Module 1

The software development life cycle, a tour of Agile methodologies, and scenarios for describing use cases

UNIVERSITY of TASMANIA

Outline

- The bigger picture: software engineering
- The software development life cycle
- Refresher: the requirements trace matrix
- Agile software development (a brief tour)
- Scenarios: making use cases concrete
- A foretaste of prototyping in systems design

Reading 1: Software Engineering

Section 1.1 (5 pages) in

Software Engineering: Principles and Practice, 3rd ed, by Hans van Vliet, 2008

Location: accompanying readings PDF

Summary: This optional reading provides some context for the unit as a whole and the notes that follow. Software engineering is an multi-disciplinary problem-solving *process* for creating *non-trivial* software systems.

While all human-created products have a life cycle (from conception through to realisation and ultimate disposal) software is unusual in that it is extremely flexible and, as a consequence, software products can become highly complex very quickly.

The processes employed over the lifecycle of software are intended to manage this complexity so that a reliable and useful product is created.

Reading 2: SDLC

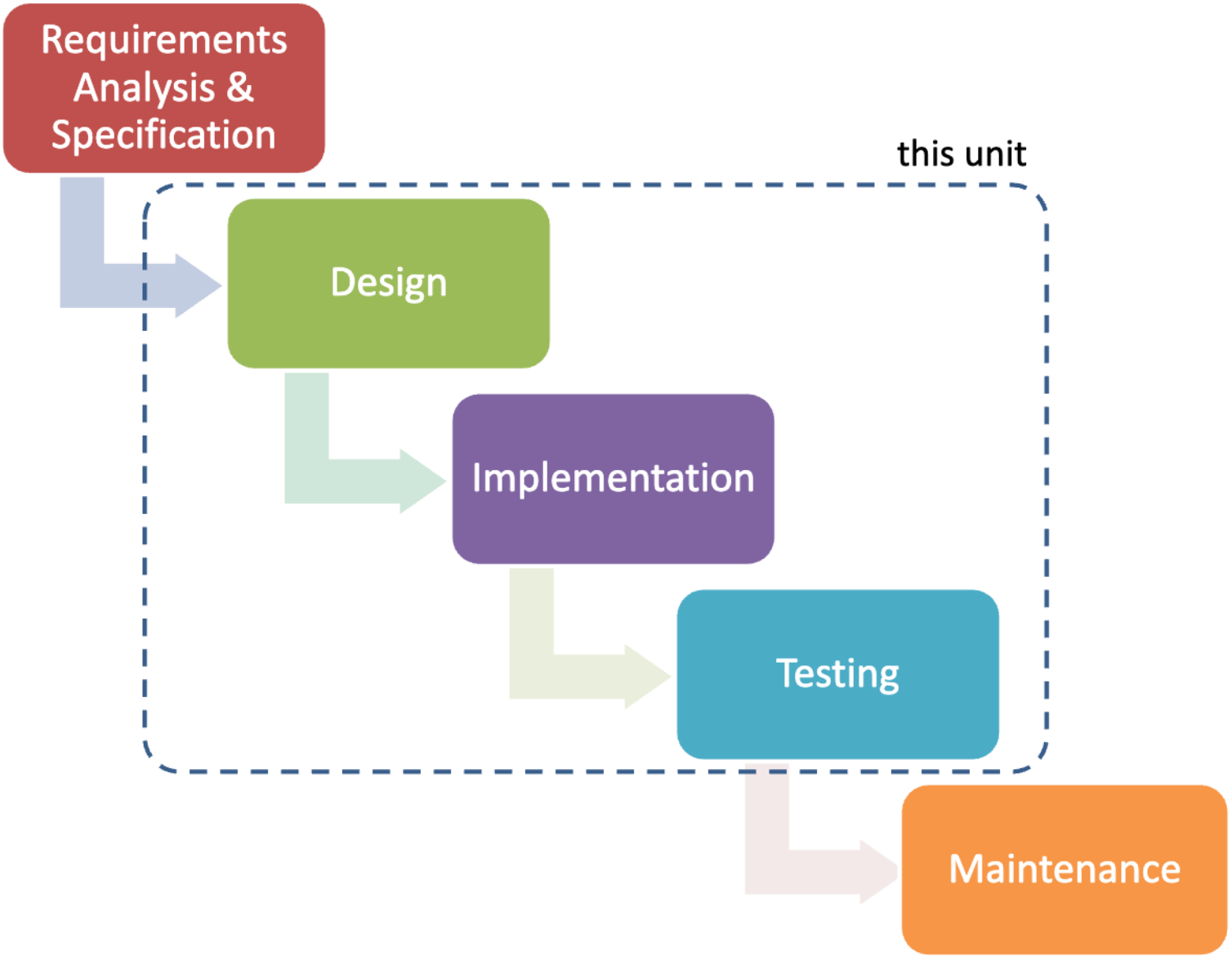
Section 1.2 (5 pages) in

Software Engineering: Principles and Practice, 3rd ed, by Hans van Vliet, 2008

Location: accompanying readings PDF

Summary: The software development life cycle can be thought of in terms of distinct phases (although the reality is rarely so neat). Indeed, different projects and different software development approaches may arrange the phases differently.

Idealised SDLC



Revisiting the Requirements Trace Matrix

If you are already familiar with the requirements trace matrix you may wish to skip to the next section on Agile software development

Requirements validation

Requirements validation is concerned with demonstrating that the requirements define the system that the customer really wants

- One way to validate the requirements is to produce a requirements trace matrix (RTM)
