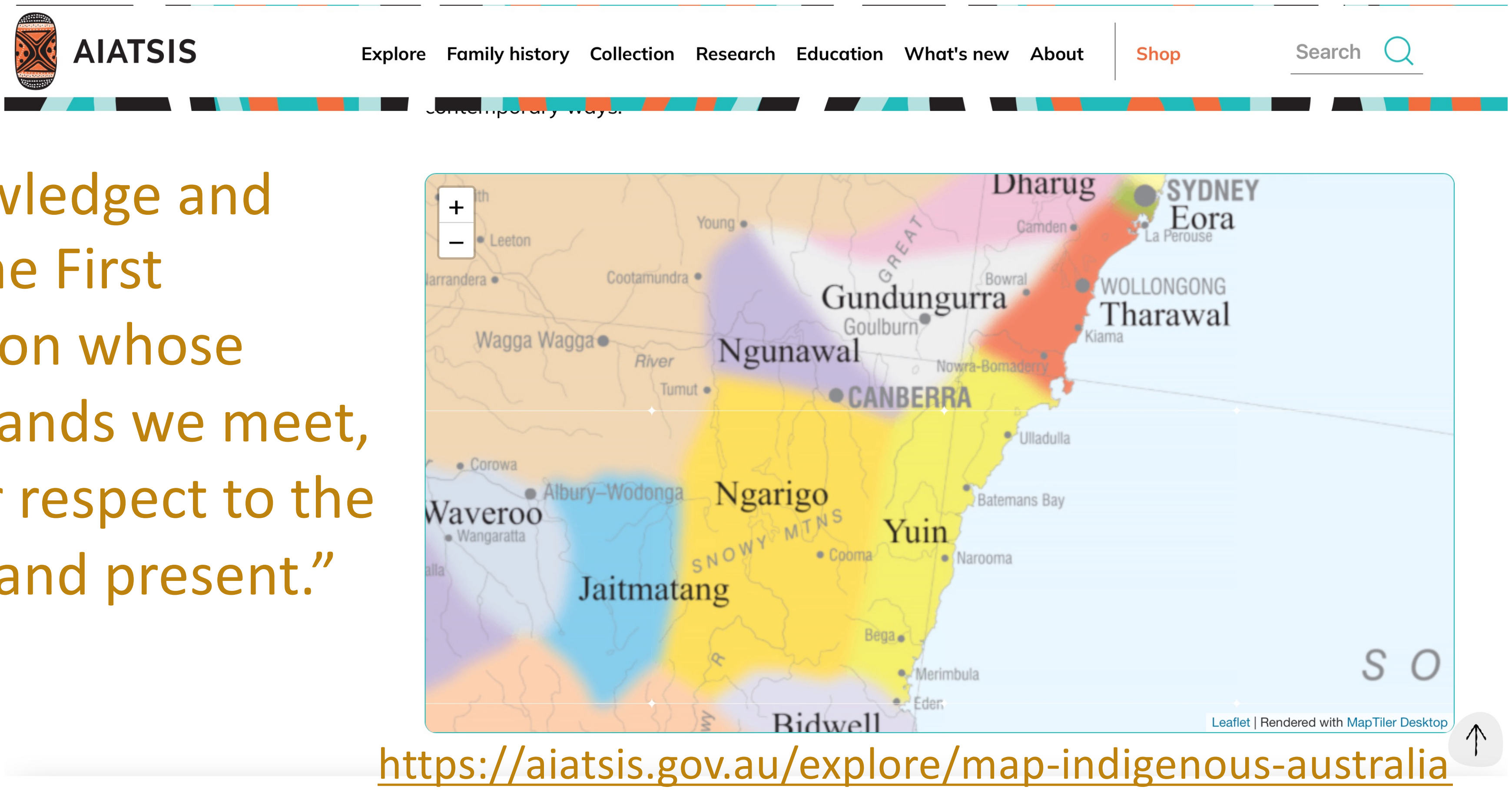
Week:
2 of 12

COMP 2120 / COMP 6120

REQUIREMENTS

A/Prof Alex Potanin and Dr Melina Vidoni

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

Personas

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- You need to have an understanding of your potential users to design features that they are likely to find useful and to design a user interface that is suited to them.
- Personas are 'imagined users' where you create a character portrait of a type of user that you think might use your product.
  - For example, if your product is aimed at managing appointments for dentists, you might create a dentist persona, a receptionist persona and a patient persona.
- Personas of different types of user help you imagine what these users may want to do with your software and how it might be used. They help you envisage difficulties that they might have in understanding and using product features.

A Persona for a Primary School Teacher

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

Jack, a primary school teacher

Jack, age 32, is a primary school (elementary school) teacher in Ullapool, a large coastal village in the Scottish Highlands. He teaches children from ages 9-12. He was born in a fishing community north of Ullapool, where his father runs a marine fuels supply business and his mother is a community nurse. He has a degree in English from Glasgow University and retrained as a teacher after several years working as a web content author for a large leisure group.

Jack's experience as a web developer means that he is confident in all aspects of digital technology. He passionately believes that the effective use of digital technologies, blended with face to face teaching, can enhance the learning experience for children. He is particularly interested in using the iLearn system for project-based teaching, where students work together across subject areas on a challenging topic.

Persona Descriptions

Engineering Software Products

An Introduction to Modern Software Engineering

Ian Sommerville

- A persona should ‘paint a picture’ of a type of product user. They should be relatively short and easy-to-read.
- You should describe their background and why they might want to use your product.
- You should also say something about their educational background and technical skills.
- These help you assess whether or not a software feature is likely to be useful, understandable and usable by typical product users.

Persona Descriptions

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

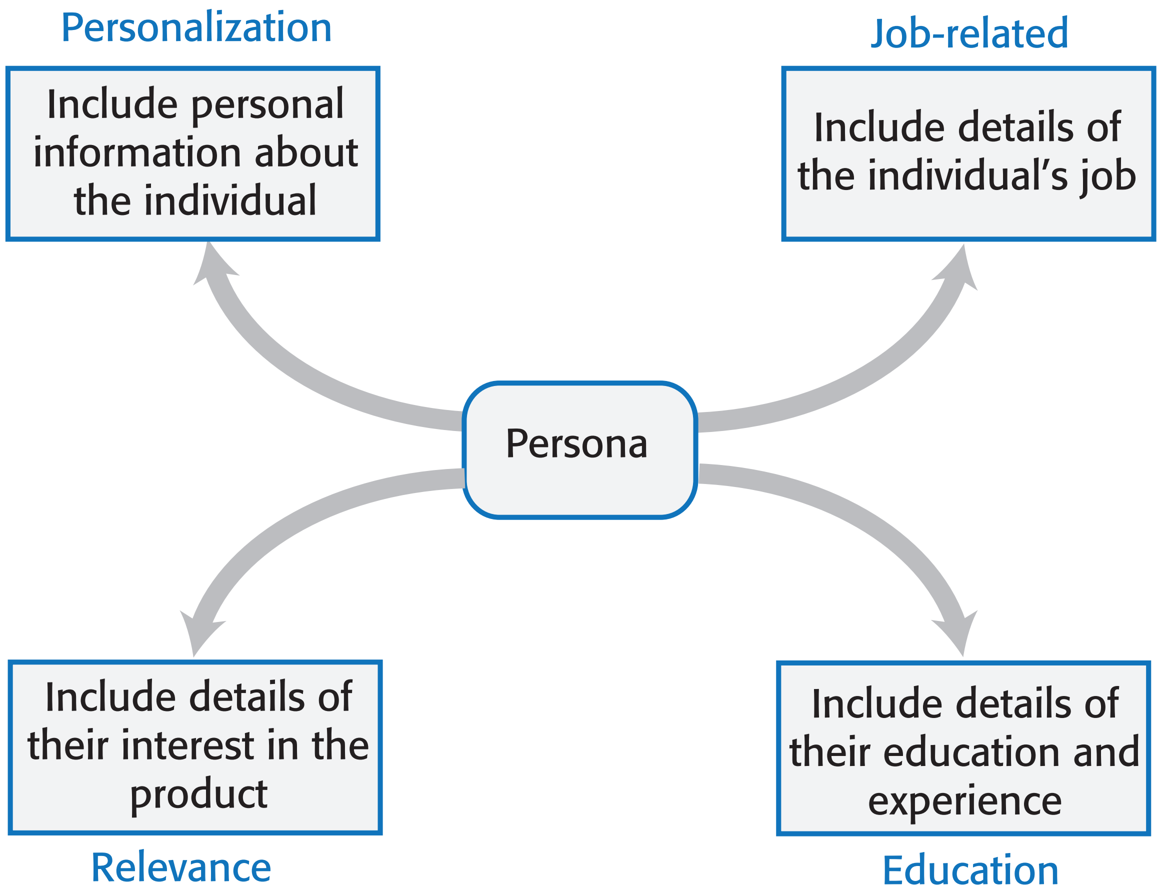

Personalization
Include personal information about the individual
Job-related
Include details of the individual's job
Persona
Include details of their interest in the product
Relevance
Include details of their education and experience
Education

Table 3.2 Aspects of a persona description

Personalization
You should give them a name and say something about their personal circumstances. This is important because you shouldn't think of a persona as a role but as an individual. It is sometimes helpful to use an appropriate stock photograph to represent the person in the persona. Some studies suggest that this helps project teams use personas more effectively.

Job-related
If your product is targeted at business, you should say something about their job and (if necessary) what that job involves. For some jobs, such as a teacher where readers are likely to be familiar with the job, this may not be necessary.

Education
You should describe their educational background and their level of technical skills and experience. This is important, especially for interface design.

Relevance
If you can, you should say why they might be interested in using the product and what they might want to do with it.

Emma, a history teacher

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

Emma, age 41, is a history teacher in a secondary school (high school) in Edinburgh. She teaches students from ages 12 to 18. She was born in Cardiff in Wales where both her father and her mother were teachers. After completing a degree in history from Newcastle University, she moved to Edinburgh to be with her partner and trained as a teacher. She has two children, aged 6 and 8, who both attend the local primary school. She likes to get home as early as she can to see her children, so often does lesson preparation, administration and marking from home.

Emma uses social media and the usual productivity applications to prepare her lessons, but is not particularly interested in digital technologies. She hates the virtual learning environment that is currently used in her school and avoids using it if she can. She believes that face-to-face teaching is most effective. She might use the iLearn system for administration and access to historic films and documents. However, she is not interested in a blended digital/face-to-face approach to teaching.

Elena, a school IT technician

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

Elena, age 28, is a senior IT technician in a large secondary school (high school) in Glasgow with over 2000 students. Originally from Poland, she has a diploma in electronics from Potsdam University. She moved to Scotland in 2011 after being unemployed for a year after graduation. She has a Scottish partner, no children, and hopes to develop her career in Scotland. She was originally appointed as a junior technician but was promoted, in 2014, to a senior post responsible for all the school computers.

Although not involved directly in teaching, Elena is often called on to help in computer science classes. She is a competent Python programmer and is a 'power user' of digital technologies. She has a long-term career goal of becoming a technical expert in digital learning technologies and being involved in their development. She wants to become an expert in the iLearn system and sees it as an experimental platform for supporting new uses for digital learning.

Persona Benefits

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- The main benefit of personas is that they help you and other development team members empathize with potential users of the software.
- Personas help because they are a tool that allows developers to ‘step into the user’s shoes’.
  - Instead of thinking about what you would do in a particular situation, you can imagine how a persona would behave and react.
- Personas can help you check your ideas to make sure that you are not including product features that aren’t really needed.
- They help you to avoid making unwarranted assumptions, based on your own knowledge, and designing an over-complicated or irrelevant product.

Deriving Personas

- Personas should be based on an understanding of the potential product users, their jobs, their background and their aspirations.
- You should study and survey potential users to understand what they want and how they might use the product.
- From this data, you can then abstract the essential information about the different types of product user and use this as a basis for creating personas.
- Personas that are developed on the basis of limited user information are called proto-personas.
  - Proto-personas may be created as a collective team exercise using whatever information is available about potential product users. They can never be as accurate as personas developed from detailed user studies, but they are better than nothing.

Scenarios

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- A scenario is a narrative that describes how a user, or a group of users, might use your system.
- There is no need to include everything in a scenario – the scenario isn't a system specification.
- It is simply a description of a situation where a user is using your product's features to do something that they want to do.
- Scenario descriptions may vary in length from two to three paragraphs up to a page of text.

Jack’s scenario: using the iLearn system for class projects

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

Fishing in Ullapool

Jack is a primary school teacher in Ullapool, teaching P6 pupils. He has decided that a class project should be focused around the fishing industry in the area, looking at the history, development and economic impact of fishing.

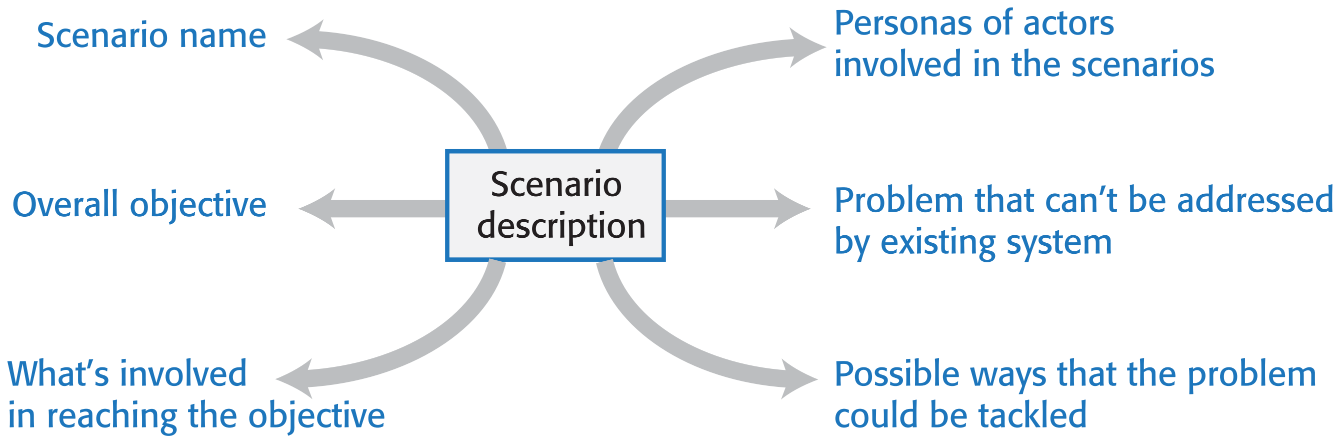
As part of this, students are asked to gather and share reminiscences from relatives, use newspaper archives and collect old photographs related to fishing and fishing communities in the area. Pupils use an iLearn wiki to gather together fishing stories and SCRAN (a history archive site) to access newspaper archives and photographs. However, Jack also needs a photo-sharing site as he wants students to take and comment on each others’ photos and to upload scans of old photographs that they may have in their families. He needs to be able to moderate posts with photos before they are shared, because pre-teen children can’t understand copyright and privacy issues.

Jack sends an email to a primary school teachers’ group to see if anyone can recommend an appropriate system. Two teachers reply and both suggest that he uses KidsTakePics, a photo-sharing site that allows teachers to check and moderate content. As KidsTakePics is not integrated with the iLearn authentication service, he sets up a teacher and a class account with KidsTakePics.

He uses the the iLearn setup service to add KidsTakePics to the services seen by the students in his class so that, when they log in, they can immediately use the system to upload photos from their phones and class computers.

Figure 3.5 Elements of a scenario description

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION


Scenario name
Overall objective
What's involved in reaching the objective
Scenario description
Personas of actors involved in the scenarios
Problem that can't be addressed by existing system
Possible ways that the problem could be tackled

Scenario elements

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- A brief statement of the overall objective.
  - In Jack's scenario, this is to support a class project on the fishing industry.
- References to the personas involved (Jack) so that you can get information about the capabilities and motivation of that user.
- Information about what is involved in doing the activity. For example, in Jack's scenario this involves gathering reminiscences from relatives, accessing newspaper archives, etc.
- An explanation of problems that can't be readily addressed using the existing system.
  - Young children don't understand issues such as copyright and privacy, so photo sharing requires a site that a teacher can moderate to make sure that published images are legal and acceptable.
- A description of one way that the identified problem might be addressed.
  - In Jack's scenario, the preferred approach is to use an external tool designed for school students.

Emma’s scenario

UNPREPARED
TO DO
ANALYSIS
DEV
GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Emma’s scenario is different from Jack’s scenario in that it describes a common and well-understood process rather than something new.
- Emma is an e-learning sceptic and she is not interested in innovative applications. She wants a system that will make her life easier and reduce the amount of routine administration that she has to do.
- The scenario discusses how parts of the process (setting up an email group and web page) are automated by the iLearn system.

