Working out What

Planning and Risks

Reusing this material

archer

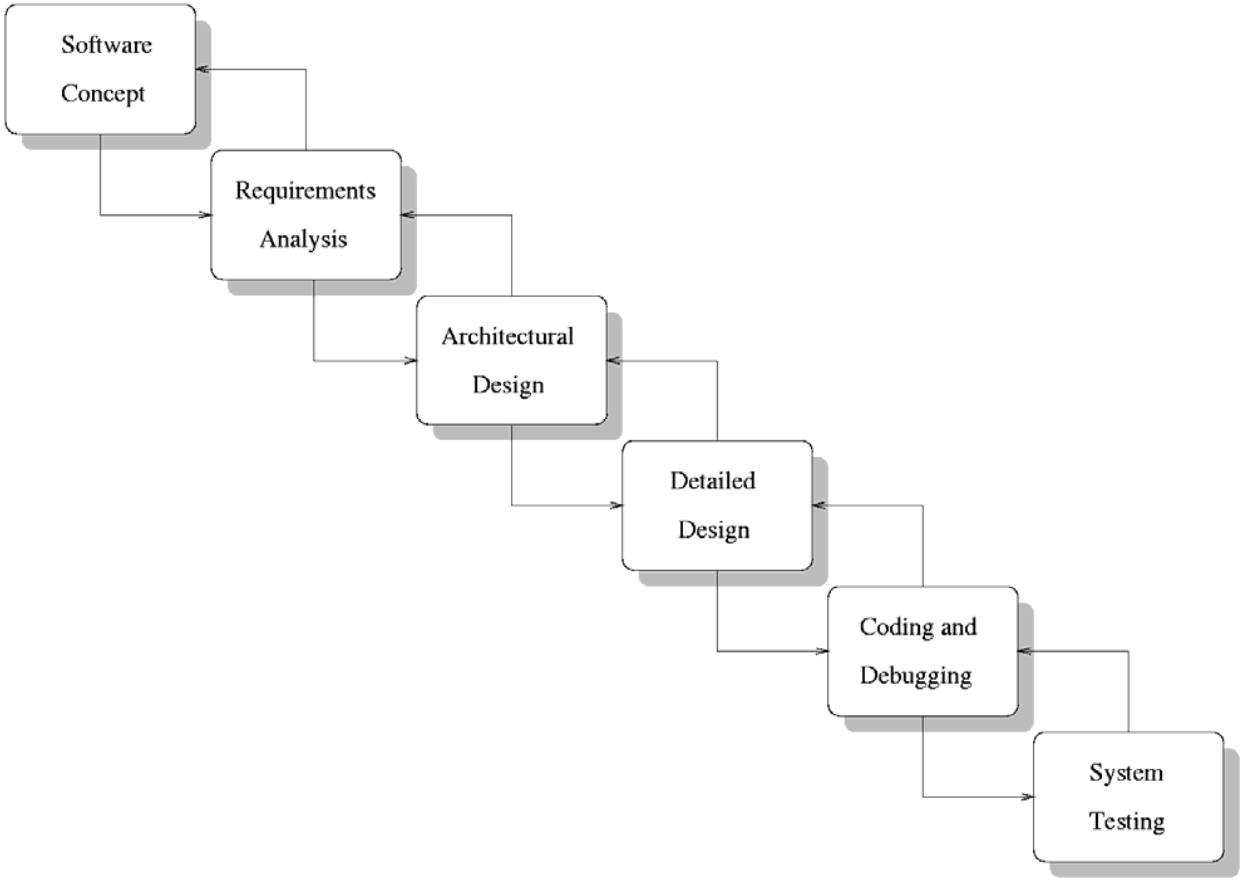
epcc

THE UNIVERSITY OF EDINBURGH

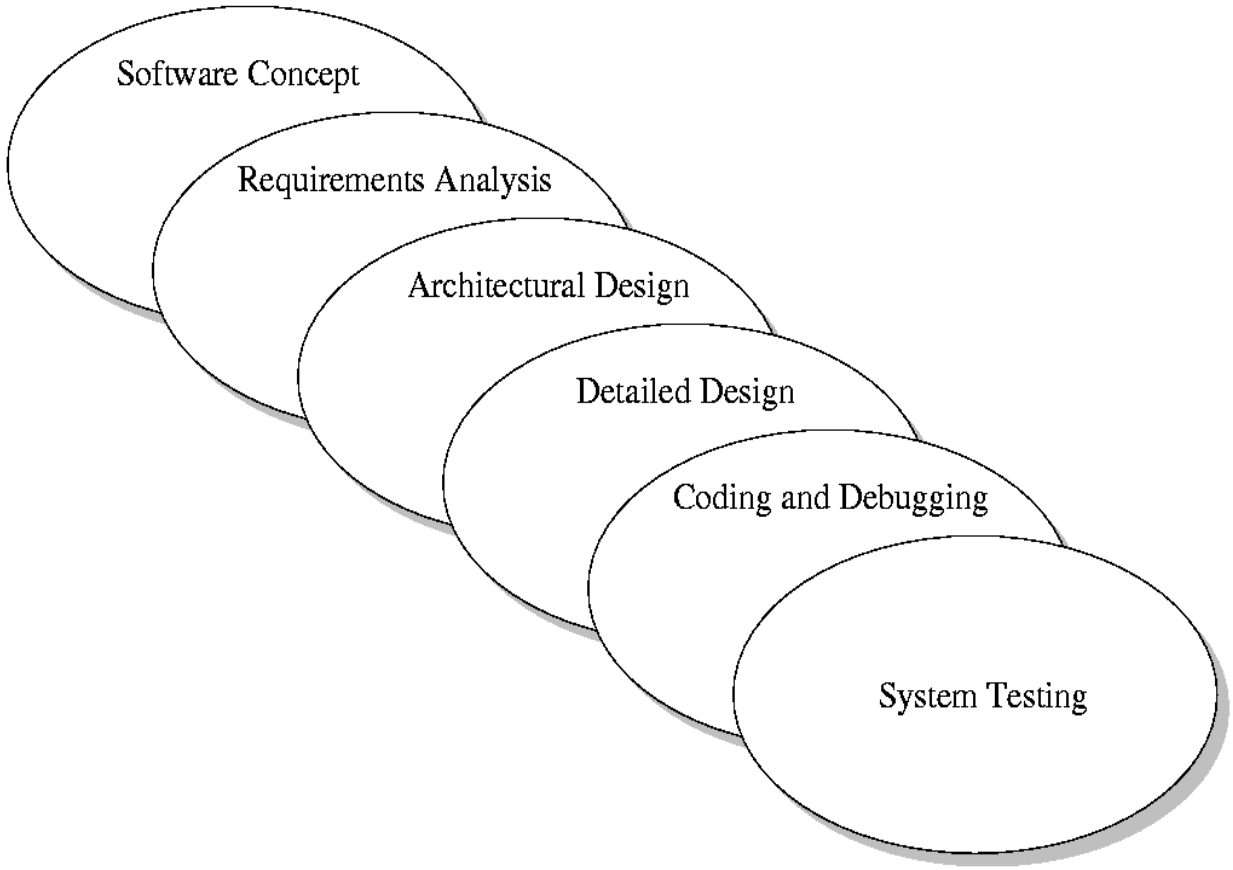
Development Models

- There are many approaches to development but most fall into the following categories:
  - Code-like-hell
  - Waterfall
  - Iterative
    - Staged Delivery Waterfall
      - Design to Schedule variant
    - Evolutionary Prototyping
    - Evolutionary Delivery
    - Spiral Model
    - Agile (many would say is a model in and of its own)

