University of
South Australia

INFS 2044

Week 1

Software Design Process

Learning Objectives

- Describe the activities of software design within the software development lifecycle
- Understand how addressing complexity yields better software

Software Design

What
is required

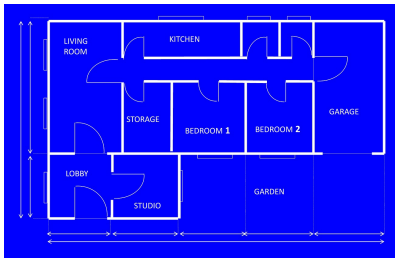
How
the system will be structured and operate

Building a house: An Analogy

- Programming Fundamentals

Systems Analysis

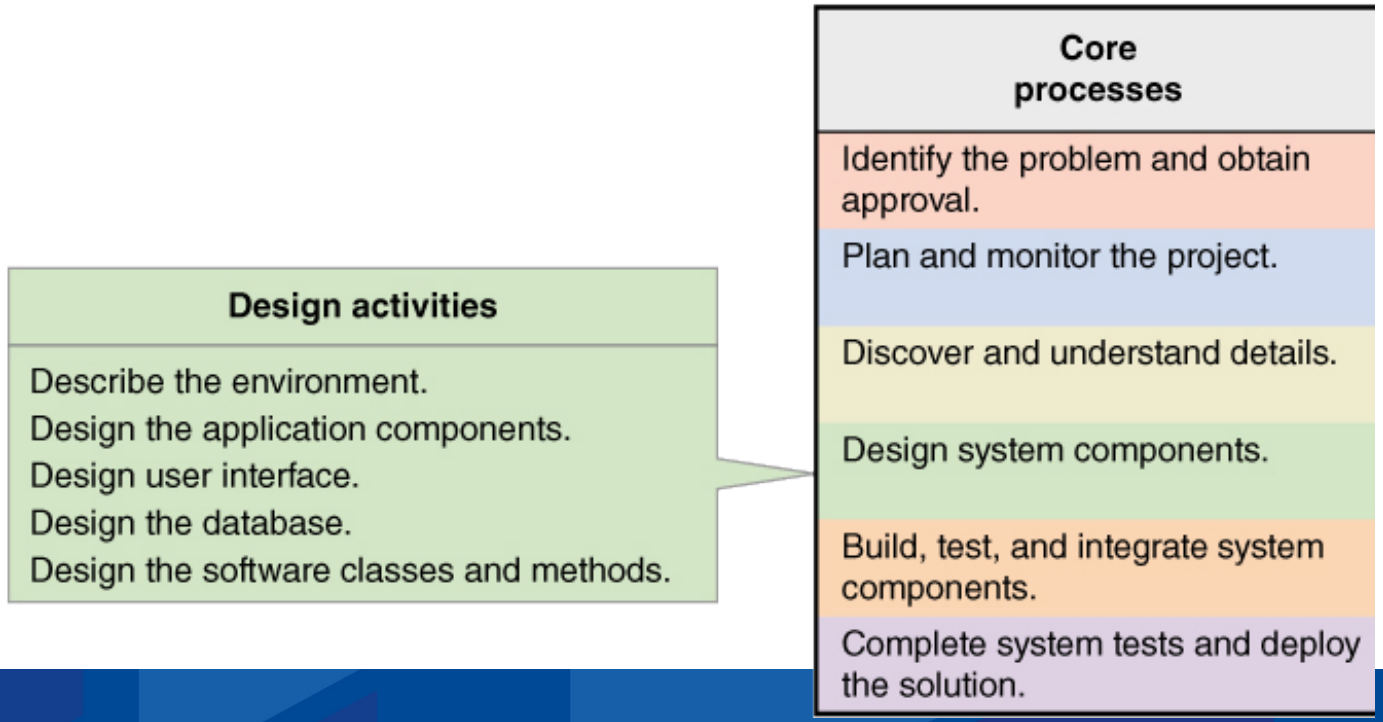
Systems Design



Systems Design Objective

- Analysis provides the starting point for design
- Design provides the starting point for implementation
- Analysis and design results are **documented** to communicate and coordinate the work
- Objective of design is to **define, organize, and structure the components** to serve as a blueprint for construction

