Software Engineering 301:
Software Analysis and Design

Modelling: Introduction

Agenda

- What is modelling & why do it?
- Unified Modeling Language (UML)

Models

- Every model is an idea or set of ideas about something
  - discards unimportant details
  - emphasizes important details

  } *abstraction*

    - "important" details depend on purpose of model
- Models must be represented
  - notation, words, images, physical constructs

An analogy

- Consider a commercial airliner
  - huge machine
  - enormously complex
  - hideously expensive
- Learn basic principles upon which it is built
  - mechanics, electronics, software
- Figure out what you want to know about it
  - e.g., manoeuvrability
- Figure out what features affect manoeuvrability
- Represent those features to inform others

An analogy

- Is this a real airplane?

An analogy

- How about this?

Abstraction versus detail

- Two key questions:
  - How much (and which) detail should be shown?
  - What should the form of the model be?
- To answer these, you must consider what is your purpose in creating a particular model
  - Simple presentation to non-technical customers?
  - Looking for any general problems?
  - Organizing differently for the sake of ease of interpretation?
  - Getting a deep, technical understanding?

Considerations in modelling

- Target audience
  - yourself, teammates, future developers, non-technical stakeholders, …
- Purpose
  - overview of ideas, detailed specification, technical explanation, exploration, …
- Maturity
  - initial ideas, specification for moving forward, documentation regarding what has been done

Software models

- Models of software are often represented as documentation
  - user documentation
    - installation manual
    - interface description
    - troubleshooting guide
  - developer documentation
    - source code itself (model of run-time behaviour)
    - comments in source
    - others???

Models

- It is a common mistake to confuse models with their representations
  - In the course, you will have to remember the difference between software models (of requirements, design, testing, …) and the diagrams and documents to represent them
- The means of representation will always bias understanding of a model
  - e.g., a toy airplane model has some properties that are like a real airplane and others very different
  - e.g., a 2D picture of an airplane has some significantly different properties than a real airplane

Bad models

- We obviously want to strive to make “good” models
- Properties to avoid:
  - Not aiming at a particular purpose
  - Too complex or too simple for its purpose
  - Misrepresents the reality (meaning that the details shown are wrong)
  - Abstracts away the wrong details
  - Uses language unfamiliar to the target audience
  - Physically implausible
- Properties that you should think twice about
  - More expensive to create than the real thing
  - Difficult to change

Are models worthwhile?

- IF …
  - they are cheaper to create than the real thing
  - they permit analysis of complex points
  - they can be understood
- Common developer's reaction to models:
  - "Models are a waste of time; I'll just write source code"
  - The mistake is in the universal assumption here:
    - SOMETIMES models are a waste of time, not always
    - Just-enough modelling is the target to strive for, not more, not less

Agenda

- ~~What is modelling & why do it?~~
- Unified Modeling Language (UML)

Unified Modeling Language (UML)

- UML can be used to represent models about software (and some other things, too)
- UML is usually graphical
- UML is descriptive rather than prescriptive
  - different ways you can say things
  - some are definitely wrong

UML vs. English

- Consider a book
  - chapters, paragraphs, sentences, words, characters (letters, numbers, punctuation)
  - meta-information (title, page numbers)
- Rules about what characters are used
- Rules about combining characters into words
  - dictionary, but new words are created …
- Rules about combining words into sentences
- Style principles at and above this level

Abstraction guidelines

- One box doesn't tell us anything
- Too many boxes tell us too much
- More lines = more confusion

- Some people have come up with style guidelines for UML
  - These are general principles, not rules

Kinds of language

- Formal (e.g., mathematics)
  - self-consistent, unambiguous, complete
  - difficult to describe fuzzier concepts therein
  - programming languages based on formal languages
- Natural (e.g., English)
  - ambiguous, incomplete?, can be inconsistent
  - difficult to explain crisp, logical concepts therein
- UML is an attempt to bridge this gap
  - some details precise, others are fuzzy
  - using it can be as difficult as using a natural language

Three levels of “(in)correctness”

- Syntactic
  - English: Are you using words in grammatical sentences?
  - Java: Are you remembering all three parts of a for-statement?
  - UML: Are you representing classes with rectangles?
- Semantic
  - English: Do your sentences mean anything?
  - Java: Have you declared the variables that you are using?
  - UML: Are there loops in your inheritance hierarchy?
- Conceptual
  - English: Do your sentences form a reasonable argument?
  - Java: Are you computing the sum, or finding the largest number?
  - UML: Do your objects communicate the information needed to perform the computations?

Differing views on software

- Structural view
  - How is the software divided up into objects, classes, packages, etc., and how do these relate to each other?
- Behavioural view
  - What happens at run-time to perform computations? How do objects interact?

Use of diagrams

- Different diagrams are used to emphasize different aspects of a model
- The same kind of diagram can be used to represent different kinds of models
- For example, consider a model of the process of doing your taxes manually vs. a model of the implementation in a tax program
  - Different models, but both could be represented with a class diagram

UML in SENG 301

- UML has lots of specialized “bells-and-whistles”
  - more diagram types
  - more annotation types
- Avoid the temptation to try to learn every possible detail behind UML
  - some details change
  - some details are not commonly used
- We will concentrate on some of the core details of UML
- REMEMBER: The point is to aid in understanding (yours and your stakeholders’)
  - UML “bells-and-whistles” tend to inhibit this!
- However, I’ve created a redacted version of the UML specification for those who “want the right answer”
  - See Resources page on course website

“How do I do <X> in a <Y> diagram?”

- If <X> is a standard thing to do in <Y> diagrams:
  - You will get a standard, as-simple-as-possible response
- Otherwise…
  - First response:
    - “Are you really sure that you need to?”
      - It will be less likely to be understood if it is not a standard thing
  - Second response:
    - “I would just do it like this …”, probably pointing to a practical approach
  - Third response:
    - “Of course, you can look up what the official specification says …” which may not actually provide the “right” answer