- The RTM is also useful for tracing the requirements to completion

Requirements Trace Matrix

- A requirements trace matrix is simply a spreadsheet (produced in Excel, for instance)
- It has the following columns:
  - Entry # (sequentially allocated)
  - Para # (identifies the location of the requirement in the Requirements doc)
  - Requirement (verbatim text from Requirements doc)
  - Type (see list)
  - Release (which release it is planned to be met by)
  - Use Case

Requirement Trace Matrix

Process used to create a RTM:

1. Agree to a starting set of documents
2. Extract the “must” statements. Start from the beginning of the document, get the entire sentence that has the word “must” in it, give it a sequential number starting from one, and take note of the paragraph that it came from. Put this information in a spreadsheet type format.
3. Extract the “should” statements in the same manner as the “must” statements.
4. Similarly for “could”
5. Extract derived requirements. A derived requirement is a requirement that is deduced from other system knowledge, but not explicitly stated statement.

Categorising requirements

Label	Meaning & usage notes
HW	Hardware Requirement
SW	Software Requirement
NTH	Nice to have (followed by the actual type)
D(#n)	Duplicate of RTM entry #n
SWC	Software Constraint (followed by # of entry to which it applies)
DR	Derived Requirement
PR	Performance Requirement

Identifying use cases in the RTM

The software system is a set of Use Cases

A **Use Case** is a statement of functionality required of the software, expressed in the format:

Actor Action Subject

When a row in the RTM indicates a use case record it using the template

UC##_Actor_Action_Subject

where ## is the RTM entry number and Actor, Action and Subject are as above

Use Cases

- An **Actor** represents a stimulus to the software system. The stimulus can be internal or external.
- An **Action** represents a capability of the software system.
- A **Subject** represents the item acted upon by an Action of the software system.

