Electronics and
Computer Science

University of
Southampton

Software Modelling and Design

COMP6226: Software Modelling Tools and Techniques for Critical Systems

Dr A. Rezazadeh (Reza)
Email: ra3@ecs.soton.ac.uk or ar4k06@soton.ac.uk

October 24

Overview

- What is Software Design?
- Design Categories
- What is Not Software Design?
- Software Design Approaches
- Software Development Process
- Modelling and making design choices

Objectives

- This lecture aims to delve deeply into the application of *design activities* in the context of *software development*.

- Software designs are more crucial as the size of the software increases.

What is Software Design?

- **Software Design** is a *Process* that, when performed, translates *Requirements* into a *Design*.
- A **Software Design Process** is a bridge between *What the Software Needs to do* (e.g., Requirements) and *the Software Implementation* (e.g., Code).
- An SDP is a description of *the steps to be performed* in order to develop software.
- **Software Design** is a *Description of the Structure and Behaviour* of Software at a *Higher Level of Abstraction* than the Code.
- **Software Design** is a *Collection of Artifacts* that describe the *Architecture*, *Data*, *Interfaces*, and *Components* of the Software.
- **Software Design** is a *High-Level Description* of the *Knowledge Represented* by the Code.

Four Design Categories

- **Architecture**: A *high-level* description of the *design elements* found within a system
- **Data**: A representations of the *logical* and *physical phenomenon/concept/artefacts* in the form of the *data structures* used by the system.
- **Interfaces**: A description of the *human–computer interface*, the *interfaces* between the *high-level design elements* (described in the architecture), and any *interfaces to external systems.*
- **Component**: A description of the *significant* or *unique processing* steps contained within a *high-level design* element.

What is Not Software Design?

- Software Design is Not Software Analysis
  - In software analysis, we focus on understanding what the software must do. Software analysis gathers and organises information that describes the needs (i.e., requirements) that the software must fulfil.
- Software Design is Not Program Design
  - In program design, we focus on the coding details of our solution.
  - Things like good variable names, good method/function names, clean method/function signatures, and the reuse of methods/functions are things we think about when doing program design.
  - In software design, we focus on a higher level of abstraction.
- Software Design is Not Programming

Software Design Approaches – Top-Down Design

- A top-down approach develops a design through a process of *subdivision*.
- You start with a high-level context of the system and subdivide this context into logically connected design elements.
- The top-down approach is also called stepwise refinement or decompositional. what aspects you will meet in your Top -Down Design?
  - You produce a high-level design (e.g., identify components of the system).
  - You then refine/decompose this design into a more detailed design (e.g., identify subcomponents for each component).
  - You then refine/decompose this design into a more detailed design (e.g., identify design elements within each subcomponent).
  - At some point, you've refined/decomposed into design elements that are small enough to be correctly implemented via code.

Top-down design – Challenges

- Decisions made at a higher level of design will directly influence the lower levels of design.
    - For example, perhaps we've decided that our *high-level design* should consist of *three components*. As we begin to *further divide* each component, we identify a *design element* that *does not fit* nicely within one of the *three components*.
- Knowing when to stop decomposing is also difficult to judge.
    - How *small* should each *decomposed design* element be before we stop? How much *detail* do we need in our design? How much design *detail* is *too much*?
- Finally, decomposing may result in duplicate design elements in two distinct parts of a system.
    - How do we avoid the possibility of *duplicate design* elements?

Software Design Approaches – Bottom-Up Design

- A bottom-up approach develops a design through a process of building up from basic elements to the entire system.
- The bottom-up approach is also called *compositional*.
  - You produce specific design elements (e.g., data structures, methods/functions, classes).
  - You then com- bine/compose these into a more general design (e.g., physical data model, class diagrams).
  - You then combine/compose these into even more general design elements (e.g., logical data model, package diagram).

Bottom-Up Design – Challenges

- Combining small design elements into a larger design may result in more rework of existing code/design.
  - In fact, the term refactoring describes this exact situation.
  - Doing lots of refactoring each time you add code may suggest that a fundamental design flaw exists in your higher-level design.
- How do we identify basic design elements from the problem description?
- Identifying and combining basic design elements may be a less intuitive approach when compared to thinking top-down (i.e., hierarchically).

