

Today's Lecture

- Software Lifecycle Models

Software Lifecycle Models

- Summary

- A software lifecycle model is a delineation of the steps that transform a customer need into a finished product.
- They are at the heart of the software product management process and contain the various components of the development project:
 - Phases - The steps in the lifecycle that mark the project's progress (e.g. the initiation phase, the development phase, etc.)
 - Activities - The actions required to deliver the products (e.g. requirements, design, code, etc.)
 - Deliverables - The products created during the project (e.g. the finished product, documentation, installation software, etc.)
 - Milestones - Important events in the lifecycle (e.g. phase hand-over dates, delivery dates, etc.)
- There is no one correct lifecycle model.

Software Lifecycle Models

- Why are they important?

- Choosing the right model is critical as the chosen model becomes the management tool and provides a common view of how the work will be done. They provide the means to:
 - Establish control points.
 - Plan and track progress.
 - Plan and track budget.
 - Assess status.
 - Manage risk.
- Choosing the wrong model can lead to:
 - Ineffective management of projects and added risk.
 - Poor team relationships and performance.
 - Inefficient process performance, increased cost, late delivery

Software Lifecycle Models

- How did they originate?

- Many of the early lifecycle models were developed during the 1970's they took their lead from the management techniques that were employed on construction projects and applied them to software.
- Compilation of code was at a premium and hence the models that were developed back then reflected this. The models were very much dependency driven - just like construction projects.
- Today the emphasis is much more on the requirements and design phases and the risk of delivering the wrong product. These models are much more content driven - unlike construction projects.

Software Lifecycle Models

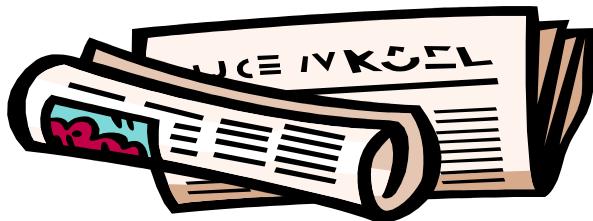
- An Analogy



- If we were building a bridge the lifecycle would be dependency driven:
 - The detailed requirements would be well understood.
 - The bases for the towers would have to be built before the towers could be erected.
 - The cables strung between the towers before the suspension cables are hung.
 - The main bridge sections added before utilities are incorporated and so on..
 - One activity doesn't start until the previous one is complete and reviewed.

Software Lifecycle Models

- Another Analogy



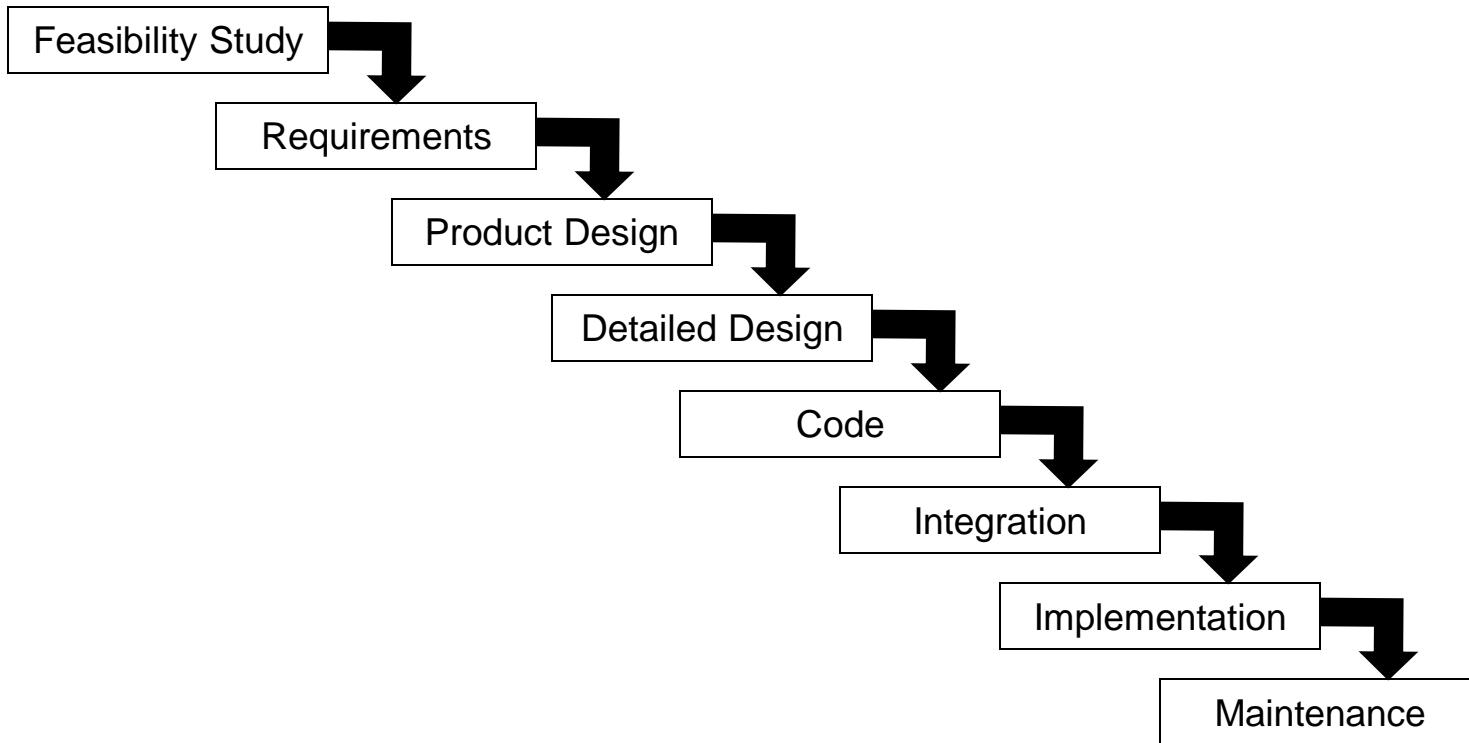
- If we were publishing a daily newspaper the lifecycle would be content driven:
 - The detailed requirements would be discovered throughout the lifecycle.
 - Events jostle to grab the headlines and priorities change.
 - Content has to be managed: veracity, 'pulling' articles, libel, etc.
 - Selling is totally focussed on 'This is what the punter wants to read about'
 - There are no dependencies events are asynchronous.
 - Shorter 'time to market' lifecycles.

Software Lifecycle Models

- Some software lifecycle models:
 - The Waterfall model
 - The ‘V’ model
 - The Spiral model
 - Agile Approaches:
 - XP
 - SCRUM
 - CRYSTAL
 - DSDM
 - ASD
 - FDD

Software Lifecycle Models

- The Waterfall model:



- Introduced by Royce in the 1970's as a response to the 'code n'fix' mentality of the 1960's - and was widely adopted by the software industry.

Software Lifecycle Models

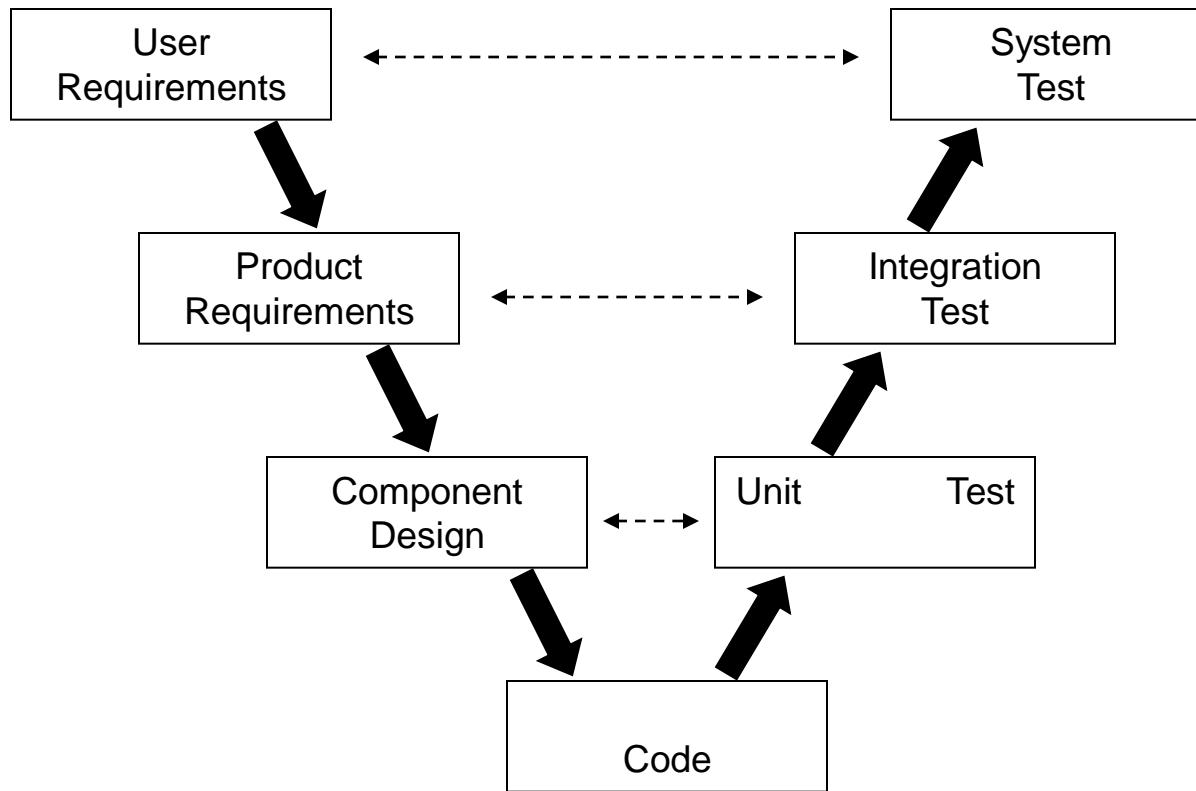
- **Feasibility Study:** Determining the design and development approach.
- **Requirements:** Development of a complete specification for the product.
- **Product Design:** Bridging the space between customer needs (as expressed in the Requirements) and the physical expression of the product. (High level).
- **Detailed Design:** Development of a complete specification (control, data structures) of the modules. (Low level).
- **Code:** Coding and unit testing of each module.
- **Integration:** Compiling and linking the various modules to produce a fully functional software product.
- **Implementation:** Installing & integrating the software into the operational environment.
- **Maintenance:** Software updates.

Software Lifecycle Models

- The goal is to fully complete each phase before moving on to the next phase. In practice a certain amount of iteration between phases occurs.
- Integration occurs as a ‘big bang’.
- Risk is not considered as a factor.
- The model is simple to understand and staff.
- Milestones are clear and there is clear division of staff involvement.
- Easily integrates with the standard formalism of project management.
- Does not truly reflect how teams do work.
- Encourages an ‘over-the-wall’ approach.
- Cost of application is low but the cost of upgrades is high.
- You can’t climb up a waterfall.

Software Lifecycle Models

- The 'V' model:



Software Lifecycle Models

- **User Requirements:** Determining the functionality required of the system by the user.
- **Product Requirements:** Bridging the space between customer needs (as expressed in the User Requirements) and the physical expression of the product. (High level).
- **Component Design:** Development of a complete specification (control, data structures) of the modules. (Low level).
- **Code:** Coding of each module.
- **Unit Test:** Testing the functionality of each module as specified in the Component Design.
- **Integration Test:** Testing functionality of the software product as specified by the Product Requirements
- **System Test:** Testing the complete functionality of the system as a whole as specified by the User Requirements

Software Lifecycle Models

- The left hand side of the V-model is concerned with the product resulting from the development activities (i.e. the code) while the right hand side is concerned with verifying that the needs of each development phase are met.
- Each verification phase is produced from the needs of, and linked to a corresponding development phase.
- Risk is not considered as a factor.
- The model is simple to understand but less simple to staff.
- Integration is gradual with distinct test phases.
- Does not truly reflect how teams do work.
- Cost of application is low but the cost of upgrades is high.
- Widely used and well known model to the majority of software organisations.

Software Lifecycle Models

- Contrast with Agile approaches
 - Extreme Programming (XP)
 - SCRUM
 - CRYSTAL
 - Dynamic Systems Development Method (DSDM)
 - Adaptive Software Development (ASD)
 - Feature Driven Development (FDD)

Software Lifecycle Models

- Next Lecture:
 - Detailed examination of Agile methods and Extreme Programming (XP)

Today's Lecture

Agile Software Development

Why Agile?

- Traditional methodologies failing to deliver
 - Too mechanistic
 - Top heavy with documentation or other non-value added tasks
- Commentators suggest traditional IS development methodologies are:
 - ‘treated primarily as a necessary fiction to present an image of control or to provide a symbolic status’
 - ‘merely unattainable ideals and hypothetical “straw men” that provide normative guidance to utopian development situations’

Why Agile?

- On February 11-13, 2001 representatives from Extreme Programming, SCRUM, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic Programming, and others sympathetic to the need for an alternative to documentation driven, heavyweight software development processes convened. They called themselves the Agile Alliance and produced a *Manifesto for Agile Software Development*

The Agile Software Development Manifesto

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over rigid contracts

Responding to change over following a plan

“...while there is value in the items on
the right, we value the items on the left more”

People over Processes

- The human role and human interaction is more important than the company's institutionalized processes and tools
 - Close team relationships
 - Close working environment
 - Boosting team spirit

Working Software

- Continuous output of working, tested software is the vital objective of the software team. Hence:
 - Frequent releases
 - Keep code simple
 - The less documentation the better

Relationship with Customer

- Co-operation with the customer over strict contracts
- Deliver software quickly so customers can see it
- Reduce the risk of not meeting terms of contract

Responding to Change

- Contracts must be worded so as to allow change
- The team must be authorized to make changes
- Team must be competent to make changes
- Customers are part of team so they are aware of, and sanction, changes

Summary

- To sum up what makes a method an agile one
 - Incremental
 - Cooperative
 - Straightforward
 - Adaptive

The Agile Software Development Manifesto

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over rigid contracts

Responding to change over following a plan

“...while there is value in the items on
the right, we value the items on the left more”

Principles Behind the Agile Manifesto

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Principles Behind the Agile Manifesto

- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Agile Methodologies

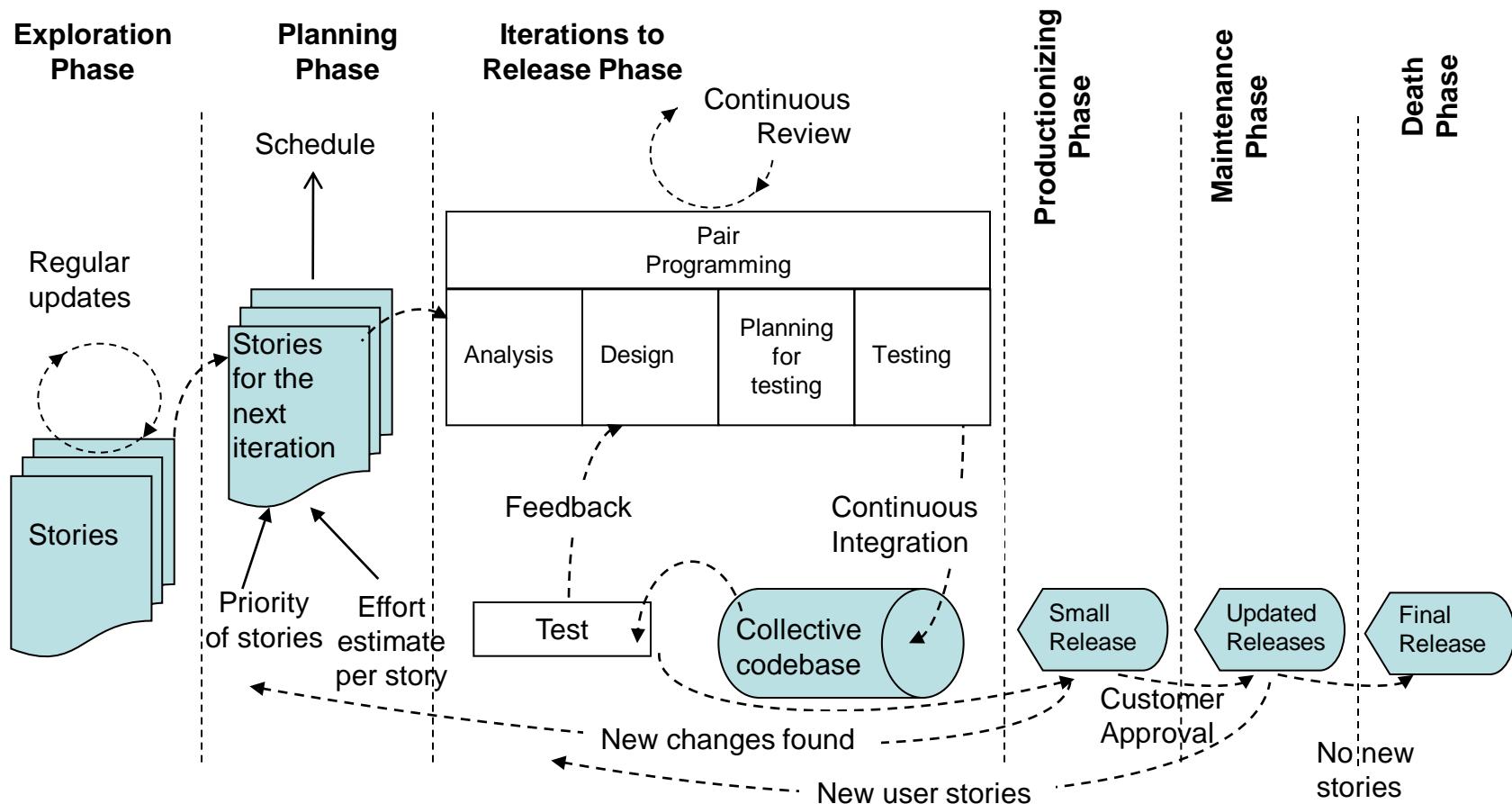
1. Extreme Programming

Extreme Programming

- Kent Beck 1999 ‘Embracing Change with Extreme Programming’, IEEE Computer 32(10)
- Also known as XP
- Enable successful software development with constantly changing requirements
- Not new techniques, more a formalization of existing techniques
- The name “Extreme” comes from taking these existing techniques to extreme levels

XP Process Lifecycle

The XP lifecycle consists of 5 phases: Exploration, Planning, Iterations to Release, Productionizing, Maintenance and Death



XP Exploration

- Customers write story cards for first release
- Familiarization with tools and techniques by team
- Initial prototype to test the technology to be used & explore system architecture possibilities
- Timescale: A few weeks to a few months

XP Exploration

- **Exercise:**
 - Form into groups and write a user story to describe how the computer system that controls the lift in the O'Rahilly Building should operate.

XP Planning

- Prioritise stories
- Agree content of first release
- Effort estimate per story
- Schedule agreed
- Time scale : A few days

XP Iterations to Release

- Schedule broken into iterations
- Timescale of each iteration should be 1 to 4 weeks
- Several iterations before first release
- First iteration should have the architecture in place for whole system
- Customer decides stories for each iteration
- Tests created by customer are run after each iteration
- After last iteration the system is ready for production

XP Productionizing

- After final iteration we move to this phase
- Extra testing & checking before release to customer
- New changes may be found & a decision has to be made if changes are included in this release
- Iterations are shortened during this phase - 1 to 3 weeks
- Postponed ideas & suggestions are documented for later implementation e.g. during Maintenance Phase

XP Maintenance and Death

- Development slows down while maintenance increases
- Additional ideas incorporated
- May need new team members
- Death phase arrives when customer has no new stories
- Documentation finally written

XP - Who Does What?

- Programmer: Coding
- Customer: Write stories & tests
- Tester: Run functional tests
- Tracker: tracks estimates
- Coach: Responsible for process as a whole
- Consultant: External member possessing the required technical knowledge
- Manager: Makes the big decisions

XP 12 Core Practices

- Keep it simple
- Pair programming
- Collective ownership
- 40 hour week
- On-site customer
- Coding standards
- Small releases
- Rules can change
- Metaphor
- Testing
- Refactoring
- Planning game

Summary

- Agile came about as a reaction to documentation intensive heavyweight processes
- Agile Manifesto & Principles
- XP most popular agile methodology

Today's Lecture

Agile Software Development

SCRUM & Feature Driven
Development (FDD)

Scrum?

- Traceable back to 1986 Hirotaka Takeuchi and Ikujiro Nonaka popularised by Ken Schwaber
- Central thesis: Systems development is highly dynamic.
 - Since systems development involves several environmental & technical variables that are likely to **change** during the process, the development process is **unpredictable & complex** and therefore the approach needs to be **flexible**.

Scrum?