Table 3.6 Emma’s scenario: using iLearn for administration

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

Emma is teaching the history of the First World War to a class of 14 year olds (S3). A group of S3 students are visiting the historic World War One battlefields in northern France. She want to set up a ‘battlefields group’ where the students who are attending the trip can share their research about the places they are visiting as well as their pictures and thoughts about the visit.

From home, she logs onto the iLearn system system using her Google account credentials. Emma has two iLearn accounts – her teacher account and a parent account associated with the local primary school. The system recognises that she is a multiple account owner and asks her to select the account to be used. She chooses the teacher account and the system generates her personal welcome screen. As well as her selected applications, this also shows management apps that help teachers create and manage student groups.

Emma selects the ‘group management’ app, which recognizes her role and school from her identity information and creates a new group. The system prompts for the class year (S3) and subject (history) and automatically populates the new group with all S3 students who are studying history. She selects those students going on the trip and adds her teacher colleagues, Jamie and Claire, to the group.

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

Table 3.6 Emma’s scenario: using iLearn for administration

She names the group and confirms that it should be created. The app sets up an icon on her iLearn screen to represent the group, creates an email alias for the group and asks Emma if she wishes to share the group. She shares access with everyone in the group, which means that they also see the icon on their screen. To avoid getting too many emails from students, restricts sharing of the email alias to Jamie and Claire.

The group management app then asksEmma if she wishes to set up a group web page, wiki and blog. Emma confirms that a web page should be created and she types some text to be included on that page.

She then accesses flickr using the icon on her screen, logs in and creates a private group to share trip photos that students and teachers have taken. She uploads some of her own photos from previous trips and emails an invitation to join the photo-sharing group to the Battlefield email list. Emma uploads material from her own laptop that she has written about the trip to iLearn and shares this with the ‘Battlefields Group’. This action adds her documents to the web page and generates an alert to group members that new material is available.

Writing scenarios

UNPREPARED
TO DO
ANALYSIS
DEV
GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Scenarios should always be written from the user's perspective and based on identified personas or real users.
- Your starting point for scenario writing should be the personas that you have created. You should normally try to imagine several scenarios from each persona.
- Ideally, scenarios should be general and should not include implementation information.
  - However, describing an implementation is often the easiest way to explain how a task is done.
- It is important to ensure that you have coverage of all of the potential user roles when describing a system.

Table 3.7 Elena's scenario: configuring the iLearn system

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

Elena has been asked by David, the head of the art department in her school, to help set up an iLearn environment for his department. David wants an environment that includes tools for making and sharing art, access to external websites to study artworks, and 'exhibition' facilities so that the students' work can be displayed.

Elena's starts by talking to art teachers to discover the tools that they recommend and the art sites that they use for studies. She also discovers that the tools they use and the sites they access vary according to the age of their students. Consequently, different student groups should be presented with a toolset that is appropriate for their age and experience.

Once she has established what is required, Elena logs into the iLearn system as an administrator and starts configuring the art environment using the iLearn setup service. She creates sub-environments for three age groups plus a shared environment that includes tools and sites that may be used by all students.

She drags and drops tools that are available locally and the URLs of external websites into each of these environments. For each of the sub-environments, she assigns an art teacher as its administrator so that they can refine the tool and web site selection that has been set up. She publishes the environments in 'review mode' and makes them available to the teachers in the art department.

After discussing the environments with the teachers, Elena shows them how to refine and extend the environments. Once they have agreed that the art environment is useful, it is released to all students in the school.

User involvement

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- It is easy for anyone to read and understand scenarios, so it is possible to get users involved in their development.
- The best approach is to develop an imaginary scenario based on our understanding of how the system might be used then ask users to explain what you have got wrong.
- They might ask about things they did not understand and suggest how the scenario could be extended and made more realistic.
- Our experience was that users are not good at writing scenarios.
  - The scenarios that they created were based on how they worked at the moment. They were far too detailed and the users couldn’t easily generalize their experience.

User stories

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

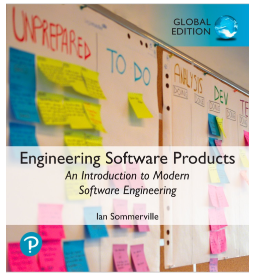
- Scenarios are high-level stories of system use. They should describe a sequence of interactions with the system but should not include details of these interactions.
- User stories are finer-grain narratives that set out in a more detailed and structured way a single thing that a user wants from a software system.
  - As an author, I need a way to organize the book that I'm writing into chapters and sections.
- This story reflects what has become the standard format of a user story:
  - **As a** <role>, I <want | need> **to** <do something>
    - As a teacher, I want to tell all members of my group when new information is available
- A variant of this standard format adds a justification for the action:
  - **As a** <role> I <want | need> **to** <do something> **so that** <reason>
    - As a teacher, I need to be able to report who is attending a class trip so that the school maintains the required health and safety records.

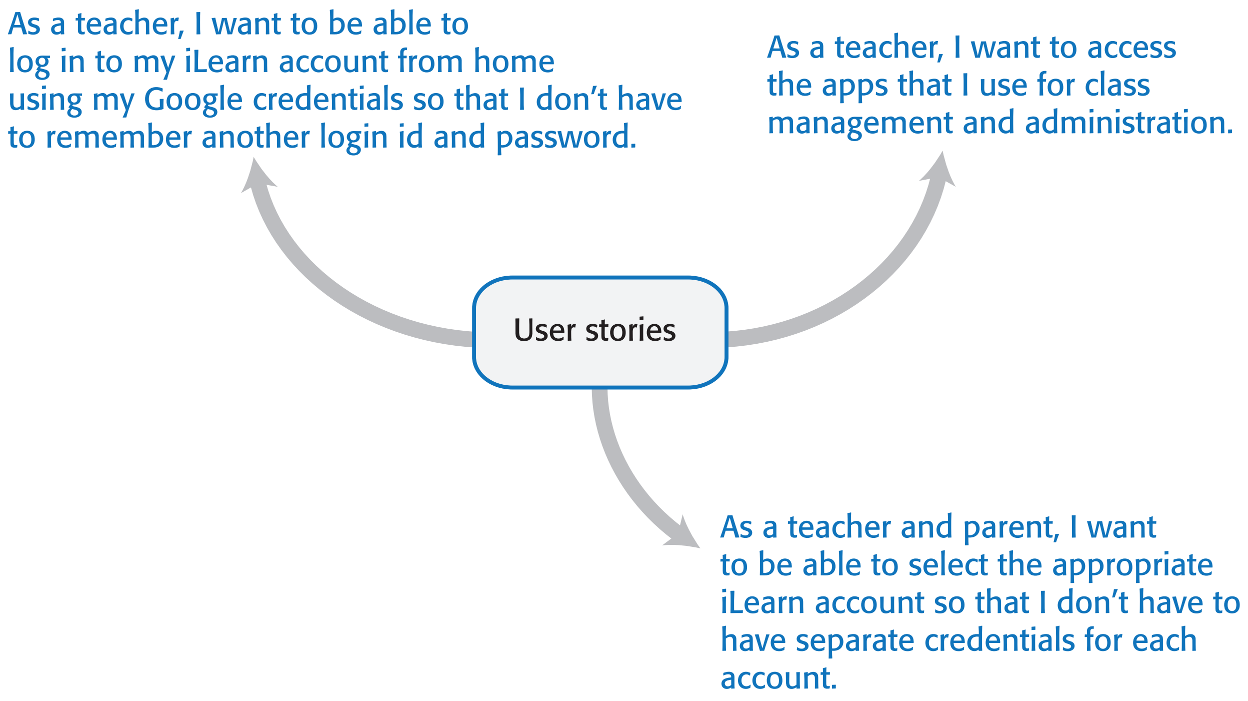
User stories in planning

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- An important use of user stories is in planning.
  - Many users of the Scrum method represent the product backlog as a set of user stories.
- User stories should focus on a clearly defined system feature or aspect of a feature that can be implemented within a single sprint.
- If the story is about a more complex feature that might take several sprints to implement, then it is called an epic.
  - As a system manager, I need a way to backup the system and restore either individual applications, files, directories or the whole system.
  - There is a lot of functionality associated with this user story. For implementation, it should be broken down into simpler stories with each story focusing on a single aspect of the backup system.

Figure 3.6 User stories from Emma's scenario


Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION



As a teacher, I want to be able to log in to my iLearn account from home using my Google credentials so that I don't have to remember another login id and password.

As a teacher, I want to access the apps that I use for class management and administration.

As a teacher and parent, I want to be able to select the appropriate iLearn account so that I don't have to have separate credentials for each account.

Feature description using user stories

GLOBAL EDITION

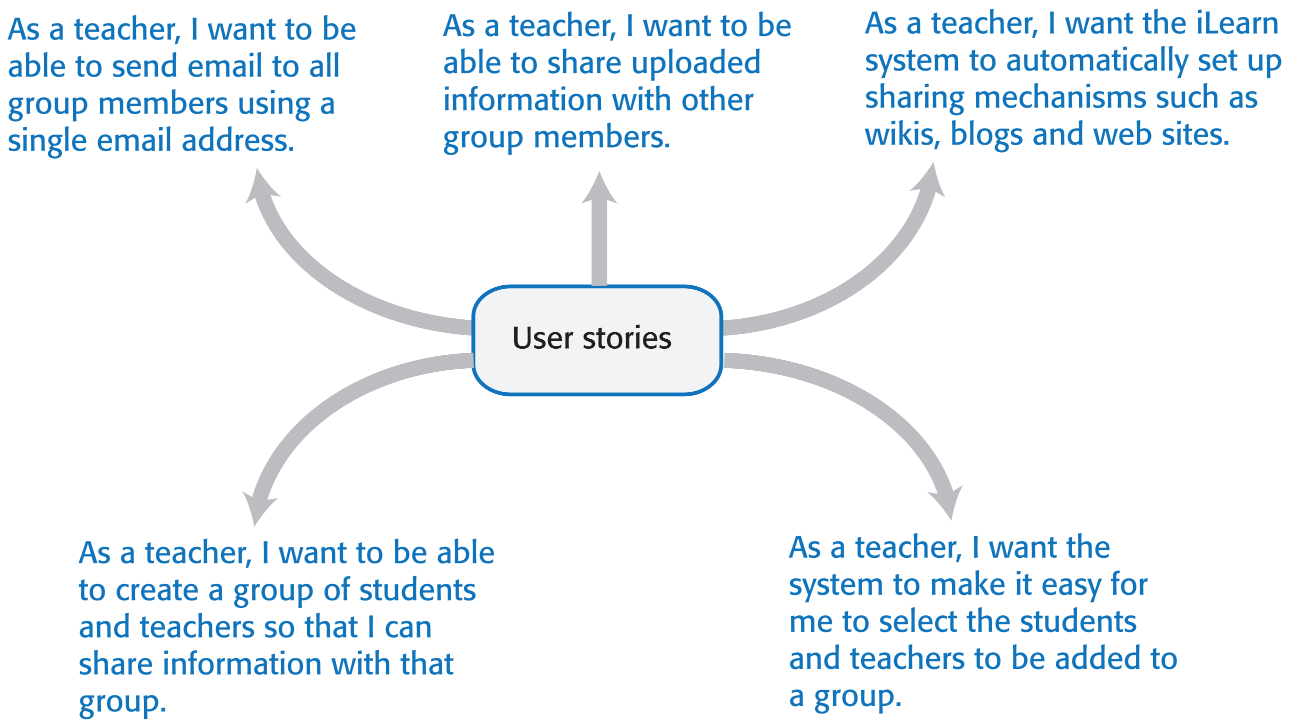
Engineering Software Products

An Introduction to Modern Software Engineering

Ian Sommerville

- Stories can be used to describe features in your product that should be implemented.
- Each feature can have a set of associated stories that describe how that feature is used.

Figure 3.7 User stories describing the Groups feature


As a teacher, I want to be able to send email to all group members using a single email address.
As a teacher, I want to be able to share uploaded information with other group members.
As a teacher, I want the iLearn system to automatically set up sharing mechanisms such as wikis, blogs and web sites.
User stories
As a teacher, I want to be able to create a group of students and teachers so that I can share information with that group.
As a teacher, I want the system to make it easy for me to select the students and teachers to be added to a group.

Stories and scenarios

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- As you can express all of the functionality described in a scenario as user stories, do you really need scenarios?’

- Scenarios are more natural and are helpful for the following reasons:

  - Scenarios read more naturally because they describe what a user of a system is actually doing with that system. People often find it easier to relate to this specific information rather than the statement of wants or needs set out in a set of user stories.

  - If you are interviewing real users or are checking a scenario with real users, they don’t talk in the stylized way that is used in user stories. People relate better to the more natural narrative in scenarios.

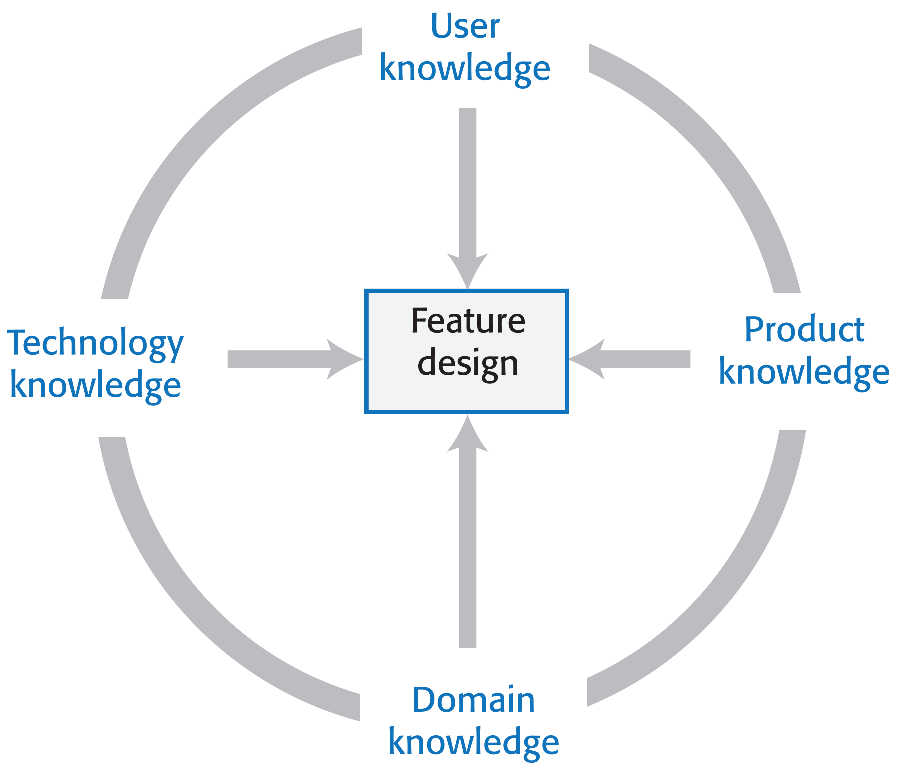
  - Scenarios often provide more context - information about what the user is trying to do and their normal ways of working. You can do this in user stories, but it means that they are no longer simple statements about the use of a system feature.

Feature identification

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Your aim in the initial stage of product design should be to create a list of features that define your product.
- A feature is a way of allowing users to access and use your product's functionality so the feature list defines the overall functionality of the system.
- Features should be independent, coherent and relevant:
  - *Independence*
    Features should not depend on how other system features are implemented and should not be affected by the order of activation of other features.
  - *Coherence*
    Features should be linked to a single item of functionality. They should not do more than one thing and they should never have side-effects.
  - *Relevance*
    Features should reflect the way that users normally carry out some task. They should not provide obscure functionality that is hardly ever required.