Design Activities

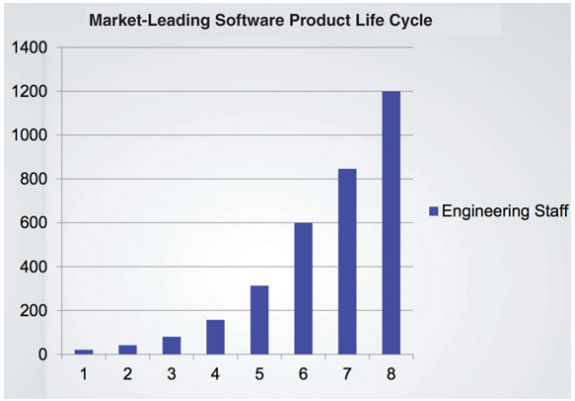


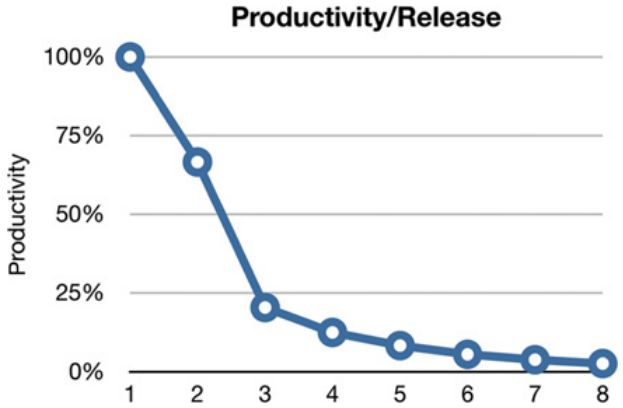
Key Design Questions

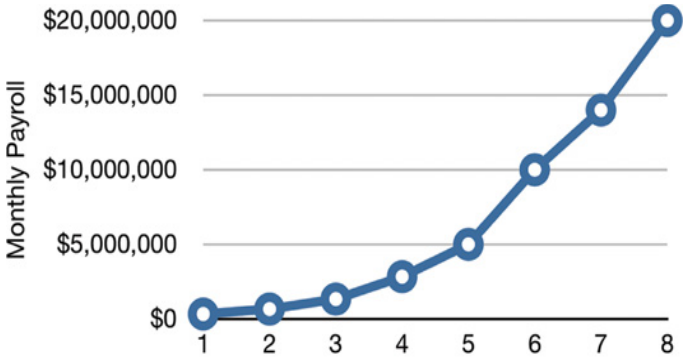
Design activity	Key question
Describe the environment	How will this system interact with other systems and with the organization's existing technologies?
Design the application components	What are the key parts of the information system and how will they interact when the system is deployed?
Design the user interface	How will users interact with the information system?
Design the database	How will data be captured, structured, and stored for later use by the information system?
Design the software classes and methods	What internal structure for each application component will ensure efficient construction, rapid deployment, and reliable operation?

Designing = Managing Complexity

- The goal of software architecture [and design] is to minimize the human resources required to build and maintain the required system. [R.C.Martin, Clean Architecture, 2017]
- Complexity is anything related to the structure of a software system that makes it hard to understand and modify the system. [Ousterhout, 2018]







Figures from R.C. Martin, Clean Architecture

Nature of Complexity

- Apparent complexity: what a developer experiences
  - Hard to understand how a program works
  - High effort to implement a small improvement
  - Unclear which parts of the system must be modified
  - Difficult to fix one bug without introducing another

- Judged by the reader of the code
- Unrelated to program size

Symptoms of Complexity

- **Change amplification:** a simple change requires many code modifications.
- **Cognitive load**: have to load a lot of information in your mind in order to make a change.
- **Unknown unknowns**: important information you need to know before making a change, but not obvious where to find it, or even that it is needed.

Causes of Complexity

- **Dependencies**: one piece of code is tightly coupled with another
  - If one changes, the other must change also
- **Obscurity**: not clear how things work or why the code is the way it is
  - The opposite of obscure is *obvious*: a developer's first guess about how it works or what to do will be correct.

