COMP 1531
Software Engineering Fundamentals

Week 01: Wednesday

Course Introduction

About me…

Aarthi Natarajan: (a.natarajan@unsw.edu.au)

Background:

- 16 years software development at IBM, Westpac, Lendlease…
- 8 years instructor at Oracle Corporation
- Email is the best way to contact me
- I believe in interactive lectures, so please ask questions!

COMP 1531 18s1

Software Engineering Fundamentals

Our Team

LiC: Aarthi Natarajan

a.natarajan@unsw.edu.au

Web: http://webcms3.cse.unsw.edu.au/COMP1531/18s1/

Course Admin:

Isaac Carr, Matthew Phillips (GitHub)

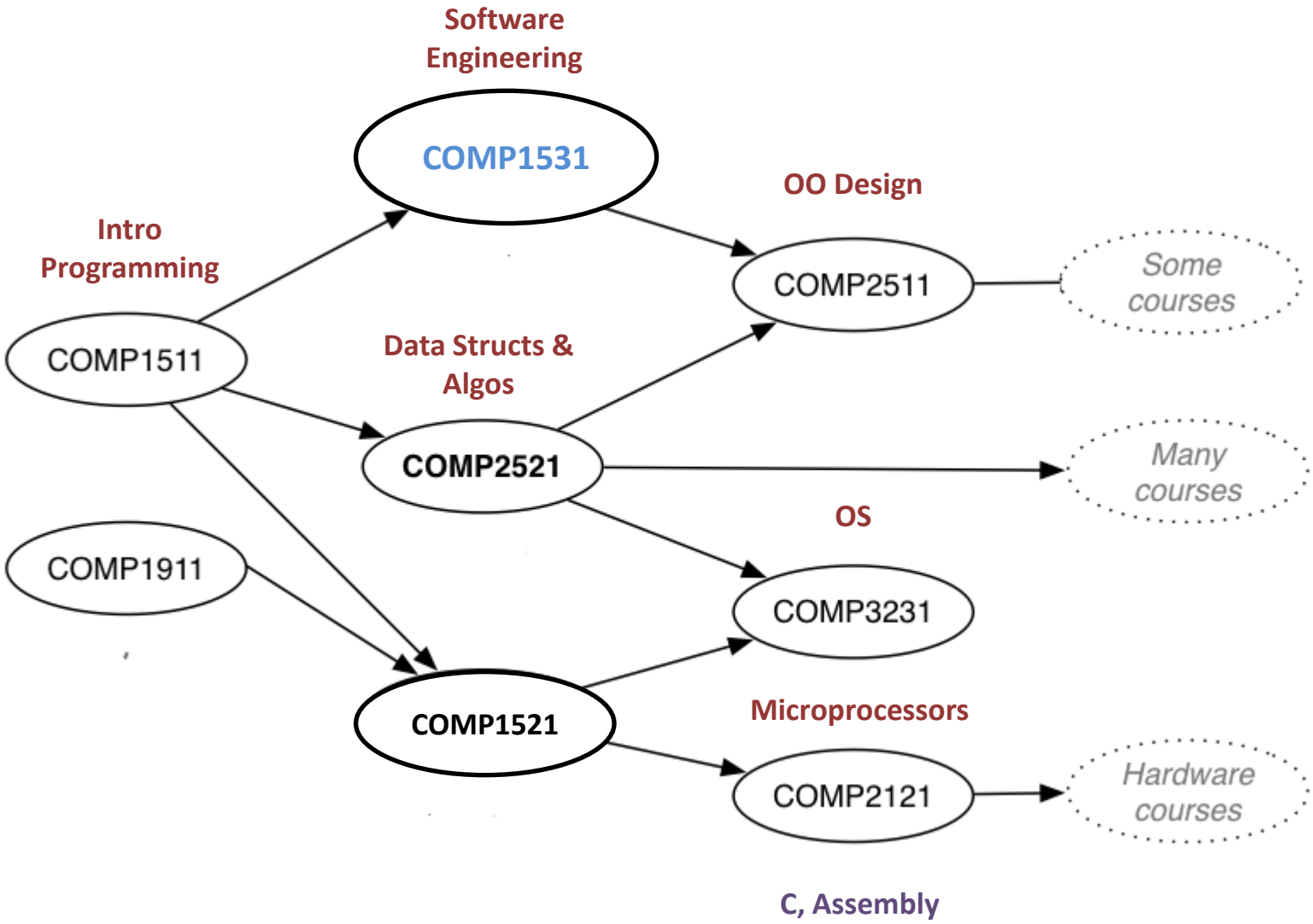
Tutors & Lab Assistants:

Deepanjan Chakrbarthy, Matthew Phillips, Matthew Perry, Hussein Debel, Isaac Carr, Anna Azzam, Kongzhang Hao, Riyasat Saber, George Mountakis, Xiaocong Chen, Vivian Dang, Ryan Fallah, Ian Park,Ian Thorvaldson

COMP 1531 thoughts

- What have you heard about COMP1531?
  - The worst thing that you've heard…
  - The best thing that you've heard…
- Why take COMP1531?