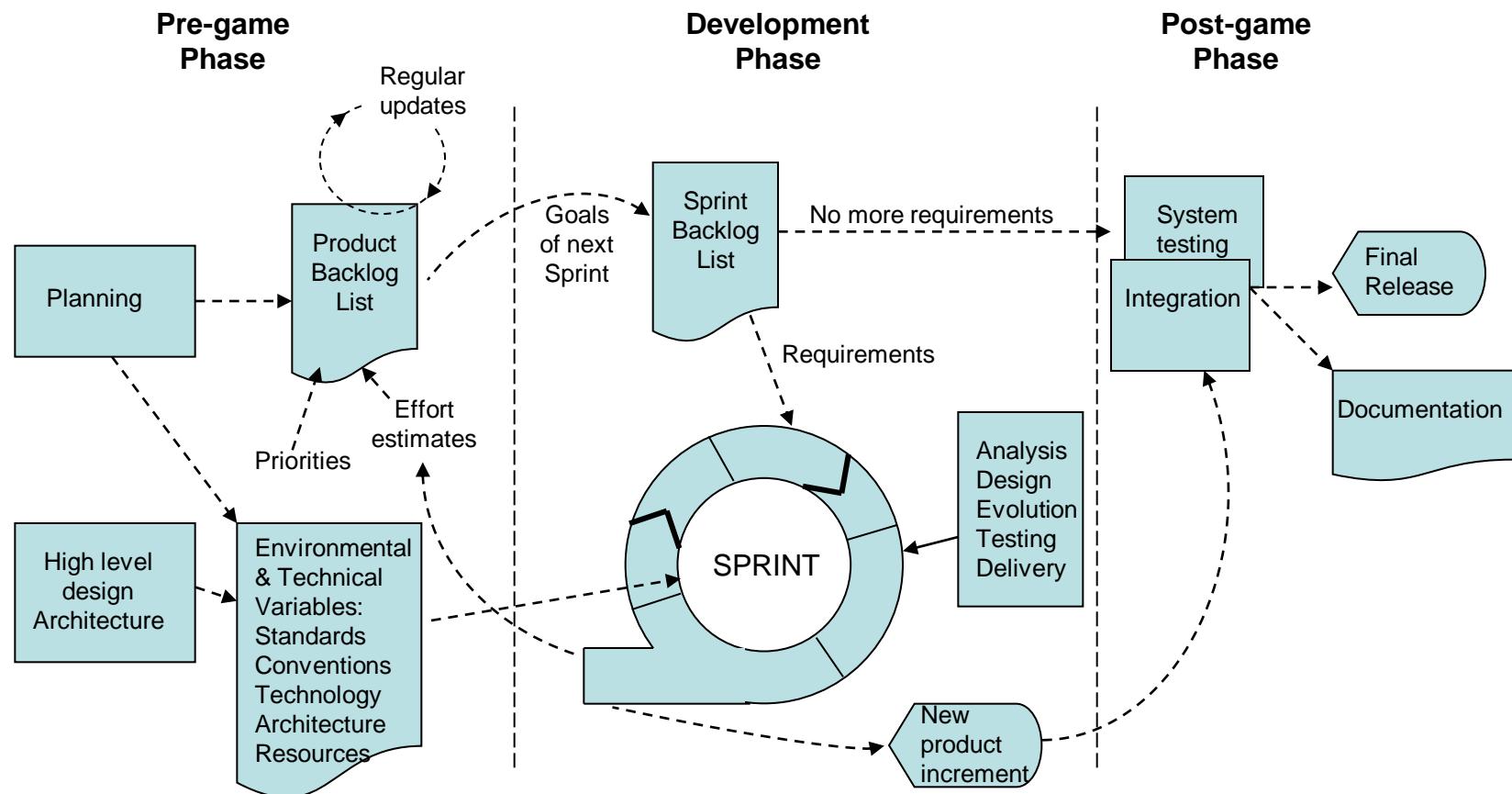
- Empirical approach which concentrates on how team members should function to produce a system flexibly in a constant changing environment.
- Involves frequent project management activities aimed at improving the development process & practices.
- In rugby the scrum is a way of restarting the game after an interruption.
- The Scrum methodology uses rugby as an analogy

Scrum



Scrum Process

- There are 3 phases:
 - Pre-game
 - Development
 - Post-game



Pre-Game Phase

- Incorporates two sub-phases: Planning & High Level Design Architecture
- In planning, a product backlog list is created containing all the known requirements
- Requirements originate with stakeholders
- Requirements are given a priority and an effort estimate
- Requirements are stored in a Product Backlog List

Pre-Game Phase

- Backlog is constantly updated as new information becomes available
- In the architecture, the High Level design is planned using the product backlog list.
- Initial plans for the contents of the releases are created

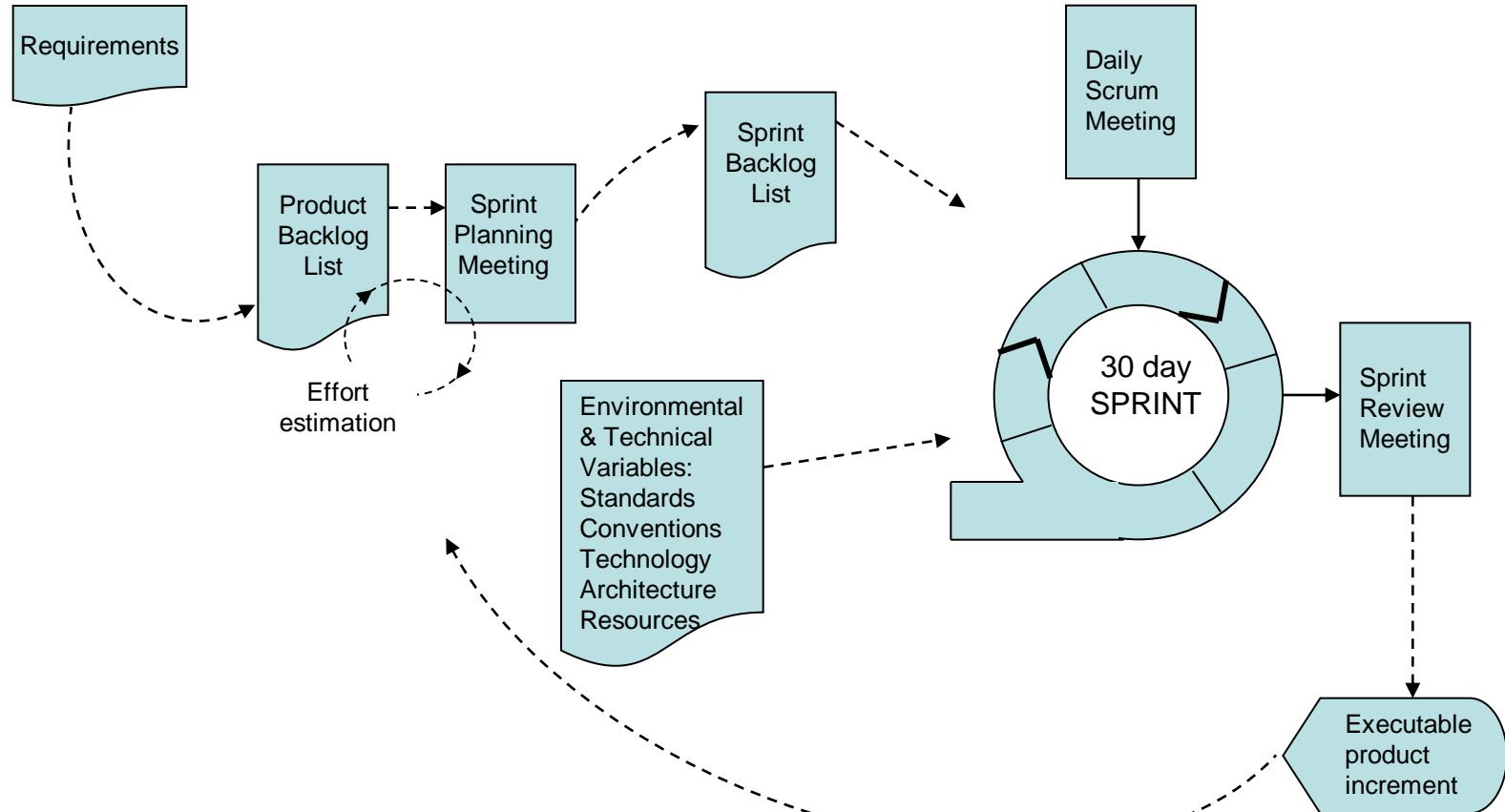
Development phase

- This is the agile part of the methodology
- Changes in environmental & technical variables are expected in this phase
- Sprints are at the core

Development phase

- Each sprint is like a self contained life cycle with requirements, analysis, design, development, delivery.
- A sprint iteration lasts from 1 week to 1 month
- The design evolves during this phase.

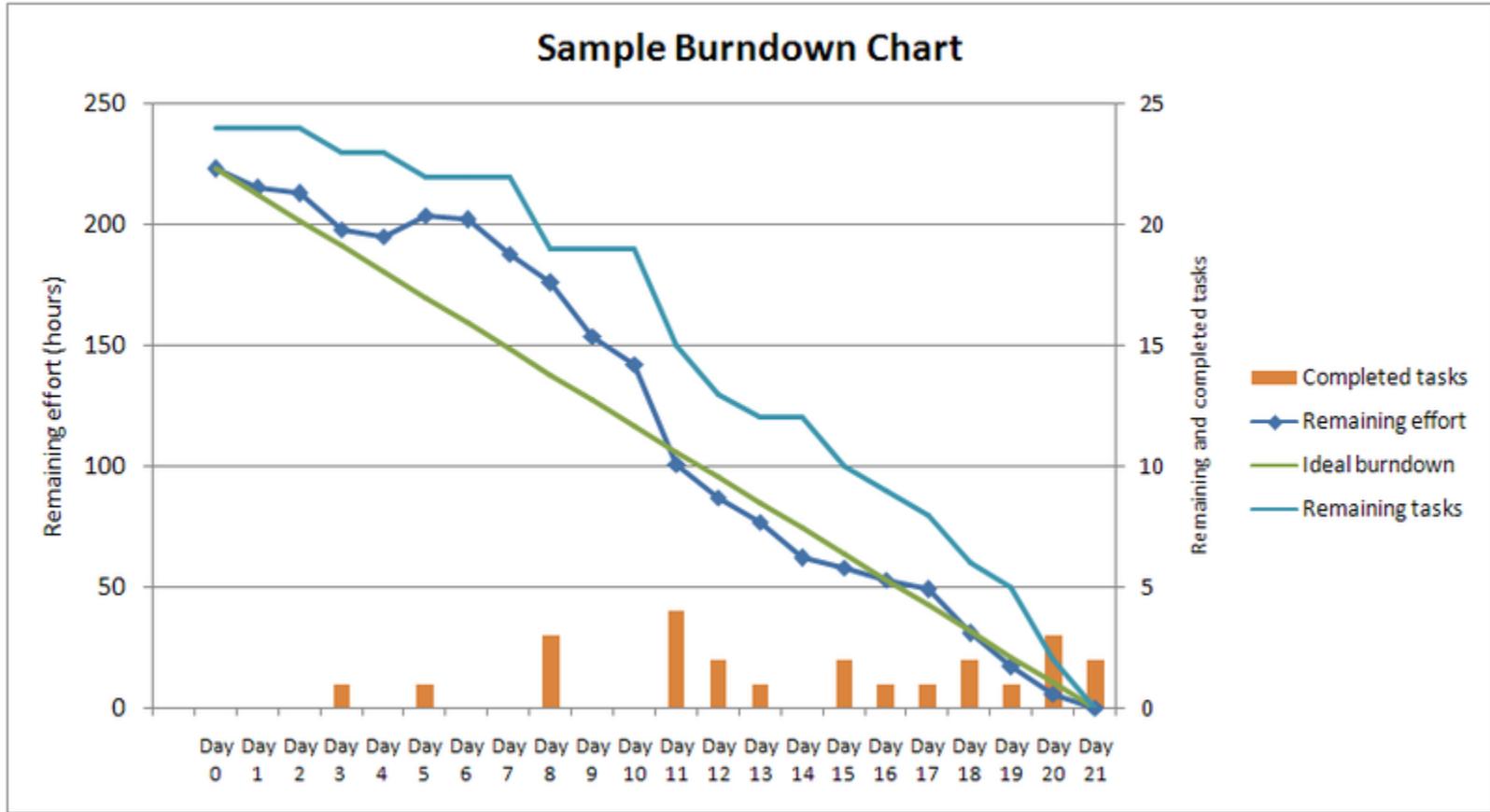
Sprints Illustrated: Practices & Inputs



Sprints

- Sprint Backlog: A list of product backlog items to be addressed in this sprint
- Sprint Planning Meeting: Start of each sprint.
Decide on goals and requirements for the sprint.
Decide on how to implement the sprint.
- Daily Scrum Meeting: Progress tracking.
- Sprint Review Meeting: End of Sprint. Results presented.

Scrum Project Tracking



Produced for each sprint - updated daily and publicly displayed

Post-Game Phase

- This phase starts when all requirements are met
- No more requirements or issues can be found.
- This phase involves integration, system testing and documentation

Scrum – who does what

- Scrum Master: Ensures project follows Scrum rules and values & progresses as planned
- Product Owner: Officially responsible for project – akin to project manager
- Scrum Team: Authority to organise itself & to devise actions to achieve goals of each sprint
- Customer: Involved in creating product backlog list
- Management: Final decision making

Agile Software Development

Scrum & FDD

FDD – Feature Driven Development

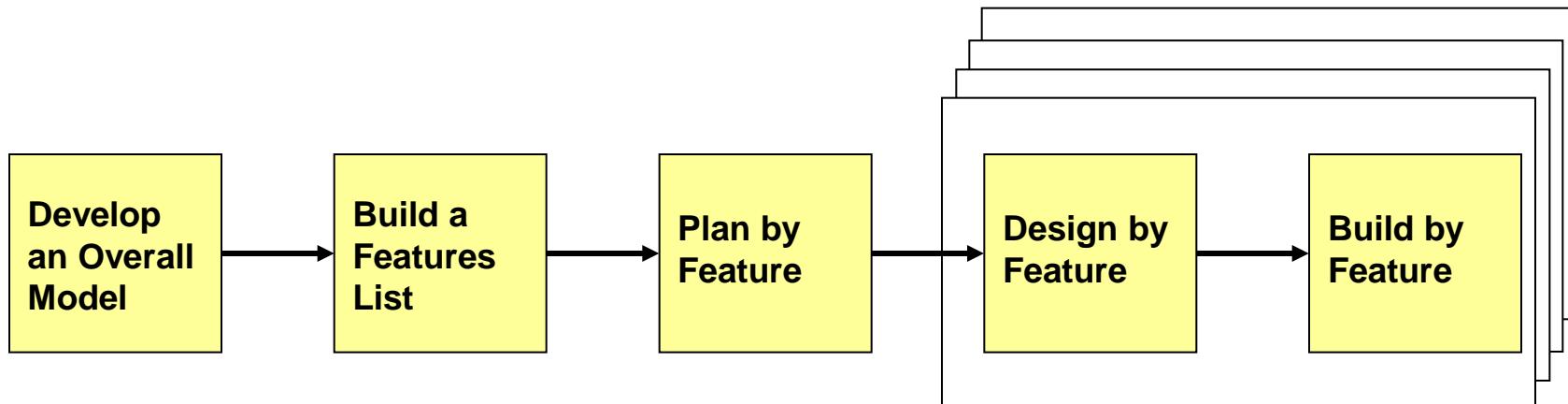
FDD

- Driven from a client-valued functionality perspective
- Focuses on Design and Building Phases and will work with other project activities
- Embodies iterative development
- Emphasis on:
 - Quality
 - Frequent deliveries
 - Monitoring of progress

FDD

- Consists of 5 sequential processes providing methods, techniques and guidelines needed by stakeholders
- Claims to be suitable for critical systems

FDD Process Model



Processes of FDD (Palmer & Felsing 2002)

1: Develop a Model

- Requirements (and associated documents), context and scope are already known & available
- A high level design of the system is described to the project team by domain experts
- Overall domain is further sub-divided into smaller domains and assigned to small teams

1: Develop a Model

- Each sub-domain has its own High Level Design (HLD) described to the team
- The teams decide on an object model for each domain area
- Overall object model is produced from these

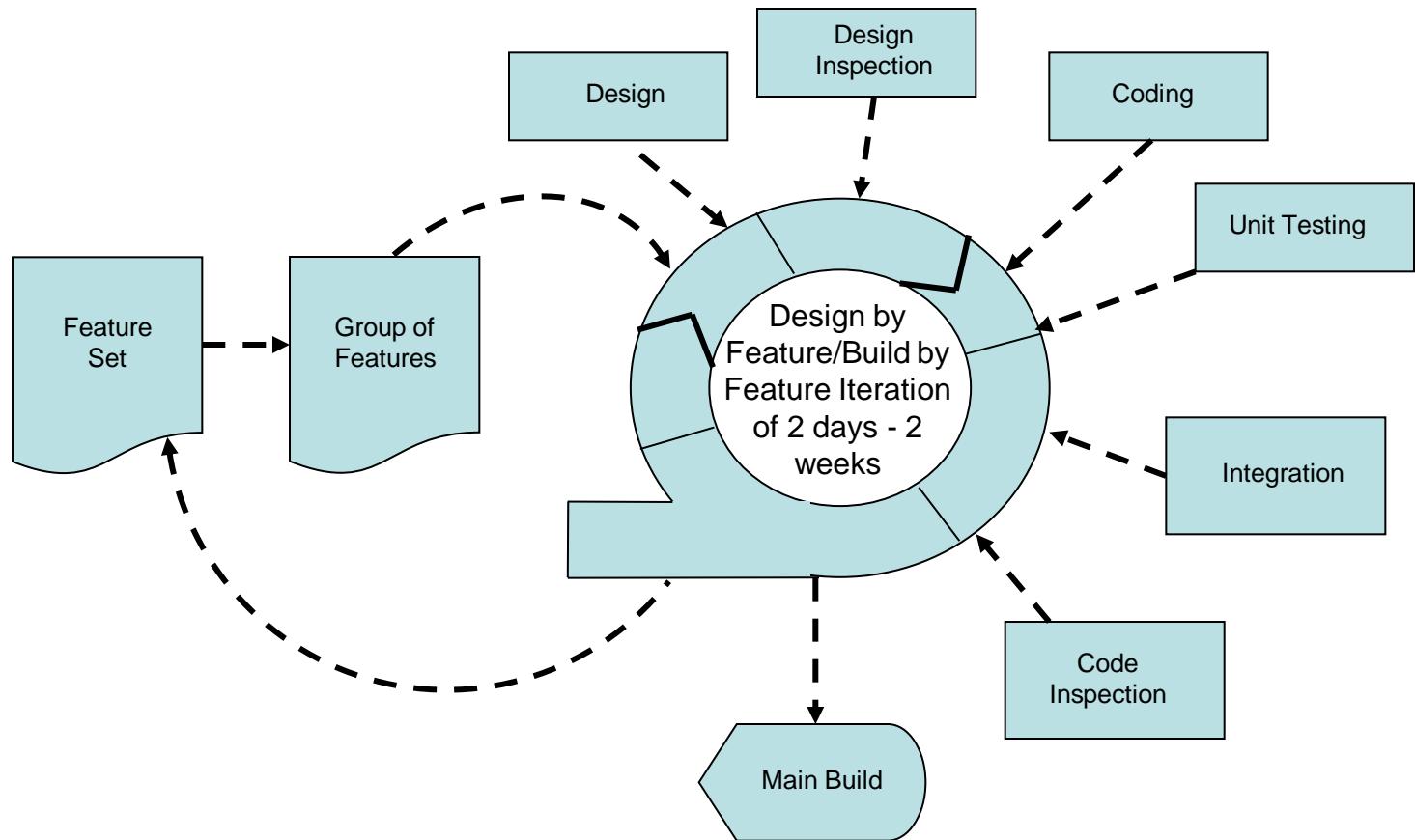
2: Build a features list

- Output from stage one is used to develop a features list.
- The features are “client valued functions” included in the system
- Users and sponsors review feature lists to ensure everything is covered

3: Plan by feature

- Create a High Level Plan
- Features are sequenced according to their priority
- Features are assigned to chief programmers.
- Classes identified assigned to developers (class owners)
- Schedules & milestones created for features (or sets of features)

Design & Build by Feature - Process



4 & 5 Design and Build by Feature

- Small group of features is selected & feature teams are formed
- Iterations take a few days to a max of two weeks
- Iterative approach on a feature basis
 - Design
 - Design inspection
 - Coding
 - Unit testing
 - Integration
 - Code inspection
 - Main build

4 & 5 Design and Build by Feature

- Iterations take a few days to a max of two weeks
- Multiple teams working concurrently

Key roles

- Project Manager: Ultimate authority on scope, schedule, and resourcing
- Chief architect: Overall design and presents HLD to teams
- Development Manager: Owns daily development work

Key roles

- Chief Programmer: Technical lead of teams
- Class Owner: Responsible for development of assigned class
- Domain Expert: Knows the requirements and what's expected

Supporting roles

- Release Manager: Reports progress to project manager
- Language Guru: Expert in a programming language or new technology
- Build Engineer: Maintains the build process
- Toolsmith: Builds small tools for development

Supporting roles

- Systems Administrator: Manages Servers etc.
- Tester: Can be independent of team
- Deployer: Can be independent of team
- Technical Writer: Can be independent of team

Next Week

- Conclude Agile

Today's Lecture

Agile Software Development

DSDM, & ASD

**DSDM - Dynamic Systems
Development Method**

Previously

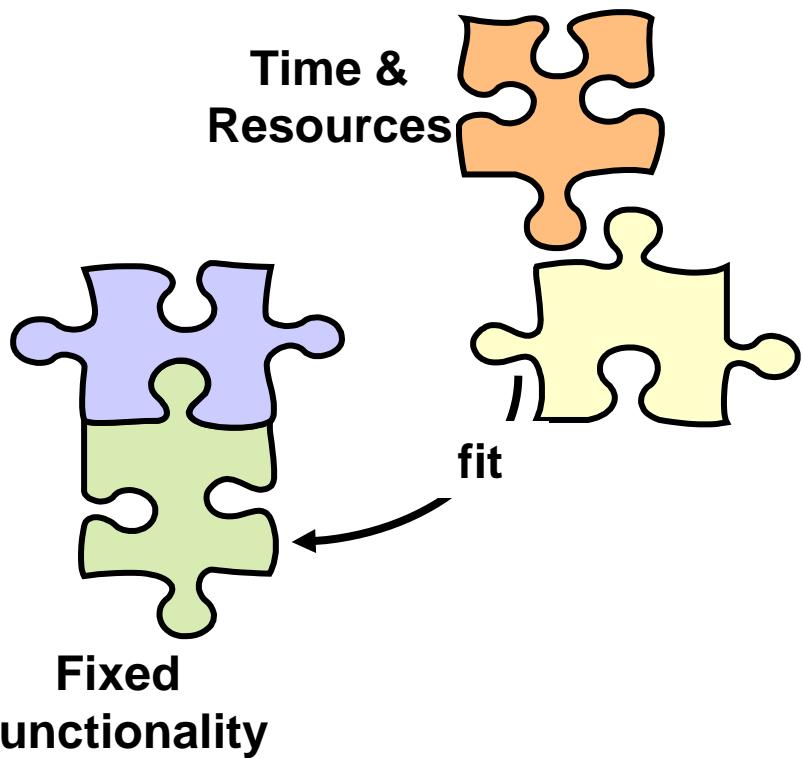
- Looked in detail at Extreme Programming (XP) - most popular Agile approach
- Compared and contrasted with the older dependency driven models
- Looked in detail at 2 other Agile methods
 - SCRUM - greater focus on teaming and flexibility
 - FDD - Feature Driven Development - with a focus on the design & building phases of development

DSDM

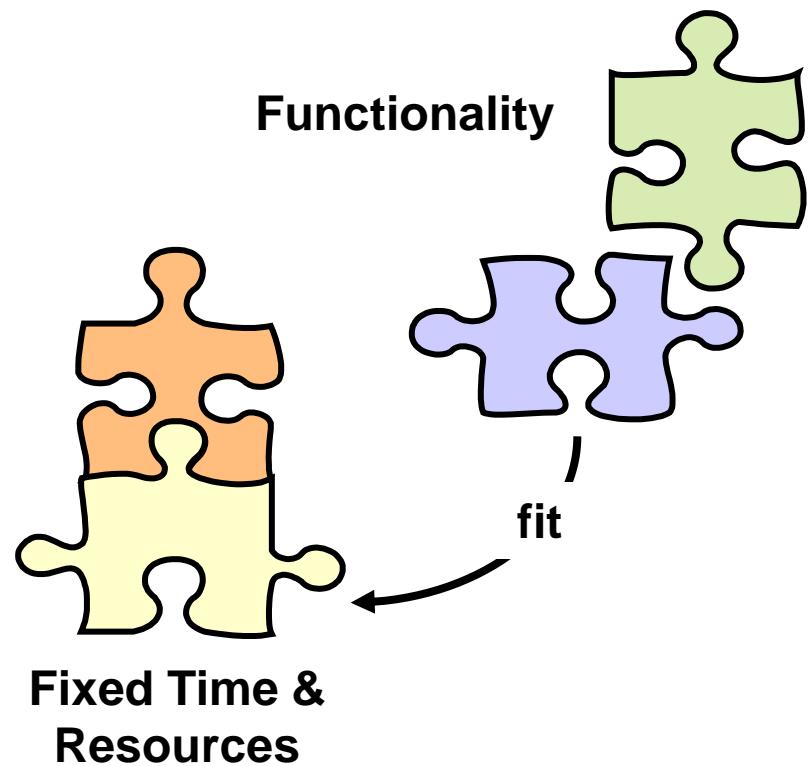
- Dynamic Systems Development Method (DSDM)
Originated in 1994 most recent version 2007.
- The DSDM consortium maintain it
- DSDM focuses on Information Systems projects
that are characterized by tight schedules and
budgets
- Uses MoSCoW for prioritisation of scope
(Must's, Should's, Could's, and Wont haves)

