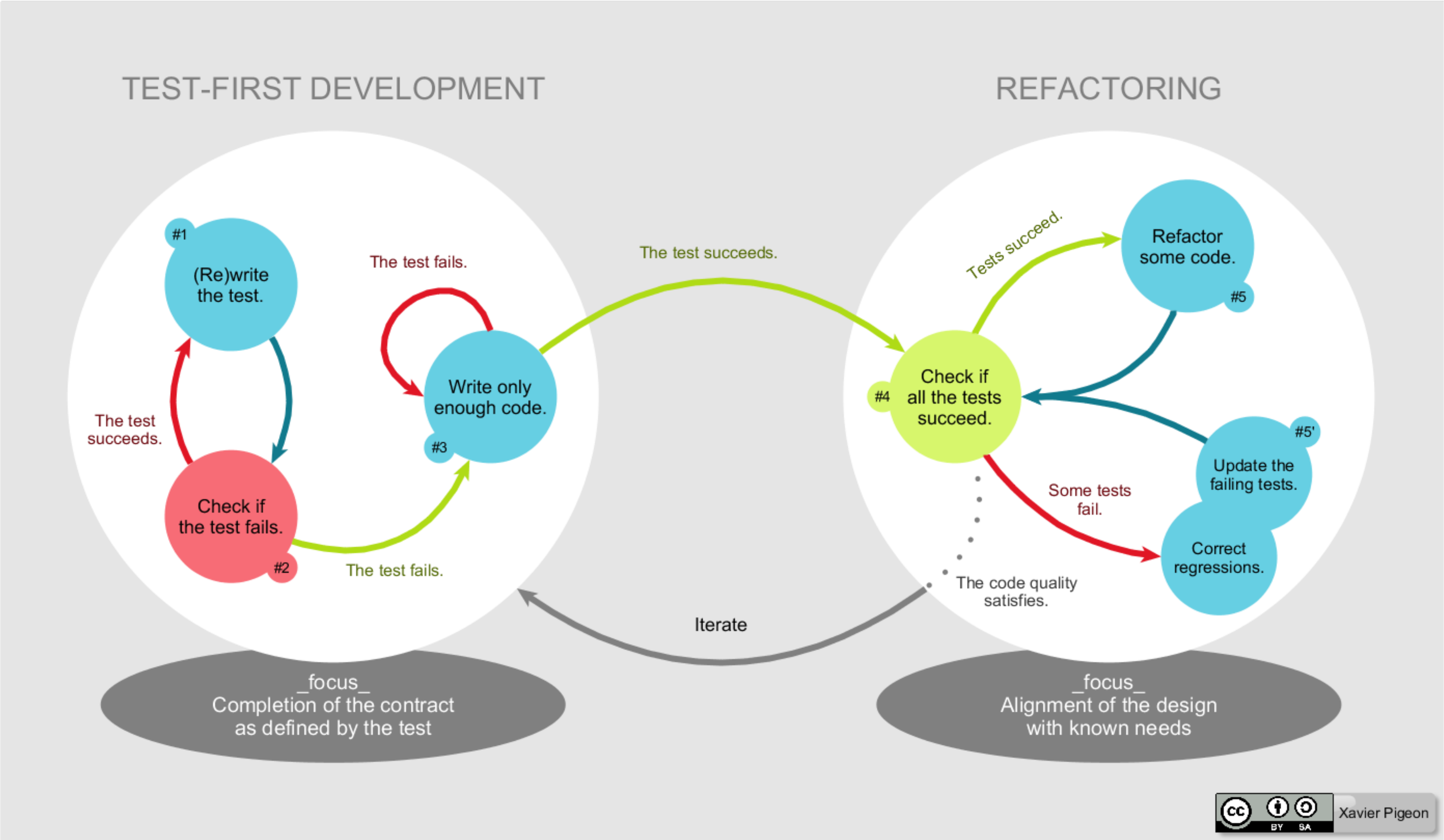
- **Writing tests before code** clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
  - Usually relies on a testing framework such as Junit.
- All previous and new tests are run automatically when new functionality is added, thus checking that the new functionality has not introduced errors.
- TDD
  - is a software development process that relies on the repetition of a very short development cycle. **Requirements are turned into very specific test cases**, then the software is improved to pass the new tests.
    - The test-code-refactor cycle



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

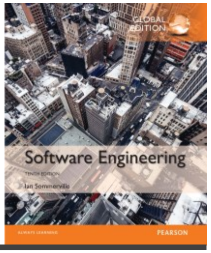
Test-driven development (TDD) (from wiki)





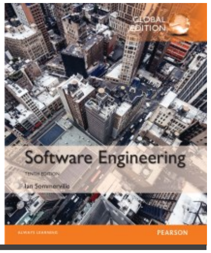
Xavier Pigeon

Customer involvement



- The role of the customer in the testing process is to help develop **acceptance tests** for the stories that are to be implemented in the next release of the system.
- The customer who is part of the team 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.

Test case description for dose checking



Test 4: Dose checking

Input:

1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

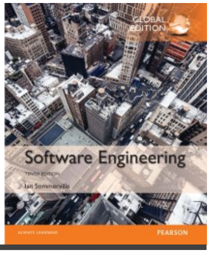
Tests:

1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

Output:

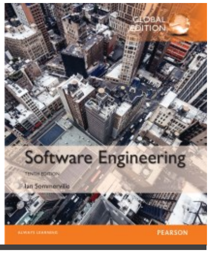
OK or error message indicating that the dose is outside the safe range.

Test automation



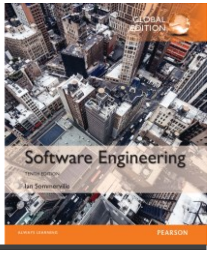
- ✧ Test automation means that tests are written as executable components before the task is implemented
  - These testing components should be stand-alone, should simulate the submission of input to be tested and should check that the result meets the output specification. An automated test framework (e.g. Junit) is a system that makes it easy to write executable tests and submit a set of tests for execution.
- ✧ 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.

Problems with test-first development



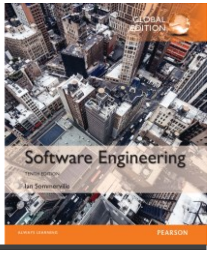
- ✧ Programmers prefer programming to testing and sometimes they take short cuts when writing tests. For example, they may write incomplete tests that do not check for all possible exceptions that may occur.
- ✧ Some tests can be very difficult to write incrementally. For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the 'display logic' and workflow between screens.
- ✧ It difficult to judge the completeness of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage.

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.
- It encourages refactoring as the whole team can benefit from improving the system code.

Pair programming

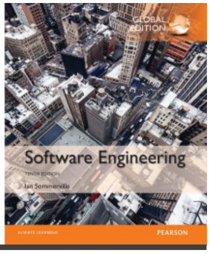


✧ In pair programming, programmers sit together at the same computer to develop the software.

✧ Pairs are created dynamically so that all team members work with each other during the development process.

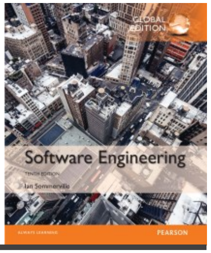
✧ The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.

✧ 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.



