Bournemouth
University

ssrc
software systems research centre

Software Project Management:

Specific issues for managing software projects

Dr Keith Phalp

The course in a nutshell

- Two main aspects
  - General project management
  - Software Project Management: concentrates upon those aspects of Software Engineering which require a specific approach.
    - That is, what is different about Software engineering projects and what we do to address this.
    - Use of particular expertise for different elements (see scheme). That is, we will utilise the experience of those who have research (and industry) expertise, in particular areas.
    - Will also draw upon ***your*** experiences.

Key SSRC Themes

- ***Research relevant to the profession of Computing.***
  - Requirements Engineering
    - Alignment of Business and IT
    - Process modelling and Process Oriented Requirements.
    - Requirements within a Model Driven Development Process
  - Software Modelling
    - Model Driven Development
    - Domain Specific Languages
    - Software Product Lines
    - Automotive Software Engineering, Bosch, Germany
  - Software Process and Quality
  - Global Software Systems
    - Data Mining
    - Web Systems
    - Global Software Development
  - Networks

Unit themes & topics

- The Nature of Software Engineering. What is so difficult about software projects. Software production organisations, roles, activities, phases and software life-cyles and implications for projects. *Dr Keith Phalp: Head of Academic Group and Reader in Software Engineering*.
- The most crucial aspects of the software project. 'Upstream', software development and the importance of *requirements engineering*.
- Roles in the software process, and meeting the needs of stakeholders, business and IT alignment.
- Estimating Software Project Cost & Time. Approaches to cost estimation, what we can learn from them, and practical considerations.
- Acquisition choices: Development with COTS. Open Source Software
- Agile projects – where is the agile approach useful? What are the shortcomings. Guest expert: Sherry Jeary Director of SSRC, and expert in software methods.
- Global software development Outsourcing and Offshoring – What are the benefits or are there any? Guest expert session by Dr Val Casey author of recent book, and of many papers and workshops in this area.

Some of my Background and relevance to unit

- Over 80 refereed publications covering software engineering, software process, software requirements, business and IT alignment, software modelling, software quality, estimation and prediction, plus a variety of technical reports and consortia deliverables.
- Worked on a variety of projects to bring improvements to real industrial processes.
- Ran major European projects, with collaborate software development deliverables.
- Own PhD in Software Process Modelling: Understanding, modelling and improving software development processes.
- Supervised PhDs in software requirements, software modelling, software cost estimation, natural methods for prediction…
- Course leader for Masters in Software Engineering from 1997-2006 (Reader in Software Engineering (2006), HoAG (2007).
  - Taught units including: business process modelling, requirements engineering, systems, software design (both structured and object oriented), software development methodology, software development (programming), **software engineering management**, information systems methods, business systems integration, databases and integrating assignments.

- Current profile at: http://dec.bournemouth.ac.uk/staff/kphalp/

Managing Software Engineering Projects

- What is Software Engineering?
- Problems with Software Engineering
  - The one-off, problems with a manufacturing paradigm.
  - Software as description.
  - Complexity
    - Growth of complexity.
    - The complexity of software.
  - *NOTE: we will consider further problems with software projects when we discuss software requirements.*
- The software engineering approach.
  - Some definitions and laws.

The problem of the ‘one off’

- Many ask why Software suffers compared to other engineering domains, bridges, buildings etc.
- Some attempts to impose a manufacturing approach.
- In other engineering (e.g., cars) we often spend a large budget on a prototype (or lots of these), but the major cost is the replication (production of copies – or similar).
- All of our implementation (software development) is actually like building the prototype in manufacturing; the replication (manufacturing is trivial).
- Hence, we are estimating the cost of the first one.
- Similar, it is argued, to civil engineering (bridges, buildings etc.,), but, as we will see, in these cases there is greater similarity in projects, less complexity, and more tangible (concrete) elements.

- *Hence, estimation, planning and managing all bring unique challenges for software.*

Software is Description

- *Carrying our theme further…*
- With hardware engineering (cars, aeroplanes, computers etc.) you have to **design** it and then **build** it. But . .
  - **“To build software is to build a machine simply by describing it”**

    (Michael Jackson (the software requirements one), 1995)
- You can’t see it, you can’t touch it and you can’t weigh it.
- Its the holes.. (Richard Feynman). It’s “think-ware”.

- This means that it is hard to measure, to quantify, to ensure quality and so on. For example, we test for bugs, but we can’t see them, or be sure we have eradicated them.