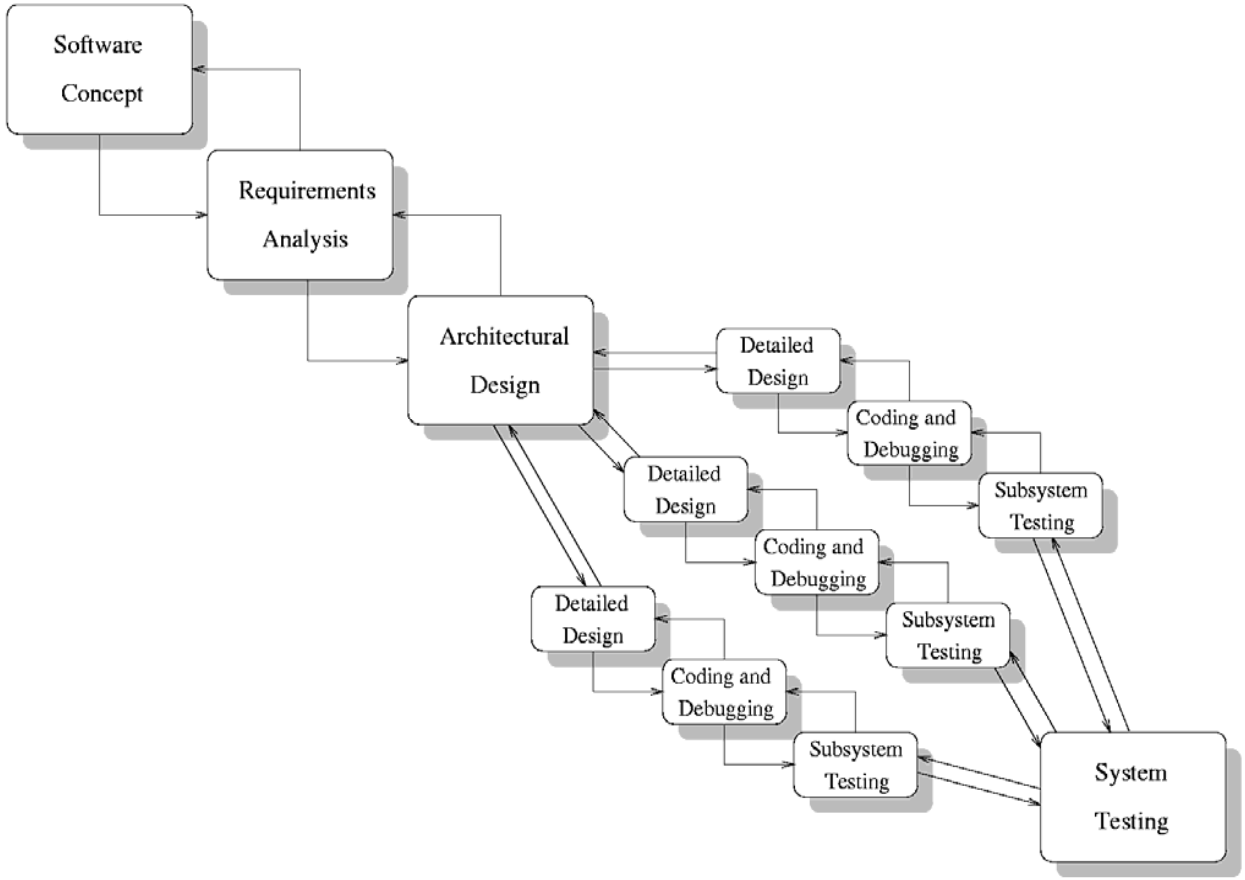
The Waterfall Model



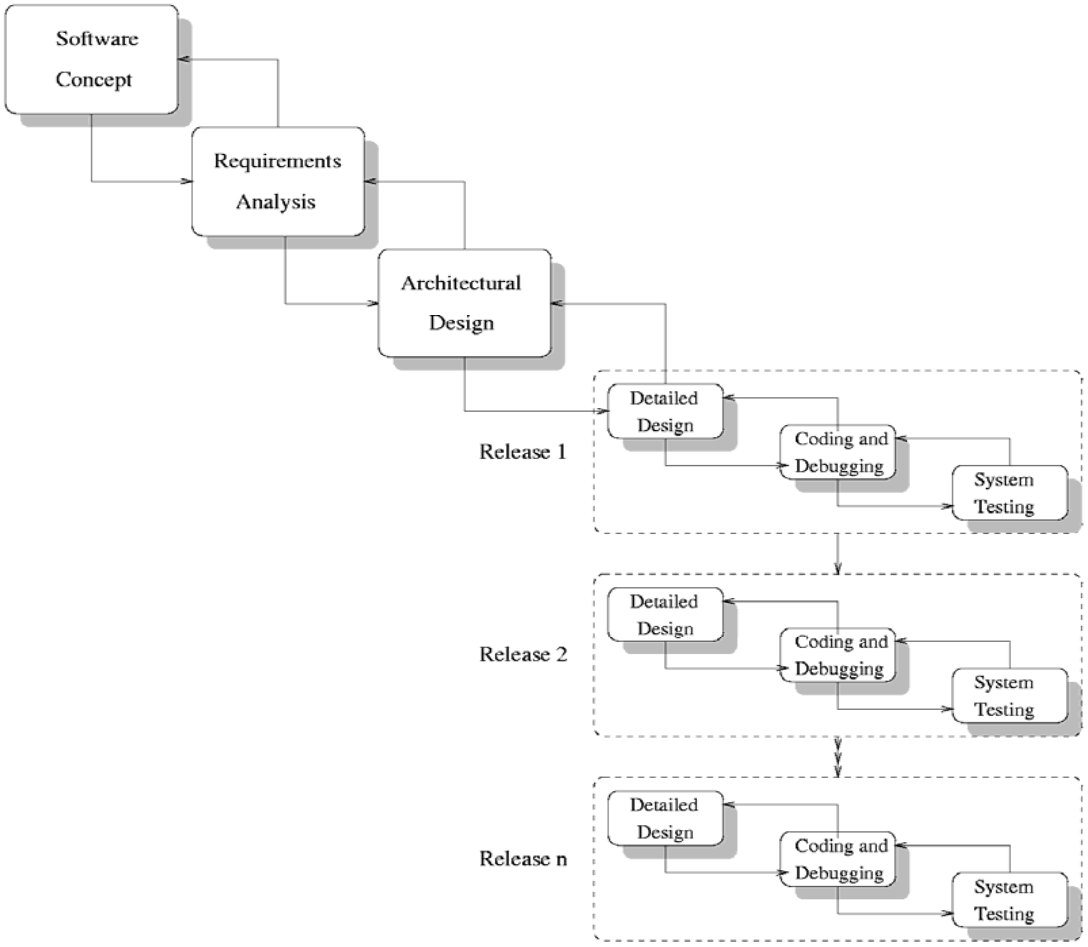
Overlapped Waterfall



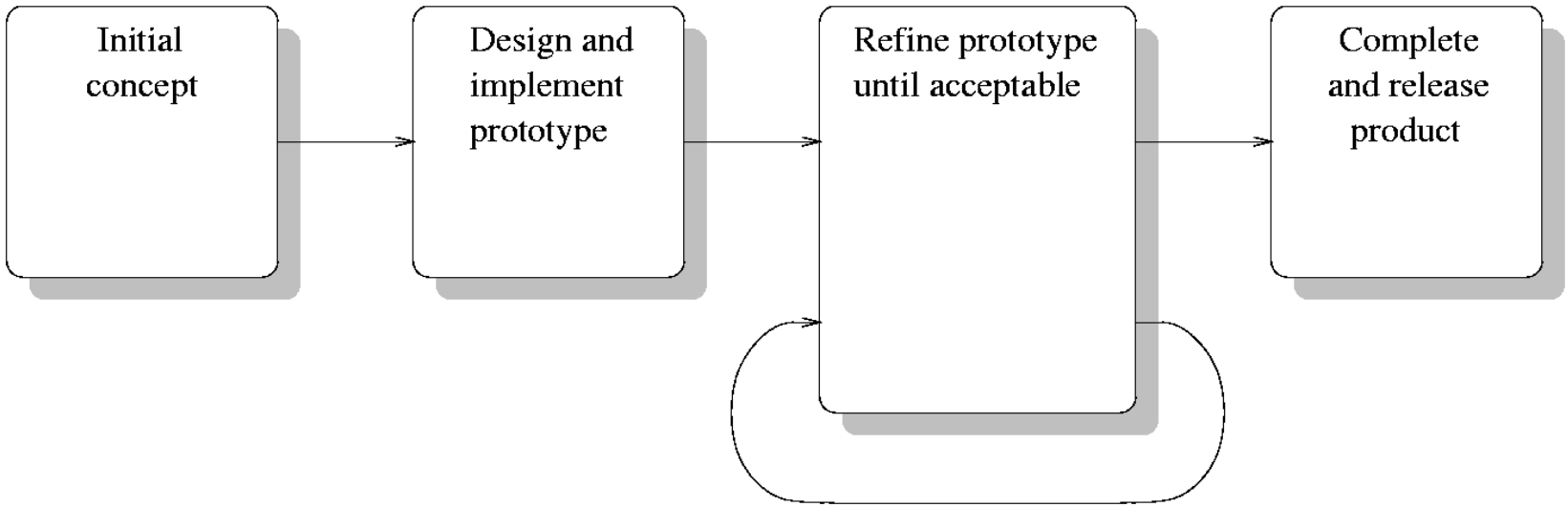
Waterfall with Subprojects



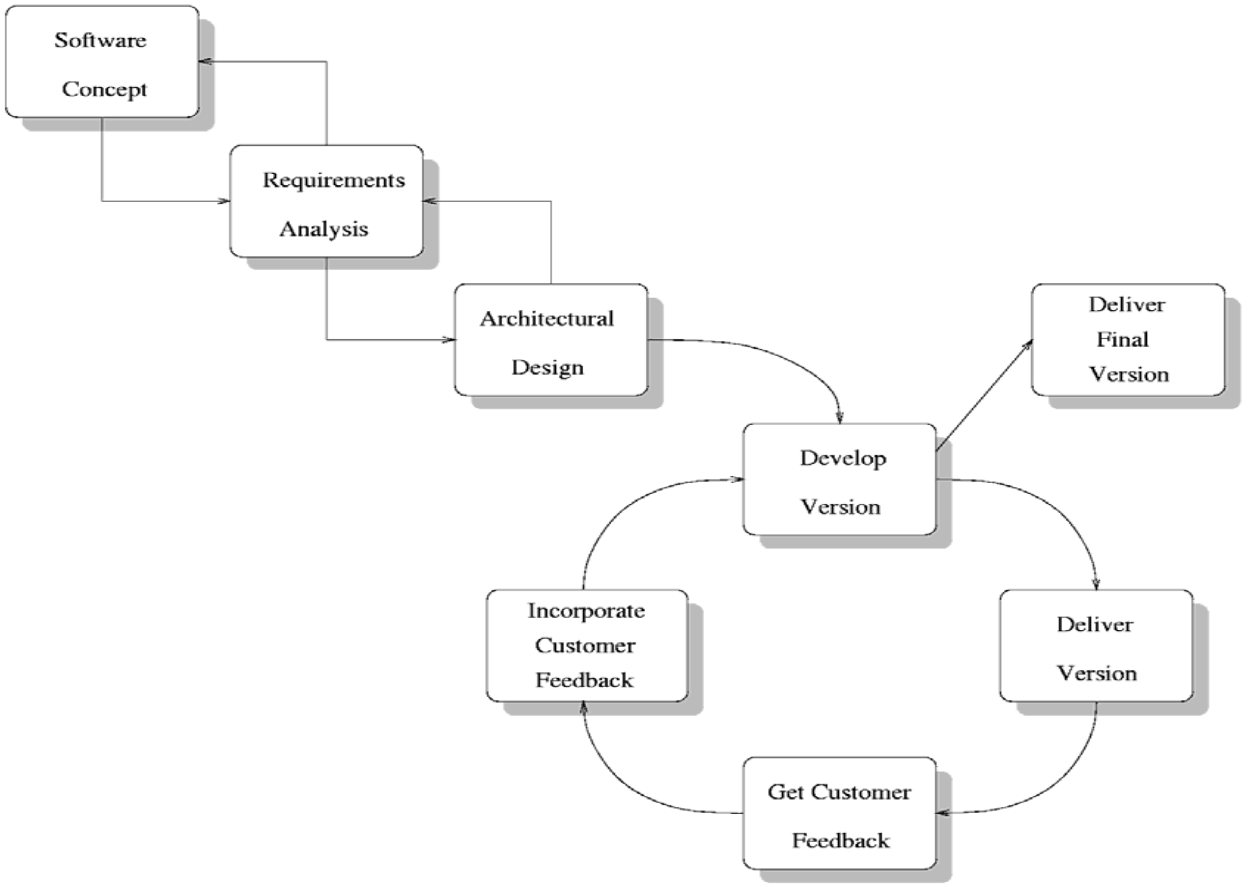