DSDM Fundamental Idea

Traditional Approach



DSDM



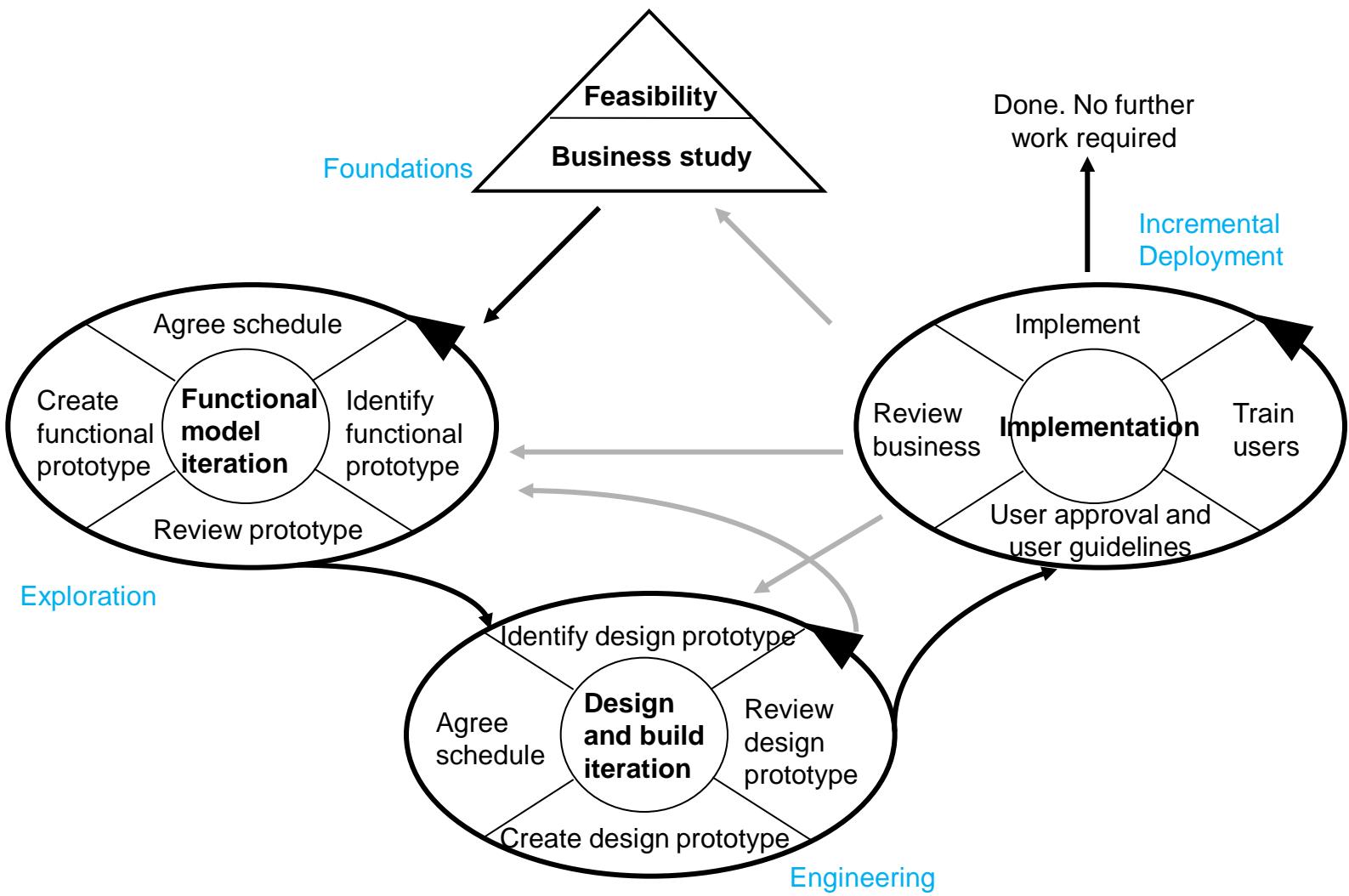
DSDM Phases

- DSDM has pre-project and post-project components but the substantive component is the project lifecycle embedded within the model

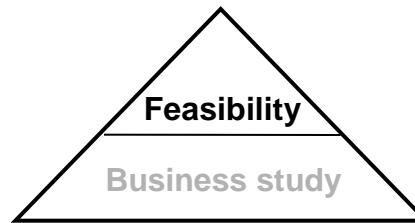
DSDM Phases

- There are five phases in DSDM project lifecycle
 - Feasibility study
 - Business study
 - Functional model iteration
 - Design and build iteration
 - Implementation
- First two are done once sequentially and only once
- The last three are iterative and incremental

DSDM Process

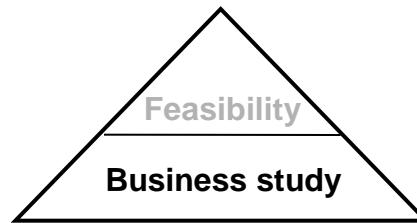


Feasibility Study Phase



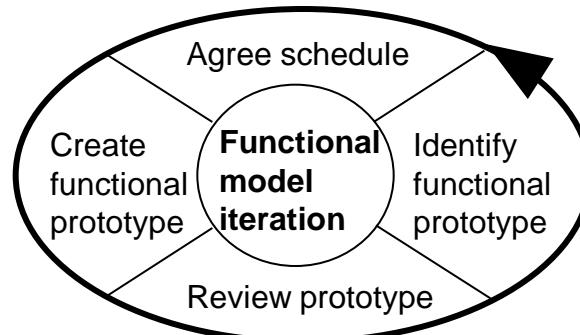
- The decision is made whether or not to use DSDM
- Is it possible to do this project?
- What are the risks?
- Two outputs:
 - Feasibility report
 - Outline of development plan

Business Study Phase



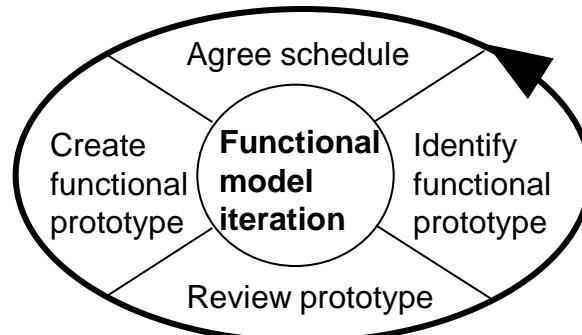
- Recalling that we want to fit project needs to limited time & resources - workshops are organised to consider and prioritise project needs
- 3 outputs
 - High Level Design of processes to be followed
 - System Architecture Definition
 - Outline Prototype Plan

Functional Model Iteration Phase



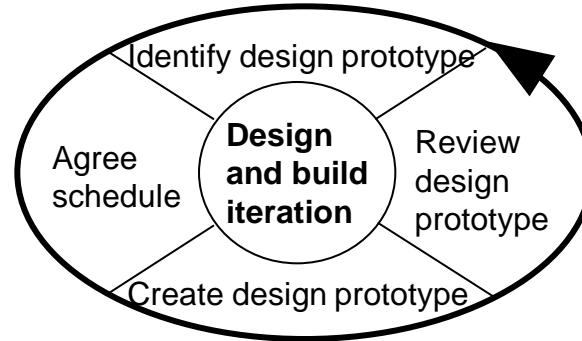
- Identify a functional prototype: Analyse the requirements, List requirements of current iteration, List the non-functional requirements, Create the functional model
- Agree the schedule: Determine the time required for the prototype and agree with the team and customer
- Create the Prototype: based on functional prototype
- Review the prototype: Includes testing the prototype and integration with other functions

Functional Model Iteration Phase



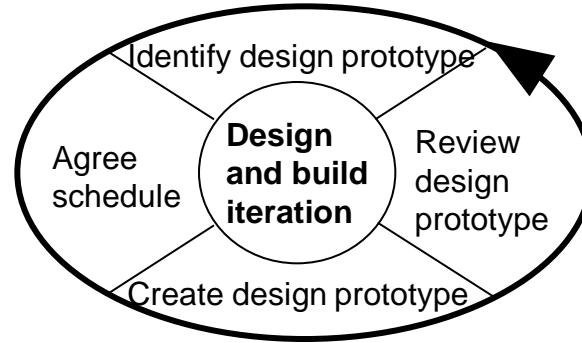
- Outputs from the Functional Model Iteration Phase
 - A functional model
 - Prototype code
 - Analysis models
 - Prioritized functions
 - Prototype review documents
 - Non functional requirements identified
 - Risk analysis performed

Design and Build Phase



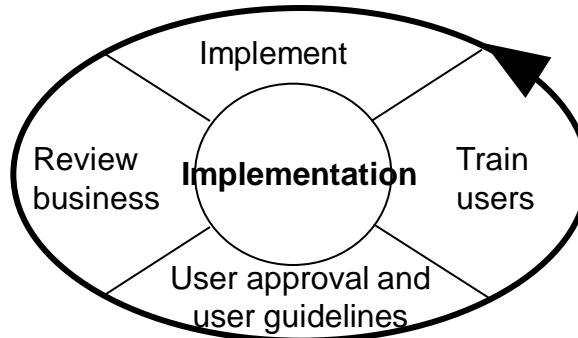
- System is mainly built but the functional components from the earlier phase are now integrated to produce a complete system
- Identifying the functional and non-functional requirements that need to be in the system
- Agree the schedule
- Produce the system: Integrate system components
- Review the system: System testing and review

Design and Build Phase



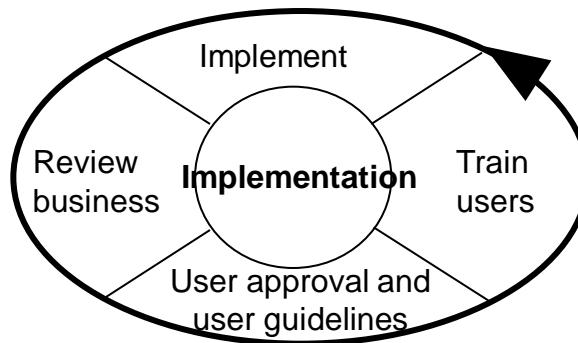
- Output is a tested system plus user documentation
- Functioning prototypes are reviewed by customers and further requirements may be found (and will result in further iterations)

Implementation Phase



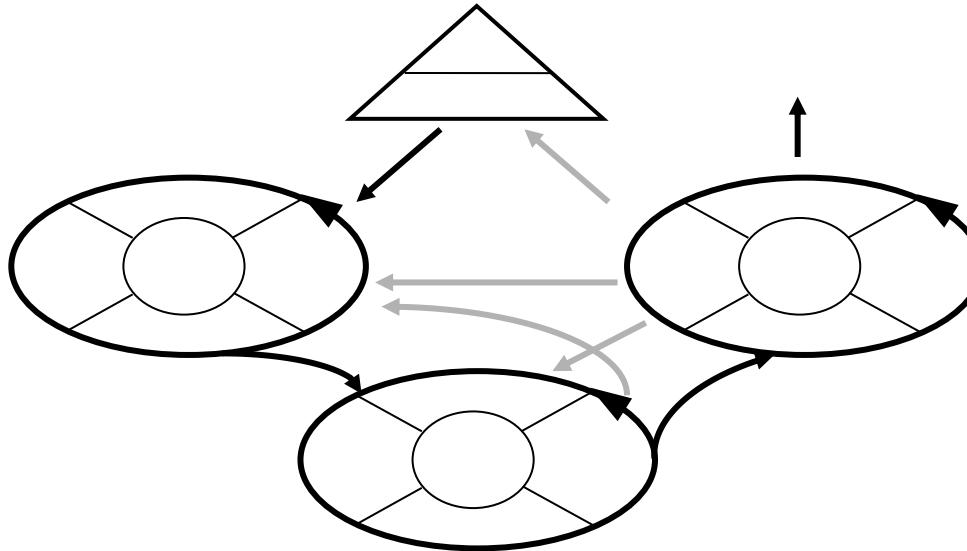
- System moves from development to production
- User training is done
- System implemented at customer site
- Handover to customers
- Review held to ensure system meets business needs

Implementation Phase



- The phase may require several iterations if a wide number of users & if implementation is done over an extended period of time
- Outputs
 - Delivered system
 - Project review report
 - User manuals & trained users

Outcome of DSDM Project



- Four possible outcomes
 - All requirements met – No further work
 - Large number of requirements not met – Re-run process from start
 - Low priority (less critical) requirements not met – Re-run starting at functional model iteration
 - Some requirements couldn't be met due to time constraints – iterate again from design and build phase

Roles in DSDM

- Developers : The only development role, also includes testers, analysts, etc.
- Technical Co-ordinator: Defines architecture and owns technical quality
- Ambassador User: Most important user role; Bring the users views/knowledge to project and to report back to users
- Visionary: User with most accurate knowledge of business requirements
- Executive Sponsor: Representative of the user company who has financial and ultimate decision making authority

DSDM Practices & Principles

- Active user participation is vital
- Teams must be empowered to make decisions
- Short delivery cycle
- Fitness for business purpose
- Iterative development
- All changes can be undone
- Only freeze core requirements at a high level
- Continual testing
- Compromise and cooperate

Agile Software Development

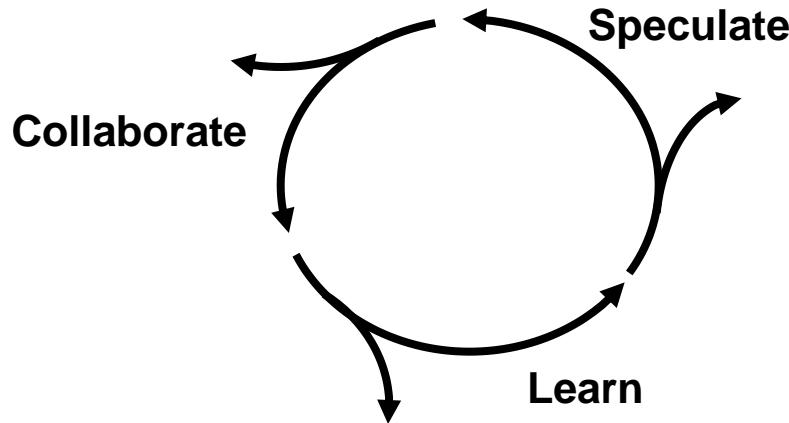
DSDM, & ASD

ASD – Adaptive Software Development

Adaptive Software Development (ASD)

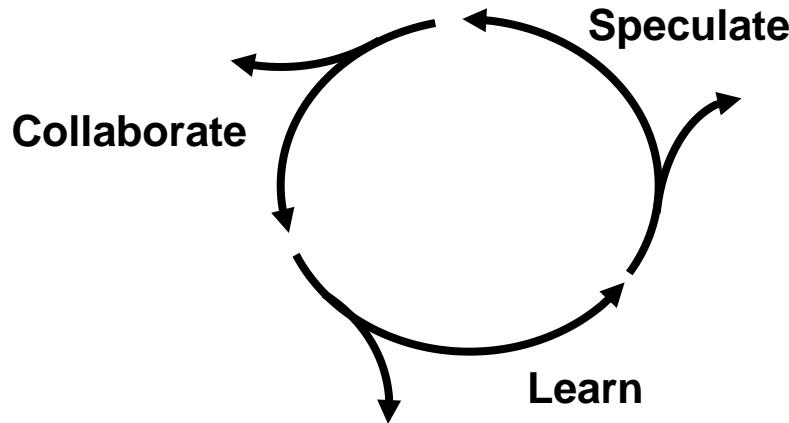
- Developed by James Highsmith in 2000
- Concentrates on developing complex large systems
- Incremental, iterative development with constant prototyping
- Aim is to keep projects “balancing on the edge of chaos” without falling in
 - Balance of control versus creativity

ASD Process



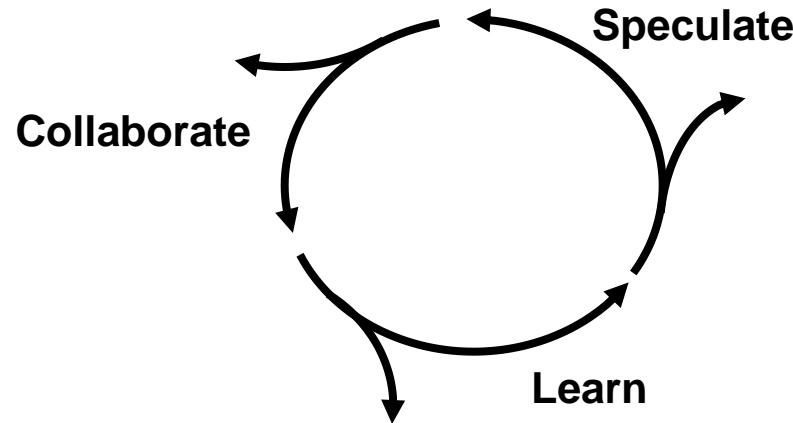
- General ASD Process Model
 - ASD carried out in cycles of 3 phases emphasising role of change in the process

ASD Process



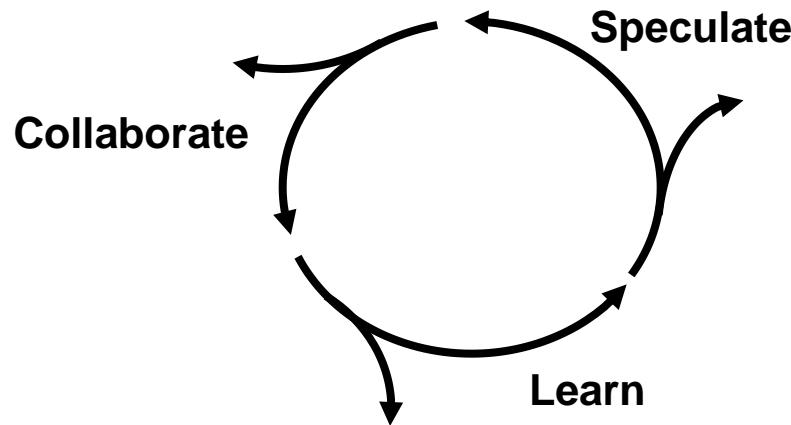
- **Speculate**: Used instead of “planning” as ‘a plan’ suggests we know what the software has to do. We don’t know, but we can speculate as to what it is the customer wants

ASD Process



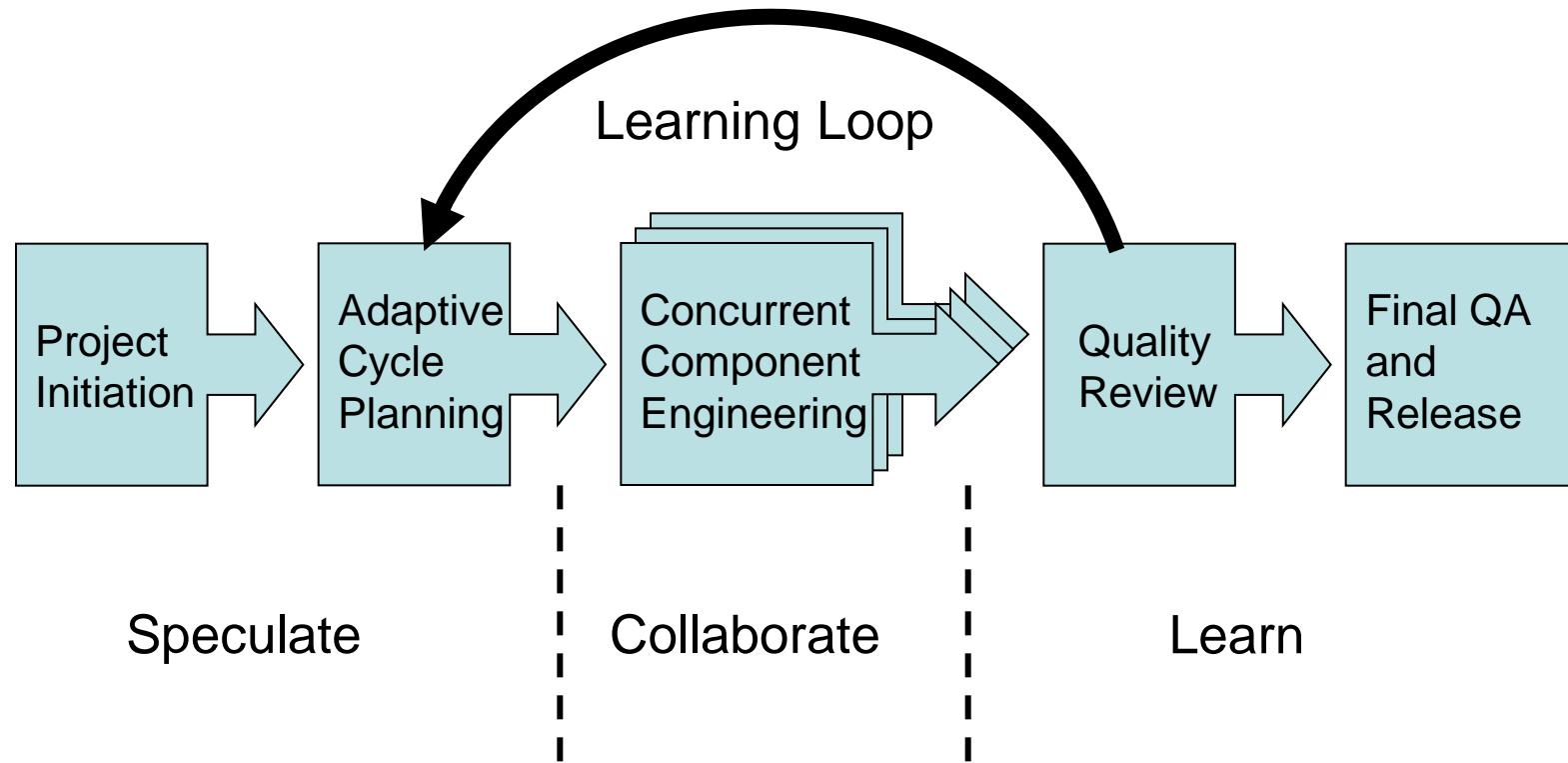
- **Collaborate**: Stresses the importance of teamwork in a volatile and uncertain undertaking - such as software development

ASD Process



- **Learn**: Stresses the need to acknowledge & react to mistakes and the fact that requirements are likely to change during development

ASD Phases & Components



ASD Speculate Phase

- Project initiation
 - Sets the project mission
 - Defines information needed to complete the project
 - Fixes the overall schedule, plus schedule & objectives for development cycles: 4-8 weeks
 - Outputs
 - Project vision charter
 - Project data sheet
 - Product specification outline

ASD Speculate Phase

- Adaptive cycle planning
 - Planning the component development
 - Refining component cycle planning as component definitions become clearer

ASD Collaborate Phase

- Phase is focussed on team collaboration to meet the needs of the project
- The development is component oriented more than task oriented
- Component development can be concurrent hence a high degree of collaboration is needed
- The definitions of the components will change, and are expected to change as the project evolves

ASD Learn Phase

- The learning loop gathers information from the quality reviews for feedback to the next speculate phase
- Repeated quality reviews:
 - Demonstrate the functionality of the software developed during cycle.
 - It is important that customer(s) are part of the review
 - Quality reviews only take place at the end of a cycle but more customer interaction is usually required so there may be Joint Application Development (JAD) sessions scheduled as well.

ASD Learn Phase

- Final QA & Release
 - Hold project postmortem to gather feedback from all stakeholders
 - Capture lessons learned to improve the overall process

ASD Principles

- Mission driven
- Component based development
- Iterative
- Tangible deadlines
- Change tolerant
- Risk - integral part of the process

Vagueness

- ASD does not define roles/structures in detail
- ASD is more a list of examples than rules
- Does not state that teams must be co-located

Exercise

- Your company has been contacted by ABC Printing inc. ABC want your company to develop a MIS system that will allow ABC to exercise greater control over its business processes for purchasing, distribution and organising its market intelligence. Getting effective control over these processes is essential for ABC's future. This will be a large system development with multiple subsystems affected, mission critical, with very tight deadlines.
- Your company has always used a waterfall approach to software development but now some of the development team and their management are arguing to take an agile approach. What are some of the difficulties they might face?

Today's Lecture

Software Testing Part 1

Software testing?

- What is software testing?

Software testing?

- What is software testing?
 - Many different definitions:
 - some focus on what is done
 - others focus on general objectives like quality, or customer satisfaction
 - and others focus on specific goals like expected (measureable) results

Software testing?

- Most important:



If the goal of testing is to show the absence of faults very few will be discovered



If the goal of testing is to show the presence of faults a large number will be discovered

Software testing?

- Hence a reasonable definition of testing is:
 - The purpose of testing is to discover errors (*faults*). Testing is the process of trying to discover every conceivable fault or weakness in a work product (Myers, 1979)
- But even this definition needs to be qualified because complete testing is impossible.

Complete testing is impossible?

