Development Methodologies

A methodology is a system of methods and principles used in a particular sub-discipline of software design.

There are a large number of these, reflecting the way in which software design in practice has specialised. Those which are mature usually are supported by specialist tools and techniques.

We discuss two:

- The Unified Process
- Extreme Programming

The Unified Process

A traditional style of incremental design driven by constructing views of a system architecture.

- Component based.
- Uses UML for all for all blueprints.
- Use-case driven.
- Architecture centric.
- Iterative and incremental.

Details in Jacobson, Booch & Rumbaugh *et.al.* 1998, *The Unified Software Development Process.*

Phases of Design Cycles

Design proceeds through a series of cycles, each of which has phases:

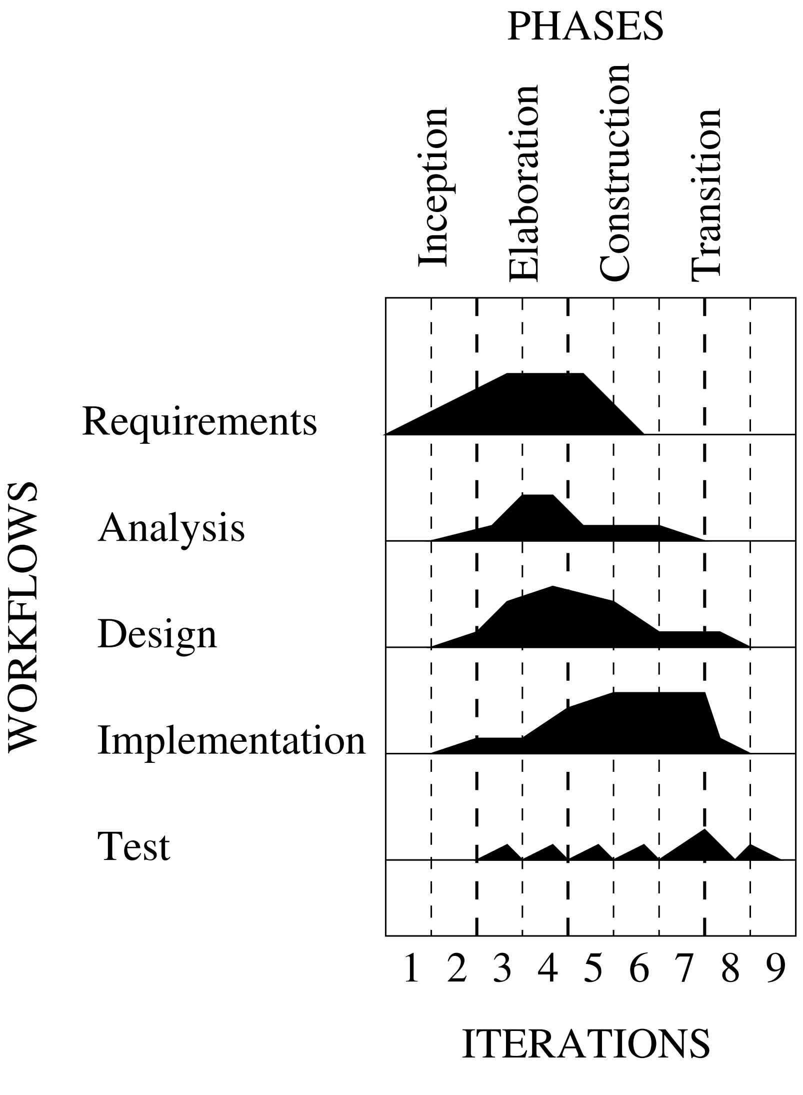
Inception : Produces commitment to go ahead (business case feasibility and scope known).

Elaboration : Produces basic architecture; plan of construction; significant risks identified; major risks addressed.

