Software Development

Understanding How Software Is Built and Maintained

Sample Content

What is Software Development?

A successful software organisation is one that consistently deploys quality software that meets the needs of its users. An organisation that can develop such software in a timely and predictable fashion, with an efficient and effective use of resources, both human and material, is one that has a sustainable business.

What is Software Development?

- Software development is a process:
  - The input is users requirements
  - The output is executable code
  - The process must deliver the functionality needed by clients
    - Not always the same as what clients ask for…
- There are many kinds of software:
  - The gaming industry
  - Embedded and realtime
  - Business applications
    - Developed for internal use only
    - To be sold externally

What is Software Development?

- Software combines the worst aspects of:
  - Engineering
    - Software must run properly and perform adequately
  - Literature
    - We must address the users needs and desires
    - Production depends entirely on the suitability, skills and motivation of the development team
- Software is ultimately about people
  - Clients, developers and managers
  - Project management is notoriously hard
    - Circumstances tend to encourage false optimism…

Misconceptions

- Software development is not about maths
  - Profiling your code has not been important to mainstream developers since the early 1970's
  - There is a strong distinction between
    - Academic Computing
    - Scientific Computing
    - Business Computing
- Software development is not difficult
  - Once you obtain a core set of skills
  - There are natural software developers…

The Software Industry

- Initially software could be developed slowly
  - Computers were highly expensive
  - All development was bespoke
  - The time and cost was relatively inexpensive
- Development was very gradual and careful
  - Because compiling code took so long
  - Low level code had to be written on each project

The Software Industry

- Software was a victim of its own success
  - Demand increased exponentially
  - Hardware prices fell rapidly
  - Software development took an ever increasing percentage of costs and time
- PC's and the Internet accelerated the process
  - Operating Systems provided basic services
  - We are now all developing in 'Internet Time'

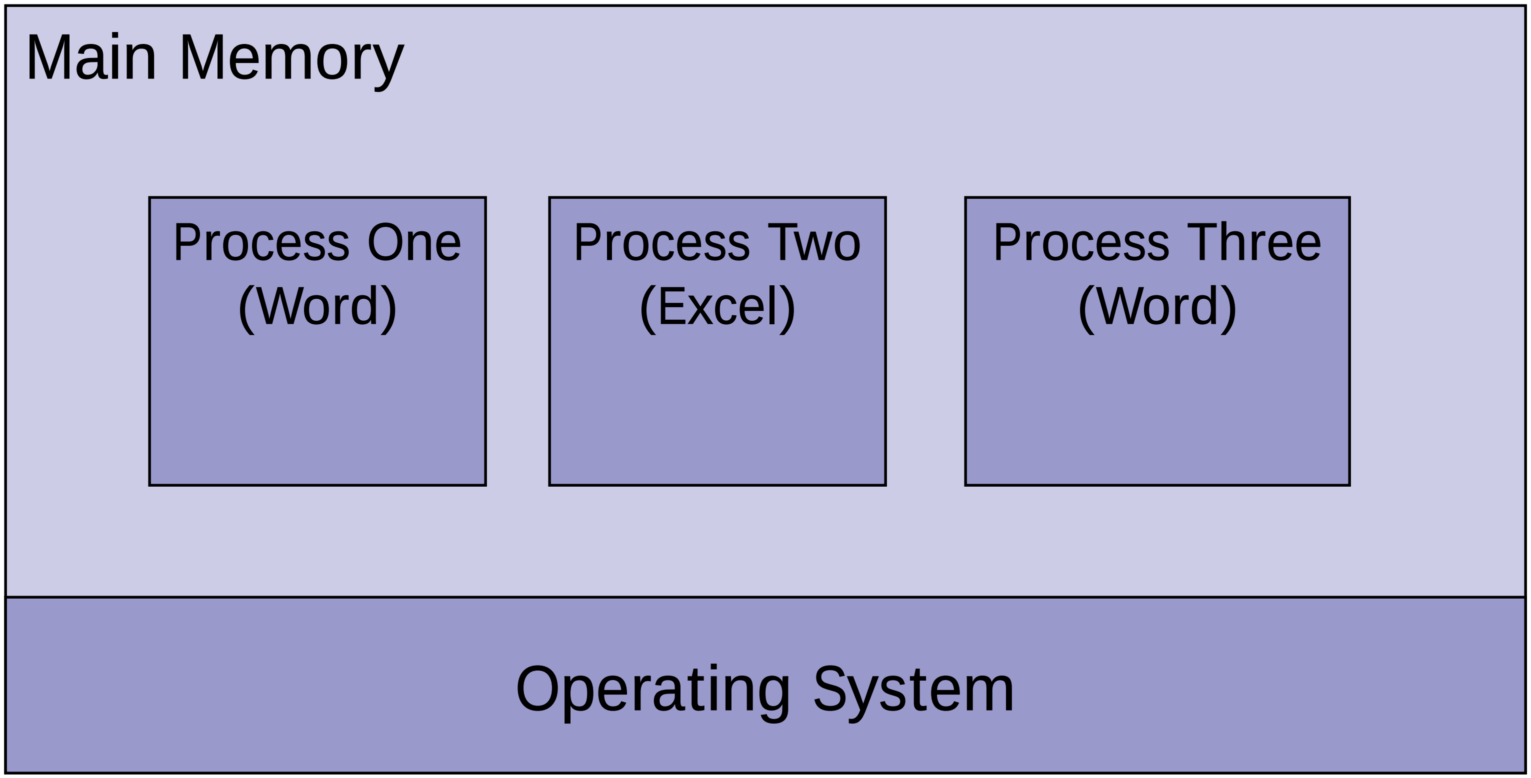
The Operating System

- Originally computers ran a single program
  - With the code on punch cards
- The program was responsible for everything
  - Causing huge redundancy
  - Only 10% of the code was unique
- Operating Systems were created to:
  - Allow more than one program to be run
  - Provide a core API for common tasks

The Operating System

- The OS coordinates all activity
  - Every program is allocated a process
    - The resources it needs to run
  - Multiple copies of a program may be running
    - Each will be allocated its own process
  - A process uses the OS to perform common tasks
    - These are what we refer to as ‘system calls’
- It is not necessary that an OS offer a GUI
  - Windows and Apple do but UNIX does not