Supplemental slides

The Waterfall Process

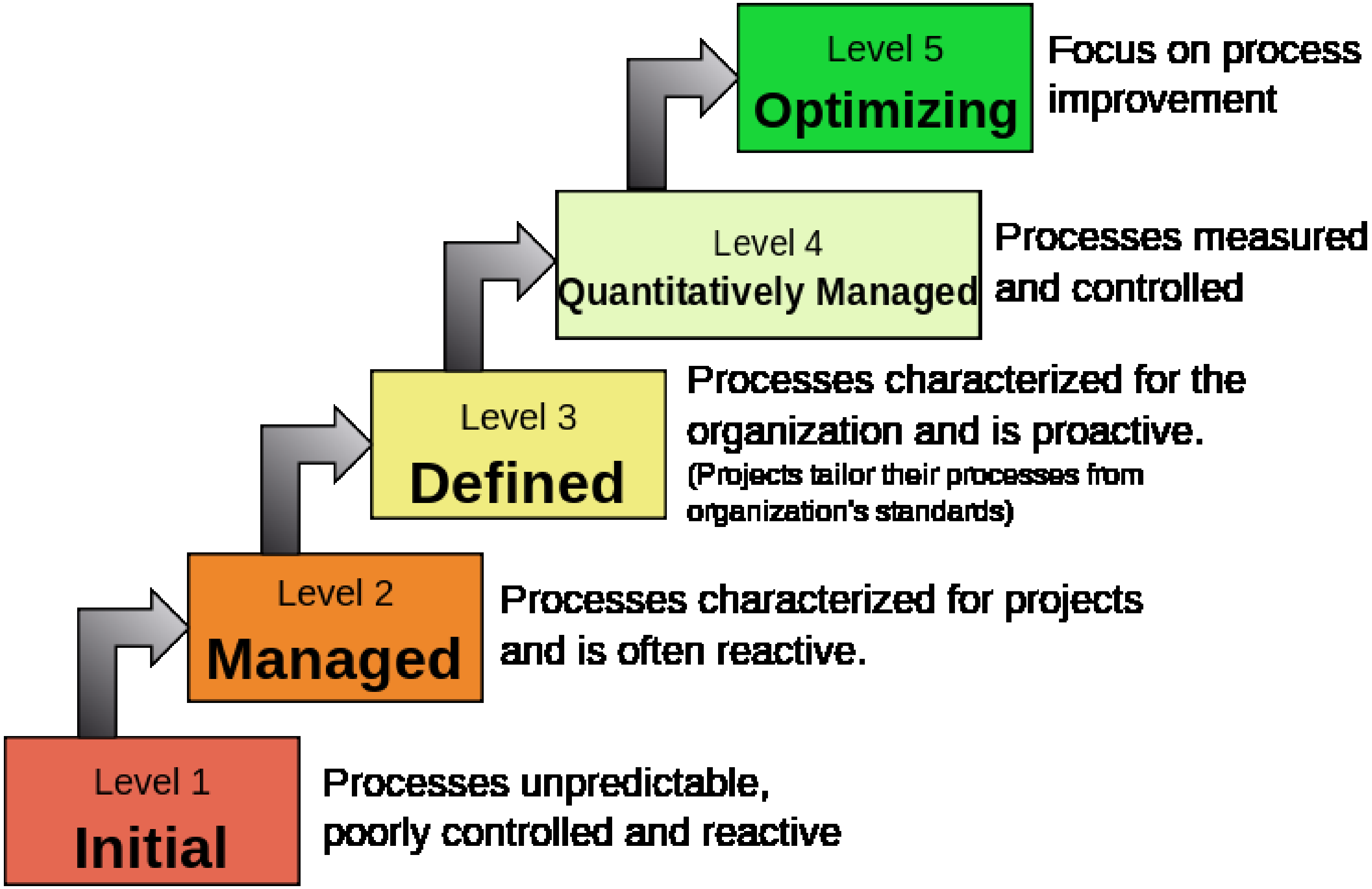


- It is supremely logical, think before you build, write it all down, follow a plan, and keep everything as organized as possible
  - Good for large and long-life systems in large organizations

- **Plan-driven** process requires that the good ideas all come at the beginning of the release cycle, where they can be incorporated into the plan
  - This also makes it difficult to accommodate changes after the process is underway
- It also places a great emphasis on writing things down as a primary method for communicating critical information
  - Documents have merits. However, engineers are not good at writing documents. Incomplete, ambiguous, and inconsistent documents can also cause problems and waste efforts

CMMI (Capability Maturity Model Integration)

Characteristics of the Maturity levels



Level 5
Optimizing
Focus on process improvement

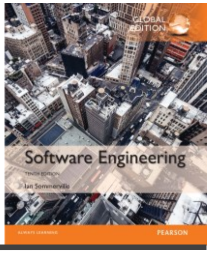
Level 4
Quantitatively Managed
Processes measured and controlled

Level 3
Defined
Processes characterized for the organization and is proactive. (Projects tailor their processes from organization's standards)

Level 2
Managed
Processes characterized for projects and is often reactive.

Level 1
Initial
Processes unpredictable, poorly controlled and reactive

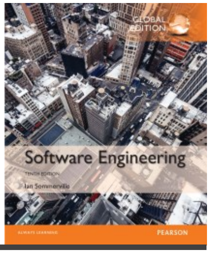
Scrum



- ✧ Scrum原始意義是橄欖球運動中的**列陣爭球**
- ✧ Sprint的意義是**衝刺**
- ✧ Developed in early 1990 by
  - Ken Schwaber
  - Jeff Sutherland

- ✧ Scrum
  - taking a short step of development
  - **inspecting** both the resulting product and the efficacy of current practices
  - **adapting** the product goals and process practices
  - *Repeat forever*

Definition of Scrum

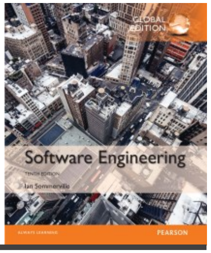


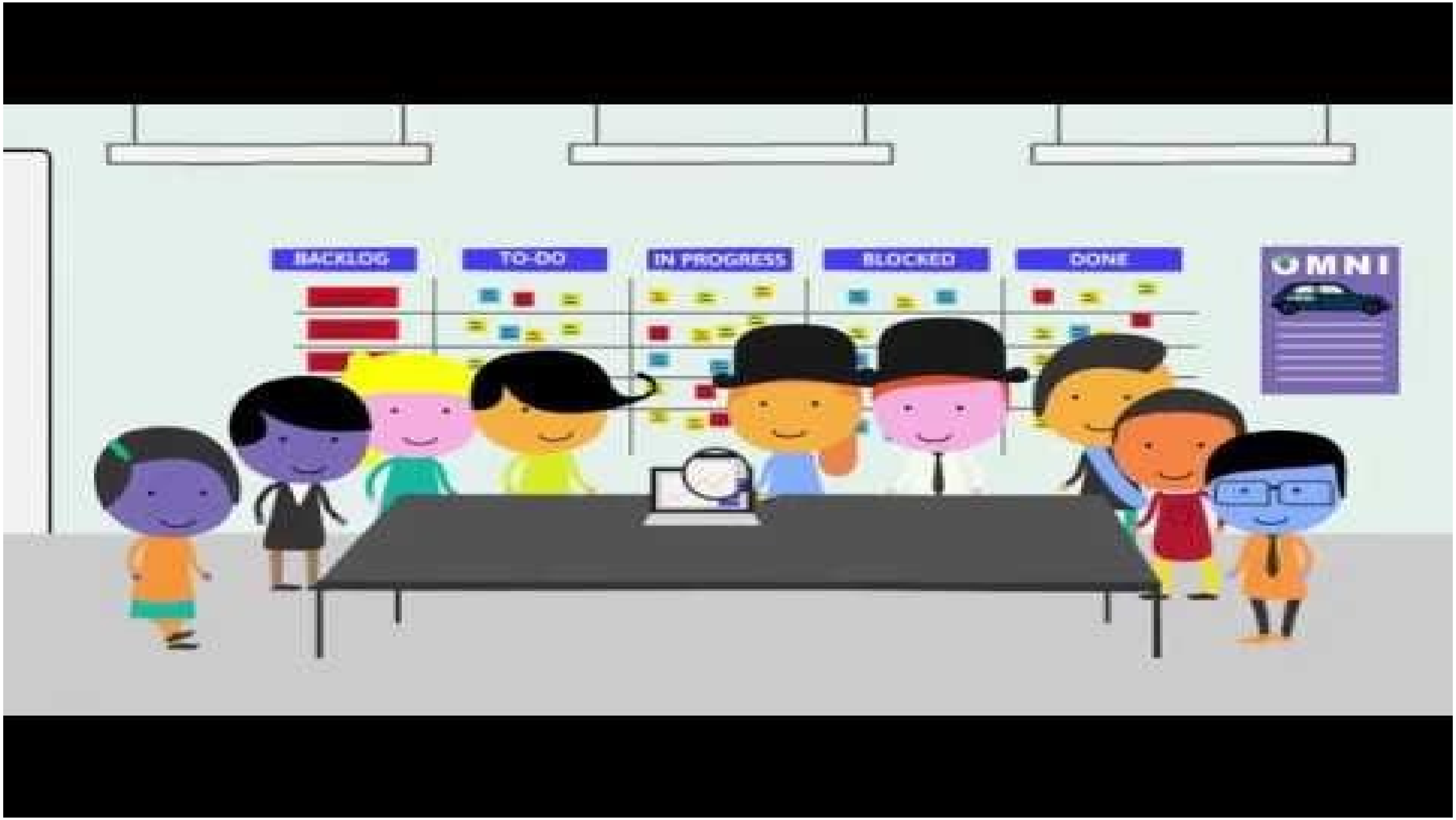
- ✧ Scrum is a **framework** for developing, delivering, and sustaining complex products
  - A **framework** within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value

- ✧ Scrum is:
  - Lightweight
  - Simple to understand
  - **Difficult to master**