Complexity is incremental

- No one thing makes a system complicated
- Accumulation of thousands of small dependencies and obscurities
- Once complexity arises, it is hard to eliminate

Continuous Evolution

- Programs evolve continuously
  - Cannot design the whole system at once
  - Cannot get the design right the first time
  - Requirements change

Good Software Design

- Good software design reduces apparent complexity
  - Minimize dependencies: modular design
  - Make system structure and behaviour obvious

- **Good design is easy to change**

Example

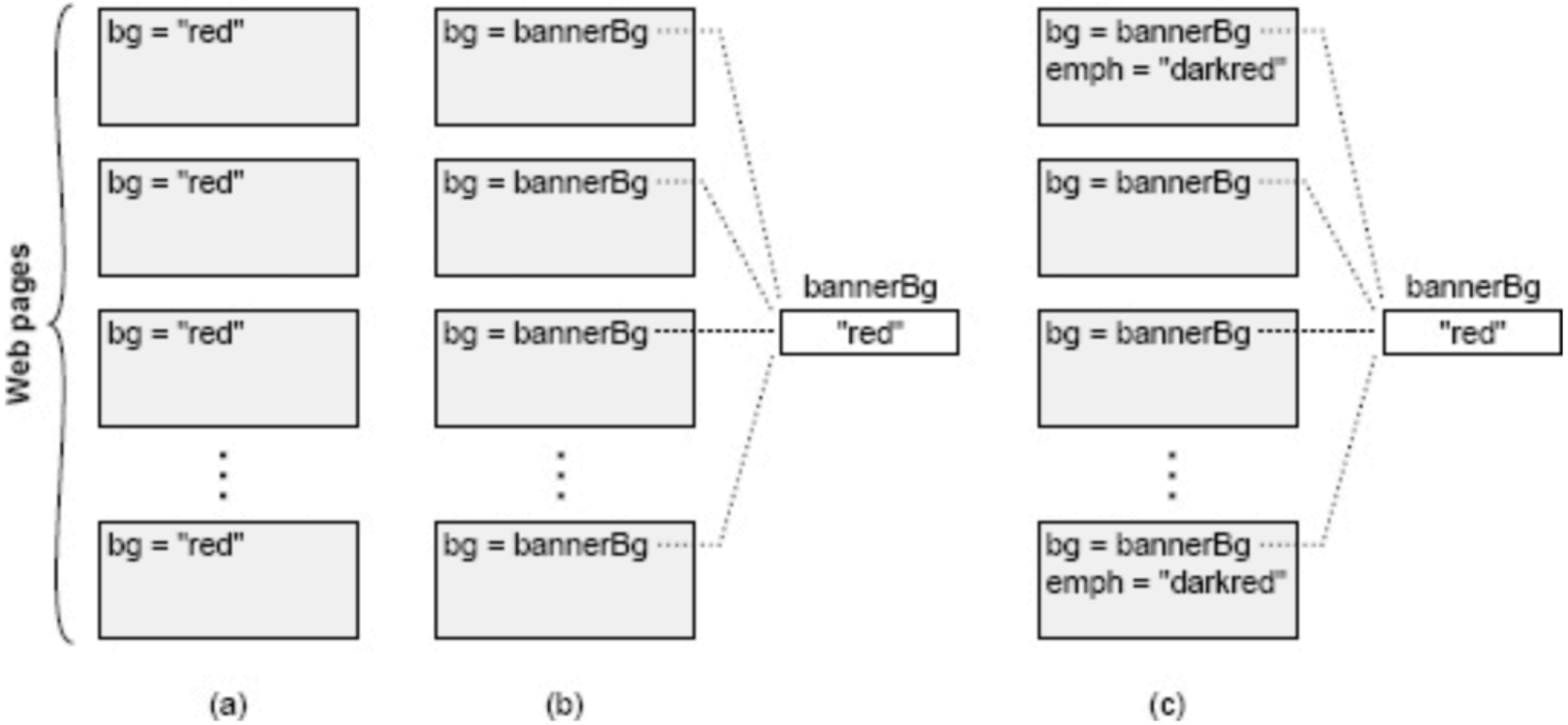
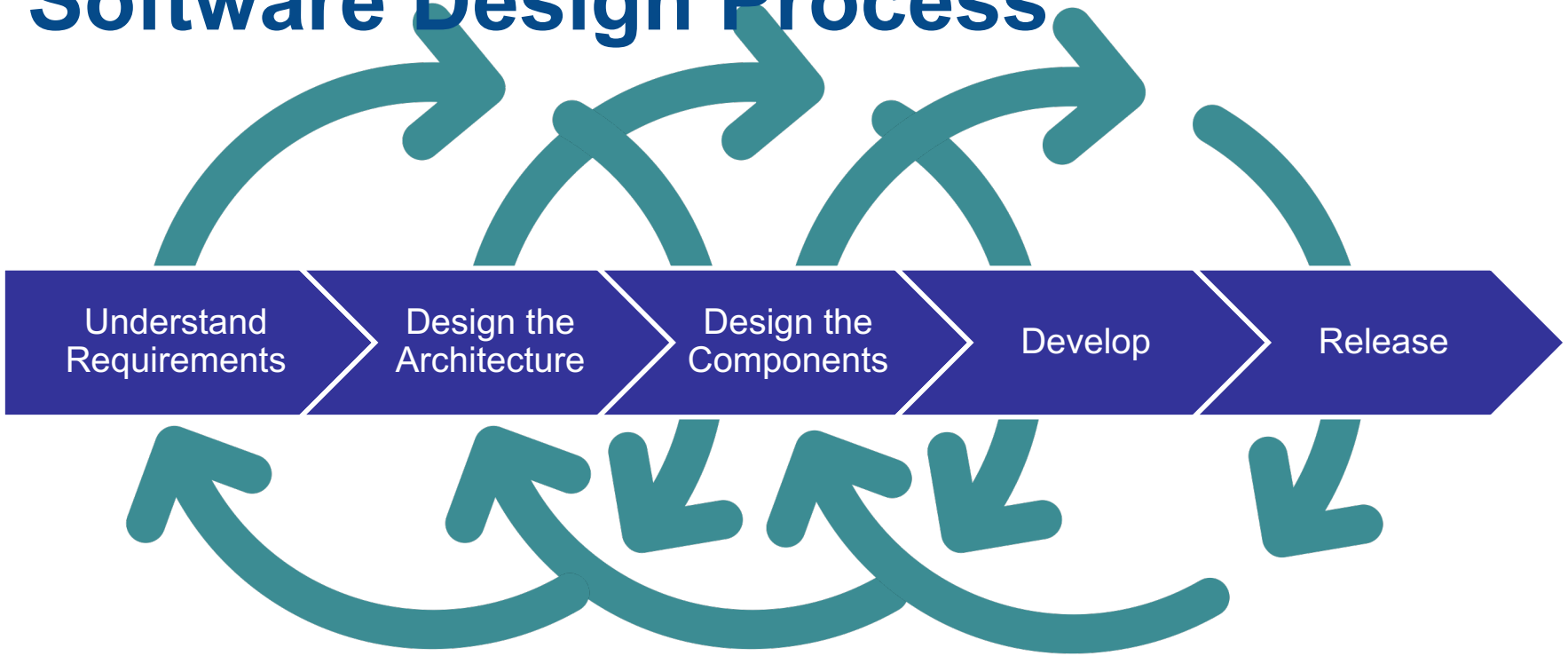