Change and uncertainty

- *Change and uncertainty typify the environment for Computing projects.*
- Operating environment, other software, networks, business models.
- The old (or new) legacy systems problem …
- Deadline, budget constraints, priorities.
- Staffing. Also access to client’s staff.
- Technology (again often changing even during the project…, or may even be working towards other new technologies).
- Requirements (what the user wants) change throughout the development of the project and even when delivered.
- ***Why would this be?***

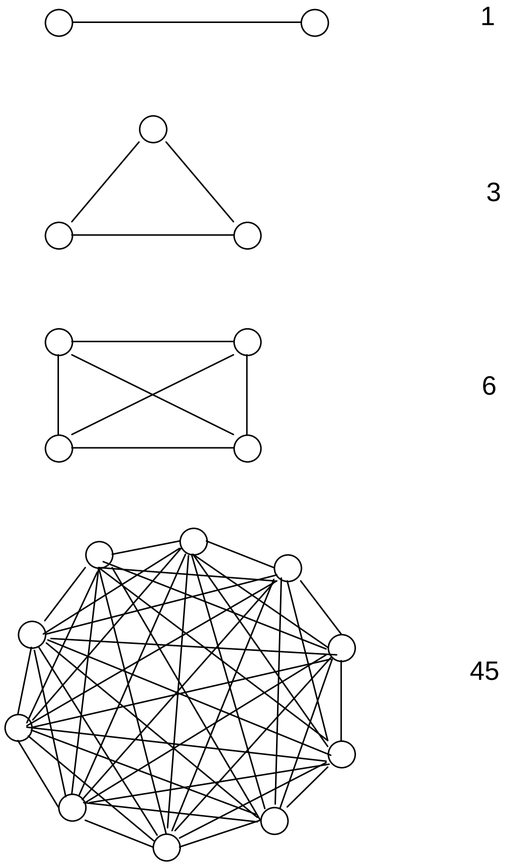
Complexity

- Many of you may have come across the idea of complexity in previous work.
- Complexity (particularly of code) is a well trodden topic in software.
- However, as an aside, let’s think about some (hugely simplified) estimates of the complexity of software compared to other constructions.
- Of course structure (structured code, OO) etc., reduces the genuine complexity, but theoretically one can still consider a line of code (actually it may be worse than this – statements) as being reached potentially from another.

Are programs complex

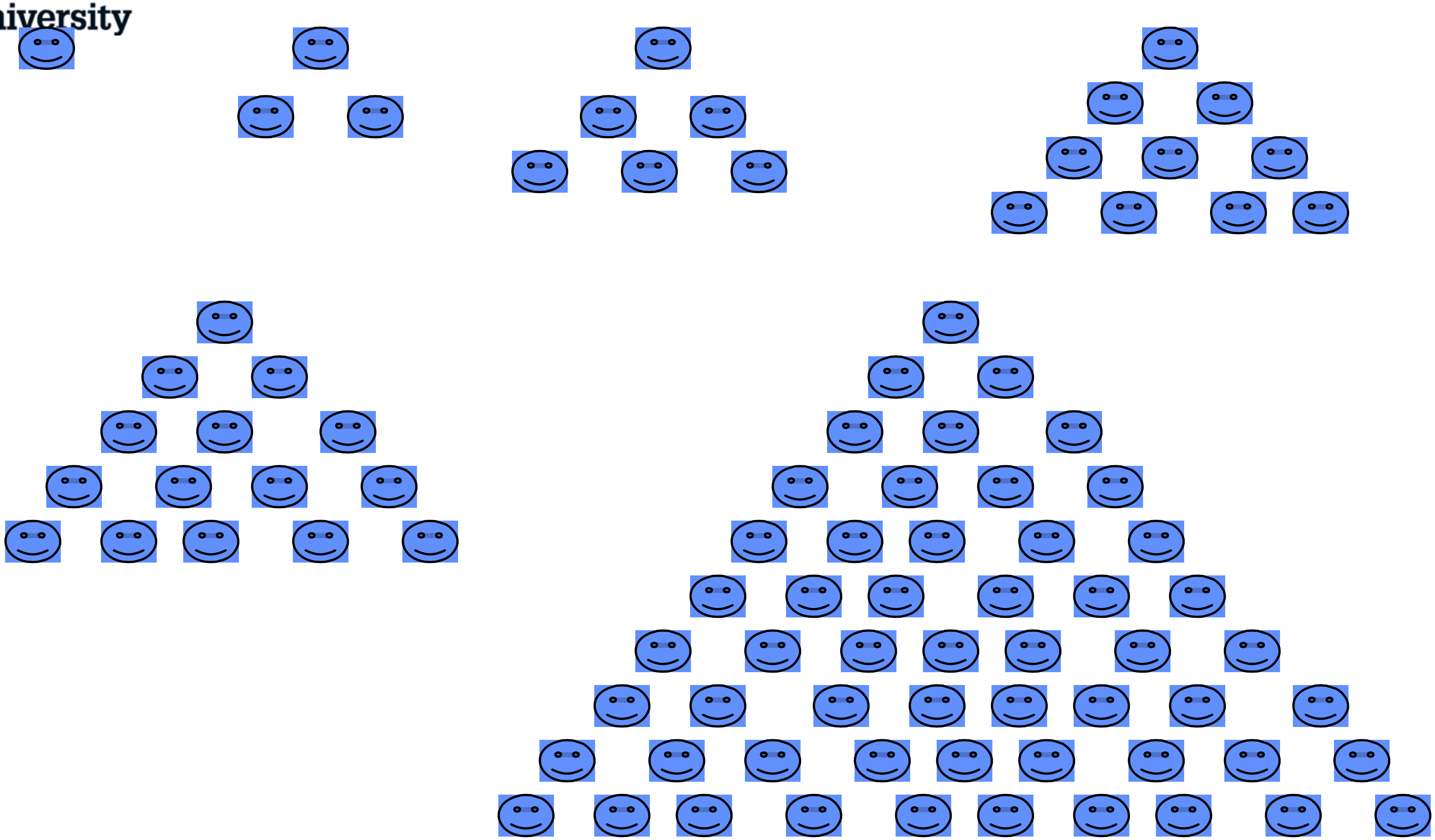
- GOTO. How many lines can I go to from another.
  - E.g., 2 line program, 3 liner, 10 liner…100 liner.
  - (actually more complex than lines of course).
  - Lets assume (another simplification) that we don’t go to where we already are.
    - How does this aspect of complexity grow...
    - So lines are like nodes, and connections are like *connections.*