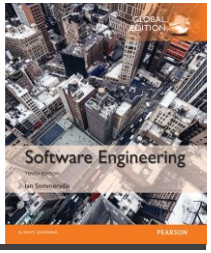
- ✧ **Scrum is not a process, technique, or definitive method**. Rather, it is a **framework** within which you can employ various processes and techniques

A Brief Overview of the Scrum Framework



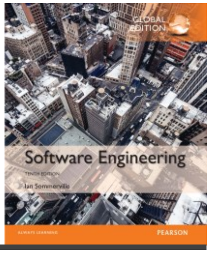


https://youtu.be/gy1c4_YixCo (2:21)



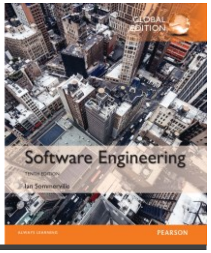
Agile project management

Agile project management



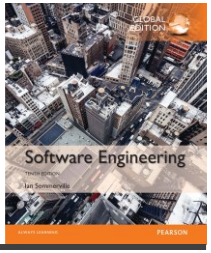
- The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.
- The standard approach to project management is plan-driven. Managers draw up a plan for the project showing what should be delivered, 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



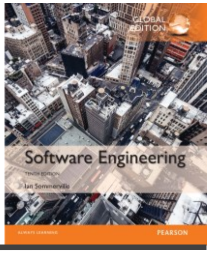
- Scrum is an agile method that focuses on managing iterative development rather than specific agile practices.
- There are three phases in Scrum.
  - The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture.
  - This 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.

Scrum terminology (a)



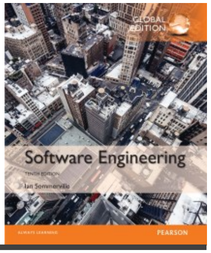
Scrum term	Definition
Development team	A self-organizing group of software developers, which should be no more than 7 people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a 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.

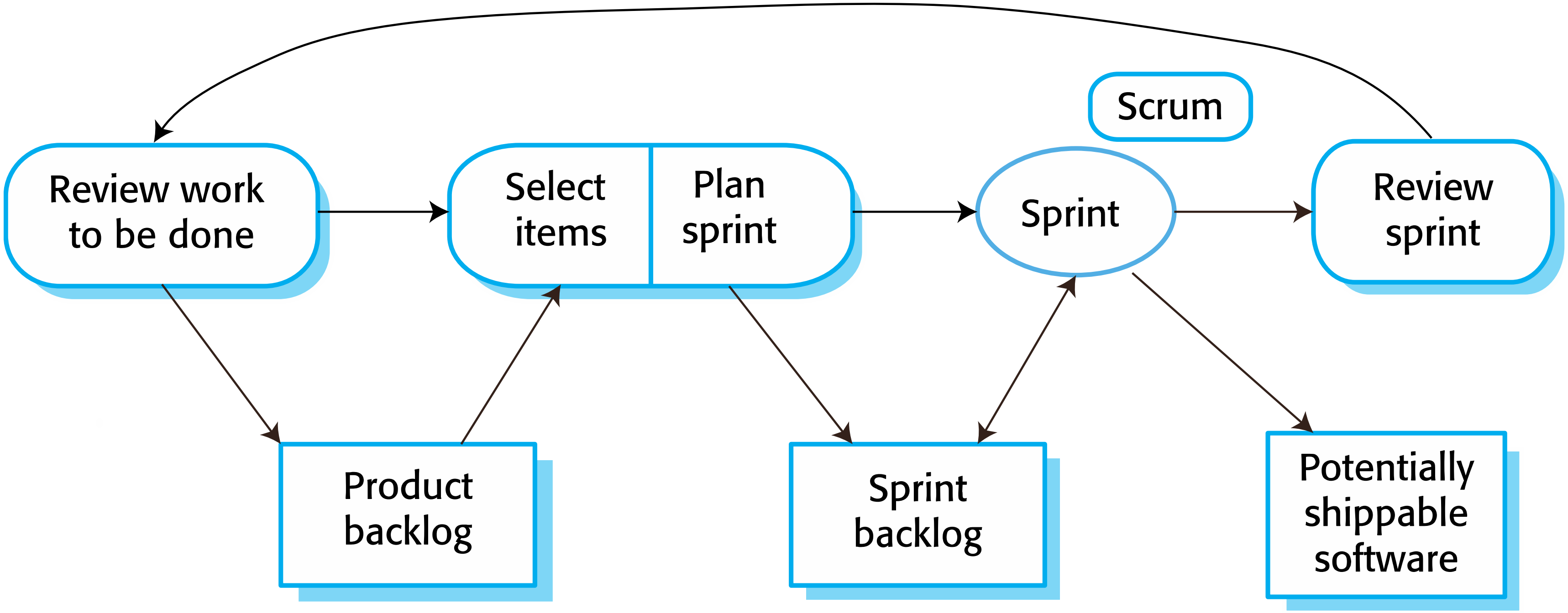
Scrum terminology (b)



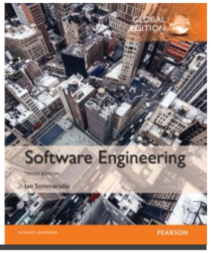
Scrum term	Definition
Scrum	A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.
ScrumMaster	The ScrumMaster is responsible for ensuring that the Scrum process is followed and 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 2-4 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



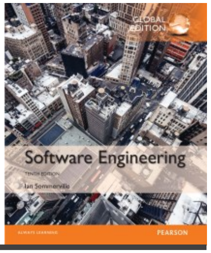


The Scrum sprint cycle



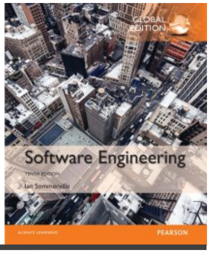
- ✧ Sprints are fixed length, normally 2–4 weeks.
- ✧ The starting point for planning is the product backlog, which is the list of work to be done on the project.
- ✧ The selection phase involves all of the project team who work with the customer to select the features and functionality from the product backlog 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.