Staged Delivery



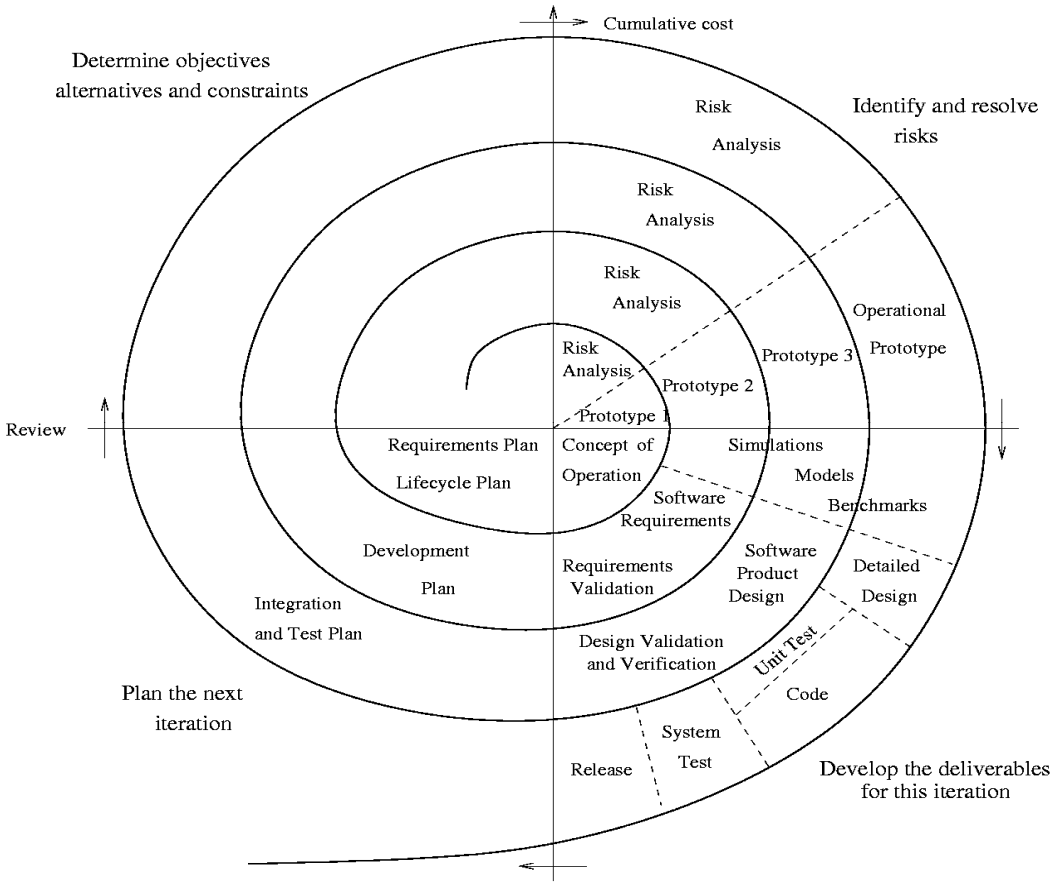
Evolutionary Prototyping



Evolutionary Delivery



The Spiral Model



Quick Question

- Which Approach do you use?
- Do you follow its rigorously?

