

Week: COMP 2120 / COMP 6120
3 of 12
REQUIREMENTS

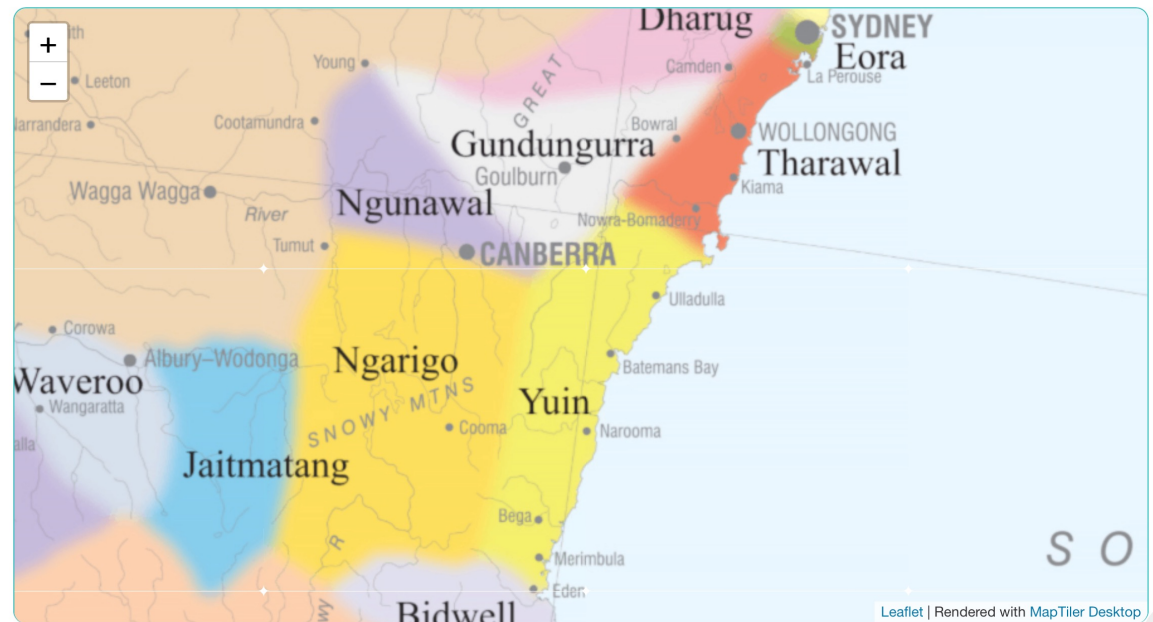
A/Prof Alex Potanin



ANU Acknowledgment of Country




“We acknowledge and celebrate the First Australians on whose traditional lands we meet, and pay our respect to the elders past and present.”



<https://aiatsis.gov.au/explore/map-indigenous-australia>





Gender Diversity in Computing: Coffee and Catch-Up

3PM Thursdays Semester 2

Hanna Neumann 3.41

Starting July 25



A chance to join fellow students in computing for a coffee and chat. A casual drop-in session each week, we will have a theme to stimulate discussion (which you are welcome to embrace or ignore). We hope for this to be a comfortable place for non-male students to meet and chat, interested allies are welcome too.

Today

- Self-Organising Teams
- Requirements
- The World and The Machine
- Quality Requirements
- Interviews
- Prototypes, Mockups, Stories
- Resolving Conflicts
- Risks



Edit the detailed description

Surprise me Upload →

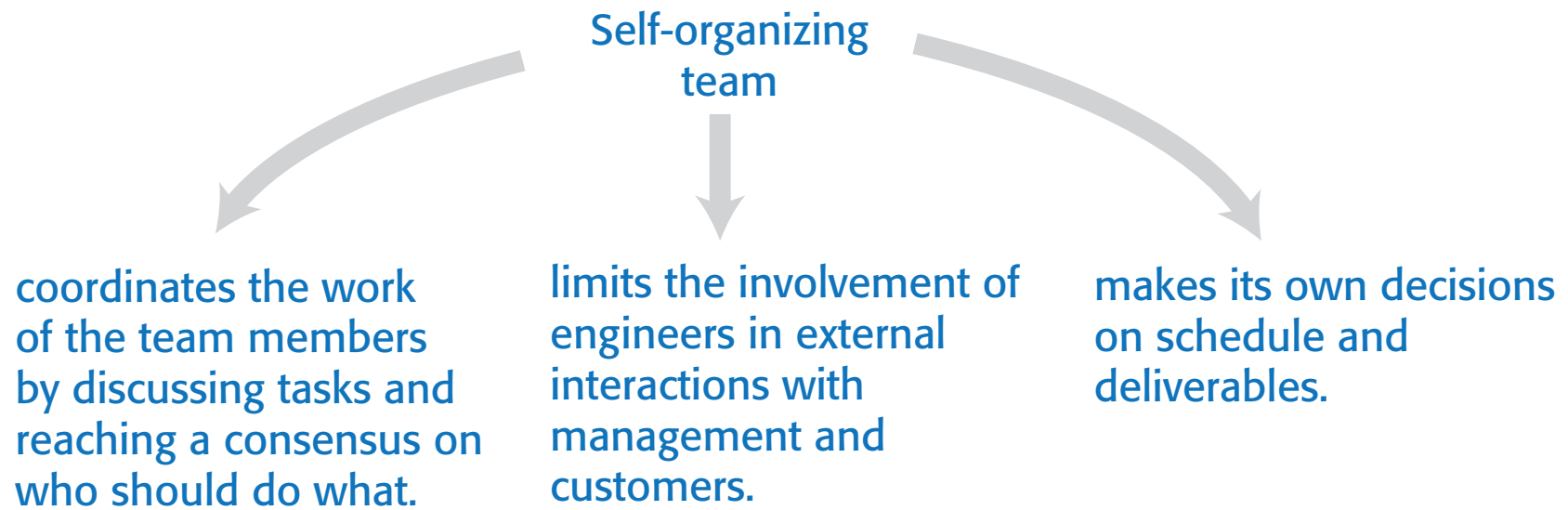
pencil drawing of teams without any words

Generate



Self-Organising Teams

Self-Organising Teams



Team Size and Composition



- The ideal Scrum team size is between 5 and 8 people.
 - Teams have to tackle diverse tasks and so usually require people with different skills, such as networking, user experience, database design and so on.
 - They usually involve people with different levels of experience.
 - A team of 5-8 people is large enough to be diverse yet small enough to communicate informally and effectively and to agree on the priorities of the team.
- The advantage of a self-organizing team is that it can be a cohesive team that can adapt to change.
 - Because the team rather than individuals take responsibility for the work, they can cope with people leaving and joining the team.
 - Good team communication means that team members inevitably learn something about each other's areas



Team Coordination



- The developers of Scrum assumed that teams would be co-located. They would work in the same room and could communicate informally.
 - Daily scrums mean that the team members know what's been done and what others are doing.
- However, the use of daily scrums as a coordination mechanism is based on two assumptions that are not always correct:
 - Scrum assumes that the team will be made up of full-time workers who share a workspace. In reality, team members may be part-time and may work in different places. For a student project team, the team members may take different classes at different times.
 - Scrum assumes that all team members can attend a morning meeting to coordinate the work for the day. However, some team members may work flexible hours (e.g. because of childcare responsibilities) or may work on several projects at the same time.



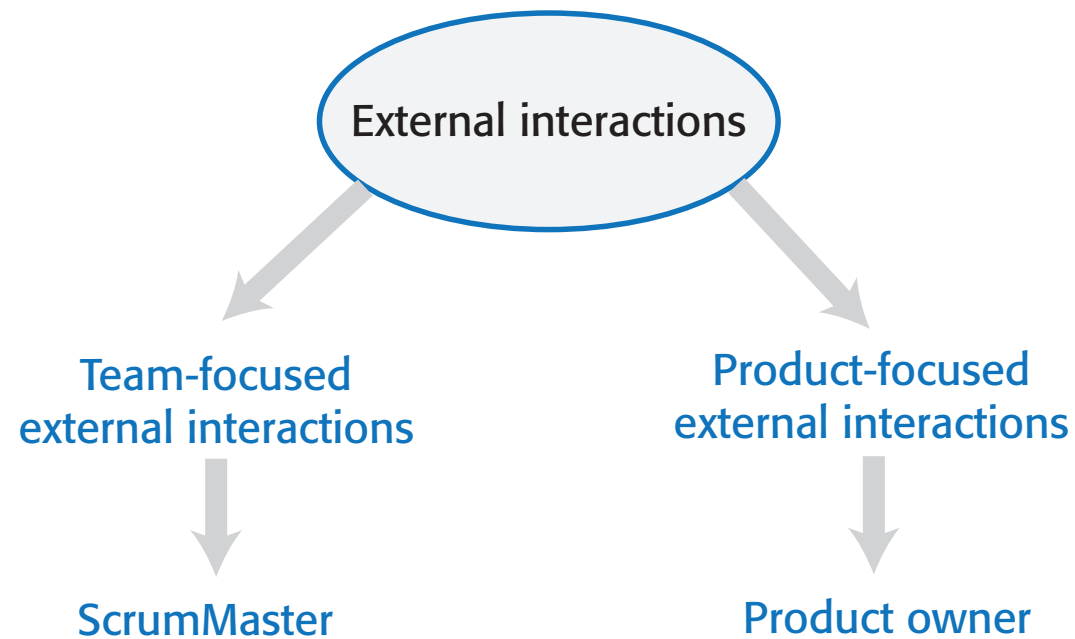
External Interactions



- External interactions are interactions that team members have with people outside of the team.
- In Scrum, the idea is that developers should focus on development and only the ScrumMaster and Product Owner should be involved in external interactions.
- The intention is that the team should be able to work on software development without external interference or distractions.



Managing External Interactions



Project Management



- In all but the smallest product development companies, there is a need for development teams to report on progress to company management.
- A self-organizing team has to appoint someone to take on these responsibilities.
 - Because of the need to maintain continuity of communication with people outside of the group, rotating these activities around team members is not a viable approach.
- The developers of Scrum did not envisage that the ScrumMaster should also have project management responsibilities.
 - In many companies, however, the ScrumMaster has to take on project management responsibilities.
 - They know the work going on and are in the best position to provide accurate information and project plans and progress.



Project Management Responsibilities




Software Products

- There are three factors that drive the design of software products
 - Business and consumer needs that are not met by current products
 - Dissatisfaction with existing business or consumer software products
 - Changes in technology that make completely new types of product possible
- In the early stage of product development, you are trying to understand, what product features would be useful to users, and what they like and dislike about the products that they use.



Poll Everywhere Time!

Join by Web [PollEv.com/potanin](https://poll-ev.com/potanin) Join by Text Send [potanin](https://poll-ev.com/potanin) to 22333 

Does a typical product need BOTH Product Owner and Project Manager?

Yes (A)

No (B)

I don't know (C)

Can you repeat the question? (D)



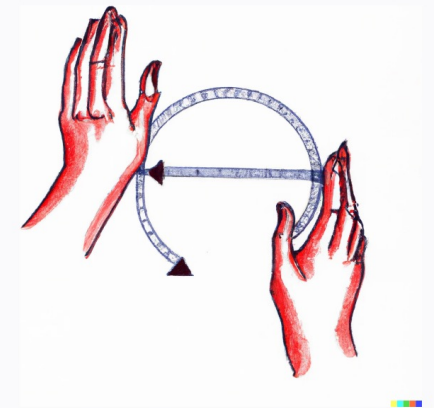
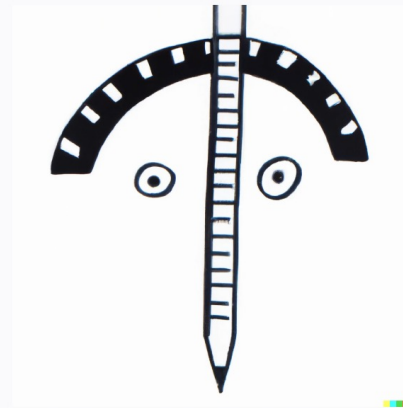
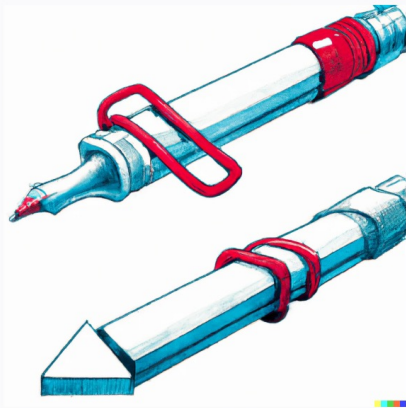


Edit the detailed description

Surprise me Upload →

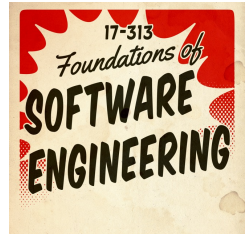
pencil drawing of requirements without words

Generate



Requirements

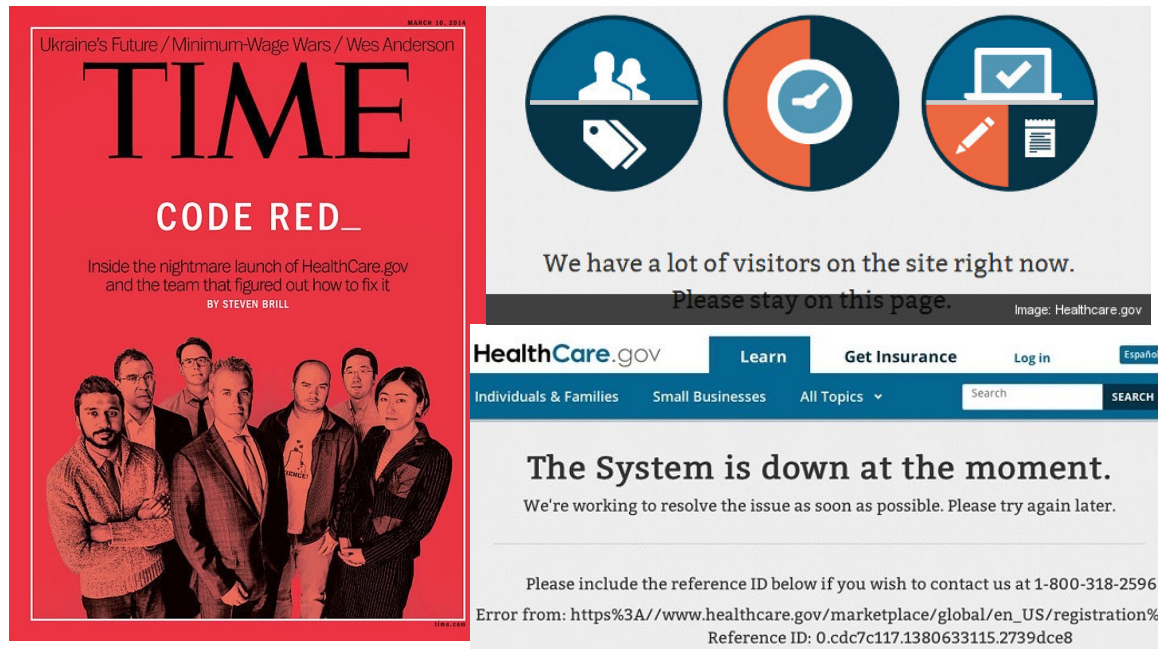
Overly simplified definition.