The Operating System



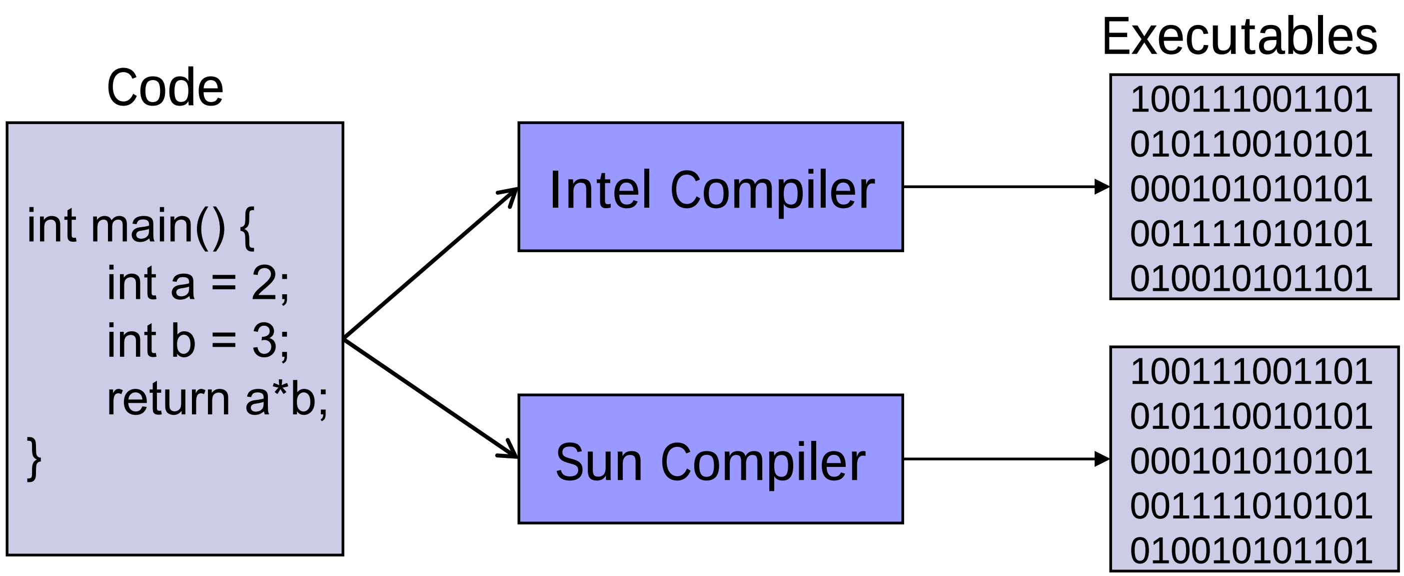
The Operating System

- Only one program can run at a time
  - The OS allows each process a slice of time on the processor
  - This happens so quickly that normally you don't notice it…
- Processes can communicate
  - A single piece of software can be made up of multiple processes
- Code for one OS will not work on another
  - Even if both OS's run on the same hardware
    - E.g. Windows and Linux on an Intel CPU

Writing Software

- Instructions are written in a programming language
  - These are just typed into a text file
  - Many programming languages exist
- The instructions must be converted
  - Into those supported by the target CPU
  - This instruction set of the CPU will be very limited
- The conversion process is carried out by a compiler
  - A compiler and code editor are a developers basic tools