Course Context



Pre-requisites

- Completed **COMP 1511**

- Learnt fundamental C programming and are familiar with:
  - variables
  - data types
  - loop structures
  - defining and using functions and returning results

COMP 1531 students

❖ COMP 1511

- Gets you thinking like a programmer
- Solving problems by developing programs and expressing your solution in C

❖ COMP 1531 Course Goals

- gets you thinking like a software engineer
- teaches you how to deliver value to customers
- learning building large software systems is more than mere programming

COMP 1531 Major Themes

❖ Software Engineering (SE)

- Understand the importance of SE
- Realise SE is **not** Programming
- Problem solving – apply SE principles to solve a real-world problem
  - Explore the key phases of SE life-cycle
  - Analyse a problem and elicit user requirements
  - Design using sound design principles
  - Effective coding and testing techniques

COMP 1531 Major Themes

❖ Team Collaboration

- Learn how to tackle large projects as a team than individual
- Team communication
- Teamwork essential for future employment
- Gain experience in using collaboration tools (GitHub)

❖ Agile Software Development

- Understand the impact of choice of software development methodology to a software development project
- Understand and gain practical experience in Agile Software Development practices

How to teach software engineering?

- Within the constraints of an academic environment, how can we best teach SE?
  - Software Engineering is not rocket science, but it is an **art** that requires the use of common sense, discipline and diligence
  - No single right way or no single text book to teach software engineering
  - My approach: to teach and educate you on what I have learnt from my experience

Credit teaching material

- No text book, comprehensive lecture slides will be provided for COMP 1531 - some content and ideas are drawn from:
  - *Software Engineerings* , by Ivan Marsic, Rutgers, The State University of New Jersey (Available for free download from: http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf)
  - *Agile Software Development: Principles, Patterns and Practice* , by Robert C Martin, Pearson

- Links to useful tutorials will be uploaded as necessary

How do we obtain our educational objectives?

- **Lectures…** (Recorded through Echo 360)
  - 4 hour lectures (instead of 3 hours in 17s1)
  - Wed (09:00 – 11:00), Thu (09:00-11:00), weeks 1-12
  - second hour of Wed lecture will be devoted to practical demos in Python/Flask or guest lectures
- **Tutorials and Labs…**
  - 3 hours per week (1 hr tut followed by lab), weeks 2 – 13
  - Tutorials re-enforce learning from lectures
  - Labs comprise small design and practical programming exercises, individual or in pairs
- **A Group Project…**
  - Project iteration demos scheduled in some lab sessions

Assessments

- Tutorial & Lab Sessions: 3 hours per week (12%)
  - Tutorials re-enforce learning from lectures and provide practical demonstrations
  - Marks will be awarded for tutorials based on participation
  - Lab sessions encompass both design and practical programming exercises
  - To obtain any lab marks for a Week X lab, you must:
    - *upload your completed solution to GitHub*
    - *demonstrate your work to your tutor in the lab class in the week it is due OR submit online through GIVE by the due-date*
  - **Cannot** obtain marks by emailing solution to tutors

Assessments

- Online Quizzes (3%)
  - Multiple-choice format, similar in style to MCQ in exam
  - Review of content covered in lectures
  - Taken in your own time (via WebCMS3)
  - Due before Sunday 11:59 at end of week
- Assignment 1 (5%)
  - Specification will be released in week 3
- Final Exam (50%)
  - 3 hour final exam, more details through the term

Group Project

- Contributes to 30% of the final course mark
- Carried out in teams of 3
- Project specification will be released in Week 05
- Implemented using an *Agile Software Development Model*
  - Working software to be delivered in iterations
  - Project specification **can change** at the end of an iteration (as customer changes mind…), so **design well** !!
  - Marks will be awarded for each iteration demo, which will count towards your overall group project mark
  - Responsibilities to be assigned to each group member during each iteration and all team members **MUST** contribute equally (Tutors will monitor GitHub)
  - Final project demo held in week 13

