Requirements Analysis

ENCE464 Embedded Software and Advanced Computing

Lecturer: Steve Weddell (steve.weddell@canterbury.ac.nz)

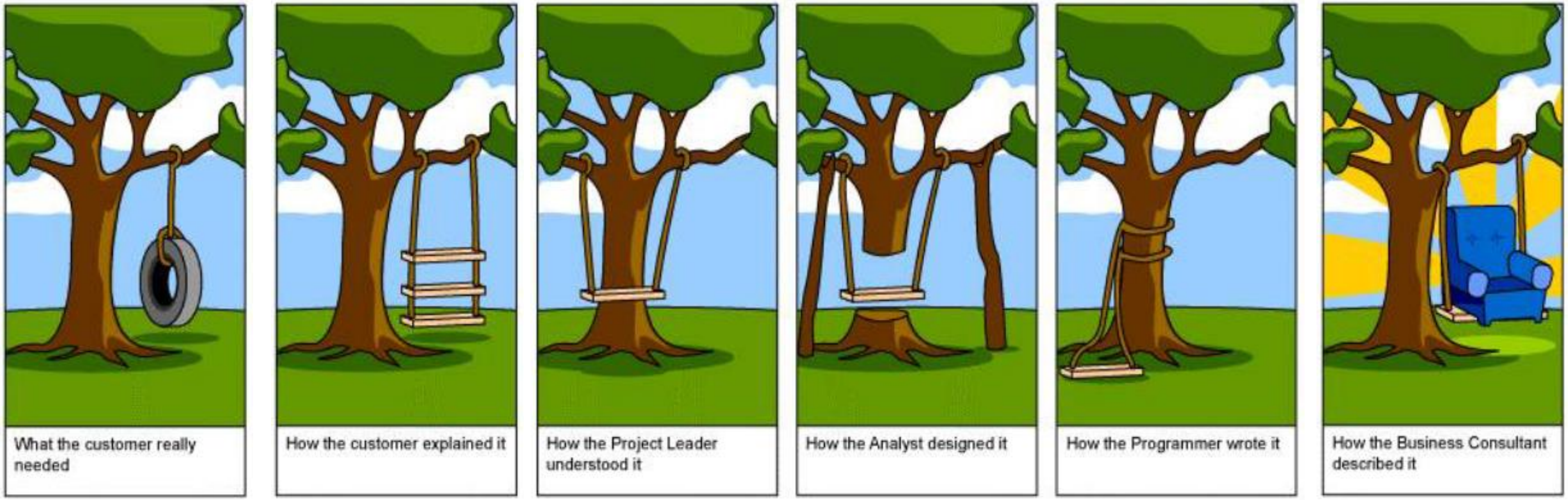
Lecturer: Le Yang (le.yang@canterbury.ac.nz)