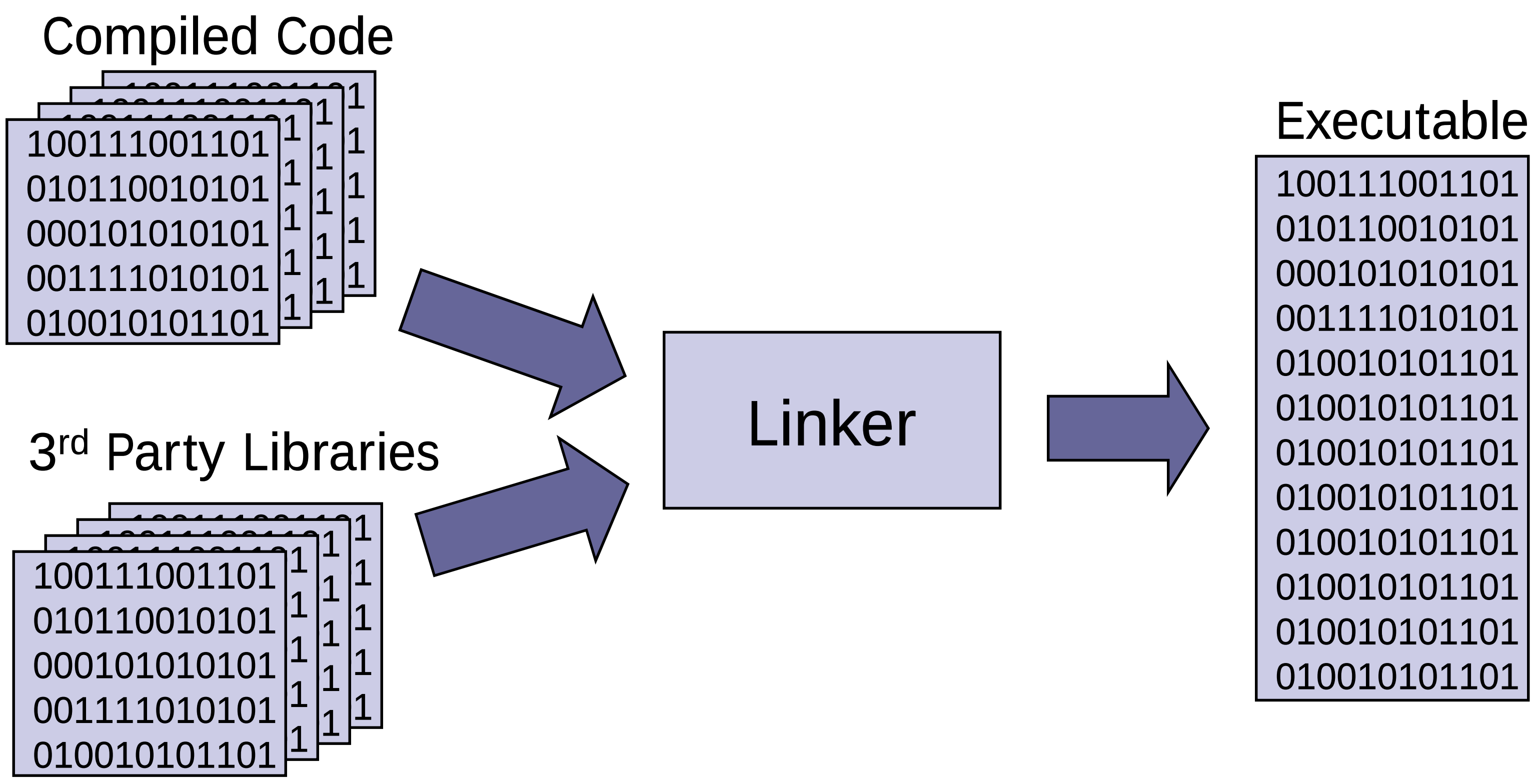
Writing Software



Writing Software

- Usually all the code does not live in one file
  - Many files are complied and then linked together into an executable
    - The linker is a separate tool
- Not all the code must be written from scratch
  - Pre compiled libraries will be supplied by:
    - Other teams
    - Third party vendors
    - The OS itself

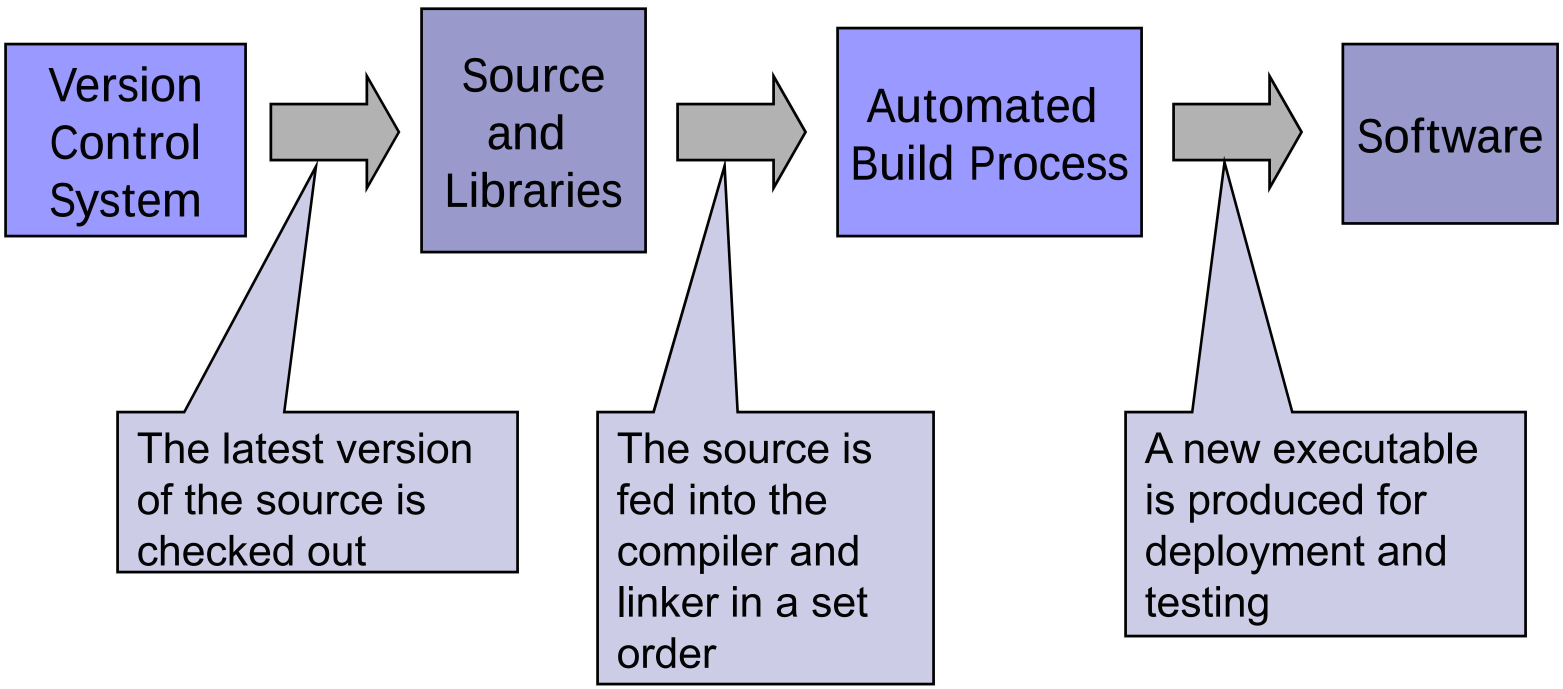
Writing Software



Building Software

- The source code and libraries are stored inside a version control system
  - Such as MS SourceSafe or Rational ClearCase
- When changes are made the affected parts of the system are rebuilt
  - This is a highly automated process
- Builds may be made daily, weekly or monthly
  - Some builds will be for release to the client
  - Others will be for internal use only