Course Mark

- Course Work Mark =

  Tut_Lab + Quiz + Assignment 1 + Group Project (out of 50)
- Exam Mark = Mark from the 3 hour final exam (out of 50)
  - ExamOK = Exam Mark >= 25/50
- Final Course Mark (out of 100) =

  Course Work Mark + Exam Mark
- Final Grade
  - UF, If !ExamOK (even if final course mark > 50)
  - FL, if Final Course Mark < 50/100
  - PS, if 50/100 ≤ Final Course Mark < 65/100
  - CR, if 65/100 ≤ Final Course Mark < 75/100
  - DN, if 75/100 ≤ Final Course Mark < 85/100
  - HD, if Final Course Mark ≥ 85/100

System

- Most work done on Linux or Mac
  - Lab work and group project can be done on the CSE machine labs or your own device
  - Technology stack – Python, Flask
  - Must use Python 3.5 onwards (Use virtual environment, more instructions will be provided in the forth-coming lecture and tutorial sessions)
  - Use your own favourite text editor (e.g., vim)
- Collaboration and Versioning Tool - GitHub

Supplementary Exam

- Students are eligible for a Supplementary Exam if and only if:
  - they cannot attend the final exam due to illness or misadventure
  - their Final Mark is in the range $47 \leq$ Final Mark < 50 AND Exam Mark ≥ 22.5/
  - a supplementary exam will not be awarded for any other reason.

Plagiarism

Just don’t do it!

COMP 1531
Software Engineering Fundamentals

Introduction to Software Engineering

Why become a Software Engineer?

Meet software engineers at Google:

- https://www.youtube.com/watch?v=9S_fnC5jPgw

6 reasons to become a software engineer:

https://www.youtube.com/watch?v=lfTwqw_gyKs

- Software is critical to society

  … Software is permeating our society, used to control functions of various machines (e.g, aircrafts, pacemakers)

- Building software is exciting, challenging and fun

  … building software for the next generation that change how billions of people interact, connect and explore

  … learn about team culture, meet new people

- $$$

So far in CSE…

- You have mostly implemented carefully defined specifications
- COMP 1917: Implement a simplified Pokemon
- For this assignment you were given:

  … a well-defined specification

  … a list of commands for the pokemon

  (e.g., a – add a pokemon to the list, > - move to the next pokemon etc)

  … a header file (typeDefs, function prototypes)

  … the abstract data type to use (e.g., list)

  … hints about the implementation (e.g., how to traverse the list)

Objective so far…

- Build an implementation that matched the specification in functionality, work complexity (O(N) or O log N)
- And you were given…
- …the specification (what to do or requirements)
- …the design (the Abstract Data Type (ADT) interface)
- …hints about implementation
- …perhaps a pointer to some libraries to use

- Someone had to figure all of the above; that's software engineering and now it's your turn

What is software engineering?

"The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software." [IEEE]

What is Software Engineering?

- "Software Engineering" is a discipline that enables customers achieve business goals through *designing* and *developing* software-based systems to solve their business problems e.g., develop a patient-record software in a doctor's surgery, a software to manage inventory
- A software engineer starts with a *problem definition* and applies tools and techniques to obtain a *problem solution*. However…
- Software engineering requires great emphasis on ***methodology*** or the *method* for managing the development process in addition to great skills with tools and techniques.

Software Engineering is not Programming

- Software engineering:
  - Understanding the business problem (system-to-be, interaction between this system and its users and environment)
  - Creative formulation of ideas to solve the problem based on this understanding
  - Designing the "blueprint" or architecture of the solution
- Programming:
  - Craft of *implementing* the given "blueprint"

Role of a software engineer (1)

