Prifysgol Abertawe
Swansea University

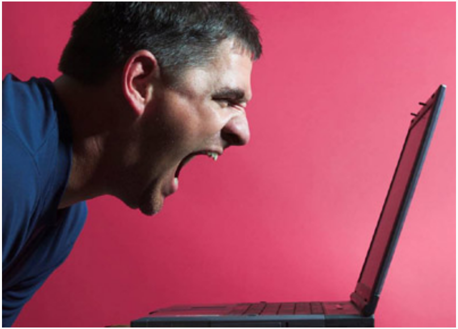
CS-230 Software Engineering

L02: Requirements and UML

Dr. Liam O'Reilly

Semester 1 – 2020

Previously in CS230...



AAAAHHHHHHHHHHHH! Rrrrrrgggghhh!

Previously in CS 230...

- *Software is in Crisis!*
- Rockets explode!
  - All because of an arithmetic overflow condition...
- The software crisis is defined as:
  - impact of increase in computational power and problem complexity.
  - makes large programs difficult to coordinate and write.

Previously in CS 230... (2)

- Why so many software problems and few hardware ones?
- An advantage/disadvantage of software is its mutability.
- Cost of an upgrade:
  - Bugs in HW = Returns of HW (substantial loss of revenue).
  - Bugs in SW = Wait for an Upgrade (little loss in revenue).

Previously in CS 230... (3)

- Various methodologies have been developed.
- Is there a “silver bullet” solution?
  - No – no single approach prevents overruns and failures in all cases
  - projects are large, complicated, poorly-specified, involve unfamiliar aspects, and vulnerable to large, unanticipated problems

Previously in CS 230... (4)

- Useful methodologies involve some of the following:
  - Software Specification, (requirements, customer-oriented planning)
  - Software Design (developer-oriented planning, organisation)
  - Software Development (implementation)
  - Software Validation (testing)
  - Documentation
  - Software Maintenance/Evolution

Previously in CS 230... (5)

- How can we tame the crisis?
- How can we engineer large software projects?

Software Life Cycles

A (Bad) Software Engineering Model

- How not to do it:
  1. Start coding.
  2. Figure out what you are really coding.
  3. Swear (This step is optional).
  4. Change code to fit new mental model.
- Iterative process that involves many frustrating passes.
- For the first time, you may not be able to hold the entire program in your head.

Module Consideration

This module has two group courseworks.

The coursework is large. Thus, I need to cover material for the coursework early on.

- Software Life Cycle Models have to wait.

Requirements

What are Requirements?

- User Requirements:
  - Statements in natural language of user expectations of system.
  - *"The system should provide an overview of the total purchases made for each weekly time period."*
- System Requirements:
  - Descriptions in natural language of functions, services, and operational constraints.
  - *"This function should return a correct result in less than 600ms."*
- Requirements form part of a contract (sign off procedure).
- Can be useful if legal issues arise.

Good Requirements Are...

- Valid: They are what the customer needs.
- Consistent: No conflict.
- Complete: Nothing is missing.
- Realistic: Can be implemented:
    - with the available technology available.
    - with reasonable costs.
    - and be verifiable.

Requirements Document

- Requirements Document is an official statement of all requirements.
    - Includes both user and system requirements.
    - Should be as detailed as possible.
    - Can be useful to anyone involved in the project.
- Not a design document.
    - Focus is on understanding the problem formally.
    - Not a particular approach to the problem.
    - Focus what and not how.

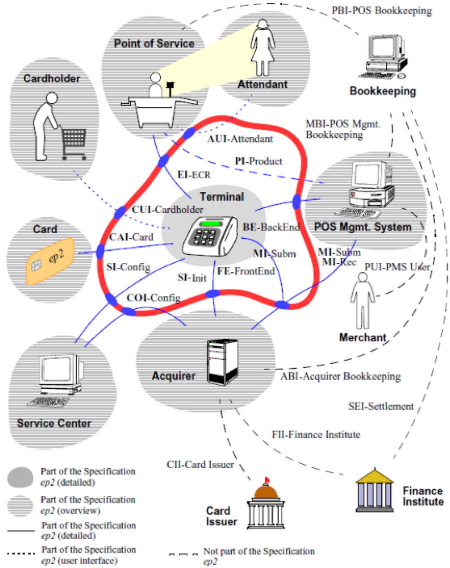
Functional and Non-Functional Requirements

- Functional Requirements:
  - What the system should do.
  - Reaction to specific scenarios & data specifications.
    - Parameter N of Fact(N) should be ≥ 0.
    - Student numbers should have 6 digits.
- Non-Functional Requirements:
  - Global statements on the system.
  - Not directly concerned with specific services to users.
    - System should be secure.
    - The cost of the system will be less than £X.
    - Must adhere to ISO standard...
- Not always clear cut distinction.