Building Software



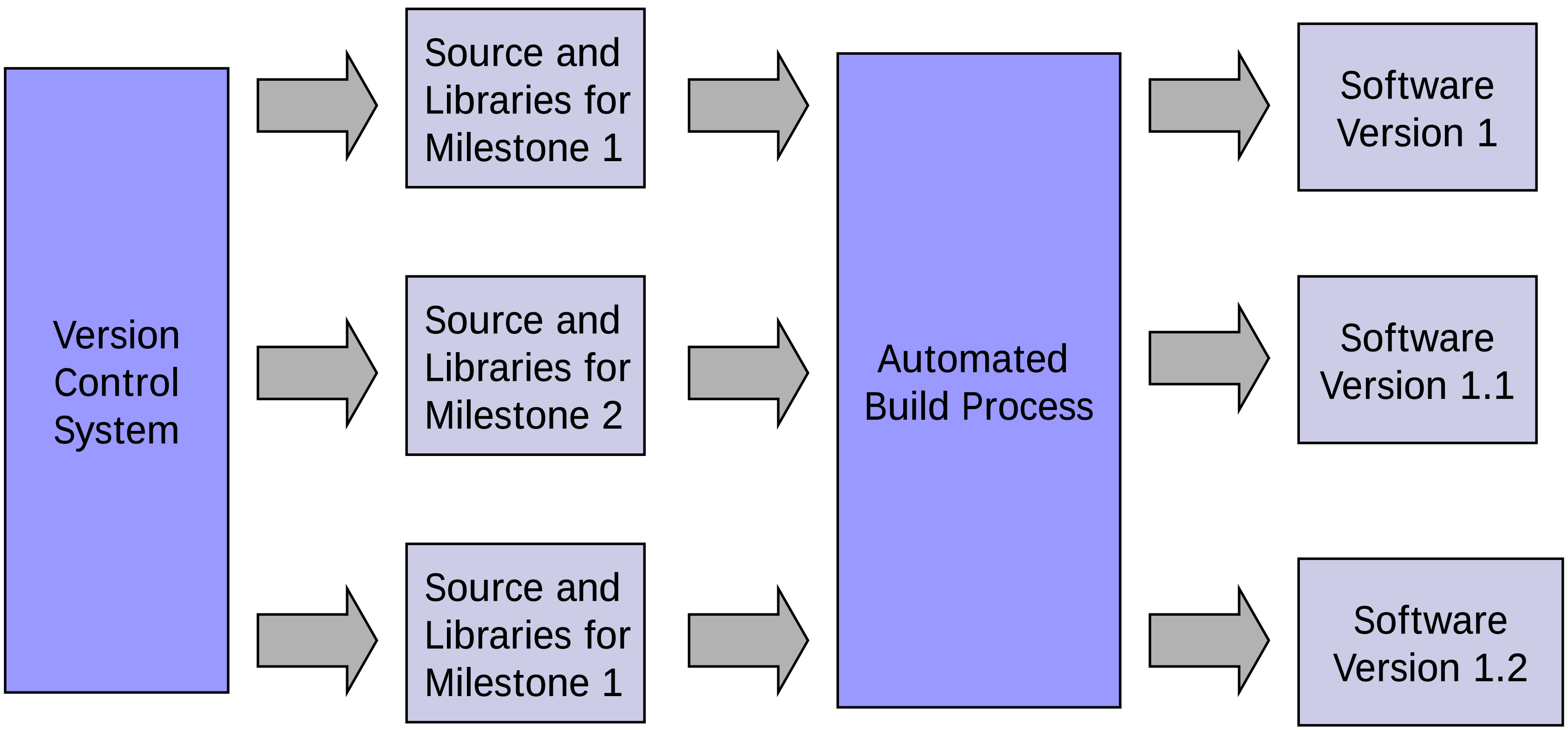
Building Software

- Each developer will have their own build
  - In order to develop and test new code
- Source code files will be changed frequently
  - Developers check these in and out of the VCS
  - The VCS records all the changes made
- Periodically milestones are reached
  - The current version of each source file is used to create a build of the product
  - This is then rigorously tested

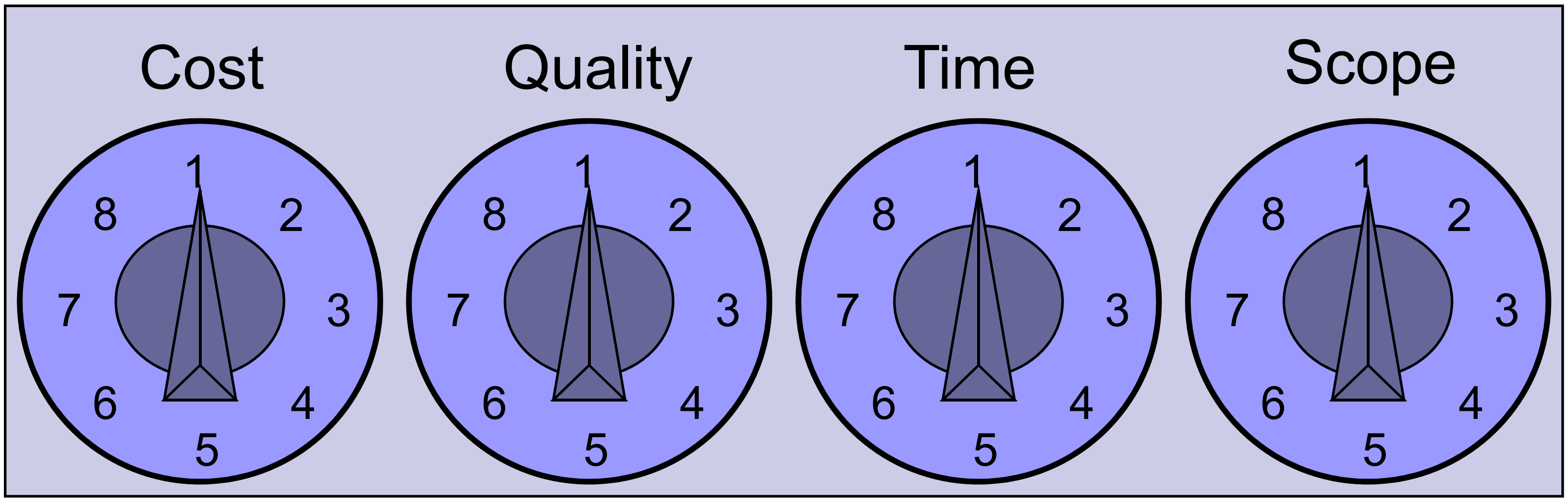
Building Software

- Efficient software development requires:
  - A properly used version control system
  - An automated build environment
  - A fast and thorough test harness
- We must be able to return to a previous build
  - A problem may appear in some builds only
  - Bug reports must be investigated against the build they appeared in, e.g. Version 3.1 SP3

Building Software



Managing Software



Managing Software

- Project management is about four variables
  - These are cost, quality, time and scope
  - Plus the psychological consequences of each
- Each variable has a complex relationship with the others
  - Changing one has a delayed effect on the others
  - It is hard to predict the time and magnitude of the effect
- You can only ever fix three out of four
  - For example if you try to fix quality, time and scope then cost will go though the roof…