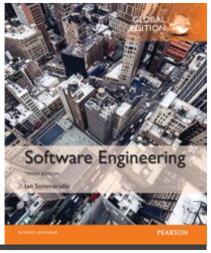
Teamwork in Scrum

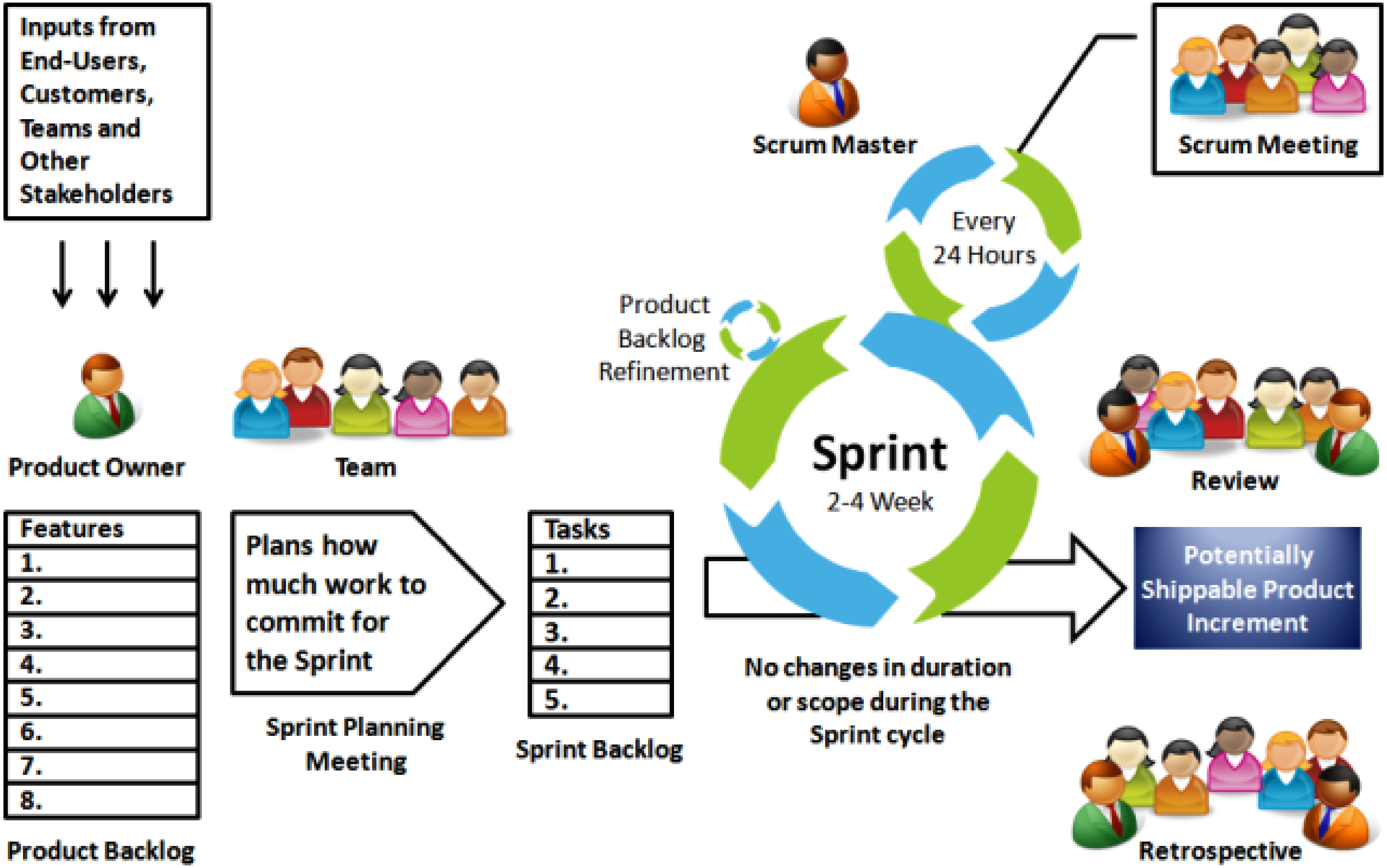


- The 'Scrum master' is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.

- 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 Process Framework





Inputs from End-Users, Customers, Teams and Other Stakeholders

Product Owner

Team

Features
1.
2.
3.
4.
5.
6.
7.
8.

Product Backlog

Plans how much work to commit for the Sprint

Sprint Planning Meeting

Tasks
1.
2.
3.
4.
5.

Sprint Backlog

Product Backlog Refinement

Scrum Master

Every 24 Hours

Scrum Meeting

Sprint
2-4 Week

No changes in duration or scope during the Sprint cycle

Review

Potentially Shippable Product Increment

Retrospective

Scrum on a Page

Roles

Product Owner
Set Priorities
Manage Product Backlog

Scrum Master
Teach Scrum
Manage Process
Protect Team
Enforce Rules
Remove Blocks

Team
Develop Product
Organize Work
Report Progress

Stakeholders
Observe & advise

Artifacts

Product Backlog
List of requirements
Owned by product owner
Anybody can add to it
Prioritized by business value
Can change without affecting the active sprint

Sprint Goal
One sentence summary
Defined by Product Owner
Accepted by Team

Sprint Backlog
Decomposed task list
Driven by a portion of Product Backlog
Owned by Team
Only Team modifies it

Blocks List
List of blocks & pending decisions
Owned by Scrum Master
Blocks stay on list until resolved

Increment
Version of the product
Potentially shippable
Working functionality
Tested & documented according to project definition of "DONE"

Meetings

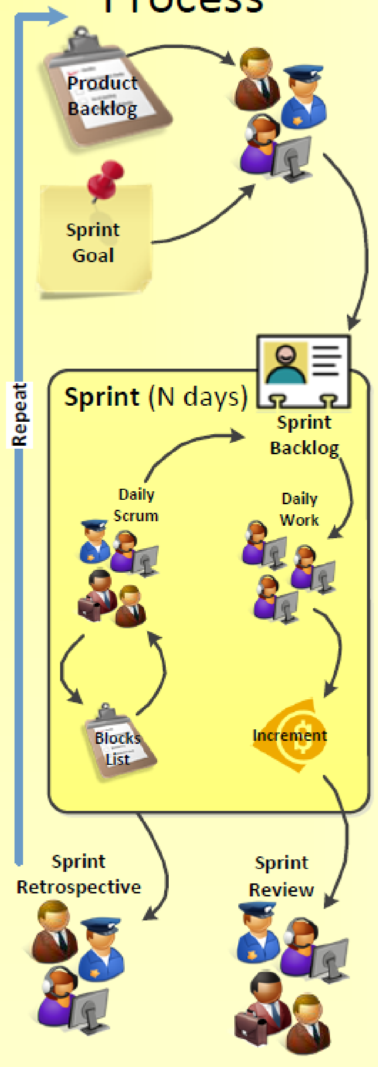
Sprint Planning
Part A
Time-boxed to 4 hours
Run by Scrum Master
Declare Sprint Goal
Top of Product Backlog presented by Product Owner to Team
Team asks questions & selects topmost features
Part B
Time-boxed to 4 hours
Run by Scrum Master
Team decomposes selected features into a Sprint Backlog
Team adjusts +/- features by estimates against sprint capacity

Daily Scrum
Time-boxed to 15 minutes
Run by Scrum Master
Attended by all
Stakeholders do not speak
Same time/place every day
Answer 3 questions:
1) What I did yesterday?
2) What I'll do today?
3) What's in my way?
Team updates the Sprint Backlog
Scrum Master updates the Blocks List

Sprint Review
Time-boxed to 2 or 4 hours
Run by Scrum Master
Attended by all
Informal, informational, Discussion
Team demonstrates increment
All discuss

Sprint Retrospective
Time-boxed to 1 or 2 hours
Run by Scrum Master
Attended by Team and Product Owner
Discuss process improvements, successes and failures
Adjust process

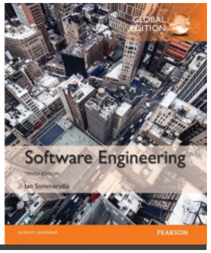
Process

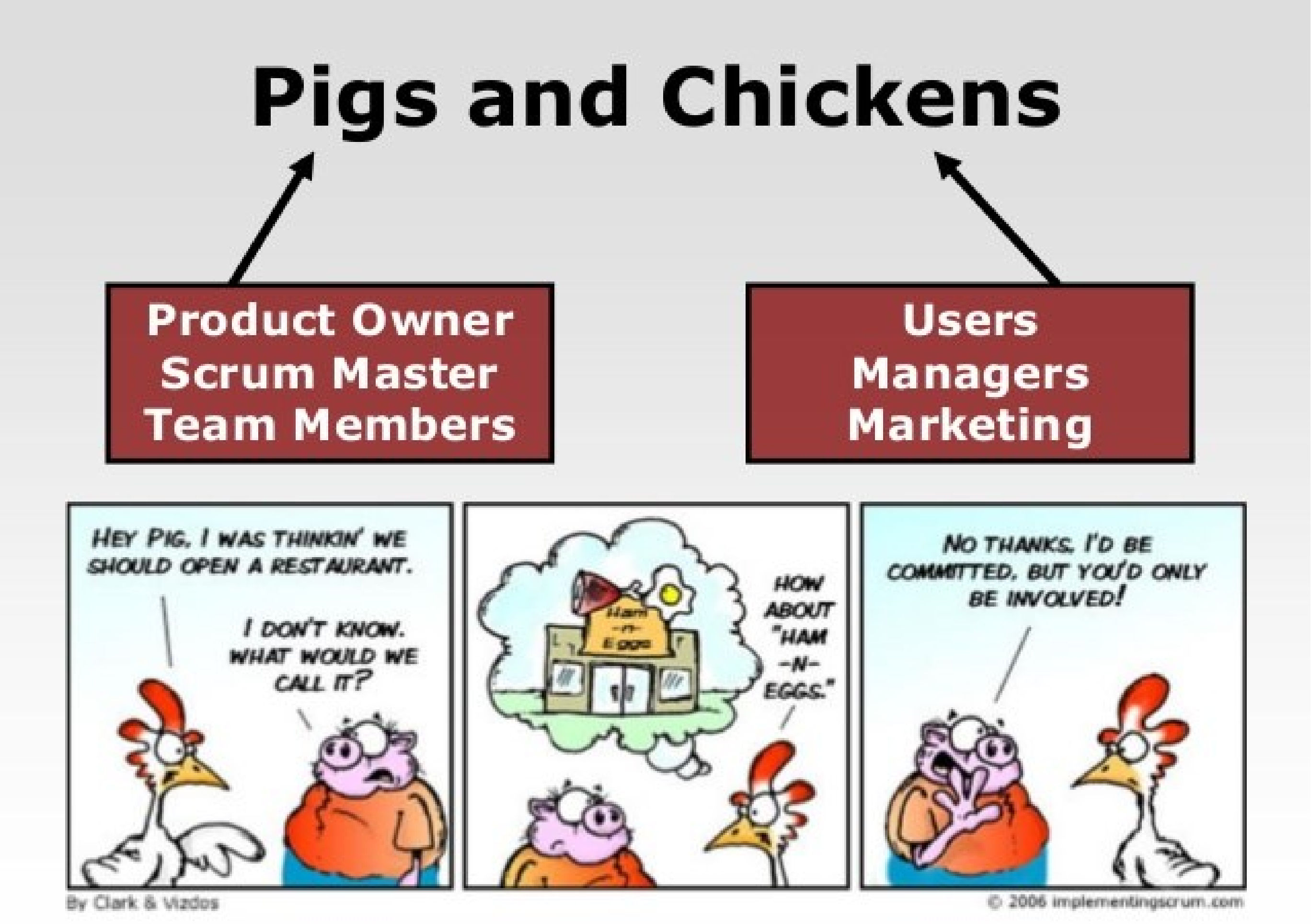


Repeat
Product Backlog
Sprint Goal
Sprint (N days)
Sprint Backlog
Daily Scrum
Daily Work
Blocks List
Increment
Sprint Retrospective
Sprint Review

