Unified Process

The Unified Process Is ...

- Architecture-centric
- Use case driven
- Supportive of object-oriented techniques
- Configurable
- In favour of quality control and risk management
- A disciplined approach showing how to allocate development work activities and responsibilities within software team
- An ordered series of iterative steps of how to go about writing software
- A collection of best practices for software development management
- Encourages use of UML to provide its graphical modelling notation

History of the Unified Process

- Started at Ericsson in Sweden
  - Concept of “traffic cases” introduced, which later became use cases
  - Reusable block diagrams
  - The AXE project was a “bet the company” project for Ericsson, but it worked well and sold worldwide in large numbers, hence the success of the company
- Objectory AB (Jacobson left Ericsson to set up Objectory)
  - Workflows
- Rational Approach (Four views - architectural, process, physical and development)
- Rational Objectory Process; Merger of Rational & Objectory
- Unified Modelling Language & Rational Unified Process
- IBM buys Rational in early 2003

The Four 'P's

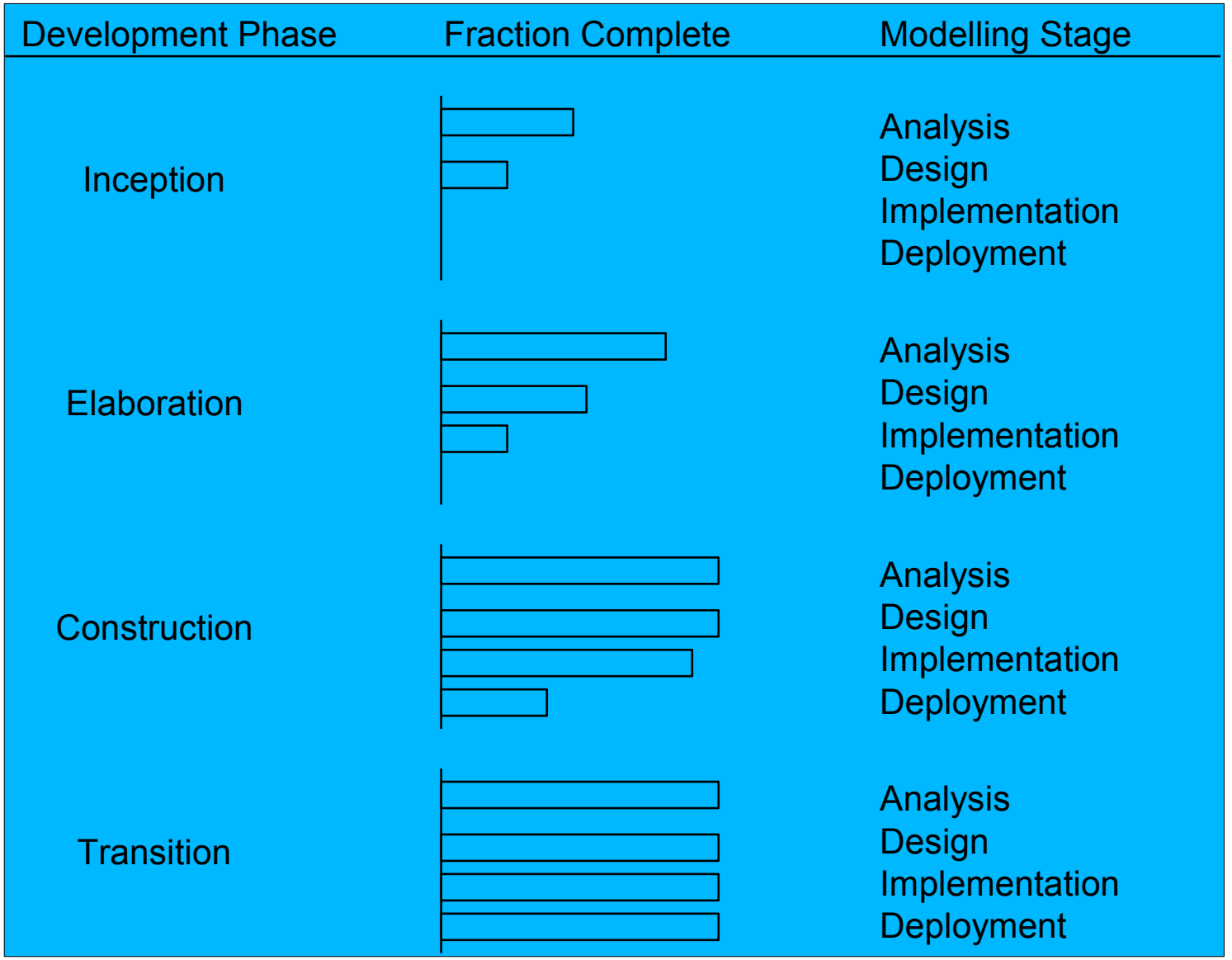
- People
  - Differing skills, experience - must make the project happen
  - Perform different roles during project’s lifetime
- Project
  - A project is the collection of activities which finally result in a product
  - A project is an “instance” of a process
- Product
  - Complete system, including software binaries and all other collateral
- Process
  - A process is a description of how to conduct activities to create products

