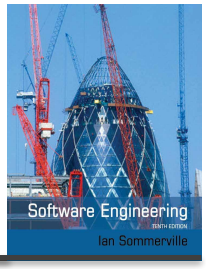
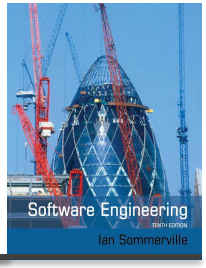


Requirements Engineering and Task Allocation

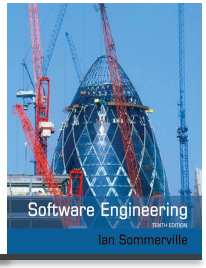
Adapted by Michael Haaf from Chapters 4 & 23 of "Software Engineering, 10th Edition" by Ian Sommerville

Topics covered

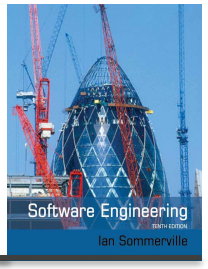


- Functional and non-functional requirements
- Requirements engineering processes
- Requirements elicitation/specification/validation/change
- User stories
- Project planning
- Task allocation

Requirements engineering

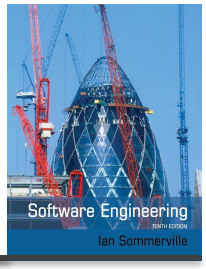


- **Requirements Engineering** is the process of establishing the expected behavior of the software system and the constraints under which it operates and is developed.
- The system requirements are the descriptions of the system services and constraints that are generated during the requirements engineering process.



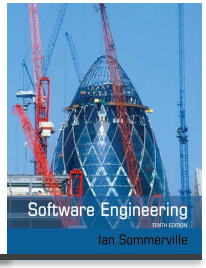
Functional and non-functional requirements

Functional and non-functional requirements



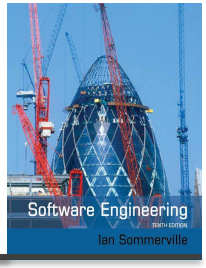
- Functional requirements
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  - These correspond to **user stories** that your team will create to describe the behavior of the software
- Non-functional requirements
  - Often apply to the system as a whole rather than individual features or services.
  - Describe non-behavioral requirements of the software

Functional requirements



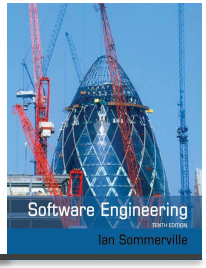
- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

Example: functional requirements of hypothetical Medical Clinic software system



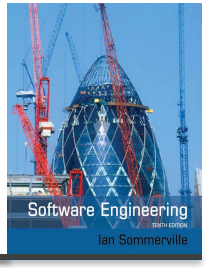
- A user shall be able to search the appointments lists for all clinics.
- The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
- Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.

Requirements imprecision



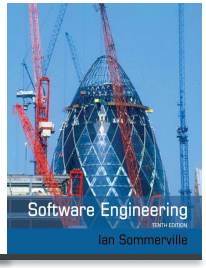
- Problems arise when functional requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
- Consider the term ‘search’ in requirement 1
  - User intention – search for a patient name across all appointments in all clinics;
  - Developer interpretation – search for a patient name in an individual clinic. User chooses clinic then search.

Requirements completeness and consistency



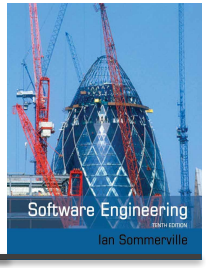
- In principle, requirements should be both complete and consistent.
- Complete
  - They should include descriptions of all facilities required.
- Consistent
  - There should be no conflicts or contradictions in the descriptions of the system facilities.
- In practice, because of system and environmental complexity, it is impossible to produce a complete and consistent requirements document.

Non-functional requirements



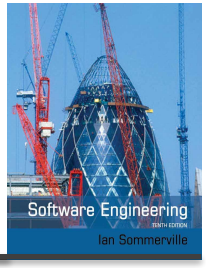
- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.
- **Refactoring**, **code-style**, **linting**, and other changes that **do not change the functional behavior of the software** fit this category.