www.andrewsiemer.com / (661) 600-2355 / asiemer@hotmail.com

Committed and Involved





Pigs and Chickens

Product Owner
Scrum Master
Team Members

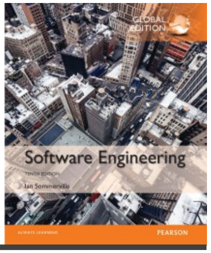
Users
Managers
Marketing

HEY PIG, I WAS THINKIN' WE SHOULD OPEN A RESTAURANT.
I DON'T KNOW. WHAT WOULD WE CALL IT?
Ham -n- Eggs
HOW ABOUT "HAM -N- EGGS."
NO THANKS. I'D BE COMMITTED, BUT YOU'D ONLY BE INVOLVED!

By Clark & Vizdos

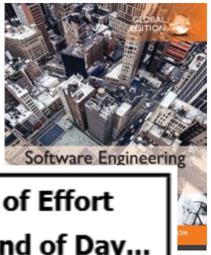
© 2006 implementingscrum.com

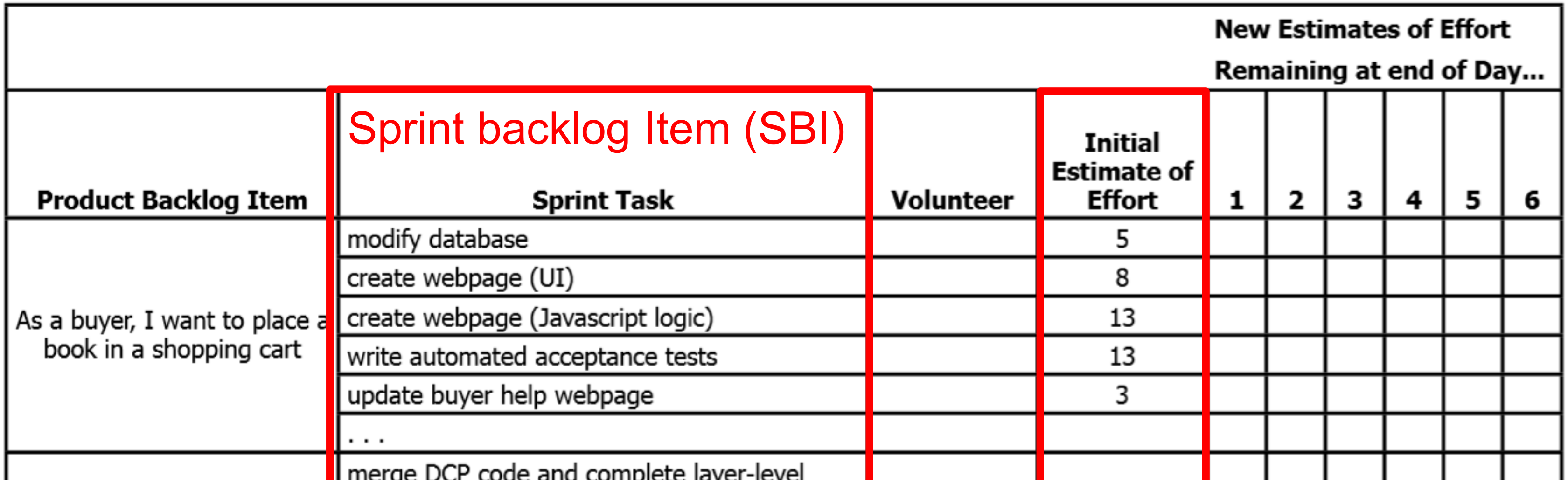
Story Point Estimation (Planning Poker)



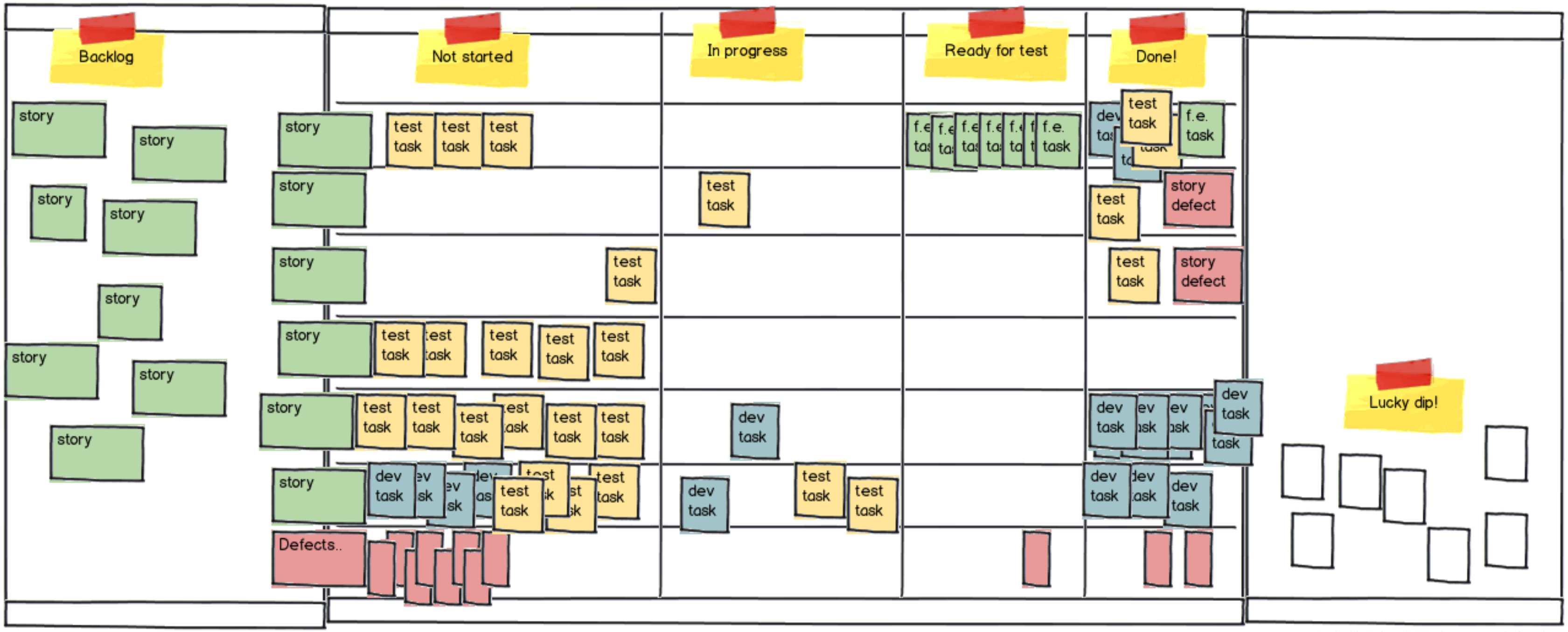
https://youtu.be/TxSzo3lwwWQ (3:14)

Create a Sprint Backlog





				New Estimates of Effort Remaining at end of Day...					
Product Backlog Item	Sprint backlog Item (SBI) Sprint Task	Volunteer	Initial Estimate of Effort	1	2	3	4	5	6
As a buyer, I want to place a book in a shopping cart	modify database		5						
	create webpage (UI)		8						
	create webpage (Javascript logic)		13						
	write automated acceptance tests		13						
	update buyer help webpage		3						
	. . .								
	merge DCP code and complete layer-level								