Phases of the Unified Process

- Inception
  - Scoping the project
  - Stating Business Case for or against
- Elaboration
  - Requirements Analysis & Risk assessment
  - Project Plan
  - General Architecture
- Construction
  - Develop the system using iterations
- Transition
  - Productising (testing, documentation, roll-out)

Modelling Stages

- Analysis
  - Capturing the requirements and gaining an in-depth understanding of the system to be built and the environment in which it will operate
- Design
  - Creating a sound architecture which will satisfy all the requirements and comply with various OO guidelines
- Implementation
  - Build the system (coding and testing)
- Deployment
  - Putting the system to work in a customer’s site (installation, configuration, maintenance)


Development Phase
Fraction Complete
Modelling Stage
Inception
Analysis
Design
Implementation
Deployment
Elaboration
Analysis
Design
Implementation
Deployment
Construction
Analysis
Design
Implementation
Deployment
Transition
Analysis
Design
Implementation
Deployment

Issues

- During each development phase, different modelling stages are carried out
- Some modelling stages are worked on more than others in some phases
- e.g. analysis is emphasised during the inception phase and deployment during the transition phase
- During each iterative release of the system, the phases will occur in sequence, but there may be multiple iterations of one phase before moving onto the next
- Different parts of the system are moved from analysis to deployment at their own pace
- It is an iterative process, with the goal that all modelling stages move to completion by the end of the transition phase
- We wish to write as little software as possible

The Iterative Approach

- The iterative approach to software development is recommended as it:
  - Provides for deliverables in small incremental steps
  - Allows you to always have a running system
  - Good for customer demos and validation
  - Useful to test new versions of components against running system, and not just some simulation of it
  - Avoids the “big bang” approach to integration at the end of a project (when time is usually desperately in short supply)
- Each iteration becomes a mini-waterfall
- Get the first iteration right is the problem

Why the process is controlled by use cases

- Use Cases are the primary representation of behavioural requirements
- A use case is one way in which a user may use the system
- The iterative approach to development encourages that every iteration completes a few use cases
  - Instead of x amount of code or a certain number of classes
- If a requirements document specifies 50 use cases, perhaps you might plan five development iterations, each which will add support for 10 use cases
- After each iteration more functionality is exposed to the user
- The user (who after all is paying for the system) can see the progress of the development of the system after each iteration

- Choreographing

Use-Case Driven

- What is the system supposed to do for each user
  - From a user's perspective, how is value added to the product
  - "A use-case is a piece of functionality in the system that gives a user a result of value"
  - Use-cases drive the entire process
  - Use-cases initiate the process
  - Design and implementation models realise the use cases
  - Progress is measured by number of completed use cases
  - Testing is based on use cases
  - User documentation can be based on use cases

Architecture-Centric

- There needs to be “something” which adds structure to the solution
- That “something” is called architecture
- Use cases describe users’ requirements and architecture describe the form the solution takes
- Think of architecture as a description of how the platform works
- This can initially be independent of use-cases, but later should take into account the most important 20% of use cases
- Architecture and use cases evolve in tandem
- Need to cover the risky aspects of a project early, and then fix stable architecture

Goals of Architecture

- Encourage parallel development
- Minimises re-work, and ensure that when work is discarded only small amounts are lost
- Increase component reuse
- Helps maintainability
- The architecture serves as a blueprint for everyone working on the project (makes sure they are all on the same track!)

Iterative and Incremental

- The iterative approach is neither chaotic nor anarchic
  - The four phases - inception, elaboration, construction and transition - add structure and control progress
  - There is a slight overhead per iteration - we do not want to duplicate work and need to minimise administration -
  - We must achieve something useful each iteration
  - Some iterations extend the previous iteration - and some iterations replace parts of the previous iterations
  - The replacing should occur early in the project
  - Later in a project, all iterations should be additive
  - “Be surprised early”
  - All surprises should come early in the project’s lifetime
  - Near the end should be totally predictable, and almost boring

Controlling Risk

- There is risk associated with each part of software development
  - Will we get it written on time, to the right quality etc.
  - Are users interested? Will the requirements change?
  - Will the marketplace conditions changes?
- You need to identify risk, quantify it and as development progresses eliminate it
  - Need to get all team-members (architects, programmers, testers and doc) to deliver at regular intervals, to ensure it all “fits together”
  - High risk is eliminated early in iterative development, but late in waterfall model
  - Cover risks early in the project, when fixing them, changing the spec or cancelling the project is easy
  - And not near the end, when all the time/money has been spent!