A bridge from customer needs to programming implementation

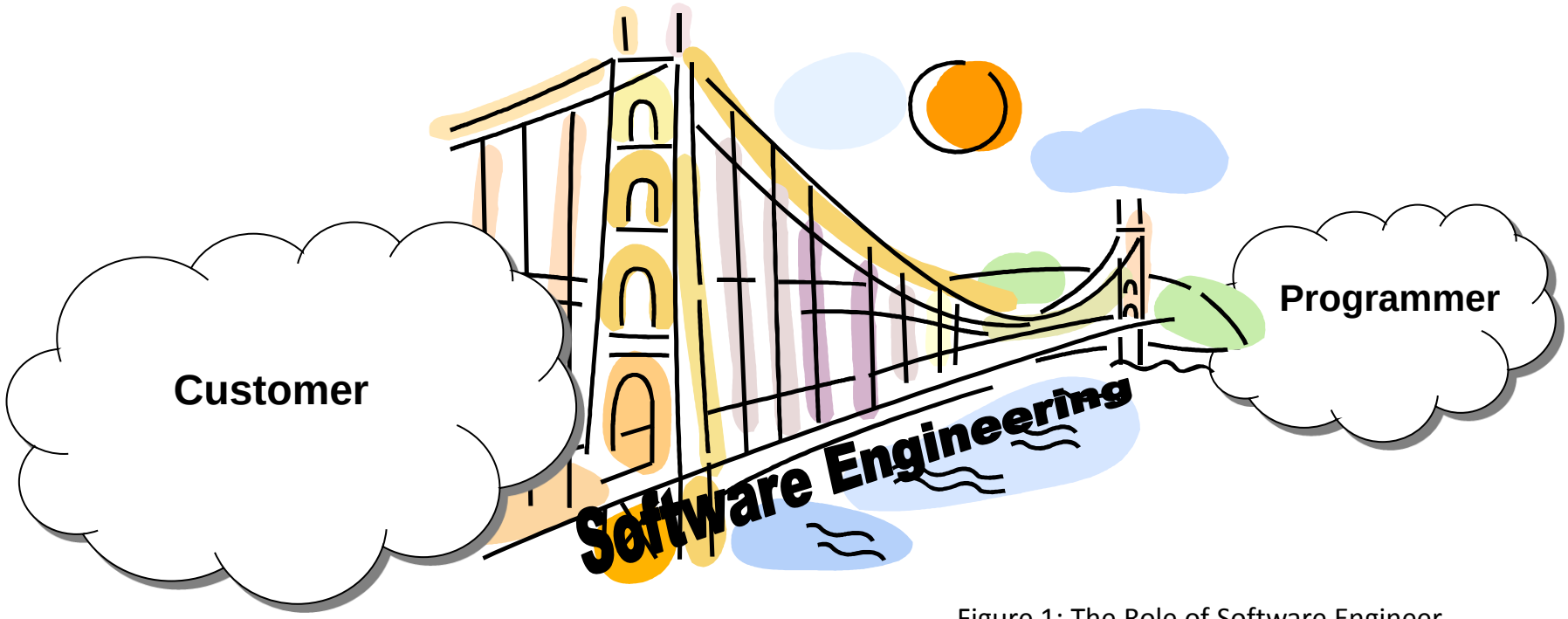


Figure 1: The Role of Software Engineer

Role of a software engineer (2)