Figure 3.8 Feature design

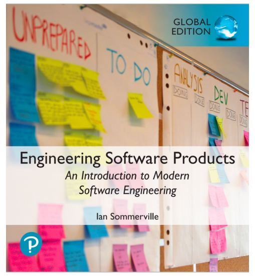


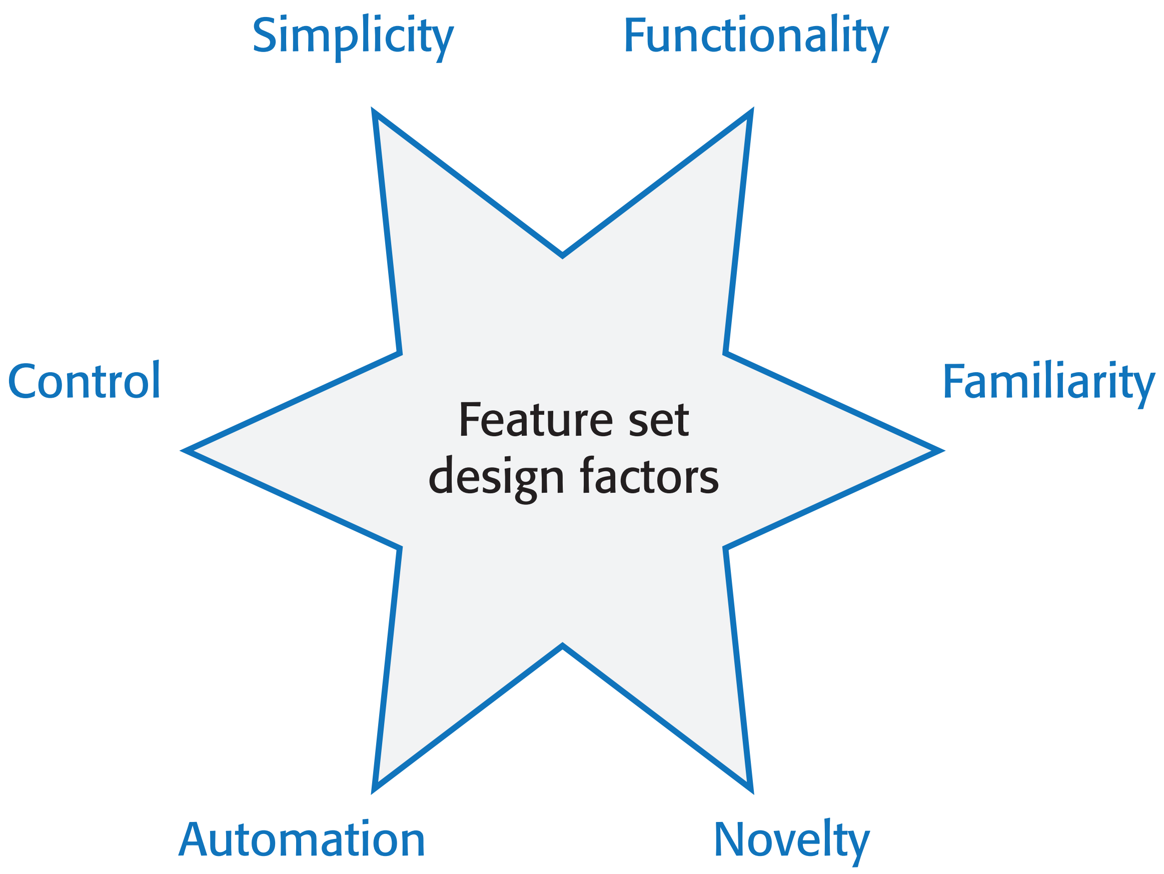
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

Table 3.8 Knowledge required for feature design

- ***User knowledge***
  You can use user scenarios and user stories to inform the team of what users want and how they might use it the software features.
- ***Product knowledge***
  You may have experience of existing products or decide to research what these products do as part of your development process. Sometimes, your features have to replicate existing features in these products because they provide fundamental functionality that is always required.
- ***Domain knowledge***
  This is knowledge of the domain or work area(e.g. finance, event booking) that your product aims to support. By understanding the domain, you can think of new innovative ways of helping users do what they want to do.
- ***Technology knowledge***
  New products often emerge to take advantage of technological developments since their competitors were launched. If you understand the latest technology, you can design features to make use of it.

Figure 3.9 Factors in feature set design


Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION



Feature trade-offs

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Simplicity and functionality
  - You need to find a balance between providing a simple, easy-to-use system and including enough functionality to attract users with a variety of needs.
- Familiarity and novelty
  - Users prefer that new software should support the familiar everyday tasks that are part of their work or life. To encourage them to adopt your system, you need to find a balance between familiar features and new features that convince users that your product can do more than its competitors.
- Automation and control
  - Some users like automation, where the software does things for them. Others prefer to have control. You have to think carefully about what can be automated, how it is automated and how users can configure the automation so that the system can be tailored to their preferences.

Feature creep

Engineering Software Products

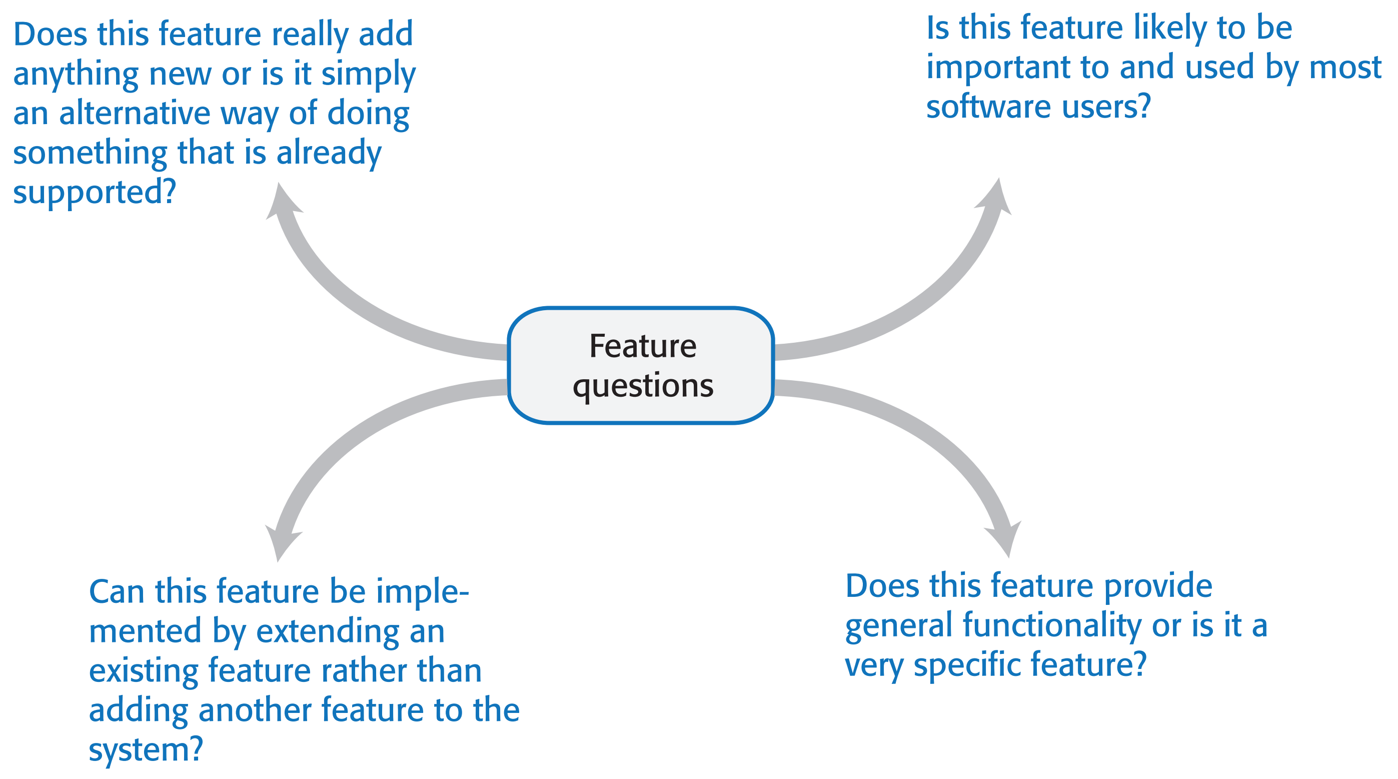
An Introduction to Modern Software Engineering

Ian Sommerville

- Feature creep occurs when new features are added in response to user requests without considering whether or not these features are generally useful or whether they can be implemented in some other way.
- Too many features make products hard to use and understand
- There are three reasons why feature creep occurs:
  - Product managers are reluctant to say ‘no’ when users ask for specific features.
  - Developers try to match features in competing products.
  - The product includes features to support both inexperienced and experienced users.

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

Figure 3.10 Avoiding feature creep


Feature questions
Does this feature really add anything new or is it simply an alternative way of doing something that is already supported?
Is this feature likely to be important to and used by most software users?
Can this feature be implemented by extending an existing feature rather than adding another feature to the system?
Does this feature provide general functionality or is it a very specific feature?

Feature derivation

UNPREPARED
TO DO
ANALYSIS
DEV
GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Features can be identified directly from the product vision or from scenarios.
- You can highlight phrases in narrative description to identify features to be included in the software.
  - You should think about the features needed to support user actions, identified by active verbs, such as use and choose.

The iLearn system vision

UNPREPARED
TO DO
ANALYSIS
DEV
GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- FOR teachers and educators WHO need a way *to help students use web-based learning resources and applications*, THE iLearn system is an open learning environment THAT *allows the set of resources used by classes and students to be easily configured for these students and classes by teachers themselves*.
- UNLIKE Virtual Learning Environments, such as Moodle, the focus of iLearn is the learning process itself, rather than the administration and management of materials, assessments and coursework. OUR product *enables teachers to create subject and age-specific environments for their students* using any web-based resources, such as videos, simulations and written materials that are appropriate

Features from the product vision

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- A feature that allows users to access and use existing web-based resources;
- A feature that allows the system to exist in multiple different instantiations;
- A feature that allows user configuration of the system to create a specific instantiation.

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

Table 3.10 Jack's scenario with highlighted phrases

Jack is a primary school teacher in Ullapool, teaching P6 pupils. He has decided that a class project should be focused around the fishing industry in the area, looking at the history, development and economic impact of fishing.

As part of this, students are asked to gather and share reminiscences from relatives, use newspaper archives and collect old photographs related to fishing and fishing communities in the area. *Students use an iLearn wiki* to gather together fishing stories and *SCRAN (a history archive) to access newspaper archives and photographs.* However, Jack also needs a photo-sharing site as he wants *pupils to take and comment on each others' photos* and to *upload scans of old photographs* that they may have in their families. He needs to be able to moderate posts with photos before they are shared, because pre-teen children can't understand copyright and privacy issues.

Jack *sends an email to a primary school teachers' group*, which he is a member of to see if anyone can recommend an appropriate system. Two teachers reply and both suggest that he uses KidsTakePics, a photo-sharing site that allows teachers to check and moderate content. As KidsTakePics *is not integrated with the iLearn authentication service*, he sets up a teacher and a class account with KidsTakePics.

He uses the the iLearn setup service to add KidsTakePics to the services seen by the students in his class so that when they log in, they can immediately use the system to upload photos from their phones and class computers.

Features from Jack’s scenario

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- A wiki for group writing.
- Access to the SCRAN history archive. This is a shared national resource that provides access to historical newspaper and magazine articles for schools and universities.
- Features to set up and access an email group.
- A feature to integrate applications with the iLearn authentication service.

The feature list

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- The output of the feature identification process should be a list of features that you use for designing and implementing your product.
- There is no need to go into a lot of detail about the features at this stage. You add detail when you are implementing the feature.
- You can describe features using a standard input-action-output template by using structured narrative descriptions or by a set of user stories.

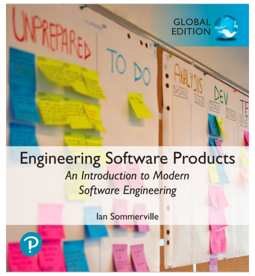
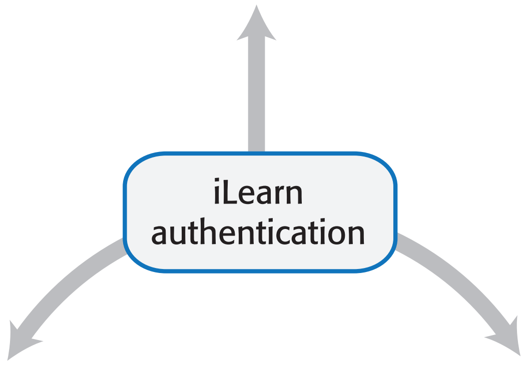

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

Figure 3.11 The iLearn authentication feature



Description
Authentication is used to identify users to the system and is currently based on a login id/password system. Users may authenticate themselves using their national user id and a personal password or may use their Google or Facebook credentials.

Constraints
All users must have a national user id and system password that they use for initial system authentication. They may then link their account with their Google/Facebook account for future authentication sessions.

Comments
Future authentication mechanisms may be based on biometrics and this should be considered in the design of the system.

Feature description using user stories

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- ***Description***
  As a system manager, I want to create and configure an iLearn environment by adding and removing services to/from that environment so that I can create environments for specific purposes.

- As a system manager, I want to set up sub-environments that include a subset of services that are included in another environment.

- As a system manager, I want to assign administrators to created environments.

- As a system manager, I want to limit the rights of environment administrators so that they cannot accidentally or deliberately disrupt the operation of key services.

- As a teacher, I want to be able to add services that are not integrated with the iLearn authentication system.

- ***Constraints***
  The use of some tools may be limited for license reasons so there may be a need to access license management tools during configuration.

- ***Comments***
  Based on Elena's and Jack's scenarios

Innovation and feature identification

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Scenarios and user stories should always be your starting point for identifying product features.
  - Scenarios tell you how users work at the moment. They don't show how they might change their way of working if they had the right software to support them.
  - Stories and scenarios are 'tools for thinking' and they help you gain an understanding of how your software might be used. You can identify a feature set from stories and scenarios.
- User research, on its own, rarely helps you innovate and invent new ways of working.
- You should also think creatively about alternative or additional features that help users to work more efficiently or to do things differently.

Key Points

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- A software product feature is a fragment of functionality that implements something that a user may need or want when using the product.
- The first stage of product development is to identify the list of product features in which you identify each feature and give a brief description of its functionality.
- Personas are 'imagined users' where you create a character portrait of a type of user that you think might use your product.
- A persona description should 'paint a picture' of a typical product user. It should describe their educational background, technology experience and why they might want to use your product.
- A scenario is a narrative that describes a situation where a user is accessing product features to do something that they want to do.

Key Points

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- A scenario is a narrative that describes a situation where a user is accessing product features to do something that they want to do.
- Scenarios should always be written from the user's perspective and should be based on identified personas or real users.
- User stories are finer-grain narratives that set out, in a structured way, something that a user wants from a software system.
- User stories may be used as a way of extending and adding detail to a scenario or as part of the description of system features.
- The key influences in feature identification and design are user research, domain knowledge, product knowledge, and technology knowledge.
- You can identify features from scenarios and stories by highlighting user actions in these narratives and thinking about the features that you need to support these actions.

Today's Goals

- Explain the importance and challenges of requirements in software engineering.
- Explain how and why requirements articulate the relationship between a desired system and its environment. Identify assumptions.
- Distinguish between and give examples of: functional and quality requirements; informal statements and verifiable requirements.
- State quality requirements in measurable ways

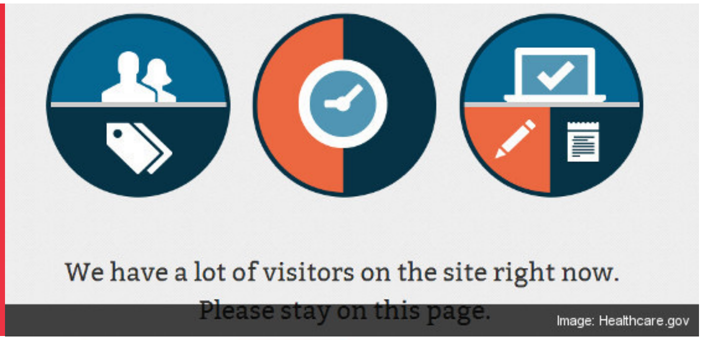
Overly simplified definition.

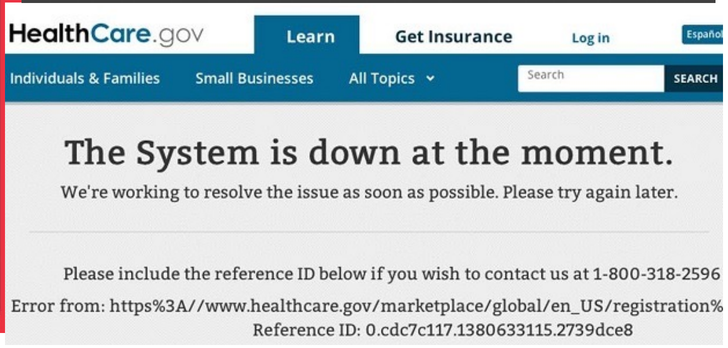
Requirements say what the system will do

(and not how it will do it)

Healthcare.gov

MARCH 10, 2014
Ukraine's Future / Minimum-Wage Wars / Wes Anderson
TIME
CODE RED_
Inside the nightmare launch of HealthCare.gov and the team that figured out how to fix it
BY STEVEN BRILL
time.com


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



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%

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 HARD???