Requirements say what the system will do
(and not how it will do it)



Healthcare.gov



The image shows a side-by-side comparison of a magazine cover and a website error page. On the left is the March 16, 2014 cover of TIME magazine, featuring a red background and the headline "CODE RED_" about the launch of Healthcare.gov. On the right is a screenshot of the Healthcare.gov website during a system outage. The website header includes navigation links like "Learn", "Get Insurance", and "Log in". The main content area displays a large error message: "The System is down at the moment. We're working to resolve the issue as soon as possible. Please try again later." Below this, it provides a reference ID and contact information.

Ukraine's Future / Minimum-Wage Wars / Wes Anderson
MARCH 16, 2014
TIME
CODE RED_
Inside the nightmare launch of HealthCare.gov and the team that figured out how to fix it
BY STEVEN BRILL

17-313
Foundations of
SOFTWARE ENGINEERING

We have a lot of visitors on the site right now.
Please stay on this page.
Image: Healthcare.gov

HealthCare.gov Learn Get Insurance Log in Español
Individuals & Families Small Businesses All Topics Search SEARCH

The System is down at the moment.
We're working to resolve the issue as soon as possible. Please try again later.

Please include the reference ID below if you wish to contact us at 1-800-318-2596
Error from: [https%3A//www.healthcare.gov/marketplace/global/en_US/registration%](https%3A//www.healthcare.gov/marketplace/global/en_US/registration%20page)
Reference ID: 0.cdc7c117.1380633115.2739dce8



Fred Brooks on Requirements



- *The hardest single part of building a software system is deciding precisely **what to build**.*
- *No other part of the conceptual work is as difficult as establishing the detailed technical requirements ...*
- *No other part of the work so cripples the resulting system if done wrong.*
- *No other part is as difficult to rectify later.*
— Fred Brooks



A Problem That Stands the Test of Time...



A 1994 survey of 8000 projects at 350 companies found: 31% of projects canceled before completed; 9% of projects delivered on time, within budget in large companies, 16% in small companies.

- Similar results reported since.

Causes:

1. Incomplete requirements (13.1%)
2. Lack of user involvement (12.4%)
3. Lack of resources (10.6%)
4. Unrealistic expectations (9.9%)
5. Lack of executive support (9.3%)
6. Changing requirements and specifications (8.7%)
7. Lack of planning (8.1%)
8. System no longer needed (7.5%)



Why Is This So Hard???



Communication Problem



Goal: figure out what should be built.

Express those ideas so that the correct thing is built.



Overall Problems

- Involved subproblems?
- Required functionality?
- Nice to have functionality?
- Expected qualities?
- How fast to deliver at what quality for what price?

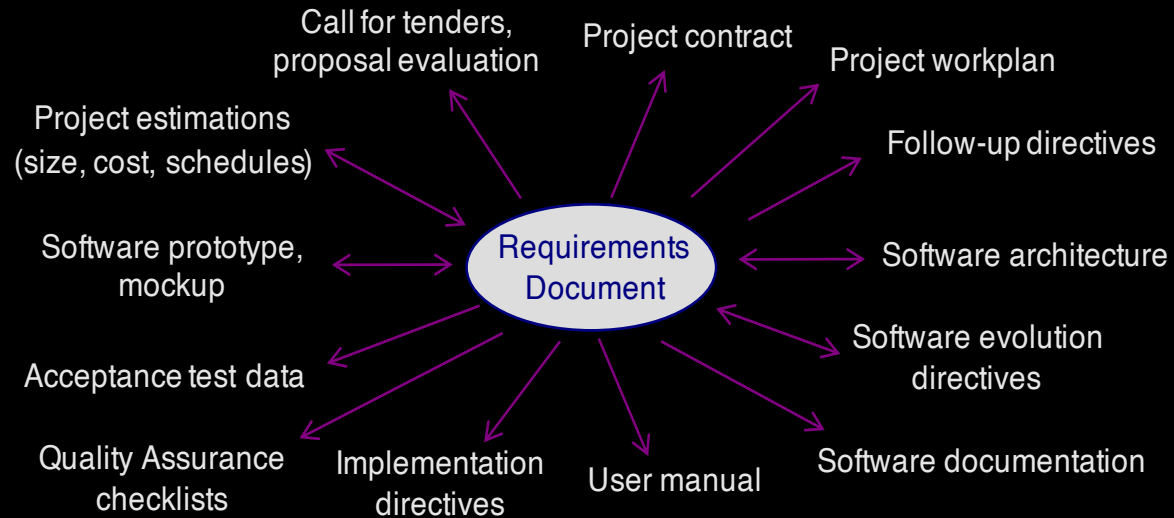


Tidbinbilla Demo/Exercise on What Not To Do

- Tidbinbilla needs a new ticket system to sell tickets online, have paperless tickets (on your smart phone), assume that now only paper tickets and requires a call (slow). ***I will be the Developer Company.***
- I suggest to use a previously built system for buses (Transport Canberra) with minor modifications **Denial by Prior Knowledge**
- Let's say you ask for a special feature (e.g. discounts for school holidays) and I will know exactly what tables in DB to modify and I won't really listen to what exactly you want **Denial by Hacking**
- Previously there were free tickets for people in need and you ask for something like that (or a lottery or special payment types) but I will downplay the need: they should just get Visa Gift Card so my system works **Denial by Abstraction**
- You complain that it's annoying to check that everybody is out at the end of the day, so I recommend we can just "count this at the gate" **Denial by Vagueness**



Requirements in software projects



Less Simplified Definition: Online Shopping



- Stories: Scenarios, Use Cases, and user stories

“After the customer submits the purchase information and the payment has been received, the order is fulfilled and shipped to the customer’s shipping address.”

- Optative statements

*The system **shall** notify clients about their shipping status*

- Domain Properties and Assumptions

Every product has a unique product code

Payments will be received after authorization



What is requirements engineering?




- Knowledge **acquisition** – how to capture relevant detail about a system?
 - Is the knowledge complete and consistent?
- Knowledge **representation** – once captured, how do we express it most effectively?
 - Express it for whom?
 - Is it received consistently by different people?
- You may sometimes see a distinction between the requirements *definition* and the requirements *specification*.



Poll Everywhere Time!

Join by Web PollEv.com/potanin



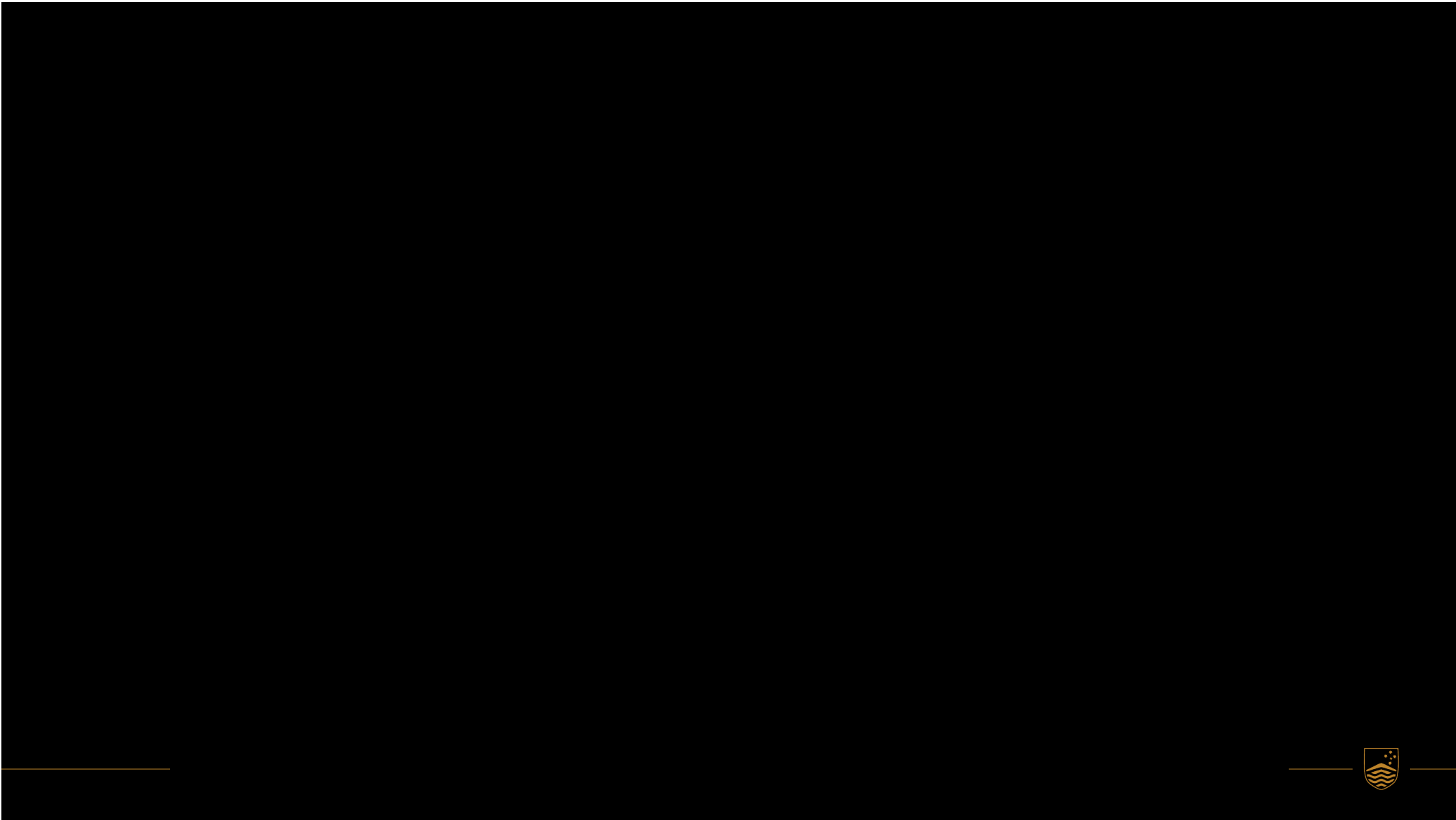
Which Requirements Denial Sin are you most likely to commit (rank from most likely to least likely)?

Denial by

- Prior Knowledge
- Hacking
- Abstraction
- Vagueness

SEE MORE ▾





Edit the detailed description

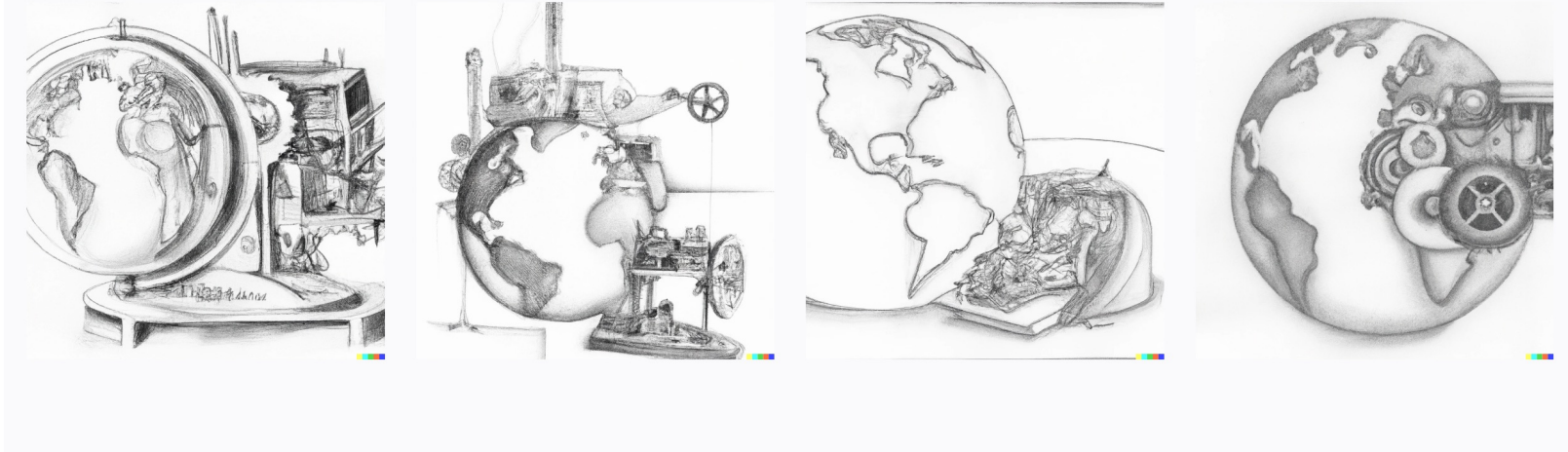
Surprise me

Upload



black and white pencil drawing of the world and the machine without any words

Generate



The World and The Machine (Quality Requirements)



Functional Requirements and Implementation Bias



Requirements say what the system will do (and not **how** it will do it).

Why not “how”?



Environment and the Machine



Four Dark Corners of Requirements Engineering

PAMELA ZAVE
AT&T Research
and
MICHAEL JACKSON
AT&T Research and MAJ Consulting Limited

Research in requirements engineering has produced an extensive body of knowledge, but there are four areas in which the foundation of the discipline seems weak or obscure. This article shines some light in the "four dark corners," exposing problems and proposing solutions. We show that all descriptions involved in requirements engineering should be descriptions of the environment. We show that certain control information is necessary for sound requirements engineering, and we explain the close association between domain knowledge and refinement of requirements. Together these conclusions explain the precise nature of requirements, specifications, and domain knowledge, as well as the precise nature of the relationships among them. They establish minimum standards for what information should be represented in a requirements language. They also make it possible to determine exactly what it means for requirements engineering to be successfully completed.

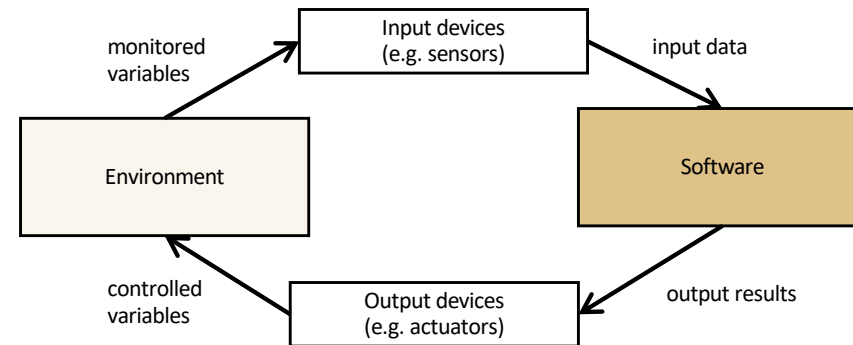
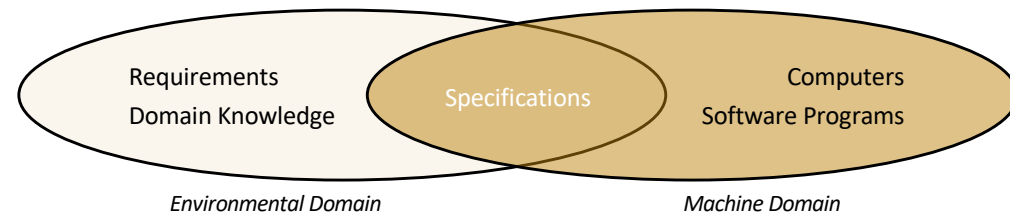
Categories and Subject Descriptors: D.2.1 [Software Engineering]: Requirements/Specifications—methodologies

General Terms: Theory, Verification

Additional Key Words and Phrases: Control of actions, domain knowledge, implementation bias, refinement of requirements

1. INTRODUCTION

There was a time when the epigram "requirements say what the system will do and not how it will do it" summarized all of requirements engineering. That time is long past. Research in requirements engineering has now produced a body of knowledge including terminology, methods, languages, tools, and issues acknowledged to be critical.