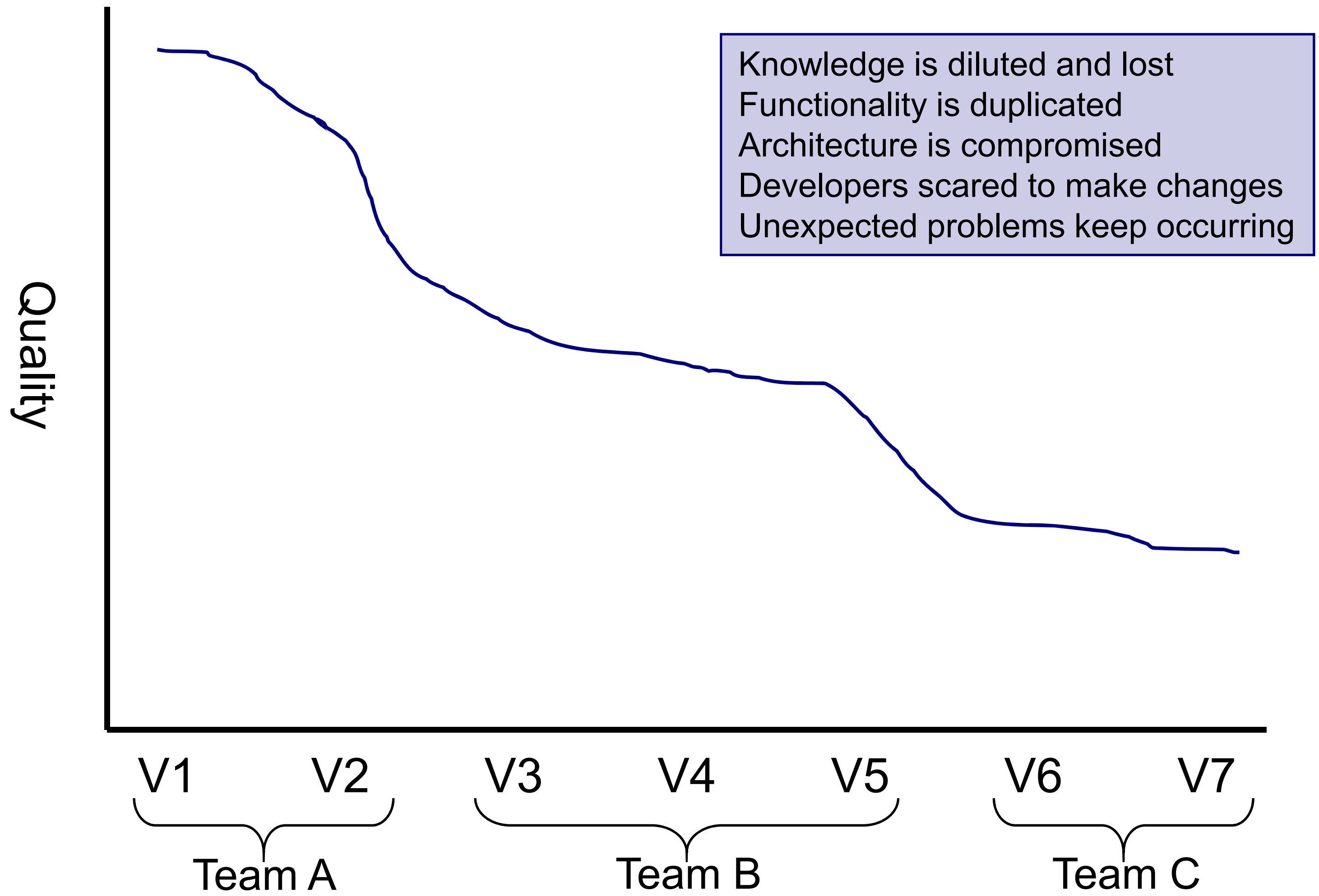
Managing Software: Cost

- Increasing cost can involve
  - Buying faster machines with better monitors
  - Providing dedicated machines and networks for testing
  - Purchasing expensive development tools for build management, automated testing, code generation etc…
  - Adding new people to the project
  - Allocating money for overtime
- Extra cost isn't a panacea
  - Adding people slows progress till they can be trained
  - Expensive tools are typically underused
  - Endless overtime kills creativity
  - Adding hardware has diminishing returns

Managing Software: Quality

- Internal quality is visible only to developers
  - Simple, efficient and well-documented designs and code
- External quality is what is visible to the user
  - This includes ‘non-techie’ issues like GUI design
  - Internal quality can temporarily be reduced without any visible effect on external quality
- Sacrificing quality is tempting but fatal in the long term
  - New developers cannot understand the code
  - Features cannot be added without causing bugs
  - Fixing reported bugs consumes most of the coding time
  - Eventually it is simpler to rewrite code than maintain it

Software Over Time



Managing Software: Scope

- Reducing scope is the best cure for a sick project
  - Unfortunately it cannot always be applied
- Projects are almost always overly ambitious
  - Writing software is ‘only’ a creative activity
  - Developers can be pathologically optimistic
  - Promises have to be made to win contracts
  - The limits of current technology aren't understood
- Features should always be prioritised to reduce scope
  - What core functionality is essential to the customer?
  - What can be pushed out into another release?
  - What features are of limited worth to the customer yet are producing major technical problems?

Managing Software: Time

- Increasing the available time ‘cools down’ a project in danger of imminent meltdown
  - Teams focussed on cramming in functionality for immediate release loose their long term perspective and overall goals
  - Continuously releasing prototypes to the client is dangerous if the client is driving the schedule
- Realistic timescales don’t solve everything
  - Some technical problems cannot be solved by extra time alone...
    - E.g. requirements that are not within the teams skill-set
  - Many projects reach 80% completeness and then stay there…