Connections



- Possible connections (lines) among nodes.
  - You might recognise this pattern.
  - How would you generate it ? (An algorithm?).
  - In theory complexity can increase as the square of the number of components.

A pattern approach



So the growth is…

- Adding a bottom row to the triangle.
- Questions might include..
  - How many rows in the triangle?
  - How many in the bottom row?
  - How many in each row?

- So the total in the triangle is…

Summing this up..

- rows in the triangle = nodes - 1
  - number in bottom row = nodes - 1
  - number in each row?
    - Top row = 1
    - Second row = 2
    - rth row = r *(nodes-1)*
  - So the total in r rows of the triangle is
    - 1 + 2 + 3 + ….+ r **or**
    - 1 + 2 + 3 + ….+ (n-1)

In other words..

- So the total in the triangle.
  - The total connections for n nodes is..
    - Total = 1 + 2 + 3 + ….+ (n-1)
  - Adding up trick (often attributed to Pascal)

$$\begin{array}{rcccccccc}
\text{Total} & = & 1 & + & 2 & + \ldots + & (n-2) & + & (n-1) \\
+\ \text{Total} & = & (n-1) & + & (n-2) & + \ldots + & 2 & + & 1 \\
\hline
2 * \text{Total} & = & n & + & n & + \ldots + & n & + & n
\end{array}$$