- Software engineering ***delivers*** value to the customer

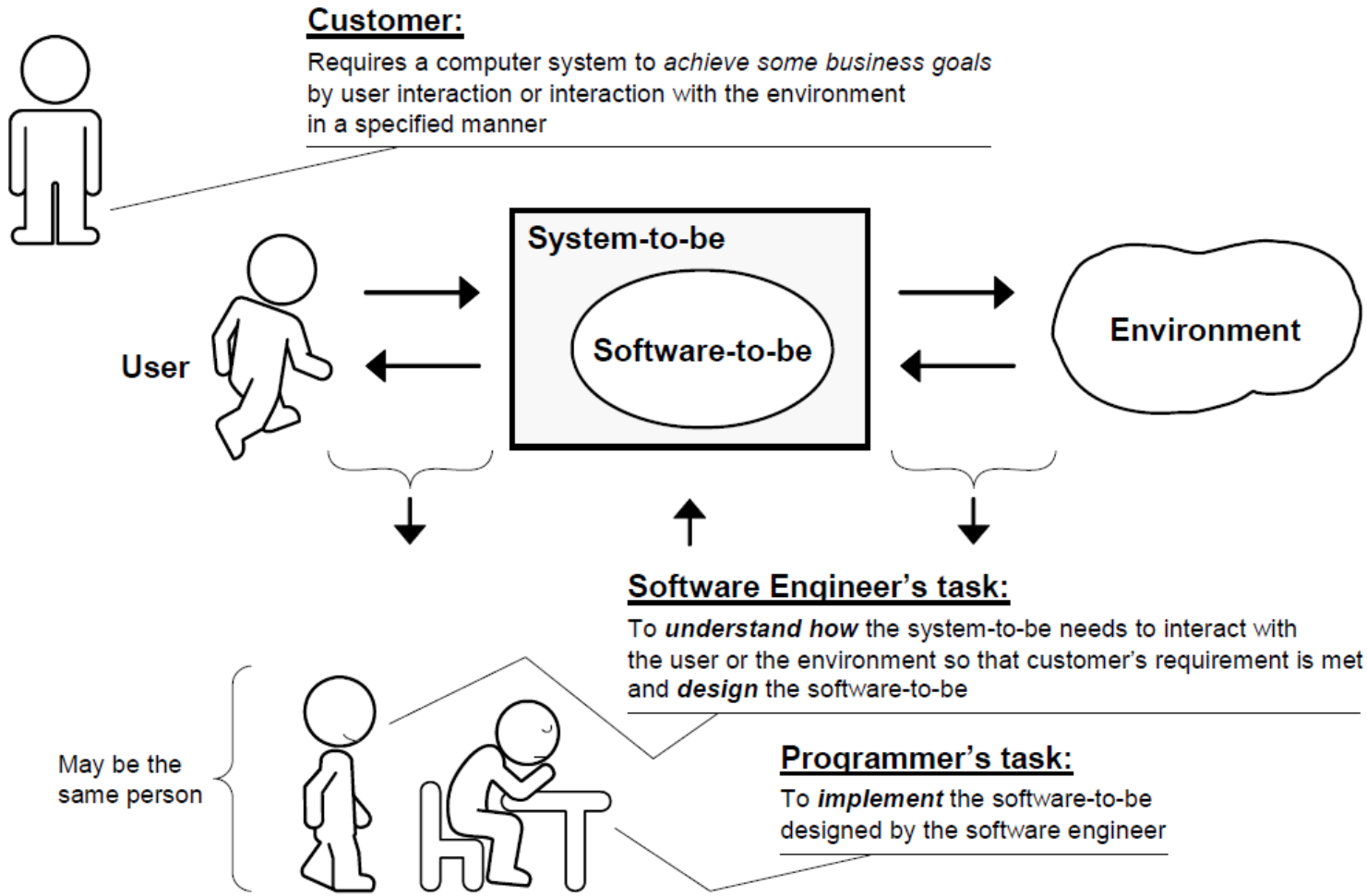


Figure 2: Role of Software Engineering

Software engineering enforces a certain discipline to building software systems that encompasses a complete life-cycle starting from requirements to design, implementation and testing

Why do we need software engineering?

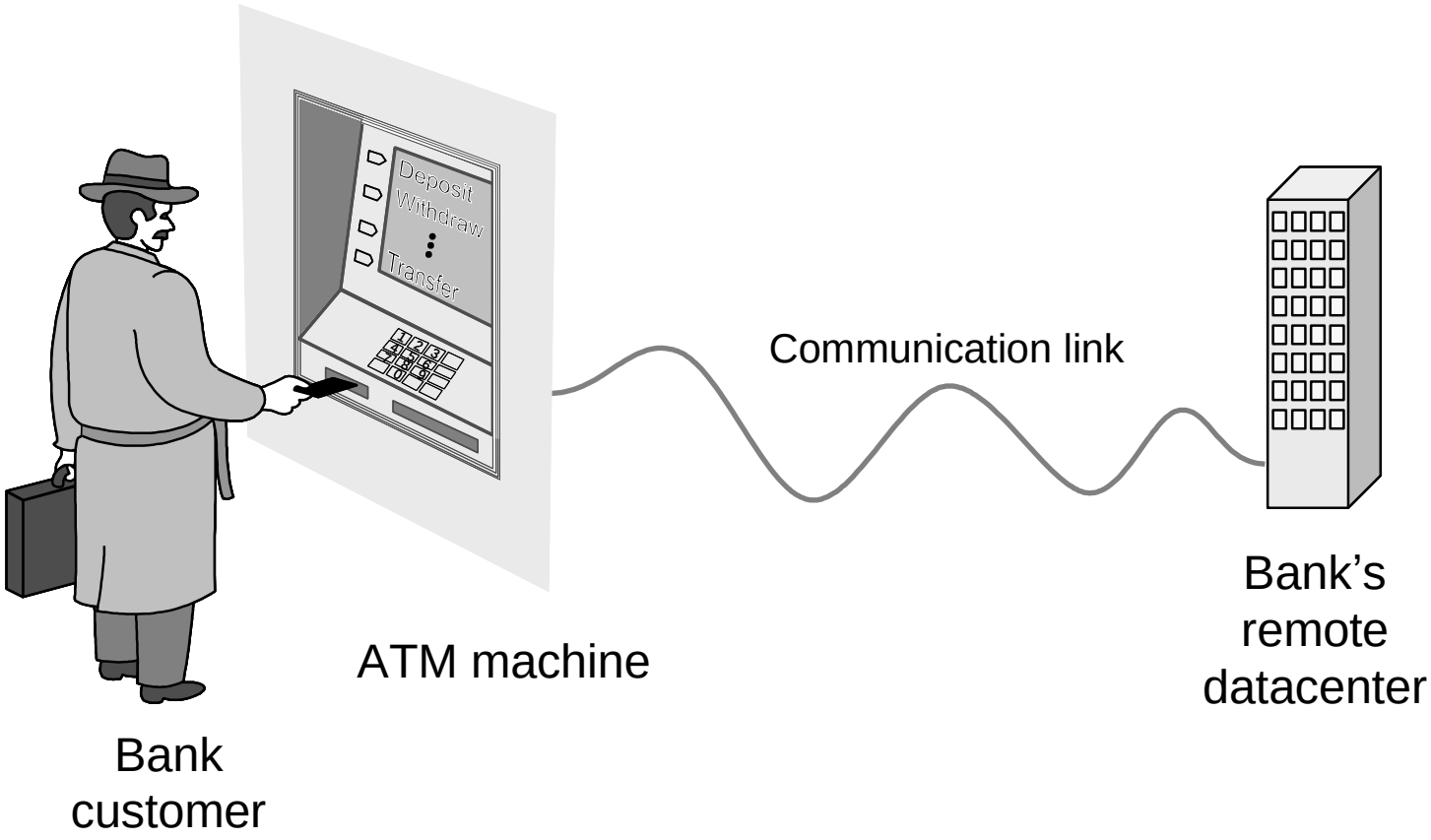
Software engineering is “hard” and there is no single reason behind this –