Department of Electrical and Computer Engineering

Where we’re going today

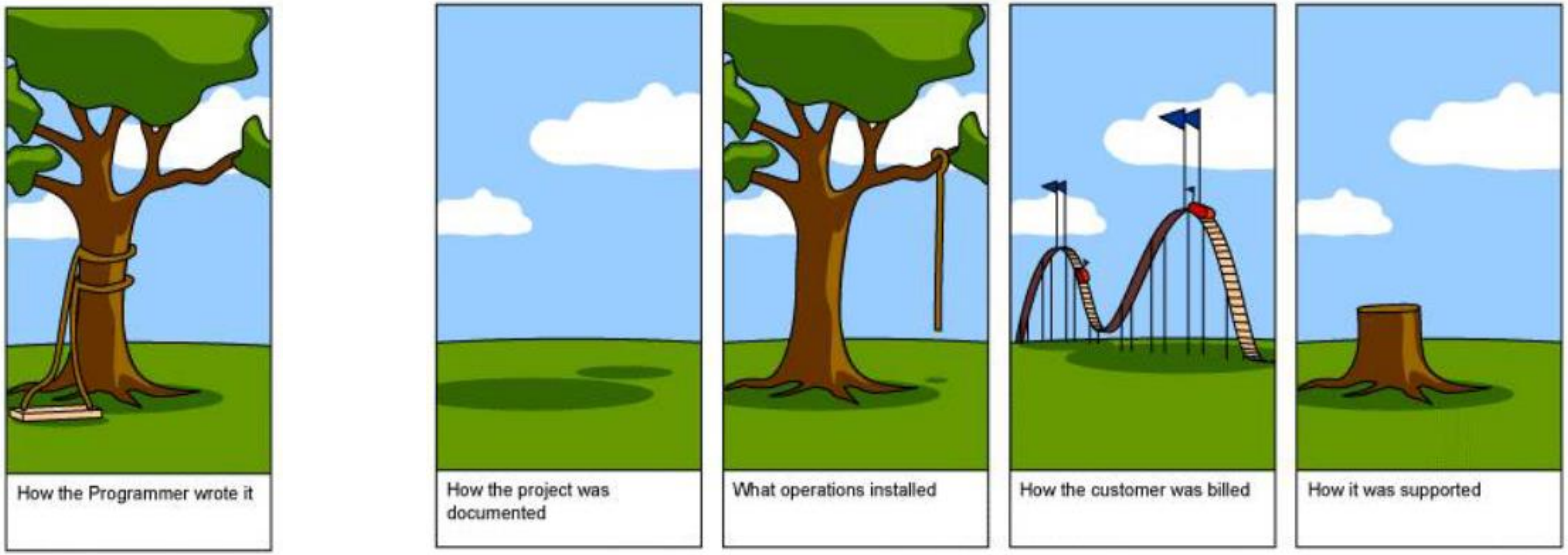
- **Motivations**
- Software requirements analysis and specification
- Architecture

Motivation (1)


What the customer really needed
How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it

Ambiguity leads to errors in information propagation

Motivation (2)


How the Programmer wrote it
How the project was documented
What operations installed
How the customer was billed
How it was supported

Lessons Learned

- Identified (43) software risk areas include
  - #3 No written requirements
  - #5 Requirements with poor measurability
  - #6 No defined software architecture
  - #19 No test plan
- Advices from **Koopman**
  - Define requirements in a measurable and traceable way
    - If you cannot measure a requirement, you will not know whether it is met
  - Document your software architecture
    - Make the code modular and avoid global variables
  - Explicitly specify and plan for performance, dependability, security and safety

From Avoiding the Top 43 Embedded Software Risks.pdf

Validation and Verification

- Validation
  - “Are we building the right product?”
  - Requirements analysis is to ensure that specifications are valid
  - Ultimately performed when software is finished
  - Must be thoroughly checked at an early development point

- Verification
  - “Are we building the product right?”
  - Performed at every development stage to make sure we conform to the specifications
  - Testing is an important part but not the only one

Where we’re going today

- Motivations
- **Software requirements analysis and specification**
- Architecture

Firmware Architecture in Five Easy Steps

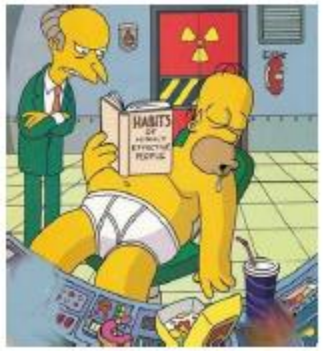
- Step 1: identify the requirements
  - Define the **WHAT** of a program
- Step 2: distinguish architecture from design
  - Outermost layer of **HOW**
- Step 3: manage time
  - Probably non-real-time, soft-real-time and hard-real-time (see next slide)
- Step 4: design for test
  - Unit, integration, system, …
- Step 5: plan for change

From Firmware architecture in five easy steps.pdf

Non-Real-Time, Soft-Real-Time, Hard-Real-Time

- Hard-real-time
  - Any missed deadline is a system failure
  - Mission critical systems where failure to conform timing constraints results in a loss of life or property
- Soft-real-time
  - Allow for frequently missed deadlines
  - Timely completed tasks may have increasing value