Estimation

- Want to start a project?
- It is important that you know *in advance*:
  - How long it will take
  - How many people it will need
  - How much effort it will require
- You can then discuss cost
  - …and shake hands
- Estimation is *hard*
  - Projects overrun
  - Projects go overbudget

Motivation

- Two thirds of all projects substantially overrun their estimates;
- The average large project misses its delivery date by 25 to 50 percent;
- The size of the average schedule slip increases with the size of the project.

Disasters

Project	*First;Last Estimate* *Cost (M$)*	*Schedule (months)*	*Status at end*
PROMS (Royalty Collection)	12;21+	22;46	Cancelled, Month 28
London Ambulance	1.5;6+	7;17+	Cancelled, Month 17
London Stock Exchange	60-75;150	19;70	Cancelled, Month 36
Confirm (Travel Reservations)	56;160+	45;60+	Cancelled, Month 48
Master Net (Banking)	22;80+	9;48+	Cancelled, Month 48

- Table from B. Boehm

Why Do We Get It Wrong?

- Any Ideas?

- Write down a few ideas

Why so hard?

- Estimates are needed and relied upon ***early***
- The functional requirements do not provide a solid background
- It is not immediately known how long it will take to develop the features
  - Particularly if the desired outcomes are genuinely novel.
- ***Feature Creep*** is a killer
  - It is the unpredictable yet near-certain change of the functionality as the project progresses.

Why so hard? *(cont)*

- Staff ability
  - Estimators
  - Programmers
- Code reuse
  - Is code reused?
  - Is code to be reused?
- Programming language used

Effects of underestimation

- Understaffing
- Underscoping the quality assurance effort
- Setting too short a schedule.
- These in turn could lead to:
  - staff burnout
  - low quality
  - loss of credibility as deadlines are missed.

Effects of overestimation

- Parkinson's Law:

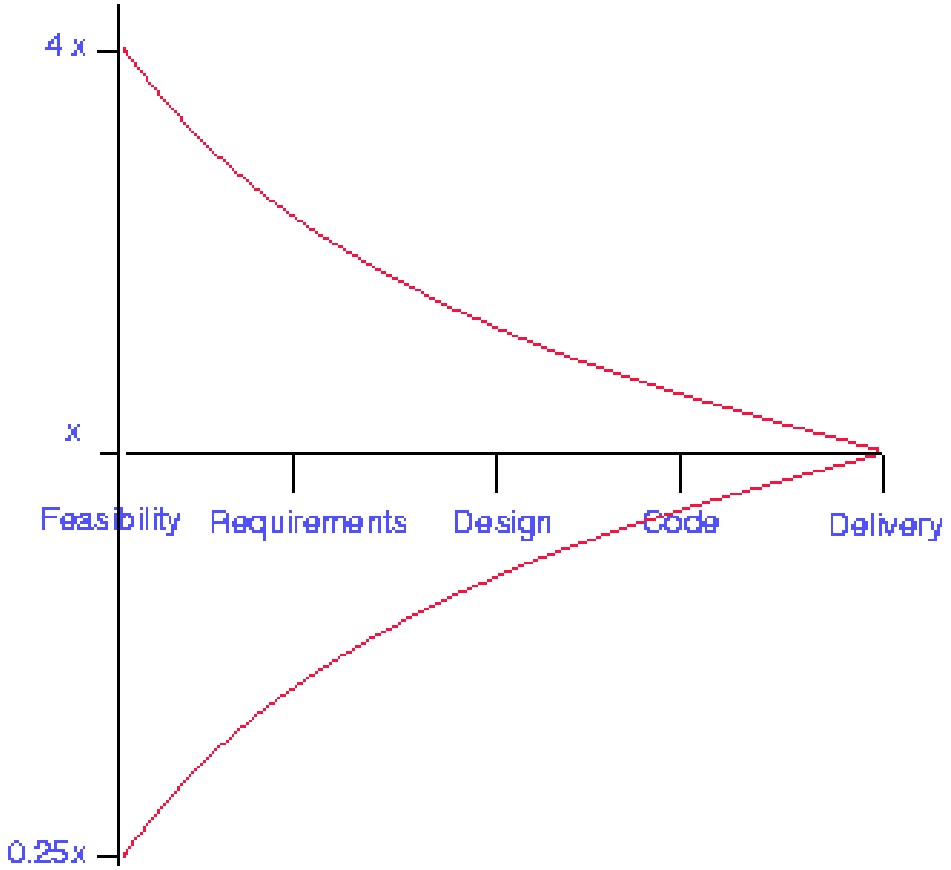
Work expands to fill available time

- The project will take at least as long as estimated even if it was originally overestimated.

When to estimate

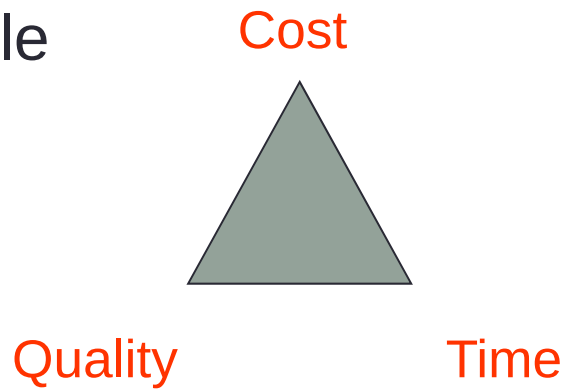
- The first estimate is necessary before the start of the project.
- Estimation is a process of gradual refinement.
- It does not finish until the project finishes.