Pamela Zave & Michael Jackson, "Four Dark Corners of Requirements Engineering," *ACM Transactions on Software Engineering and Methodology*, 6(1): 1-30, 1997.





**Actions of an ATM
customer:**

withdrawal-request(a, m)

**Properties of the
environment:**

balance(b, p)

**Actions of an ATM
machine:**

withdrawal-payout(a, m)

**Properties of the
machine:**

expected-balance(b, p)

What other **models of the
world** do machines maintain?

Domain Knowledge



- Refinement is the act of translating requirements into specifications (bridging the gap!)
- Requirements: desired behavior (effect on the environment) to be realized by the proposed system.
- Assumptions or domain knowledge: existing behavior that is unchanged by the proposed system.
 - Conditions under which the system is guaranteed to operate correctly.
 - How the environment will behave in response to the system's outputs.



Some gaps must remain...



- Unshared actions cannot be accurately expressed in the machine
 - People can jump over gates (enter without unlocking)
 - People can steal or misplace inventory
- Future requirements are also not directly implementable
 - Phone system: “After all digits have been dialed, do *ring-back*, *busy-tone* or *error-tone*.”
 - ...how do you know the user is done dialing?



Avoiding Implementation Bias



- Requirements describe what is observable at the environment-machine interface.
- Indicative mood describes the environment (as-is)
- Optative mood to describe the environment with the machine (to-be).



This can be subtle...



- “The dictionary shall be stored in a hash table” vs. “the software shall respond to requests within 5 seconds.”
- **Instead of “what” vs. “how”, ask “is this requirement only a property of the machine domain?”**
- Or is there some application domain phenomenon that justifies it?



Functional Requirements



- What the machine should do
 - Input
 - Output
 - Interface
 - Response to events
- Criteria
 - Completeness: All requirements are documented
 - Consistency: No conflicts between requirements
 - Precision: No ambiguity in requirements



Quality/Nonfunctional Requirements



- Specify not the functionality of the system, but the quality with which it delivers that functionality.
- Can be more critical than functional requirements
 - Can work around missing functionality
 - Low-quality system may be unusable
- Examples?



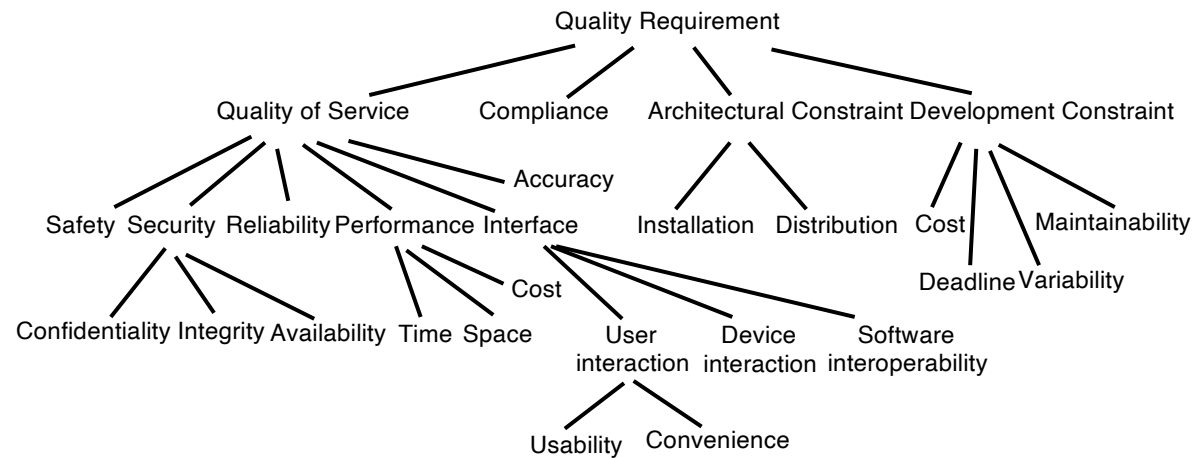
Here's the thing ...



- Who is going to ask for a slow, inefficient, unmaintainable system?
- A better way to think about quality requirements is as *design criteria to help choose between alternative implementations*.
- Question becomes: to what extent must a product satisfy these requirements to be acceptable?



Example: Selling Videos on the Web



Expressing Quality Requirements



- Requirements serve as contracts: they should be testable/falsifiable.
- Informal goal: a general intention, such as ease of use.
 - May still be helpful to developers as they convey the intentions of the system users.
- Verifiable non-functional requirement: A statement using some measure that can be objectively tested.



Examples



- **Confidentiality requirement:** A non-staff patron may never know which books have been borrowed by others.
- **Privacy requirement:** The diary constraints of a participant may never be disclosed to other invited participants without their consent.
- **Integrity requirement:** The return of book copies shall be encoded correctly and by library staff only.
- **Availability requirement:** A blacklist of bad patrons shall be made available at any time to library staff. Information about train positions shall be available at any time to the vital station computer.



Examples

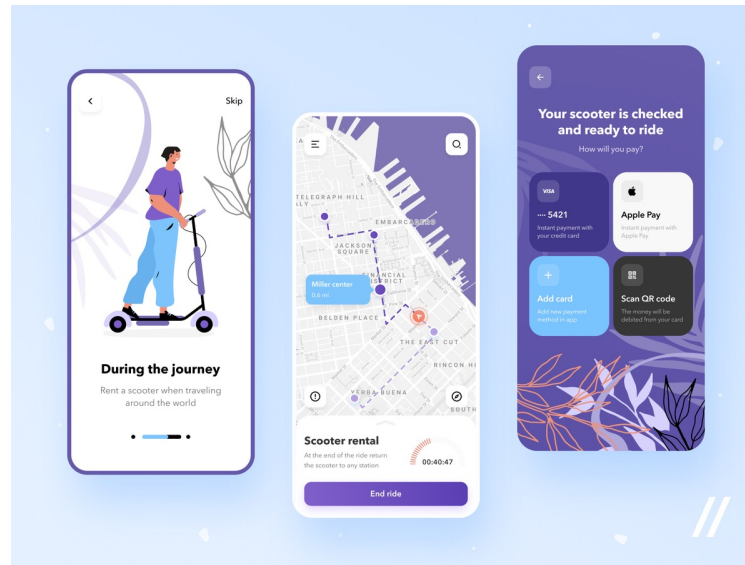


- Informal goal: “the system should be easy to use by experienced controllers and should be organized such that user errors are minimized.”
- **Verifiable non-functional requirement:** “Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day, on average.”



Exercise: back to simple

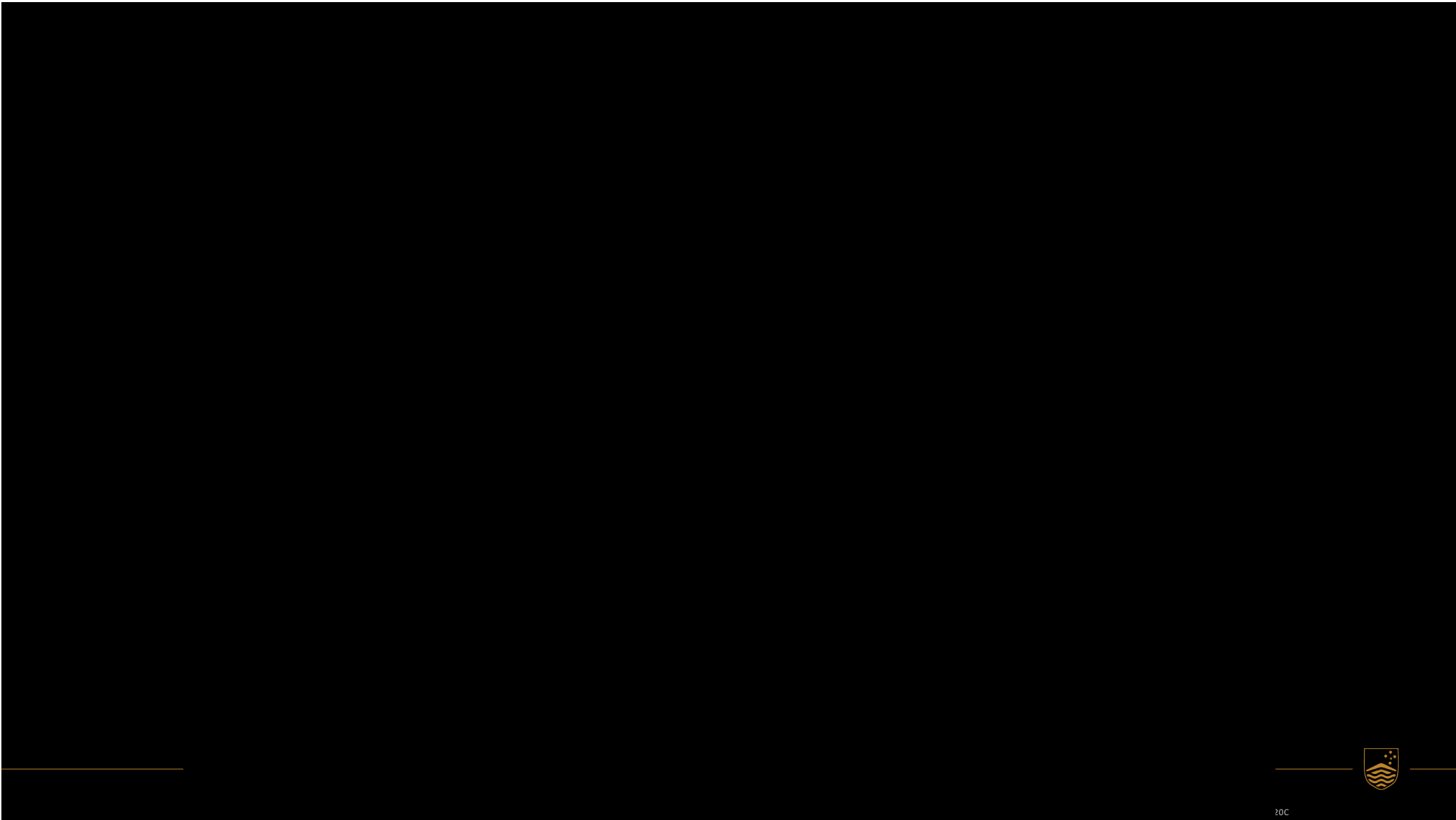
- Let's write some quality requirements!
- Try to write an informal goal, and then turn it into a verifiable non-functional requirement.



Poll Everywhere Time!

The screenshot shows a mobile interface for a PollEv poll. At the top, it says "Join by Web PollEv.com/potantin Join by Text Send **potantin** to **22333**". A QR code is in the top right corner. The poll question is "Write a Quality Requirement for the Scooter App in the Format Of: 'Property Description : How to Measure It'", with a heart icon and the number "0" to its right. Below the question, there are two columns. The left column contains "Join by Web PollEv.com/potantin" and "Join by Text Send **potantin** to **22333**". The right column contains "Join by QR code" and "Scan with your camera app" above a large QR code.





Edit the detailed description

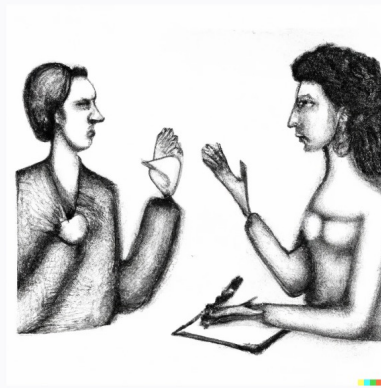
Surprise me

Upload



black and white pencil drawing of the interviews without any words

Generate



Interviews



Typical Steps

- Identify stakeholders
- Understand the domain
 - Analyze artifacts, interact with stakeholders
- Discover the real needs
 - Interview stakeholders
- Explore alternatives to address needs



Question

- Who is the system for?
- Stakeholders:
 - End users
 - System administrators
 - Engineers maintaining the system
 - Business managers
 - ...who else?



Learning goals

- Define and identify stakeholders.
- Demonstrate basic proficiency in executing effective requirements interviews.
- Evaluate risk for a product



Interviews



Interview Follow-up

- Observations?
 - Anything surprising? Unexpected?
 - Confirmations of existing ideas?
 - Generalizable knowledge?



Interview Tradeoffs



- **Strengths**

- What stakeholders do, feel, prefer
- How they interact with the system
- Challenges with current systems

- **Weaknesses**

- Subjective, inconsistencies
- Capturing domain knowledge
- Familiarity
- Technical subtlety
- Organizational issues, such as politics
- Hinges on interviewer skill

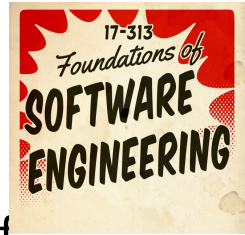


Interview Process

- Identify stakeholder of interest and target information to be gathered.
- Conduct interview.
 - (structured/unstructured, individual/group)
- Record + transcribe interview
- Report important findings.
- Check validity of report with interviewee.



Example: Identifying Problems



What problems do you run into in your day-to-day work? Is there a standard way of solving it, or do you have a workaround?

Why is this a problem? How do you solve the problem today? How would you ideally like to solve the problem?

Keep asking follow-up questions (“What else is a problem for you?”, “Are there other things that give you trouble?”) for as long as the interviewee has more problems to describe.



Example: Identifying Problems



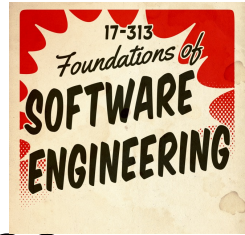
So, as I understand it, you are experiencing the following problems/needs (describe the interviewee's problems and needs in your own words – often you will discover that you do not share the same image. It is very very common to not understand each other even if at first you think you do).

Just to confirm, have I correctly understood the problems you have with the current solution?

Are there any other problems you're experiencing? If so, what are they?



Capturing v. Synthesizing



Engineers acquire requirements from many sources

- Elicit from stakeholders

- Extract from policies or other documentation

- Synthesize from above + estimation and invention

Because stakeholders do not always know what they want, engineers must...

- Be faithful to stakeholder needs and expectations

- Anticipate additional needs and risks

- Validate that “additional needs” are necessary or desired



Interview Advice



Get basic facts about the interviewee before (role, responsibilities, ...)