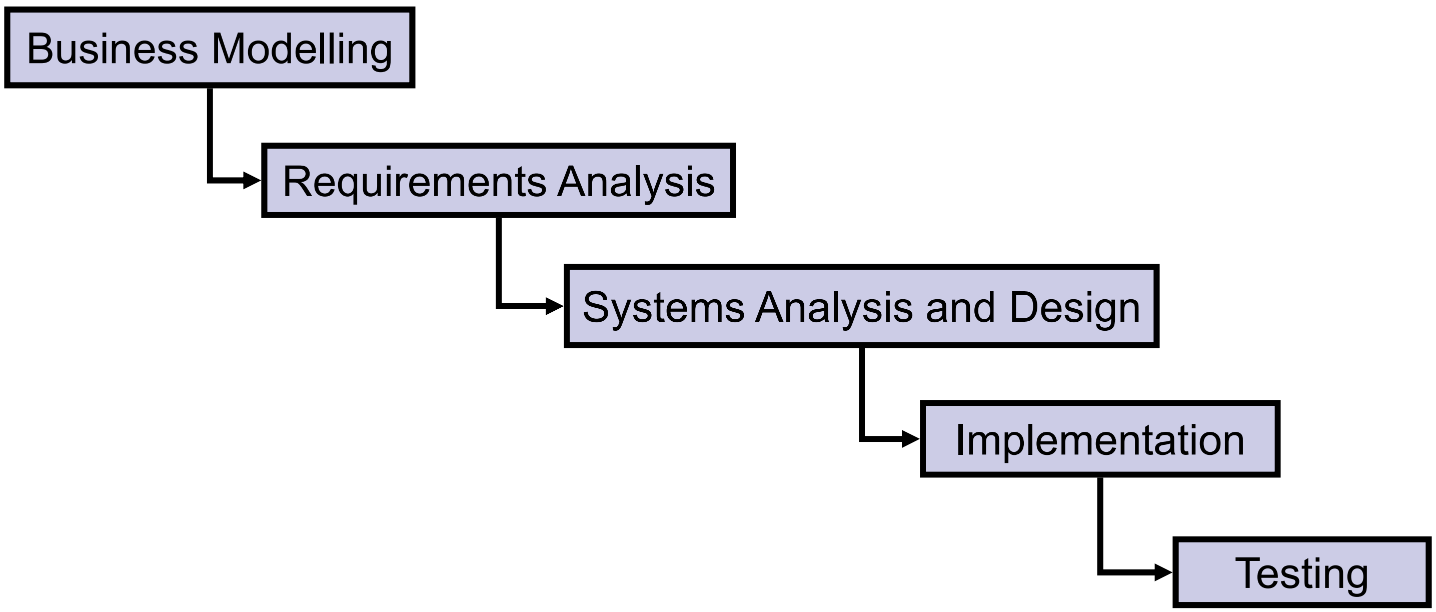
Managing Software: Summary

- A project will stand every chance of success if:
  - The scope is set at achievable goals
  - The timescale is planned realistically
  - HR, hardware and software is properly allocated
  - Maintaining and verifying quality is kept a priority
- Most projects are some distance from the ideal
  - One or more of the dials is always set too high
  - Adding time and decreasing scope is the best long term solution
  - Adding cost while decreasing quality is a tempting short term fix

Software Development & Process

- There are many formal methods for developing software
  - Very few companies use them as originally intended
- The oldest is waterfall development
  - Which simply lines up the different activities in logical order
    - Requirements → Analysis → Design → Coding → Testing
- Waterfall development is fatally flawed
  - Unless your team only develops one type of system
  - Only in step ‘B’ do you discover mistakes made during step ‘A’
- Modern methodologies stress iterative development
  - Many short waterfalls rather than a single big one
  - Each mini-waterfall both implements new functionality and fixes the problems which were identified during the last one

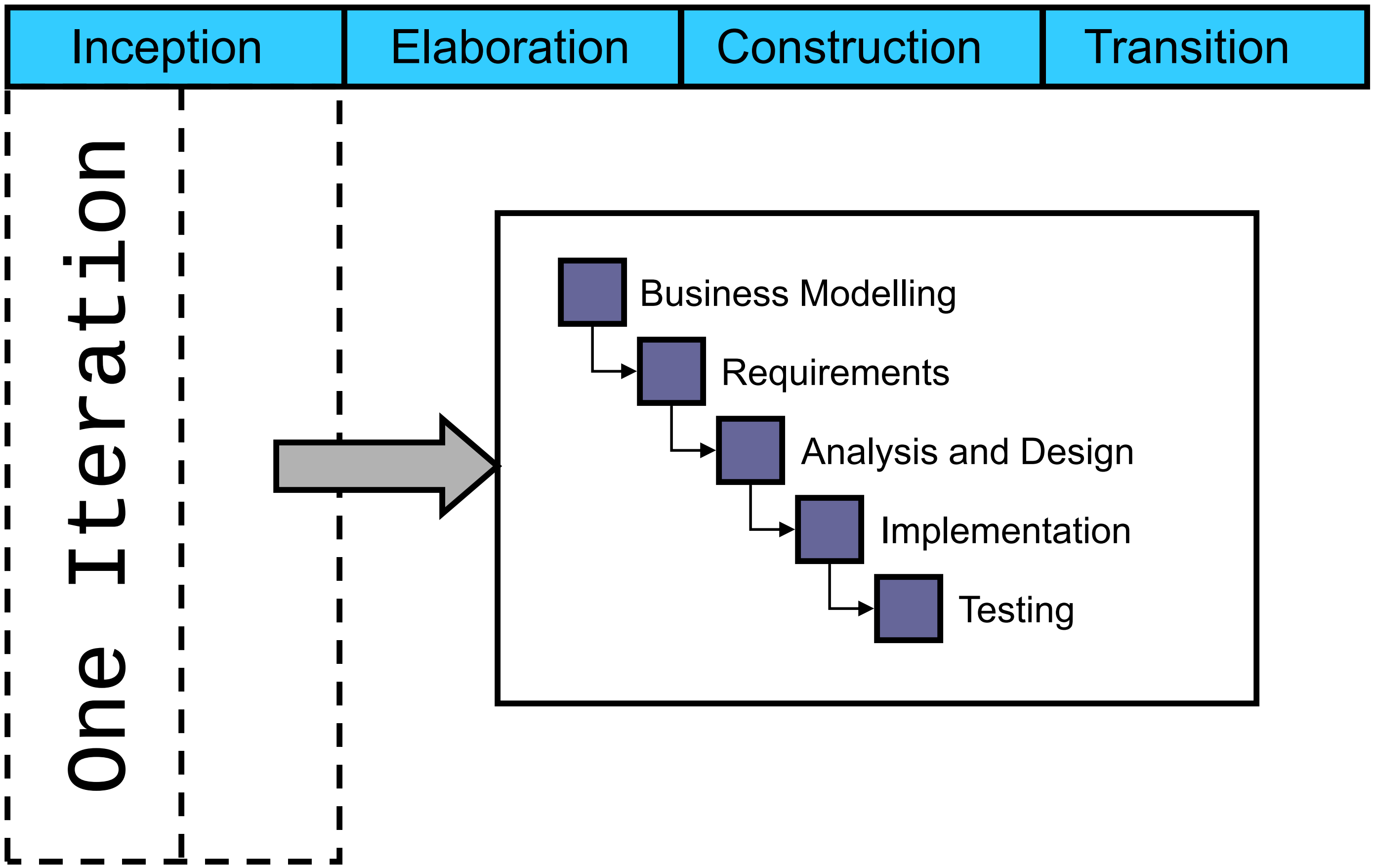
Waterfall Development



Iterative Development

- Keep releases as frequent as possible
  - Do 'mini-runs' of analysis, design, coding and testing
  - Add functionality in small cycles
    - 2/3 weeks usually works best
    - Make sure each cycle delivers completed functionality that is verifiable by the client
  - Don't be concerned with web/gui design
    - Work from basic simple screens
    - Layer on the style once the functionality is there