Figure from (Ousterhout 2018)

Working Code is not Good Enough

- **Tactical Programming**
  - Get the next feature or bug fix working
  - Cutting corners is accepted, if it gets things done quickly
  - Results in bad design, high complexity
- **Strategic Programming**
  - Primary goal is to produce a great design
  - How easy is it to evolve this code?
  - Continual small investments

Software Design Process



Understanding Requirements

https://live.staticflickr.com/194/502743697_95d6f921d0_b.jpg

Requirements as Behaviour

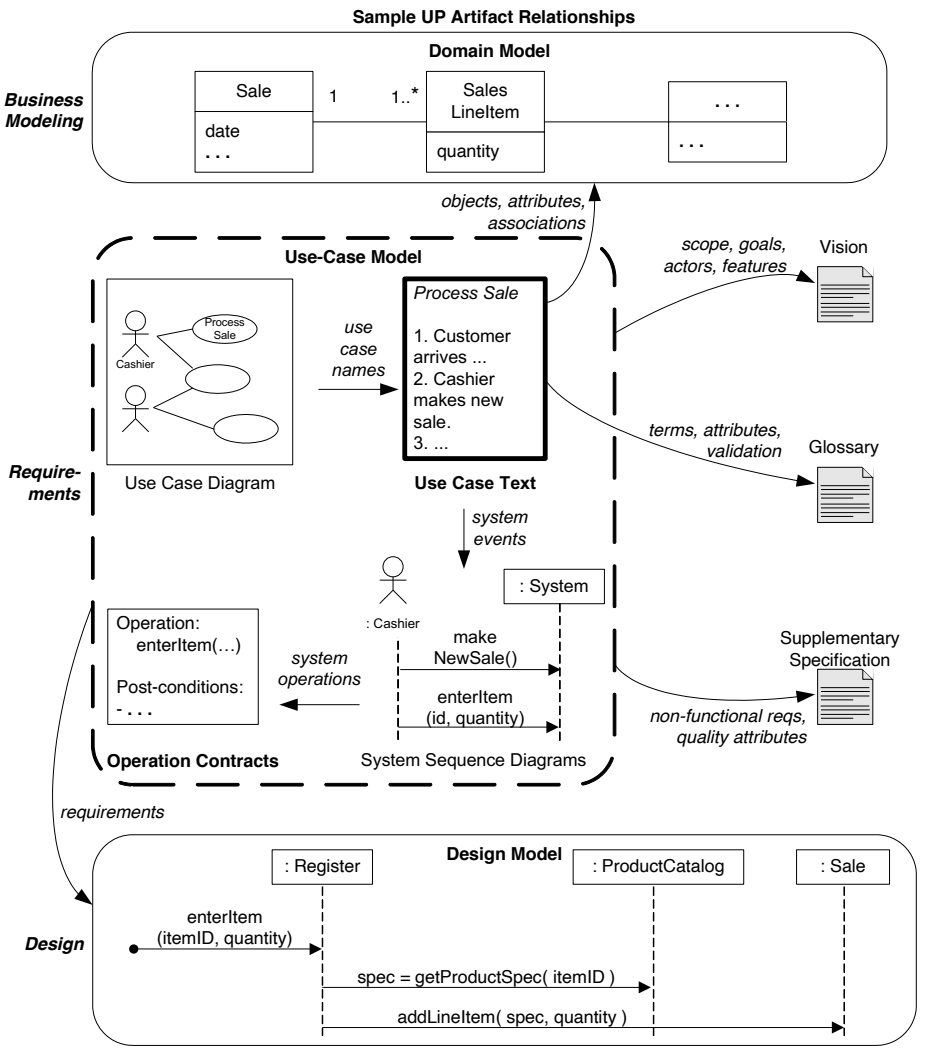
- Capture the required behaviour, not functionality

”FR1. The house shall have an alarm.”

vs

Use Case 1 “Raise Alarm”: A burglar enters the house. The house detects the presence of the burglar and notifies the owner in no later than 30 seconds.