Artefacts

- Artefacts are the deliverables associated with a software intensive system grouped into sets
- Requirements Set
  - Use case models, UI prototypes, requirements, description of business issues and context of system
- Design Set
  - Description of how the system will be constructed - static and dynamic design models
- Implementation Set
  - Source code, data files, database schema
- Deployment Set
  - How software is put on customer’s machine (installation and configuration management)

Deliverables

Inception Phase Deliverables
Project Description
Risk Analysis
Use Case Diagram
Description of actors and use cases
Project Proposal

Elaboration Phase Deliverables
Detailed Primary Scenarios
Secondary Scenarios
Activity Diagrams
User Interface Diagrammed
Architecture
Project Plan

Construction Phase Deliverables
Iteration Plans
Code
Test plans and results
Reviews of iterations
User guides/training manuals
Demo version

Transition Phase Deliverables
Installation & Configuration scripts
Sales and marketing materials
Maintenance issues

Unified Process Vs. Rational Unified Process

- The Unified Process is a theoretical description of how software development should be organised
  - Looks at workflows such as requirements, analysis, design, implementation and test
  - Places emphasis on architecture
  - The Rational Unified Process is a product, now in version 5.0, from the Rational Corporation
  - Covers everything in the Unified Process, and covers additional workflows such as business modelling, project management and configuration management
  - Rational Unified Process is a web-based product which works as an information store for stakeholders in a software project

Books

- “The Unified Software Development Process”, By Ivar Jacobson, Grady Booch, James Rumbaugh, 1999, Addison-Wesley, ISBN: 0-201-57169-2
- “The Rational Unified Process”, Phillipe Kruchten, 2000, Addison-Wesley, ISBN: 0-201-70710-1
- “UML and the Unified Process”, Lilana Favre, March 2003, Idea Group Inc., ISBN: 1931777446
- “Software Project Management: A Unified Framework”, Walker Royce, 1998, Addison-Wesley, ISBN: 0-201-309-580
  - Excellent book - very strongly recommended!
- “The Road to the Unified Software Development Process”, Ivar Jacobson, ISBN:0-521-78774-2, SIGS, 2000(backgrounder on its evolution)

Tools

- The Rational Unified Process, a web-enabled information resource
- See www.rational.com
- A project management tool such as Microsoft Project could be used to control a project using the Unified Process approach
- Could easily create your own tool, based on a web-server, a database and some HMTL forms

Models

- A set of models completely describe a software project
- Organisation of projects is model-driven (controlled by models)
- A model is an abstraction of a system
  - Relative to someone's viewpoint (customer, developers, tester etc.)
  - Models are independent yet inter-related!
  - Self-contained
  - Only one interpretation should be possible
- A model is an accurate simplification of reality which helps us understand what we are working on
- These models may be linked using trace relationships
- This means that an element in one may be related to an element in another
- Helps understanding and change control

Models in the Unified Process

- Use-Case Model
  - A description of how users utilise the system
- Analysis Model
  - A more rigorous requirements description
- Design Model
  - How sub-systems and their classes realise use cases
- Deployment Model
  - Describes how components run on computing nodes
- Implementation Model
  - The internals of the solution
- Test Model
  - How the solution is verified to perform all necessary tasks

Milestones

- Think of milestones as synchronisation points between development team, project manager and customers
- Some developers over-estimate time needed
- Just as dangerous, some enthusiastic developers under-estimate how long it takes
  - As a project manager, which would you prefer?
- At every milestone, you can decide:
  - To continue
  - To re-work last iteration if problems have been discovered
  - Fix schedule
  - Cancel project
  - Decide has the expected workitems really been completed

Workers

- The term “worker” is used to denote a position held by a human during the workflows
- Describes the responsibilities, expected behaviour and prior experience
- One person may be different workers during a project, and one worker may require multiple people on larger projects
- Workers in the Unified Process:
  - Process Engineer
  - System Analyst
  - Use Case Specifier
  - User Interface Design
  - Architect
  - Use Case Engineer