  - So 2 times the total = n(n-1)
  - Hence, Total = n(n-1)/2

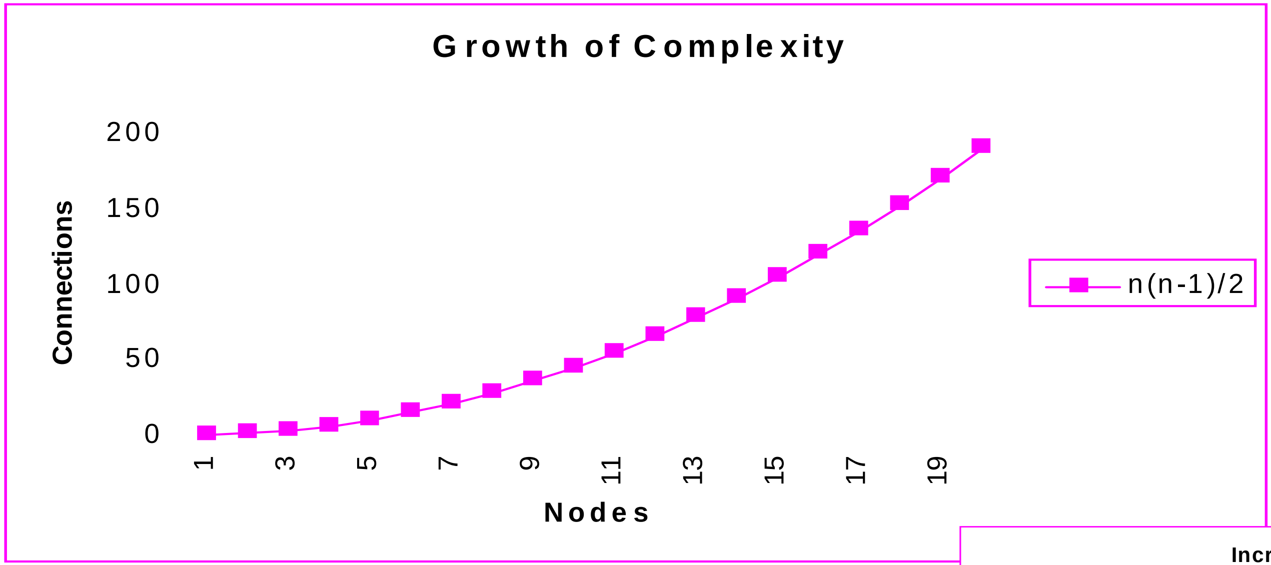
Another pattern approach

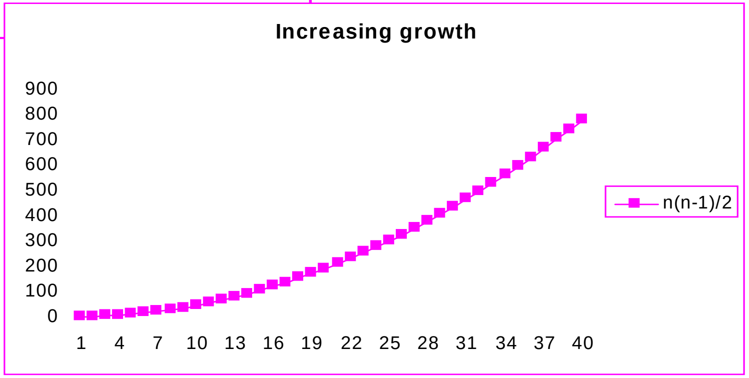
Nodes	n-1	n/2	n(n-1)/2
1	0	0.5	0
2	1	1.0	1
3	2	1.5	3
4	3	2.0	6
5	4	2.5	10
6	5	3.0	15
7	6	3.5	21
8	7	4.0	28
9	8	4.5	36
10	9	5.0	45

Look for the number pattern?

Connection to the triangles?

So growth…


Growth of Complexity
200
150
100
50
0
Connections
1
3
5
7
9
11
13
15
17
19
Nodes
n(n-1)/2


Increasing growth
900
800
700
600
500
400
300
200
100
0
1
4
7
10
13
16
19
22
25
28
31
34
37
40
n(n-1)/2

Back to the point..

- Average flat-pack wall cupboard has about 40 bits of which 4 (the hinges) are subsystems with another 12 (or so) bits. Some of these bits interact (hinges have to fit holes in doors) but mostly only with only 1 or 2 other bits.
- (Arbitrary) Complexity factor : 500
- Your average car has about 2,000 components of which several are subsystems - some with many bits. Interactions are more complex. (*Ignoring the software for now,* ***<u>that's another story</u>***)
- (Arbitrary) Complexity factor : 100,000
- Your typical house has around 20,000 components but interactions are simple
  - (Arbitrary) Complexity factor : 50,000

And there's more…

- Apollo moon rocket system (at the time the most complex engineering project ever) 2,000,000 components with many complex interactions.
- Complexity factor : 100,000,000 (10^8)
- Many commercial software applications have between 100,000 and 1,000,000 lines of code.
  - Depending upon how well engineered (interactions are kept as low as possible) - Complexity factor : 10,000,000 (10^7)
- To keep it in perspective
  - Human brain about 100,000,000,000 neurons, each of which connects to about 1000 others.
  - Complexity factor: 10,000,000,000,000 (10^{13})
- It's not rocket science (it's what rocket science has nightmares about).

We have a problem (Houston)

1 Building **simple** things is (relatively) simple (craft?).

2 Building **complex** things is difficult.

3 The **technology** that works for simple (small) things, doesn't work for complex (big) things.

4 “Real” software is (logically) both big & **complex**

- **very** complex.

5 We have to **teach** you the technology that works for big, complex systems.

6 But, all the things (software) you build here will be (relatively!) small and simple.

The solution:- You will have to **over-engineer**

Back to Software Engineering

Originally, around late 60s (software crisis), software engineering was an aspiration.

“The systematic application of an appropriate set of techniques to the whole of the software development process”.

- Are we there yet?
- What is the evidence of our success?
- (Some significant gains along the way).

The Software Engineering approach…

- targets certain fundamental **goals** or properties that the **product** will exhibit:
  - Understandability
  - Modifiability
  - Portability
  - Integrity
  - Reliability
  - Efficiency
    - in this context, the (software) product includes all supporting documentation: specification, design etc. .