Non-real-time



Soft real-time

Hard-real-time

From Avoiding the Top 43 Embedded Software Risks.pdf

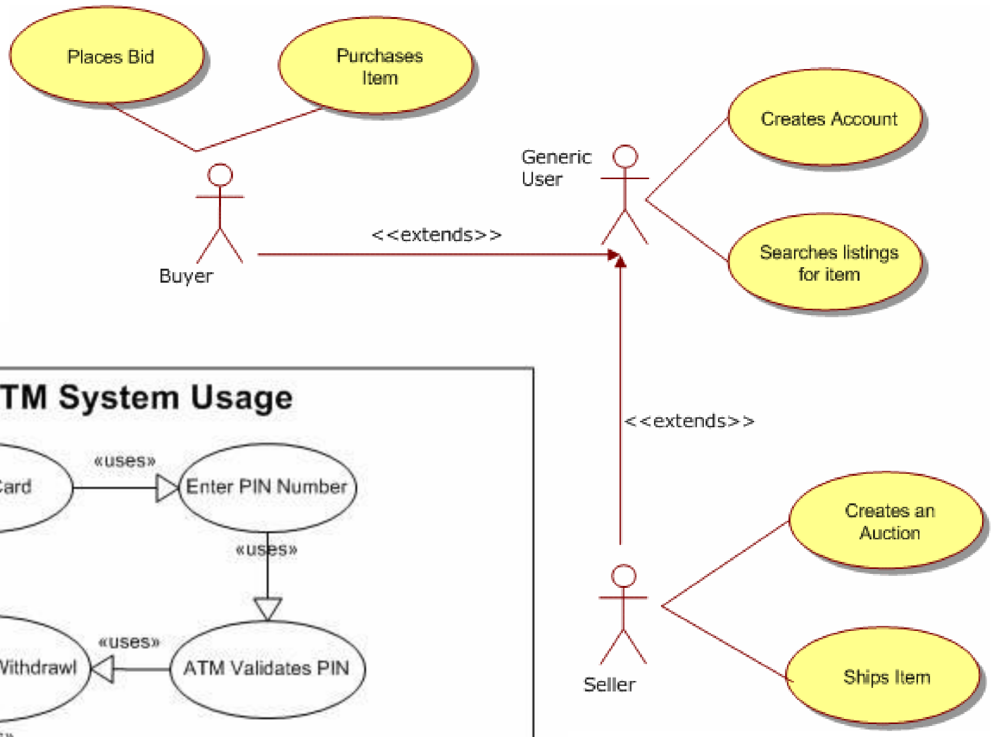
Identify Requirements

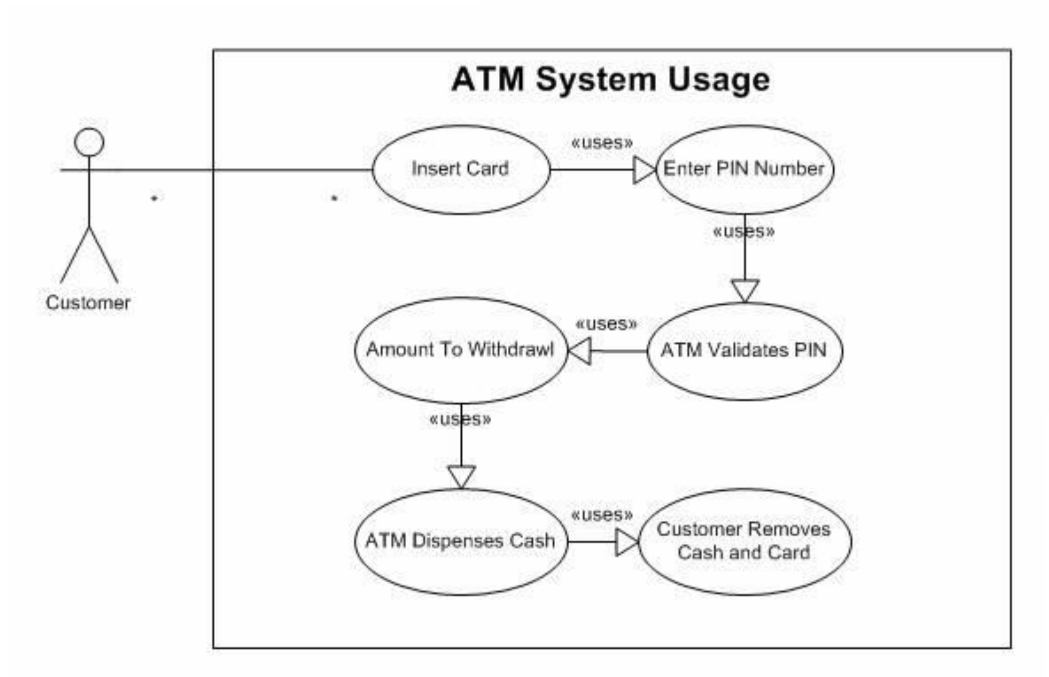
- **Properly written** requirements that define **WHAT** of a software
  - Unambiguous
    - Require no further explanations
  - Testable
    - A set of tests can be constructed to verify whether the requirement is met
  - Example: “If the power is lost during operation, the ventilator shall resume operation according to its last programmed settings within 250ms of power up”