Actors and Subjects are generally nouns, whereas Actions are generally verbs

Example: Human Resource Information System

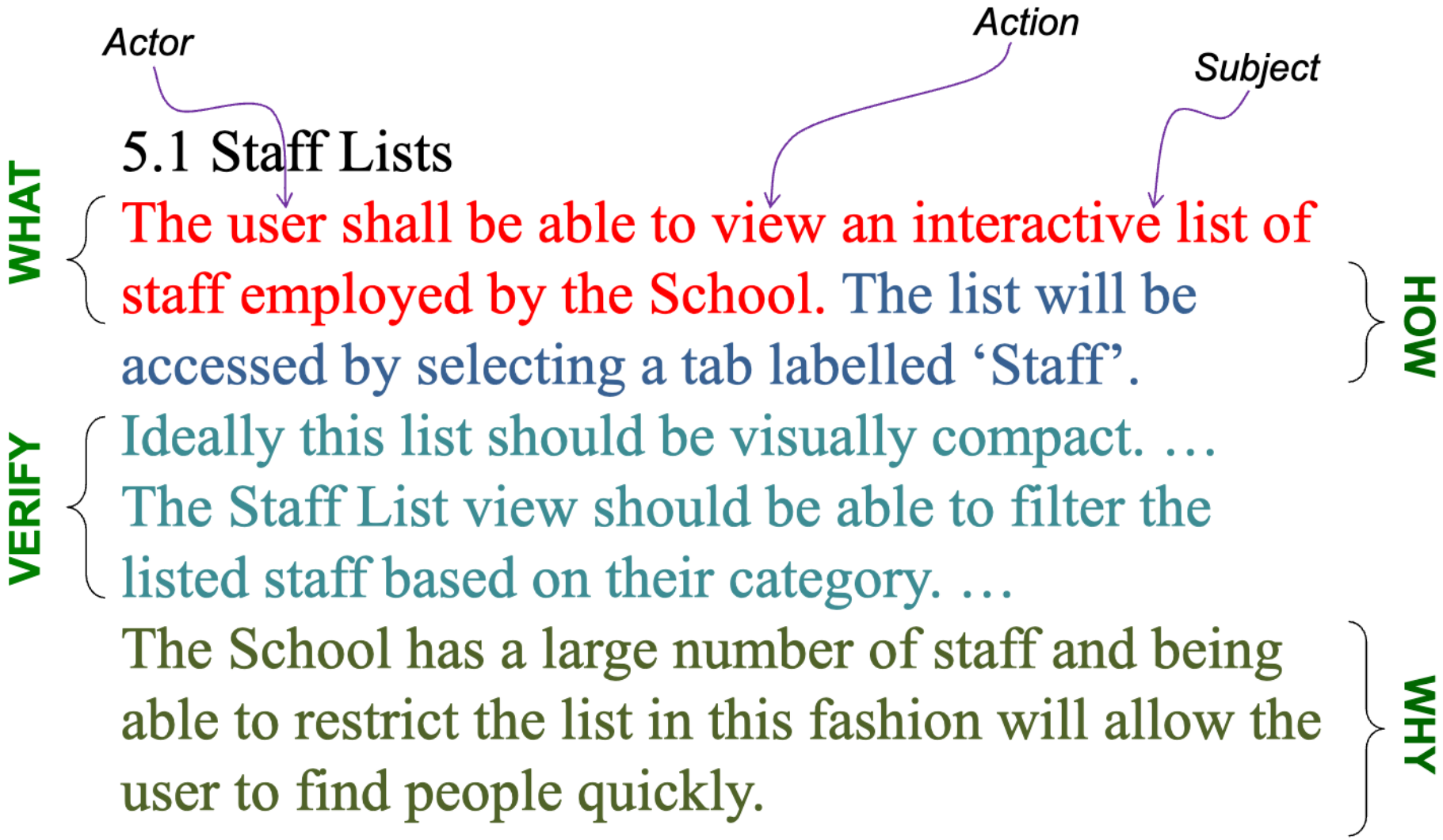
The *Human Resource Information System* was an assignment case study used in a previous year