Language of Requirements

- Often requirements use the language of another domain.
  - Aerospace engineering, business, social sciences...
- We often need to agree/learn a language in order to communicate.
- We need to make sure nothing is misunderstood.
  - Clear natural language.
  - Supported by diagrams, tables, and mathematical notations.
  - Choice of development model has implications...

Language of Requirements (2)

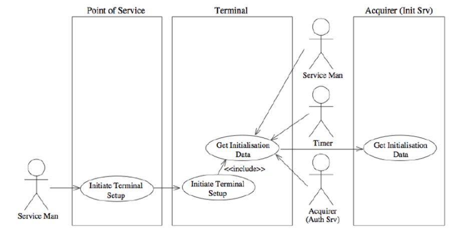


How do we specify?

- Images
- Plain English:
  - 'The interface is used to download configuration data, terminal software and some initialisation data'.

Language of Requirements (3)

- Various UML Diagrams, such as use case diagrams:



- Propositional Logic:

$$SingleAspect \equiv (tla\_g \vee tla\_r) \wedge \neg(tla\_g \wedge tla\_r).$$

"For traffic light tla the following holds:
either its signal is green g or its signal is red r."

Language of Requirements (4)

- Formal Specification Languages such as CASL:

```
spec Arithmetic [op k:Nat] given Nat =
  sort I = { n: Nat . n  < k }
  ops __add__, __sub__ : I * I -> I
  forall n,m:    . n add m = (n + m) mod k
                 . n sub m = (n -? m) mod k
```

- Process Algebras such as CSP:

$$VM = button \rightarrow coin \rightarrow candy \rightarrow VM$$

- And many many more!

Bad Requirements

- Bad requirements specifications can lead to poor designs.
- A function that returns $x * y$.
  `int multiply2(int x, int y) { return 8; }`
- If it is specified that the function should be limited to only (1,8) and (2,4) as inputs then this implementation is perfectly fine.
  - Such an implementation is called a software stub.
- If unspecified... result of unclear requirement.
- Cost money and time.
- Care needed when specifying requirements.

Requirements Engineering

- Feasibility studies:
  - Check if system will be useful for clients (user reqs & existing software).
  - Check if delivered on budget.
- Requirements elicitation and analysis:
  - Work with stakeholders to figure out what is needed
  - Prototypes & storyboards & stories
- Requirements validation:
  - Check requirements with users.
  - Requirements should be error free, consistent, and complete.
- Requirements management:
  - Identify volatile and changeable requirements.
  - (influences choice of development model).

UML

<u>U</u>nified <u>M</u>odelling <u>L</u>anguage

Principles of Modelling

- Choice of model:
  - Pick the right model for the job. There might be many alternative models.
- Different levels of abstraction:
  - High overview levels
  - Low detailed levels
- Connection to reality:
  - All abstraction levels should be able to be related back to reality.
- Independent views of the same system:
  - Why give a decorator wiring plans for your building?
  - Views must be consistent.

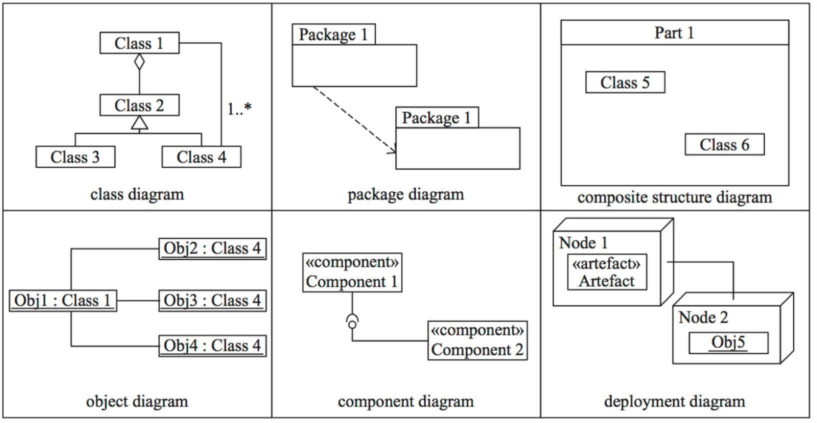
What is UML?

- UML – Unified Modelling Language.
  General purpose modelling language that is intended for software-intensive systems. However, it can be used to model system in general too.
    - Accepted by the ISO as Industrial standard
    - Adopted by the Object Management Group (OMG) in 1997.
- There are 13 diagrams types in UML (2.0).
- Diagrams belong to 1 of 2 categories:
    - Structural (also known as static).
    - Behavioural (also known as dynamic or timing).
- Each diagram describes an aspect of the model. Each aspect should be consistent with the other aspects.