it’s a “wicked problem”

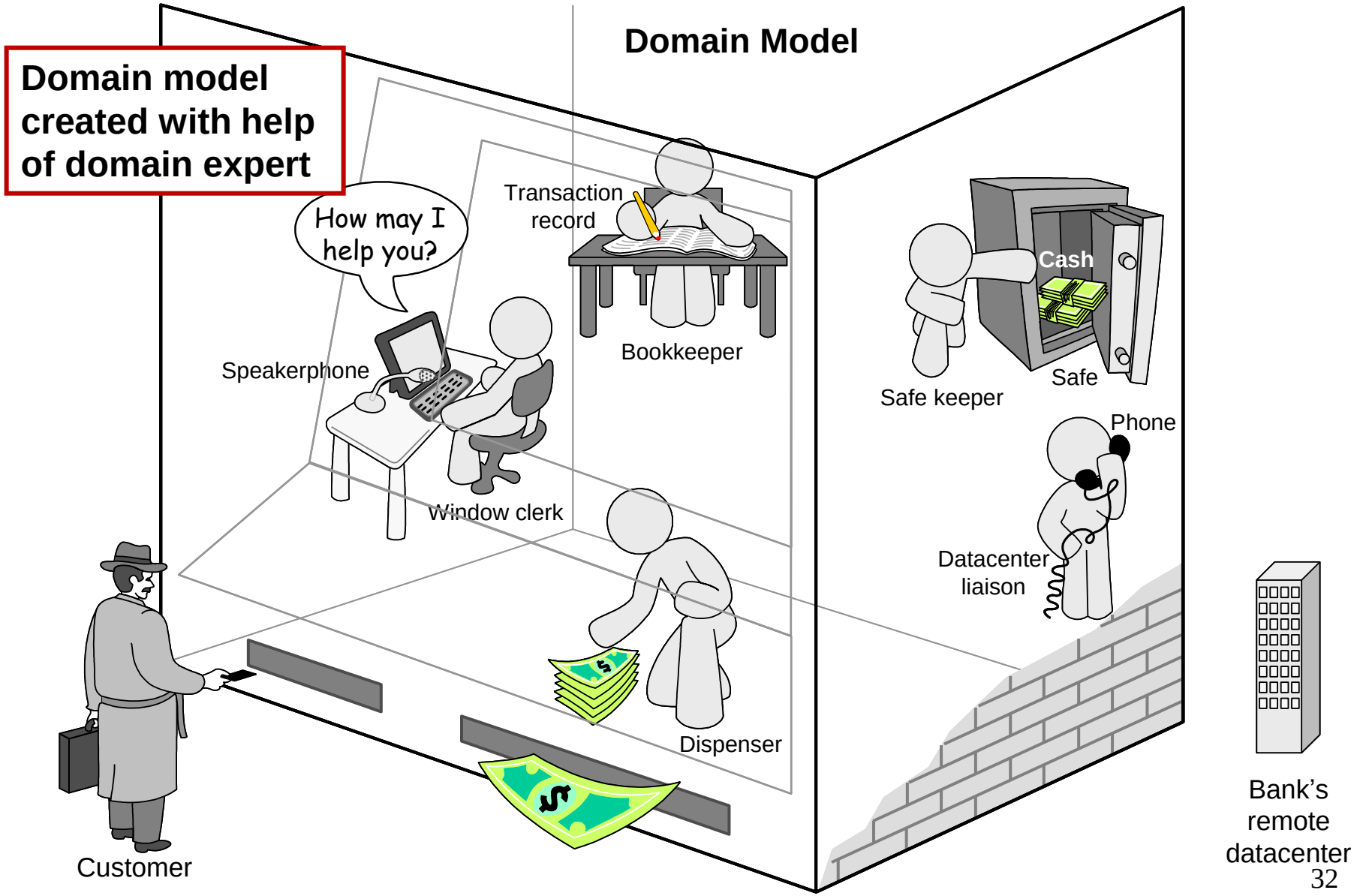
Why do we need SE (1)?