Examples will be derived from it in this and subsequent modules

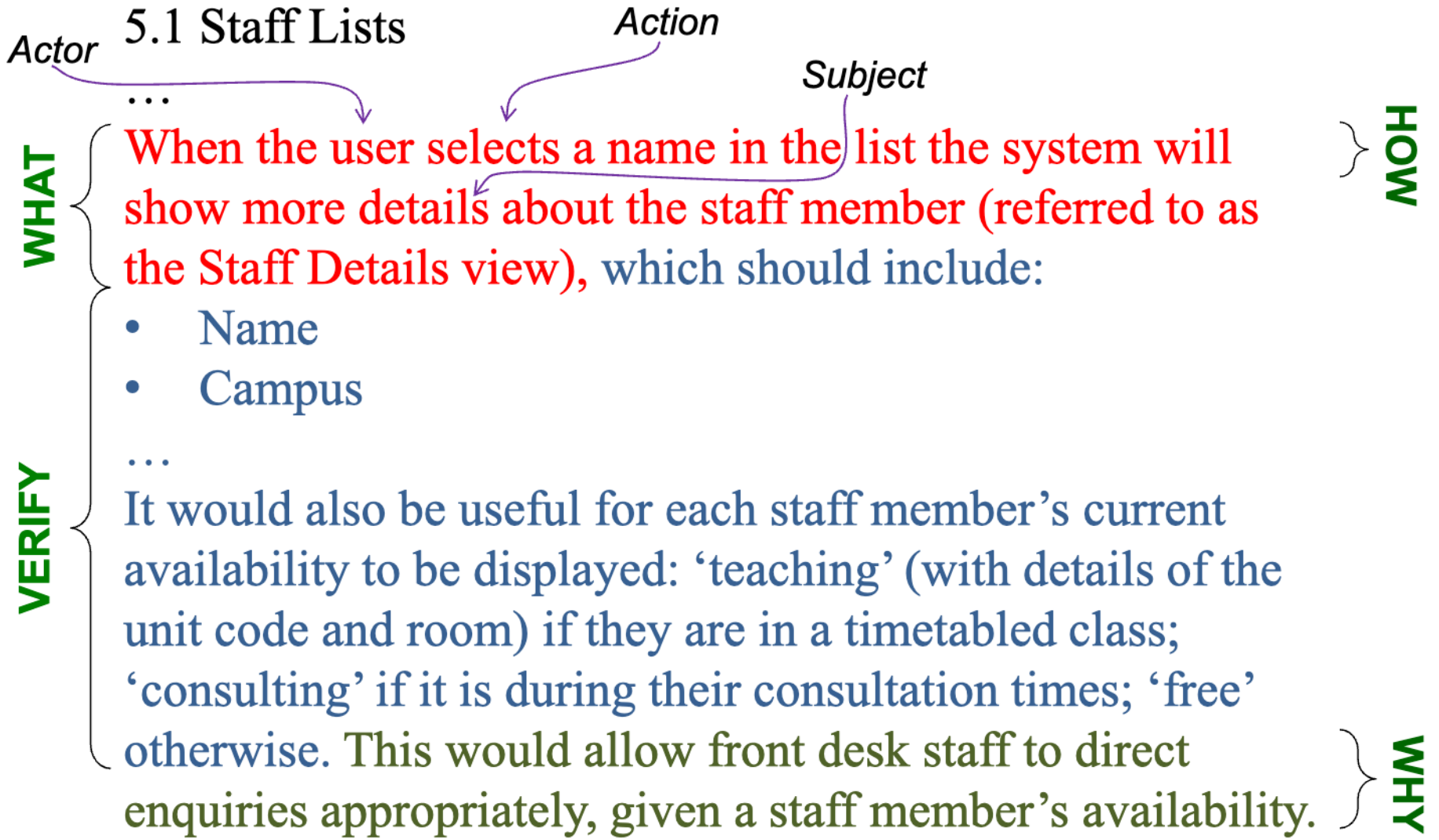
The following two examples relate to the sample Requirements Document and RTM for the HRIS, which can be found in Module 1 on MyLO

Read these two documents now (~3 pages)