Cone of uncertainty



Projects revisited

- Remember the project triangle



- Quality for software projects
  - Functionality
  - Correctness of the code
- Cost and time depend on ***Effort*** and ***Schedule***

Effort and Schedule

- ***Effort*** is the total number of time units (e.g. weeks or months) a person needs in order to complete the task.
- This may break down to effort from more than one persons, so as to take advantage of certain skills and parallelise the work to gain overall time.
- Caveat:
  - the more people one adds to a project the more one needs to work so as to coordinate them and the more they communicate so as to interact successfully, thus yielding overheads.
- COCOMO:
  - For projects that can be achieved with 2-3 people teams

    $$Effort = a*\{size\}+b$$
  - For “large” projects

    $$Effort = a*\{size\}^{b}$$

Effort and Schedule *(cont)*

- Effort division may cause gaps in personnel utilisation.
- ***Schedule*** is the breakdown of effort per person at any given time.
- It is sometimes defined as the total time for the project.
- Schedule can be derived from effort.
- Rule of thumb:

  optimal schedule = 3 * effort$^{1/3}$ (McConnell)

Estimation Methods

- Expert Opinion
- Estimation by Analogy
- Metrics

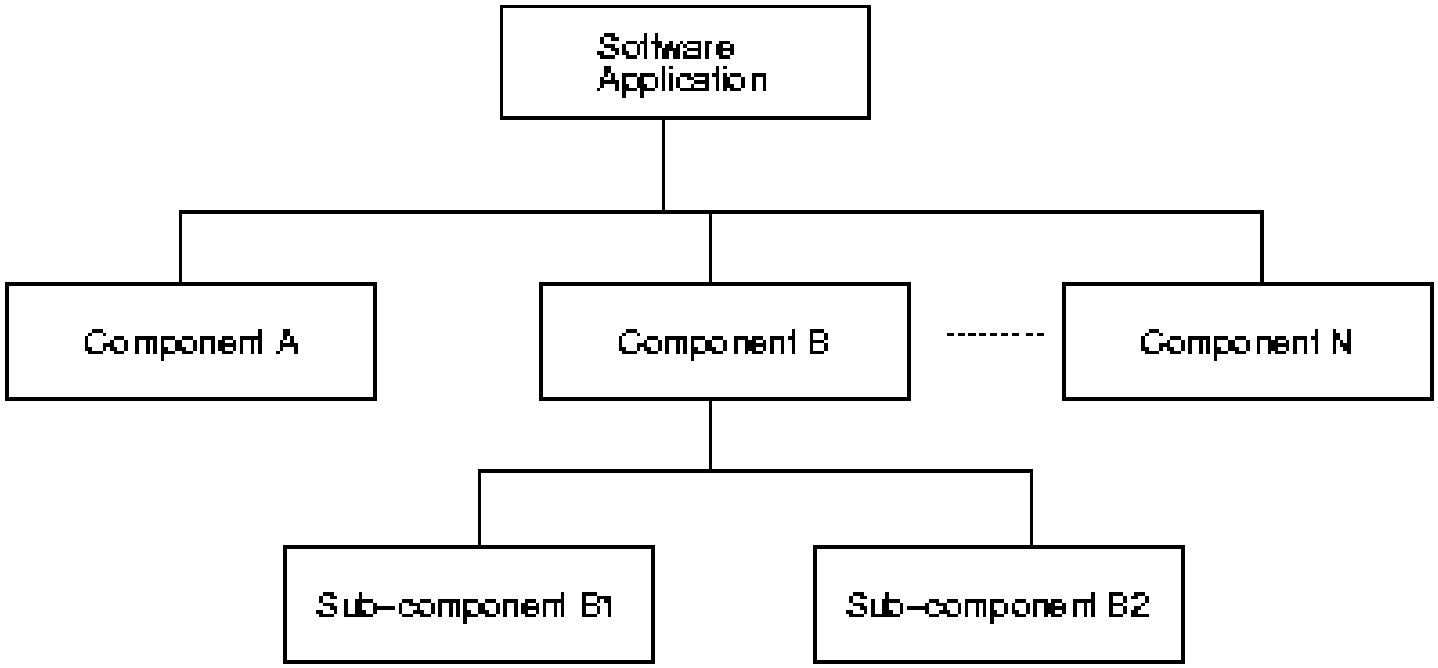
Expert Opinion

- The classic method
- Prior experience is *the* key to estimation
  - This is true in all cases
  - Good practice: only use *documented* data
- Estimation is left to a senior member of staff, possibly the Project Leader
  - Good practice: involve the developers
- Techniques to mitigate single point of failure
  - Work Break Structure
  - Delphi

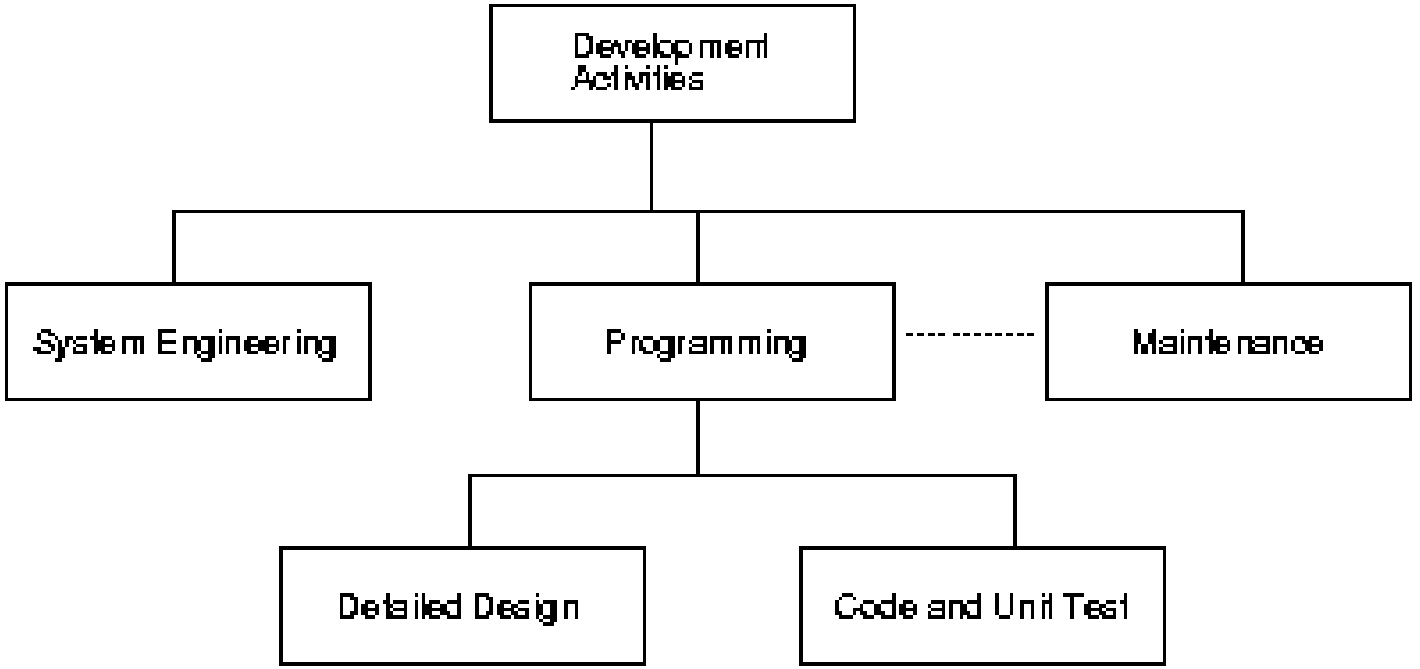
Work Break Structure (WBS)

- Two hierarchies:
  - Software product
  - Process

Product Hierarchy



Process Hierarchy



Work Break Structure *(cont)*

- Lay out what you want to do
  - Requirements at an early stage
  - Architecture document at a later one
  - Other documents (e.g. priorities list)
- Break the problem into components (*workpackages)*
- Then break the workpackages in tasks
- Perhaps go another level down
  - Good practice: the more you break and estimate the better
    - “The sum of the errors is less than the error of sums”