Review interview questions before interview

Begin concretely with specific questions, proposals; work through prototype or scenario

Relate to current system, if applicable.

Be open-minded; explore additional issues that arise naturally, but stay focused on the system.

Contrast with current system/alternatives. Explore conflicts and priorities

Plan for follow-up questions



Bonus: Guidelines for effective interviews



Identify the right interviewee sample for full coverage of issues

different responsibilities, expertise, tasks, exposure to problems

Come prepared, to focus on right issue at right time

background study first

predesign a sequence of questions for this interviewee

Centre the interview on the interviewee's work & concerns

Keep control over the interview

Make the interviewee feel comfortable

Start: break ice, provide motivation, ask easy questions

Consider the person too, not only the role

Do always appear as a trustworthy partner



Bonus: Guidelines for effective interviews



Be focused, keep open-ended questions for the end

Be open-minded, flexible in case of unexpected answers

Ask why-questions without being offending

Avoid certain types of questions ...

- opinion or biased

- affirmative

- obvious or impossible answer for this interviewee

Edit & structure interview transcripts while still fresh in mind

- including personal reactions, attitudes, etc


Keep interviewee in the loop

- co-review interview transcript for validation & refinement



Poll Everywhere Time!

Join by Web PollEv.com/potanin Join by Text Send [potanin](https://poll-ev.com/potanin) to 22333



Have you interviewed anyone before? 0

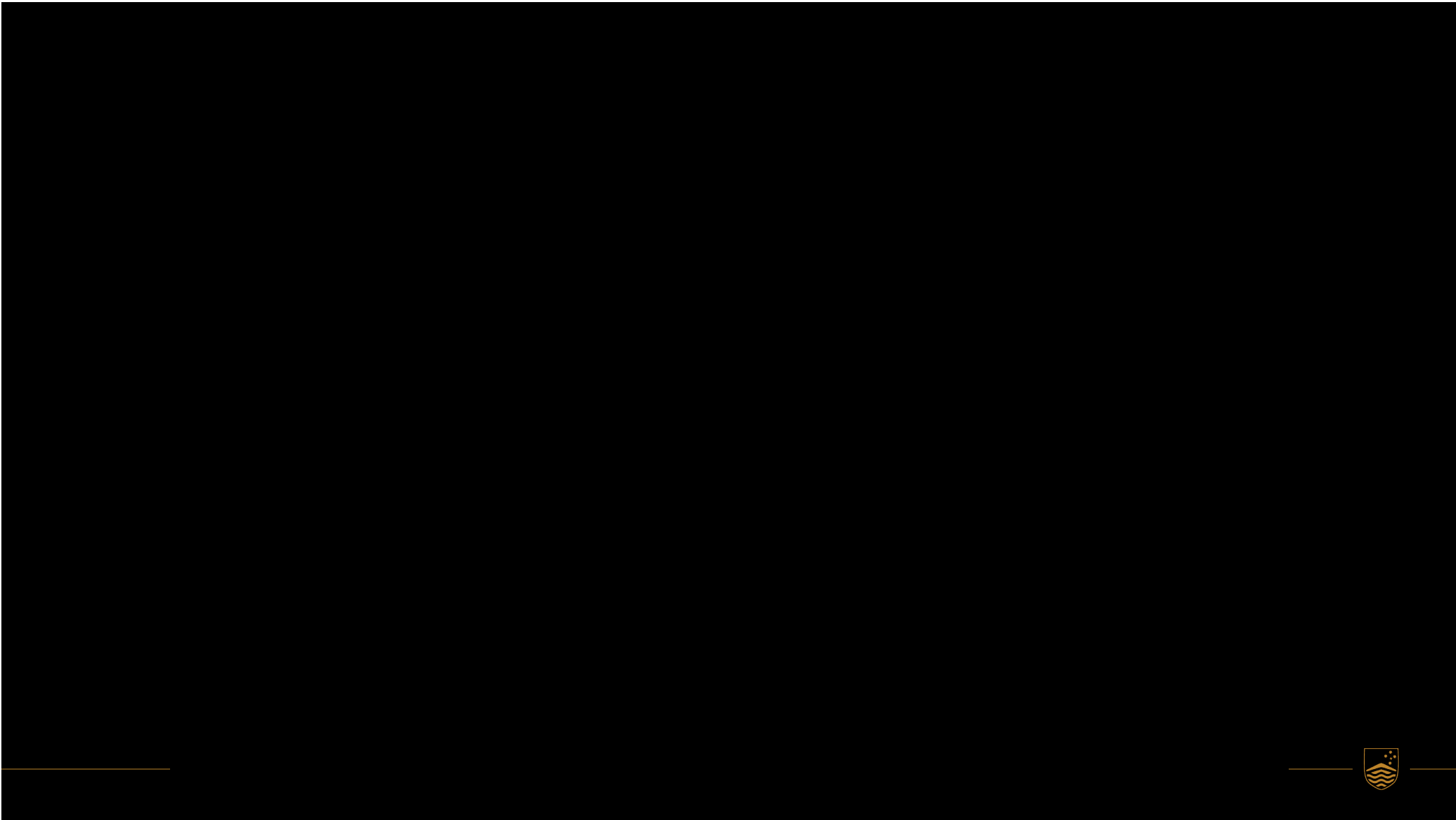
Yes **(A)**

No **(B)**

I don't know **(C)**

Can you repeat the question? **(D)**





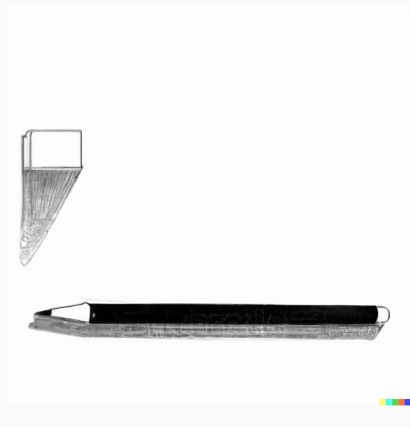
Edit the detailed description

Surprise me

Upload →

black and white pencil drawing of a mock up without any words

Generate



Prototypes, Mock-ups, Stories

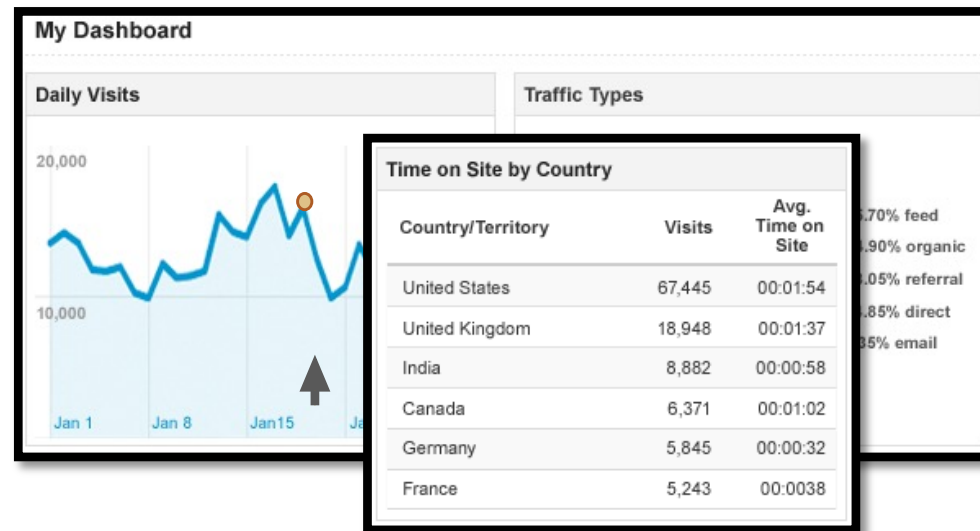
Prototypes, Mock-ups, Stories



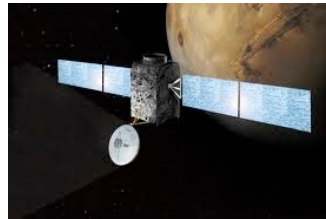
- Why? How to use?
- Stakeholders:
 - don't always know what they want or how to articulate it, or how much things cost.
 - have domain knowledge, may use jargon, or may leave out “obvious” requirements that aren't obvious to a non-expert.
 - can be hard to pin down
 - Distributed, difficult to access, have hidden needs
 - External to the system
- Risk: Missing or hidden stakeholders, “requirements—delay—surprise!”
- Risk: unidentified/unhandled **conflicts**.



High- vs low- fidelity mockups



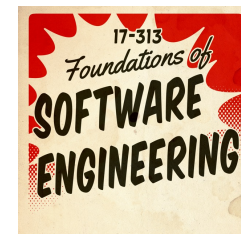
Storyboarding and scenarios



Story

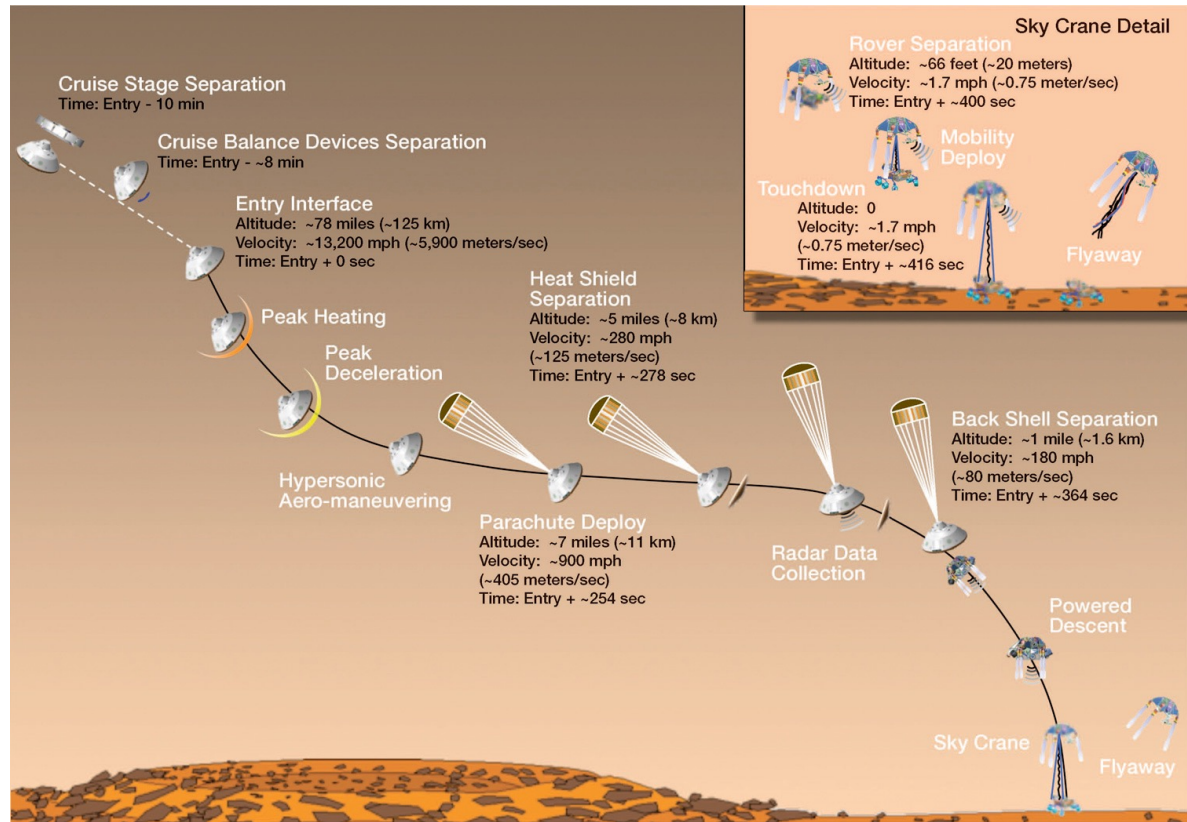
- Who the players are
- What happens to them
- How it happens through specific episode
- Why this happens
- What if such and such an event occurs
- What could go wrong as a consequence







- Storyboards illustrate scenarios: a typical sequence of interaction among system components that meets an implicit objective.
 - Storyboards explicitly cover at least who, what, and how.
- Different types:
 - Positive vs negative (should and should not happen)
 - Normal vs abnormal
- As part of elicitation:
 - Learn about current or proposed system by walking through real-life or hypothetical sequences
 - Can ask specific questions
 - Elicit the underlying objectives, generalize into models of desired behaviors.
 - Identify and resolve conflicts
- Pluses: Concrete, support narrative description
- Minuses: inherently partial.





Poll Everywhere Time!

Join by Web PollEv.com/potanin Join by Text Send [potanin](https://poll-ev.com/potanin) to 22333 

Are "Stories" Here Different from "User Stories" Before? 