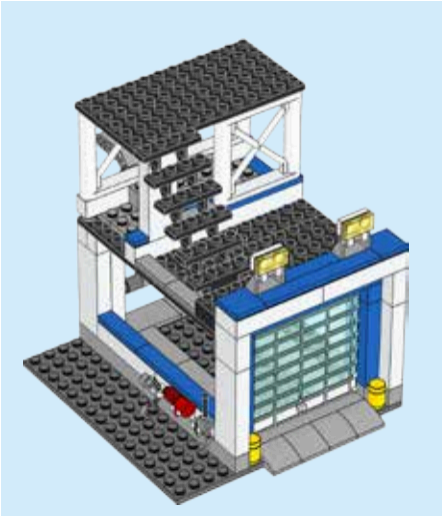
Software Models

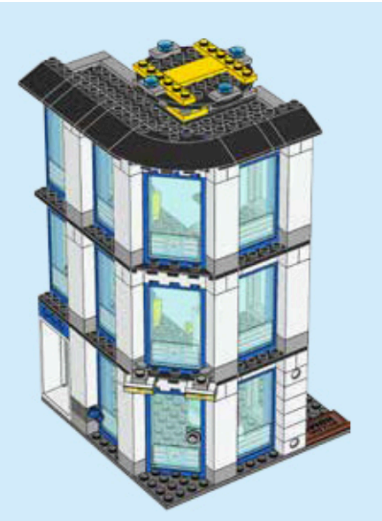


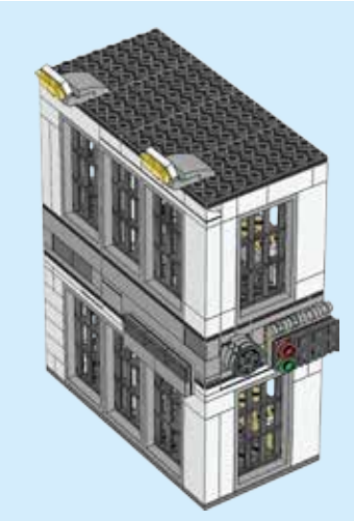
University of South Australia

Design by Decomposition

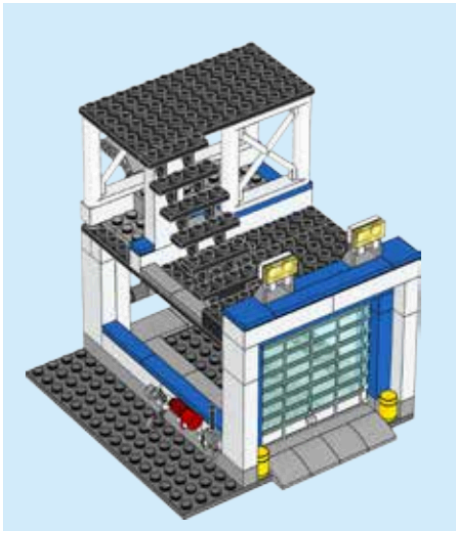
Design by Decomposition

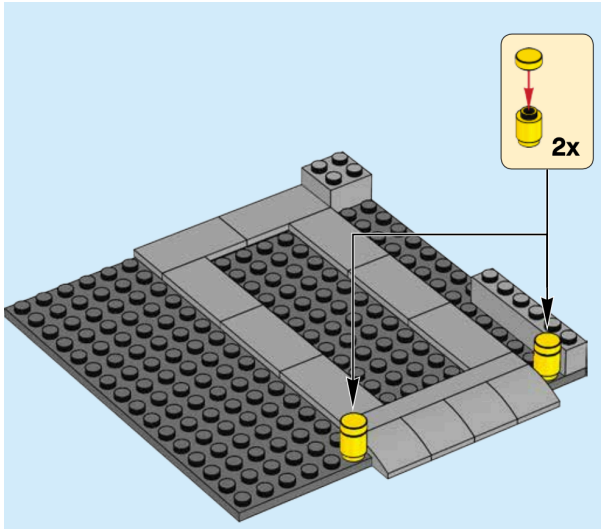






Design by Decomposition





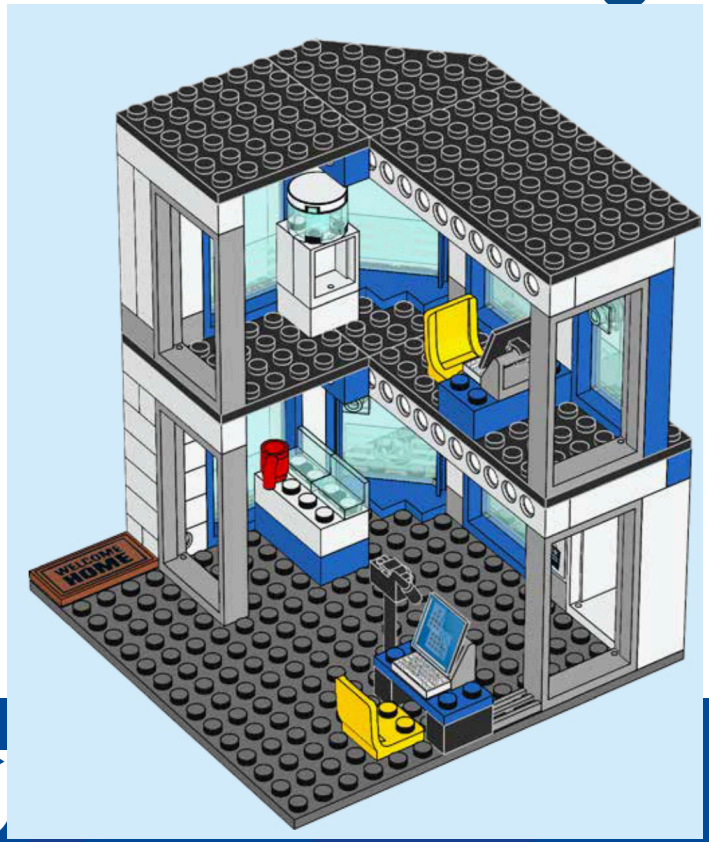
Variability Drives Design

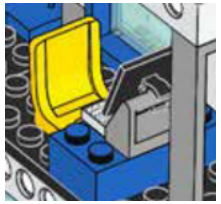
- House design is driven by variability in
  - Structure
  - Appearance
  - Utilities
  - Appliances
  - Furniture
  - Occupants
  - Environment