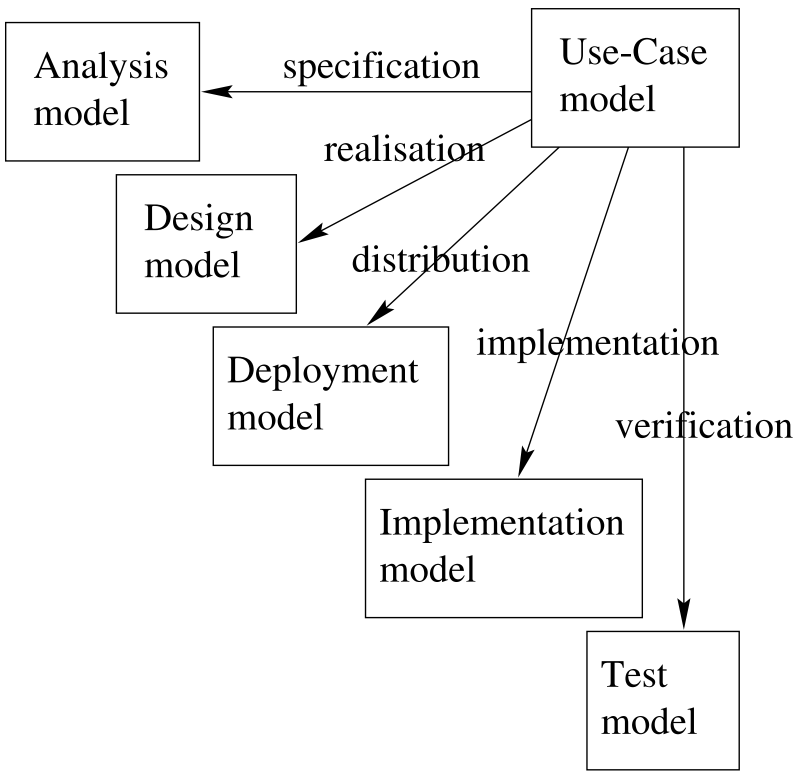
Construction : Produces beta-release system.

Transition : Introduces system to users.

Waterfall Iterations Within Phases



The Product: A Series of Models



Use Cases

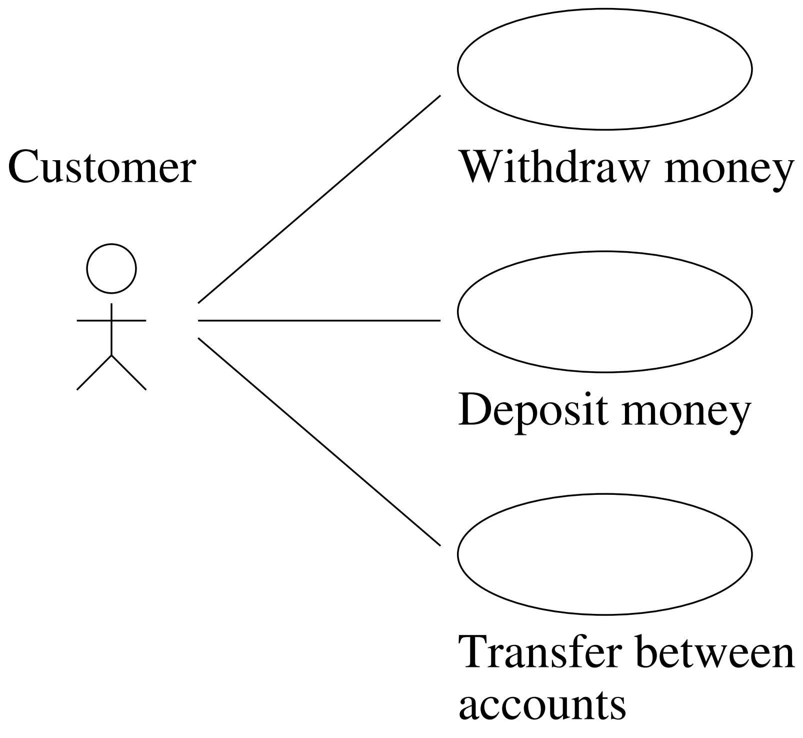
“A use case specifies a sequence of actions, including variants, that the system can perform and that yields an observable result of value to a particular actor.”