Brief Overview of Terms

- **Understandability**
  - A vital pre-requisite to meeting most other goals.
  - Achieved through the skilful application of appropriate techniques.
- **Modifiability**
  - Software (in particular) changes. Most software is scrapped not because it doesn't work but when it becomes too expensive to modify it.
  - Achieved through good design - see later
- **Portability**
  - The same **functionality** is often required on different **platforms**. Again, achieved through good design

Further terms

- **Integrity**
  - The system is complete, "hangs together" and resists corruption.
  - Yet again . . . .
- **Reliability**
  - The system tolerates an imperfect environment and operates correctly over an extended period.
  - Achieved through good design and testing.
- **Efficiency**
  - The above are achieved with the minimum of resources.
- ***Of course, no-one said it was easy***

Assorted Laws of Software

- **Bersoff's law:** No matter where you are in the development process, the system will change and the desire to change it will persist for as long as the system exists.
- **Hofstaders law**: It always takes twice as long as you think - even when you take Hofstaders law into account.
- **Brooke's first law**: adding manpower to a late software project makes it later. ***(WHY IS THIS?)***
- **Anon**: Program complexity grows until it exceeds the capability of the programmer who must maintain it.
- (So only be half as clever as you can be when you write software because you will have to be twice as clever when you come to maintain it).

Further Laws

- **Lord Falkland’ Rule:** When it is not necessary to make a decision it is necessary **not** to make a decision.
- **Brays’ law:** Given the facts the harder it is to decide between options, the less it matters which you choose.
- **Anon:** It is impossible to make anything foolproof because fools are so ingenious.
- **Lubarsky’s law:** There’s always one more bug.
- **Murphy’s law:** (also attributed to Saudde): If it can go wrong, it will go wrong, and at the worst possible time.

So to sum so far

- Software Projects are somewhat different to other disciplines.
- They are often one-offs, suffer from inherent change, and often require the development of products which are both intangible and about as complex as most human undertakings. This makes estimation, planning, managing and controlling them…
- Tricky – BUT: Interesting, fascinating, challenging, rewarding. (Lucrative ☺)
- So next let's consider the nature (and evolution) of the software development process (and of models).

Models of the Software Development Process

- Computer Scientists and Software Engineers like to develop *models* of the development process.
- To understand and, ultimately, **optimise** and control, the process.
  - And, hence, improve **quality** and **productivity** (i.e., profitability).
- Huge research area (over decades) of modelling software process and software process improvement (e.g., EuroSPI, for which I am on IPC).
- Also related themes on capability, maturity (SEI, SPICE, ISO, TickIT, and so on).
- In addition, Process Modelling a discipline in its own right.

Pragmatic Modelling Goals

- To **define** the activities to be carried out.
- To identify the outputs (**deliverables**) that must be produced.
- To provide checkpoints for **project management.**
- To introduce **consistency** across and **repeatability** within the organisation.

Further reasons given

- **Higher process visibility**
  - Better understanding of development processes [Dowson, 1986; Kellner, 1989]
  - Aids the monitoring of progress, and allows managers to give better guidance to engineers [Kellner, 1991; Kellner, 1988; Lehman, 1989].
- **Explicit descriptions** of processes encourage better communication about the process [Kellner, 1988].
- Through analysing the model we **identify areas of weakness** and possible improvements [Kellner, 1991]
- **Provides a framework for software measurement**.
  - Quantitative data about either process or product may be gathered more efficiently [Rombach, 1993; Tate, 1993; Shepperd, 1992]. ***Own Masters research did exactly this.***

And more…

- **Experimentation with processes** [Humphrey, 1989].
  - Understanding, guidance, control, simulation & enaction.
  - Models support process evolution [Kellner, 1988].
- **Model as a process template** that can be instantiated for each project [Chandrashekar, 1993; Ince, 1994].
  - Standardisation while still allowing process flexibility for individual projects [Humphrey, 1988; Ince, 1994].
- **A defined process** [ISO & BS5750: DTI, 1990; SEI CMM, Paulk, 1993; Coallier, 1994; Ince, 1994].
- Models **facilitate process reuse** [Basili, 1991, Redwine, 1988; Dyer, 1993].

