MODERN SYSTEM DESIGN AND SOFTWARE DESIGN PATTERNS (DESA)

Course Contents

A. System Scoping

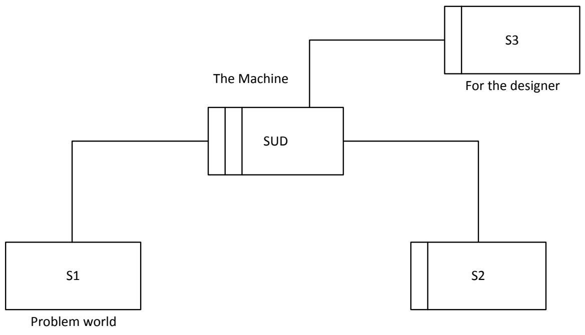
This is the first stage of the software project and the main goal is to understand the problem that we are trying to solve before we start on a design of the corresponding software system (which we call SUD or System *Under Discussion*). In particular, we wish to understand what the system to be developed must deliver and to this end we first determine what the boundaries of the system are. We achieve this end by identifying the external systems with which SUD exchanges information and data. Then we integrate the system requirements and desired features with the system model.

The products from this stage are:

- A context diagram comprising SUD and its satellite systems.
- Each system has well-defined responsibilities; system services have been identified.
- System requirements and features aligned with the systems in the context diagram.

System Context

- System goals and core process
- Major data flow; linking major output to input
- Discovering external systems
- Initial context diagram



Sample Context diagram

System Requirements and Features

- Viewpoint-oriented requirements determination
- Interviewing techniques and requirements elicitation
- Functional and non-functional requirements

System Responsibilities and Services

- What is a system service?
- *Provides* and *requires* services
- Assigning services to systems
- Documenting services

Proof-of-Concept I: Feedback from Domain Experts

- Getting critical *early wins* and results
- Major data flow
- Initial (numerical) examples
- Reaching consensus with the system architect
- Initial mission design blueprint

Proof-of-Concept II: Prototype

- What do we wish to achieve?
- Viewpoints: architect, designer and project manager
- Creating a prototype
- Review and planning for the next stage

B. Detailed Architectural Design

In this stage we refine and extend the context diagram to a form that is more closely related to software architectures. We produce an unambiguous description of system components, their interfaces and how they communicate. We consider several candidate *connection architectures* that we can implement using objects, interfaces and modules during the detailed design stage. We do not yet commit to a specific programming style or language and we keep the design flexible by creating *logical interfaces* that we can later implement using a combination of object-oriented, generic and functional programming styles.

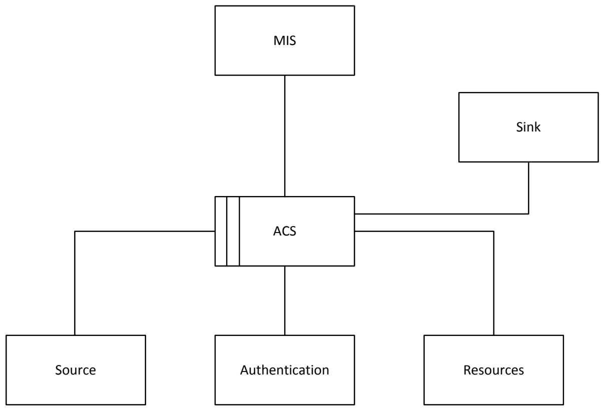
The products from this stage are:

- Specification of the components of a system and the communication between them.

- *Connection architectures*: object, interface, plug and socket.
- Discovering reusable components using *Domain Architectures* and *Architectural Styles*.
- Pay some attention to multitasking programs and parallel programming systems.

Domain Architectures

- What is a domain architecture?
- Behavioural and structural properties
- The *big five* categories (MAN, ACS, RAT, PCS, MIS)
- Analogical reasoning and system discovery



Acces Control System

Architectural Styles

- Design vocabulary and allowed structural patterns
- Computational model; relationship with Domain Architectures
- Examples (categories) and specialisations
- Basis of Communication (shared state, events)

Multitasking Applications

- Task and data decomposition
- Data dependency graph
- Concurrency versus parallelism
- Coarse-grained and find-grained parallelism

Component Connection Architectures

- System specification
- Provided and required behaviour features
- Conforming components
- Configuring an architecture