These drive:

- Requirements capture.
- Analysis and design of how system realises use cases..
- Acceptance/system testing.
- Planning of development tasks.
- Traceability of design decisions back to use cases.

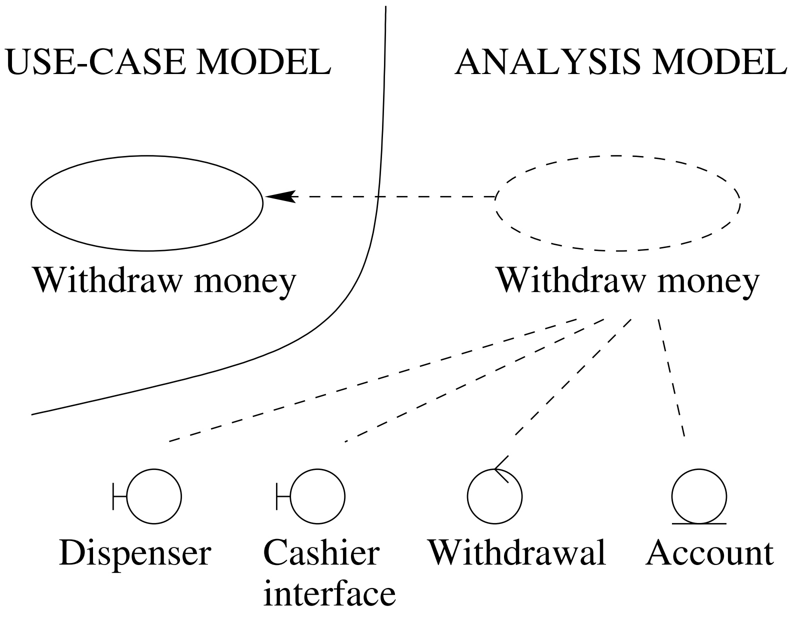
Use Case Example: 1

Initial use-case diagram.