A simple Model

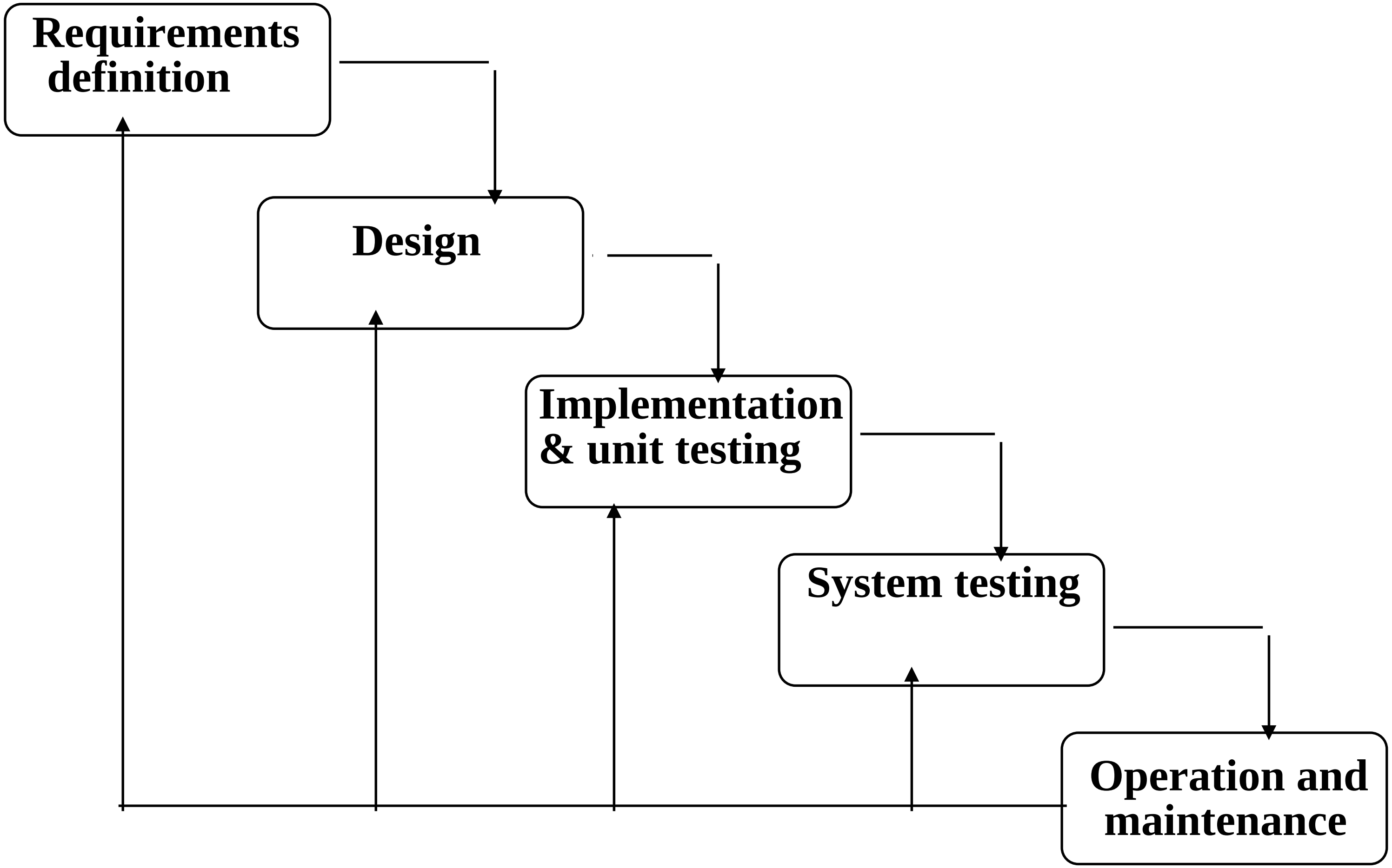
- In essence, it is just like making anything else, and very simple.
  1. Decide what you are going to build.
  2. Work out how to build it.
  3. Build it.
  4. Test, deliver, maintain, evolve ….
  - The commonest mistake is (almost incredibly) to start with 3.

  - Historically we have slightly more complex models.

Waterfall and Lifecycles

- Process as ‘phases’ or ‘chunks’ of different activity types
  - (earliest Bennington, 9 phases, 1956).
- Waterfall model (Royce 1970) is perhaps the best known of all the lifecycle models
- Sommerville uses a waterfall model which consists of five distinct stages.
  - (Software Engineering Third edition).

Simple ‘Waterfall’



Advantages and Disadvantages

- Simplicity.
- Convey major software engineering activities to the uninitiated. (*next few slides illustrate*).
- Clear high level view of the process.
- Provide crude milestones.

Inability to cope with:

- Iteration.
- Prototyping.
- Levels of detail or abstraction.

More latterly

- Modern methods, rapid development, OO etc..
- ***Agile methods***

Life-cycle models might include..