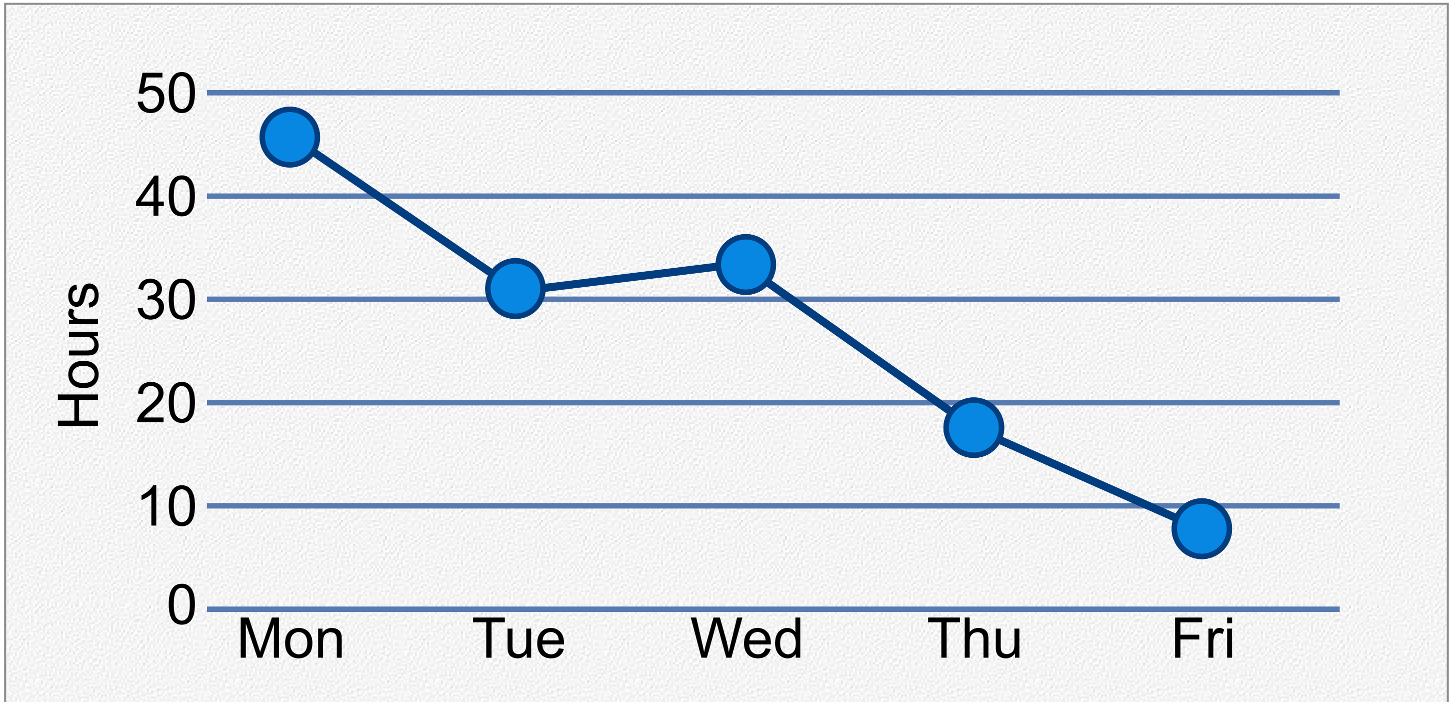


created with Balsamiq Mockups - www.balsamiq.com

Daily Scrum Meeting

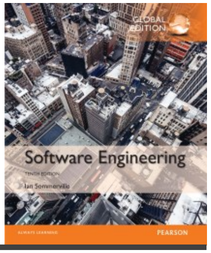
- Summary:
  - Update and coordination between Team members
- Participants:
  - Team is required (everyone on the Team attends, anyone late pays a $1 fee); ScrumMaster is usually present but ensures Team holds daily Scrum meeting; Product Owner is optional
- Duration: (daily)
  - Maximum length of 15 minutes (stand-up)
- Three questions answered by each team member:
  - What did you do yesterday? (to help Team meet the Sprint Goal)
  - What will you do today? (to help Team meet the Sprint Goal)
  - What obstacles are in your way? (for meeting the Sprint Goal)
  - The intent of these questions is to emphasize **completions** of tasks, rather than effort spent
    - Having a clear definition of Done may help

Tasks	Mon	Tue	Wed	Thu	Fri
Code the user interface	8	4	8		
Code the middle tier	16	12	10	7	
Test the middle tier	8	16	16	11	8
Write online help	12				



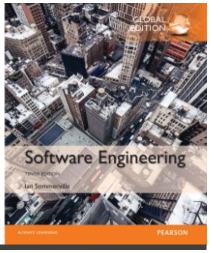
Burndown Chart

Scrum benefits



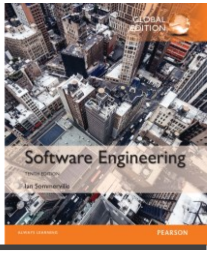
- ✧ The product is broken down into a set of manageable and understandable chunks.
- ✧ Unstable requirements do not hold up progress.
- ✧ The whole team have visibility of everything and consequently team communication is improved.
- ✧ Customers see on-time delivery of increments and gain feedback on 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.

Scrum in under 5 Minutes



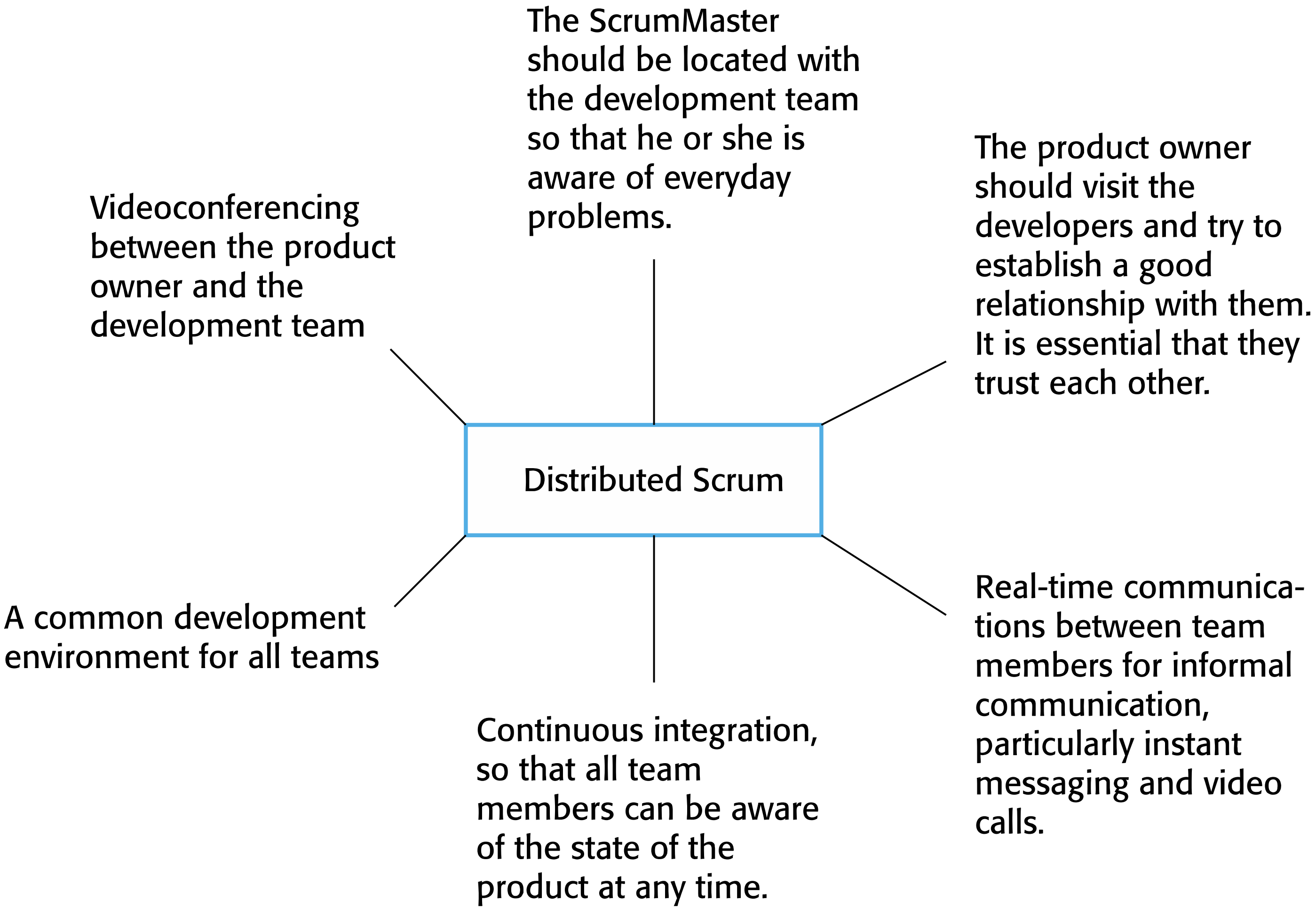
https://youtu.be/2Vt7Ik8Ublw (4:31)

Distributed Scrum



- ✧ Scrum , as originally designed, was intended for use with **co-located teams** where all team members could get together every day in stand-up meeting.
- ✧ For software development involving distributed teams, with team members located in different places around the world to take advantages of lower costs staffs, accessing specialist skills, and allowing for 24-hour development, Scrum for distributed development environments and multi-team working is needed.
  - **Distributed Scrum**

The Requirements of Distributed Scrum



Distributed Scrum

Videoconferencing between the product owner and the development team

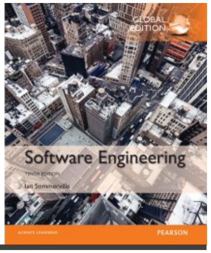
The ScrumMaster should be located with the development team so that he or she is aware of everyday problems.