- Software development is complex, so building software without discipline is crazy
- Software is deceptively intangible and SE requires imagination
  - Understand both the problem domain and software domain
  - Challenge is to find a set of *good abstractions* that is representative of the problem domain and build a conceptual domain model
  - But, we live in a changing world…so *good abstractions* wear out, break and get dispersed
  - Creative formulation of solutions, design alternatives, trade-offs, difficult to come up with the "correct solution"

Software Engineering requires imagination
e.g., ATM Machine – understanding the money-machine problem



How ATM Machine Might Work



Why do we need SE (2)?

- Software errors, poor design and inadequate testing have led to loss of time, money, human life
  - **Case of the Therac-25:** one of the most well-known killer software bugs in history

  - **Explosion of Ariane-501 in 1996:** blew up 37 seconds after initial launch

Software Engineering Life-Cycle

- Tomorrow …

  Software Engineering Life-Cycle

  Using GitHub

COMP 1531
Software Engineering Fundamentals

Week 01: Thursday

Topics covered…

- Software Development Life-Cycle
- Git and GitHub
- More Python basics

Software Engineering Life-Cycle

- We described software engineering as a complex, organised process with a great emphasis on *methodology*. This organised process can be broken into the following phases:
  - Analysis and Specification
  - Design
  - Implementation
  - Testing
  - Release & Maintenance
- Each of the above phases can be accompanied by an artifact or deliverable to be achieved at the completion of this phase
- The life-cycle usually comprises peripheral activities such as feasibility studies, software maintenance, software configuration management etc.

Software Engineering Life-Cycle

1. Analysis:

A process of knowledge-discovery about the "system-to-be" and list of features, where software engineers need to:

- understand the problem definition – identify the system's services (*behavioural characteristics)*
- abstract the problem to define a domain model (*structural characteristics*)

- Comprises both functional (inputs and outputs) and non-functional requirements (performance, security, quality, maintainability, extensibility)
- Popular techniques include use-case modelling, user-stories…