- Basic stages (reminder):
  - Requirements engineering.
  - Design.
  - Implementation.
- Usually fleshed out in more detail (next slide).
- Note. Assuming a procedural approach.
  - This may not be the best modelling strategy.

General Life-cycle tasks

- Strategic planning, bright idea
- Preliminary investigation of problem area
- **Requirements Engineering**
  - Investigate system requirements
  - (Requirements Elicitation)
  - Analyse requirements
  - (Systems Analysis)
  - Determine feasibility
  - Specify the Required System
  - (Write the specification)
- **Design**
  - High level design (HLL)
  - Low level design (LLD)
- **Implementation**
  - Coding
  - Integration
  - System testing
- Installation (hand-over / commissioning)
- Maintenance
- De-commission (scrap)

Deliverables include…

- Requirements Engineering
  - Analysis/requirements document
  - System specification
  - Acceptance test plan
  - Cost benefit analysis
- Design
  - High level design documents
  - Low level design documents
  - Functional/integration test plan
- Implementation
  - Source code
  - Integrated/tested system
  - User documentation

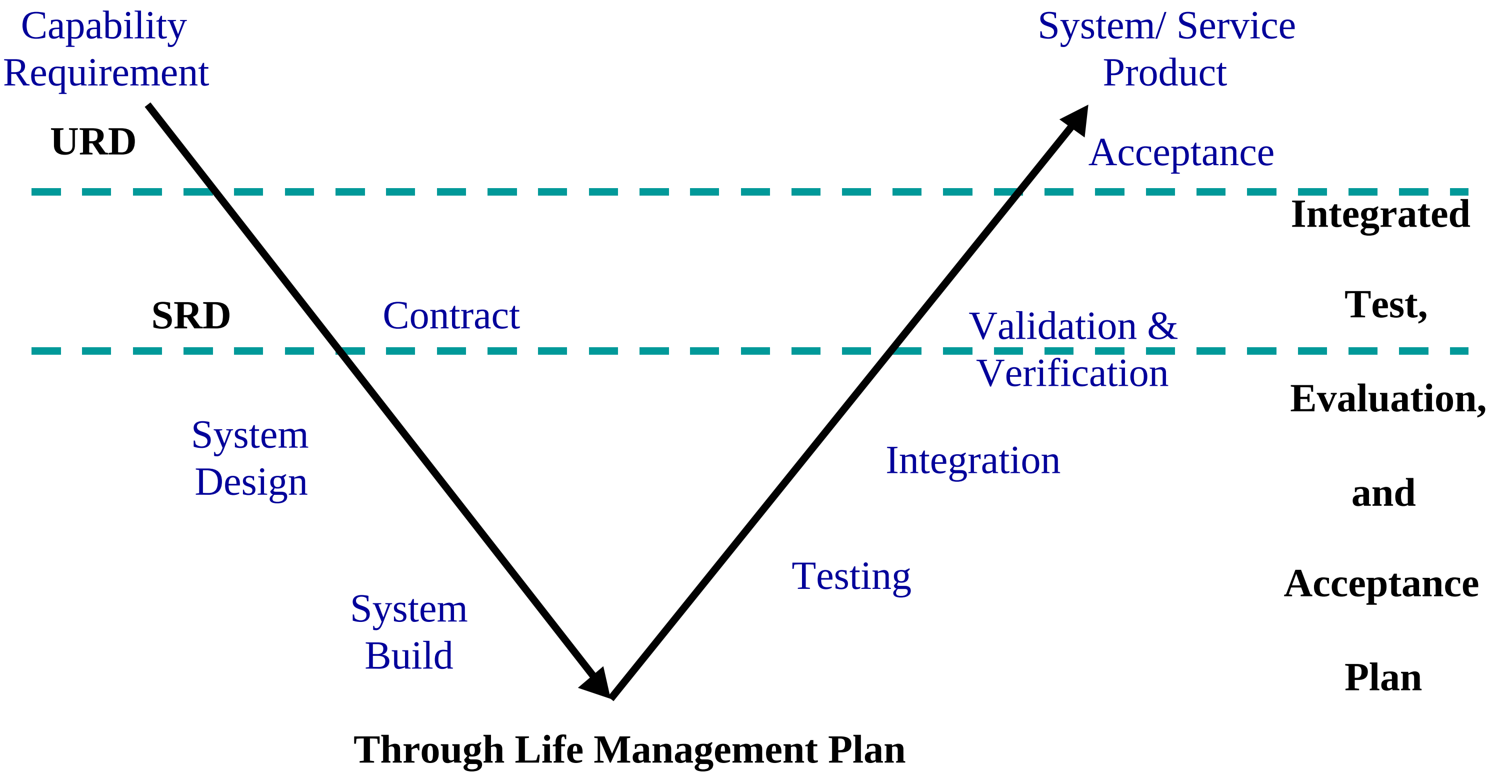
Other concurrent activities…

- Often (amazingly) omitted from such process models.
  - Project management
  - Configuration Management
  - Quality Assurance
  - Feasibility (cost benefit) assessment
  - Testing.