The product owner should visit the developers and try to establish a good relationship with them. It is essential that they trust each other.

A common development environment for all teams

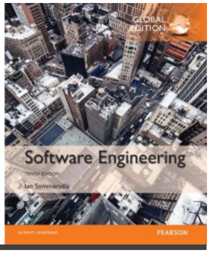
Continuous integration, so that all team members can be aware of the state of the product at any time.

Real-time communications between team members for informal communication, particularly instant messaging and video calls.



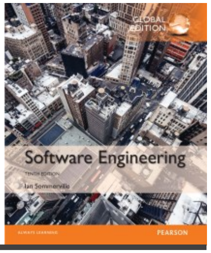
Scaling agile methods

Scaling agile methods



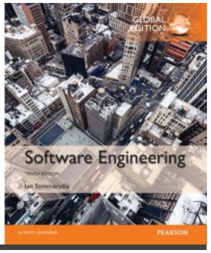
- Agile methods have proved to be successful for small and medium sized projects that can be developed by a small co-located team.
- It is sometimes argued that the success of these methods comes because of improved communications which is possible when **everyone is working together**.
- Scaling up 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



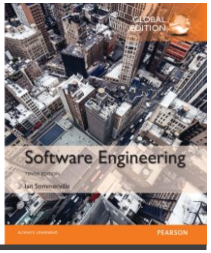
- ✧ **'Scaling up'** is concerned with using agile methods for developing large software systems that cannot be developed by a small team.
- ✧ **'Scaling out'** is concerned with how agile methods can be introduced across a large organization with many years of software development experience.
  - Multiple teams
- ✧ Scaling up and scaling out are **closely related** and the **large companies** will face the problems of scaling up and scaling out **at the same time**
- ✧ 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.
- ✧ Ambler suggests that an organization moving to agile methods can expect to see productivity improvement across the organization of about 15% over 3 years, with similar reductions in the number of product defects

Problems with agile methods



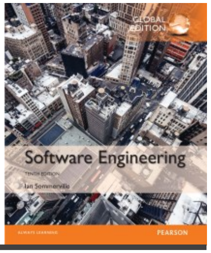
- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the intense involvement that characterises agile methods.
- Prioritising changes can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work.
- Contracts may be a problem as with other approaches to iterative development
- Organization culture may not fit into the agile working model in which processes are informal and defined by development team.

Practical problems with agile methods



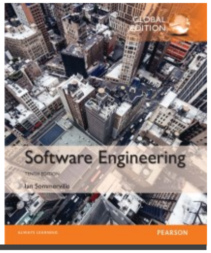
- ✧ For large, long-lifetime systems that are developed by a software company for an external client, using an agile approach presents a number of problems
  - The informality of agile development is incompatible with the legal approach to contract definition that is commonly used in large companies.
  - 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.

Agile and plan-driven methods



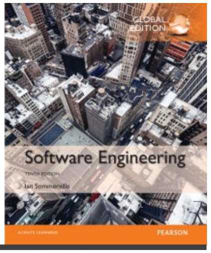
- ✧ Most projects include elements of plan-driven and agile processes. Deciding on the balance depends on:
  - Is it important to have a very detailed specification and design before moving to implementation? If so, you probably need to use a plan-driven approach.
  - Is an incremental delivery strategy, 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



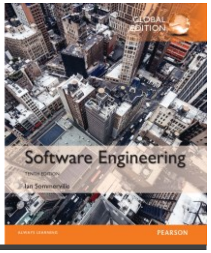
Principle	Practice
Customer involvement	This depends on having a customer who is willing and able to spend time with the development team and who can represent all system stakeholders. Often, customer representatives have other demands on their time and cannot play a full part in the software development. Where there are external stakeholders, such as regulators, it is difficult to represent their views to the agile team.
Embrace change	Prioritizing changes can be extremely difficult, especially in systems for which there are many stakeholders. Typically, each stakeholder gives different priorities to different changes.
Incremental delivery	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.

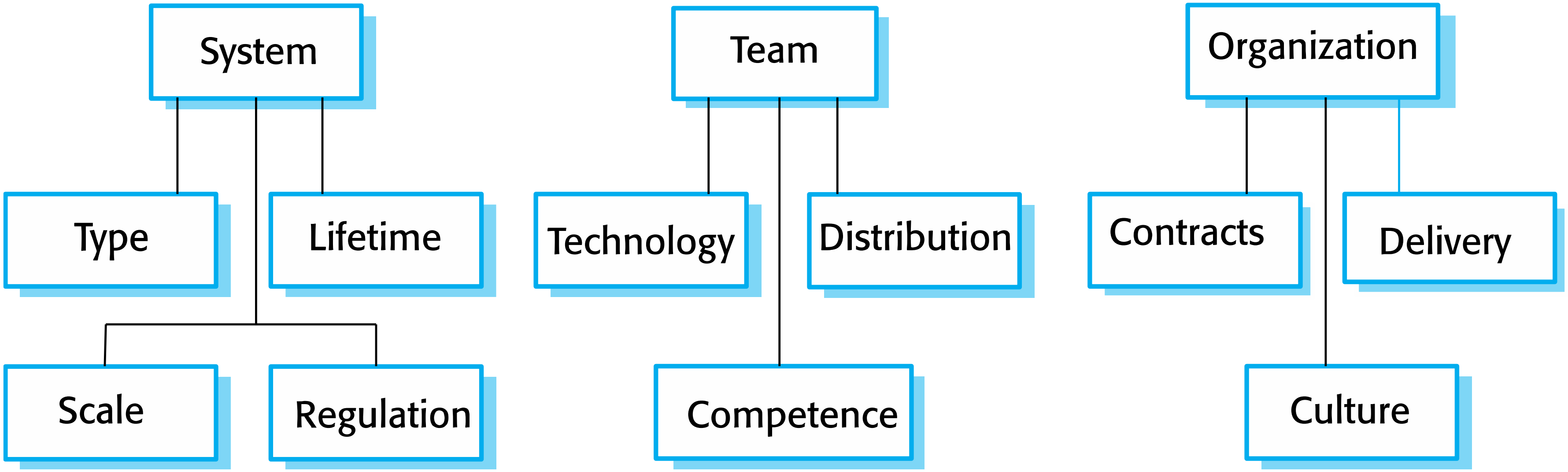
Agile principles and organizational practice



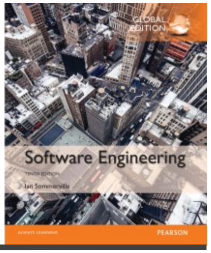
Principle	Practice
Maintain simplicity	Under pressure from delivery schedules, team members may not have time to carry out desirable system simplifications.
People not process	Individual team members may not have suitable personalities for the intense involvement that is typical of agile methods, and therefore may not interact well with other team members.

Agile and plan-based development choice factors



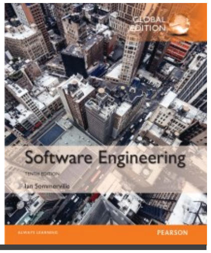


System issues



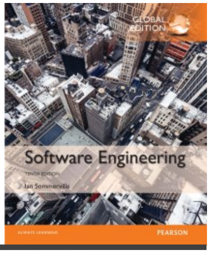
- How large is the system being developed?
  - Agile methods are most effective a relatively small co-located team who can communicate informally.
- What type of system is being developed?
  - Systems that require a lot of analysis before implementation need a fairly detailed design to carry out this 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



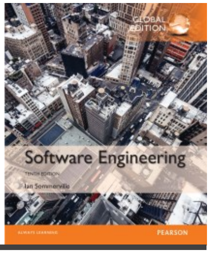
- How good are the designers and programmers in the development team?
  - It is sometimes argued that agile methods require **higher skill levels** than plan-based approaches in which programmers simply translate a detailed design into code.
- How is the development team organized?
  - **Design documents** may be required for communication across development teams if the team is **distributed** or part of the development is being **outsourced**.
- What support technologies are available?
  - IDE support for **visualisation** and **program analysis** is essential if design documentation is not available.