A use case from the HRIS



Another use case from the HRIS



RTM Example

Entry	Para	Requirement	Type	Use Case
:				
8	5.1.1	The user shall be able to view an interactive list of staff employed by the School.	SW	UC8_User_views_StaffList
9	5.1.1	The list will be accessed by selecting a tab labelled 'Staff'.	SWC 8	
10	5.1.1	Ideally this list should be visually compact.	SWC 8	
11	5.1.1	When the users selects a name in the list the system will show more details about the staff member (referred to as the Staff Details view), which should include: Name; Campus; Phone Number; Room Location; Email Address; Photo; Consultation hours; Table of units he or she is involved with in the current semester.	SW	UC11_User_selects_StaffDetails
…	…	…	…	…
13	5.1.2	It would also be useful for each staff member's current availability to be displayed: 'teaching' (with details of the unit code and room) if they are in a timetabled class; 'consulting' if it is during their consultation times; 'free' otherwise.	SWC 11	
14	5.1.3	The Staff List view should be able to filter the listed staff based on their category.	SWC 8	
15	5.1.3	The user should be able to list all staff, academic, technical, administrative, and casual.	SWC 8	

This excerpt from the RTM is based on the previous two examples

Agile software development

Reading 3: SDLC, introducing Agile

Chapter 3 up to 3.2.1 (exclusive) (6 pages) in

Software Engineering: Principles and Practice, 3rd ed, by Hans van Vliet, 2008

Location: accompanying readings PDF

Summary: While the Waterfall Model of software development is conceptually simple and appealing it often struggles with the realities of building complex systems, requiring (near) perfect knowledge at the outset and a client who never changes his or her mind. Agile approaches, in addition to using a catchy name, treat software development as a *collaborative* and *adaptive* process. The ethos of Agile is captured in the Agile Manifesto (reproduced on the next page).

The Agile Manifesto

The key values of people who practise Agile are:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

http://agilemanifesto.org

Agile in KIT206/506

Because Agile is fundamentally a philosophy there is no *one true Agile methodology™*, but instead a large number of methodologies that fit with the aims in the Manifesto.

Because the progression of topics in this unit is linear, like the Waterfall Model, you will not practise many of the specific Agile methodologies (such as Scrum), but you will

- experiment briefly with user stories as a means of understanding the requirements of a software system;
- and engage in pair programming when implementing your application.

Reading 4: User stories in Agile

Web page: User Stories and User Story Examples by Mike Cohn (see the 6 headings in the accordion view; click each to reveal)

Location: http://www.mountaingoatsoftware.com/agile/user-stories

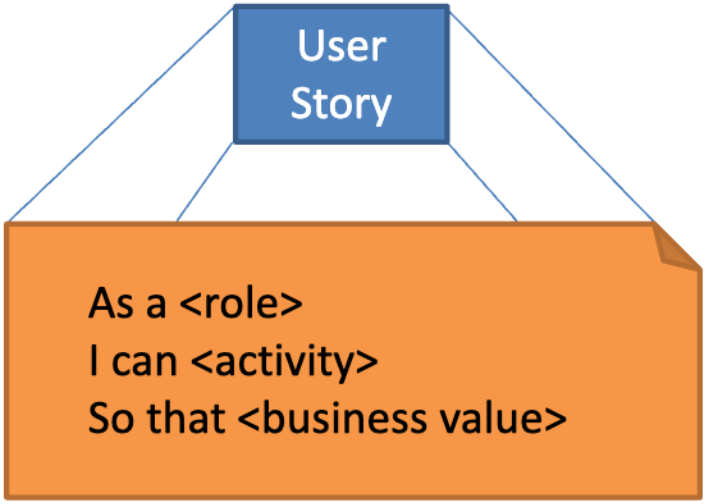
Time: 5 minutes approximately

For context: consider reading first three paragraphs of http://www.mountaingoatsoftware.com/agile/scrum/product-owner

Summary: Agile's user stories serve two purposes: requirements elicitation (which is what the Requirements Specification and RTM capture) and as an ongoing 'to-do' list in the Scrum methodology, which we will not be using.