Communication Problem

Goal: figure out what should be built.

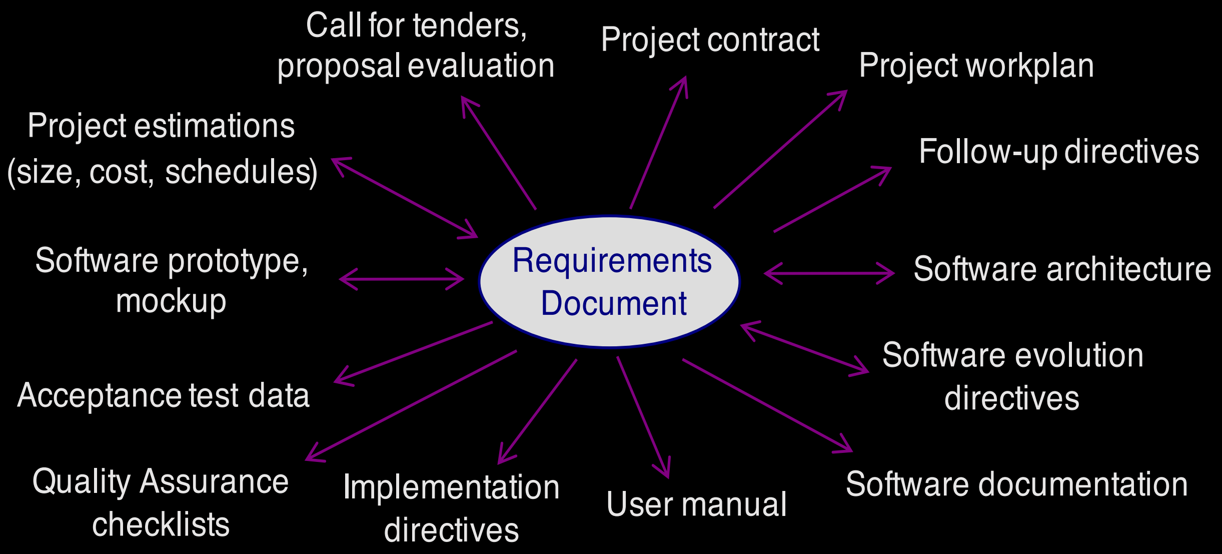
Express those ideas so that the correct thing is built.

Overall Problems

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

Mini Break in Monday Lecture

Requirements in software projects


Call for tenders, proposal evaluation
Project contract
Project workplan
Project estimations (size, cost, schedules)
Follow-up directives
Software prototype, mockup
Requirements Document
Software architecture
Software evolution directives
Acceptance test data
Quality Assurance checklists
Implementation directives
User manual
Software documentation

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*.

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/Non Functional 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
- (We'll come back to these in a bit.)

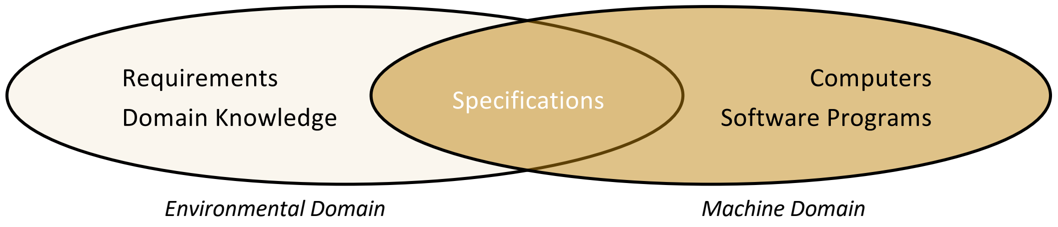
Functional Requirements and Implementation Bias

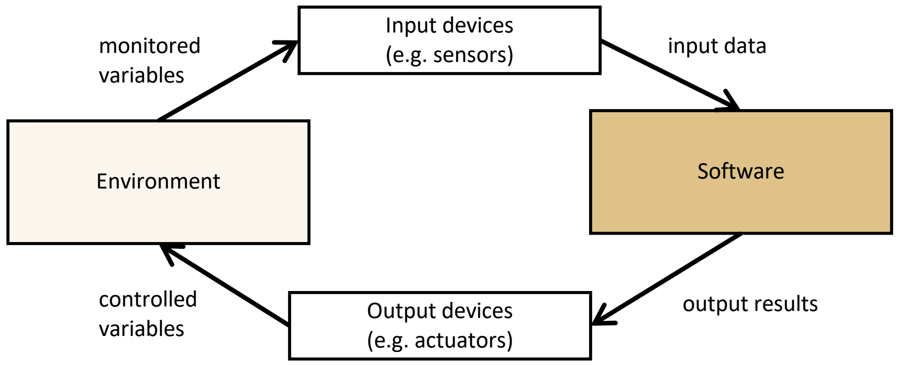
Requirements say what the system will do (**and not how it will do it**).

Why not "how"?

THE WORLD AND THE MACHINE

Environment and the Machine





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?

IMPLEMENTATION BIAS

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

Why not “how”?

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?

QUALITY REQUIREMENTS

Functional Requirements

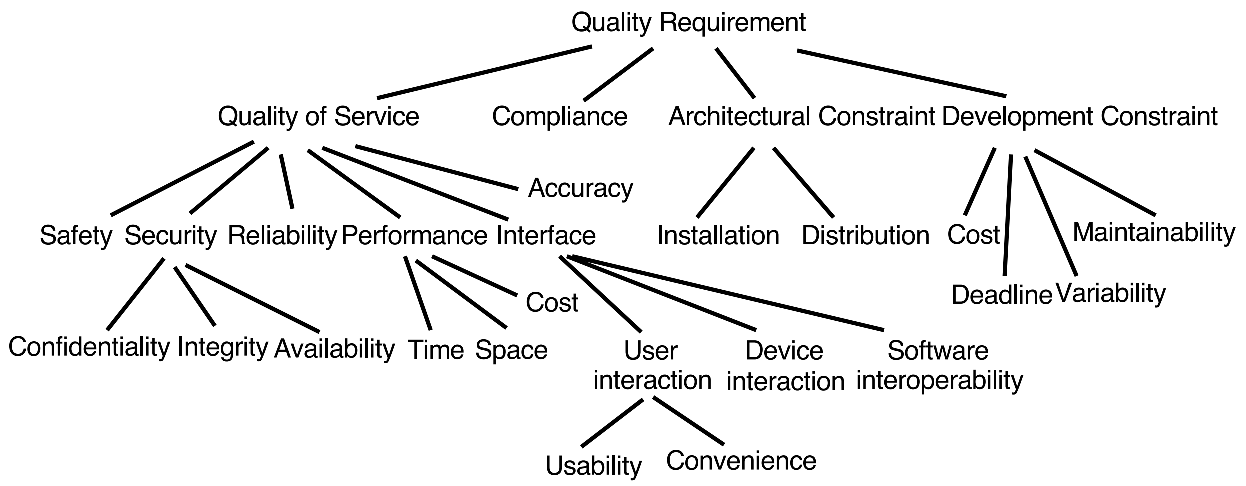
- What the machine should do
  - Input
  - Output
  - Interface
  - Response to events
- Criteria
  - Completeness: All requirements are documented
  - Consistency: No conflicts between requirements
  - Precision: No ambiguity in requirements

Quality/Non Functional 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?



Selling videos on the web?

74

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

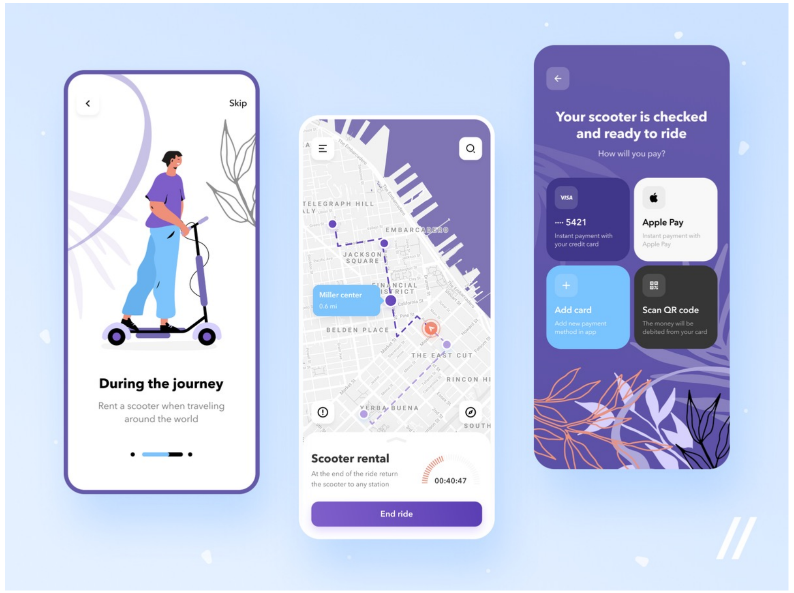
- **Confidentiality requirement**: A non-staff patron may never know which books have been borrowed by others.
- **Privacy requirement**: The diary constraints of a participant may never be disclosed to other invited participants without his or her consent.
- **Integrity req**: The return of book copies shall be encoded correctly and by library staff only.
- **Availability req:** 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.


Skip
During the journey
Rent a scooter when traveling around the world
TELEGRAPH HILL
EMBARCADERO
JACKSON SQUARE
FINANCIAL DISTRICT
Miller center
0.6 mi
BELDEN PLACE
THE EAST CUT
RINCON HI
YERBA BUENA
SOUTH
Scooter rental
At the end of the ride return the scooter to any station
00:40:47
End ride
Your scooter is checked and ready to ride
How will you pay?
VISA
···· 5421
Instant payment with your credit card
Apple Pay
Instant payment with Apple Pay
Add card
Add new payment method in app
Scan QR code
The money will be debited from your card

Requirements Metrics

Property	Measure

ACTIVITIES OF REQUIREMENTS ENGINEERING

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 intervie

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 intervie

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

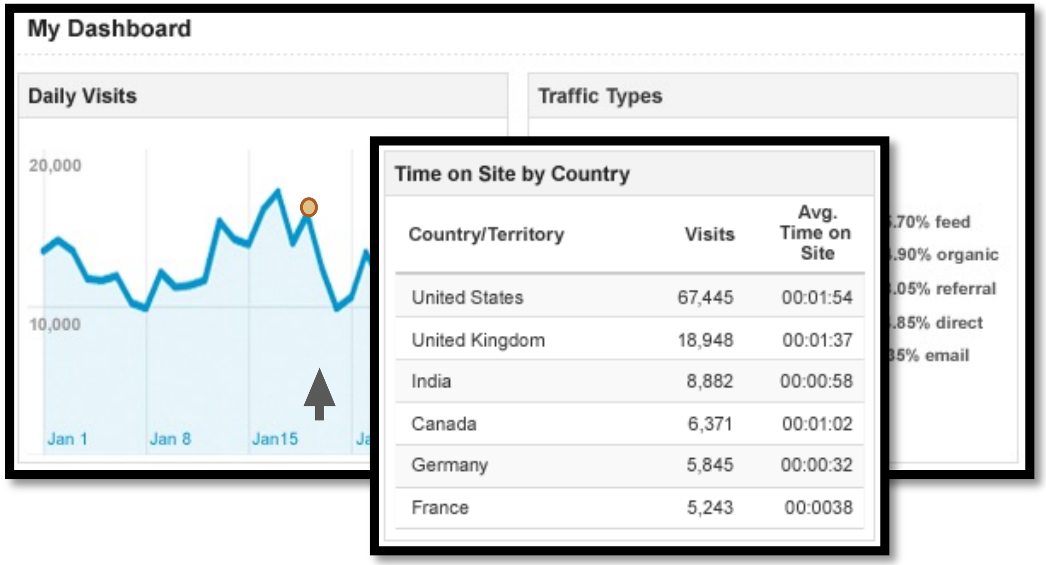
Keep interviewee in the loop

- co-review interview transcript for validation & refinement

Prototypes, Mockups, 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.**

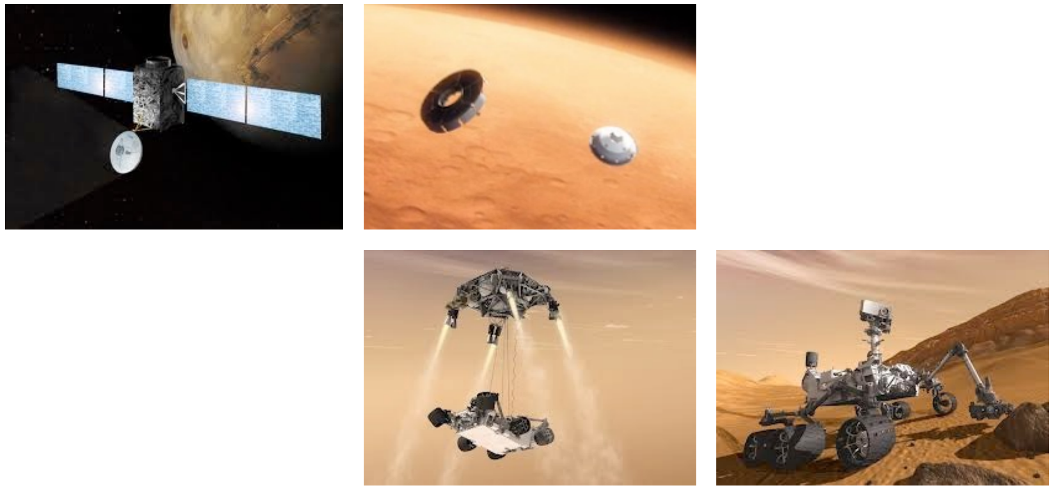
High- vs low- fidelity mockups



Time on Site by Country

Country/Territory	Visits	Avg. Time on Site
United States	67,445	00:01:54
United Kingdom	18,948	00:01:37
India	8,882	00:00:58
Canada	6,371	00:01:02
Germany	5,845	00:00:32
France	5,243	00:0038