Organizational issues



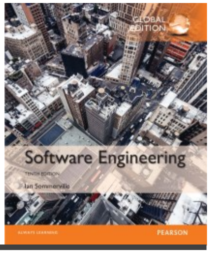
- ✧ Most software is developed in large companies that have established their own working **practices** and **procedures**. Management in these companies may be uncomfortable with the lack of documentation and the informal decision making in agile methods. Key issues are:
  - Traditional engineering 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?
- ✧ Ultimately, **software buyers concern if the system meets their needs** and do not care about whether the system is developed using plan-driven or agile method

Agile methods for large systems



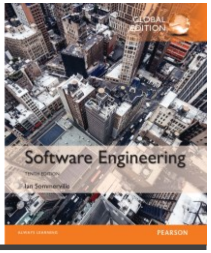
- Large-scale software systems are more complex and difficult to understand and manage. Six principal factors contribute to this complexity:
  - Large systems are usually **systems of systems** - collections of separate, communicating systems, where separate teams develop each system. Frequently, these 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.

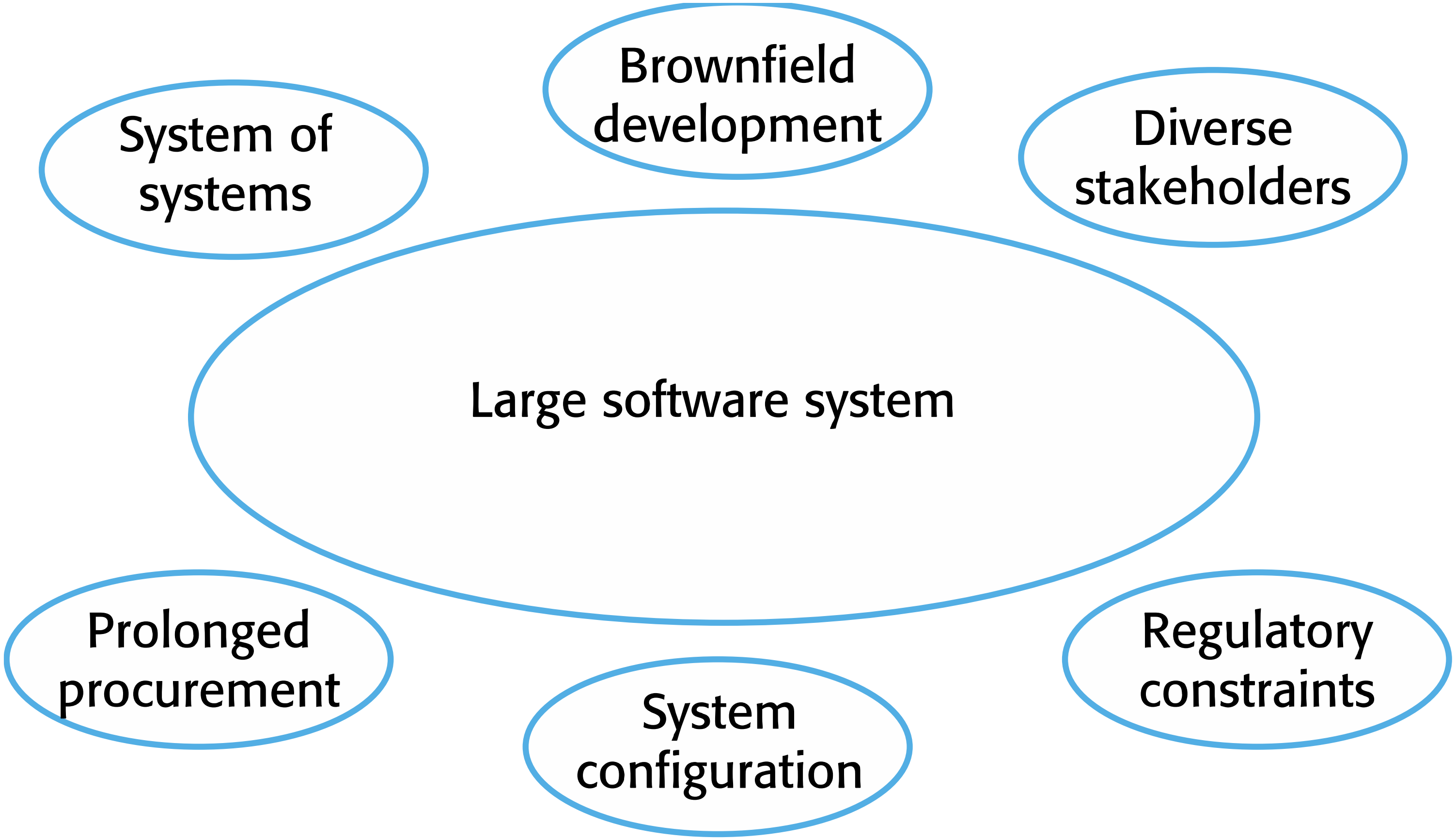
Large system development



- 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





Brownfield development

System of systems

Diverse stakeholders

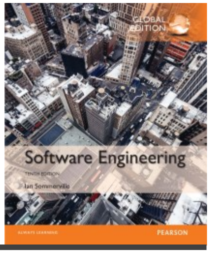
Large software system

Prolonged procurement

System configuration

Regulatory constraints

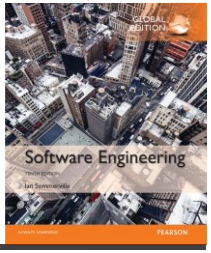
Scaling up to large systems



✧ No single model is appropriate for all large-scale agile products as the type of product, the customer requirements, and the people available are all different. However, approaches to scaling agile methods have a number of things in common:

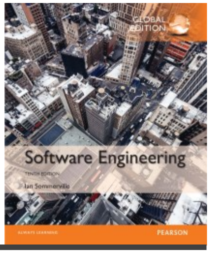
- A completely incremental approach to requirements engineering is **impossible**.
- There **cannot** be a **single** product owner or customer representative.
- For large systems development, it is **not possible to focus only on the code** of the system. (need more up-front design and documentation)
- Cross-team communication mechanisms have to be designed and used.
- **Continuous integration** is practically **impossible** **(when several separate programs have to be integrated to create the system)**. However, it is essential to maintain frequent system builds and regular releases of the system.

The key characteristics of Multi-team Scrum



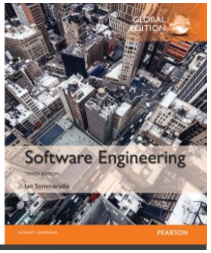
- ***Role replication***
  - Each team has a Product Owner for their work component and ScrumMaster. There may be a **chief** Product Owner and ScrumMaster for the entire project
- ***Product architects***
  - Each team chooses a **product architect** and these architects collaborate to design and evolve 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 (scale out)



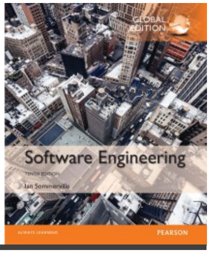
- Larger companies have also experimented with agile methods in specific project, but it is much more difficult for them to scale out agile methods across the organization. The reasons are
  - Project managers who do not have experience of agile methods may be reluctant to accept the risk of a new approach.
  - 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.

Scaling out to large companies



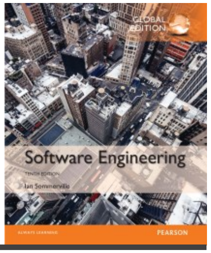
- **Change management** is the process of controlling changes to a system, so that the impact of changes is predicable and costs are controlled
  - All changes have to be approved in advance before they are made and this conflicts with the notion of **refactoring** in XP, where any developer can improve any code without getting external approval
- For large systems, there are **testing procedures and standards** where a system build is handed over to an external testing team
  - This may conflict with the **test-first** and **test-often** approaches used in XP
- Introducing and sustaining the use of agile methods across a large organization is a process of **cultural change**
  - Cultural change takes a long time to implement and often requires a change of management before it can be accomplished

Key points



- ✧ 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.

Key points



- 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 <u>a mixture of plan-based and agile development</u>.
- Scaling agile methods for large systems is difficult.
  - Large systems need <u>up-front design</u> and some <u>documentation</u> and <u>organizational practice</u> may conflict with the informality of agile approaches.