Requirements creation: Stories and scenarios

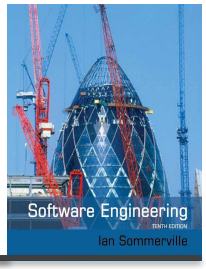


- Scenarios and user stories are real-life examples of how a system can be used.
- Stories and scenarios are a description of how a system may be used for a particular task.
- Because they are based on a practical situation, stakeholders can relate to them and can comment on their situation with respect to the story.

Scenarios

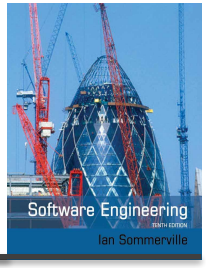


- A structured form of user story
- Scenarios should include
  - A description of the starting situation;
  - A description of the normal flow of events;
  - A description of what can go wrong;
  - Information about other concurrent activities;
  - A description of the state when the scenario finishes.



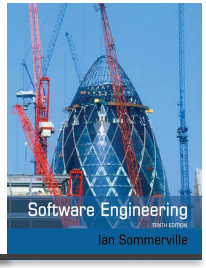
Requirements specification

Requirements specification



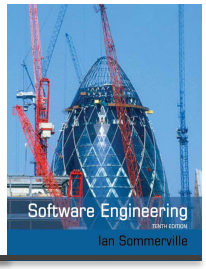
- The process of writing down the user and system requirements in a requirements document.
- User requirements have to be understandable by end-users and customers who do not have a technical background.

Natural language specification



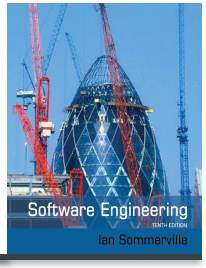
- Requirements are written as natural language sentences supplemented by diagrams and tables.
- Used for writing requirements because it is expressive, intuitive and universal. This means that the requirements can be understood by users and customers.

Guidelines for writing requirements



- Invent a standard format and use it for all requirements.
- Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements.
- Use text highlighting to identify key parts of the requirement.
- Avoid the use of computer jargon.
- Include an explanation (rationale) of why a requirement is necessary.

Writing a User Story to specify a requirement

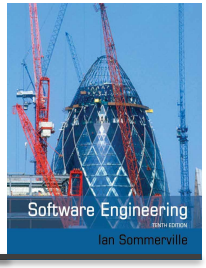


Consider the following when writing user stories:

- **Definition of "done"** — The story is generally "done" when the user can complete the outlined task, but make sure to define what that is.
- **Outline subtasks or tasks** — Decide which specific steps need to be completed and who is responsible for each of them.
- **User personas** — For whom? If there are multiple end users, consider making multiple stories.
- **Ordered Steps** — Write a story for each step in a larger process.
- **Listen to feedback** — Talk to your users and capture the problem or need in their words. No need to guess at stories when you can source them from your customers.
- **Time** — Time is a touchy subject. Many development teams avoid discussions of time altogether, relying instead on their estimation frameworks. Since stories should be completable in one sprint, stories that might take weeks or months to complete should be broken up into smaller stories or should be considered their own epic.

See https://www.atlassian.com/agile/project-management/user-stories for more detail.

Writing a User Story to specify a requirement



“As a [persona], I [want to], [so that].”

Breaking this down:

- "As a [persona]": Who are we building this for? We’re not just after a job title, we’re after the persona of the person. Max. Our team should have a shared understanding of who Max is. We’ve hopefully interviewed plenty of Max’s. We understand how that person works, how they think and what they feel. We have empathy for Max.
- “Wants to”: Here we’re describing their intent — not the features they use. What is it they’re actually trying to achieve? This statement should be implementation free — if you’re describing any part of the UI and not what the user goal is you're missing the point.
- “So that”: how does their immediate desire to do something this fit into their bigger picture? What’s the overall benefit they’re trying to achieve? What is the big problem that needs solving?