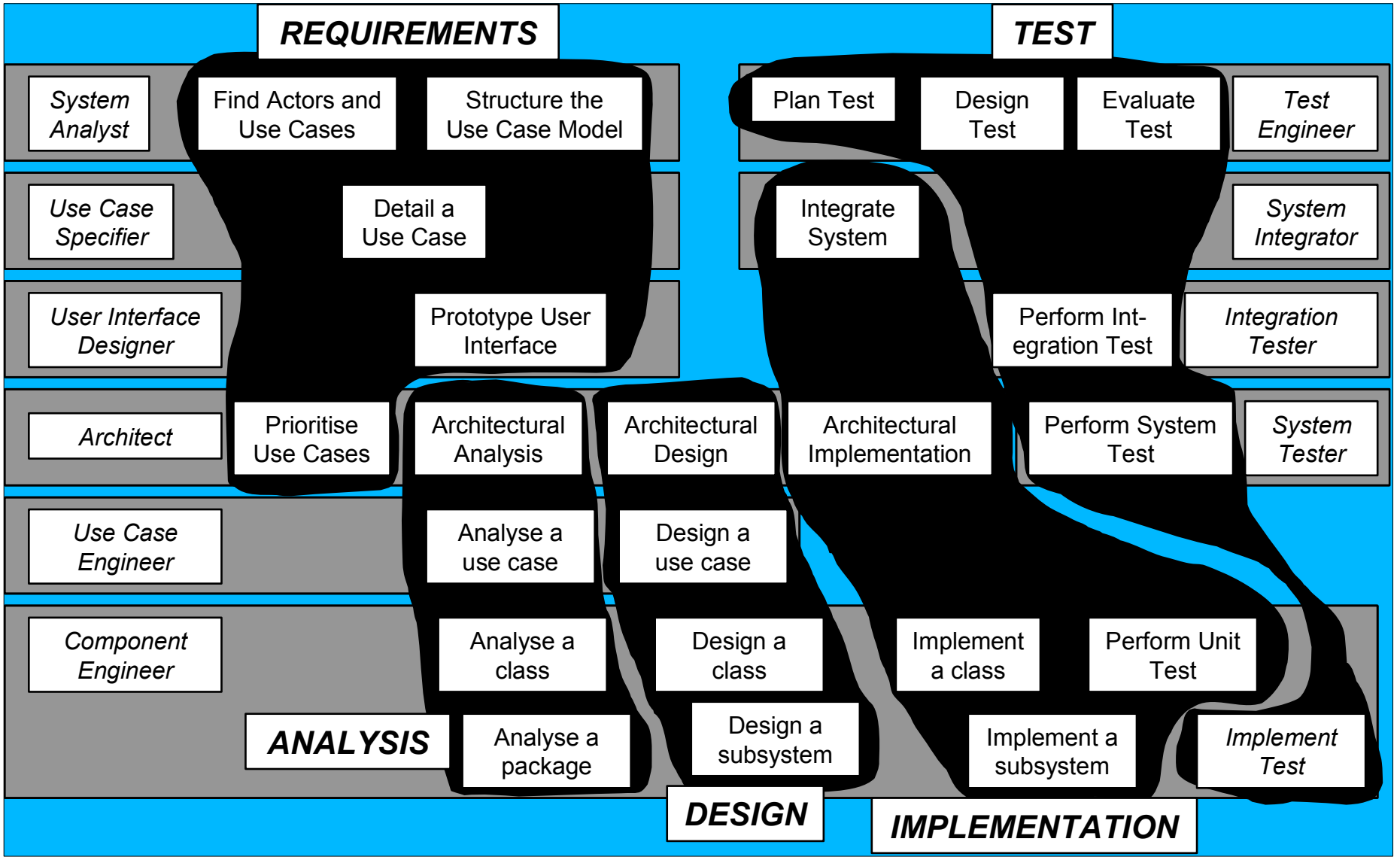
» Use Case Engineer
» Component Engineer
» Test Engineer
» System Integrator
» Integration Tester
» System Tester

Workflows

- Workflows are ordered series of activities, performed by different workers
- As workers perform activities, they create and modify artifacts
- Artefact output from one activity can become an input for the next activity

Unified Process Workflows

(Adopted from Jacobson et al., 1999, p324)



Workflows In More Detail

- To better understand workflows, we will examine two in more depth
  - Requirements workflow
  - Analysis Workflow

Requirements Workflow

- The aim of the requirements workflow is to point us in the right direction
- Describe what needs to be done
- Get agreement from client on this, before main work begins
- Enable the preparation of accurate estimate