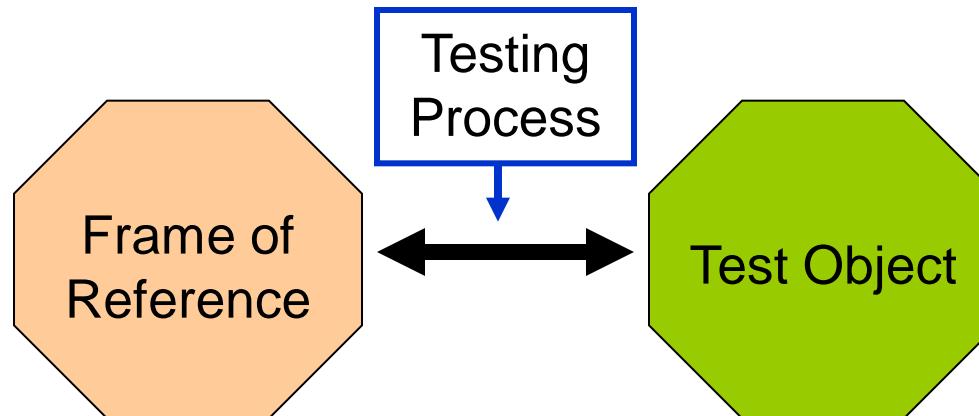
- Myers (1979) described a simple program of a loop and a few IF statements that, in most languages, could be written in about 20 lines of code. This program has **100 trillion paths**. Testing them all would take a billion years.
 - Simple truths: You can't ...
 - test the program's response to every possible input
 - test every path the program can take
 - find every design fault
 - prove the program's correct using logic
- ...so what can you do?

The job of a software tester

- To identify as many faults as possible & the more serious the problem the better!
- Specifically:
 - Testers hunt software faults
 - Testers are creatively destructive
 - Testers pursue software faults, not people
 - Testers add value
- A test that reveals a problem is a success.
A test that does not reveal a problem is a waste of time.

Testing

- In essence testing is comparison; it requires a test object and a frame of reference with which the object should comply, a test basis.
- It follows a software process within which test planning, design, execution etc. takes place



Testing: What it's not - What it is

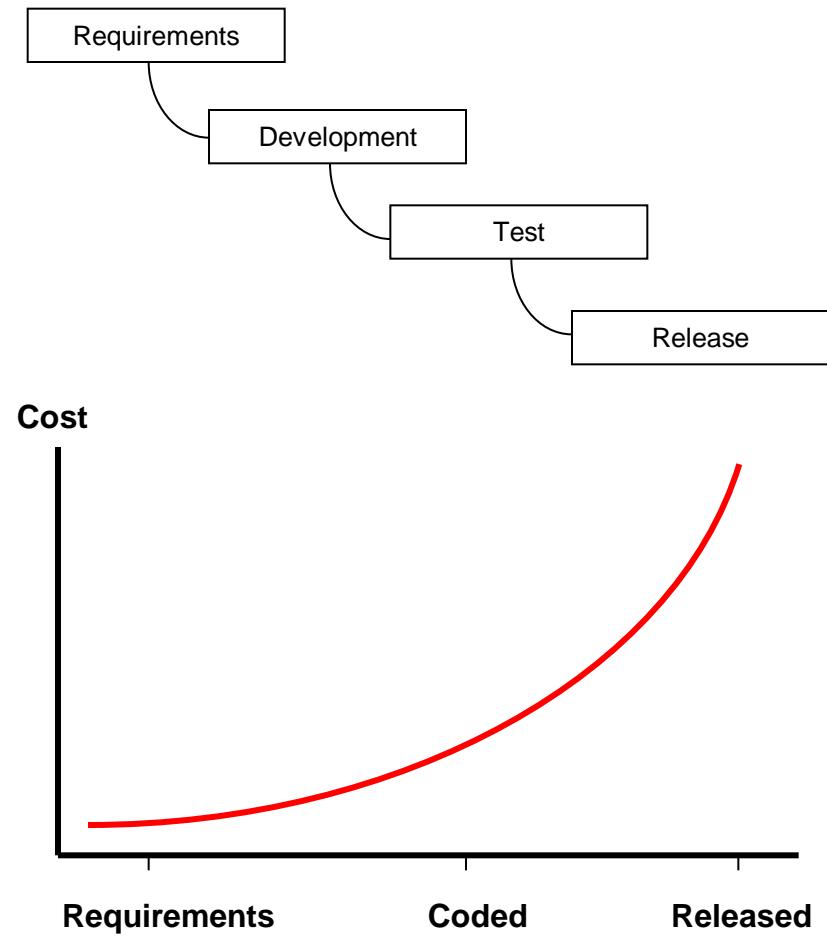
- Testing is not code debugging, which is typically unstructured and reactionary. 
- Testing is taking a professional, systematic, disciplined, planned approach to the discovery, identification, and removal of faults in software products. 

Outcomes from Effective Testing

- Timely elimination of software faults
- Cleaner software products delivered to customers
- Developers saved from building products on error-ridden sources

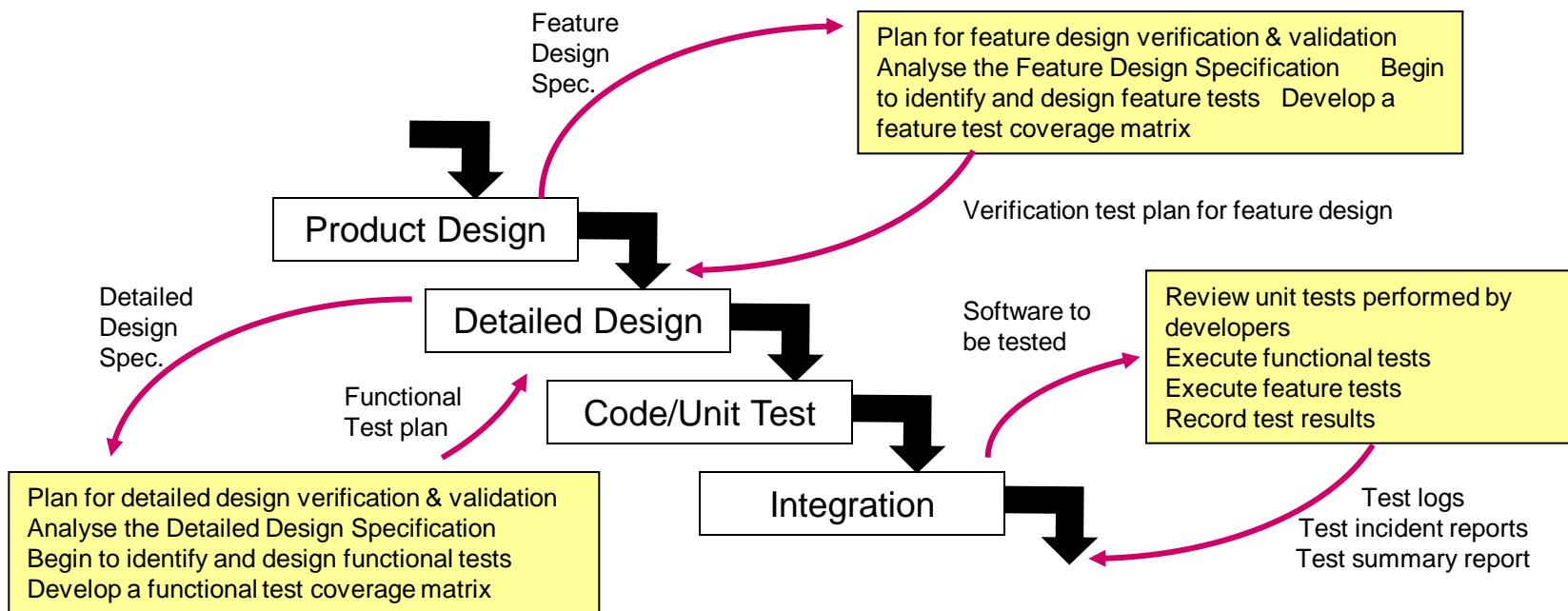
What gets tested?

- Each phase in a software lifecycle creates software products that are susceptible of testing (Specifications, Designs, Code, etc.)
- The problem is that the cost of finding & fixing faults rises exponentially the further along the lifecycle we progress
- So, while testing has explicit phases in the software lifecycle in reality testing activities need to start much earlier



What gets tested?

- Using a portion of the waterfall model as a demonstration vehicle to illustrate some of the testing activities and deliverables



How is Testing Performed?

- Testing can be separated into two basic forms:
 - Verification: does the software comply with constraints – ‘did we build it the right way’, use the correct coding standards, follow design conventions and templates, etc.
 - Validation: does the software satisfy the specified requirements – ‘did we build the right thing’

How is Testing Performed?

- **Verification:**
 - Characterised as a ‘human’ examination or review of the work products (specifications, designs, code...)
 - Verification has proven to be one of the most cost-effective routes to quality improvement
 - Verification has a large educational component
 - Over time organisations can tailor the verification techniques to meet their specific needs

How is Testing Performed?

- **Verification Techniques:**
 - Formal inspections
 - Highly structured & centred around a 1-2 hour inspection meeting preceded by a walkthrough
 - Defined roles (Author, Moderator, Tester & Reader), & preparation by all participants usually 4 people
 - Inspection team go through the work product looking for faults or potential faults
 - Formal fault classification recording (type & severity)
 - Plus spin-off's: education & process improvement

How is Testing Performed?

- **Verification Techniques:**
 - Walkthroughs
 - Closely related to, but less formal than inspections
 - Only the presenter needs to prepare
 - Questioning by the audience

How is Testing Performed?

- **Verification Techniques:**
 - Buddy checks
 - Checking of software products performed by someone other than the author

How is Testing Performed?

- **Verification Techniques:**
 - Checklists
 - More of a tool than technique
 - Quick and highly effective
 - Used in formal inspections
 - Developed for specific work products from different phases (e.g. Requirements verification checklist, High level design checklist, etc), or can be generic (General document verification checklist)
 - Can be useful for training & education

How is Testing Performed?

- **Validation:**
 - Is characterised as ‘computer based’ testing
 - Historically: testing = validation
 - Now: testing = verification + validation
 - Two fundamental validation strategies
 - **Black-box testing**
 - Functional tests performed without knowledge of the internals of the test object
 - **White-box testing**
 - Tests performed on internal program structure

How is Testing Performed?

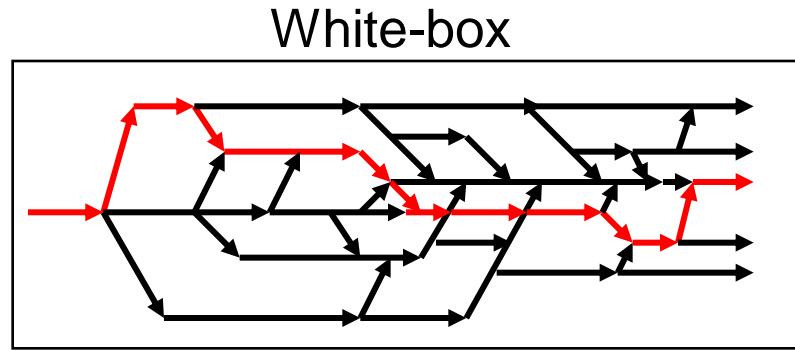
- Black-box testing:



- Based upon functional specifications
- Valid & Invalid inputs to verify correct output
- Will not test internal paths or hidden functions
- Can detect missing functionality

How is Testing Performed?

- White-box testing:



- Based upon detailed design specifications or the code itself
- Requires knowledge of the internal program structure
- Will not detect missing functionality
- Each statement is executed at least once

Methods

- Black-box testing methods:
 - Equivalence Partitioning
 - Attempts to reduce the number of tests to the necessary minimum. Then select the right test cases to cover all possible scenarios

Methods

- Black-box testing methods:
 - Equivalence Partitioning
 - A program requires a valid input for “Weekday” to be 1-7. Describe the minimum set of test cases to validate an input.
- Hint: one valid class and two invalid classes**

Methods

- Black-box testing methods:
 - Equivalence Partitioning
 - All possible responses can be tested by defining one valid class (partition) (i.e. any number between 1-7), and two invalid classes(<1 and >7)
 - Since we are treating all the values in the partition as equivalent it is sufficient to select one test case from the partition to check the behaviour of the program.

Methods

- Black-box testing methods:
 - Boundary Value Analysis
 - Closely related to Equivalence Partitioning but pays greater attention to the boundary values. Simple in above example (0 - 1, and 7 – 8), but in practice very difficult as boundaries may be difficult to identify
 - Should also be applied to the output values expected

Methods

- White-box testing methods: Path Testing
 - Statement Coverage
 - Each statement is executed at least once
 - Weakest coverage criteria

Methods

- White-box testing methods: Path Testing
 - Branch Coverage
 - Each statement is executed at least once; each decision takes on all possible outcomes at least once
 - An improvement on Statement Coverage but only tests the logicality (true or false) of each branch.

Methods

- White-box testing methods: Path Testing
 - Condition Coverage
 - Each statement is executed at least once; each condition in a decision takes on all possible outcomes at least once
 - Stronger level of coverage as it checks the way that each condition in a branch can be made true or false

Methods

Illustration : White-box testing methods

- Consider the following code:

```
IF (A > B and C == 5)
    THEN do SOMETHING;
    SET D = 5;
```

- Possible test cases:
 - A > B and C = 5 (SOMETHING is done, then D is set to 5)
 - A > B and C ≠ 5 (SOMETHING is not done, D is set to 5)
 - A ≤ B and C = 5 (SOMETHING is not done, D is set to 5)
 - A ≤ B and C ≠ 5 (SOMETHING is not done, D is set to 5)
- With Statement Coverage a person can execute all 3 lines of code by testing case a)
- With Branch Coverage a person can test the branch by testing case a) and testing any of the other cases
- Conditional Coverage requires the person to test all the cases above.

Methods

Exercise : White-box testing methods

- Consider the following code:

```
IF (A > B and C == 5 and D > E)
    THEN do XYZ
    ELSE do PQR;
SET E = A;
```

- Describe test cases for Conditional Coverage for the above code.

Interesting Dichotomies

- Describe ways in which testing can help build customer confidence or assure the quality manager of the likely quality of the delivered products?

Next Lecture

- Types of testing
- Test planning
- Test organisation
- Test measurements

Today's Lecture

Software Testing Part 2

Previous Lecture

- Discussed what testing is and what testers do
 - Uncertainty associated with testing coverage
 - Testing as a distinct software discipline
- Benefits of early testing and fit with the lifecycle
- Verification & validation
- Black-box and White-box testing

Today

- Types of testing

Types of Testing

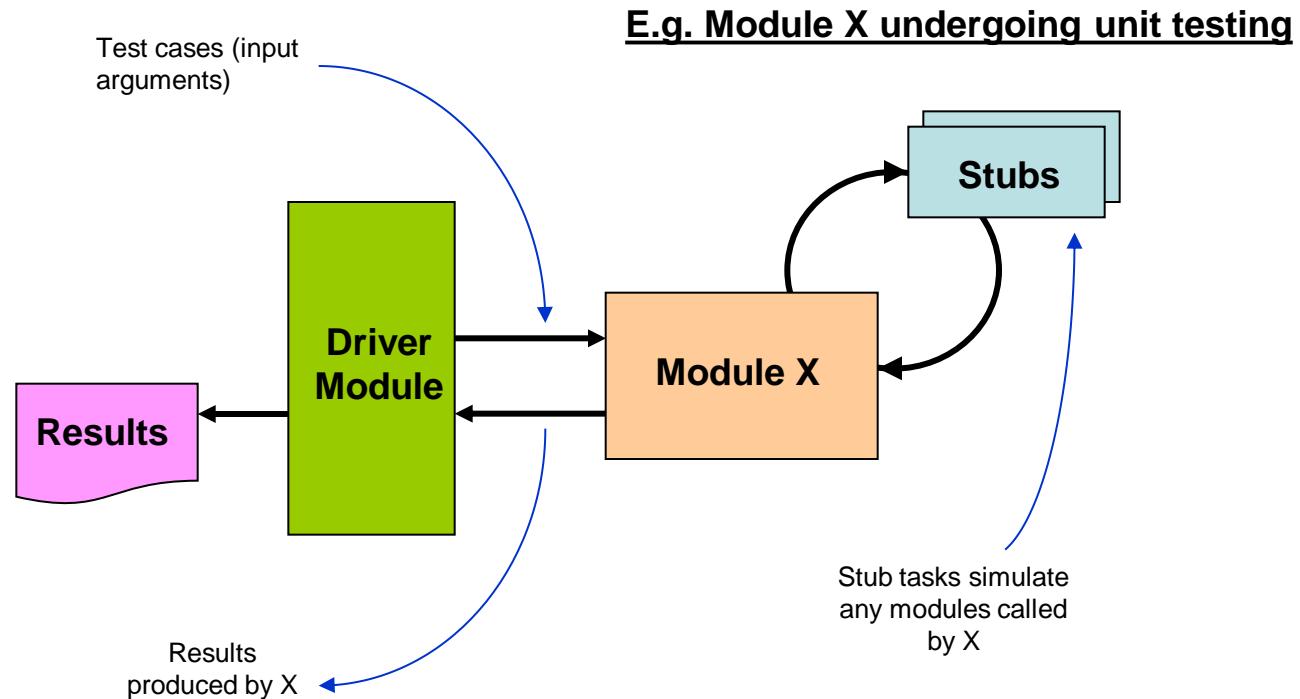
- Low level testing
 - Unit (module) testing
 - Integration testing
- High level testing
 - Usability testing
 - Function testing
 - System testing
 - Acceptance testing
- Other test types
 - Regression testing

Types of Testing: Low Level

- Low level testing
 - Testing individual components
 - Requires intimate knowledge of program's internal structure
 - Usually performed by development engineers
 - Two types:
 - Unit (module) testing
 - Integration testing

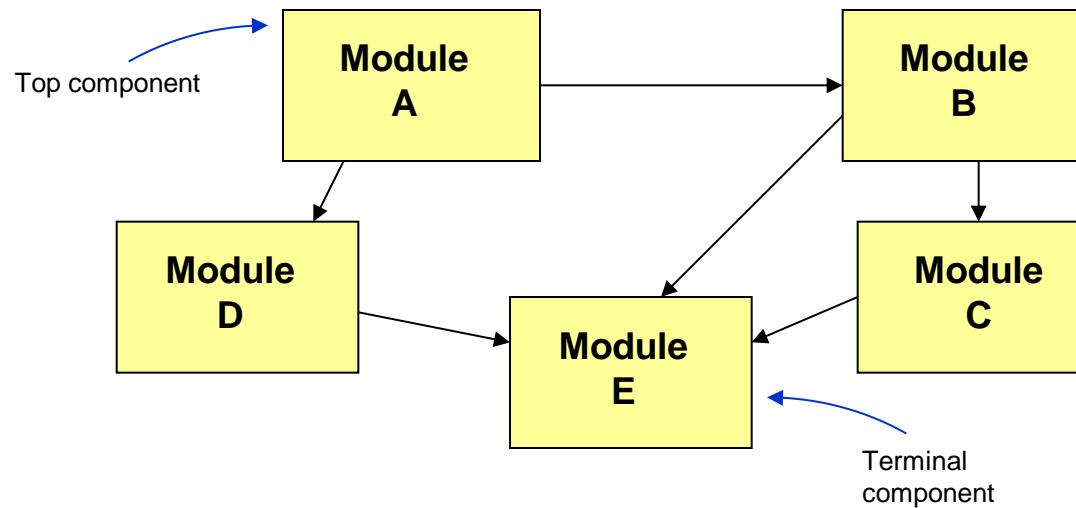
Types of Testing: Low Level

- Unit (module) testing
 - Testing individual components to discover discrepancies between the modules interface specification and actual behaviour.
 - Facilitates error diagnosis and correction by development



Types of Testing: Low Level

- Integration testing
 - Process of combining and testing multiple components – may be done incrementally using **top-down** or **bottom-up** approaches or more usually, but least effectively as a non-incremental '**big-bang**' integration.
 - Primarily used to discover errors in the interfaces between modules but can be performed on several levels; modular, subsystem, system and network level; hierarchical



Integration Approaches

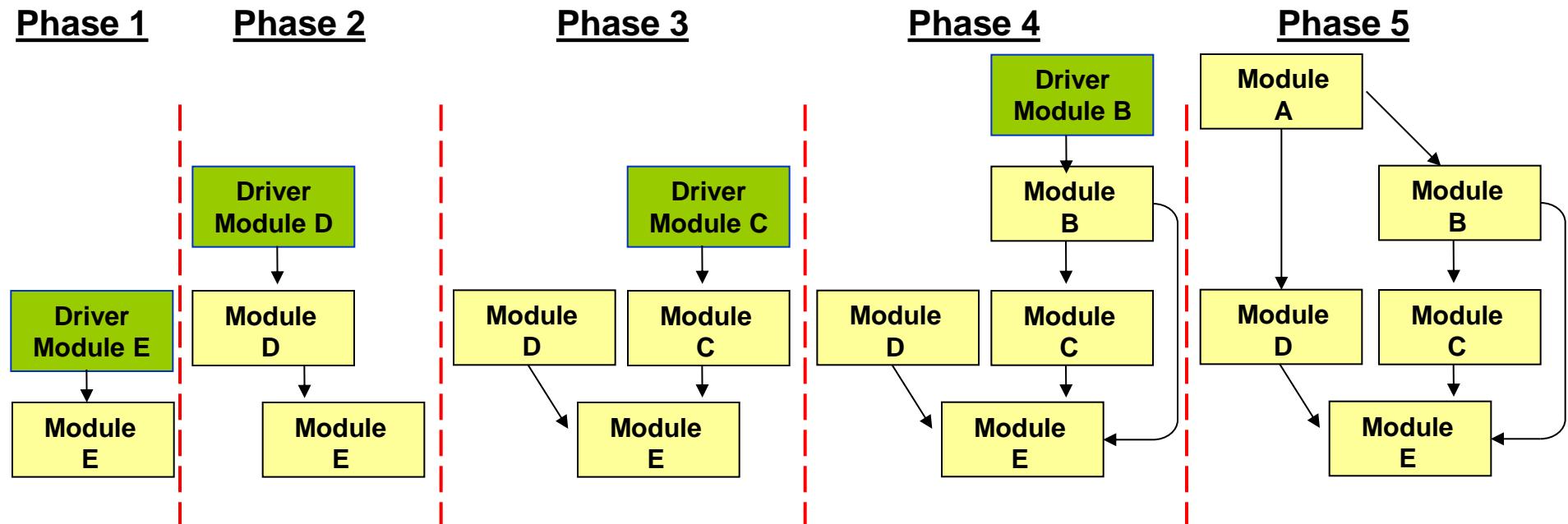
- ‘Big-Bang’ Integration
 - All components are combined at once to form the program
 - The integrated result is then tested
 - Testing is difficult since a fault can be associated with any component

Integration Approaches

- ‘**Incremental integration**
 - Unit test the next program component after combining it with the set of previously tested components.
 - Requires less work, as fewer driver and stub modules
 - Earlier detection of mismatched component interfaces
 - Fault finding is easier because faults found are usually associated with the most recently added component
 - More thorough testing may result because the testing of each new component can provide further function and logic coverage of previously integrated components.