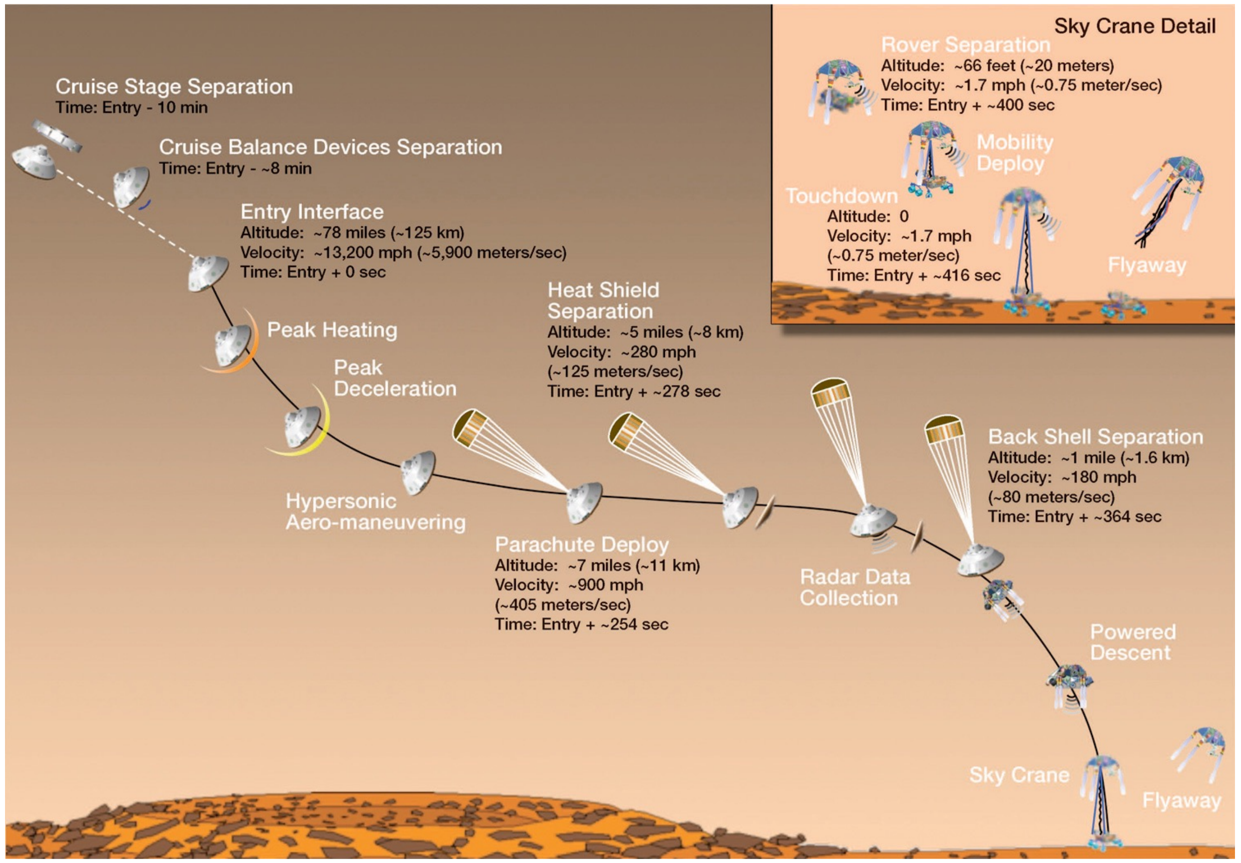
Storyboarding and scenarios



Story

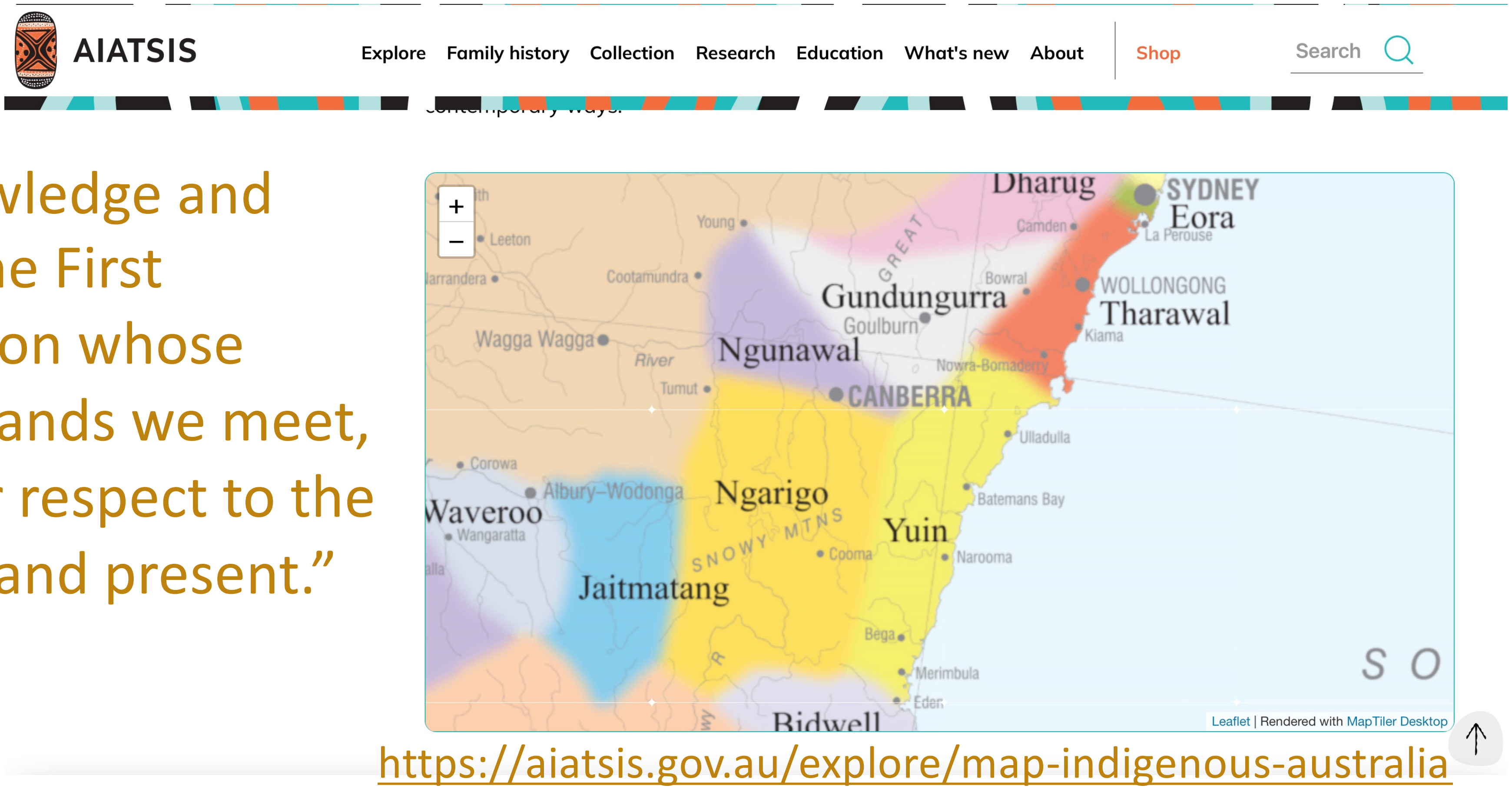
- Who the players are
- What happens to them
- How it happens through specific episode
- Why this happens
- What if such and such an event occurs
- What could go wrong as a consequence

- 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.


Sky Crane Detail
Rover Separation
Altitude: ~66 feet (~20 meters)
Velocity: ~1.7 mph (~0.75 meter/sec)
Time: Entry + ~400 sec
Mobility Deploy
Touchdown
Altitude: 0
Velocity: ~1.7 mph
(~0.75 meter/sec)
Time: Entry + ~416 sec
Flyaway
Cruise Stage Separation
Time: Entry - 10 min
Cruise Balance Devices Separation
Time: Entry - ~8 min
Entry Interface
Altitude: ~78 miles (~125 km)
Velocity: ~13,200 mph (~5,900 meters/sec)
Time: Entry + 0 sec
Peak Heating
Peak Deceleration
Heat Shield Separation
Altitude: ~5 miles (~8 km)
Velocity: ~280 mph
(~125 meters/sec)
Time: Entry + ~278 sec
Hypersonic Aero-maneuvering
Parachute Deploy
Altitude: ~7 miles (~11 km)
Velocity: ~900 mph
(~405 meters/sec)
Time: Entry + ~254 sec
Radar Data Collection
Back Shell Separation
Altitude: ~1 mile (~1.6 km)
Velocity: ~180 mph
(~80 meters/sec)
Time: Entry + ~364 sec
Powered Descent
Sky Crane
Flyaway

End of Monday Lecture/Start of Tuesday Lecture

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

RESOLVING CONFLICTS

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 prioritization

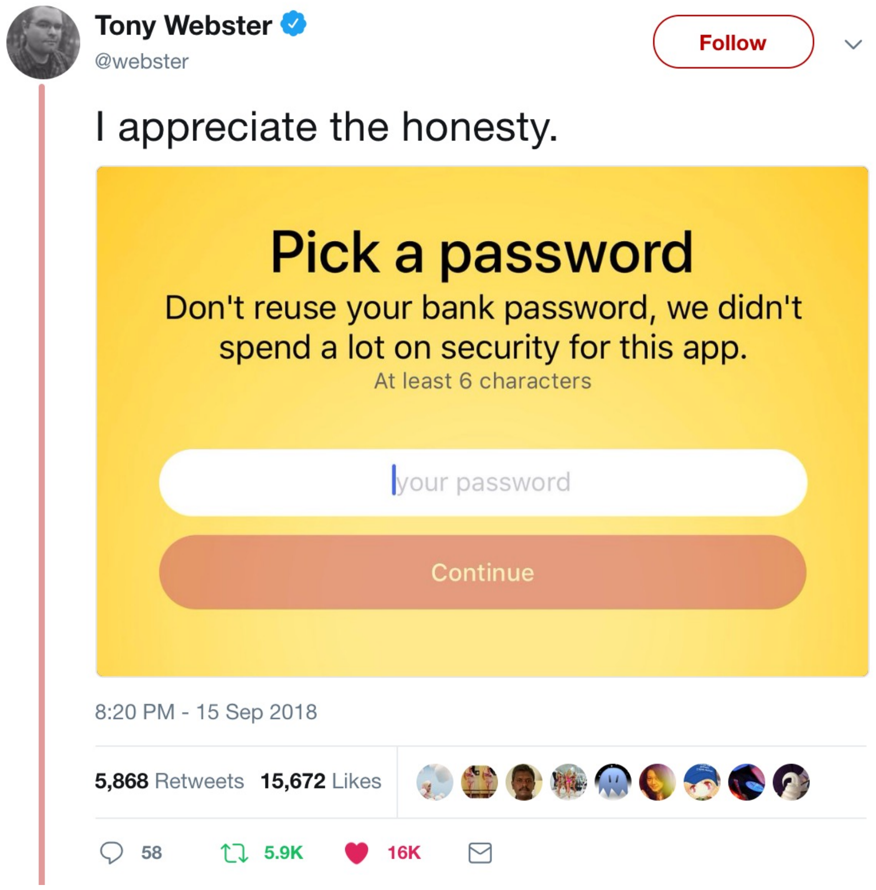
- Cost, time, and other limits
- Dependencies among requirements
- Nice to have

- Strategies to base on value contribution

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*

Risk


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

your password

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

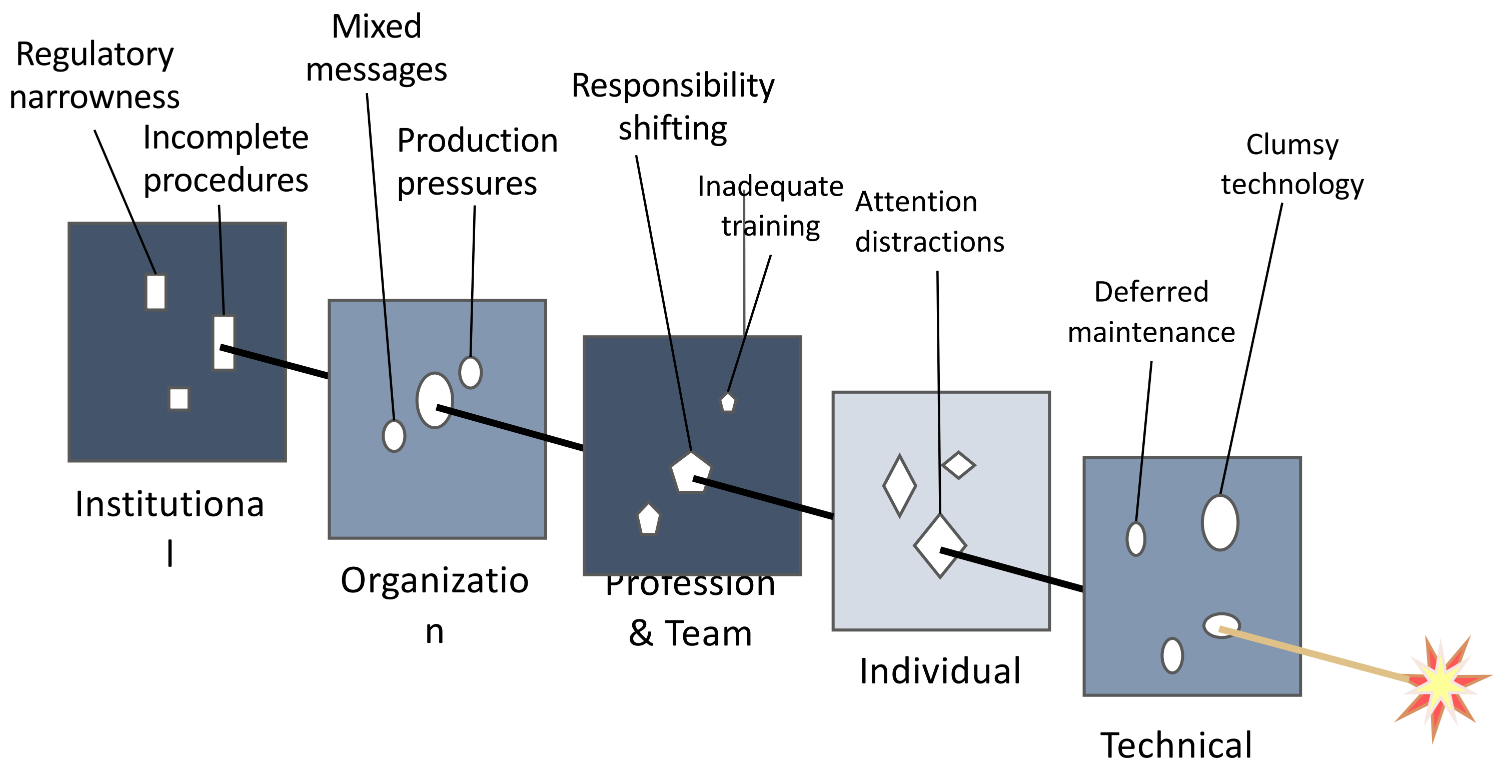
For example…

- 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

The Swiss cheese model



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

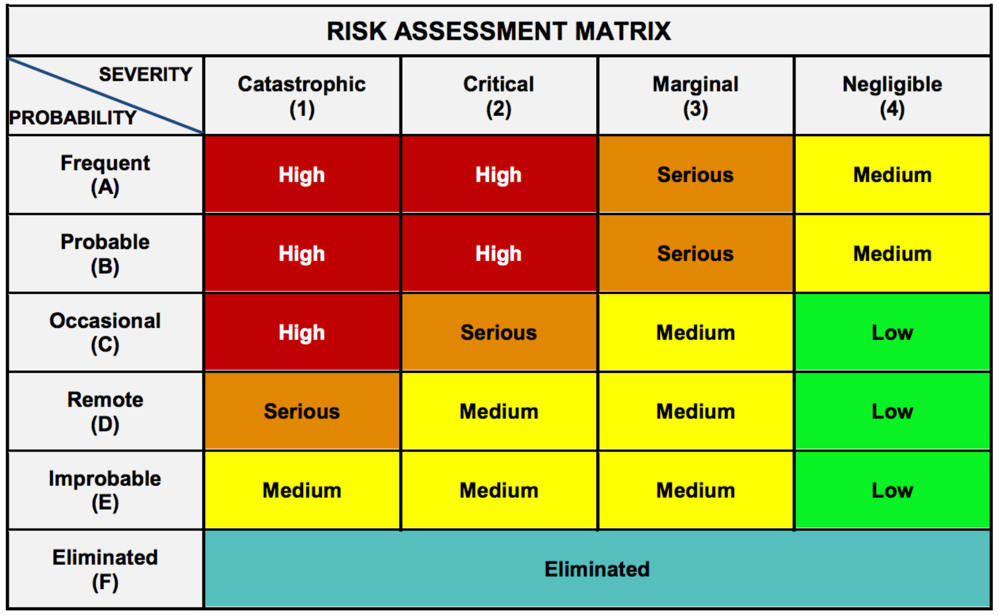
Aviation failure impact categories

- 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

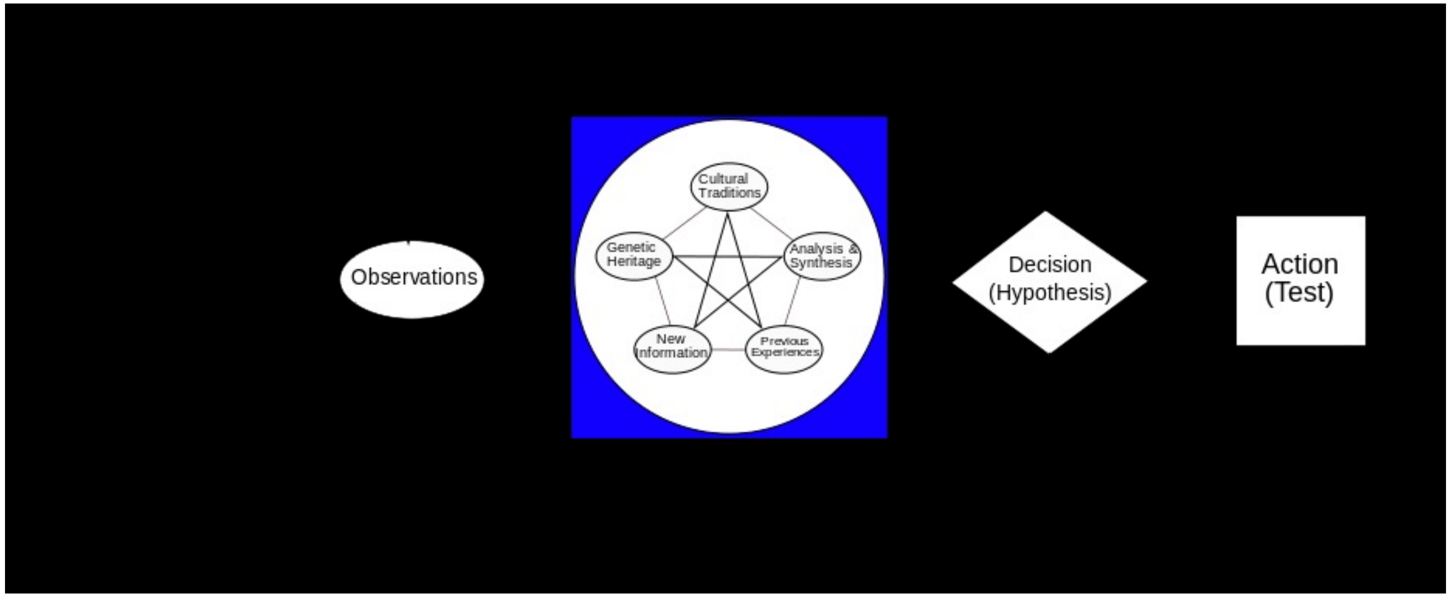
Identify actions needed in order to achieve the chosen option