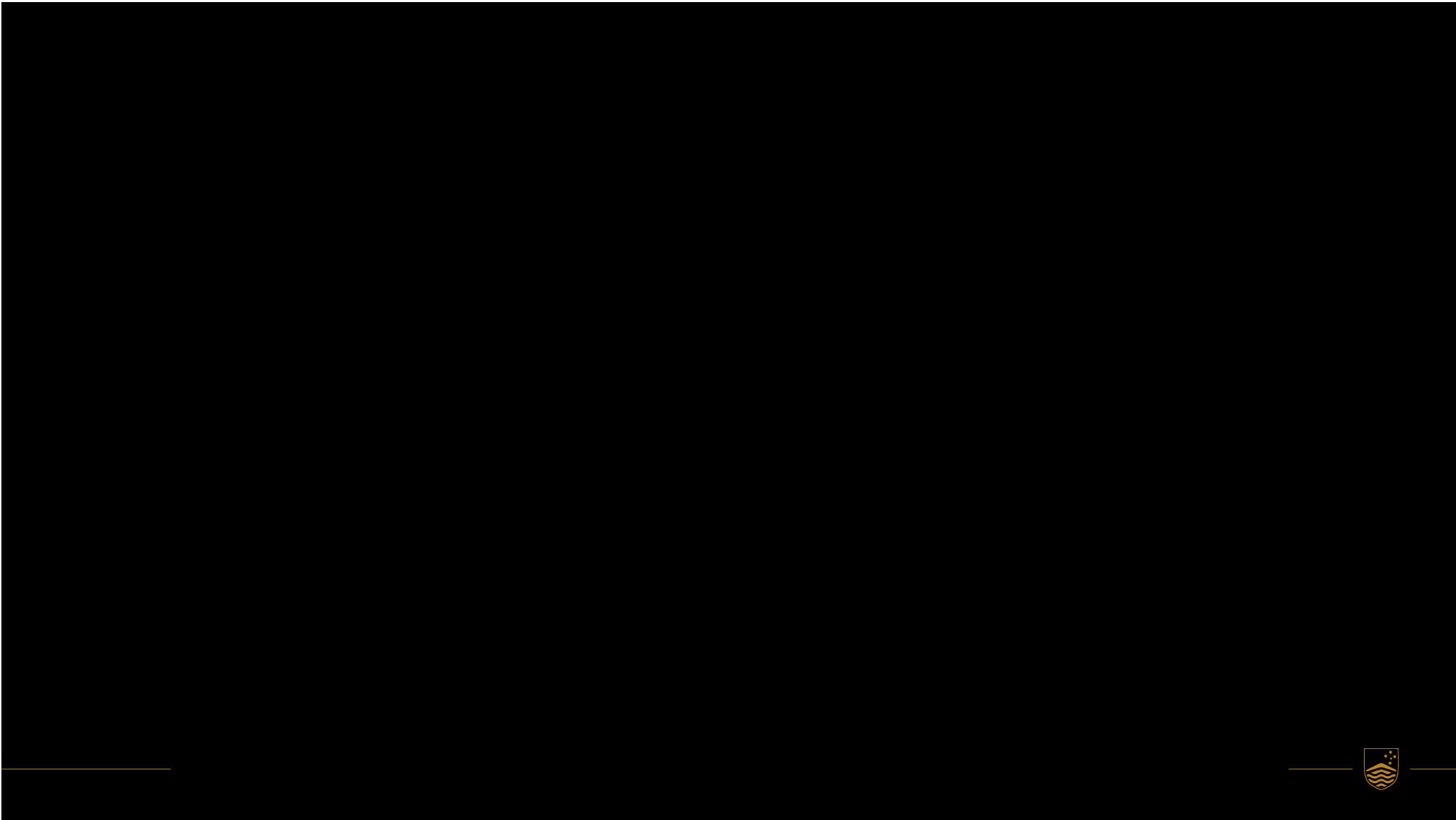
Yes **(A)**

No **(B)**

Maybe **(C)**

I am asleep right now **(D)**





Edit the detailed description

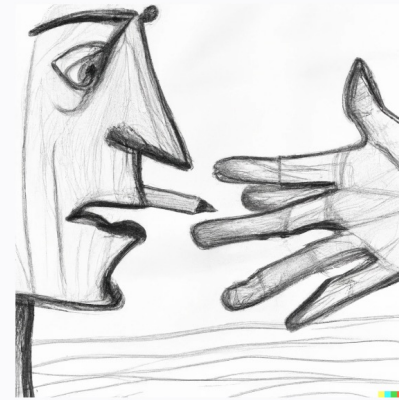
Surprise me

Upload



black and white pencil drawing of the conflict without any words

Generate



Resolving Conflicts and Risks



Types of inconsistency



- Terminology clash: same concept named differently in different statements
 - e.g. library management: “borrower” vs. “patron”
- Designation clash: same name for different concepts in different statements
 - e.g. “user” for “library user” vs. “library software user”
- Structure clash: same concept structured differently in different statements
 - e.g. “latest return date” as time point (e.g. Fri 5pm)
 - vs. time interval (e.g. Friday)



Types of inconsistency



- Strong conflict: statements not satisfiable together
 - e.g. “participant constraints may not be disclosed to anyone else” vs. “the meeting initiator should know participant constraints”
- Weak conflict (divergence): statements not satisfiable together under some boundary condition
 - “patrons shall return borrowed copies within X weeks” vs “patrons shall keep borrowed copies as long as needed” contradict only if “needed > x weeks”



Handling inconsistencies



- Terminology, designation, structure: Build glossary
- Weak, strong conflicts: Negotiation required
 - Cause: different objectives of stakeholders => resolve outside of requirements
 - Cause: quality tradeoffs => explore preferences



Requirements Traceability

- Keep connections between requirements
- What follows from what



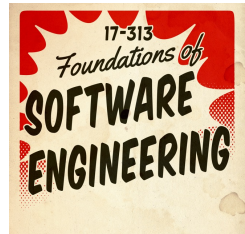
Requirements Prioritisation

- Cost, time, and other limits
- Dependencies among requirements
- Nice to have

- Strategies to base on value contribution



Risks



 **Tony Webster** 
@webster Follow 

I appreciate the honesty.

Pick a password

Don't reuse your bank password, we didn't spend a lot on security for this app.
At least 6 characters

Continue

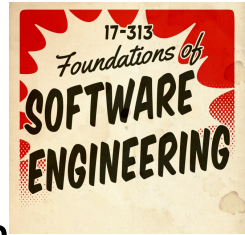
8:20 PM - 15 Sep 2018

5,868 Retweets 15,672 Likes 

 58  5.9K  16K 



What are risks?



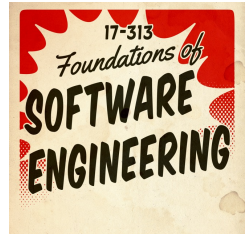
- A **risk** is an uncertain factor that may result in a loss of satisfaction of a corresponding objective

For example...

- System delivers a radiation overdose to patients (Therac-25, Theratron-780)
- Medication administration record (MAR) knockout
- Premier Election Solutions vote-dropping “glitch”



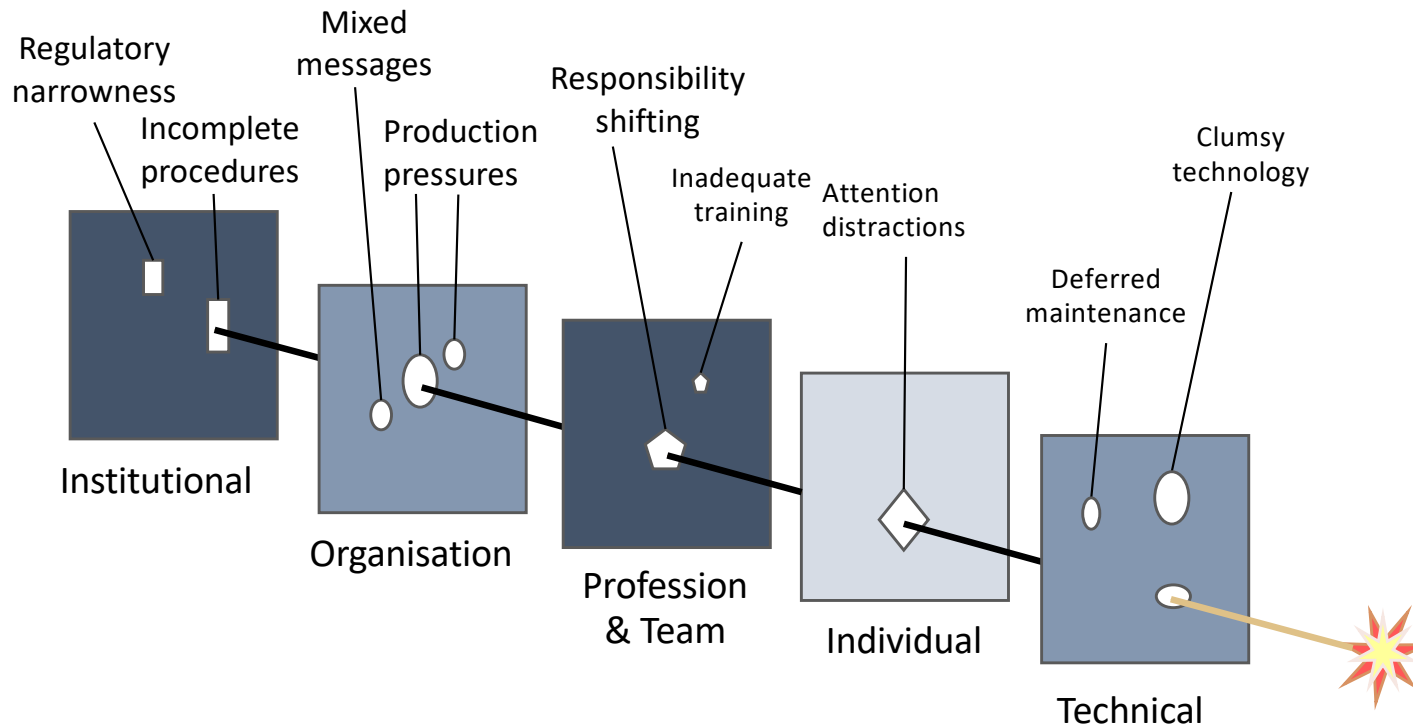
How to assess the level of risk?



- Risks consist of multiple parts:
 - Likelihood of failure
 - Negative consequences or impact of failure
 - Causal agent and weakness (in advanced models)
- Risk = Likelihood x Impact



Swiss Cheese Model



Modified from Reason, 1999, by R.I. Crook



Aviation failure impact categories



- No effect – failure has no impact on safety, aircraft operation, or crew workload
- Minor – failure is noticeable, causing passenger inconvenience or flight plan change
- Major – failure is significant, causing passenger discomfort and slight workload increase
- Hazardous – high workload, serious or fatal injuries
- Catastrophic – loss of critical function to safely fly and land

DO-178b, Software Considerations in Airborne Systems and Equipment Certification, RTCA, 1992



Risk assessment matrix

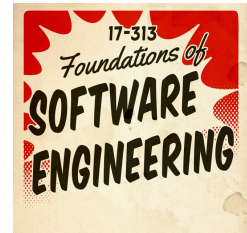


TABLE III. Risk assessment matrix

RISK ASSESSMENT MATRIX				
SEVERITY PROBABILITY	Catastrophic (1)	Critical (2)	Marginal (3)	Negligible (4)
Frequent (A)	High	High	Serious	Medium
Probable (B)	High	High	Serious	Medium
Occasional (C)	High	Serious	Medium	Low
Remote (D)	Serious	Medium	Medium	Low
Improbable (E)	Medium	Medium	Medium	Low
Eliminated (F)	Eliminated			

MIL-STD-882E

<https://www.system-safety.org/Documents/MIL-STD-882E.pdf>



DECIDE Model

Detect that the action necessary

Estimate the significance of the action

Choose a desirable outcome

Identify actions needed in order to achieve the chosen option

Do the necessary action to achieve change

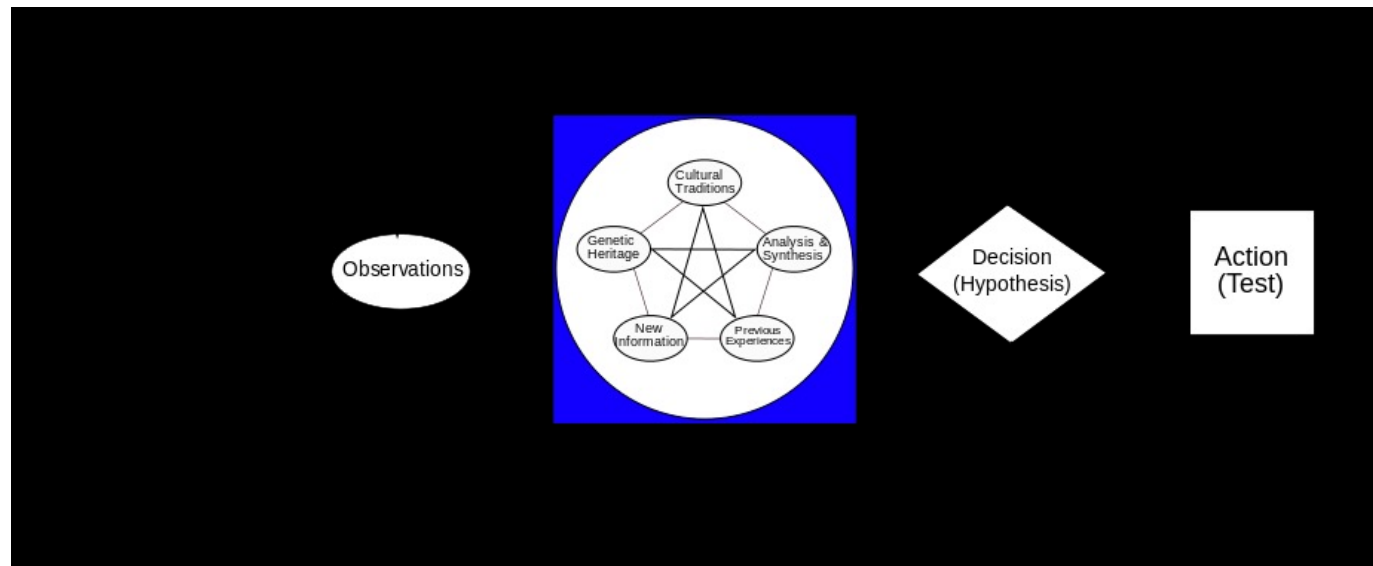
Evaluate the effects of the action



https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/media/FAA-H-8083-2.pdf



OODA Loop



By Patrick Edwin Moran - Own work, CC BY 3.0,
<https://commons.wikimedia.org/w/index.php?curid=3904554>



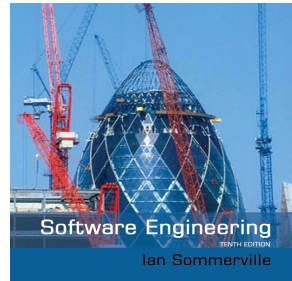
Summary



- Many solicitation strategies, including document analysis, interviews, and ethnography
- Do not underestimate the challenge of interviews
- Resolving conflicts
- Using prototypes to enhance discussions and decision making
- Many documentation strategies; our focus is on *user stories*



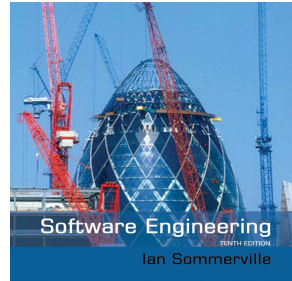
Configuration management



- Configuration management is the name given to the general process of managing a changing software system.
- The aim of configuration management is to support the system integration process so that all developers can access the project code and documents in a controlled way, find out what changes have been made, and compile and link components to create a system.