Sources of Requirements

- Customers
- Marketing team
- Software development team

- Use cases
- Creative ideas
- Available technologies





From Firmware architecture in five easy steps.pdf

Software Requirements Specifications (1)

- Introduction
  - Purpose
  - Definitions
  - System overview
  - References
- Overall description
  - Product perspective
    - System/user/hardware/software/communication interfaces
    - Memory constraints, operations …
  - Site adaptation requirements
    - Software functions, user characteristics …

Software Requirements Specifications (2)

- Specific requirements
    - External interface requirements
    - Performance requirements

    - Design constraints
        - Standards compliance
    - Software system attributes
        - Reliability
        - Availability
        - Security
        - Maintainability/portability

    - Other requirements

Principles for Specifications (1)

- Separate functionality from implementation (**WHAT**, not **HOW**)
- Language appropriate for customers should be used

- Specifications must encompass the system (organization, domain, users) of which the software is a component
- Specifications must encompass the environment in which the software operates

- Specifications must be operational, localized, loosely coupled and tolerant of incompleteness and augmentable

Principles for Specifications (2)

- Wording should be present tense and definitive rather than indefinite
  - *If the mouse is clicked on the 'Abandon ship' button, the klaxon sounds twice*
- Avoid words that do not denote present tense or are misleading: might, always, sometimes, instantly

- Point of view should be third person omniscient (knows everything)
  - *The user has the choice of accepting current configuration or invoking 'Change Configuration' dialog box*
- Statements must be viable. You cannot prove that things occurs instantaneously or optimally
  - *The time taken to sort the file is system dependent. An estimate of the sort time is made (as detailed in Appendix 3) and if the estimated time is larger than 10 seconds, a visual indication of the estimate time is given*

Example: Phil's ideal polyline drawing function

- Definitions of terminology (selected)
  - Clicks
    - Clicks are push-and-release operations of the main mouse button
  - Pushes
    - A push is a single push-and-hold operation of the main mouse button
  - Polyline
    - A poly line is a set of one or more connected line segments, the end of one segment is the start of next
- Tool: Line button on toolbar
  - A single line is drawn by a single click at the start point and a single click at the end point. As the cursor is moved away from the start point, a rubber line is displayed from the start point to the tip of the cursor and this is redisplayed as a solid line after the end point is set.
  - A line may be deleted by selecting it, then operating the <del> key

Where we’re going today

- Motivations
- Software requirements analysis and specification
- **Architecture**

Distinguish Architecture from Design

- Architecture: the outermost layer of **HOW**
  - Persistent features, hard to change
  - Careful examination of intended and permissible use of the software

  - Example: architecture for a new office building
    - Outer dimensions
    - Foundation
    - Number of floors

- Design: the middle layer of **HOW**
  - Fine grained details, such as names and responsibilities of tasks in specific subsystems, brand of RTOS used, interfaces between subsystems

Software Architecture

- Divide things into major modules or subsystems
  - Decide the module for each possible input and output pair
    - Define basic behavior of each module
  - Encapsulate decisions that may change

  - High cohesion and low coupling
    - Aggregate or split modules when necessary
  - Push timing requirements towards hardware or hardware-level modules to simplify non-real-time component design and timing analysis
    - Timers over software loops
    - Interrupts servicing over polling
    - Task synchronization

“Six-Pack Test”

- Architecture diagram should pass ‘six-pack test’
  - Even after drinking six packs of beer, every team member is still able to understand the architecture
  - It is devoid of confusing details

From Firmware architecture in five easy steps.pdf