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

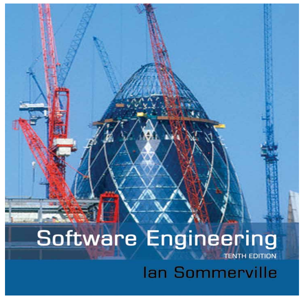
OODA Loop



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

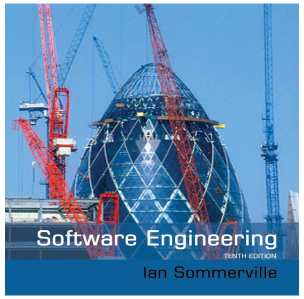
HISTORICAL DETOUR INTO UML (OR BACK INTO THE 1990'S)

Design and implementation



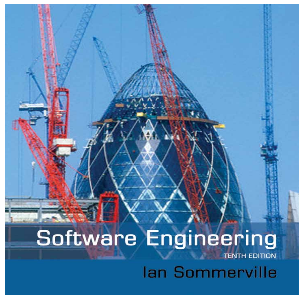
- Software design and implementation is the stage in the software engineering process at which an executable software system is developed.
- Software design and implementation activities are invariably inter-leaved.
  - Software design is a creative activity in which you identify software components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

Build or buy



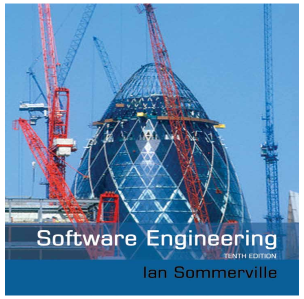
- In a wide range of domains, it is now possible to buy off-the-shelf systems (COTS) that can be adapted and tailored to the users' requirements.
  - For example, if you want to implement a medical records system, you can buy a package that is already used in hospitals. It can be cheaper and faster to 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



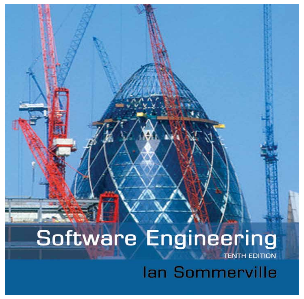
- Structured object-oriented design processes involve developing a number of different system models.
- They require a lot of effort for development and maintenance of these models and, for small systems, this may not be cost-effective.
- However, for large systems developed by different groups design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

Process stages



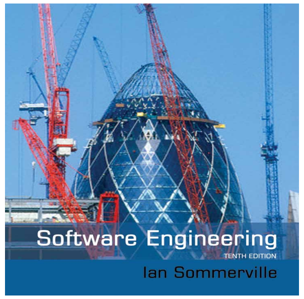
- There are a variety of different object-oriented design processes that depend on the organization using the process.
- Common activities in these processes include:
  - Define the context and modes of use of the system;
  - Design the system architecture;
  - Identify the principal system objects;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

System context and interactions



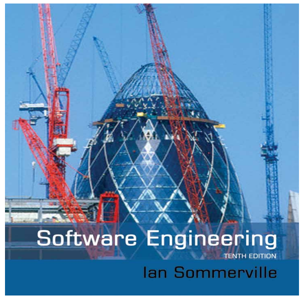
- Understanding the relationships between the software that is being designed and its external environment is essential for deciding how to provide the required system functionality and how to structure the system to communicate with its environment.
- Understanding of the context also lets you establish 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.

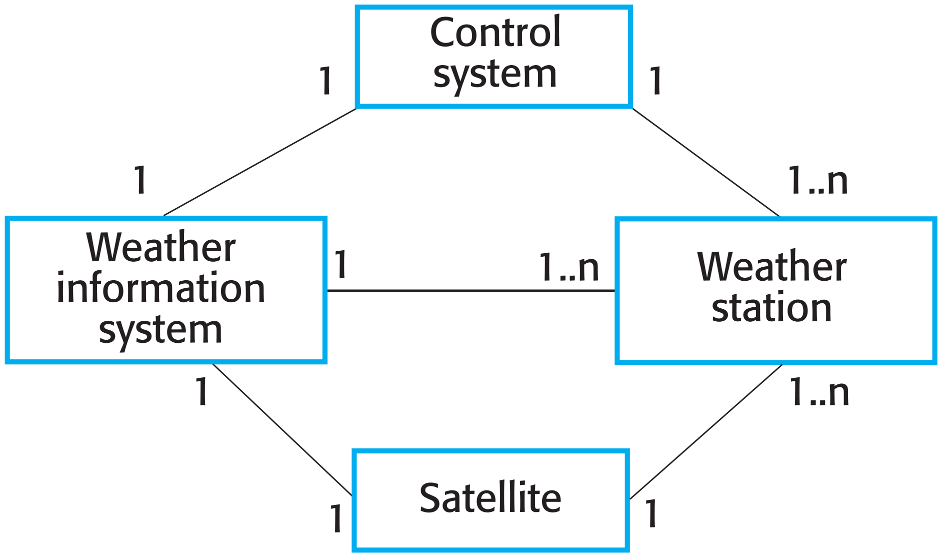
Context and interaction models



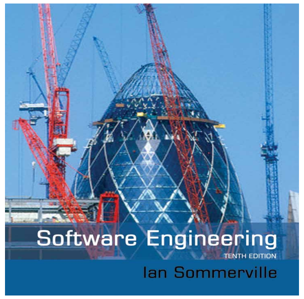
- A system context model is a structural model that demonstrates the other systems in the environment of the system being developed.
- An interaction model is a dynamic model that shows how the system interacts with its environment as it is used.

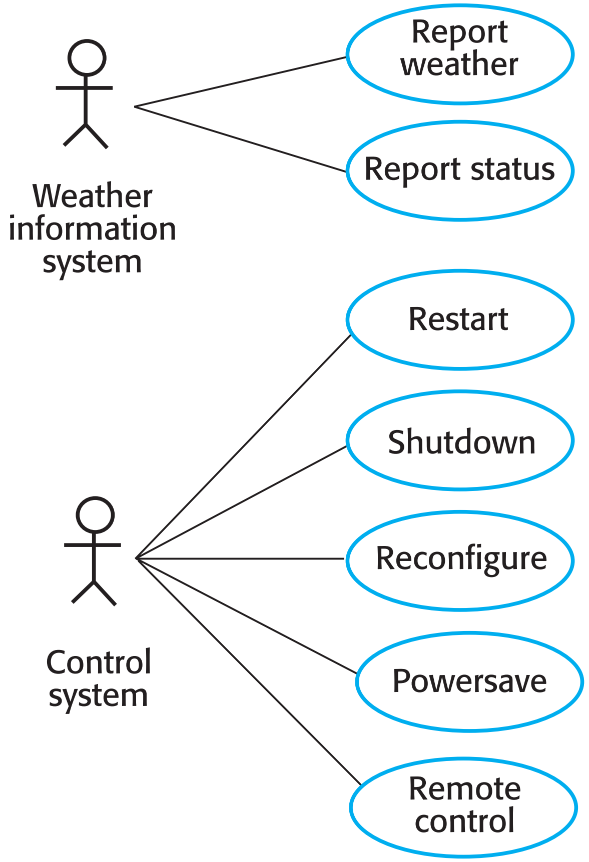
System context for the weather station



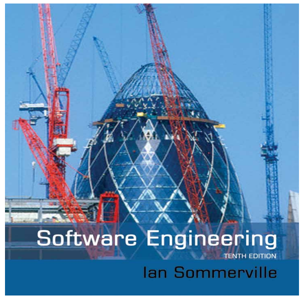


Weather station use cases



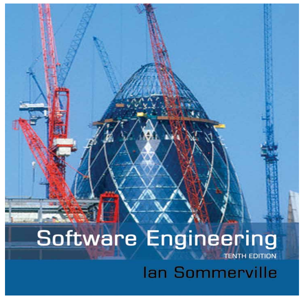


Use case description—Report weather



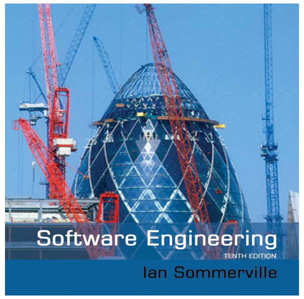
System	Weather station
Use case	Report weather
Actors	Weather information system, Weather station
Description	The weather station sends a summary of the weather data that has been collected from the instruments in the collection period to the weather information system. The data sent are the maximum, 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.

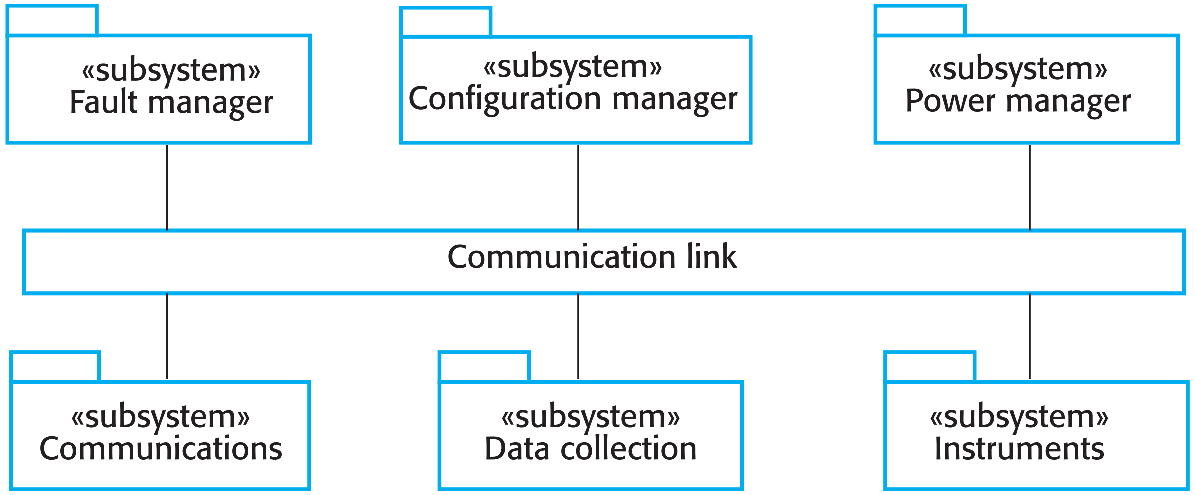
Architectural design



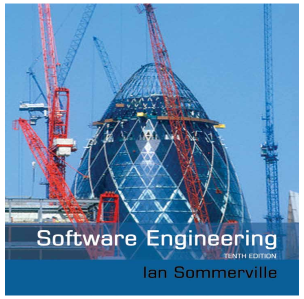
- Once interactions between the system and its environment have been understood, you use this information for designing the system architecture.
- You identify the major components that make up the system and their interactions, and then may organize the components using an architectural pattern such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

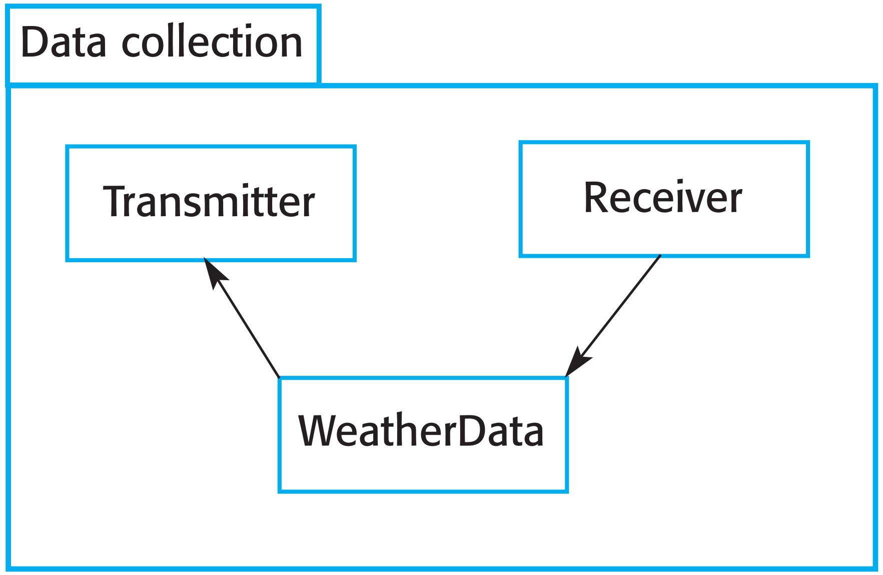
High-level architecture of the weather station



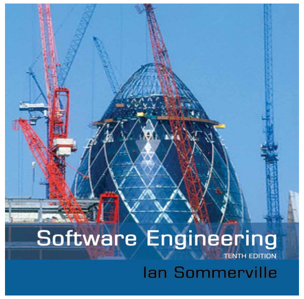

«subsystem» Fault manager
«subsystem» Configuration manager
«subsystem» Power manager
Communication link
«subsystem» Communications
«subsystem» Data collection
«subsystem» Instruments

Architecture of data collection system



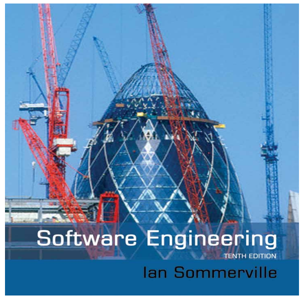


Object class identification



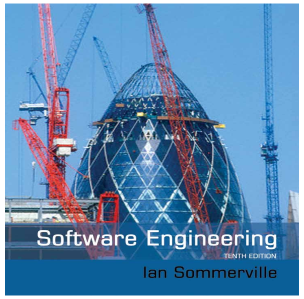
- Identifying object classes is often a difficult part of object oriented design.
- There is no 'magic formula' for object identification. It relies on the skill, experience and domain knowledge of system designers.
- Object identification is an iterative process. You are unlikely to get it right first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

Approaches to identification



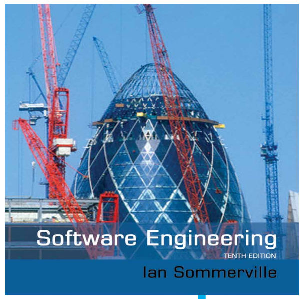
- Use a grammatical approach based on a natural language description of the system.
- Base the identification on tangible things in the application domain.
- Use a behavioural approach and identify objects based on what participates in what behaviour.
- Use a scenario-based analysis. The objects, attributes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

