COMP23420 Revision

Basics

Why Do Software Projects fail?

- Unrealistic goals
- Badly defined system requirements
- Poor communication

The list goes on.

Functional & Non-Functional Requirements

Functional

- *Relates directly to a specific process the system should perform or information it should contain*
- *The system should **DO** <req>*

Non-Functional

- *Relates to how the system should operate as a whole, e.g., performance, ethical standards*
- *The system should **BE** <req>*

Business Process

- *A set of structured tasks that gives rise to a service or product*
- *Requirements gathering uncovers the business process*
- *Functional modelling describes the business process*

Activity Diagram

- *Models both the "is-as" and "to-be"*
- *A **logical model** - describes what it does*
- *Captures the activities and how objects flow between them*
- *An **action**: a non-decomposable piece of behaviour*
- *An **activity**: a composed set of actions*
- ***Control Flow**: shows sequence of execution*

Requirements Analysis & Specification

A requirement is a statement of what the system must/what characteristic it must have.

Actors and Use Cases

- Perform ***use cases*** upon a system
- Have uses cases performed upon them
- An entity, human or otherwise that either directly uses or is used by the system
- Activities in an ***activity diagram*** reflect the things that will happen in the business process that will be realised in the new system
- Activities map some form to use cases
- Need to know ***entry/exit conditions*** (what it true/before after the use case)
- UC's uses CRUD, describes functional requirements and written in verb-noun form

Realising Use Cases, Domain and System

- Realised by domain classes

Domain Class

- Sum total of its responsibilities in all the UCR's in which it plays a part
- Refined into system classes
    - Real classes used for implementing the system
    - May be refined further into classes used in final impl'
- Conceptual classes defined in domain model
- Captures most important type of entities + relationships
- DM defined as: set of classes with attributes, no operations
- Inside iterative process
- Attributes should be primitive

System Class

- Created from DC's, designed for impl' environment
- Show software objects and include operations as well as attributes
- High cohesion: class defines a single entity
- Low coupling: class interacts with as few other classes as possible
- **Refactoring:** Create modules that account for similarities/differences between units of interest
- **Partition:** Create smaller (subsystem) or larger units, group units that collaborate, objects in same partition have **high cohesion**
- **Layers:** Separate app logic from UI logic

- **Design Pattern:** Reusable design that can be customised to many recurring problems

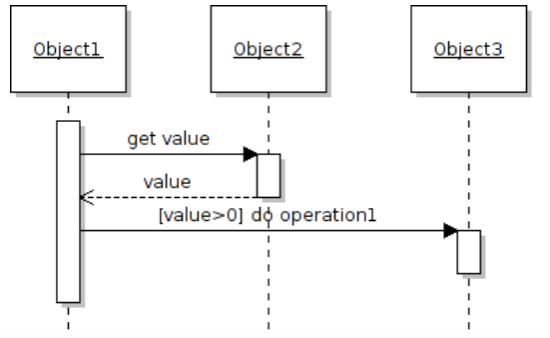
Ethics in SE

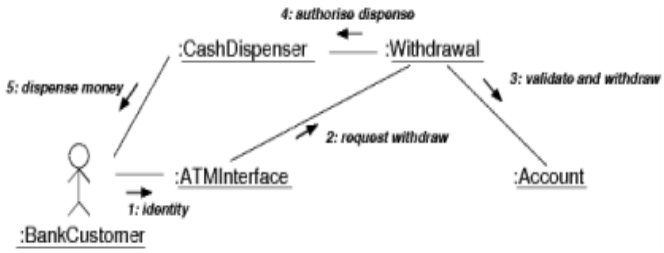
- **Morality:** Set of values that are widely shared with a community
- **Ethical Principles:** Attempts to operationalise/critically examine the moral that guides human conduct
- Ethics types: Virtue, duty-based, utilitarian, applied