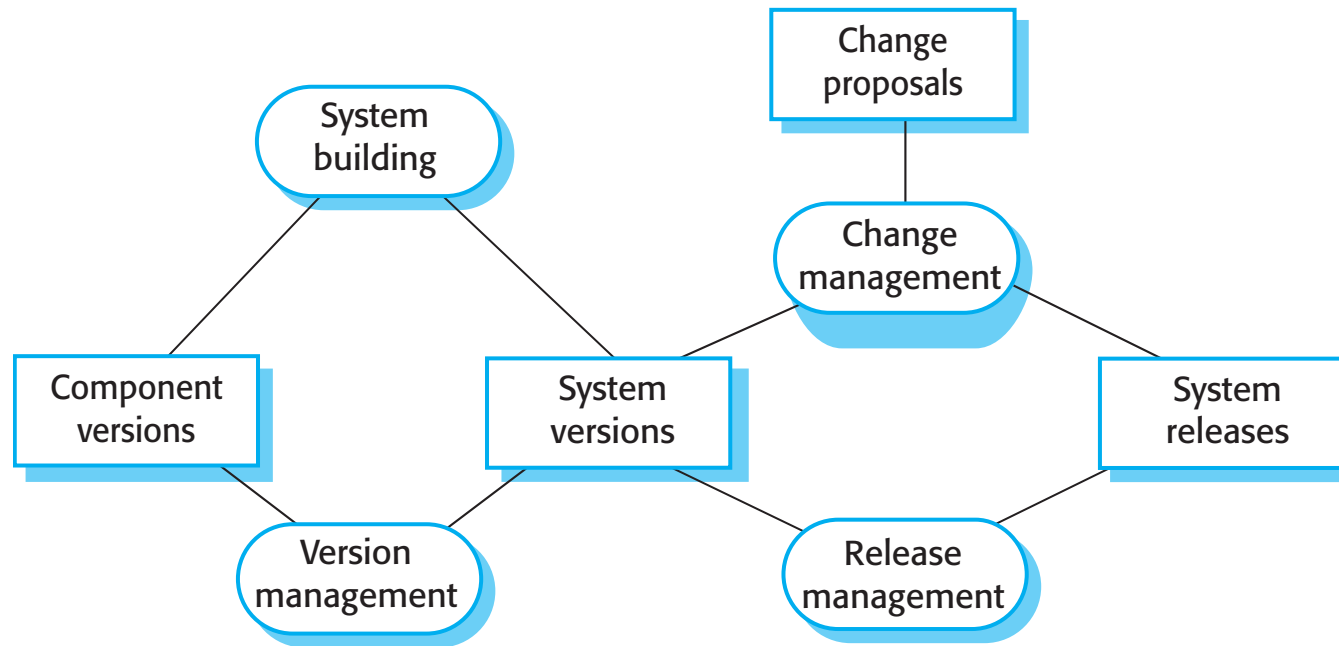
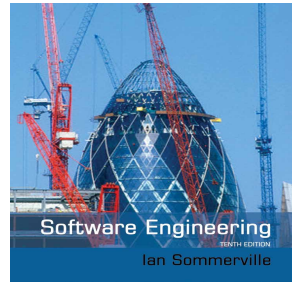
Configuration management activities



- **Version management,**
 - where support is provided to keep track of the different versions of software components. Version management systems include facilities to coordinate development by several programmers.
- **System integration,**
 - where support is provided to help developers define what versions of components are used to create each version of a system. This description is then used to build a system automatically by compiling and linking the required components.
- **Problem tracking,**
 - where support is provided to allow users to report bugs and other problems, and to allow all developers to see who is working on these problems and when they are fixed.



Configuration management tool interaction



Poll Everywhere Time!

The screenshot shows a mobile interface for a PollEv poll. At the top, there are two options to join: 'Join by Web' with the link PollEv.com/potantin and 'Join by Text' with the instruction 'Send **potantin** to **22333**'. A QR code is located in the top right corner. The main question is 'What Are The Four Words in OODA?' with a heart icon and the number '0' to its right. Below the question, the interface is split into two columns. The left column contains the same 'Join by Web' link and 'Join by Text' instruction. The right column is titled 'Join by QR code' and 'Scan with your camera app', featuring a large QR code.



<https://www.youtube.com/watch?v=NpnQ2oIRgB0>



BBG HD

ANU SCHOOL OF COMPUTING | COMP 2120 / COMP 6120 | WEEK 3 OF 12: REQUIREMENTS

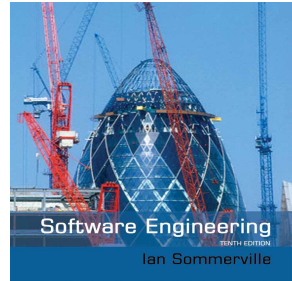




**HISTORICAL DETOUR INTO UML (OR BACK INTO THE 1990'S)
ONLY IF TIME ALLOWS!**



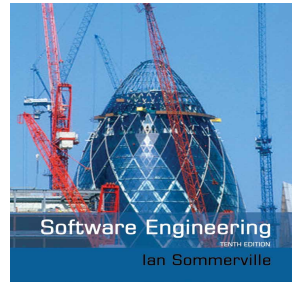
Design and implementation



- Software design and implementation is the stage in the software engineering process at which an executable software system is developed.
- Software design and implementation activities are invariably inter-leaved.
 - Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements.
 - Implementation is the process of realizing the design as a program.



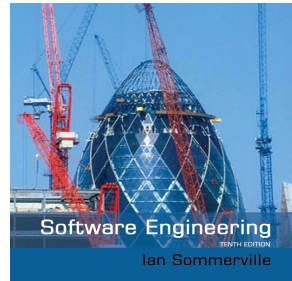
Build or buy



- In a wide range of domains, it is now possible to buy commercial off-the-shelf systems (COTS) that can be adapted and tailored to the users' requirements.
 - For example, if you want to implement a medical records system, you can buy a package that is already used in hospitals. It can be cheaper and faster to use this approach rather than developing a system in a conventional programming language.
- When you develop an application in this way, the design process becomes concerned with how to use the configuration features of that system to deliver the system requirements.



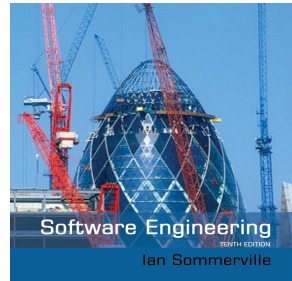
An object-oriented design process



- Structured object-oriented design processes involve developing a number of different system models.
- They require a lot of effort for development and maintenance of these models and, for small systems, this may not be cost-effective.
- However, for large systems developed by different groups design models are an important communication mechanism.



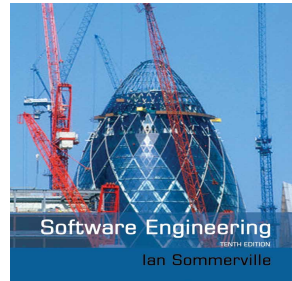
Process stages



- There are a variety of different object-oriented design processes that depend on the organization using the process.
- Common activities in these processes include:
 - Define the context and modes of use of the system;
 - Design the system architecture;
 - Identify the principal system objects;
 - Develop design models;
 - Specify object interfaces.
- Process illustrated here using a design for a wilderness weather station.



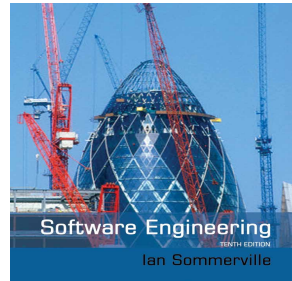
System context and interactions



- Understanding the relationships between the software that is being designed and its external environment is essential for deciding how to provide the required system functionality and how to structure the system to communicate with its environment.
- Understanding of the context also lets you establish the boundaries of the system. Setting the system boundaries helps you decide what features are implemented in the system being designed and what features are in other associated systems.



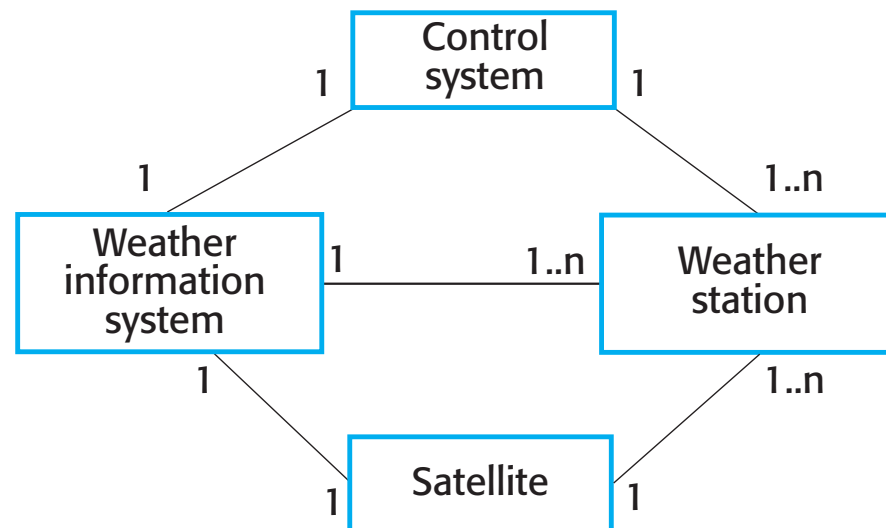
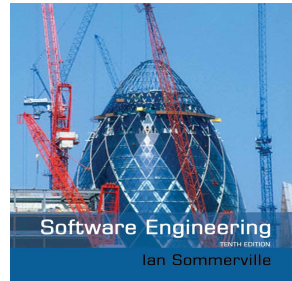
Context and interaction models



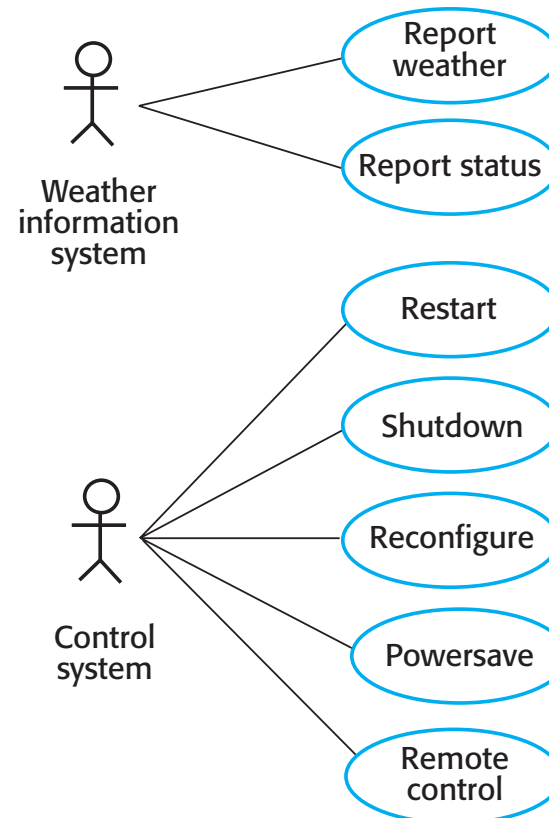
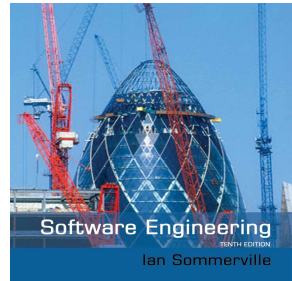
- A system context model is a structural model that demonstrates the other systems in the environment of the system being developed.
- An interaction model is a dynamic model that shows how the system interacts with its environment as it is used.



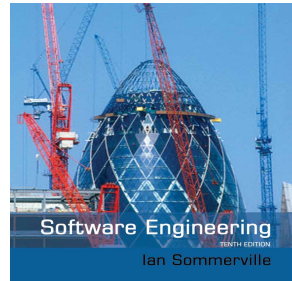
System context for the weather station



Weather station use cases



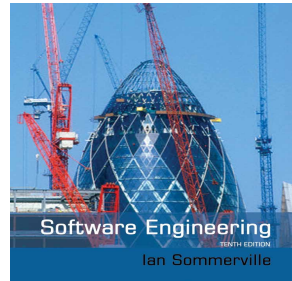
Use case description—Report weather



System	Weather station
Use case	Report weather
Actors	Weather information system, Weather station
Description	The weather station sends a summary of the weather data that has been collected from the instruments in the collection period to the weather information system. The data sent are the maximum, minimum, and average ground and air temperatures; the maximum, minimum, and average air pressures; the maximum, minimum, and average wind speeds; the total rainfall; and the wind direction as sampled at five-minute intervals.
Stimulus	The weather information system establishes a satellite communication link with the weather station and requests transmission of the data.
Response	The summarized data is sent to the weather information system.
Comments	Weather stations are usually asked to report once per hour but this frequency may differ from one station to another and may be modified in the future.



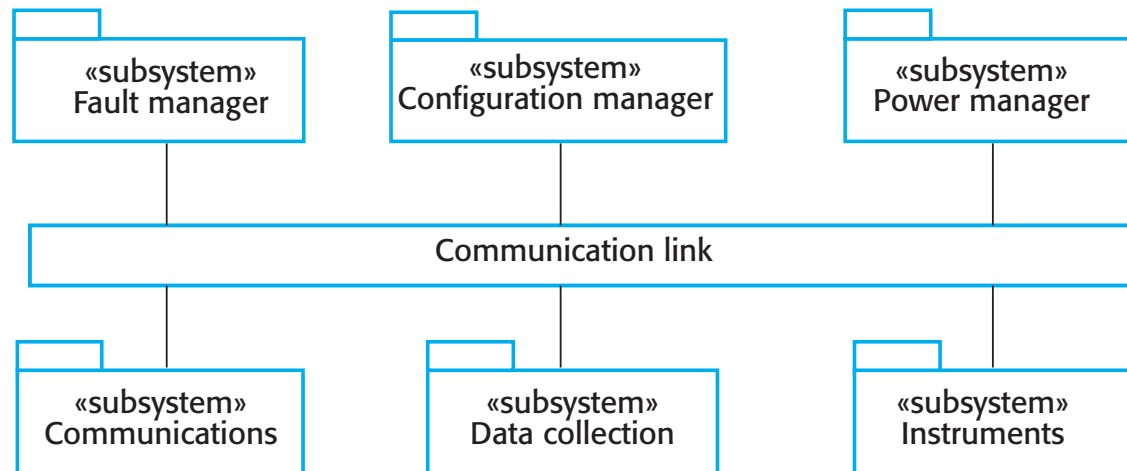
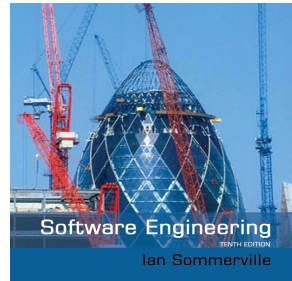
Architectural design



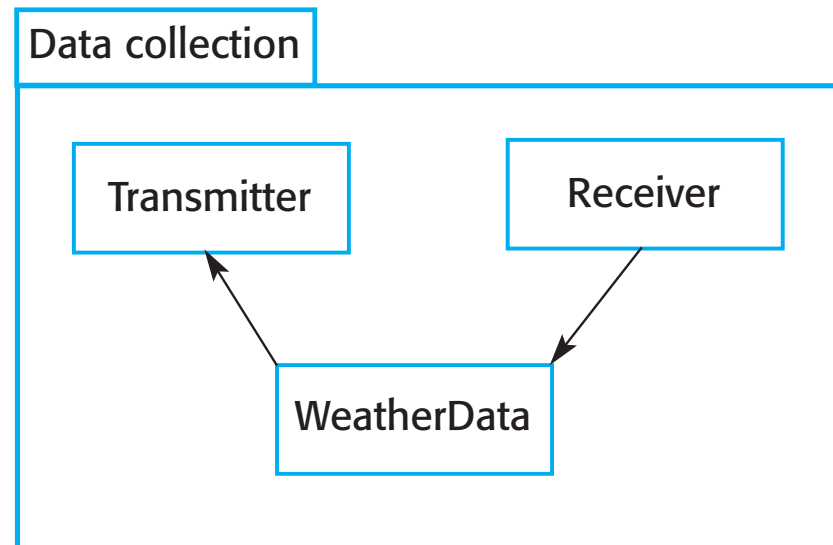
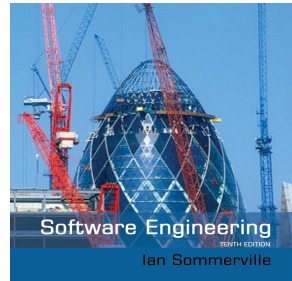
- Once interactions between the system and its environment have been understood, you use this information for designing the system architecture.
- You identify the major components that make up the system and their interactions, and then may organize the components using an architectural pattern such as a layered or client-server model.
- The weather station is composed of independent subsystems that communicate by broadcasting messages on a common infrastructure.