System Context

- Software systems must “fit” within a larger context
- Need to accurately describe the system context
- At this stage, concentrate on the system context, not the internals of the solution
- We are trying to firm up on the boundary between the system we are trying to construct, and the outside world
- The system context is a description of that outside world, but it is an abstraction, in the sense that only parts of it of interest to our system are represented
- A well defined system content can help with communication between software developers and clients
- Each initially probably know little about the work of the other

Goals

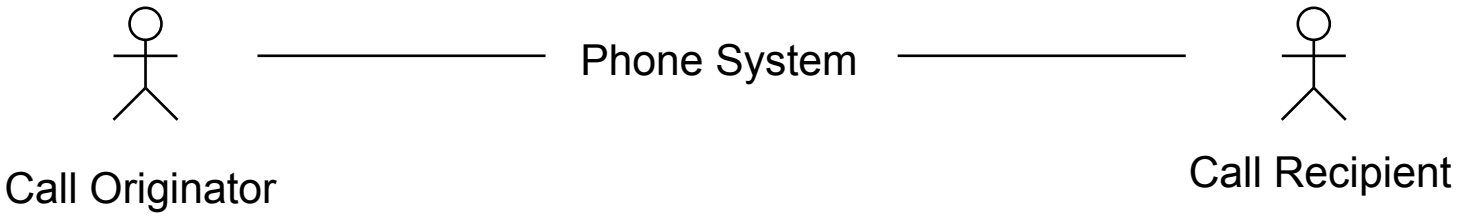
- Describes the environment in which the solution will run
- A glossary of terms should also be created, so that everyone gets the same meaning from project-specific terms
- Usually come up with lots and lots of possibilities
- Need to discover how the business operates
- (or how it should operate - a big difference!)
- Need to work on the glossary in tandem

Business Model

- A business model describes business processes comprehensively
- Business processes are how the business is used by its customers and partners
- The software system we wish to develop is part of the business, but so are the staff, equipment, premises, and other things in the business
- Customers and partners are modelled as business actors (they might or might not come into direct contact with the software system - for example, in a call centre environment, a customer rings a company representative, who in turn uses the software system)
- Business processes describe how the significant aspects of a business cooperate to provide services business actors
- Business use cases describe the services the business supplies

Sample Business Use Case

- The Make A Phone Call Business Use Case involves these steps:
  - 1. Call originator picks up phone and dials
  - 2. Call recipient answers
  - 3. Conversation ensues
  - 4. When finished, both put down the phone