Software Engineering Life-Cycle

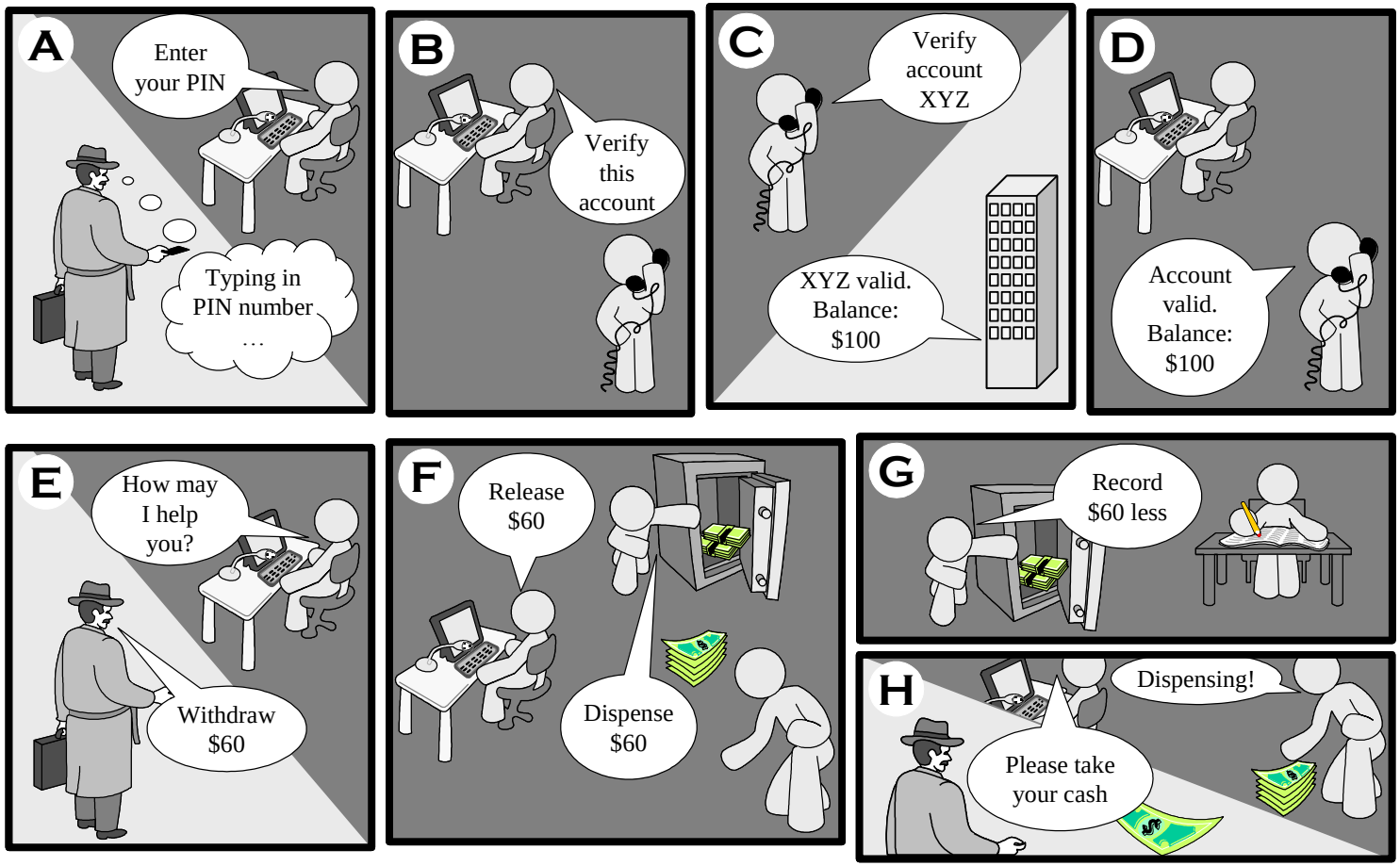
2. Design:

- A problem-solving activity that involves a "Creative process of searching **how to implement all of the customer's requirements"** and generating software engineering blue-prints (design artifacts e.g., domain model)

Software Engineering Life-Cycle

In generating these “blue-prints”….

Cartoon Strip Writing OR more formal symbols (e.g., UML symbols – use-case diagram, class diagram, component diagram…?



Cartoon Strip: How ATM Machine Works

Software Engineering Life-Cycle

3. Implementation:
   - The process of encoding the design in a programming language to deliver a software product
4. Testing:
   - A process of verification that our system works correctly and realises the goals
   - Testing process encompasses unit tests (individual components are tested), integration tests (the whole system is testing), user acceptance tests (the system achieves the customer requirements)
5. Operation & Maintenance:
   - Running the system; Fixing defects, adding new funtionality

Tasks to be completed this week…

1. Create a GitHub account with your UNSW email address: z1234567@student.unsw.edu.au (if you already have a GitHub account with a non-UNSW email address, you can go to step 2 and link a UNSW email address to your existing account
2. After you have created your GitHub account, please go to this link: https://cgi.cse.unsw.edu.au/~cs1531/18s1/github/run.cgi/labs and complete the SETUP exercise in Lab 01 prior to your lab

   (You will not be able to do lab01 if this is not done prior)
3. If you encounter any problem while doing this, either post it in the course forum (include your GitHub username and user email) or email **Matthew Phillips** (matthew.phillips1@unsw.edu.au) with your GitHub username and email.

More tasks this week…

- Group Project
  - Start planning your team of **no more than 3**!
- Python
  - Topics covered so far:
    - Using python interpreter
    - Defining variables, numbers and strings
    - Control flow (if, for, range, while, break and continue)
    - Input and Output
- Make your familiar with GitHub and Python (on the topics covered so far)
  - Some useful tutorial resources

More tasks this week…

- Useful links to GitHub and Python Resources
  - Start planning your team of **no more than 3**!
- Python
  - Python tutorial: https://docs.python.org/3/tutorial/
  - Get started with GitHub: https://guides.github.com/activities/hello-world/
  - An interactive Git tutorial: https://learngitbranching.js.org/

  (Strongly recommended for people who have never used Git before)

Next week…

- Requirements Engineering (User-stories and Use-Cases
- Using UML in Requirements Analysis
- Python (Data-structures – Lists, Tuples, Dictionaries, Defining functions)