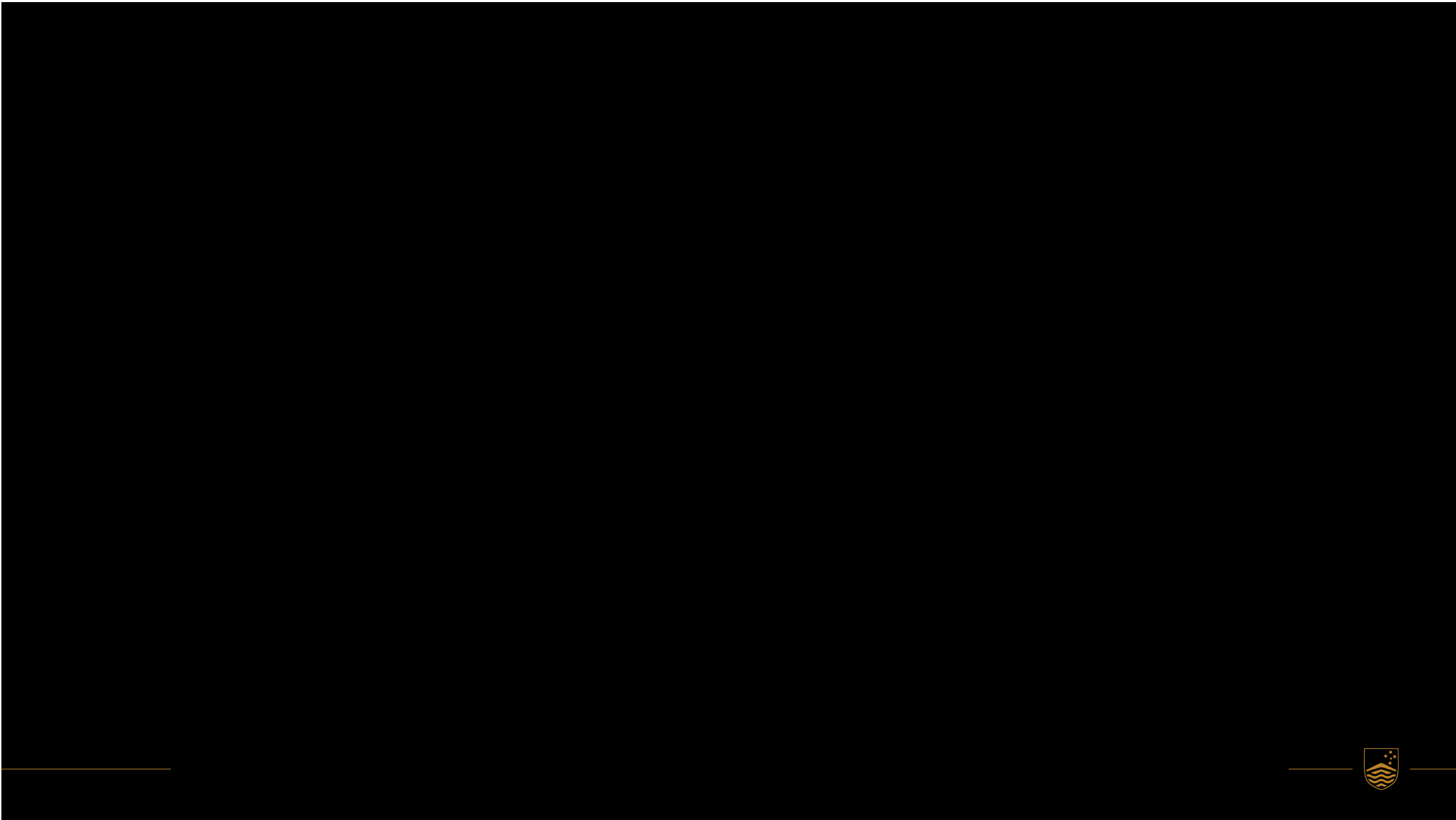


High-level architecture of the weather station



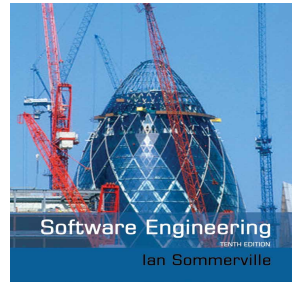
Architecture of data collection system



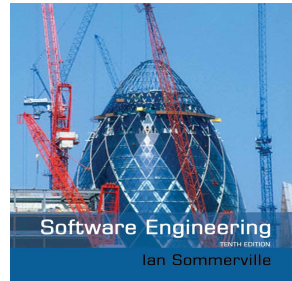


Object class identification

- Identifying object classes is often a difficult part of object oriented design.
- There is no 'magic formula' for object identification. It relies on the skill, experience and domain knowledge of system designers.
- Object identification is an iterative process. You are unlikely to get it right first time.



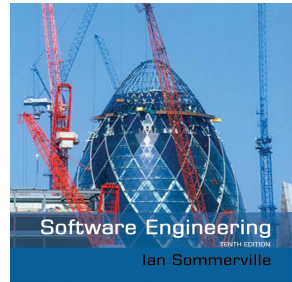
Approaches to identification



- Use a grammatical approach based on a natural language description of the system.
- Base the identification on tangible things in the application domain.
- Use a behavioural approach and identify objects based on what participates in what behaviour.
- Use a scenario-based analysis. The objects, attributes and methods in each scenario are identified.



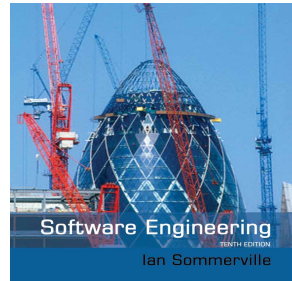
Weather station object classes



- Object class identification in the weather station system may be based on the tangible hardware and data in the system:
 - Ground thermometer, Anemometer, Barometer
 - Application domain objects that are 'hardware' objects related to the instruments in the system.
 - Weather station
 - The basic interface of the weather station to its environment. It therefore reflects the interactions identified in the use-case model.
 - Weather data
 - Encapsulates the summarized data from the instruments.



Weather station object classes



reportWeather ()
reportStatus ()
powerSave (instruments)
remoteControl (commands)
reconfigure (commands)
restart (instruments)
shutdown (instruments)

groundTemperatures
windSpeeds
windDirections
pressures
rainfall

collect ()
summarize ()

Ground thermometer

gt_Ident
temperature

Anemometer

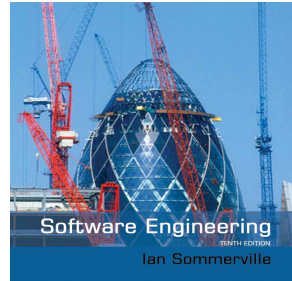
an_Ident
windSpeed
windDirection

Barometer

bar_Ident
pressure
height



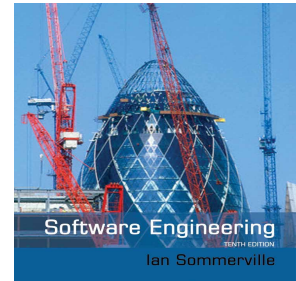
Design models



- Design models show the objects and object classes and relationships between these entities.
- There are two kinds of design model:
 - Structural models describe the static structure of the system in terms of object classes and relationships.
 - Dynamic models describe the dynamic interactions between objects.



Examples of design models

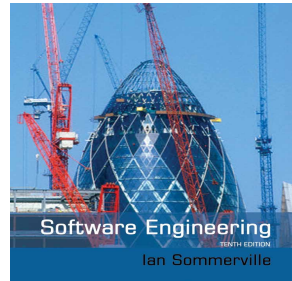


- Subsystem models that show logical groupings of objects into coherent subsystems.
- Sequence models that show the sequence of object interactions.
- State machine models that show how individual objects change their state in response to events.
- Other models include use-case models, aggregation models, generalisation models, etc.



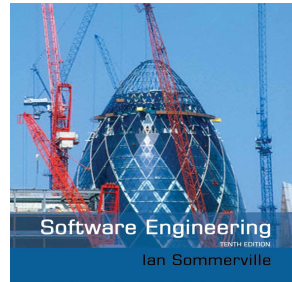
Subsystem models

- Shows how the design is organised into logically related groups of objects.
- In the UML, these are shown using packages - an encapsulation construct. This is a logical model. The actual organisation of objects in the system may be different.

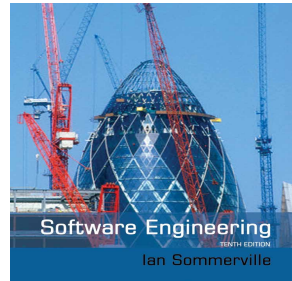


Sequence models

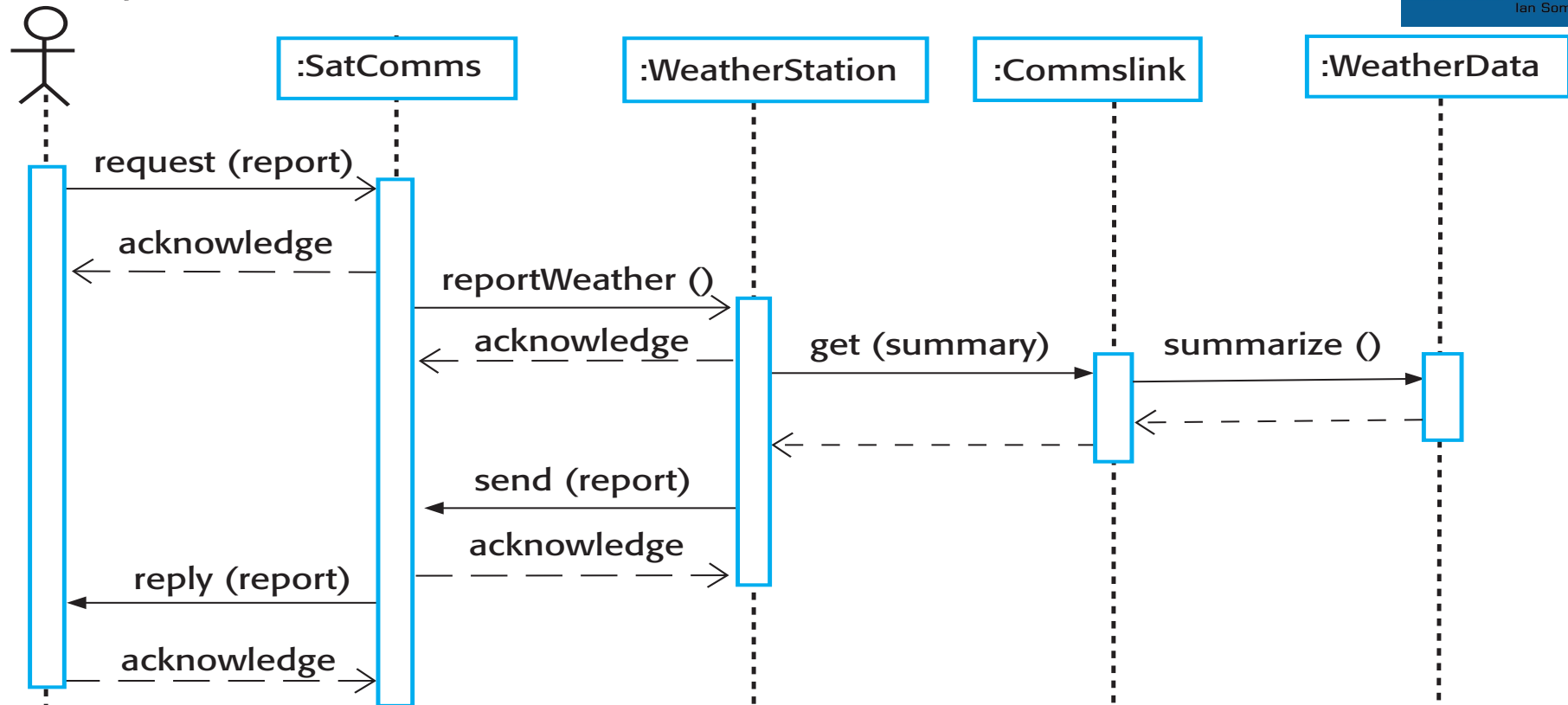
- Sequence models show the sequence of object interactions that take place
 - Objects are arranged horizontally across the top;
 - Time is represented vertically so models are read top to bottom;
 - Interactions are represented by labelled arrows, Different styles of arrow represent different types of interaction;
 - A thin rectangle in an object lifeline represents the time when the object is the controlling object in the system.