Further Models and Issues

- Lots of variations on a theme.
- More iterative (cyclical), e.g., Boehm's spiral model.
- Some to illustrate different relationships among activities (variants of V model).
- Most convey similar activities.
- Most organisations have in-house defined process (variations on a theme, e.g., JP Morgan V model).

Documents in V model (MOD variant)



MODAF-M03-004

An Exercise

- Given below (in a random order) are the tasks identified in one company's software development life-cycle model. Draw a flow diagram indicating the likely ordering of tasks. Note that your flow diagram can (should !) include branches and loops.

- Main module coding
- Preliminary investigation
- Prototyping main functions
- Modification of system requirements
- Outline design
- Main module specification
- Sub-module design
- Sub-module coding
- Requirements analysis
- Main module design
- System specification
- System integration
- Main module testing
- System requirements gathering
- Sub-module testing
- Feasibility study
- Prototype evaluation
- Delivery/installation
- Sub-module specification
- Whole system testing
- Sub-module integration

Process Maturity

- Software Engineering Institute (SEI) at Carnegie Mellon University proposed a ‘Capability Maturity Model’.
    - Categorises (and ranks) an organisation in terms of its process capability.
    - Later versions move towards profiles rather than levels - but the idea of levels has stuck.
- Many similar models (e.g., SPICE in EU).
- CMM hugely influential.

CMM levels

LEVEL	CHARACTERISTIC
5. Optimizing	Improvement fed back in to process
4. Managed	Measured Process (Quantitative)
3. Defined	Process defined and institutionalized (Qualitative)
2. Repeatable	Process dependent on individuals
1. Initial	Ad hoc or chaotic process

CMM Strengths

- Realistic and defined improvement targets.
- Emphasis on analysis and design methods.
- Lots of supporting materials.
- Emphasis on people and organisational issues.
- Quick solutions, as no need to understand current process.

Critics said…

- Few organisations exist on which to base the higher levels.
- Improvement is self-fulfilling.
- Only concentrates on the positive (good practice) and not concerned with current process. Hence :
  - No increase in understanding or help in documenting, guiding or controlling process.
  - No elimination of bad practices.

Criticisms 2

- Ordinal scale bands together too many at lower levels.
- Focus on score. Not on improvement
- Very biased to DoD and real-time applications.
- Archaic view of CASE technology.
  - E.g., level two being a prerequisite for CASE).
- Little empirical evidence of project improvement.

Management of Projects

- To consider how a range of computing activities can be satisfactorily managed.
- The management of the provision of computing products and services should be considered from both suppliers' and consumers' perspectives

Management – a definition

- “All the activities and tasks undertaken by one or more persons for the purpose of planning and controlling the activities of others in order to achieve an objective or complete an activity that could not be achieved by the others acting alone.”
- **Thayer, R.H., 1988. *Software Engineering Project Management.* IEEE Computer Society.**
- Note: Long-term *and* short-term considerations.

List the activities undertaken by a project manager.

Risk Management

A *risk* is anything that could cause a project to fail to meet one or more of its goals.

Risk Management activities

- Risk identification.
- Risk analysis.
- Risk prioritisation.
- Risk avoidance/containment actions.
- Contingency planning.
- Risk monitoring/resolution.
- Maintaining the Risk Management Plan.

Software Production Organisations

- IT Group
  - Systems for internal use.
  - Systems provide part of business service.
- Embedded Systems - Develop the software component of an engineering product.
- Software House - Develop bespoke software system for a customer, under contract.
- Package Developers - Develop software system for a mass market.

Seminar Task (1)

- Read the journal article:
  - Yourdon, E., 1995. When Good Enough Software is Best. *IEEE Software*, 12 (3), May 1995, pp79-81.
- Summarise Yourdon's argument. Critically review whether that argument has practical use in the management of computing projects.

Seminar Task (2)

- Read either or both these journal articles:
  - Reel, J.S., 1999. Critical Success Factors in Software Projects. *IEEE Software*, 16 (3), May/June 1999, pp18-23.
  - Yeo, K.T., 2002. Critical failure factors in information systems projects. *Intl. Jl. Of Project Management*, 20 (3), 2002, pp241-246.
- What did you learn about:
  - Factors that affect the success of computing projects?
  - The definition of success (or failure) of a computing project?
- What are the limitations of these papers in answering these questions?