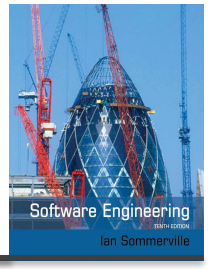
For example, user stories might look like:

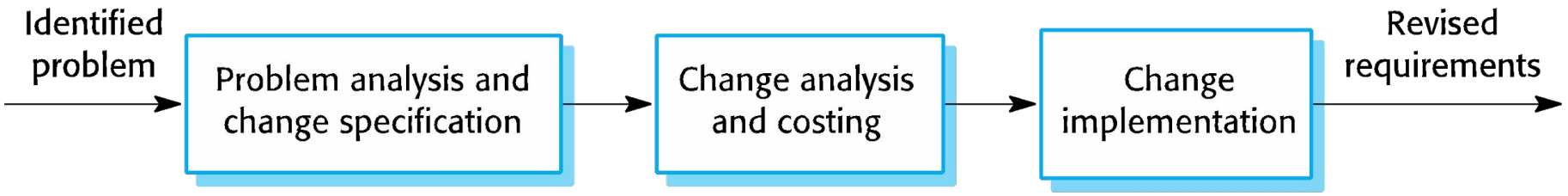
- As Max, I want to invite my friends, so we can enjoy this service together.
- As Sascha, I want to organize my work, so I can feel more in control.
- As a manager, I want to be able to understand my colleagues progress, so I can better report our sucess and failures.

This structure is not required, but it is helpful for defining done. When that persona can capture their desired value, then the story is complete.

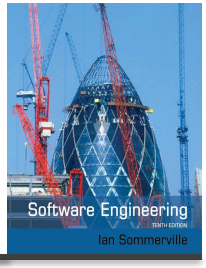
See https://www.atlassian.com/agile/project-management/user-stories for more detail.

Requirements change management



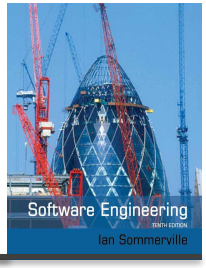


Key points



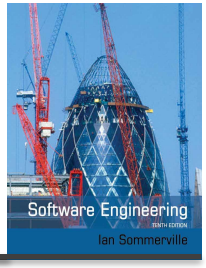
- **Requirements** for a software system set out what the system should do and define constraints on its operation and implementation.
- **Functional requirements** are statements of the services that the system must provide or are descriptions of how some computations must be carried out.
- **Non-functional requirements** often constrain the system being developed and the development process being used. They often relate to the emergent properties of the system and therefore apply to the system as a whole.

Key points



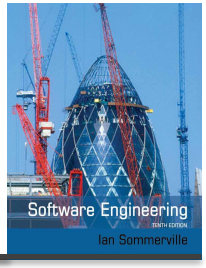
- The requirements engineering process is an iterative process that includes requirements elicitation, specification and validation.
- **Requirements elicitation** is an iterative process that can be represented as a spiral of activities – requirements discovery, requirements classification and organization, requirements negotiation and requirements documentation.
-

Key points



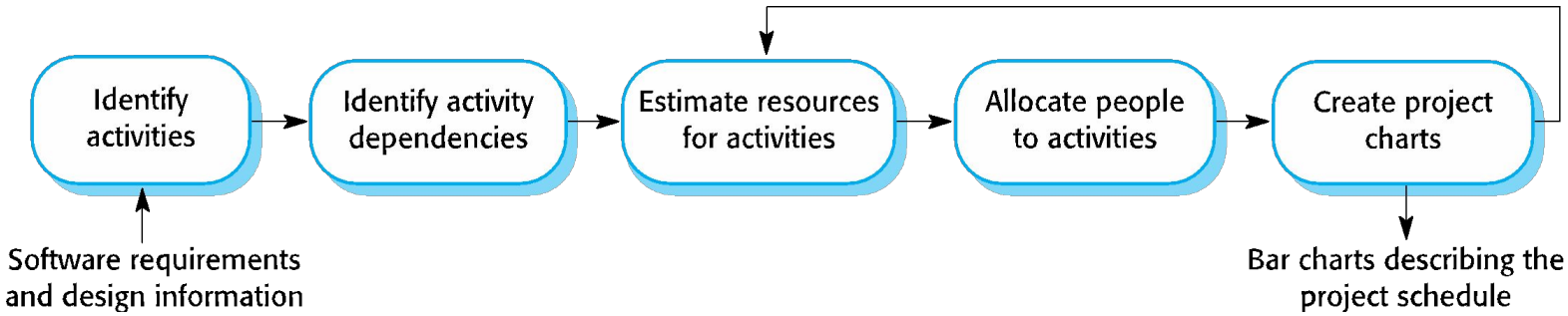
- **Requirements specification** is the process of formally documenting the user and system requirements and creating a software requirements document.
- **User stories and scenarios** are tools for expressing and documenting requirement specifications.