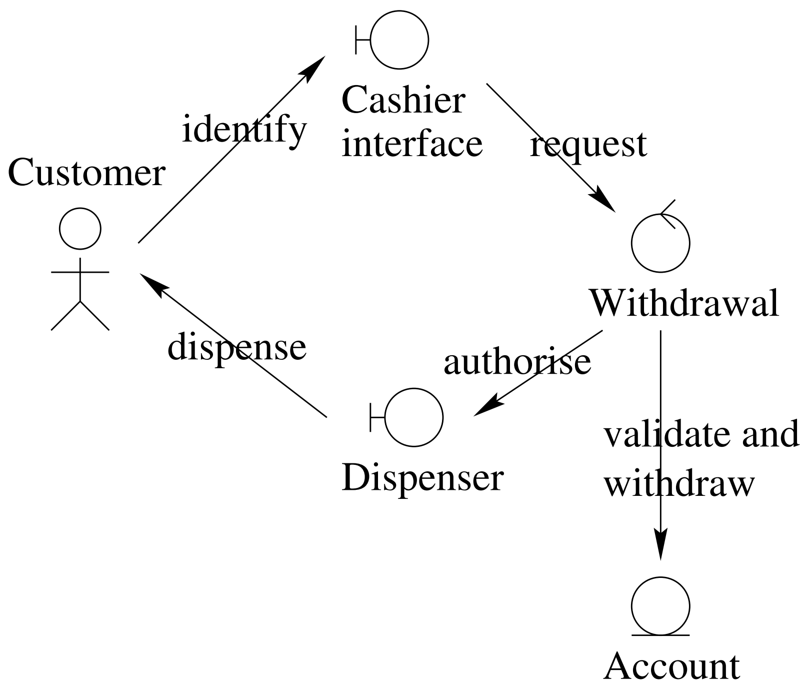
Use Case Example: 2

Analysis classes for withdrawing money



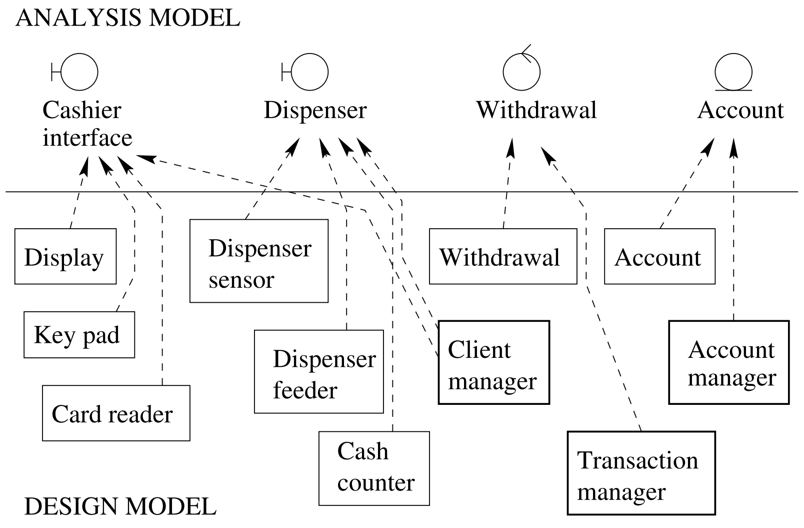
Use Case Example: 3

Collaboration diagram for withdrawing money.



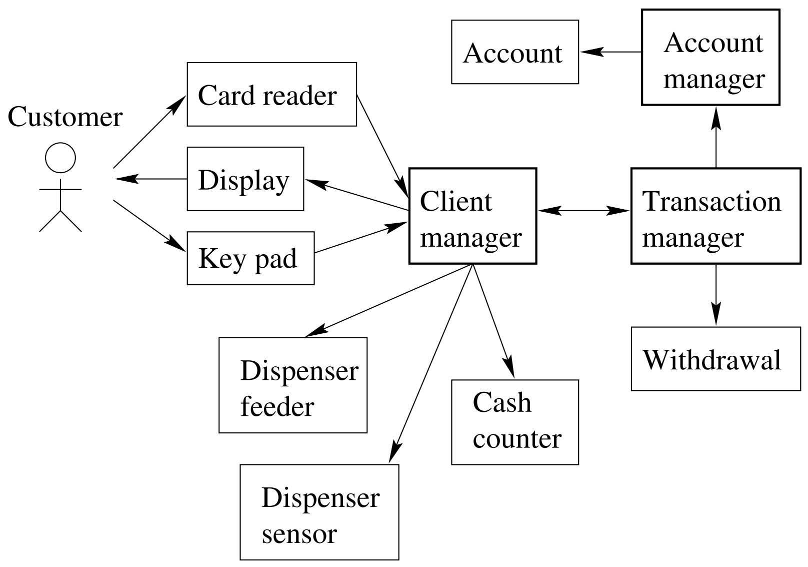
Use Case Example: 4

Design classes introduced for analysis classes.