Software Design Approaches – Process-Oriented Design

- A process-oriented approach develops a design by focusing on the processes/ functions being automated.
  - Here, the terms process and function refer to the activities being performed within an organization.
  - Design models created in a process-oriented approach emphasise how processing steps are organized (i.e., performed sequentially or concurrently) to meet the needs of the organization.
  - While this approach focuses on process/function, many of the process-oriented design models will show how information/data is being created or manipulated by the activities being automated.
  - The terms business workflow and business process reengineering are often used to describe this approach.

Process-Oriented Design – Challenges

- Following a process-oriented approach to designing software may place constraints on your design, limiting your ability to innovate.
  - This may happen if you focus on automating existing processes without keeping an eye on making the pro- cess more efficient.
- Another possible weakness is that data tends to be included in a design only when viewed from the perspective of the processes that create or use the data.
  - This could result in data structures that are designed based on workflow without regard to how the data is related to each other.

Software Design Approaches – Data-Driven Design

- A data-driven approach develops a design by focusing on the information/data being processed.
  - Design models created in a data-driven approach emphasise how information/data are organised (i.e., relationships between data elements) to meet the needs of the organisation.
  - While information/data is the focus, many of the data-oriented design models will show how processing steps use the information/data.
  - Doing a data-driven design approach allows a designer to develop the data structures first, and then apply these structures within the appropriate processing elements identified in the design.

example of the approaches

Data-Driven Design – Challenges

- Creating a data-driven design may limit your ability to streamline the processing involved in using the data.
- This is because the design is focusing on data relationships instead of processing steps.
- The weaknesses of the data-driven approach is analogous to the process-oriented approach.
- Both of these design approaches can result in weaker designs since their focus is on only one aspect of software.

Software Design Approaches – Object-Oriented Design

- An object-oriented approach develops a design by focusing on both process and data. In this approach, processing steps and information/data are combined into design elements called classes.
  - A class encapsulates the processing and information into a single software element.
  - Creating an object-oriented design has the benefits associated with the process-oriented and data-driven approaches, while addressing many of the weaknesses of these two approaches.
  - However, the notion of a class as a software abstraction is not an easy thing to learn.
  - It often takes years to develop the experience necessary to consistently develop good object-oriented software designs.

face the challenges

Software Design Approaches – Function-oriented

Web application use

- Another design approach included in an SDP may involve doing a function-oriented design or doing an event-driven design.
- A function-oriented design focuses on functions as the primary software abstraction.
- Functions can be described at a high-level of abstraction and then be subdivided into smaller functions that may be directly translated into code.

Software Design Approaches – Event-driven

- An event-driven design focuses on how software should react to external stimuli.
- Certain types of software, including embedded software and graphical user interfaces, use this design approach.
- The design identifies the types of events to occur along with how the software should react to each type of event.

some examples :real-time system

Hybrid Design Approaches

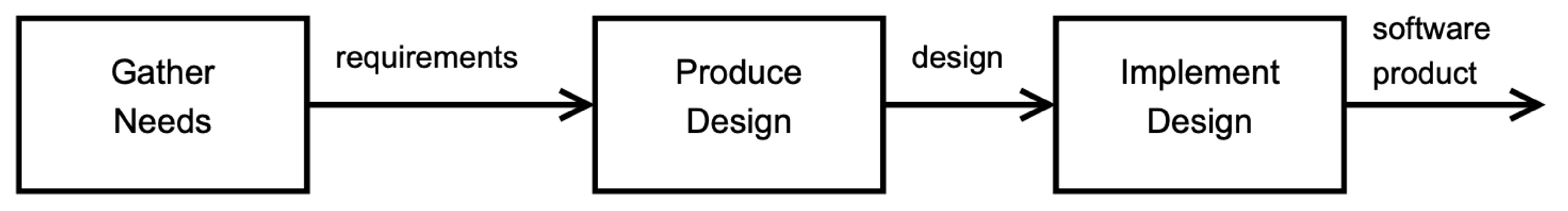
- Often, a design phase in an SDP *does not strictly adhere to just one* of the above approaches.
  - For example, a design phase may describe a series of steps to be per- formed where some of the steps ask a designer to think about the entire system, with the goal of establishing some *high-level design structures*. These steps have the designer following a top-down approach.
  - Other steps in the design phase may ask a designer to think about specific details of the system, with the goal of establishing some detailed design elements. These steps have the designer following a bottom-up approach.
  - The use of both top-down and bottom-up approaches is a natural way to develop a software design.