User stories in Agile

A user story is a brief summary of an action that a user wishes to perform with the system. They're often low-tech, written on small cards.



"As a teacher, I can select a student's entry in a class list to view details of their results"

Optional reading: Use Cases to User Stories

"Mapping Use Cases to User Stories" in

Agile Software Development 2nd ed, Alistair Cockburn, 2006

Location: starts half way down this page
http://proquest.safaribooksonline.com/book/software-engineering-and-development/agile-development/0321482751/agile-and-self-adapting-evolution/226?uicode=utasedu

Time: 2 minutes

Summary: This brief excerpt examines one way to link use cases to Agile user stories. Even if the *process* that you employ is not strictly Agile, this illustrates how a user-centred approach to identifying a user's requirements can help in planning and in the eventual implementation of a system.

Reverse engineer some user stories

Task: Think of some software or a web service you use and different activities you use it for.

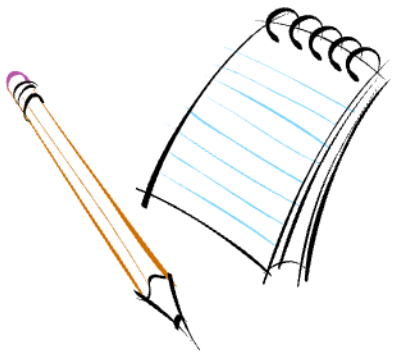
Write down user stories to describe those activities. What is your role? What is the activity? What is the need it fulfils?

(Yes, this is the reverse of a how a user story is normally generated, since the functionality clearly already exists, but it should still be possible.)

Writing your own stories

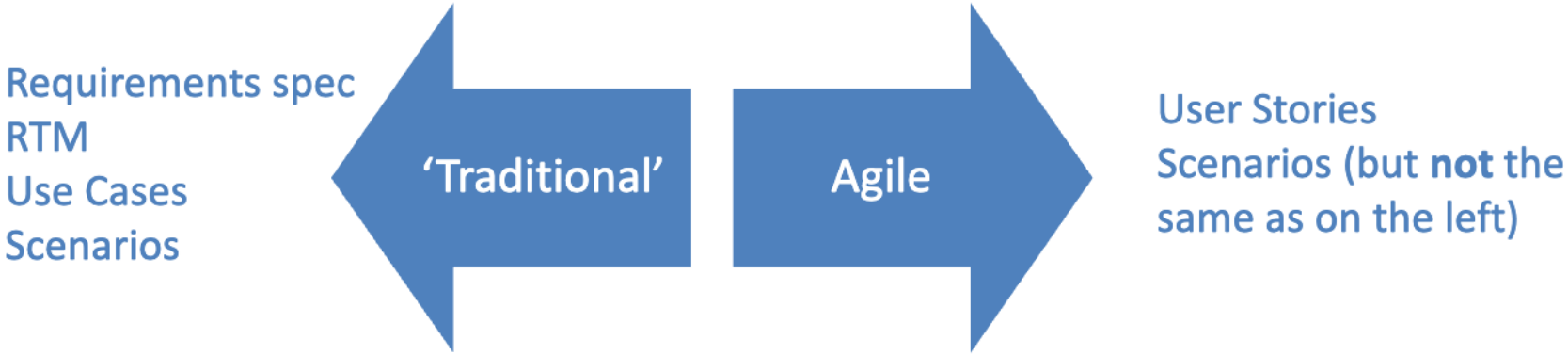
Task: Thinking about the same software or web service, what's some action it doesn't yet support that would add value for you?

Write down at least one user story to describe it



Document-driven versus Agile

There is a tension, bordering on incompatibility, between 'traditional' approaches to capturing and validating requirements and the Agile approaches. We cover Agile all too briefly here (and in greater depth in other units) so that you are aware of some of the differences.