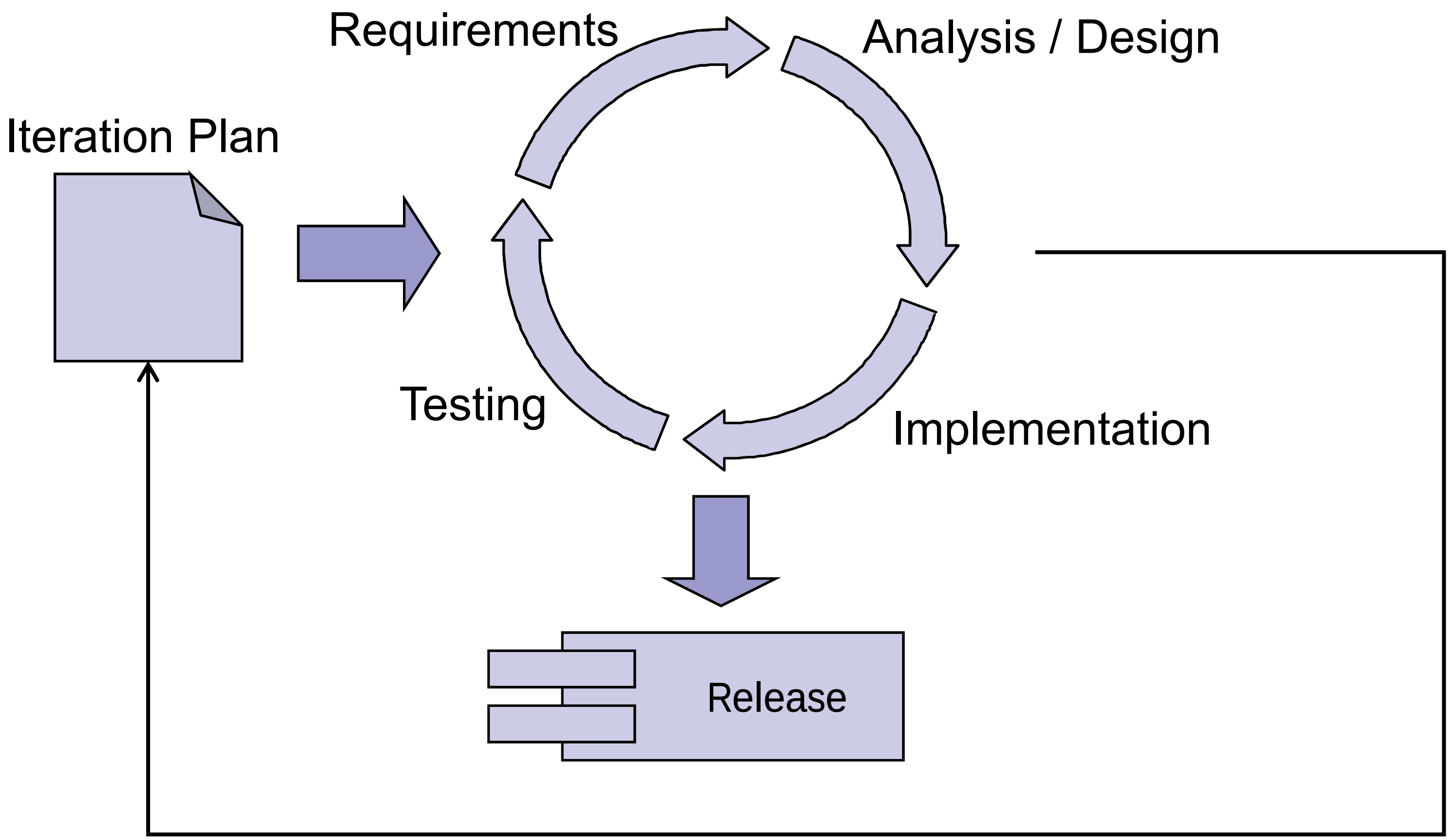
Iterative Development (RUP)



Iterative Development

- Prioritise use cases according to:
  - Those that have value to the development team
  - Those that have value to the customer
- Developers should prioritise according to
  - Verticality (functional coverage)
  - Knowledge of the problem domain
- Customers prioritise according to
  - What will earn them money
  - The simplest functionality that can be deployed in the production environment

Iterative Development



Iterative Planning

- All planning is iterative
  - Full up front planning is counterproductive
    - Too much ‘guestimation’ is involved
  - Each iteration is planned around
    - The use cases to be implemented
    - The best estimates available for development time
    - A comprehensive test plan to ensure quality
- New plans are written during each iteration
  - These need to be revised based on
    - Development time actually required on the last iteration
    - Any functionality that failed tests or was left over

Iterative Planning

- Negotiation and re-prioritisation are natural consequences of iterative development
  - We continually revise what we have achieved and adjust the iteration plan
  - We increase or reduce the scope of the current version of the project based on feedback
- Iterations prevents nasty surprises
  - Hidden problems with the technology
  - Building a system the customer doesn't want

Use Case Estimation

- You cannot replace ‘yesterdays weather’
  - Sensible estimates can only be given when you have:
    - A skeleton architecture
    - Key vertical and business use cases
- One interim measure is ideal days
  - ‘Pure’ development days without meetings, logistical problems, etc
  - Developers naturally think in terms of ideal days

Project Estimates

- You must allow time for
  - Reworking the design and code
  - Studying and responding to customer feedback
  - Changing requirements
  - Functional and load testing
  - Web design and usability testing
  - Logistical and network problems
  - Deployment issues