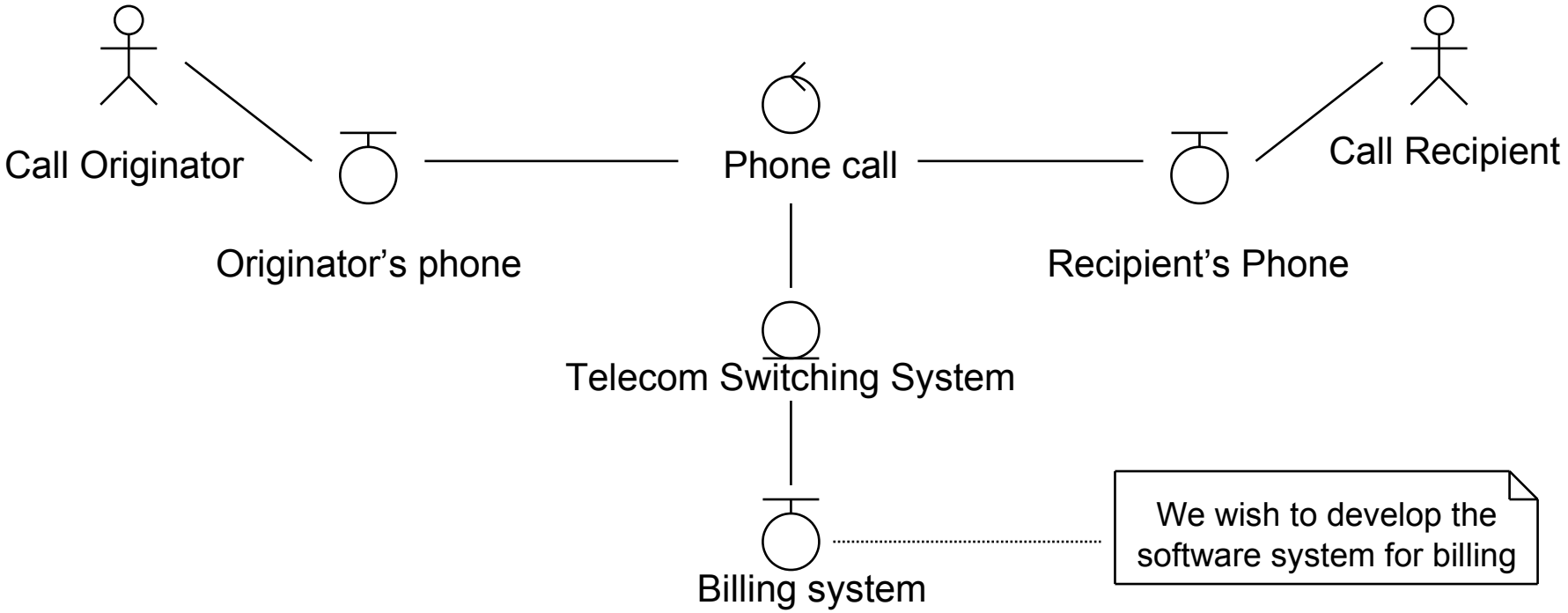


Inside Business Processes

- How a business use case is realised is described by a business object model, which can consist of interaction and / or activity diagrams
- It describes the internals of the business
- The software system we wish to develop is just one part of this
- Often one business actor talks to another who talks to another, who finally talks to our software system - for software use cases only the last actor is represented, but to gain a wider picture if often helps software engineers to see all the business actors
- Sometimes there is ambiguity of what is an actor, or the granularity of the actor - by showing all the actors this problems is solved
- End user might refer to an actor which does not appear in the software use cases, but it should appear in business use case, albeit on its own or combined or partitioned with

Sample Business Object Model

- Need to describe workers, business entities and work units for the realisation of each business use case



Features of Business Modelling

- Origin
  - Business entities originate by examining how business actors interact with the business
- Content
  - Business entities cover both static and dynamic - they have attributes and operations as we describe how things get used
- Trace relationship to use cases
  - Business modelling is a starting point for system use cases

Main Aspects of the Requirements Workflow

- Once we have a business model defining the system context, we are ready to start preparing a detailed description of requirements
- How we go about this is know as the requirements workflow
- We need to consider:
  - Artefacts (what “things” we need to create)
  - Workers (what type of people perform each aspect of the workflow)
  - Workflow (the ordering of the work)

Artefacts (1)

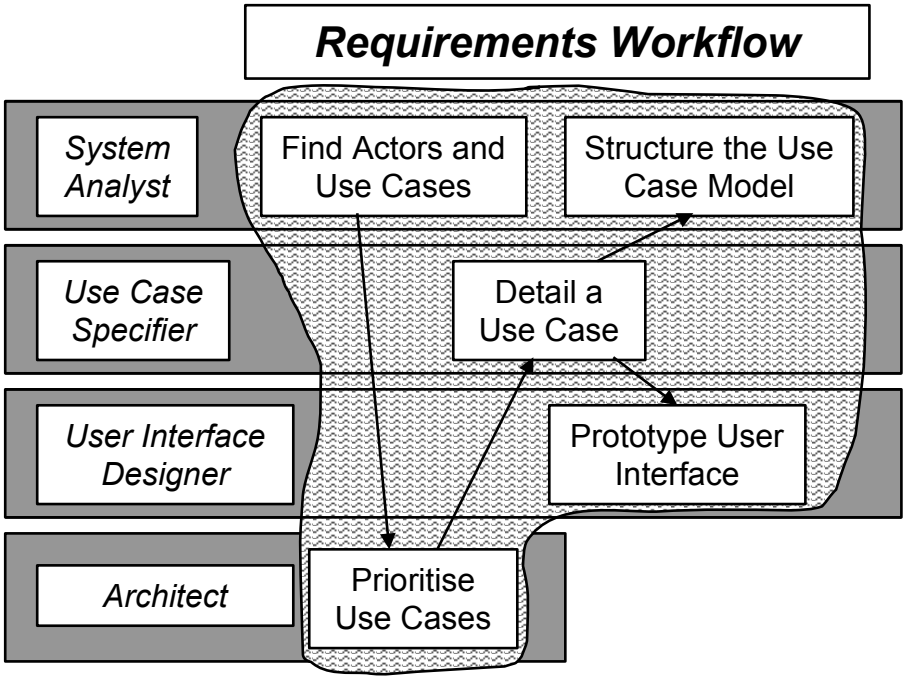
- Use Case Model
  - One or more use case diagrams, containing actors, use cases, interfaces and generalisation/includes and extends relationships
- Actor
  - Type of user with which the system directly interacts
- Use Case
  - System functionality which provides a result of value to actors
  - Sequence of actions showing how the system and actors interact

Artefacts (2)