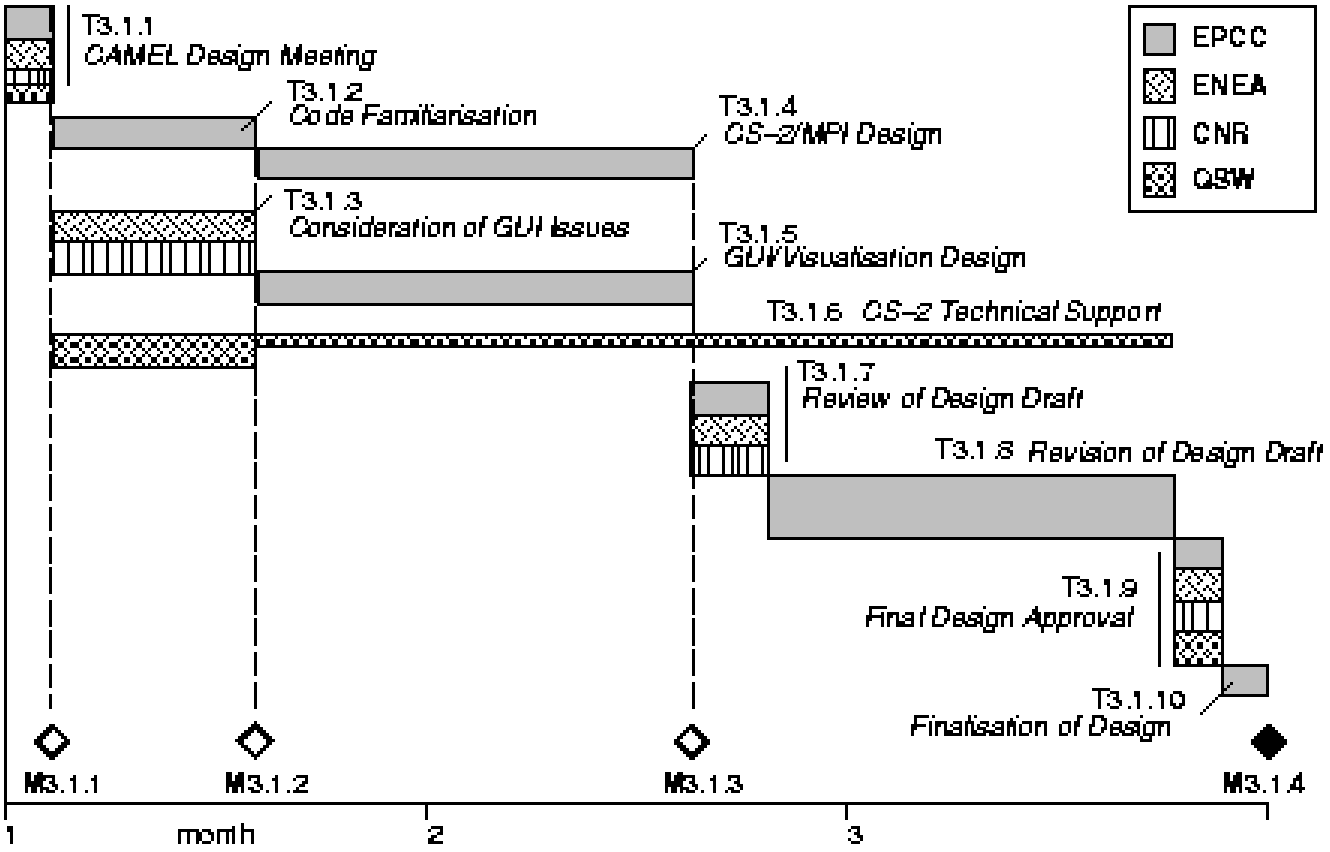
Work Break Structure *(cont)*

- Appreciate the workload of each component
  - Good practice: How many days to the week?
  - Good practice: Check *documented* experience of similar tasks

Gantt Charts

- Turn back to the workpackages and seek dependencies
  - E.g. “WP1 and WP2 can run in parallel, but can’t start WP3 until WP1 is finished”
- How to create a Gantt chart
  - Get a biig piece of paper and loads of post-its
  - Each post-it is a task
  - Try to parallelise tasks as much as dependencies let you
    - horizontal axis represents time, vertical axis resources
  - Length of chart is *time*

Gantt Chart Example



Lets Try This Out

- Lets go to a non-software project and try and break it down into tasks, work out an order and try some rough estimates
- *Building a House – You have to build, decorate and have certified a house*
- *Staff:*
  - *Builder(s)*
  - *Decorator(s)*
  - *Electrician(s)*
  - *Gardener(s)*
  - *Joiner(s)*
  - *Plumber/Gasman*
  - *Roofer(s)*
- *Create a tasklist and put together a Gantt chart*

Delphi

- Participants are asked to make assessment individually
- Results are collected, tabulated, and then returned
  - Group discussion optional
- Second round of individual assessments taking into account the previous results.

Expert Judgment evaluation

- Pros:
  - Applicable to very original projects.
  - Inherent local calibration.
  - WBS can lead to a well documented process.
- Cons:
  - Big dependency on experts' abilities.
  - Big dependency on experts' presence; how easy does Delphi cope with high staff turnover in a small organisation?
  - There is so much information outside your establishment

Estimation by Analogy

- The cost of a new project is estimated by analogy to similar completed projects
- Estimates based on ***historical*** data from within an organisation are more accurate than estimates based on rules of thumb or educated guesswork.
- International Software Benchmarking Standards Group (ISBSG) maintains and exploits a repository of international software project metrics to help software and IT business customers with project estimation, risk analysis, productivity and benchmarking.

Estimation by Analogy *(cont)*

- Pros:
  - Can be accurate.
  - Simple if the organisation repeats similar work.
  - Estimates are immediately available.
  - Encourages detailed documentation.
- Cons:
  - Can be very unreliable.
  - It is very hard to assess the differences between the environments and thus assess the accuracy of the ISBSG data.

Estimation Metrics

- No definition for project size
  - “Something that encompasses all 3 aspects of a project”
  - “The area of the project triangle”
- A metric makes estimation:
  - Transparent: The abilities of the practitioner are irrelevant
  - Repeatable: Exercised by various people at different times yields the same results
  - Reliable: The end results, although never accurate, can be closer to the Truth
- Additionally:
  - The productivity of staff and organisations can be monitored.
  - The results can be shared across the globe.

Lines of Code

- How many lines in your files?
- Pros:
  - Very natural
  - Easily countable
- Cons:
  - Available too late
  - What is a line?
    - Comments?
    - Compare

```
d = ++c;
```

with

```
c++;
d = c;
```

Function Points

- Function Points provide a more objective and reliable estimate of the size of a project
- *"Measure of the size of computer applications and the projects that build them"*
- With Function Points one can:
  - Measure Productivity (100 FPs produced this month)
  - Estimate development and support (100 FPs required thus *XXX* man months)
  - Monitor outsourcing agreements (this library requires 100 FPs, says the contract, these are only *XXX*)
  - Normalise other measures (100 defects in a 100 FPs is far worse than in 10000 FPs)

Function Points Evaluation

- Pros:
  - Prescriptive method for sizing.
  - Can be applied reasonably early in project lifetime.
  - Immune to language and platform idiosyncrasies.
  - Large user base-active effort.
- Cons:
  - Manual, fiddly process.
  - Disagreement about its applicability across the various types of modern projects.
  - Not ideal in the requirements capture period; although the method can be applied, it is too much work for such a volatile description.