Which approach you use will depend on the system you are building, the client and your development team.

Regardless of the *name* of the methodology you use, we recommend that you take a *user-centred* approach, as promoted by Agile.

Scenarios

…but not of the Agile variety

Scenarios

A **scenario** is a formatted description of the steps required for the completion of a Use Case.

A template is available with this module on MyLO.

Note that the Action – Software Reaction table is the most important section

Generating scenarios

Given the RTM,

complete the scenario template for each use case,

- One for every requirement given type SW in RTM
- Only one scenario per use case (even if tempting to create more than one)

prototype the interfaces for each scenario, and

develop OO Diagrams from each scenario (later in the unit)

Template scenario p 1/4

Use Case #: Title

Requirement:

<The text from requirement column of RTM>

Overview:

<Description of what is happening in use case>

Preconditions:

<List numerically the assumptions required before this Use Case can be executed.>

Scenario:

Action	Software Reaction
1. <Specify an Action>	1. <Describe the software reaction>
2. <Specify an Action>	1. <Describe the software reaction> 2. <second step of software reaction>

Scenario Notes:

<Indicate the currency of Actions, any additional information, such as optional steps and branching and iteration steps>

Post Conditions:

<List sequentially the conditions expected at the completion of the scenario>

Required GUIs:

<List the names of the GUIs utilised by this scenario>

Template scenario p 4/4

Exceptions:

<List sequentially any failure conditions that can affect the Scenarios, and how the system should respond>

Use Cases utilised:

<List other Use Cases used>

Timing Constraints:

<Specify any timing constraints for the Use Case or portion of the Use Case>

We will begin working on scenarios in the first tutorial

Reading 5: Usability

Usability 101: Introduction to Usability,
by Jakob Nielsen
(about 3 pages)

Location:
https://www.nngroup.com/articles/usability-101-introduction-to-usability/

Summary: Usable systems are *learnable* (without referring to a manual), *efficient*, *memorable* (easy to pick up again later), *handle errors well*, and ideally lead to a *satisfied user*.

Although we won't be undertaking participatory studies to evaluate the usability of the application you develop, it is worth familiarising yourself with these common principles of usability systems. As you sketch prototype UIs, and specify the sequence of actions a user will take in each scenario, keep these principles in mind.

User Interfaces and Interface Prototyping

Clients' (users'/players') satisfaction with the system will predominantly be **determined by the user interface**

- So you should always prototype your interfaces
- Divide the system into a number of meaningful interfaces
- Develop interface prototypes that contain useful, meaningful content that helps envisage the final system (these can be done on paper)
- Seek feedback from end users and client

Interface Prototypes

- Develop diagrams of the UI that have a consistent artistic style across the set of interfaces and use appropriate UI & design principles
- Clearly explain each interface and how the user interacts with each item
- Clearly explain how the user navigates between the different interfaces

Interfaces (for Scenarios)

As you've seen in the template:

- In a scenario every screen is given a name
- Every button, text field, etc. is mentioned
- Draw a draft of each GUI identified as being required by the scenarios
- Produce executable prototypes of each GUI in implementation language (optional in this unit as we are yet to cover C# development)

A brief word on prototyping in Agile & other methodologies

Reading 6: Prototyping

Section 3.2.1 (4 pages); you may optionally continue to the end of 3.2 (9 pages) to complete your introduction to Agile.

in *Software Engineering: Principles and Practice*, 3rd ed, by Hans van Vliet, 2008

Location: accompanying readings PDF

Summary: Prototyping is a key part of the iterative process of Agile and maintains the approach's focus on the user and their needs.

UI prototyping (to identify a good UI design rather than to elicit requirements) is a related but separate topic, and is part of the process of creating (non-Agile) structured scenarios.

Next…

The next module will look at object-oriented modelling and design, including using the Unified Modelling Language's package, class and sequence diagrams