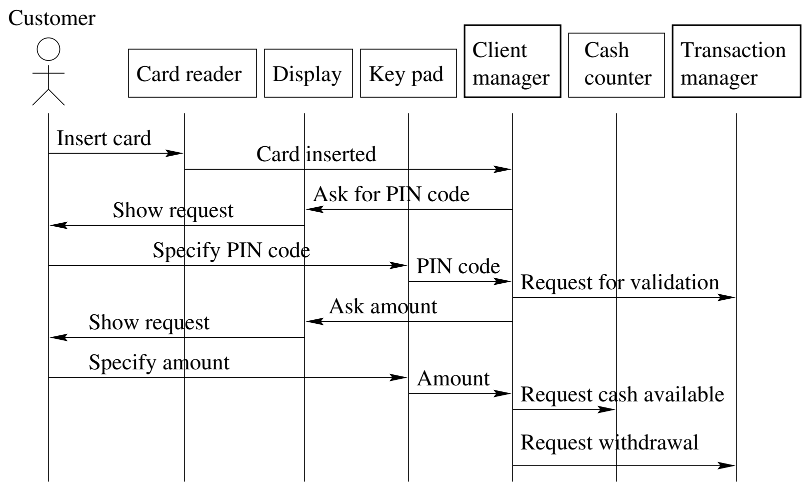
Use Case Example: 5

Class diagram which is part of the realisation of the design model.



Use Case Example: 6

Sequence diagram for part of the realisation.



Extreme Programming (XP)

Traditional "heavyweight" methodologies (*e.g.* the Unified Process) concentrate on carefully controlled, up-front, documented thinking.

Assumption : Cost of unravelling decisions made in early stages rises (exponentially) as we go through later stages.

Benefit : Global control throughout minimises risk of unravelling.

XP is more "lightweight" and concentrates on the the dynamics of closely knit, fast moving design/coding teams.

Assumption : Reaction to change can be made constant through lifecycle.

Benefit : Design can be more flexible - in particular we may re-visit early decisions more easily.

See Kent Beck, 1999, *Extreme Programming Explained.*

XP is Controversial

An IBM Java poll on XP (currently cited at `www.xprogramming.com`) said roughly this:

- "I've tried it and loved it" (51%)
- "I've tried it and hated it" (8%)
- "It's a good idea but it could never work" (25%)
- "It's a bad idea - it could never work" (16%)

How XP Imposes Control

Through twelve “practices” to which designers adhere (using whatever other compatible methods and tools they prefer).

Not strongly influenced by a particular design paradigm (like the Unified Process).

Does require a strongly held view of how to approach design.

We consider some key practices in the following slides.

The Planning Process

The “customer” defines the business value of desired features.

The programmers provide cost estimates for producing them in appropriate combinations.

Not allowed to speculate about producing a total system which costs less than the sum of its parts.

Small Releases

Put a simple system into production early.

Re-release it as frequently as possible while adding significant business value on each release (*e.g.* Aim for monthly rather than annual release cycles).

The aim is to get feedback as soon as possible.

Simple Design

Do the simplest thing that could possibly work.

Don’t design for tomorrow - you might not need it.

Testing

Focus on validation at all times.

Write the tests before writing the software.

Customers provide acceptance tests.

All within a rapid design cycle.

Refactoring

XP dives straight into coding, so re-design is vital.

“Three strikes and you refactor” principle - *e.g.* consider removing code duplication if:

- The 1st time you need the code you write it.
- The 2nd time, you reluctantly duplicate it.
- The 3rd time, you refactor and share the resulting code.

This needs a system of permissions for change between teams.

Pair Programming

All code is written by **a pair** of people at one machine.

- One partner is doing the coding.
- The other is considering strategy (Is the approach going to work? What other test cases might we need? Could we simplify the problem so we don't have to do this? *etc*).

This is unpalatable to some but appears vital to the XP method.

Collective Ownership

Put a good configuration management tool in place.

Then anyone is allowed to change anyone else's code modules, without permission, if he or she believes that this would improve the overall system.

Continuous Integration

Integration and testing happens no more than a day after code is written.

This means that individual teams don't accumulate a library of possibly relevant but obscure code.

40-Hour week

XP is intense so it is necessary to prevent "burnout".

Designers are discouraged from working more than 40 hours per week.

If it is essential to work harder in one week then the following week should drop back to normal (or less).

On-site customer

Someone who is knowledgeable about the business value of the system sits with the design team.

This means there is always someone on hand to clarify the business purpose; help write realistic tests; and make small scale priority decision.

Coding Standard

Since XP requires collective ownership (anyone can adapt anyone else's code) the conventions for writing code must be uniform across the project.

This requires a single coding standard to which everyone adheres.