Project planning

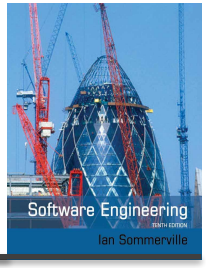


- Project planning involves breaking down the work into parts and assign these to project team members, anticipate problems that might arise and prepare tentative solutions to those problems.
- The project plan, which is created at the start of a project, is used to communicate how the work will be done to the project team and customers, and to help assess progress on the project.

The project scheduling process

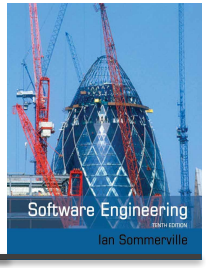


Project activities



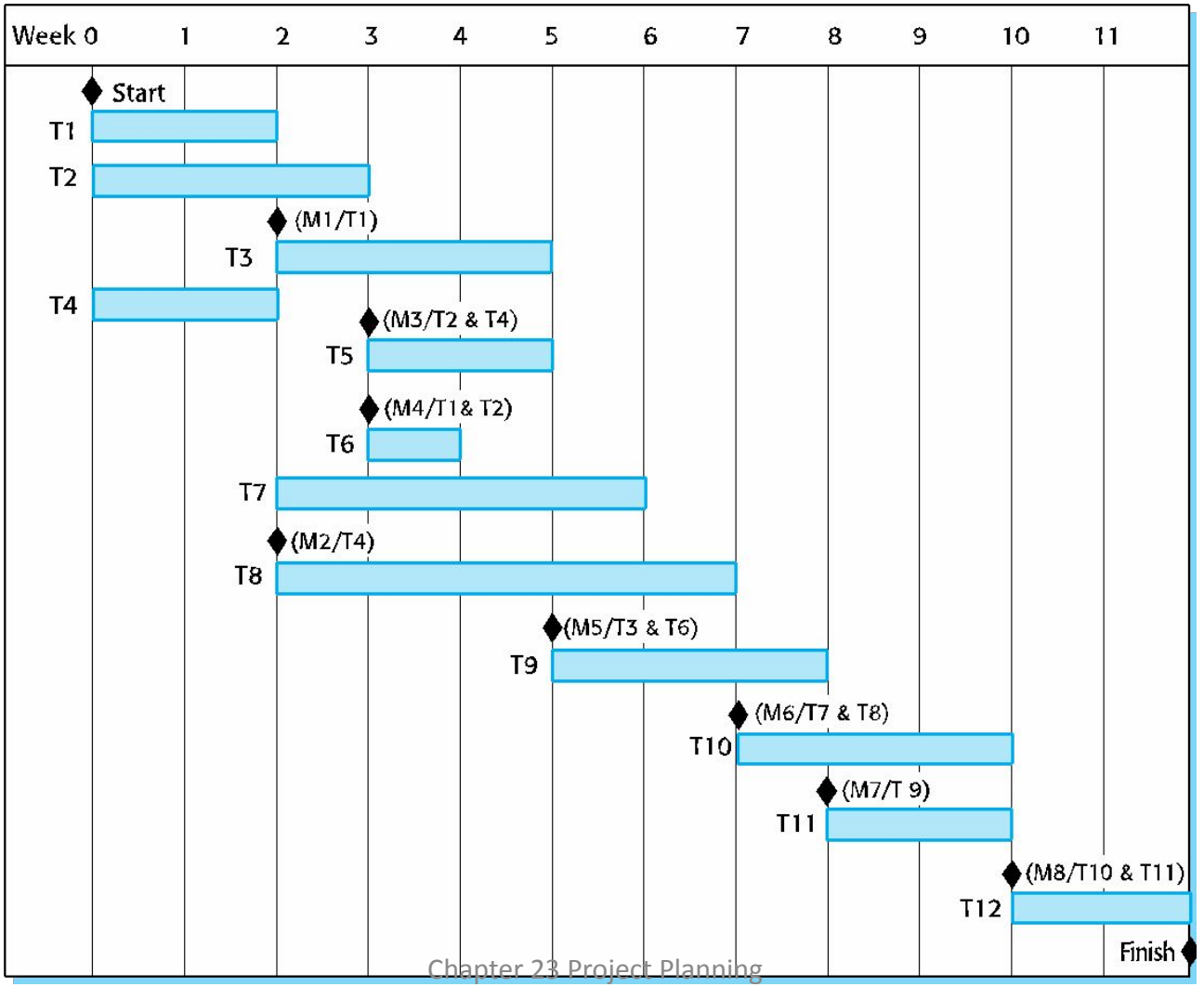
- Project activities (tasks) are the basic planning element. Each activity has:
  - a duration in calendar days or months,
  - an effort estimate, which shows the number of person-days or person-months to complete the work,
  - a deadline by which the activity should be complete,
  - a defined end-point, which might be a document, the holding of a review meeting, the successful execution of all tests, etc.