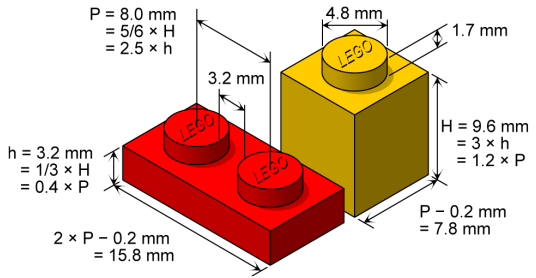
Architecture: Compose Components

https://i.pinimg.com/originals/74/c7/8a/74c78abeee9d3402fcbbd7d903e424dc.jpg

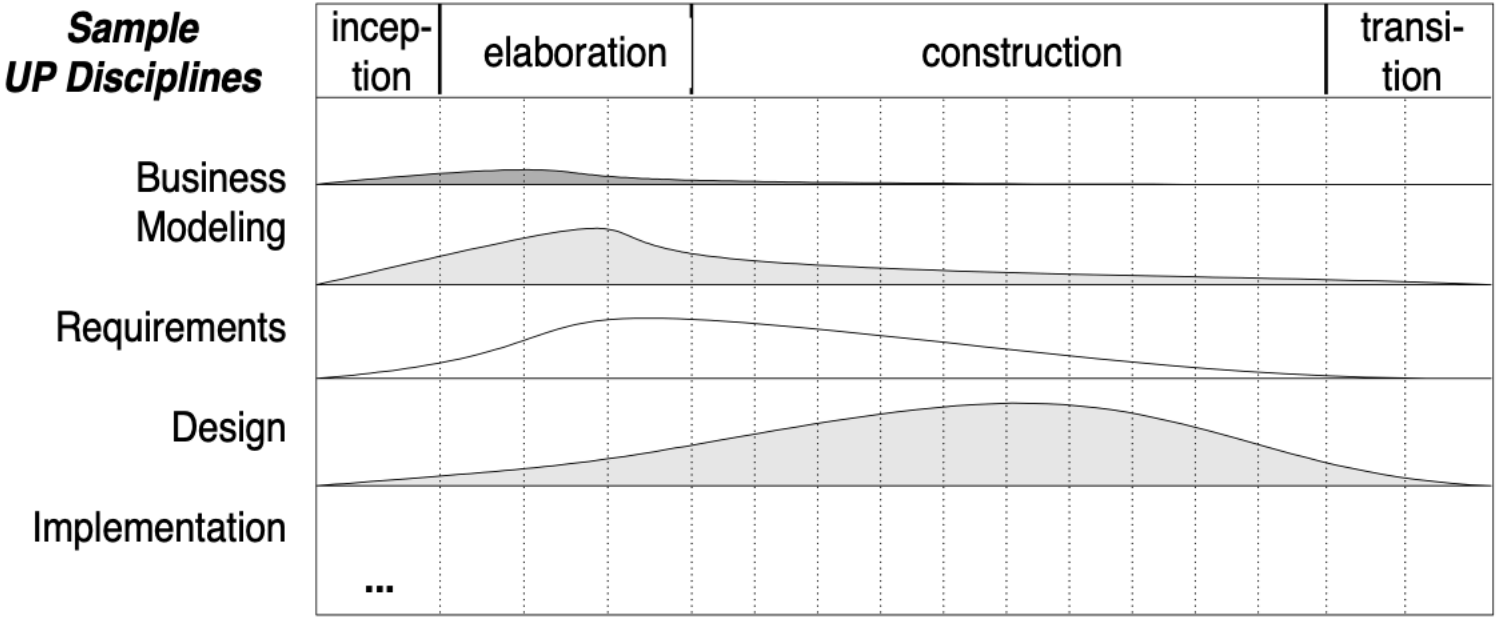
Detailed Design of Components







Software Process Phases



Summary of Steps

1. **Understand requirements**
2. **Define architecture: components and connections**
3. **Design components and their interfaces**
4. **Design implementation of components (and data stores)**
5. Develop components (incrementally)
6. Integrate & test
7. Release
8. Repeat from 3.

Summary

- Design decisions are driven by complexity and variability
- Key objective of design is to enable evolution by reducing complexity
- Requirements shall capture the behaviour of the system

Activities this Week

- Participate in Workshop 1
  - Refine and validate requirements
- Complete Quiz 1

University of
South Australia