Modeling Software

- Software spends 10% of its life in development
  - The rest is spent being used, maintained and upgraded (often by different teams of developers)
  - The quality of software inevitably degenerates over time
- Lucky developers work on green field projects
  - They are ‘plank holders’ in new systems
  - A great opportunity for personal growth
    - But also a scary amount of responsibility
- Unlucky ones work with an existing code base
  - Similar to doing someone else's laundry
  - This is the majority of developers in the industry

Modeling Software

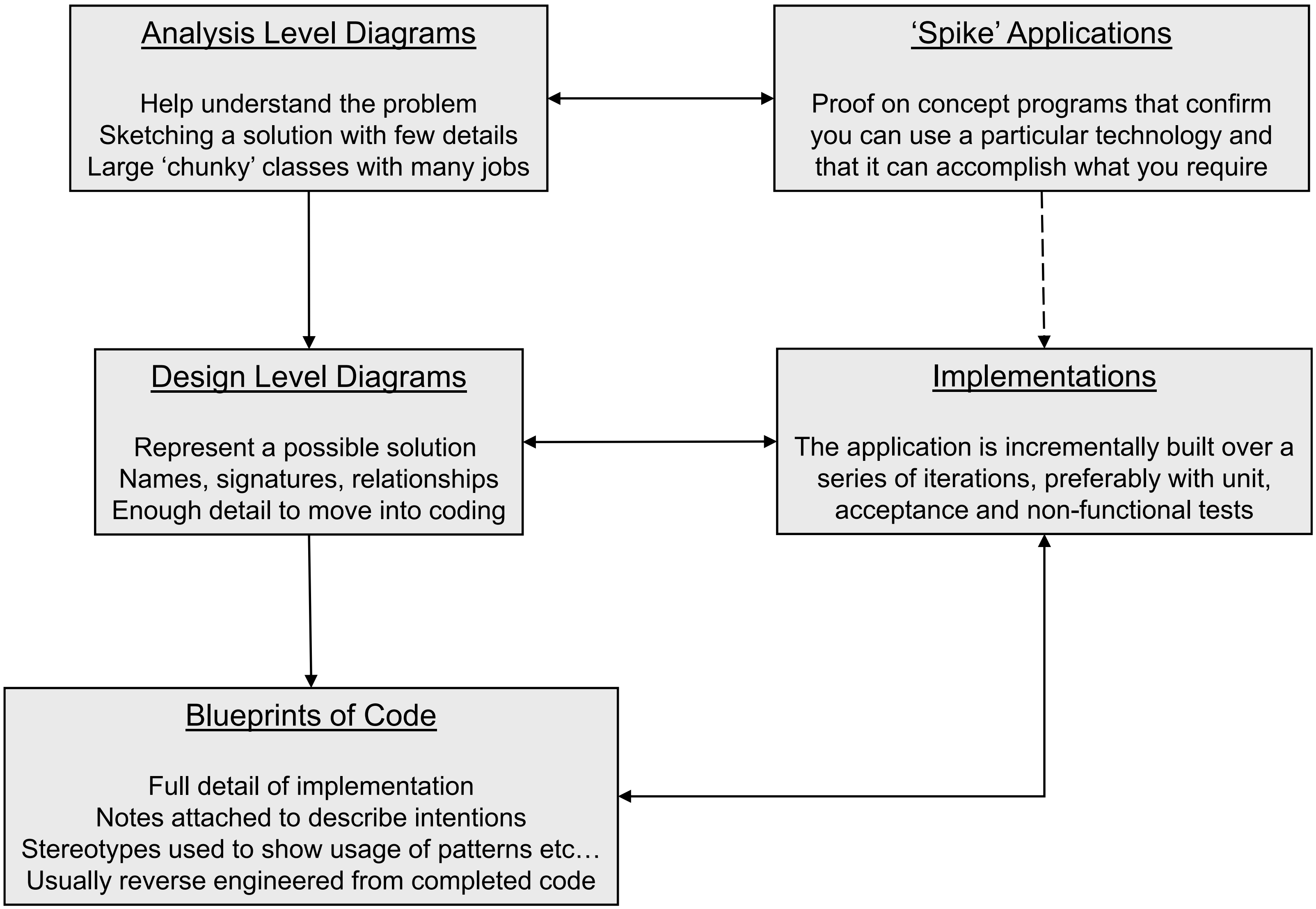
- Divining the intent of foreign code is very hard
  - Although you can make a good living out of it
  - The '10 foot stick' approach
- The best sources of information are
  - Requirements Documents
  - Suites of Unit Tests
  - Design Diagrams
- There is one current standard for modelling
  - Drawing diagrams to describe software
  - The Unified Modeling Language (UML)
    - Created by merging several earlier standards

The UML Diagrams

Diagram	Description
Use Case	Provides an overview of requirements
Activity	Details a flow of events (usually requirements)
Class	Defines a set of classes and relationships between them
Sequence	Illustrates how messages pass between objects
Collaboration	Same as above but from a different perspective
Object	Shows the values within a set of objects
State	Details the lifecycle of an object
Component	Defines a set of components
Deployment	Describes how components are placed in nodes

Details Shown On UML Diagrams

- In real life diagrams start vague and become precise
  - The first draft captures the essentials and is low on detail
  - Subsequent drafts add detail and bring us closer to code
  - The final version is an accurate representation of the code
- Don’t be afraid to use the UML for ‘sketching’
  - Your diagram has value as long as it clarifies your thinking
- However don’t use ‘sketching’ as an excuse
  - Too many projects sketch some vague UML diagrams and hope the details will sort themselves out in code
  - This will only happen in small projects with open lines of communication and talented and experienced developers


Analysis Level Diagrams
Help understand the problem
Sketching a solution with few details
Large 'chunky' classes with many jobs
'Spike' Applications
Proof on concept programs that confirm
you can use a particular technology and
that it can accomplish what you require
Design Level Diagrams
Represent a possible solution
Names, signatures, relationships
Enough detail to move into coding
Implementations
The application is incrementally built over a
series of iterations, preferably with unit,
acceptance and non-functional tests
Blueprints of Code
Full detail of implementation
Notes attached to describe intentions
Stereotypes used to show usage of patterns etc…
Usually reverse engineered from completed code