Milestones and deliverables

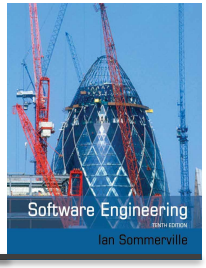


- Milestones are points in the schedule against which you can assess progress, for example, the handover of the system for testing.
- Deliverables are work products that are delivered to the customer, e.g. a requirements document for the system.

Activity bar chart

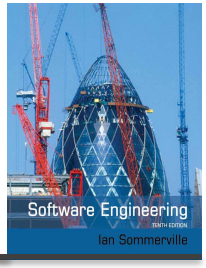


Agile planning



- Agile methods of software development are iterative approaches where the software is developed and delivered to customers in increments.
- Unlike plan-driven approaches, the functionality of these increments is not planned in advance but is decided during the development.
  - The decision on what to include in an increment depends on progress and on the customer's priorities.
- The customer's priorities and requirements change so it makes sense to have a flexible plan that can accommodate these changes.

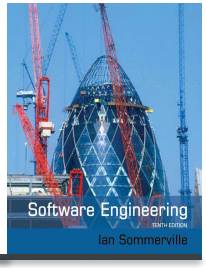
Story-based planning



- The planning game is based on **creating user stories** that reflect the features that should be included in the system.
- The project team read and discuss the stories and rank them in order of the amount of time they think it will take to implement the story.
- Stories are assigned ‘effort points’ reflecting their size and difficulty of implementation
- The number of effort points implemented per day is measured giving an estimate of the team’s ‘velocity’
- This allows the total effort required to implement the system to be estimated

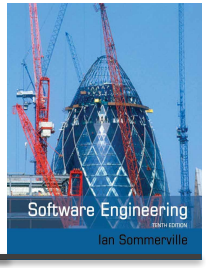
The planning game

Release and iteration planning



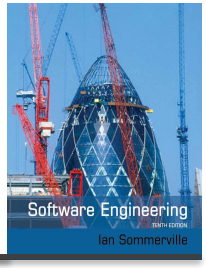
- Release planning involves selecting and refining the stories that will reflect the features to be implemented in a release of a system and the order in which the stories should be implemented.
- Stories to be implemented in each iteration are chosen, with the number of stories reflecting the time to deliver an iteration (usually 2 or 3 weeks).
- The team's velocity is used to guide the choice of stories so that they can be delivered within an iteration.

Task allocation



- During the task planning stage, the developers break down stories into development tasks.
  - A development task should take 4–16 hours.
  - All of the tasks that must be completed to implement all of the stories in that iteration are listed.
  - The individual developers then sign up for the specific tasks that they will implement.
- Benefits of this approach:
  - The whole team gets an overview of the tasks to be completed in an iteration.
  - Developers have a sense of ownership in these tasks and this is likely to motivate them to complete the task.

Software delivery



- A software increment is always delivered at the end of each project iteration.
- If the features to be included in the increment cannot be completed in the time allowed, the scope of the work is reduced.
- The delivery schedule is never extended.