- Glossary
  - Clear description of all terms used on the project
  - This avoids ambiguity and misinterpretation
  - Starts with terms from business model, but can be extended with software-specific terms
- User Interface Prototype
  - Rough sketch of GUI
  - Good idea to keep separate from use cases, as this will evolve differently
- Architecture Description
  - The architecture of a project usually contains the 20% significant use cases (and later: the design of their implementation)
  - The architectural significant use cases

Workers

- System Analyst
  - Responsible for the use case model, discovering actors and writing the glossary
  - Finding the system boundary
  - Ensures a complete and consistent use case model is created
  - Model leader and coordinator
- Use Case Specifier
  - Responsible for individual use cases
  - Talks to real users
- User Interface Designer
  - Prototypes the user interface
- Architect
  - Responsible for the architectural view of the requirements


Requirements Workflow
System Analyst
Find Actors and Use Cases
Structure the Use Case Model
Use Case Specifier
Detail a Use Case
User Interface Designer
Prototype User Interface
Architect
Prioritise Use Cases

Find Actors and Use Cases

- Performed by system analyst
- First step in requirements workflow
- Inputs could be business model, supplementary requirements or feature list drawn up by marketing/users etc.
- Outputs are the use case model (in an outlined state) and a glossary
  - Need to determine initial set of actors and use cases
  - This will be iterative, and subsequent passes will identify more actors and use cases, and re-arrange existing ones
  - Need to briefly describe each actor and use case
  - (Usually) need to supply some commentary on the use case model as a whole (explaining reasons for various decisions)

Prioritise Use Cases

- The architect needs to assign priority to each use case
- The top-priority 20% of use cases will become part of the architecture
- The priority of use cases will be used to decided which use cases will be completed in which iteration
- Issues to consider include:
  - How vital the use case is to the functioning of the system
  - What risk is attached to this use case
  - Sequencing of use cases (some must be completed before others - it does not make sense doing a “generate database report” use case before the “create database” use case)
  - Business issues

Detail a Use Case

- The use case specifier needs to talk to users to describe in detail a use case
- Inputs are the outline use case model, supplementary requirements and the glossary
- Outputs are detailed use cases
- The use case specifier will have to specify the pre-conditions and post-conditions of the use case, the primary scenario and alternative scenarios
- Activity diagrams or statechart diagrams could be used to depict more complicated use cases

Prototype the User Interface

- Inputs are the outline use case model, supplementary requirements, the glossary, and detailed use cases
- Output is a user interface prototype
- The user interface designer needs to examine the use cases and determine how the user needs to interact with the system
- At this stage we do not want to make detailed decisions about the UI, rather we wish to detail the services it will provide
- Later the UI designer can actually create GUI prototypes which will help users visualise the solution

Structure the Use Case Model

- After a few passes through the use case model, it becomes apparent that certain pieces of requirements should be moved around to better structure the use case model
  - Common requirements should be specified once, by adding generalisation relationships among use cases
  - Some use cases extend the requirements of other use cases, and this should be modelled with the extends relationship and extension points
  - Use cases can be included (“as is”, without any modification) in other use cases with the includes relationship
- Note that in the use case model we are capturing requirements
  - It is not where functional decomposition should occur

Moving from Requirements to Design

- The Requirements Workflow has produced a Use Case Model and other artefacts
- These describe what needs to be done from a user's perspective
- They are written in the language of the end-user's domain
- Hence may be ambiguous, non-technical, but understandable by the end-user
- Use cases may overlap
- Each fully describes functionality for a particular user
- The Design Workflow will produce a Design Model and a Deployment Model
- These describe the internals of the system in considerable detail, the interfaces and components needed and how to deploy them

Why do we need Analysis?

- The Analysis Workflow acts as a bridge between requirements and design
- It is another look at requirements, but with different goals
- Defines what needs to be done from a software engineer’s perspective
- A “technical view” of the requirements
- A complete and accurate description, with no holes
- Analysis is not repeating the requirements and not starting the design, but in reality often involves a bit of both!
- Analysis can become the first version of the design
- It is not bundled in with requirements or design because we need “a separation of concerns”

Aims

- Refining - Structuring - Arranging
- Improve our understanding of the proposed system
- Narrow its description by eliminating vagueness
- Precision is a goal
- Much higher level of detailed is needed
- Start to add structure to the evolving system
- Discuss the internals of the system
- Find commonality - with an eye to design reuse (design patterns) and code reuse (reuse code libraries)
- Evaluate how internal resources are shared
- Identify resource contention
- Handle the overlap among use cases

Use Case Model vs. Analysis Model

- The target audience for a Use Case Model consists of end-users
  - Examining from the outside the services the system provides
- The target audience for an Analysis Model consists of software engineers working on design and implementation
  - Examining from the inside the services the system provides
- A Use Case Model is a collection of use cases (a use case diagram and scenario descriptions) whereas an Analysis Model is a collection of classes (a class diagram) and possibly their interactions (sequence and collaboration diagrams)
- These classes are prototypical
- Overview of how system could be realised
- A use case model can become part of the legal contract

What about OO and Reusability?