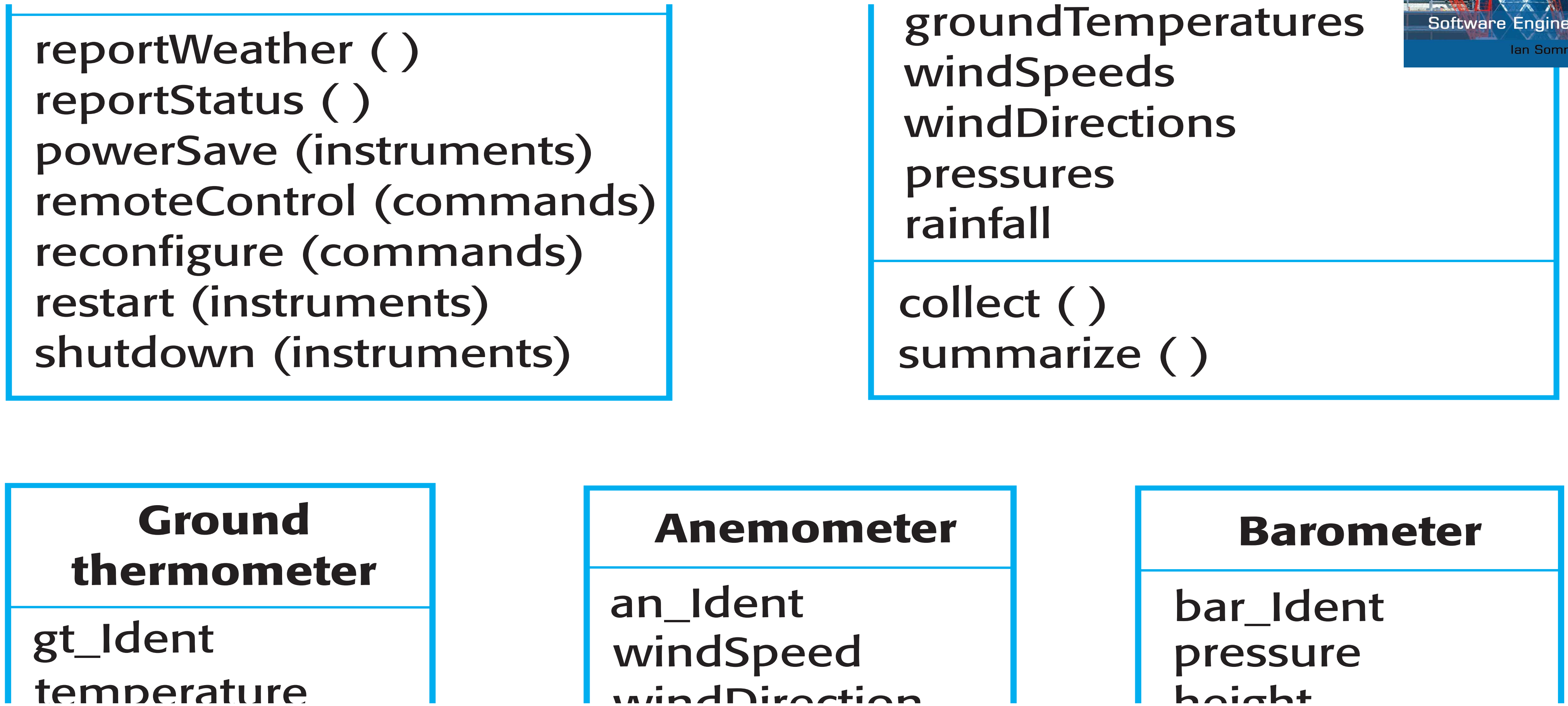
Weather station object classes



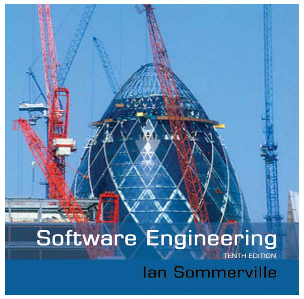
- Object class identification in the weather station system may be based on the tangible hardware and data in the system:
  - Ground thermometer, Anemometer, Barometer
    - Application domain objects that are 'hardware' objects related to the instruments in the system.
  - Weather station
    - The 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

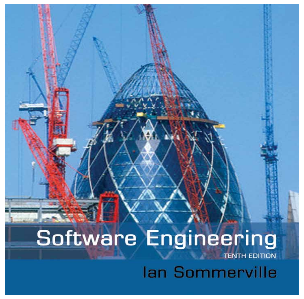




Design models



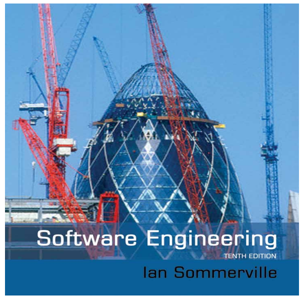
- Design models show the objects and object classes and relationships between these entities.
- There are two kinds of design model:
  - Structural models describe the static structure of the system in terms of object classes and relationships.
  - Dynamic models describe the dynamic interactions between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



Examples of design models

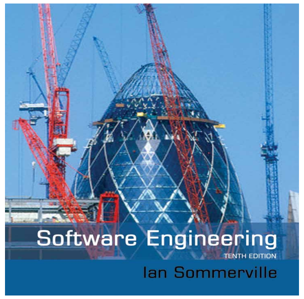
- Subsystem models that show logical groupings of objects into coherent subsystems.
- Sequence models that show the sequence of object interactions.
- State machine models that show how individual objects change their state in response to events.
- Other models include use-case models, aggregation models,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

Subsystem models



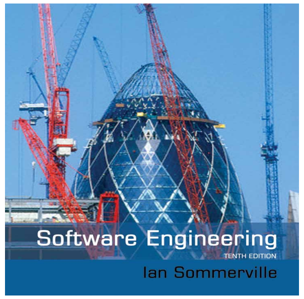
- Shows how the design is organised into logically related groups of objects.
- In the UML, these are shown using packages - an encapsulation construct. This is a logical model. The actual organisation of objects in the system may be different.

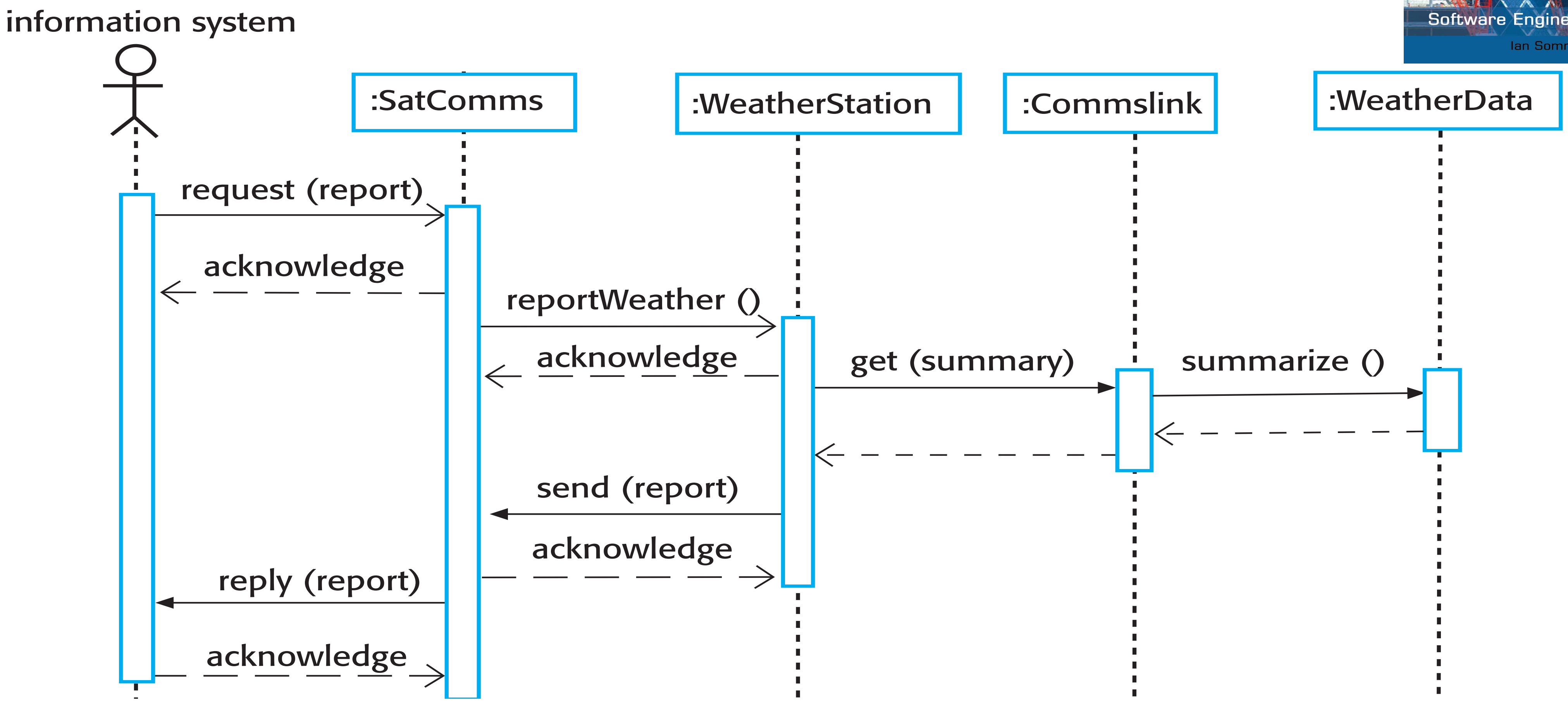
Sequence models



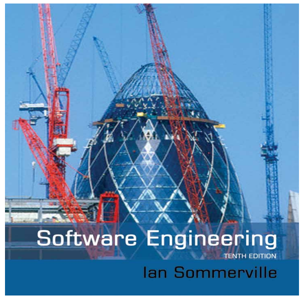
- Sequence models show the sequence of object interactions that take place
  - Objects are arranged horizontally across the top;
  - Time is represented vertically so models are read top to bottom;
  - Interactions are represented by labelled arrows, Different styles of arrow represent different types 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



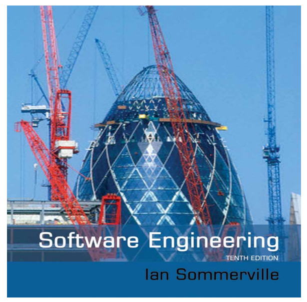


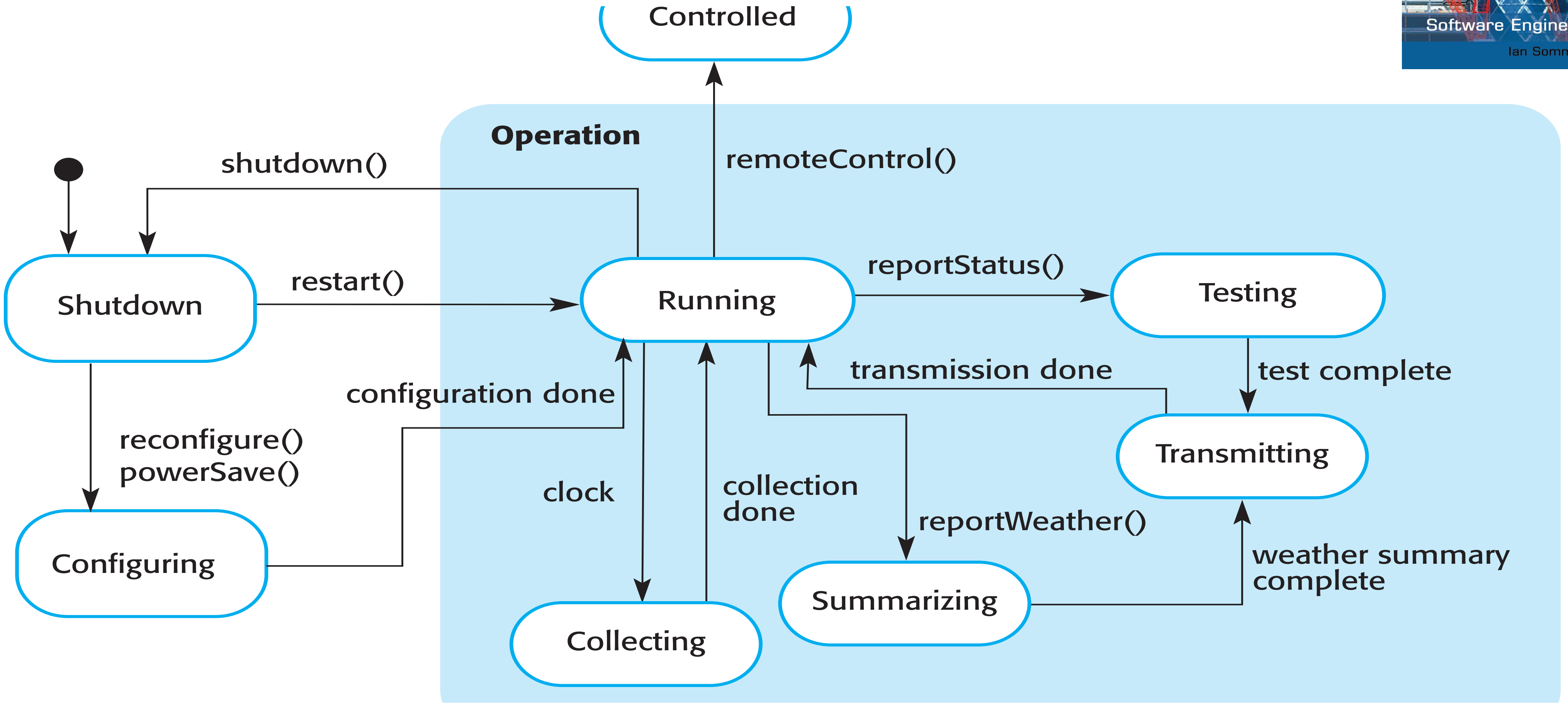
State diagrams



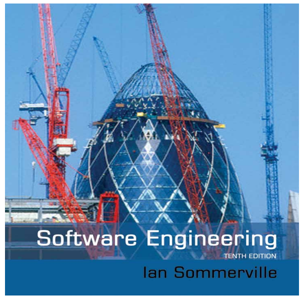
- State diagrams are used to show how objects respond to different service requests and the state transitions triggered by these requests.
- State diagrams are useful high-level models of a system or an object’s run-time behavior.
- You don’t usually need a state diagram for all of the objects in the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



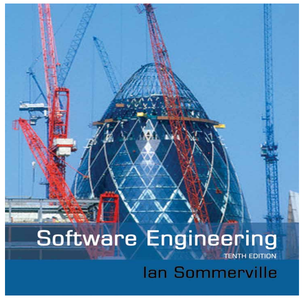


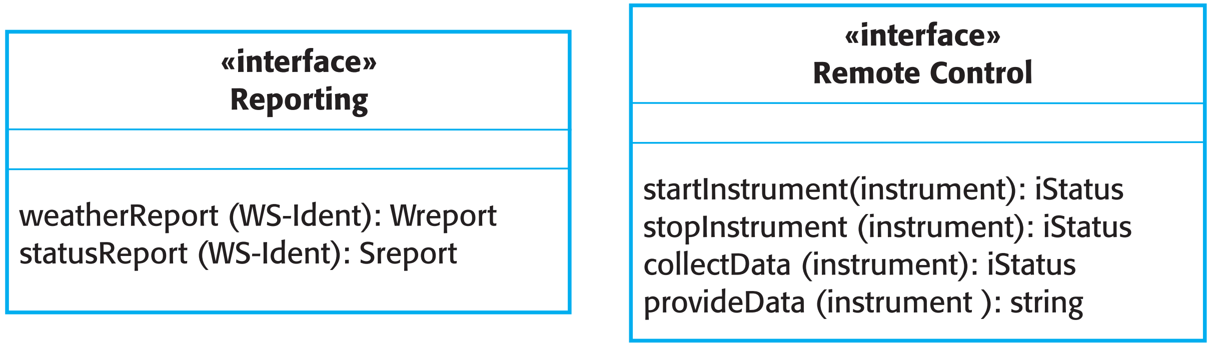
Interface specification



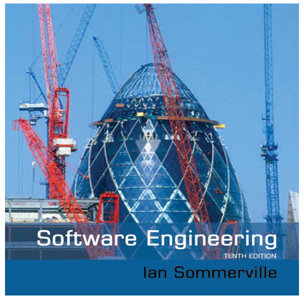
- Object interfaces have to be specified so that the objects and other components can be designed in parallel.
- Designers should avoid designing the interface representation but should hide this in the object itself.
- Objects may have several interfaces which are viewpoints on the methods provided.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