Behavioural Modelling

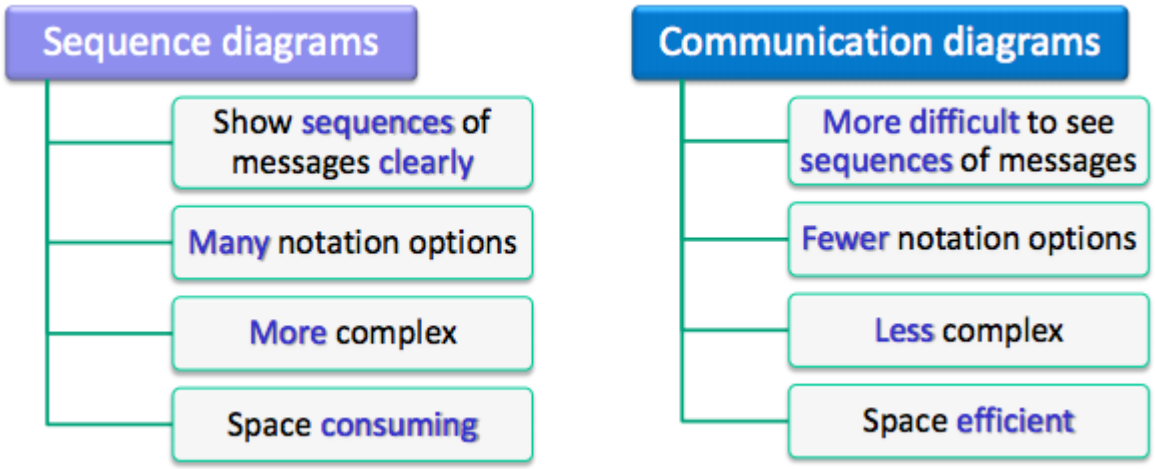
Communication & Sequence Diagrams

- All use cases need to be realised
- Behaviour of objects that collaborate in UCR can be specified as a ***sequence of interactions between them***
    - Interactions are messages and subsequence method executions
    - Defined by a sequence/communication diagram



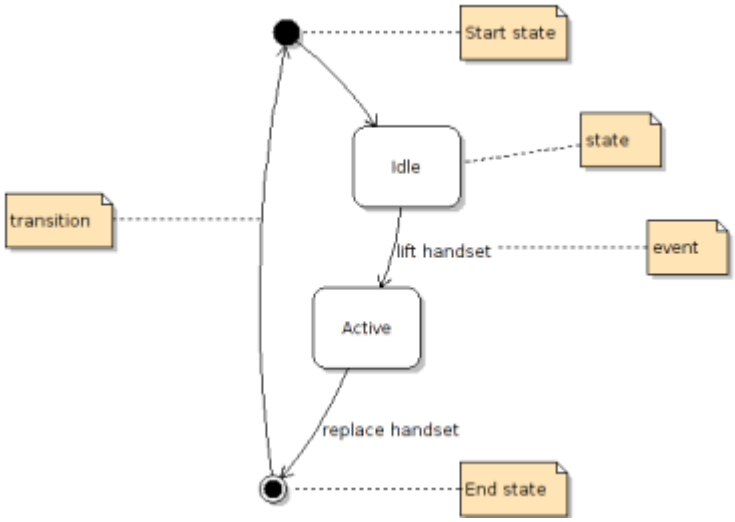


Differences Between Communication and Sequence Diagrams



State Machine Diagram

- Specifies ***internal*** behaviour of a ***single object***
- A SM has ***states*** and ***transitions between states*** *(triggered by events)*



Software Development Process

Sequential

- Strong emphasis on well-defined complete phases
- Each phase has well-defined deliverables
- “ “ deals with complete systems, feedback may cause adjustment

Waterfall

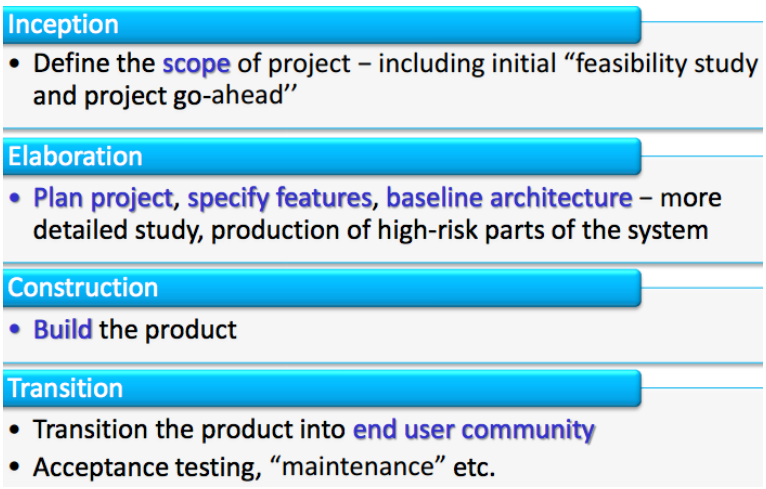
- Iteration between phases (in sequence, deliverables at each phase)
- Costly if late-on problems
- Everything well-documented
- Specialisation can cause communication problems
- Doesn’t work as:
    - Won’t get req’ right first time, customers will likely change req’

Iterative

- **Iterates** over well-defined **cycles**
- Each **cycle** has short **phases**
- Each **phase** has well-defined **activities** and iterations produces product release

Unified Process

- OO systems, industry-standard
- Assumes you can’t get everything right at start
- Phases, disciplines, artifacts



- **Timeboxed:** If you don’t get it right you change the plan (not extend iteration)

Agile

- **Iterates** over short **cycles**
- Each iteration produces part of the SW
- SW presented to customer after each iteration

- Requirements changes checked with customer after each iteration

XP

- Req. determined from user stories
- Unit tests before code, pair programming, simplicity, refactoring