C. System Decomposition Patterns

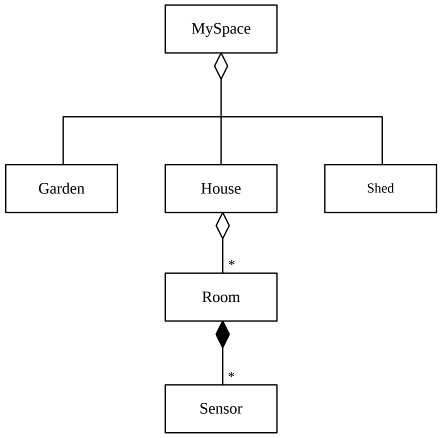
The goal of this stage is to carry out a detailed decomposition of system components into more fine-grained modules and objects. We address the full lifecycle from the object creation process to defining object structure and the desired level of behavioural flexibility. We decompose the components to a stage whereby it is possible to begin with the detailed design of classes and modules as well as determining the level of flexibility that they should satisfy.

The products from this stage are:

- A detailed overview of the most important system and parallel design patterns.
- Decompose logical components into more fine-grained objects.
- System assembly using assemblies, static and dynamic link libraries.

High-impact of Systems Patterns

- (Super) Builder and sub-contractor factories
- Mediator
- Whole Part and Composite
- Layers
- Statecharts and state machines



Whole-Part pattern

Parallel Design Patterns

- Divide and Conquer
- Geometric Decomposition
- Pipeline
- Event-based coordination

The Actor Model

- What is an actor?
- The message-passing model; message blocks
- Sources and targets
- Starting and stopping an actor-based application

Configuration, Packaging and Lifecycle Specification

- Assemblies, static libraries and dynamic link libraries
- Lifetime management of multi-levelled domain architectures
- Metadata and reflection patterns
- Maintainability and version control

D. Modern Software (GOF) Design Patterns Process

The main goal of this stage in the software project is to create the *detailed design blueprints* that are used as input to programming activities using a multiparadigm style in languages such as C++, C# and Java. We commit to a given language and specific interface implementation. We also pay attention to reusability issues by determining if we can use standard libraries instead of creating home-grown code. Finally, we need to determine the levels of portability, maintainability and efficiency that the software modules, classes and components should satisfy.

The products from this stage are:

- Design blueprints that can be directly implemented in order to produce a working prototype.
- Plans for the next prototype (allocation of resources).
- Discovering new, evolving and missing requirements.

Modelling Classes

- Inheritance and composition
- Aggregation and association
- What constitutes a good object model?
- Object-oriented software metrics

Structural Design Patterns

- Adapter
- Bridge
- Proxy
- Façade

Behavioural Flexibility

- Command
- Strategy and Template Method
- Next-generation Observer and Delegates
- Visitor
- Role (Facet) pattern

E. Test Cases and Model Problems

In this section we examine a number of ready-to-run applications from various domains to show how we applied the design principles and software patterns that we discussed in this course. The applications have common features but each one has its own particular requirements that need to be satisfied.

Environment controller (Process Control)

- Product vending machine (Access Control)
- Customer helpdesk (Resource Allocation and Tracking)
- Extended builders (Manufacturing)
- High-level reporting (Management Information)
- Environment controller (Process Control)

Your Trainer

Daniel J. Duffy started the company Datasim in 1987 to promote C++ as a new object-oriented language for developing applications in the roles of developer, architect and requirements analyst to help clients design and analyse software systems for Computer Aided Design (CAD), process control and hardware-software systems, logistics, holography (optical technology) and computational finance. He used a combination of top-down functional decomposition and bottom-up object-oriented programming techniques to create stable and extendible applications (for a discussion, see Duffy 2004 where we have grouped applications into domain categories). Previous to Datasim he worked on engineering applications in oil and gas and semiconductor industries using a range of numerical methods (for example, the finite element method (FEM)) on mainframe and mini-computers.

Daniel Duffy has BA (Mod), MSc and PhD degrees in pure and applied mathematics and has been active in promoting partial differential equation (PDE) and finite difference methods (FDM) for applications in computational finance. He was responsible for the introduction of the Fractional Step (Soviet Splitting) method and the Alternating Direction Explicit (ADE) method in computational finance. He is also the originator of the exponential fitting method for time-dependent partial differential equations.

He is the originator of two very popular C++ online courses (both C++98 and C++11/14) on www.quantnet.com in cooperation with Quantnet LLC and Baruch College (CUNY), NYC. He also trains developers and designers around the world. He can be contacted dduffy@datasim.nl for queries, information and course venues, in-company course and course dates.

References

Duffy, D.J. (2004) Domain Architectures, Wiley Chichester.