Estimation Tools

- Approaches depending on estimation method:
  - Expert Judgement
  - Algorithmic Models
    - FPs, for size
    - COCOMO, or SLIM, or tables based on size estimation for effort
    - SLIM or tables based on size of effort estimation for schedule
- Most products are hybrids:
  - Organising expert judgement
  - Exploiting machine learning
  - Refining algorithmic models

Estimation Tools *(cont)*

- With estimation tools you can:
  - Estimate size using Function Points or other metrics
  - Derive effort and schedule using various algorithms and techniques
  - Perform “what if” analyses with staffing, duration etc. and appreciate how realistic they are
  - Produce and update Gantt and other charts easily
  - Maintain and exploit a database of historic data
  - Import data from other projects run in organisations with which you have no connection

Estimation Tools *(cont)*

- Tools are important for estimation
  - They make estimation feel like an algorithm
  - They prevent from skipping necessary tasks
  - They help organise, update and archive the results
- Additionally, they provoke you to think about the lifetime of the project and at the same time do the dirty job for you
- Make your own research for tools and their efficiency
  - Compare them against already finished projects
  - Use them in parallel with your current estimation technique

Estimation is a part of Process

- The most important issue about estimation is frame of mind
- It takes more than a good estimate to keep a project in good shape
- Important things to note:
  - The silver bullet syndrome
  - Choice of development model
  - Track the project progress
  - Estimate in ranges
  - Think before you quote
  - Bring in the right people

The silver bullet syndrome

- None of the estimation methods is perfect
- None of the estimation tools is perfect
- Evaluate and calibrate methods and tools

Development model

- Many models are flexible enough to accommodate changes.
- Design to schedule is great for fixed time-fixed cost projects.
- Iterative models are better for projects with many unknowns.
  - Staged delivery
  - Design to schedule
  - Evolutionary delivery
  - Spiral

Keep track of your project

- A software project is a live entity with complex behaviour
- Monitor and update estimates
  - Even after the design stage you can expect to be off by 25%
- Complement your estimates with a good tracking policy
  - Various tools available
- Always update the risks list and the priorities list.

Estimate in ranges

- The tools and methods may give you absolute numbers
- Add some padding, i.e. slack time to make up for errors
  - Do it and the customer does not trust your estimate
  - Don't do it (or don't pad enough) and you may overrun
  - Good practice: estimation *range*
  - Good practice: buffers for *specific* risks
- Risk analyses will show things that can go wrong
- Evaluate their impact to the schedule
- Provide conditional estimates
  - Start with raw numbers (from tools and methods)
  - Consider risks
  - Consider what could help

Estimate in ranges *(cont)*

- Example:
  - “The GUI will take 6 months, +2 if the tool-generated code is useless, +1 if Jo goes on holiday, -1 if we can reuse the ‘File’ menu functionality from project X.

    6+3 -1".
- Alternatively:

  [5-9]
- Very different from “8”
  - 6+2+1-1=8.
- Incorporation of risks to estimate

Think before you quote

- When tempted to provide an estimate thinking on your feet...

DON'T!

- People remember it, pass it on and hold you accountable.
  - Although it is normal for the estimate at the feasibility stage to be off by 400%.

Who Should be involved in Estimation

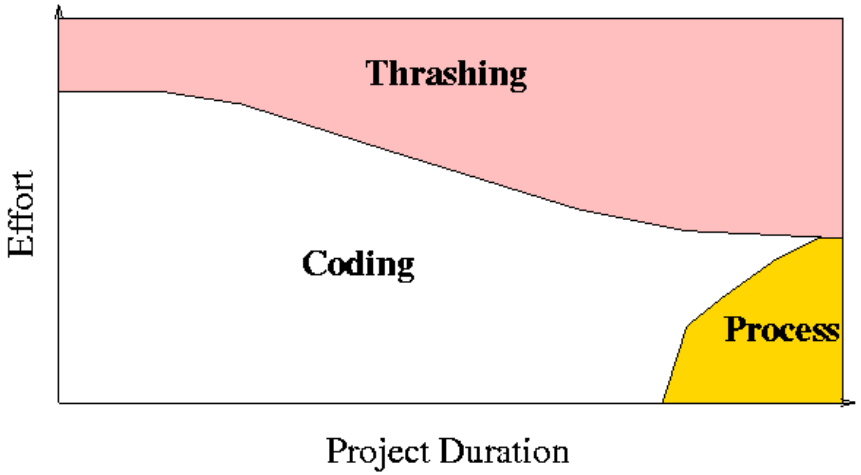
- Who should be doing estimation work?
- Why?

Bring in the right people

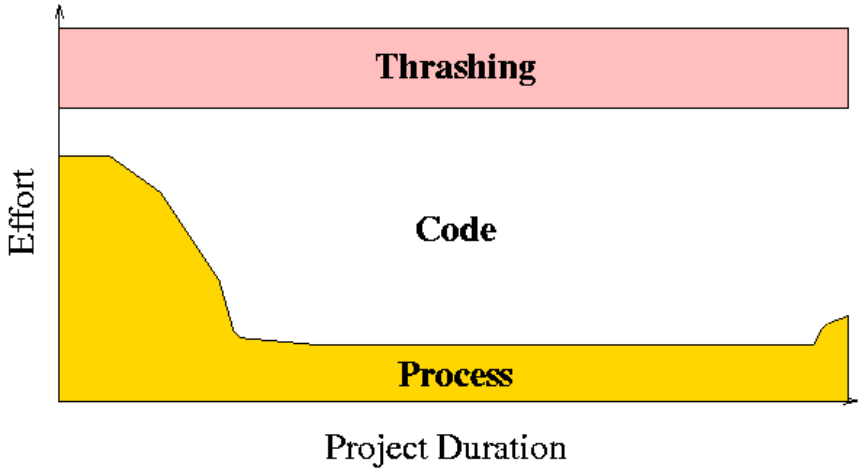
- 2 groups of colleagues one cannot ignore.
- Group 1: those experienced in estimation.
  - They know how to appreciate the workload of components
    - Trust them in conjunction with documented past estimates
  - They know how to do it.
- Group 2: those who will do the work
  - They will know how long it will take them
    - Do not judge the effort from your own standards
  - They can see things from a different angle
    - And foresee the technical challenge
  - They get a feeling of ownership
    - Persistence and application to the project

Recap on Process

"No-Process" Project



"Process-Based" Project



Process: *"A set of practices which serve to eliminate or mitigate the risks which can affect software development"*

Risk Management

- Risk management is the core concept of all Process
  - All Process that we apply acts to reduce risk
  - Risk of disorderly development practices causes
    - Risk of schedule slippage
    - Risk of quality slippage
    - Risk of morale problems in staff
    - etc
  - But there are Process practices which act to specifically identify and handle project risks
    - This is the area of Process known as *Risk Management*

Risk Management Motivation

- "*If you don't actively attack the risks, they will attack you*" - Tom Gilb
- Risk management is the most overlooked element in a poor Process
  - But failure to identify risks and handle them is one of the biggest reasons for project failure
- Why do we let risk rule our projects?
  - Risk analysis is not easily quantifiable
  - We consider risk to be integral part of software development
    - But previously we saw that most risk can be dealt with
    - And risk which cannot be dealt with can at least be planned for

Example: Beagle 2 Mars Probe

- Edward Leigh, Chair of Enquiry Committee:
  - "You probably think we're just boring bean counters but it is public money and we are spending a lot of money on this and frankly the Beagle 2 project failed because, as we said, there was an over-ambitious time-table, there were last-minute technical changes, there was uncertain funding, [and] there was poor risk management.”
  - Poor risk management left the project with "no real prospect of success".