Incremental Integration

- ‘Bottom-up’ Approach
 - Begin with terminal modules of hierarchy
 - A driver module is produced for every module
 - The next module to be tested is any module whose subordinate modules (the modules it calls) have all been tested
 - After a module has been tested, its driver is replaced by an actual module (the next one to be tested) and its driver

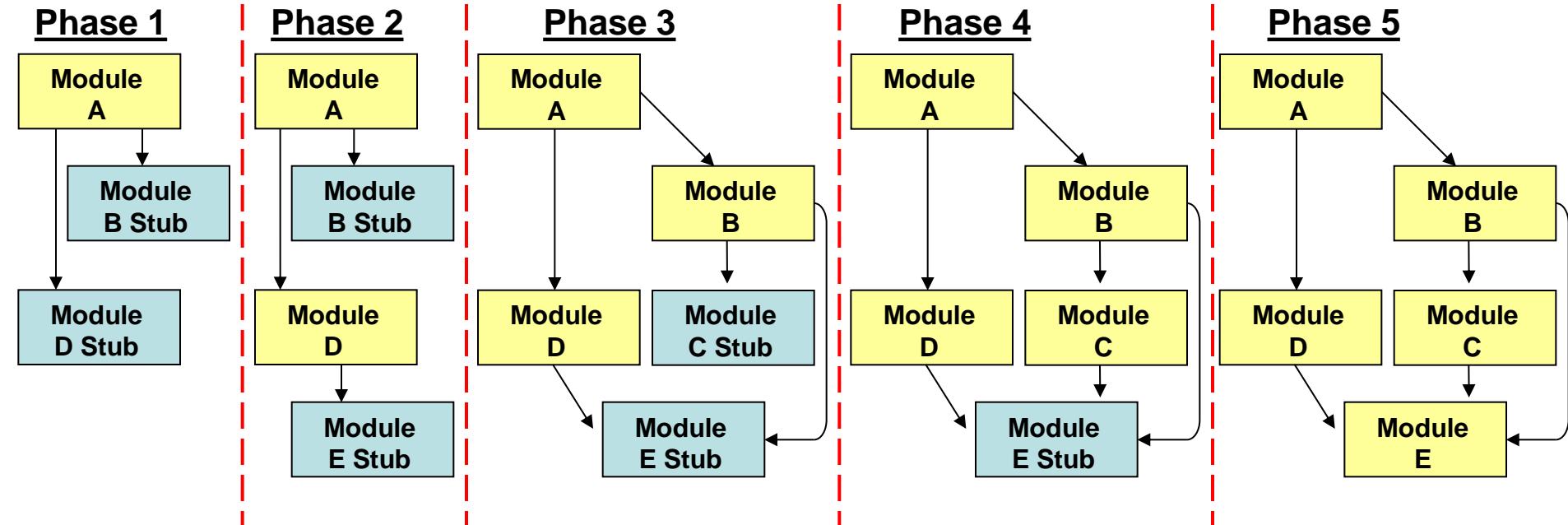


Exercise

- Describe the ‘bottom-up’ testing sequence for the handout

Incremental Integration

- ‘Top-down’ Approach
 - Begin with top module of hierarchy
 - Stub modules are produced (may be more complicated than they first appear – see below)
 - The next module to be tested is any module with at least one previously tested calling module
 - After a module has been tested, one of its stubs is replaced by an actual module (the next one to be tested) and its required stubs



Exercise

- Describe the ‘top down’ testing sequence for the handout

Question?

- What do you think are the advantages of ‘top-down’ testing over ‘bottom-up’?
- What do you think are the advantages of ‘bottom-up’ testing over ‘top-down’?

Advantages????

- ‘Bottom-up’ integration has the disadvantage that the program as a whole does not exist until the last module is added.
- ‘Bottom up’ integration has the advantage that driver modules are less expensive to create than stubs & less needed – 4 driver modules in the example
- ‘Top-down’ integration has the advantage that a skeletal version of the program can exist early and allows demonstrations.
- ‘Top-down’ integration has the disadvantage that the required stubs could be expensive – 7 stub modules to be written and modified in the example
- No real agreement on which is the best approach

Types of Testing: High Level

- High level testing
 - Testing whole complete product
 - Usually performed by independent test group
 - Types include:
 - Usability testing
 - Function testing
 - System testing
 - Acceptance testing

Types of Testing: High Level

- Usability testing
 - Goal is to test how well the software is adapted to users' work styles, rather than have users adapt their work style to the software.
 - Having users work with the product and observing their responses to it.
 - Done as early as possible with the real customer
 - Functional design specification must be available
 - Test criteria identified (How easy is it for user to navigate? How many mouse clicks to perform a function? etc.)
 - Often concerned with issues of presentation as opposed to functionality – the user experience
 - Usability characteristics can include; Accessibility, Responsiveness, Efficiency, Comprehensibility



Types of Testing: High Level

- Function testing
 - Goal is to detect discrepancies between a program's functional specification and its actual behaviour.
 - When a discrepancy is found is it the program or the specification which is wrong?
 - All black-box methods for function-based testing are applicable
 - Performed by independent testing group
 - Can begin when sufficient functionality is available or after unit and integration testing have been completed

Types of Testing: High Level

- System testing
 - System testing, despite its name, is not function testing the entire system – rather it tries to show the system does not meet its original requirements and objectives
 - Requirements or specification documents do not describe the system's functions in precise terms – so system testing is difficult to do.
 - No methodology can be applied so testers must use a great deal of creativity, and especially thinking from a users perspective

Types of Testing: High Level

- System testing (Contd.)
 - Approaches that can be used in system test
 - Volume testing (can system handle required volumes of data)
 - Stress testing (can system handle peak load conditions)
 - Security testing (can the systems security be breached)
 - Performance testing (can system meet performance needs)
 - Resource usage testing (will system exceed its resource limits)
 - Configuration testing (will system fail if differently configured)
 - Other popular approaches:
 - **Compatibility testing**
 - **Installability testing**
 - **Recovery testing**
 - **Serviceability testing**
 - **Reliability testing**
 - **Availability testing**

Types of Testing: High Level

- Acceptance testing
 - Goal is to detect discrepancies in system's behaviour in actual target environment.
 - Tests usually run by customers
 - Run for a pre-specified period which has to reflect:
 - Longest functional response time (e.g. System has a major feature, a 6-month datalog function so reasonably minimum period for acceptance testing ≥ 6 months)
 - Learning curve for using new system

Types of Testing: Other

- Regression testing
 - Most test cases rather than simply being discarded will become part of a regression test suite which is maintained for life of the product.
 - Regression tests are tests that have already been run and passed previously but can be exercised (re-tested) again as new functionality is introduced.
 - Makes sure that the new stuff doesn't break existing functionality
 - In many organisations regression testing is automated and run off-peak

Software Testing: Group Exercise

- Can testers stop a product shipping?

Can testers stop a product shipping?

- Yes and No – Will certainly have an influence
- Tends to be overshadowed by business needs
- Are the testers knowledgeable enough to make the decision?
- If testers have ultimate authority can this lead to others (Project Managers, Line Managers, QA Personnel...) lessening their responsibility?

Next Lecture

- Test Planning
- Test Organisation
- Test Measurements

Today's Lecture

Software Testing Part 3

Previous Lecture

- Discussed role, benefits & organisational fit of software test
- Testing techniques & approaches
- High Level, low level & regression testing

Today

- Test planning
- Test organisation
- Test measurements

Software Testing: Test Management

- Testing needs a focus/context; to be structured, and to be objectively understood
- 3 key elements:
 - **Test organisation** – the individual(s) or group responsible for testing
 - **Test planning** – providing a systematic, disciplined approach to the testing
 - **Test Measurements** – understanding the effectiveness of the test process, and product under test

Software Testing: Organisation

- The test group is no different to any other professional software group & in recent years has gained growing respect for its contribution.
 - ⇒ Obviously its size & structure must be tailored to fit the business needs
 - ⇒ In medium to large software enterprises the Test group will likely consist of:
 - ⇒ A test manager
 - ⇒ Test technical leads
 - ⇒ Test engineers
 - ⇒ Junior engineers

Software Testing: Organisation

- Hence the test group has **similarities** in hierarchical structure and responsibilities to the development group
 - ⇒ Test has been established as a distinct professional discipline
 - ⇒ The test team are also part of the **product development team** - shared vision & cooperation with other groups
 - ⇒ The test group also have **early involvement** in the software lifecycle

Software Testing: Organisation

- Fundamentally testing differs in its focus:
 - Development; creative focus - which is inventive - transforming abstractions (designs, specifications, etc.) into software products
 - Test; creative focus - but is destructive - breaking software and understanding in precise terms why it broke. Added value – helping identify what can be done to stop failure



Software Testing: Organisation

- The approach that organisations take to testing will be varied and will depend upon many organisational factors such as:
 - Size of the organisation
 - Software lifecycles used
 - Tall or flat organisational architecture
 - Market or product oriented
 - Centralised or decentralised organisation
 - Hierarchical or diffused
 - and so on...

Software Testing: Organisation

- Hence there is no best test structure or approach
- The organisation's basic design elements can be combined to produce many different test structures which typically, in practice, reflect the maturity of the organisation & how much it values testing

Software Testing: Planning

- Effective software testing requires planning, and following a methodical approach enforcing discipline and structure on the testing
- This requires the development of a test plan
- A test plan is:

“A document describing the scope, approach, resources, and schedule of intended testing activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning” ANSI/IEEE Standard 829-1983

Software Testing: Planning

- Appropriate methods and tools are used for planning and executing tests
- Test personnel are trained to perform their testing duties
- Effective measurement and analysis techniques are used to evaluate testing effectiveness

Software Testing: Planning

- Test plan typically covers the following;

- Test plan identifier - unique name or number
- Introduction – references, objectives
- Test items – modules that will be tested
- Features to be tested – cross referenced to the design specification
- Features not to be tested – which ones and why not
- Approach:
 - Who will do the testing?
 - Main activities, techniques, and tools to be used
 - How much testing needs to be done?
 - What are the constraints – deadlines, testers availability, test items

Software Testing: Planning

- Test plan typically covers the following (contd.)

- **Pass/fail criteria** – how does a tester decide if a test has passed or failed
- **Suspension criteria and resumption requirements** – what happens when we get a ‘show-stopper’
- **Test deliverables** – all artefacts that will be created in the testing of the software product
- **Testing Tasks** – all tasks necessary to prepare for & do the testing
- **Environmental needs** – hardware, software, facilities etc. to perform the testing
- **Responsibilities** – groups or people responsible for the various testing tasks

Software Testing: Planning

- Test plan typically covers the following (contd.)

- **Staffing & training needs** – people that are required & training they need to perform the testing
- **Schedule** – test milestones with dates, & when resources will be needed to perform the test functions
- **Risks and contingencies** – prioritised list of risks associated with the testing & plans to mitigate them
- **Measurements to be collected** – to evaluate test effectiveness
- **Approvals** – signoffs that are required from management or the Quality department

Software Testing: Planning

- In reality there may be multiple **test planning** documents
 -  **Test plan** – Provides an overview of the testing effort for the product. In small organisations this may be the only document produced & may contain everything.
 -  **Test Design Specification** – Specifies how a feature or a group of features will be tested
 -  **Test Case Specification** – Defines the test cases
 -  **Test Procedure Specification** – Describes the steps for executing a set of test cases

Software Testing: Execution

- There may also be multiple **test execution** documents created
 - **Test Log** – A chronological record of the test executions and events that happened during testing
 - **Test Incident Report** – This is a problem report. Contains information such as: inputs, expected results, actual results, anomalies, date & time, observations...
 - **Test Summary Report** – A summary of the series of tests run – issued after completing a cycle of testing

Software Testing

- Test measurements
 - Test Metrics are measurements that are taken to assess how effective and thorough the testing is

Software Testing: Metrics

- Big, difficult questions arise for software test managers:
 - Questions about product quality:
 - ‘Does the product have an acceptable level of quality?’
 - ‘Can we ship the product?’
 - ‘Is the software ready to be released to the next phase of testing?’

Software Testing: Metrics

- Questions about the testing process:
 - ‘Was the testing performed effectively?’
 - ‘When will we have enough testing done?’
 - ‘What is our ROI for all the testing we’re doing?’
- Questions about the past:
 - ‘How many high severity faults have we found so far?’
 - ‘What is the testing effort expended so far?’
- Questions about the future:
 - ‘How many high severity faults do we estimate are remaining?’
 - ‘How much more time do we estimate we need to spend testing product X?’

Software Testing: Metrics

- *Some* of these questions we will be able to answer with a high degree of accuracy – especially if they concern historical data - and provided we have collected and processed that data.
- *But other questions* we will only be able to answer in terms of probabilities or statistical inference based on previous experience. Again this pre-supposes the data has been collected and processed.

Software Testing: Metrics

- Hence, software testing (just like other project and organisational activities) needs to use data – giving a best possible factual basis for answering questions:
 - about the product under test
 - about the testing process itself

Software Testing

- **Exercise**
 - As a manager compose a list of measures that you can use to indicate:
 - The effectiveness of your test team
 - The quality of the software products under test
 - The thoroughness of the testing performed
 - State:
 - The name of the measurement
 - Who should collect the data for the measurement
 - The measurement units
 - The frequency of collection
 - Reporting format (e.g. graphs, pie charts, etc.)

Software Testing: Metrics

- Test data is typically in the form of measures that apply to both Verification and Validation activities:
 - Is verification cost effective?
 - What is the frequency of a particular type of fault?
 - How many high severity faults are found in verification?
 - How many faults are escaping verification activities & are found in validation?
 - How does the number of tests run compare with numbers that are passing or failing?

Software Testing: Metrics

- The key to getting through the maze of test measurements is - only measure something if it is useful
- Measurement just for the sake of measurement alone is a waste of time

Software Testing: Metrics

- Some examples of measurement classes:
 - Code complexity Measures
 - Test efficiency Measures
 - Test Coverage Metrics
 - Test Execution Status Reports
 - Efficiency Reports
 - Fault Cost Analysis

Software Testing: Metrics examples

- Complexity measures:
 - Code size - (Lines of Code (LOC), Source Lines of Code (SLOC), Thousands of Assembler Equivalent Lines of Code (KAELOC))
 - Function points – pieces of functionality an information system provides to a user
 - Cyclomatic complexity index – Calculating the number of independent paths in source code

Software Testing: Metrics examples

- Test efficiency measures:
 -  Faults found in the current phase v's escaped defects from previous phases
 -  Faults escaping verification but found in validation
- Test coverage measures:
 -  Trace matrices (Tests x Requirements)
 -  Statement coverage

Software Testing: Metrics examples

- Test execution status:
 - % of system tested to date
 - % of total test cases run to date
 - % passed/failed/retested to date
 - % problems found still open to date

Software Testing: Metrics examples

- Efficiency reports:
 - Arrival rate of faults
 - Closure rate of faults
 - Problem severity
 - Aging of Faults
 - Fix Response Time

Software Testing: Metrics examples

- Fault cost reports:
 - Reporting by severity
 - Reporting by resource requirements
 - Reporting by business (€) impact

Software Testing: Metrics

- Overly complex metrics are probably not worth computing
- ‘Everything should be as simple as possible – but not simpler’ (Einstein)

Software Testing: Conclusion

- Software testing - not an undisciplined afterthought of the software development process
- Emerging as a discipline in own right
- Mature approaches and testing techniques
- Requires organisation & planning
- Needs to deliver measurable results

Next Lecture

- Project planning

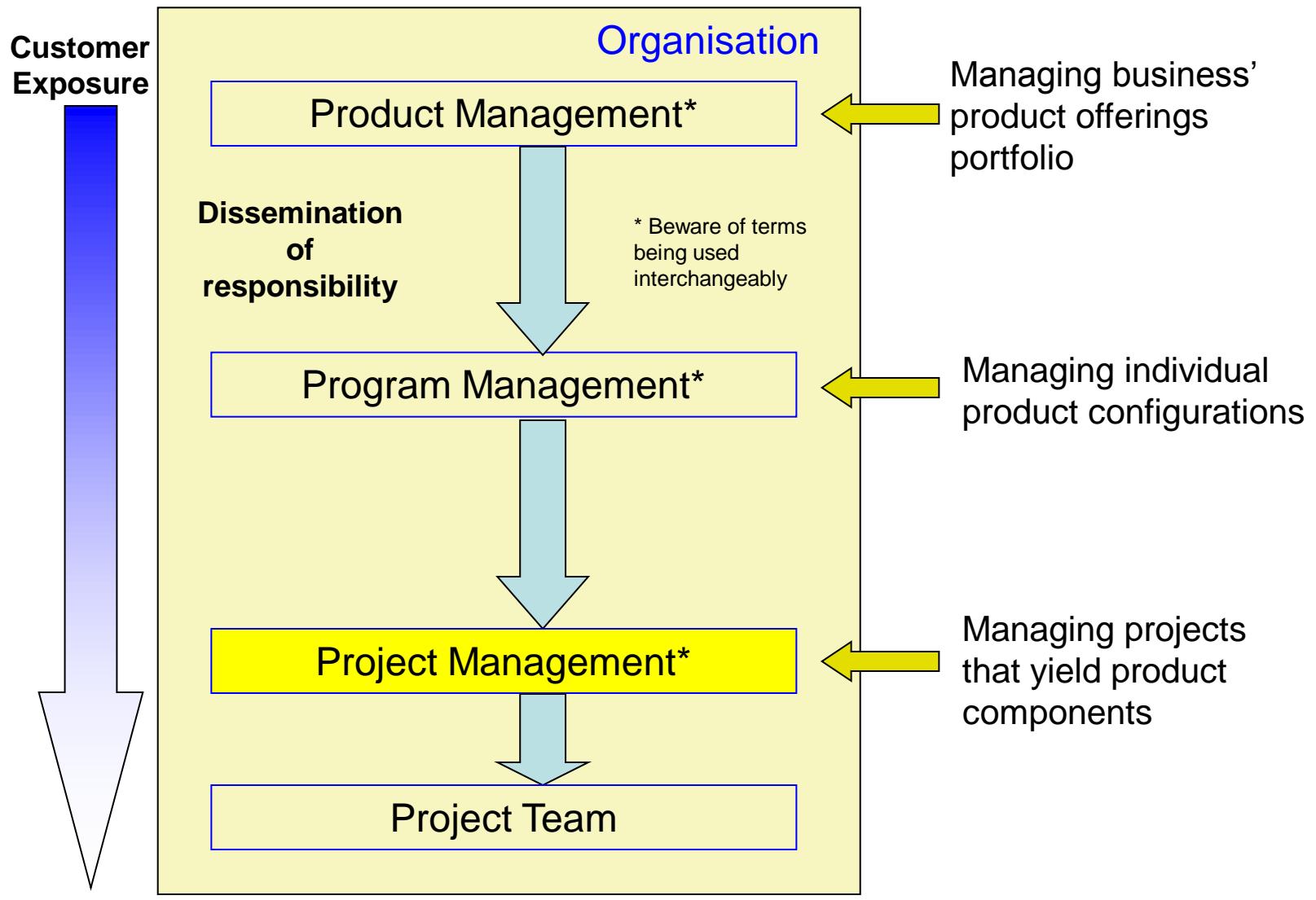
Test Metrics Exercise

To measure	Name of metric	To be collected by	Units	Frequency	Format
The effectiveness of your test organisation					
The quality of the software products under test					
The thoroughness of the testing performed					

Today's Lecture

Software Project Planning

Software Proj. Mgmt.: Organisational Fit



But this hierarchy & terminology is tailored to suit organisational size!

Software Project Characteristics

Project – A temporary undertaking to create a unique product or service with a defined start and end point and specific objectives that, when obtained, signify completion.

General project characteristics:



- non-routine tasks are involved
- planning is required
- specific objectives to be met / specific products to be created
- predetermined time-spans
- work usually carried out by a team with several specialisations
- work is done in several phases
- resources are constrained
- project is large or complex

The more of these factors apply – the more difficult the project



Major obstacles

Picture is further complicated because of the nature of software projects themselves: Unique **Inherent Difficulties**

- **Invisibility** – ‘unvisualizable’
- **Complexity** – more so than other engineered artefacts
- **Conformity** – with other systems interfaces, many of them arbitrary
- **Flexibility** – easy to change.... too easy!!!

Fred Brooks: ‘No Silver Bullet – Essence and Accidents of Software Engineering’, Computer, 1987

Further Obstacles

More bad news!

... the situation is made worse because software projects try to plan with limited up front information...

'...the underlying philosophy ... is that the requirements of the software project are completely locked in and frozen before the design and development commences'

McCauley, R., 'Agile development methods poised to upset status quo', SIGCSE Bulletin, 2001

Further Obstacles

but, '....in most cases the people who commission the building of a software system do not know exactly what they want and are unable to tell (a software project) all that they know'.

Parnas, D.L. & Clements, P.C., 'A Rational Design Process: How and Why to Fake It', in Software Project Management, Kemerer, C.F. (ed.), McGraw Hill, 1997



Recent figures: >60% of software systems fail to operate as intended or can be used at all, while ~25% are never completed. Software projects are often excessively late, and cost multiple times their planned budgets.

Planning difficulties

Example: PPARS (Personnel, Payroll & Related Systems) for HSE

- Project estimates: €9m (2002)
- Actual cost to date: €220m (25x over estimate)
- Schedule: Over 4 years behind
- Content/Functionality: Still has major flaws, “to date (*Feb 2007*) can only pay 30,000 of 136,000 HSE staff”
- System Deployment: “Major problems with adoption & rollout of system” – Comptroller & Auditor General
- Conclusion: “needs replacing” (HSE, July 2007)).

Effective Project Management

So effective project management is essential if costly disasters are to be avoided.

More specifically project planning needs to be accurate, and especially it hinges on:

Project Estimates

Project Schedules

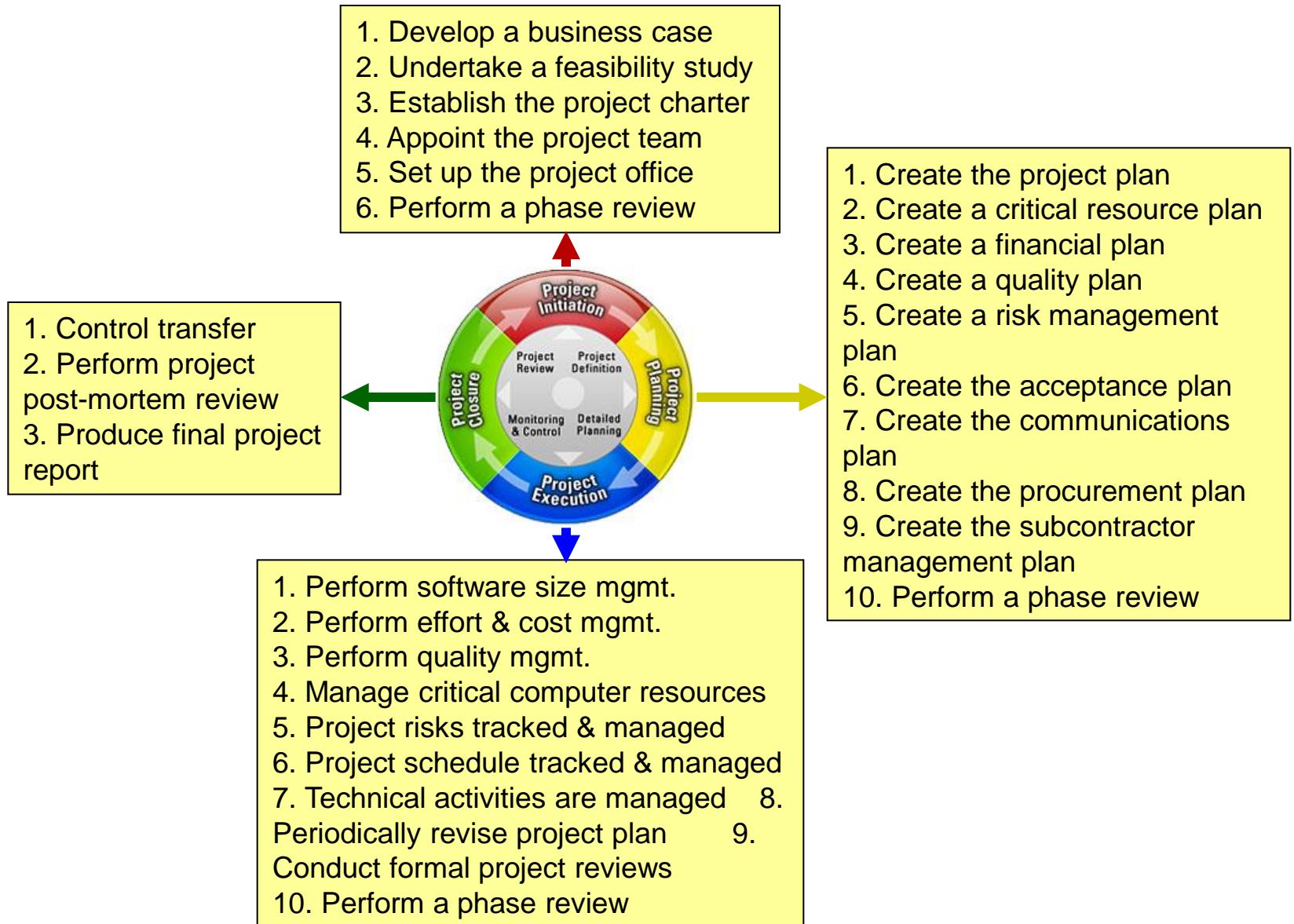
Software Proj. Mgmt.: General Phases

Fairly well accepted that project management spans 4 different phases; Project Initiation, Project Planning, Project Execution, and Project closure – (some models include a 5th phase to Evaluate the Project)



and there are specific activities associated with each phase.