- Some people complain that use cases are not “object-oriented” but they miss the point that looking at the system from an OO-perspective begins with analysis
- Some people also complain that use cases do not encourage reuse - again the answer it that reuse starts with analysis, and here one should keep an eye out for repeating functionality and possibilities for reuse
- Use cases are the external view, and the main goal is to clearly agree with the end-user what needs to be done (discussion of “objects” and “reusability” at this point does not make sense)

Shape & Structure

- Use Case Models often consist of seemingly unrelated requirements with little cohesion
- How do they interact with each other?
- What do they share?
- A major problem software designers/developers have when presented with a bunch of use cases is:
- “Well, how do I start to create a system for all that!”
- The Analysis Model re-arranges them and finds the common strands among the use cases
- It provides the initial shape of the system
- “Getting started is half the work”

Shape & Structure (2)

- Sometimes things don’t fit (or don’t make sense)
- This will be detected (and hopefully resolved in conjunction with end-users) early during analysis (when it is cheap) rather than later during design or worse implementation (when it is expensive)
- Late problem-solving involves interacting with lots more people

Maintenance

- Easier to maintain well structured requirements
- Our goal is to be resilient to changes in requirements
- Maintainable

Lifetime of Analysis Model

- A use case model must be fully maintained over the lifetime of a project
- The project manager needs to make a decision about whether the Analysis Model should be continuously updated or not
- It might be temporary:
  - Because once the Design Model is reasonable developed it supersedes the Analysis Model
  - The system structure originally defined in the Analysis Model could be changed in the Design Model
  - During design, in addition to requirements need to consider implementation details such as operating systems and networking
  - The real architecture is discovered during design - in analysis we merely suggest!

Do we need analysis?

- Sometimes people do not bother with analysis, or merge it (whether in name or in deed) into design
- 5:1 Ratio
- We can cover much more grounds with analysis compared to design (helicopter view)
- Useful to gain understanding, to decide how to sub-divide project etc.
- Analysis provides a technical overview of the project, and so for someone new to the project it is a good starting point (architectural overview)
- Multiple implementations (with different designs) all based on the same analysis
- Legacy systems - what they do (without going into the un-important detail)

Standard Class Stereotypes

- Entity
  - An entity class models data which exists for a long period (e.g. in collections) and may even be persisted in a database
  - Non-transient pieces of information
- Boundary
  - Points of interaction between system and external world
  - Conceptual interfaces
- Control
  - Co-ordination, sequencing and transactions
  - Where active work is completed
- How are these used during analysis?

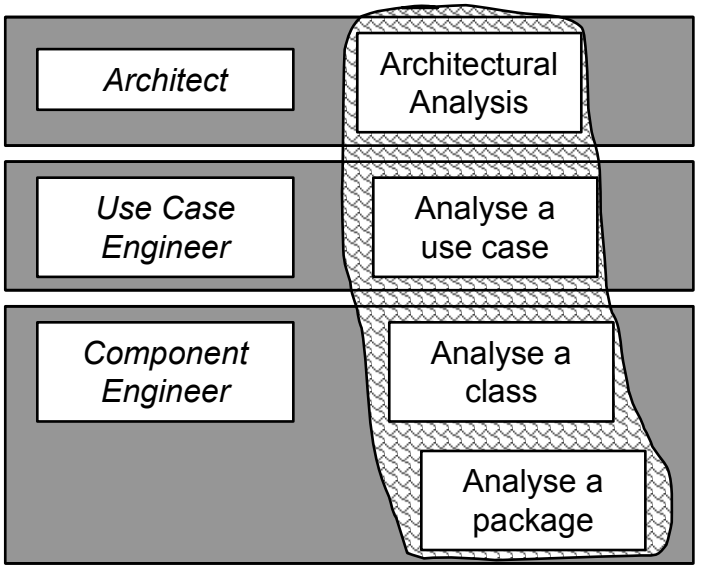
Artefacts

- Analysis Model
- Analysis Class
- Use Case Realisation - Analysis
- Analysis Package
- Architecture Description
    - (View of the Analysis Model)

Workers

- Architect
- Use Case Engineer
- Component Engineer

Workflow


Architect
Use Case Engineer
Component Engineer
Architectural Analysis
Analyse a use case
Analyse a class
Analyse a package