Risk Management Activities

- Two principal activities in Risk Management
- *Risk assessment*, which is comprised of
  - Risk identification
  - Risk analysis
  - Risk prioritisation
- *Risk control*, which is comprised of
  - Risk management planning
  - Risk resolution
  - Risk monitoring

Risk Assessment

- The goal of this practice is to produce a risk list
  - This is a prioritised list of risks which combines:
    - The chance of the risk occurring
    - The impact on the schedule should it occur
- How?
  - A simple but effective method is to take the impact on the schedule of the risk and multiply it by the probability of the risk occurring
    - This will give us an estimate of the time aspect of the risk
    - Sometimes, it is more useful to use the financial impact, thus measuring the cost aspect of the risk
    - Quality aspects are usually harder to quantify easily

Quantitative Risk Assessment

- Risk 1: Users will want documentation screen shots in colour
  - Chance of risk occurring : 80%
  - Impact on schedule : 5 days
- Importance of risk 1 = 5 * 80/100 = 4 risk units

- Risk 2: Graphics library might not have all necessary functionality
  - Chance of risk occurring : 10%
  - Impact on schedule : 40 days
- Importance of risk 2 = 40 * 10/100 = 4 risk units

Risk Assessment

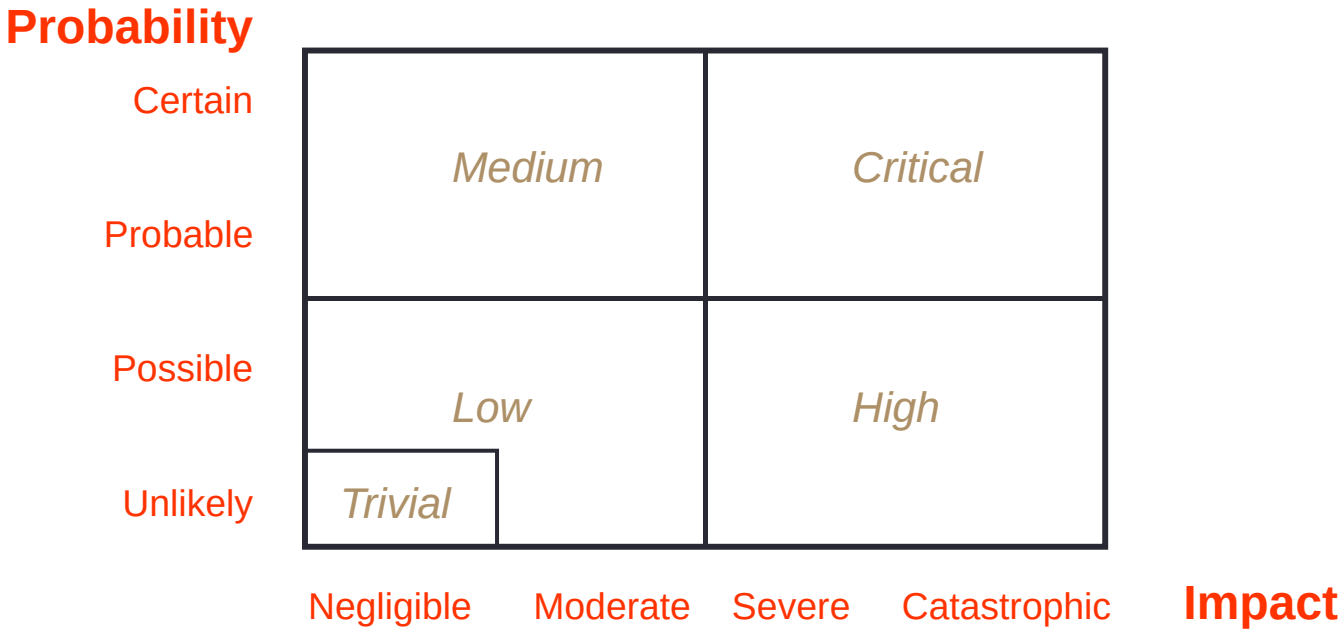
- We can see the two risks pose the same level of threat to the project
  - But how do we determine the estimates?
  - There are several methods but none are foolproof
- Sometimes, it is difficult to use a quantitative approach to risk assessment, due to
  - Uncertainties in estimates
  - Difficulties in finding appropriate common “units” to measure the relative impact of the risks
- Can take a more *qualitative* approach

Qualitative Risk Assessment

- Rather than give precise numbers to the probability and impact, it may sometimes be more practical simply to categorise these

- Example categories:
  - Probability: *Unlikely, Possible, Probable, Certain*
  - Impact: *Negligible, Moderate, Severe, Catastrophic*

Qualitative Risk Assessment



- The importance/risk exposure of the risks can then be assessed using a diagram like the one above
- Prioritised risk list contains *Critical* risks, then *High* importance, *Medium* and finally *Low*
  - *High impact-low probability risks are deemed of higher importance than those of high probability-low impact*

Risk Assessment

- The biggest chance for success in any risk assessment lies simply in the project team taking risk management seriously

- The most important part of risk assessment is trying to identify and record *all* the risks

- So, we produce our prioritised risk list, but what do we do with it?

Risk Control

- Without controlling the risks we have assessed, we are wasting our time assessing them
  - The problem is that what is good for one risk is probably wholly inapplicable to another
    - Developers losing productivity due to poor office environments will benefit from window blinds and better lighting
    - That won't help the gap in functionality of the graphics package...
  - Each risk can also probably be solved in several different ways
    - New graphics package
    - Employees go on a course
    - Contact developers of the graphics package and get answers

Task

- In your groups, come up with common risks that you would encounter on software projects
- Can these be placed roughly into the Probability/Impact diagram?
- What other method could you use?

Some Common Risks

- Here is a list of common project risks
  - Design deviation
  - Feature creep / Gold-plating
  - Short-changed quality
  - Optimistic scheduling
  - Inadequate design
  - Silver bullet syndrome
  - Research oriented development
  - Resource / facilities shortage
  - External developer failure
  - Communication breakdown

Risk Control

- Some of these risks seem pretty obvious
  - But the process of identifying risks and finding solutions is very important
    - It might be obvious that if there is a chance of a gap in the graphics package functionality you should consult its developers
    - Its better to determine and solve this question now rather than wait for the answer on the critical path
    - The earlier you can control a risk, the less of a risk it becomes
- So, how can we resolve risks on our project?
  - Different risks require different solutions
  - But there are generic methods of addressing risks

Risk Handling Methods

- Some methods for handling risks
  - *Avoid* the risk
    - e.g. Offset risky design area to team with more experience
  - *Transfer* the risk from one part of a system to another
    - e.g. Getting risk off critical path so it is less of a risk
  - *Buy* information about the risk
    - e.g. Give team time to research graphics library for functionality
  - *Eliminate* the root cause of the risk
    - e.g. Remove risky functionality from current project version and treat it as a research project

Risk Handling Methods

- *Assume* the risk
  - e.g. Understand the risk can happen and be prepared to take the hit if it does (small risks only)
- *Publicise* the risk
  - e.g. Inform customers/upper management that the risk exists to minimise surprise if it occurs
- *Control* the risk
  - e.g. Alter schedule and resource allocation to accommodate the risk and lessen its effect
- *Remember* the risk
  - e.g. Document all risks which affect the project for reference in further projects

Task

- For the list of common risks, what control methods could you use?

- Discuss projects you have been involved in – and discuss specific risks and what controls were applied

Risk Management Timing

- When do we apply risk management?
  - Risks can come in any form at any time
  - Risk management should follow the full course of the project
  - Risks corrected earlier are done at less cost though
  - Risk management can easily be integrated into development models

- Above all, keep your risk list up to date
  - Risks come and go as a project progresses
  - Risk monitoring
    - check progress towards resolving each risk
    - identify new risks and add to the risk list
  - Reassessing your risk list will alert you to future problems

Summary

- Risk management is the core concept in Process
  - ...and therefore critical for a successful project
- Risk assessment
  - Identify and prioritise risks

Maintain your risk list!

- Risk control
  - Eliminate risks when possible
  - Mitigate risks which cannot be eliminated
  - Monitor risks throughout the course of a project

Don't stick your head in the sand and hope that the risks will go away!