Software Proj. Mgmt.: General Activities



Software Proj. Mgmt.: Stakeholders

People who have a stake or interest in the project:

- Different backgrounds
- Different motivations
- Different objectives
- Different information needs

May be internal to the project:

- Engineers
- Project managers
- Quality personnel, etc.

External to the project but within the same company:

- Other project managers
- IS manager
- Intellectual property manager

Totally external to the project and the company:

- Customers
- Regulatory bodies
- Contractors

Software Proj. Mgmt.: Planning

Major Phases:

- Project Evaluation
 - Technical Analysis
 - Strategic Analysis
 - Cost/Benefit Analysis
 - Risk Profile Analysis
- Project Approach
 - Technical Plan
 - Lifecycle Model
- Estimation
 - Size & Effort
 - Application of Techniques
- Scheduling
 - Network Planning
 - Critical Path Analysis
- Risk Assessment
- Project Resourcing

Project Evaluation

- The success of a project, both technically & in business terms is built upon a sound initial evaluation of the technical, organizational and financial feasibility of the project.
- Every project needs to be strategically evaluated for feasibility against alternatives
- “Do nothing” is also an alternative to be considered

Project Evaluation (contd.)

- Risks involved are part of the evaluation
- Project evaluation is usually carried out very early in the lifecycle and again when required
- The initial evaluation of a single project also takes into account strategic organizational goals

Project Approach

- Development of a high-level technical plan
 - Introduction and summary of constraints
 - Characteristics of the system, Risks & Uncertainties
 - Recommended approach
 - Selected methodology, Required tools, Target environment
 - Implementation considerations
 - Development environment, Training needs
 - Implications
 - Financial, Software product offerings

Project Approach

- Selection of an appropriate lifecycle model for the software development
 - Waterfall
 - ‘V’-model
 - XP
 - Scrum

Software Project Planning

Major Phases:

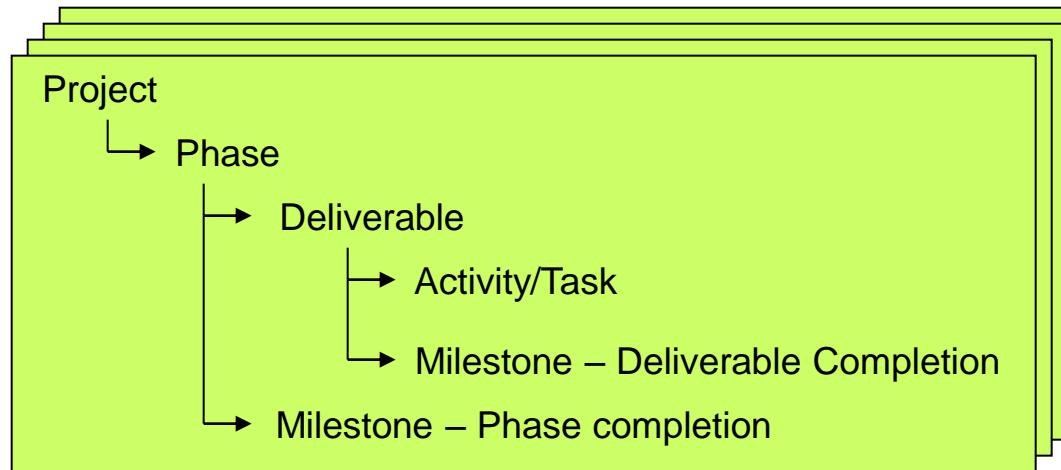
- Project Evaluation
 - Technical Analysis
 - Strategic Analysis
 - Cost/Benefit Analysis
 - Risk Profile Analysis
- Project Approach
 - Technical Plan
 - Lifecycle Model
- Estimation
 - Size & Effort
 - Application of Techniques
- Scheduling
 - Network Planning
 - Critical Path Analysis
- Risk Assessment
- Project Resourcing

Estimation

- Starting Point: The Work Breakdown Structure
 - “A WBS represents a logical decomposition of the work to be performed and focuses on how the product, service or result is naturally subdivided. It is an outline of what work is to be performed” (Haugan, G.T., “Effective Work Breakdown Structures”, Vienna Va.,: Management Concepts, 2002)
 - Groups low level parts of the project into deliverable sections which, all together, make up the project scope

Work Breakdown Structure (WBS)

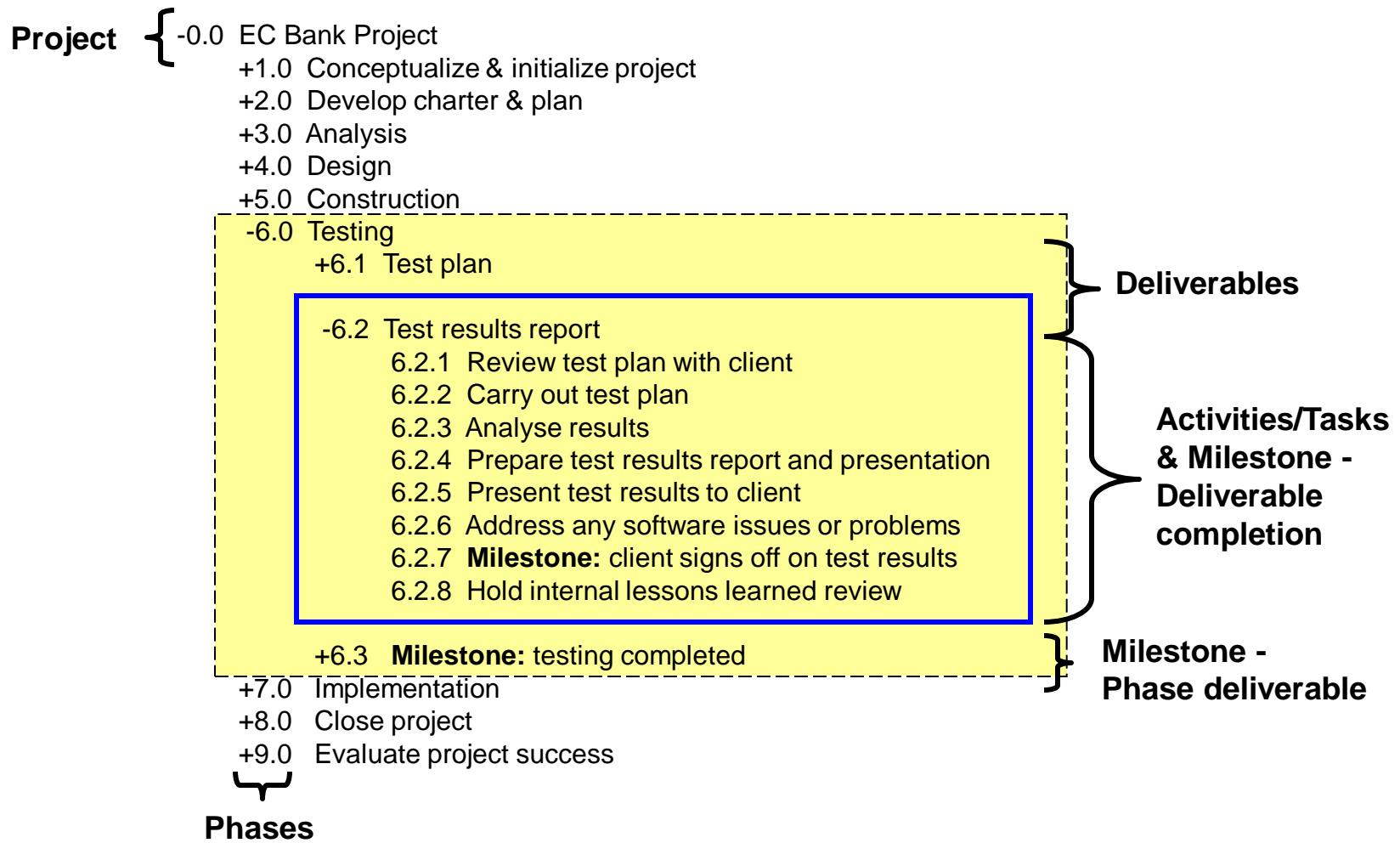
- WBS decomposes the project into *work packages*.



- May be described as structured lists instead of diagrams.

Work Breakdown Structure (WBS)

Example; with details shown only for testing phase of project



Work Breakdown Structure (WBS)

- WBS summary
 - The WBS must be deliverable-oriented
 - Must ensure the business achieves its business goals
 - Level of detail should support planning and control
 - Developing the WBS should involve those doing the work; Technical tasks, Management tasks, & Support tasks
 - Learning cycles & Lessons learned can support the development of a WBS
- With WBS complete - now in a position to do more accurate estimation

Estimation

- What happens when estimates are wrong?
 - Project behind schedule
 - Low-quality deliverables
 - Over budget
 - Missing functionality

Estimation

- What else is affected?
 - Staff frustration
 - Damaged project management reputation within the business, and project management's self confidence takes a hit
 - Organization's reputation suffers
 - Competitors reputation possibly enhanced
 - Lapse back into a 'code it and fix it' mentality

Size Estimation

- Most authorities recommend developing size estimates first and then deriving other estimates from them:

LOC – Lines of code – could include comment lines, data declarations & so on

SLOC: Source lines of code or NCSL (Non-Commentary Source Lines) but includes data declarations???

KAELOC (K Assembler Equivalent Lines of Code) – to normalise lines of code from different computer languages

- Lines of code estimations most widely used but have been heavily criticised as *complexity* needs to be evaluated as well - So other approaches include function points and algorithmic models
- Authorities also strongly recommend using historical data but this needs to be done with care

Estimation

- Example: project data efforts in work-months per phase

Project	Design		Coding		Testing		Total	SLOC	Productivity SLOC/month
	wm	%	wm	%	wm	%			
A	3.9	23%	5.3	32%	7.4	45%	16.7	6050	362
B	2.7	12%	13.4	59%	6.5	29%	22.6	8363	370
C	3.5	11%	26.8	83%	1.9	6%	32.2	13334	414
D	0.8	21%	2.4	62%	0.7	18%	3.9	5942	1524
E	1.8	10%	7.7	45%	7.8	45%	17.3	3315	192
F	19	28%	29.7	44%	19	28%	67.7	38988	576
G	2.1	21%	7.4	74%	0.5	5%	10.1	38614	3823
H	1.3	7%	12.7	66%	5.3	27%	19.3	12762	661
I	8.5	14%	22.7	38%	28.2	47%	59.5	26500	445
						Overall		249.3 153,868	617

Estimation

- Example Continued:
 - If the project leaders for projects A & D had correctly estimated the source lines of code (SLOC) and then used the average productivity of the organisation to calculate the effort needed to complete the projects how far out would their estimates have been from the actual effort?

Project	Estimated work-months	Actual	Difference
A	$6050/617 = 9.80$	16.7	6.90
D	$5942/617 = 9.63$	3.9	-5.73

There would be an under-estimate of 6.9 work-months for project A and an over-estimate of 5.7 for project D

Historical data should be collected and used when estimating - but with great care!

Estimation: Common Techniques

- Common Techniques
 - *Top-down* – Where an overall estimate is formulated for the whole project and then is broken down into the effort required for component tasks
 - *Bottom-up* – Where component tasks are identified and sized and these individual estimates are aggregated

Estimation Summary

- Software projects differ in numerous characteristics
- A technical plan clarifies some of the specific project characteristics
- A work breakdown structure logically decomposes the work to be performed
- Effort estimation is complex
- Various techniques exist to perform realistic development effort estimation
- Previous data about completed projects is mandatory for better project estimation but should be used with care

Next Lecture

- Project Planning:
 - Scheduling

Today's Lecture

Software Project Planning Part 2

Software Project Planning

Major Phases:

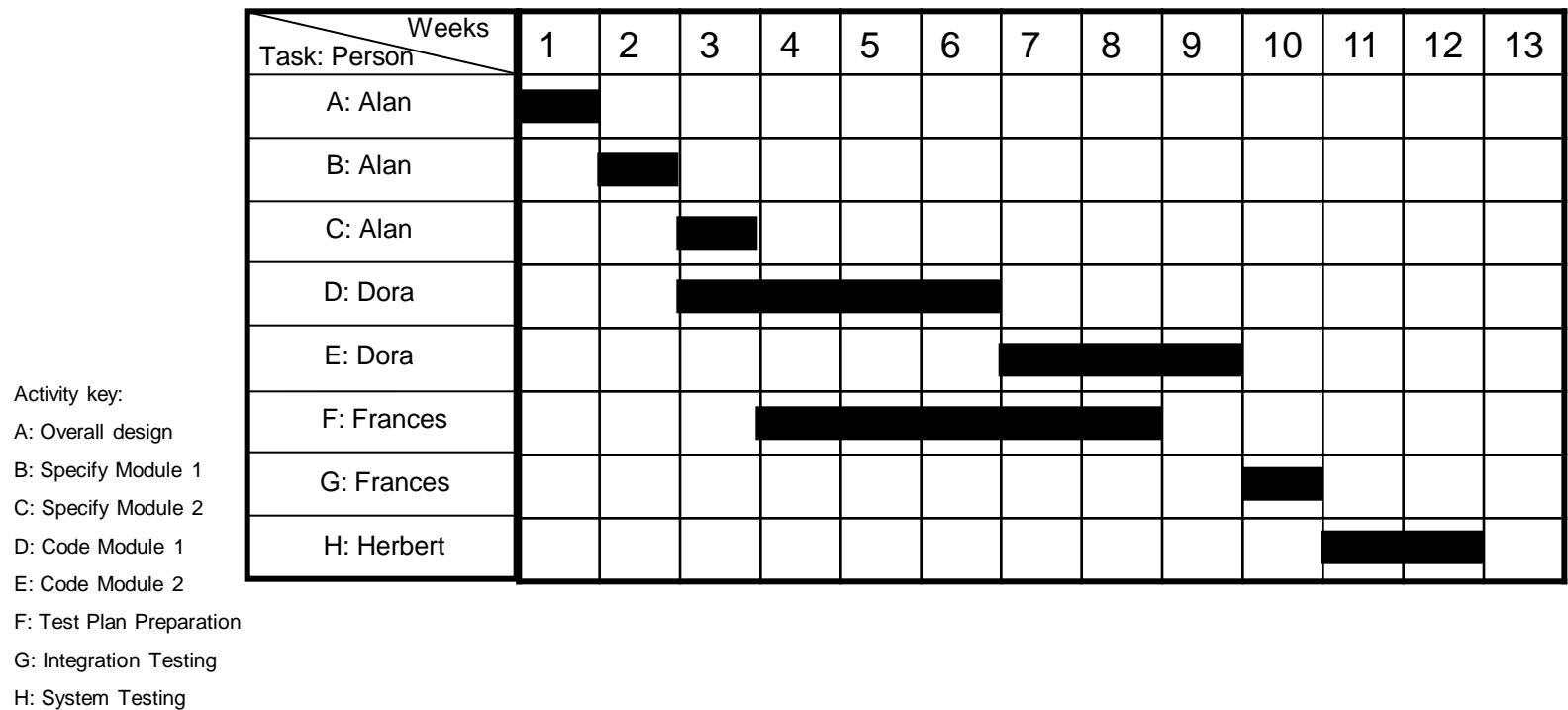
- Project Evaluation
 - Technical Analysis
 - Strategic Analysis
 - Cost/Benefit Analysis
 - Risk Profile Analysis
- Project Approach
 - Technical Plan
 - Lifecycle Model
- Estimation
 - Size & Effort
 - Application of Techniques
- **Scheduling**
 - Network Planning
 - Critical Path Analysis
- Risk Assessment
- Project Resourcing

Sequencing and scheduling tasks & activities

- In the Work breakdown structure we saw how the project can be decomposed down to discrete activities – with associated deliverables. Now each WBS task needs to be scheduled
- Simplest representation of a project schedule is a bar (Gantt) chart

Gantt or Bar Chart

- For each task specify:
 - Start date
 - End date
 - Resource
- Tasks may be carried out in sequence or in parallel (depending on the task or resource constraints)



Gantt or Bar Charts

- Gantt Charts – Some advantages
 - Simple, easy to construct
 - Universal communications tool
 - Clear start and finish times
 - Good for displaying milestones and comparing actual progress with planned
 - Good for grouping similar activities
 - Very good for small projects

Gantt or Bar Charts

- Gantt Charts – Some disadvantages
 - 👎 Logic not clearly represented – in the example why the gap at week 9 (a dependency or has Frances gone on holiday?)
 - 👎 Can be unwieldy for projects with more than about 30 activities
 - 👎 Projects are often considerably more complex than can be communicated effectively with a Gantt chart
 - 👎 Do not show relationships between activities – in the example between activities B & D

Gantt or Bar Charts

- Gantt Charts – Limitations

👉 Simple projects may get away with using a bar chart for sequencing and scheduling but in practice on larger projects its better to separate them out – so we use network planning models

Network Planning Models

- Two best known network planning models are CPM (Critical Path Method) and PERT (Program Evaluation Review Technique).
- Generally follow two different approaches:
 - ‘activity-on-arrow’ visualising the project as a network where activities are drawn as arrows between nodes representing start and/or completion points

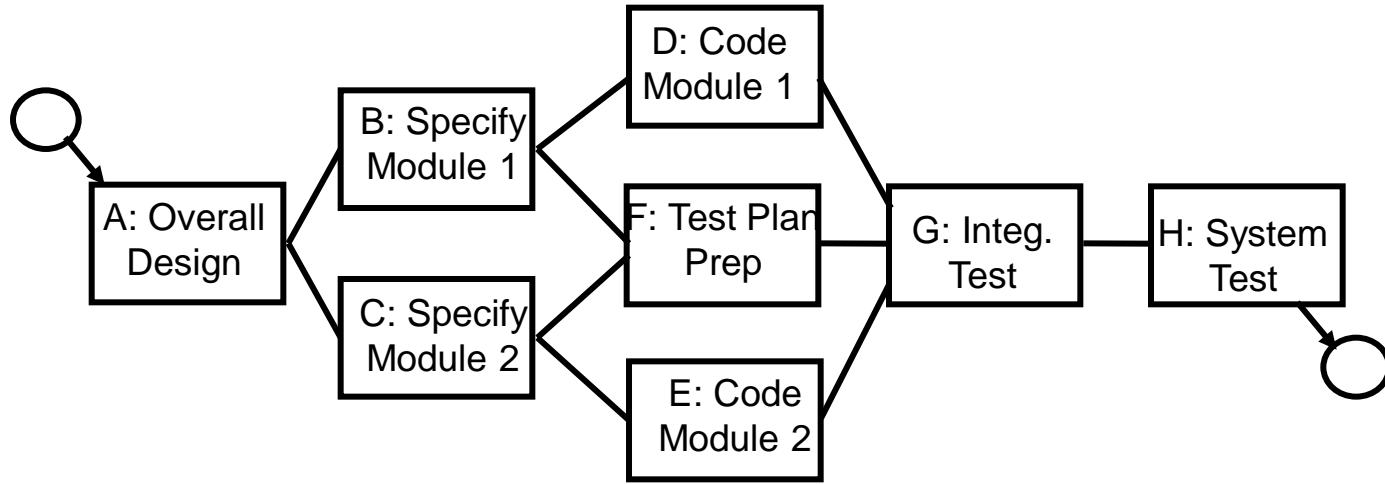


- ‘activity-on-node’ with activities represented by nodes and the arcs between nodes representing precedence – most popular



Formulating a Network Model

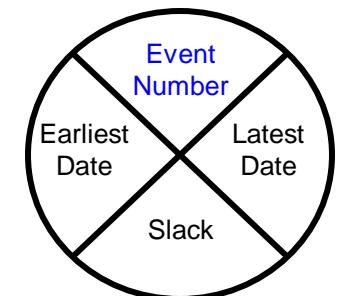
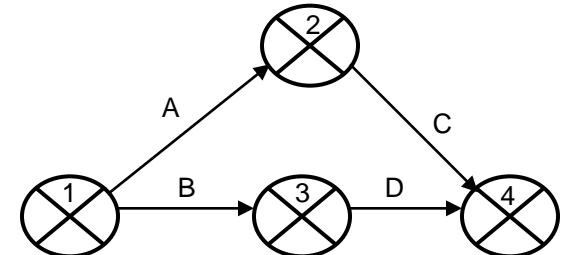
- In either case the first process is to use the WBS to establish a logical relationship between the activities using a *network diagram or graph*.



- We shall look at the Critical Path Method (CPM) Activity-on-Arrow technique in more detail

Critical Path Method

- CPM will tell us the shortest time in which project can be completed
- Activity-on-Arrow Networks
 - Activities are represented by links
 - Nodes represent events (activities starting or finishing)
 - Construction rules and conventions
 - Only one start node
 - Only one end node
 - Links have durations
 - Nodes have no durations (events)
 - Read left to right
 - Nodes are numbered sequentially
 - May not contain loops
 - May not contain dangles



Critical Path Method: Network Construction

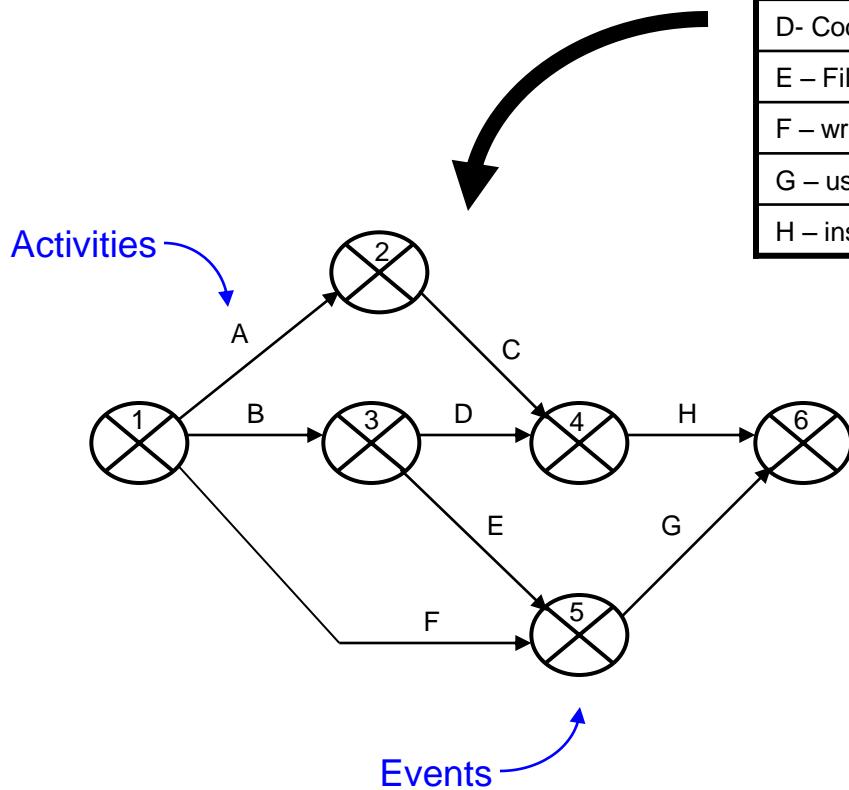
- **Activity-on-Arrow**

- For each activity we need an estimate of its duration and we then carry out a forward pass to calculate the earliest dates at which activities may commence and complete.
- Use the following Activity Logic table as an example:

Activity	Duration (weeks)	Precedents
A- hardware selection	6	
B- software design	4	
C - install hardware	3	A
D- Code and test software	4	B
E – File acceptance	3	B
F – write user manuals	10	
G – user training	3	E, F
H – install and test system	2	C, D

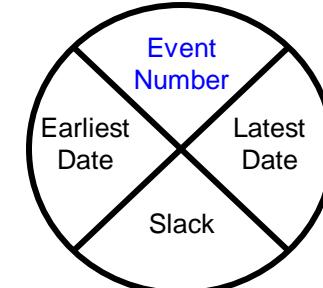
Activity-on-Arrow Networks

- Construct the following activity-on-arrow network from the precedence table provided:



Activity	Duration (weeks)	Precedents
A- hardware selection	6	
B- software design	4	
C - install hardware	3	A
D- Code and test software	4	B
E – File acceptance	3	B
F – write user manuals	10	
G – user training	3	E, F
H – install and test system	1	C, D

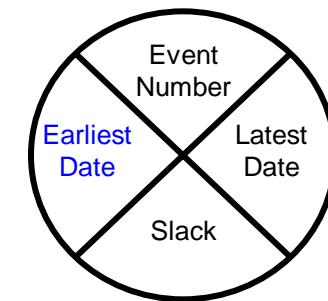
Common labelling convention



Activity-on-arrow

- Forward Pass to calculate *Earliest Date*
 - The *earliest date* for an event is the earliest finish date by which all prior activities upon which the event depends can be completed (i.e. earliest finish date = previous nodes' earliest date + activity duration) .