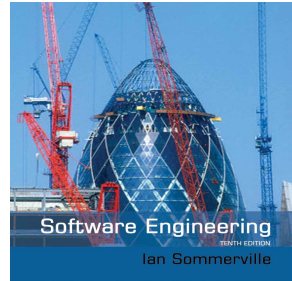
Sequence diagram describing data collection



information system



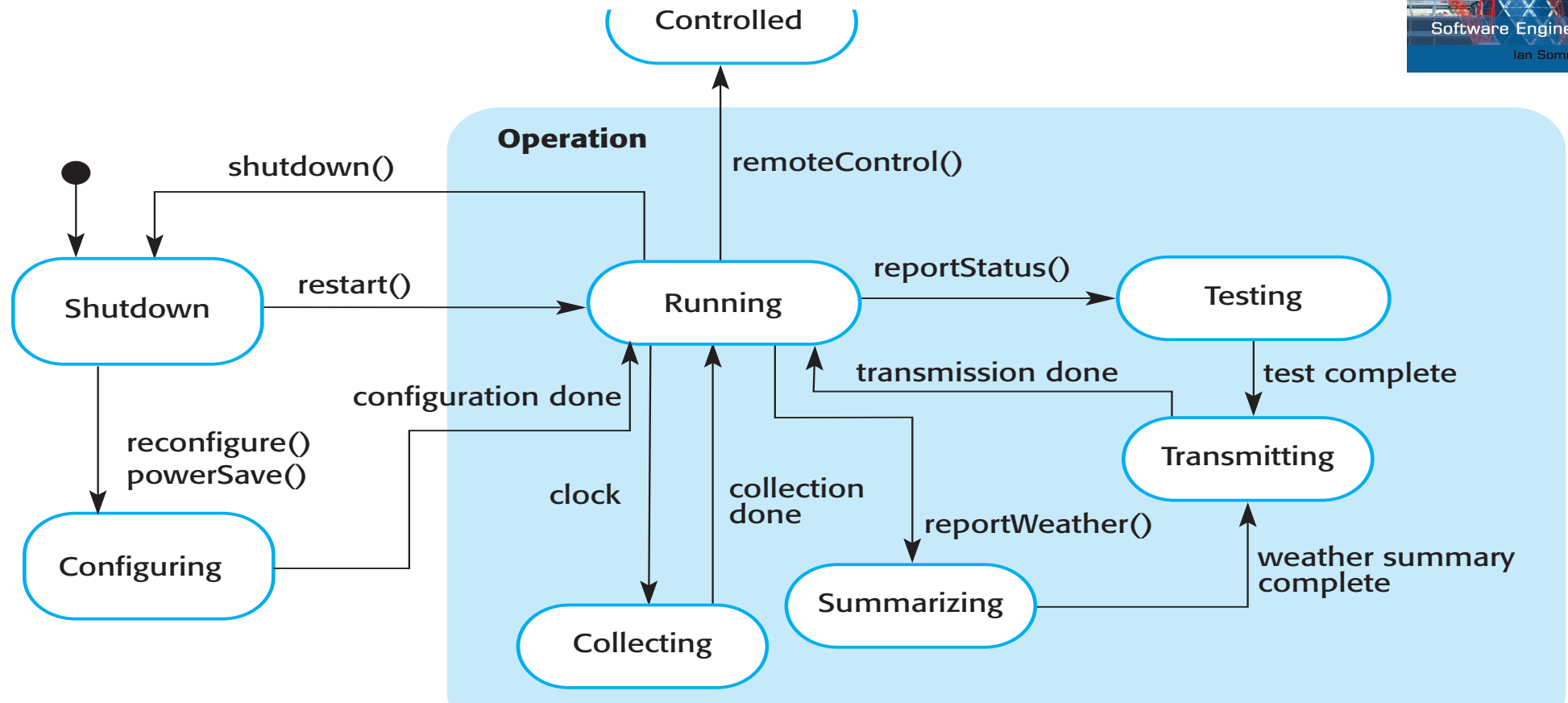
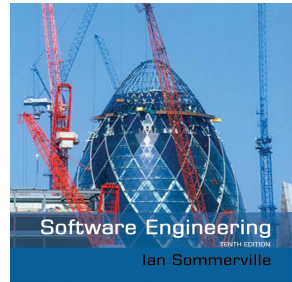
State diagrams



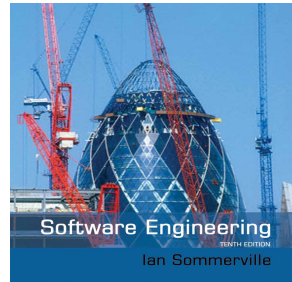
- State diagrams are used to show how objects respond to different service requests and the state transitions triggered by these requests.
- State diagrams are useful high-level models of a system or an object's run-time behavior.
- You don't usually need a state diagram for all of the objects in the system. Many of the objects in a system are relatively simple and a state model adds unnecessary detail to the design.



Weather station state diagram



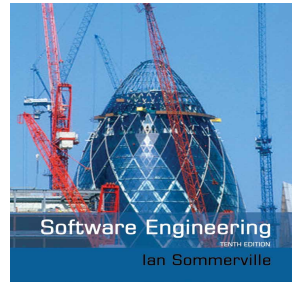
Interface specification



- Object interfaces have to be specified so that the objects and other components can be designed in parallel.
- Designers should avoid designing the interface representation but should hide this in the object itself.
- Objects may have several interfaces which are viewpoints on the methods provided.
- The UML uses class diagrams for interface specification but Java may also be used.



Weather Station Interfaces

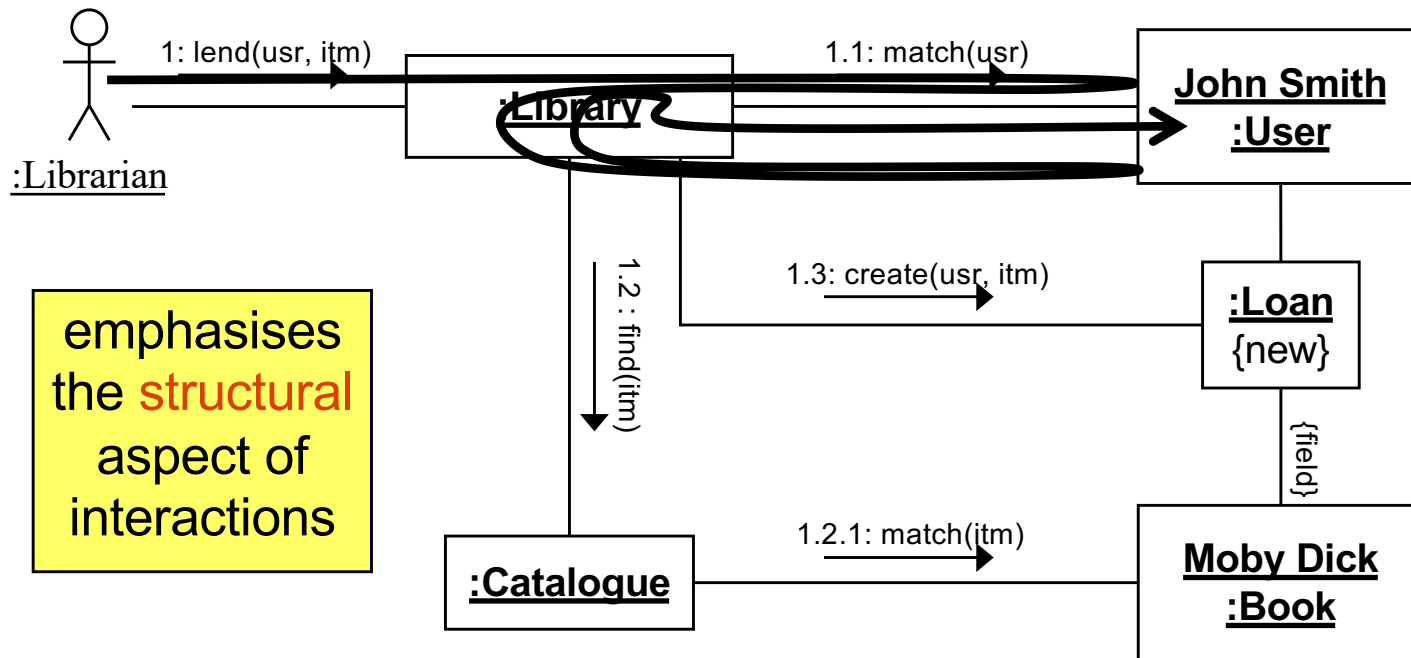
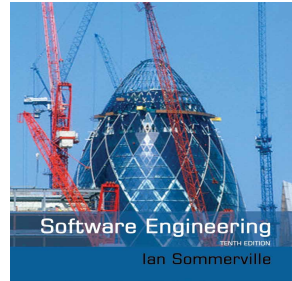


«interface» Reporting
weatherReport (WS-Ident): Wreport statusReport (WS-Ident): Sreport

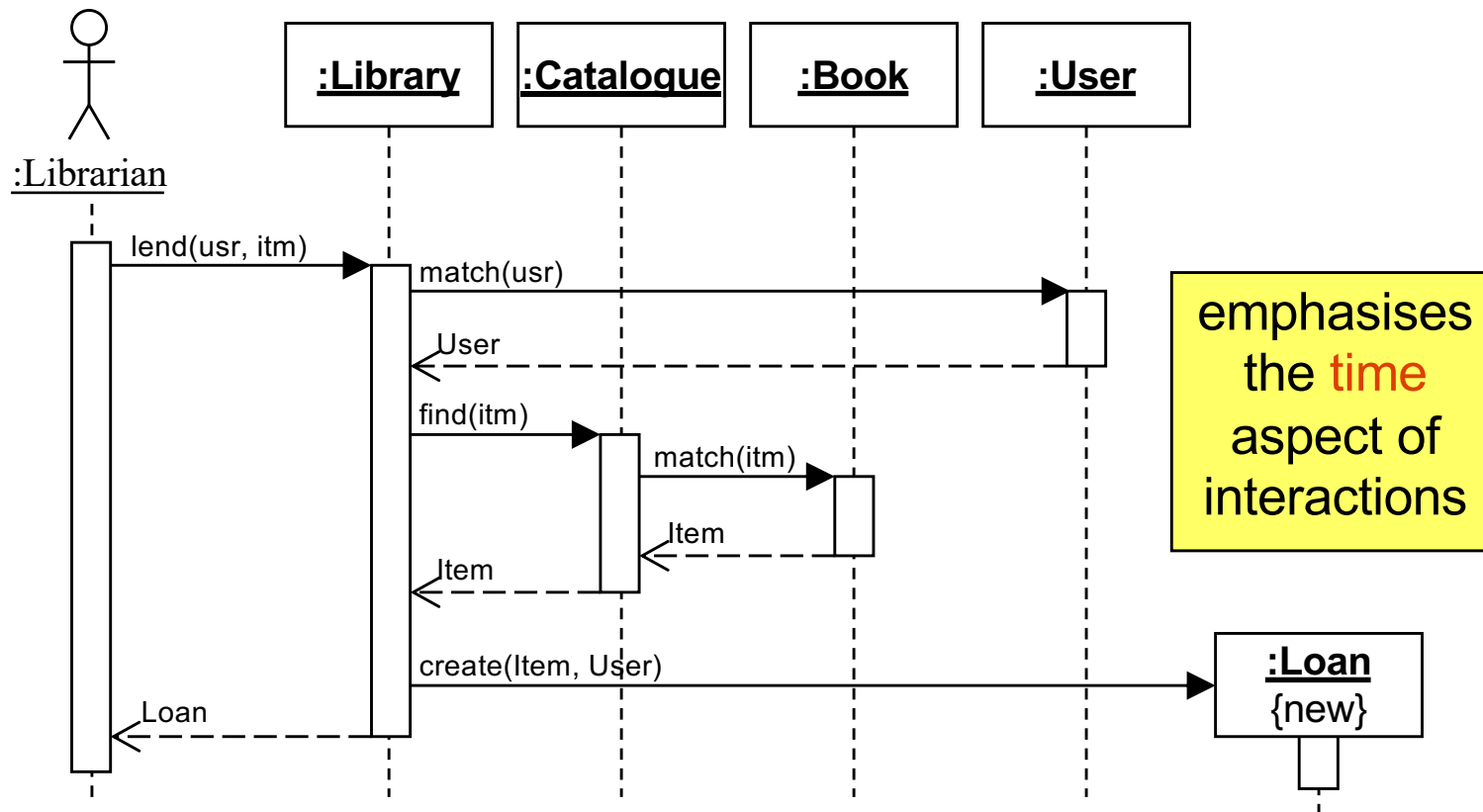
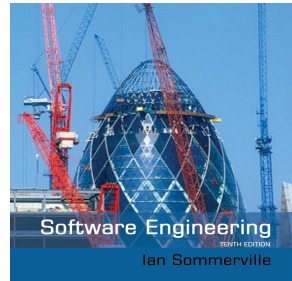
«interface» Remote Control
startInstrument(instrument): iStatus stopInstrument (instrument): iStatus collectData (instrument): iStatus provideData (instrument): string



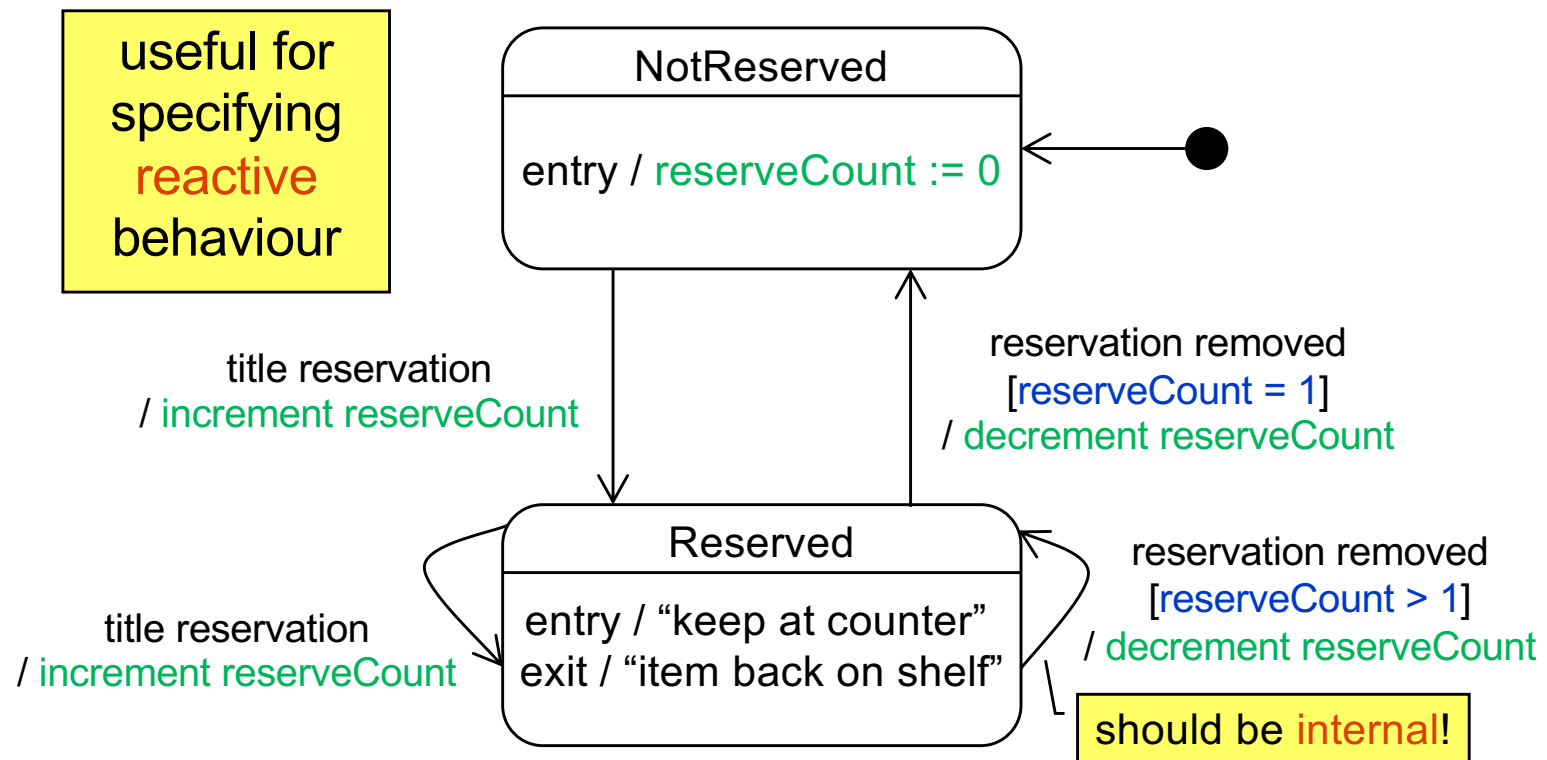
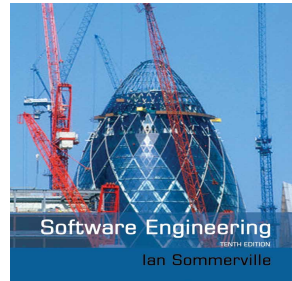
Communication Diagram



Sequence Diagram



State Diagram



State Diagram