Hybrid Design Approaches – Cont.

- Similarly, a design phase in an SDP may include steps describing the use of process-oriented, data-driven, and object-oriented approaches.
  - For example, a design phase may include steps describing the development of design models using the *Unified Modelling Language (UML)* (i.e., object-oriented approach);
  - Development of logical data models using entity–relationship notation (i.e., data-driven approach);
  - and development of *workflow diagrams* representing activities to be automated (i.e., process-oriented approach).

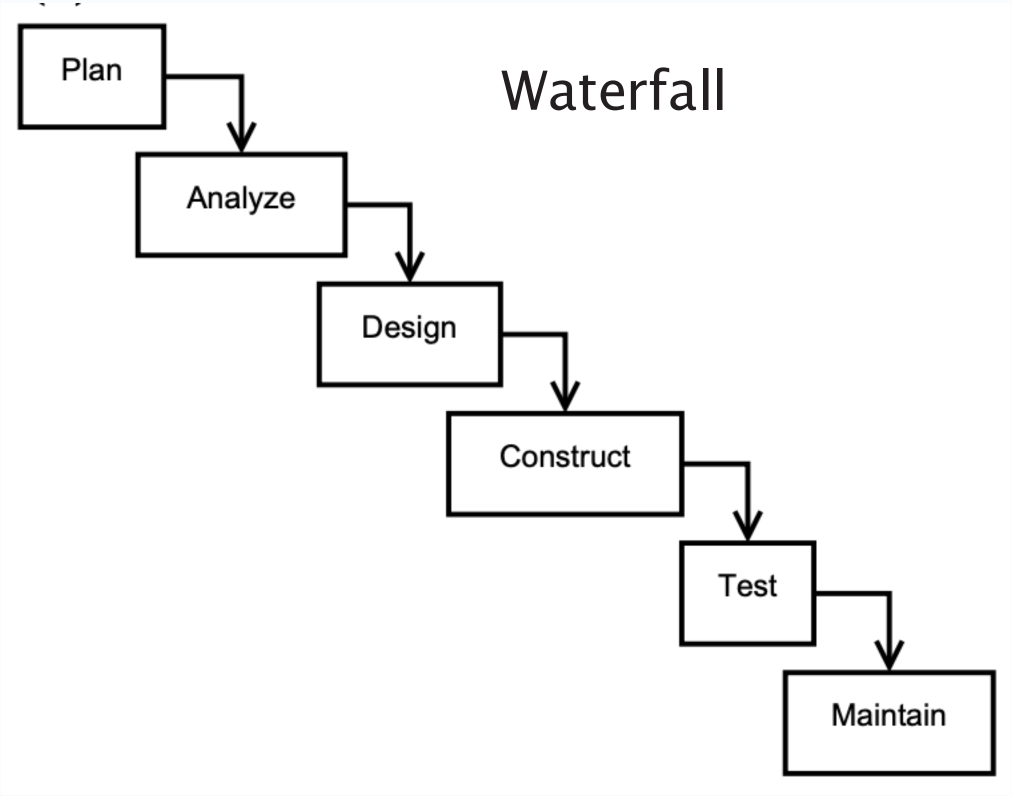
Software Development Process

- A software development process (SDP) is a series of steps describing the tasks to perform to create a software product.
- There are many different types of SDPs, each describing a unique approach to developing software.
- While each type of SDP describes a different approach, all SDPs have a few key things in common.