Where more than one activity terminates at an event take the latest of the set of earliest finish dates i.e least optimistic

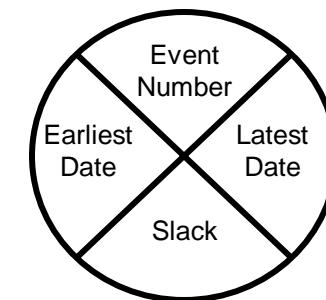
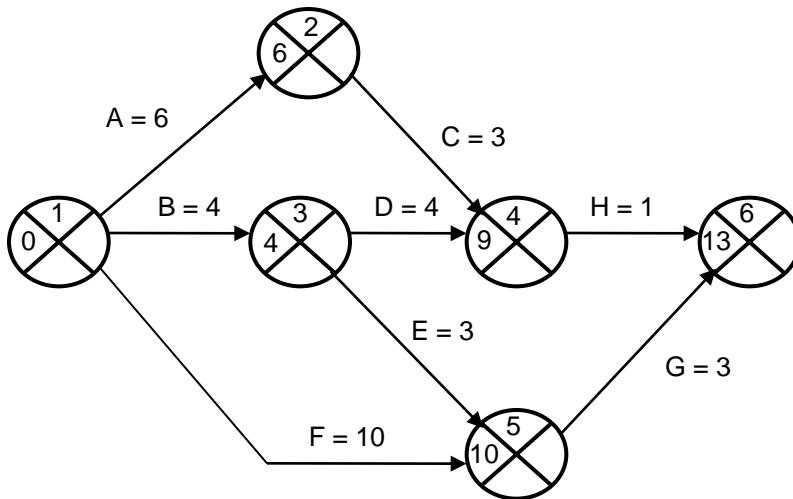


Activity-on-arrow

- Forward Pass to calculate *Earliest Date*
 - Activities A,B & F may start immediately so their earliest date for Event 1 is 0
 - A will take 6 weeks so the earliest date for Event 2 = 6
 - B will take 4 weeks so the earliest date for Event 3 = 4
 - F will take 10 weeks but we don't know yet if this is also the earliest date for Event 5 as we have not yet calculated Activity E
 - Activity E can start as early as week 4 and is forecast to take 3 weeks & so is planned to be completed at week 7
 - Earliest date for Event 5 is latest of Activities E (=7) & F (=10) and thus is 10
 - Earliest date for Event 4 is latest of Activities D (=8) & C (=9) and thus is 9
 - Earliest date for Event 6, completion of the project, is latest of Activities H (=10) & G (=13) and thus is 13

Activity-on-Arrow

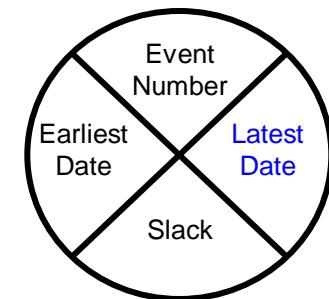
- Network after forward pass



Activity-on-arrow

- Backward Pass to calculate *Latest Date*
 - The *latest date* for an event is the latest start date for all the activities that may commence from that event (i.e. latest start date = succeeding nodes' latest date minus the activity duration).

Where more than one activity commences at an event, take the earliest of the set of latest start dates i.e. least optimistic

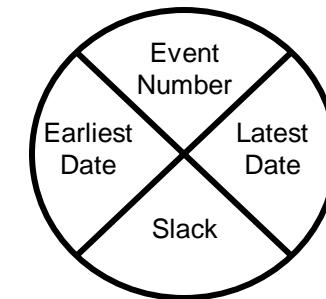
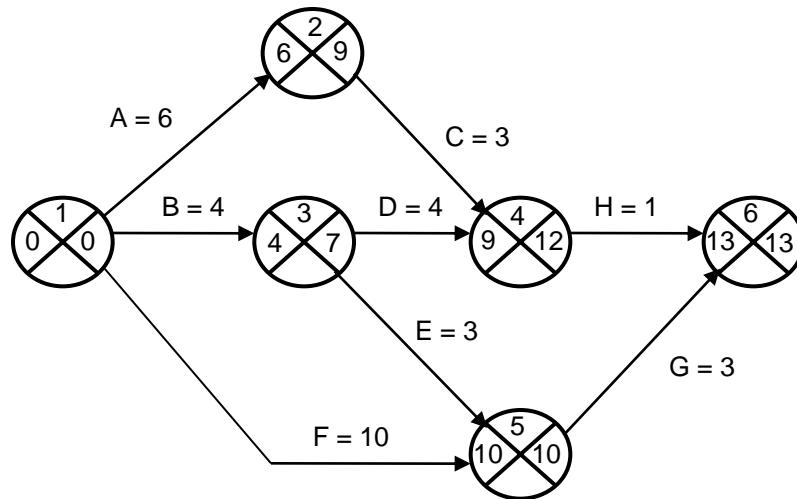


Activity-on-arrow

- Backward Pass to calculate *Latest Date*
 - Latest date for Event 6 = 13
 - Hence latest date for Event 5 = $13 - 3 = 10$ and for Event 4 = $13 - 1 = 12$
 - Latest date for Event 2 = $12 - 3 = 9$
 - Latest date for Event 3 = Earliest of:
 - Event 4 latest start date = $12 - 4 = 8$
Or
 - Event 5 latest start date = $10 - 3 = 7$ Thus latest date for Event 3 = 7
 - Latest date for Event 1 = Earliest of:
 - Event 2 latest start date = $8 - 6 = 2$
or
 - Event 3 latest start date = $7 - 4 = 3$
or
 - Event 5 latest start date = $10 - 10 = 0$ Thus latest date for Event 1 = 0

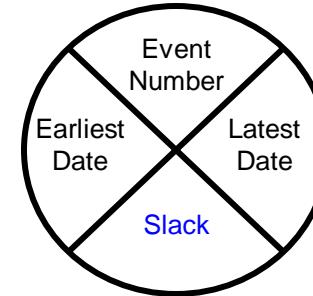
Activity-on-Arrow

- Network after backward pass



Activity-on-Arrow

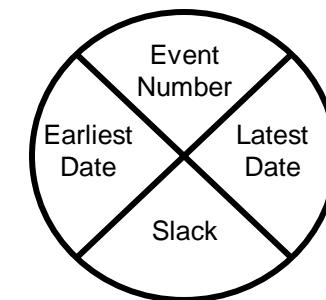
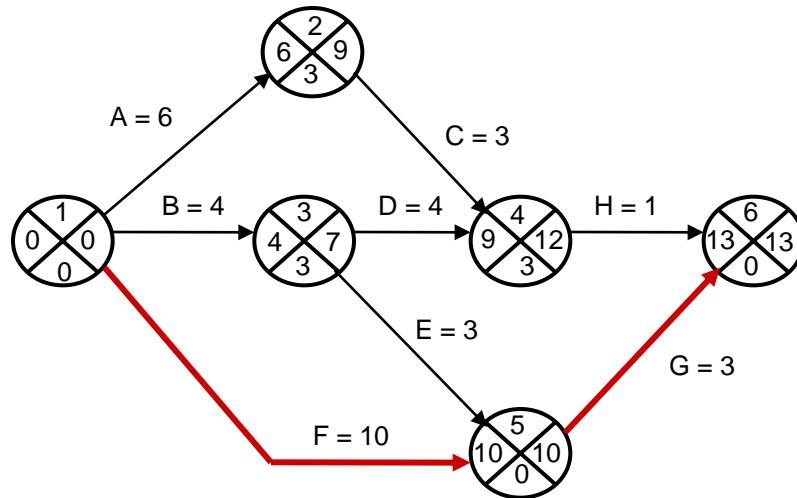
- Slack & Critical Path identification
 - The *slack* for an event is calculated as the difference between the earliest date and the latest date for an event. Measures how late the event may be without affecting the end date of the project.



- The *critical path* is the path joining all nodes with a zero slack.

Activity-on-Arrow

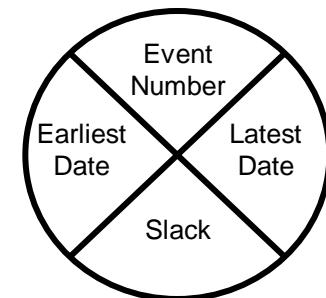
- Slack calculated and Critical Path identified



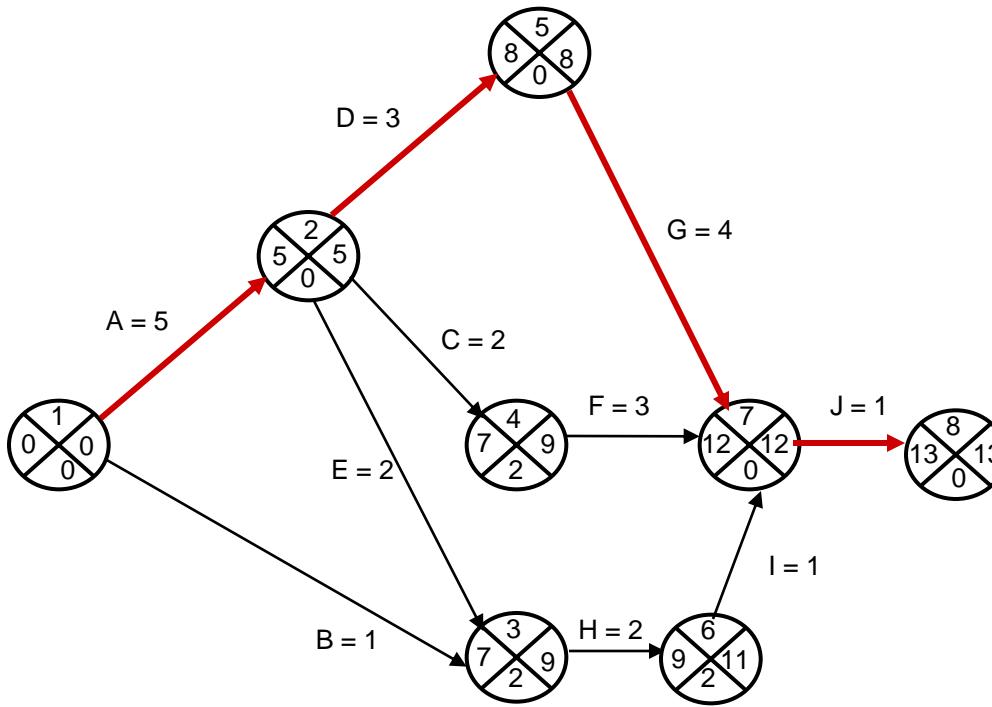
Exercise

Using activity-on–arrow notation identify the earliest completion date and critical path for the following network

Activity	Duration (months)	Precedents
A- Product Design	5	
B- Market Research	1	
C- Production Analysis	2	A
D- Product Model	3	A
E – Sales Brochure	2	A
F – Cost Analysis	3	C
G – Product Testing	4	D
H – User Training	2	B,E
I – Pricing	1	H
J – Product Report	1	F,G,I



Exercise



Software Project Planning

Major Phases:

- Project Evaluation
 - Technical Analysis
 - Strategic Analysis
 - Cost/Benefit Analysis
 - Risk Profile Analysis
- Project Approach
 - Technical Plan
 - Lifecycle Model
- Estimation
 - Size & Effort
 - Application of Techniques
- Scheduling
 - Network Planning
 - Critical Path Analysis
- Risk Assessment
- Project Resourcing

Project Risk

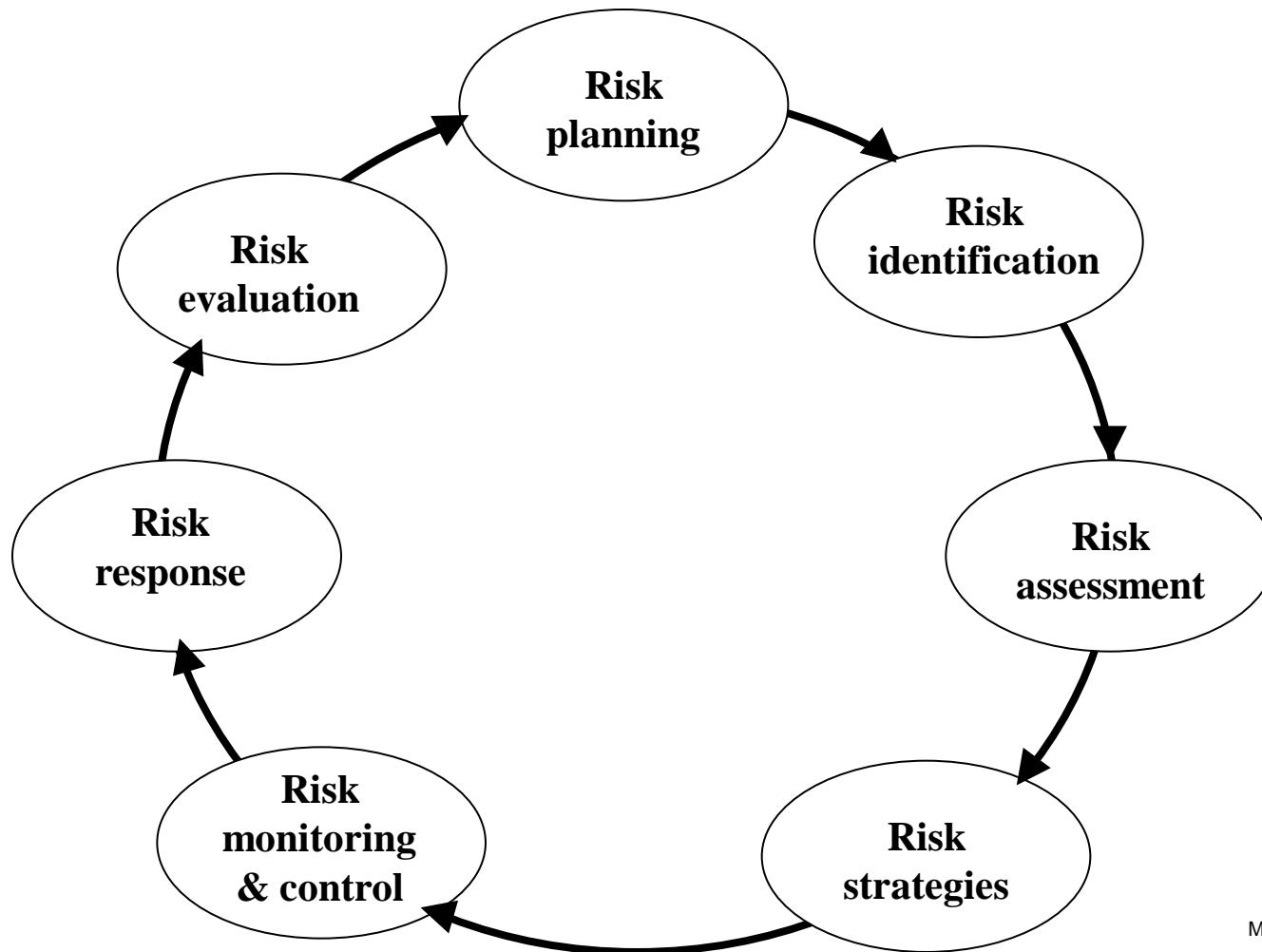
- Three major categories of risks can cause the project to overrun its timescale or budget:
 - Those caused by the inherent difficulties of estimation
 - Those due to assumptions made during the planning process
 - Those of unforeseen (or unplanned) events

Risk Assessment

- In all cases we need to:
 - Identify the risk
 - Quantify the impact
 - Plan responses
 - Monitor the risk over the project lifetime
- Models available to help us do that

Risk Management: Process

For example, Marchewka suggests a seven stage process that emphasizes the need for a proactive approach to managing risk.



Marchewka, 2003

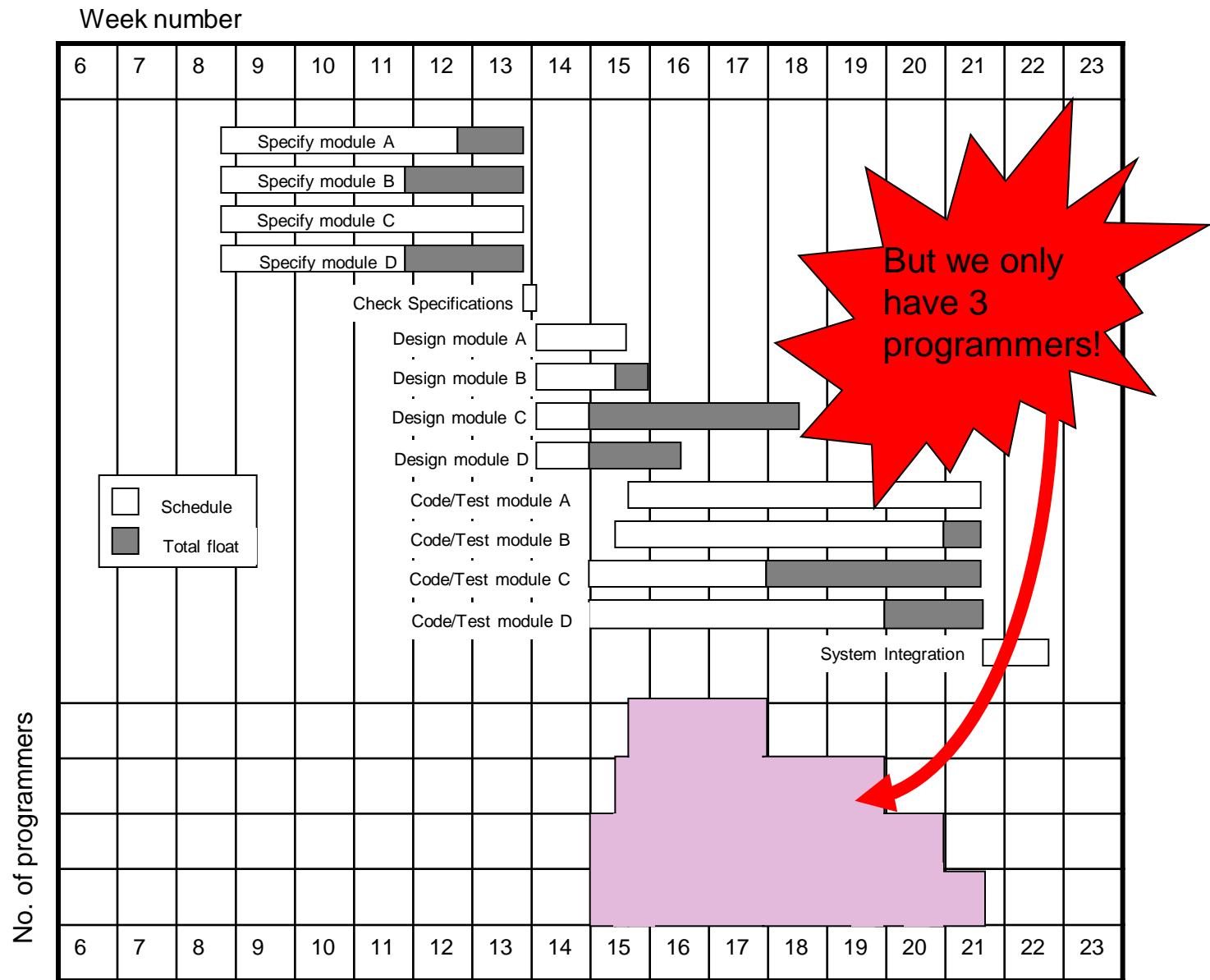
Risk Mitigation

- Risk mitigation can take the form of:
 - Risk transfer - insurance
 - Risk reduction – take precautions
 - Risk acceptance – do nothing
 - Risk avoidance – don't go in the water

Project Resourcing

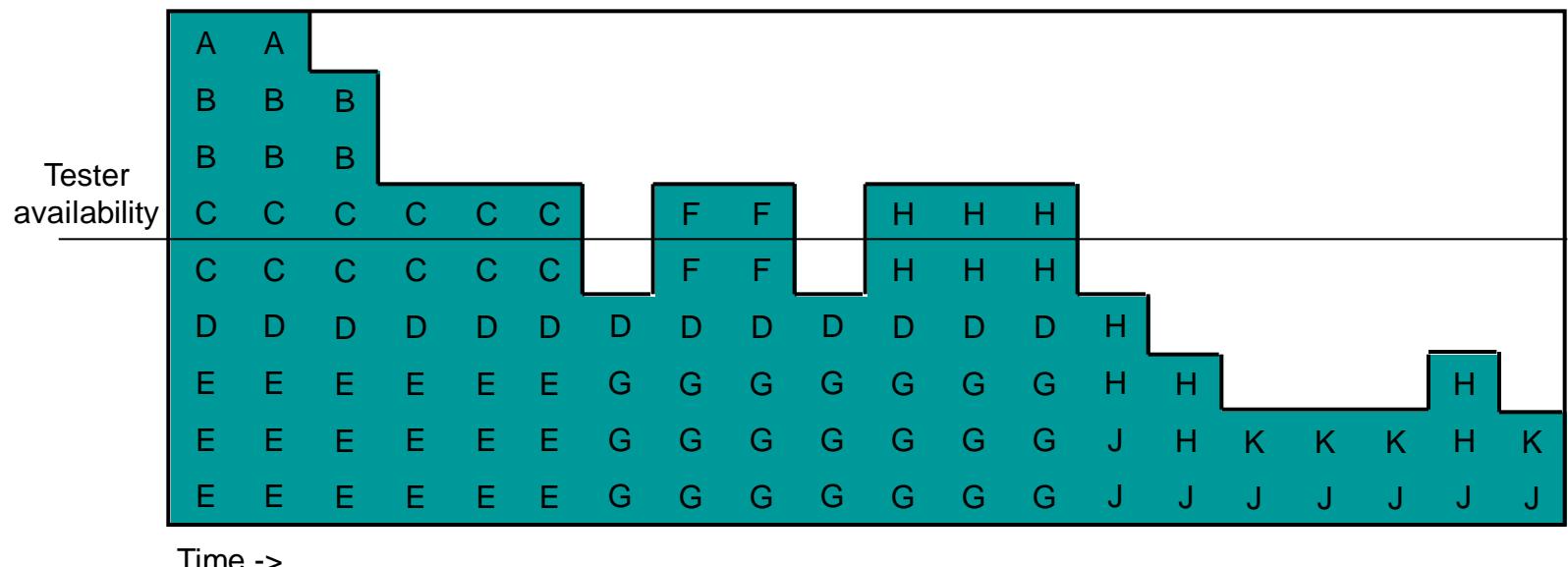
- Closely associated with risk is project resourcing
- Resource availability and scheduling provide additional constraints to a project
- For instance in the following scenarios what do we do if we have limited resources?