Structural Diagrams

- Describes the "things" or entities in a system and the relationships between them.
- Shows "what" the system should look like and "what" it does, but not the "how"!
- A structural aspect of the model may be thought of as a snapshot in time of any system.
- 6 diagrams in UML 2.0.
  - Class diagram
  - Component diagrams
  - Composite structure diagram
  - Development diagram
  - Object diagram
  - Package diagram

Structural Diagrams



Reference: Jon Holt, UML for Systems Engineering: Watching the wheels, IET, 2007.

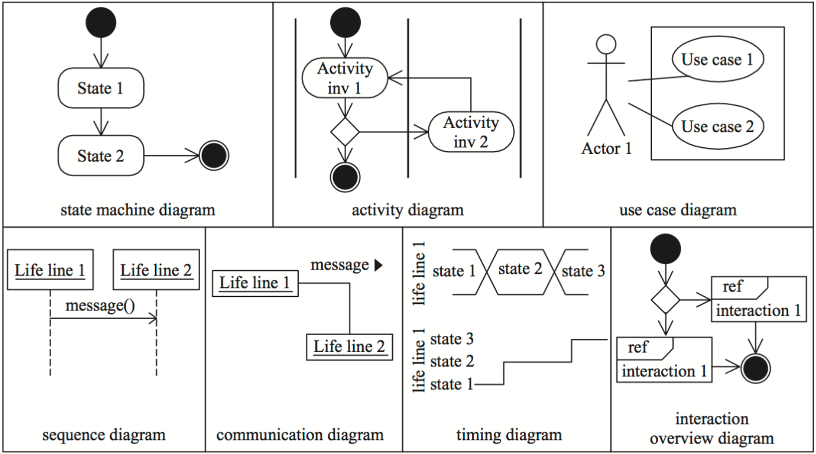
Structural Diagrams (2)

- Class diagram
  - Describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.
- Component diagram
  - Depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems.
- Composite structure diagram
  - Shows the internal structure of a class and the collaborations that this structure makes possible.

Structural Diagrams (3)

- Deployment diagram
  - Models the physical deployment of artefacts on nodes.
  - To describe a web site, for example, a deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software components ("artefacts") run on each node (e.g., web application, database), and how the different pieces are connected (e.g. JDBC, REST, RMI).
- Object diagram
  - Shows what actual things (instances of classes) exist and their relationships. Shows a complete or partial view of the structure of a modelled system at a specific time.
- Package diagram
  - Depicts the dependencies between the packages that make up a model.

Behavioural Diagrams



Reference: Jon Holt, UML for Systems Engineering: Watching the wheels, IET, 2007.

Behavioural Diagrams (1)

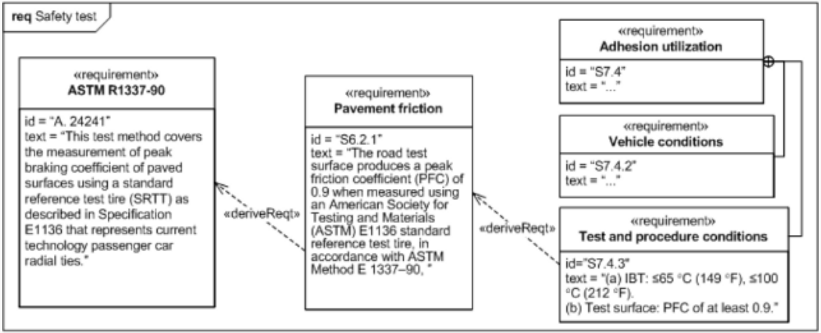
- Activity diagram
  - Shows graphical representations of workflows of step-wise activities and actions with support for choice, iteration and concurrency.
- Communication diagram
  - Models the interactions between objects or parts in terms of sequenced messages.
- Interaction overview diagram
  - Shows a control flow with nodes that can contain interaction diagrams.
  - The interaction overview diagram is similar to the activity diagram, in that both visualise a sequence of activities. The difference is that, for an interaction overview, each individual activity is pictured as a frame which can contain a nested interaction diagram.

Behavioural Diagrams (2)

- Sequence diagram
  - Shows how processes operate with one another and in what order. Shows object interactions arranged in time sequence.
- State machine diagram
  - Describe the different states of a system and the transitions between them.
- Timing diagram
  - Used to explore the behaviours of objects throughout a given period of time.
- Use case diagram
  - Representation of a user's interaction with the system and depicting the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system.

SysML

- SysML is an extension of UML for Systems Engineering.
- It contains new types of diagram such as Requirements Diagram.



- Can capture requirements in a model-based approach.

Summary

- Different types of requirements.
  - User Requirements.
  - System Requirements.
- Further classification of requirements:
  - Functional Requirements.
  - Non-Functional Requirements.
- UML – Used to Model Systems.
  - Use of this is coming soon!