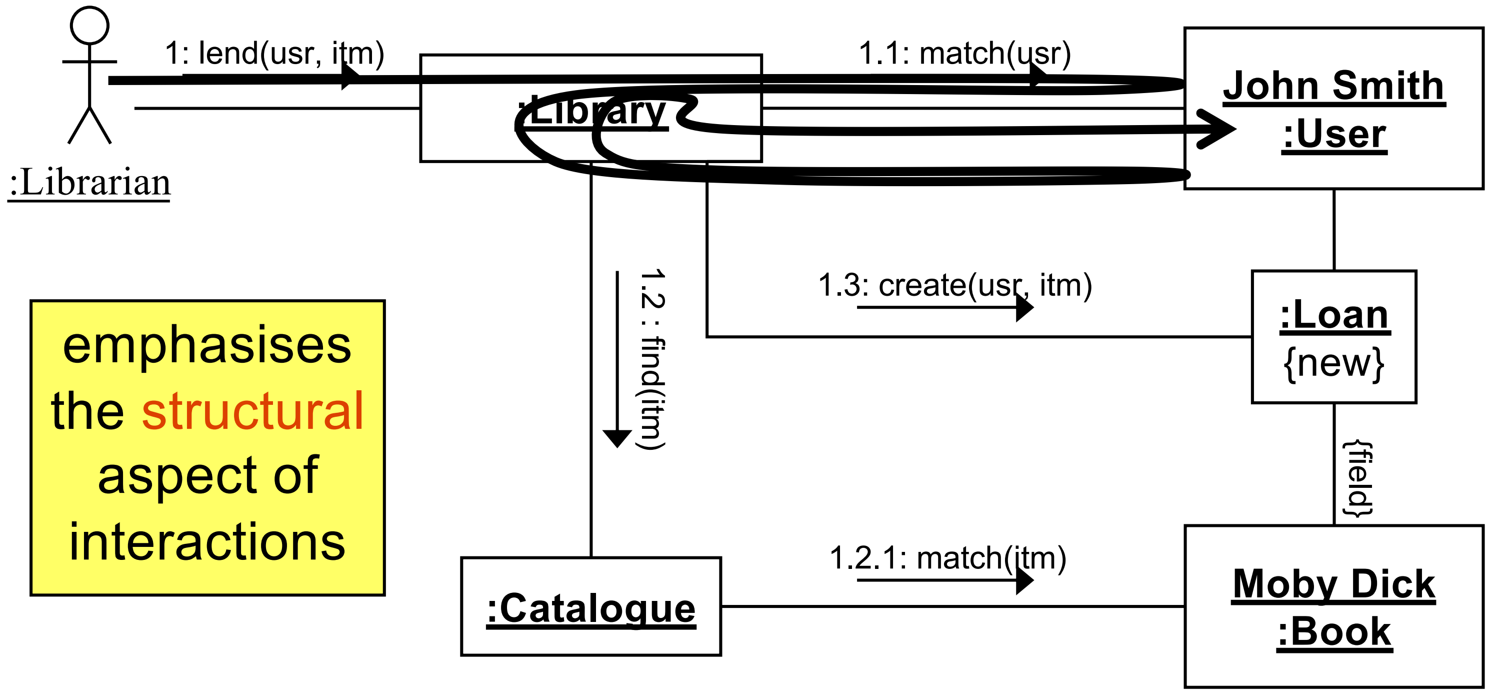
Weather Station Interfaces




«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

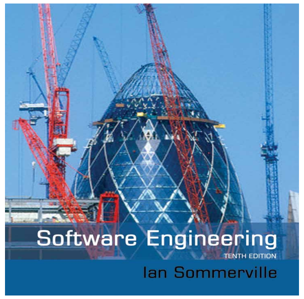
Communication Diagram

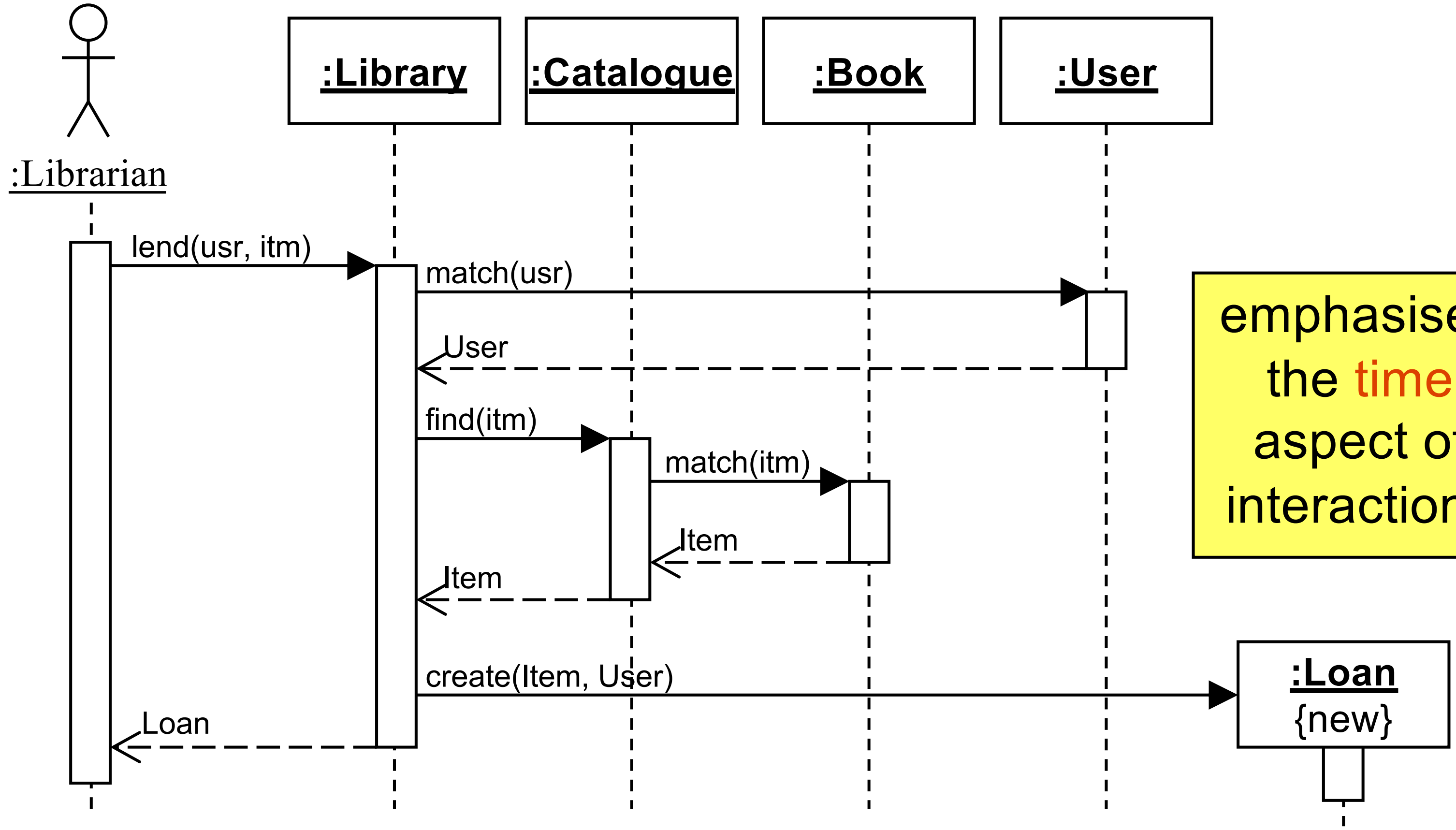




emphasises the structural aspect of interactions

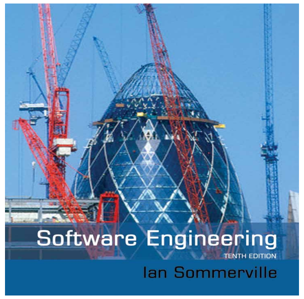
Sequence Diagram



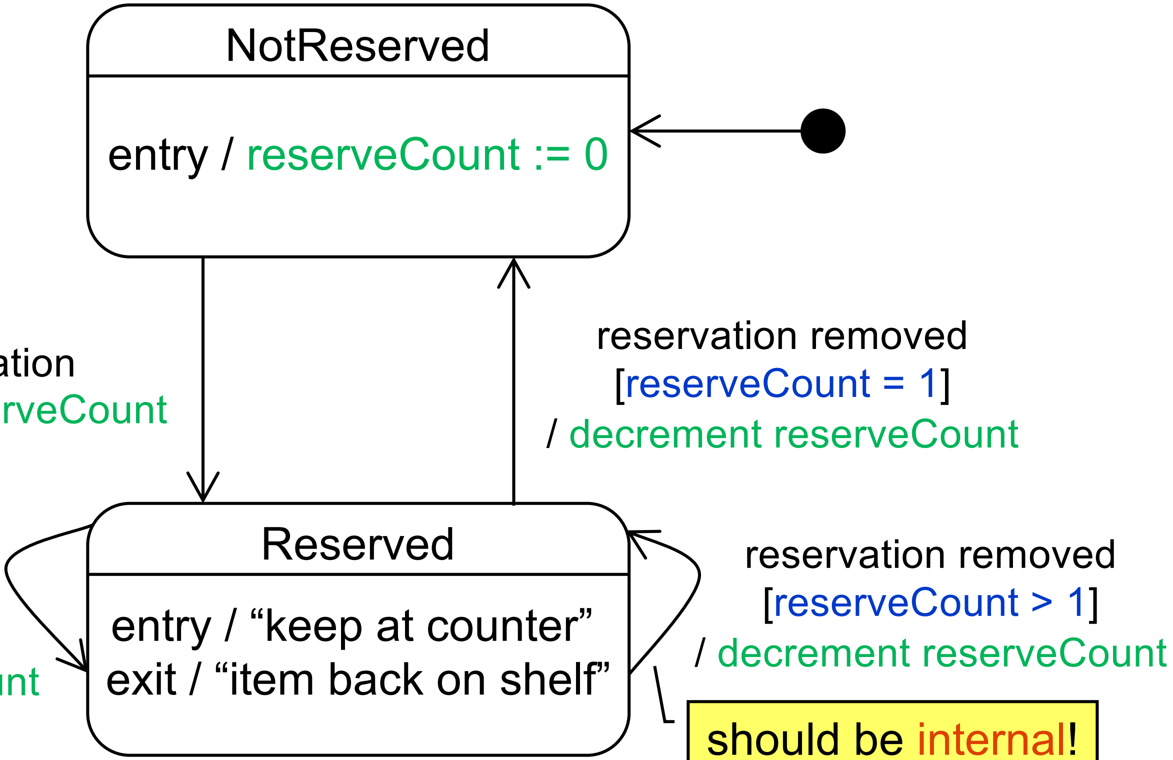


emphasises the time aspect of interactions

State Diagram

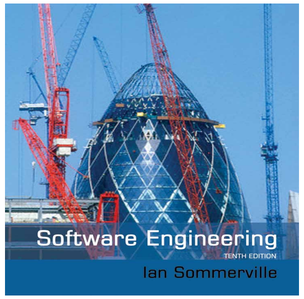


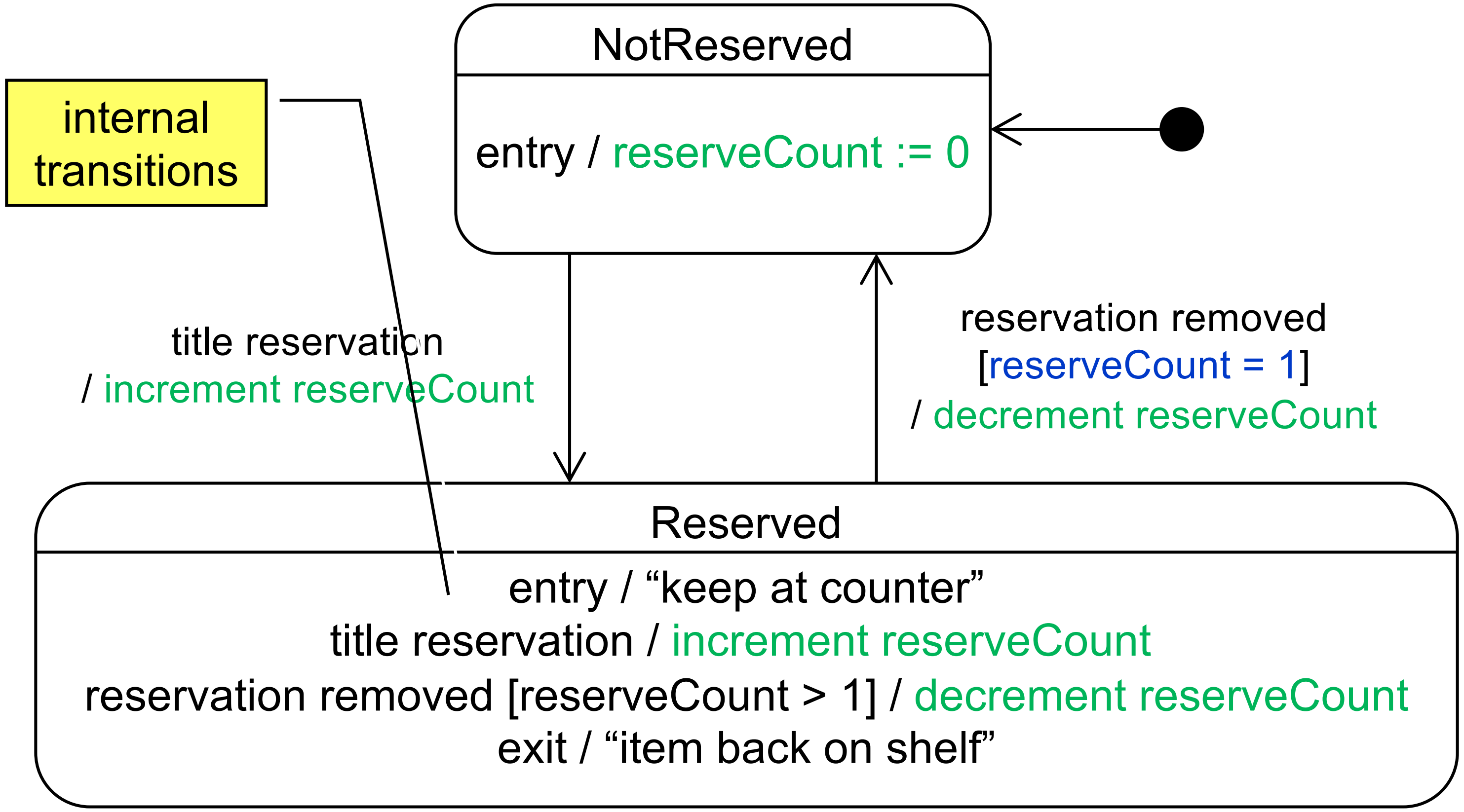
useful for specifying reactive behaviour



should be internal!

State Diagram




internal transitions
NotReserved
entry / reserveCount := 0
title reservation
/ increment reserveCount
reservation removed
[reserveCount = 1]
/ decrement reserveCount
Reserved
entry / "keep at counter"
title reservation / increment reserveCount
reservation removed [reserveCount > 1] / decrement reserveCount
exit / "item back on shelf"

Reflection for the Group Assignments Wattle Quiz

[PROCESS]

At our first meeting, we developed an initial outline of our approach. This was followed by preparing a list of tasks which were required for implementing the X system. Next, we divided the tasks among ourselves and came up with a rough timeline of the process to be followed."

[SCHEDULE]

"Although we managed to meet all the milestones and implement all the desired features, the exact dates for the same could not be followed towards the end."

[PLANNING]

"Learning how to use API X took a little longer than expected, which caused a setback of a day; but overall we managed to complete the entire project before the deadline and adhered to the timeline."

[TEAM WORK AND COMMUNICATION]

"We all agreed to use tool Y to keep in touch. We used it to announce when we started or completed individual tasks, current milestone statuses.. We also used Y to schedule a group meeting for the integration portion of our coding assignment"

Reflection for the Group Assignments Wattle Quiz

[PLANNING / PROCESS]

“Since I was interested in the planning, we decided as a team I would be in charge of documenting our progress.. It worked really well to have one person managing what needed to get done or who needed to do it, and ensuring a shared single vision and set of goals as a group. However, there exist negatives approaching things this way…I found that my teammates sometimes would rely on me too heavily.”

[TEAM WORK AND COMMUNICATION]

“An example of something that [would] work well is...issue tracking – something I asked them to do since first meeting. It’s easy to forget this information over time... If we had simply reminded ourselves on a regular basis, we would have had fewer problems forgetting these things.”

[PLANNING]

“People seemed to be annoyed because X “was not doing any work”. I believe X did the least amount of work, but we also assigned X the least amount of work. I wonder if this can all be traced back to the fact that X could not attend our first group meeting”

Reflection for the Group Assignments Wattle Quiz

[TEAMWORK]

“It helps to say ‘thank you’ before complaining about a teammate’s work. Only take conflict-inducing action if you think it is extremely important and are willing to follow up. Otherwise, you are wasting everyone’s time. Would we have treated each other differently if we had known we would be partnered up on more than just this assignment for the class?”

[TEAMWORK]

“two takeaways I had from this project are :

– It is best to present yourself as someone who is willing to help out, and do more than what was originally asked of you. This way, if people decline your offer to help out, they will be okay with the fact that you may not be working as hard as them at that point in time.

– Respect other people’s time and work, and take that into consideration when you decide to criticize their work or bring up issues. “