Common steps in SDPs

Different Software Development Processes



Waterfall

Plan

Analyze

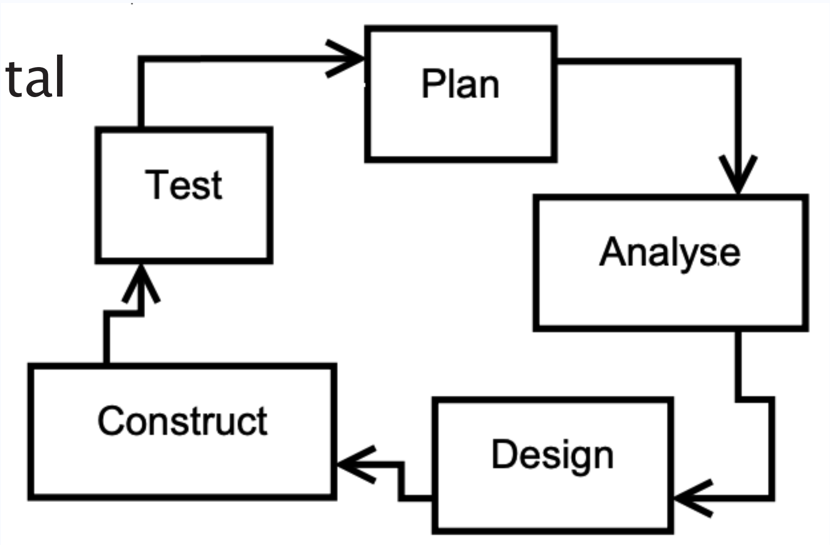
Design

Construct

Test

Maintain

Incremental



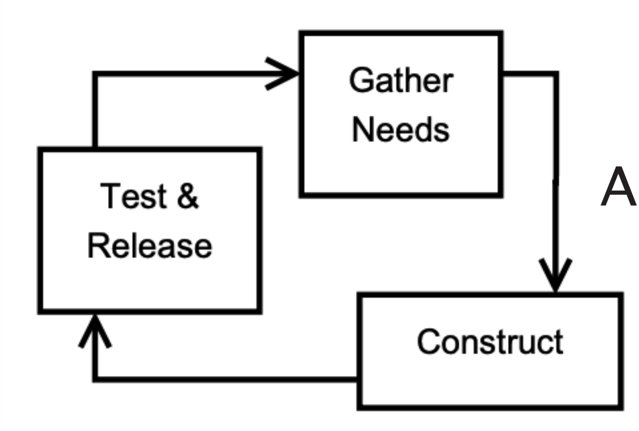
Plan

Analyse

Design

Construct

Test



Agile

Gather Needs

Construct

Test & Release

difference between design and modelling

Modelling and making design choices

- When describing what designing is about, we might say that our aim is to produce a **design solution** that will meet a specific need.
- The act of designing something usually involves identifying suitable options and represent them for the key features of our 'design solution' and then evaluating the consequences of adopting each of them.
  - The documentation of this *representation* is called *modelling.*
  - For example, one of our first decisions may well be to choose an appropriate architectural style for our ap- plication.
- When we carry out modelling, a *representation* is used to provide a particular abstract description of particular characteristics of the designer's ideas for an application.

Why do designers use models?

- Models are typically needed for purposes such as:
  - Allowing the designer to *express/document* their ideas about their ‘solution’
  - Helping the designer to *explain/communicate* their ideas to others (such as customers, fellow designers, implementors, managers);
  - Assisting the designer to *check/validate* he design model for important features such as completeness and consistency.
  - Modelling can guide designer to explore different ideas and to uncover problems

What is Abstraction?

- In order to create manageable models of the *intended system* and envisage how it will behave, we need to find ways to put aside the details that don't matter and to focus upon the factors that do matter.
- The concept of *abstraction* is an *essential* one when *formulating design ideas* in any branch of engineering.
- The process of *abstraction* involves *removing* or *omitting unneeded detail* from a *description of a model* or plan, while still *retaining* a description of those *properties* that are relevant.
- Doing so *reduces* the *cognitive load* involved in *understanding* particular aspects of a design model.
  - For example, if the requirements indicate the need for a *baseball*, *softball*, and *kickball*, then we should see an opportunity to *generalise* these into the need for a *ball* that is a *sphere*.

Designing as a creative process

- A *distinctive characteristic* of humans is the ability to use tools to *create new artifacts*, where these are new in the sense that we end up with something that did not exist before.
  - The meaning of '*artifact*' is "something made by humans"
- **Creativity** requires the ability to *conceptualise*
  - inventing or contriving an idea or explanation and formulating it mentally
- **Creativity** involves the ability to "think outside the box" and to find new and effective ways to meet a particular need.
  - While **creativity** may occasionally involve coming up with a radically different way of doing something, quite often the creative element will be realising that doing things a bit differently might produce something that is more effective, *robust*, *elegant*, or some *combination* of such *attributes*.

YOUR QUESTIONS