Project Resourcing



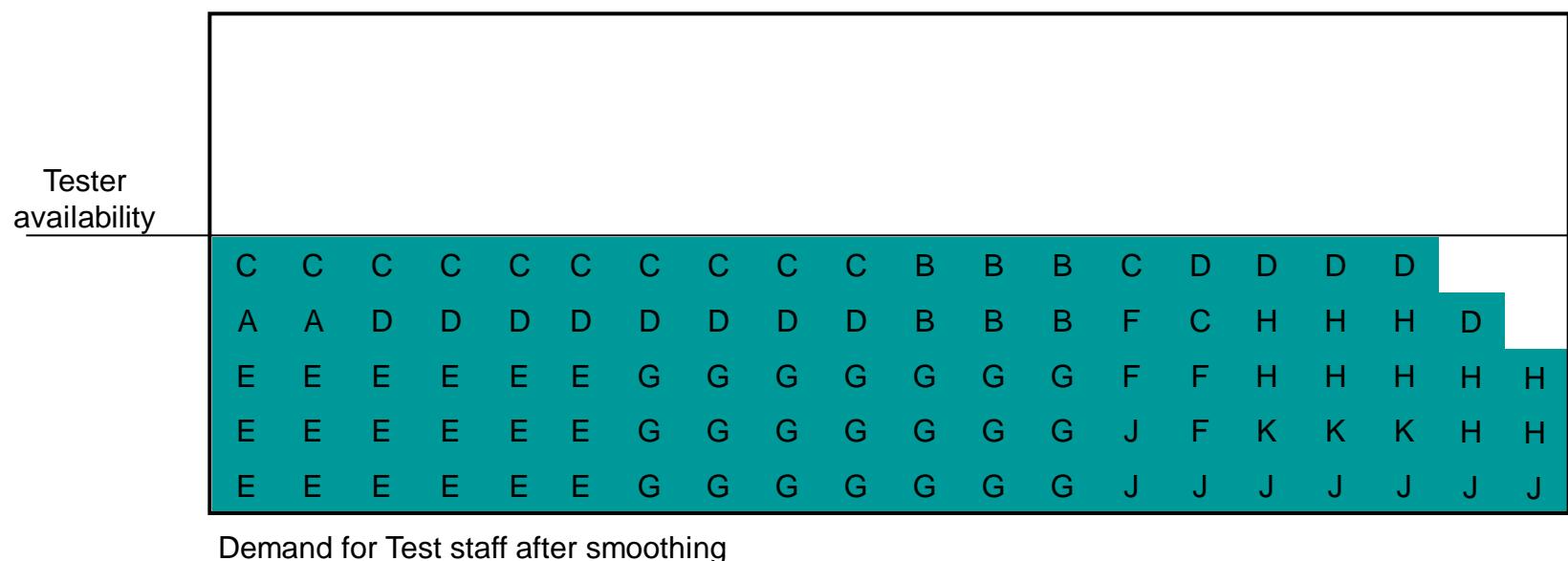
Project Resourcing

- A -> J are tasks testers are working on. 1 on A, 2 on B and so on. But we only have 5 testers where 9 are sought
- So what can be done?



Project Resourcing

- By delaying start dates of some activities e.g. B and splitting others e.g. C & D a resource histogram can be smoothed to contain resource demands at reasonable levels
- Where non-critical activities can be split they can provide a useful way of filling troughs – but with software this can be difficult and may increase time they take
- Other options: increase resources or change work practices



Additional planning required

- Scope management
- Cost management
- Quality management
- Procurement

Next

- Software Process



IS4415 Advanced Tools and Methods for Business Applications

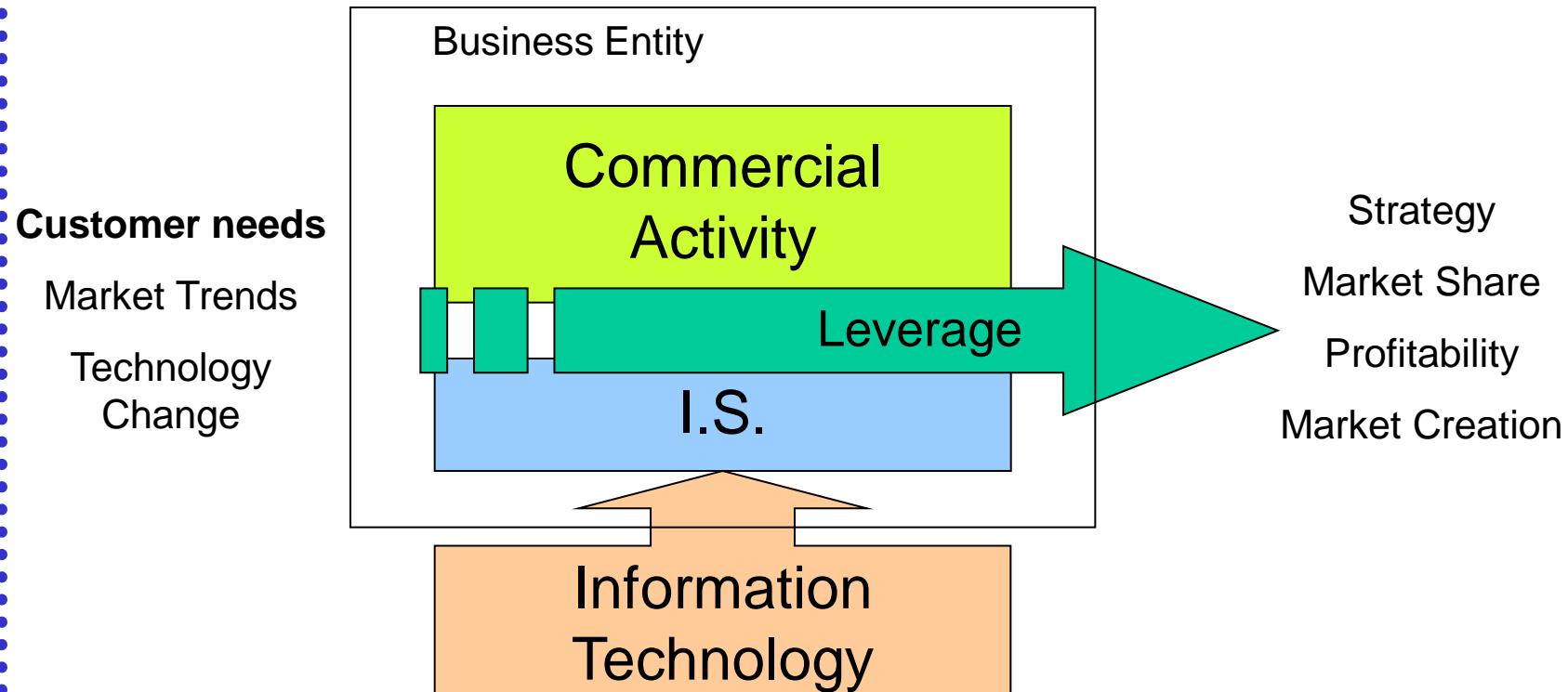
Previously...

- Range of specialist IS topics:
 - Software lifecycle models
 - Agile development methods
 - Software testing
 - Software project planning
 - Software project management

This Lecture...

- Back to Basics: Business & Information Systems
- The Business of Software Development
- A Brief History of Process & Process Improvement
- Process Models for Software

BIS Simple Model



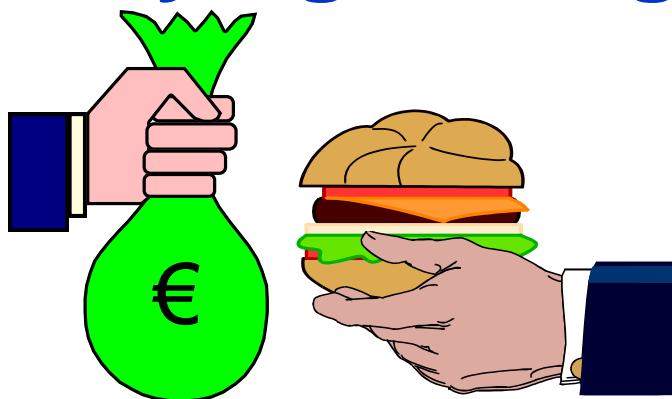
A Simple Question.....



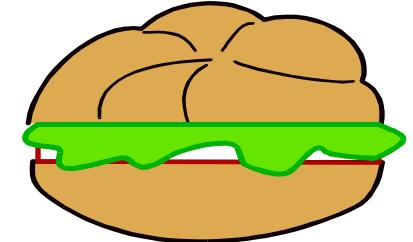
Another Simple Question.....



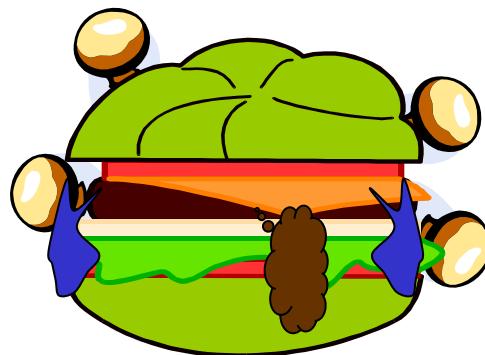
Buying a Burger or a Battleship!



Value for Money



All Requirements Met



High Quality Product



On-time Delivery

So...



But we tend to forget...

**Software is a product that people buy
too!!!**

Either on its own or as an integral part of another
product offering or service (recall our BIS model)

&

It's subject to the same rules

Customers demand what?

High quality products?

European Space Agency's Ariane-5 rocket: untested guidance software – had to blow it up

Mars Climate Orbiter: mixed up pounds & kilos – lost
Intel Pentium Processor: Bugs galore!

Therac-25 Radiation Therapy Machine: Patient overdoses & deaths

Plus: Y2K bugs, Banking errors, Voting machines that don't work, Aircraft disasters, Telecommunication failures, Power outages

Customers demand what?

On-time delivery & value for money?

PPARS Payroll System: 25 times the original estimate,
and four years behind schedule

(the original cost estimate for the project in 2002 was
€8.8 million, but the final cost was likely to be €231
million!!)

E-Voting machines: Can be hacked & reprogrammed
with fraudulent votes (or can be reprogrammed to play
chess!!) – Cost to date €53million & were hanging
around for 10 years until 2012!!

Customers demand what?

All requirements met?

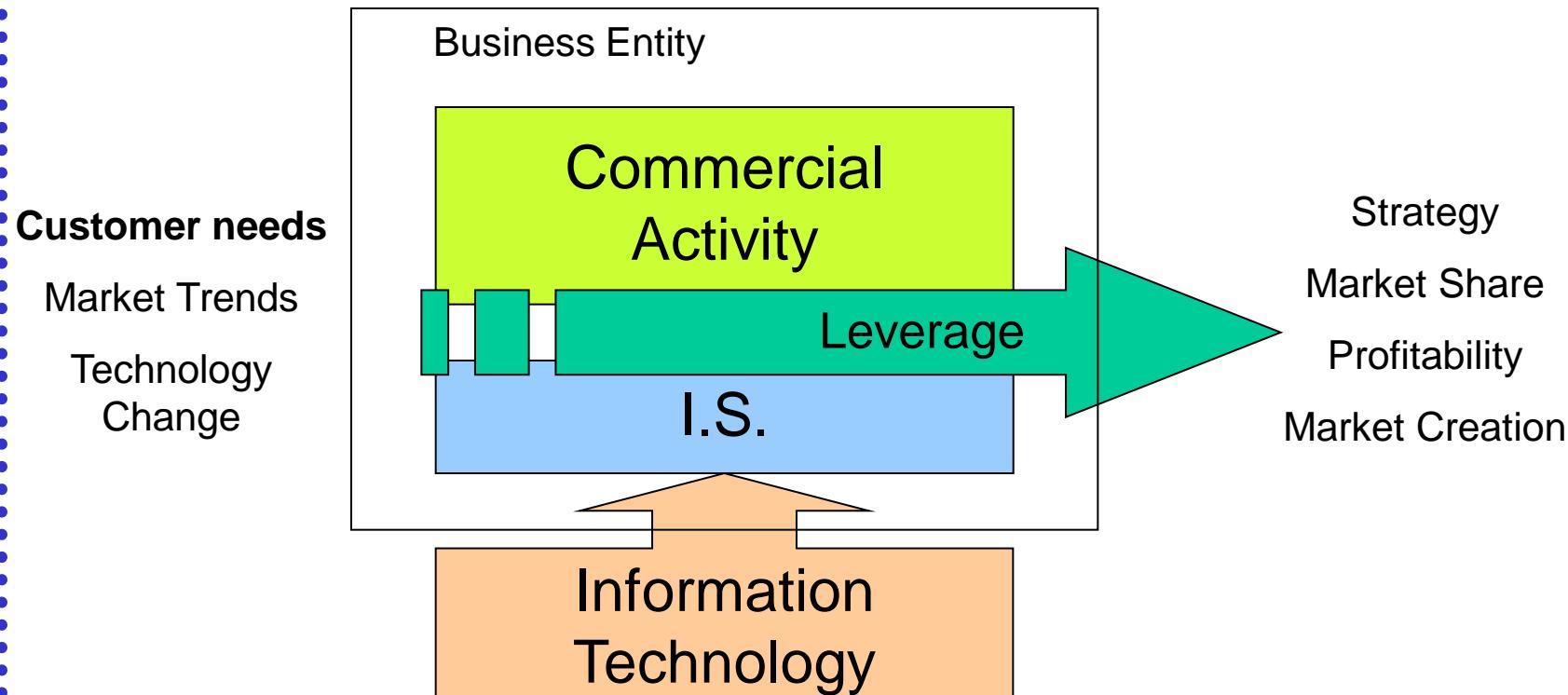
The Institute of Electrical & Electronic Engineers (IEEE) reported – “Three-quarters of large systems fail to operate as intended or, indeed, can be used at all”

Software Failure

These are damning indictments.

The continuing failure to produce high quality software: on-time, on-budget, with all requirements, has huge implications for us & the leverage we expected to gain

BIS Simple Model



Why is this happening?

It's happening because software development has some Essential difficulties:

Complexity - software is the most complex human artefact to construct.

Conformity - software usually has to interface with existing systems

Changeability - software is easier to change than hardware.

Invisibility - software constructs are not visible entities

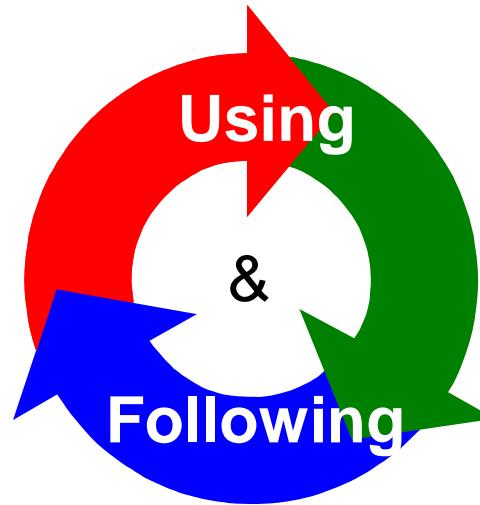
(‘No Silver Bullet - Essence and Accidents of Software Engineering’,
Fred Brooks, Computer, 1987)

Producing Software

To produce software 3 elements are necessary...

1. People

2. Technology



3. Process



SOFTWARE PRODUCTS

Hence, if software products are defective there must be something wrong with one (or more) of the 3 elements

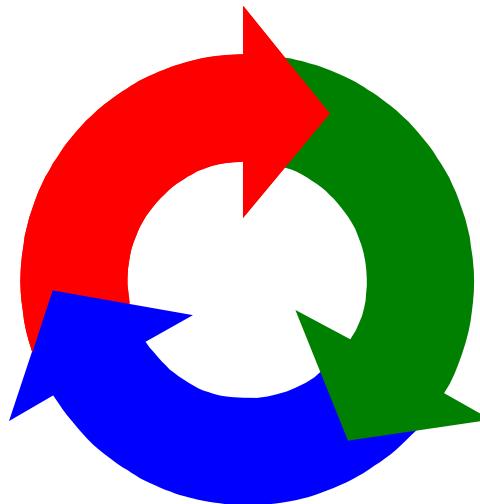
So which element is failing?

People

Less well educated?

Not as intelligent?

Is our understanding of organisations & group behaviours wrong?



Technology

Is technology failing us?

Is it less dependable & not meeting needs?

More difficult to use or less user-friendly?

Process

Have the development processes & controls we use to create software kept pace with other advances?

So which element is failing?

For many experts the processes we use for software development are the weak link in the chain:

We simply fail to apply the same degree of rigour and control to software creation as we do to other product developments

We have failed to understand what a ‘Software Process’ actually means...

A Software Process?

The idea of a work Process -

coincided with the invention of the ‘Job’.

In the 18th century people didn’t have ‘jobs’ -
they did work

The job was created in the 19th century: a field of work was partitioned into a series of discrete operations called ‘jobs’ – & these linked together into a work process

(William Bridges - ‘JobShift’, 1996)

Work Processes

An Early Example of a work process
“The Wealth of Nations” — Adam Smith (1776)

Smith called the principle “the division of labour”

He described how a number of specialised workers, each performing a single step in the manufacture of a pin could make far more pins in a day than the same number of workers engaged in making whole pins.

Making a pin....

Smith wrote:

“One man draws out the wire, another straightens it, a third cuts it, a fourth points it, a fifth grinds it at the top for receiving the head; to make the head requires two or three distinct operations; to put it on is a peculiar business, to whiten the pins is another; it is even a trade by itself to put them into the paper”

Quality control?

A single craftsperson was obviously responsible for the quality of their work (wheelwrights, sailmakers, weavers, carpenters, etc.)

But with the advent of the ‘job’ & the work process where did ultimate responsibility for product quality reside?

Product Quality?

In the factories and industries of the 1800's:

“Experienced men knew the job - experienced foremen and operators knew what the machines could do ... inspectors could find all the defects ... there was no need for records, sampling or measures of quality - experienced inspectors and supervisors had their hands on the reins”

Joseph Juran, “Pioneering in Quality Control”, 1961

In other words...

People knew how things were done; the process

They knew what that process was expected to deliver

They knew it's strengths & weaknesses & where
defects could be found

They knew what the process was *capable* of & how to
improve it

The Software Process

IMPORTANTLY: There is a process for producing good software - just as there are processes for producing good cars, ensuring good air traffic control, or serving good food

- and that process can be understood, modeled, & improved.

Software Process Models

There are many different models for software process improvement, assessment and evaluation.

Some of the better known models include:

The Capability Maturity Model (CMM)& (CMMI)

SPICE

ISO 9000

TickIT

Trillium

Bootstrap

IEEE

Malcolm Baldridge

Software Process Models

The improvement and assessment model that we shall cover in greater detail is the CMMI - but the original CMM model is still widely used

The CMM & CMMI were developed by the Software Engineering Institute (SEI) of Carnegie Mellon University in the U.S.

Today's Lecture.....A Summary

- Provide a foundation for subsequent lectures
- Business Rationale for S/W Process Improvement
- A Brief History of Process & Process Improvement
- Process Models for Software

Winter Exam

- January 13th Winter Exam

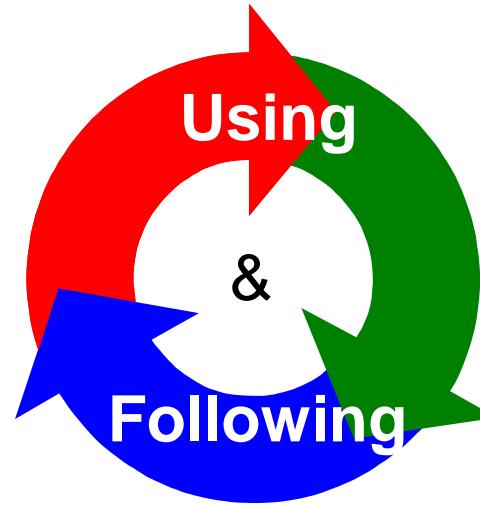
Today's Lecture...

- Software Process Improvement 1

Producing Software

To produce software **3** elements are necessary...

People



Technology

Process



SOFTWARE PRODUCTS

If software products are defective there must be something wrong with one (or more) of the 3 elements.

Software Process Improvement

There are many different models & methodologies for improving the software process

We are going to take a closer look at 3 of them:

- Lean
- Maturity Models
- 6-Sigma

Software Process Improvement

Each has a distinct focus:

- Lean
 - The **elimination of waste**, where waste is defined as having no added value to the software product or service being provided
- Maturity Models
 - Guiding software organisations' improvement efforts by providing normative references that facilitate **continuous improvement** in stages
- 6-Sigma
 - The **removal of variability** within a process, where variability is defined in terms of exceeding statistically calculated limits

Software Process Improvement

LEAN: Origins

- Originated in the manufacturing domain
 - Based upon the Toyota Production System (TPS) originally called Just-in-Time Production the focus is on consistent delivery without stress, overburdening processes or excess inventory.
 - Basic premise: The expenditure of resources for any purpose other than to add customer value is waste and should be removed.

Software Process Improvement

LEAN: Origins

- Principles
 - Continuous Improvement (Kaizen)
 - Respect for People (Trust & teamwork)
 - Long-term Philosophy (Over short-term financial goals)
 - The right process will yield the right results (Standardise tasks where possible)
 - Develop people (Grow the organisation)
 - Going after root problems (Reflection and organisational learning)

Software Process Improvement

LEAN: Elimination of Waste is Paramount

- Waste is considered in the broadest terms – 3 Types:
 - **Muri** (overburden, unreasonableness) – Is the type of waste that is imposed on the organisation because of **poor design of work processes or practices**.
 - Overcoming *muri* means proactively designing out of the process work that can be avoided.

Software Process Improvement

LEAN: Elimination of Waste is Paramount

- Waste is considered in the broadest terms – 3 Types:
 - **Mura** (unevenness, inconsistency) – Is the type of waste that occurs as a result of **poor implementation or production irregularities**.
 - *Mura* can be avoided by creating a ‘pull’ system to keep inventory levels at optimum (e.g. a washing machine production line will pull frames, from the paint shop, which will pull from the welding shop, that will pull from Stamping, that will pull from the sheet metal suppliers, etc.)

Software Process Improvement

LEAN: Elimination of Waste is Paramount

- Waste is considered in the broadest terms – 3 Types:
 - **Muda** (wasteful, unproductive) – Is discovered after the work process is in place and usually seen through **variation in output**.
 - Dealing with *muda* is reactive and causes of variation need to be eliminated by considering the *muri* & *mura* of the system.

Software Process Improvement

LEAN: Waste versus Value

- **Value**
 - Value is defined as any activity the customer is willing to pay for.
 - **Waste (muda) is categorised as:**
 - Defects
 - Overproduction (Overprovisioning)
 - Waiting
 - Non-value Added Processing
 - Transportation
 - Excess inventory
 - Excess motion
 - Unused employee knowledge
- 
- 
- Original 7 muda
- Added 8th waste

Software Process Improvement

Suppose we were manufacturing Electric Kettles

Waste Type	Examples	Results
Defects	<ul style="list-style-type: none">Faulty switchesLeaks	Rework, additional costs
Overproduction (Overprovisioning)	<ul style="list-style-type: none">Two power checksSilver plated lid	Increased costs, wasted resources
Waiting	<ul style="list-style-type: none">Slow parts deliveryDelayed responses	Missed delivery dates, overtime pay
Non-value added processing	<ul style="list-style-type: none">Maintenance reports prepared for director	Miscommunication, wasted time
Transportation	<ul style="list-style-type: none">Travel to sort out distributor issues	Higher operational expenses
Excess Inventory	<ul style="list-style-type: none">Excess steelUnused machinery	Additional costs
Excess motion (Repetition)	<ul style="list-style-type: none">Recurring problems	Rework, wasted resources
Unused employee knowledge	<ul style="list-style-type: none">Better wiring methodBetter process suggestions	Loss of talent, continuing high production or maintenance costs

Software Process Improvement

LEAN & Software/IT

- Relatively new idea
- IS/IT becoming increasingly integral to the business & in some cases (e-Business) it is the business
- Significant challenges in IS/IT domain (re: Brook's essential difficulties)
- Longer term process with greater institutionalisation difficulties (greater variability in IS processes)

Software Process Improvement

Exercise: Examples of Waste in Software Dev.

Waste Type	Examples
Defects	
Overproduction (Overprovisioning)	
Waiting	
Non-value added processing	
Transportation	
Excess Inventory	
Excess motion (Repetition)	
Unused employee knowledge	

Software Process Improvement

Exercise: Examples of Waste in Software Dev.

Waste Type	Examples	Results
Defects	<ul style="list-style-type: none">• Coding errors• Design errors	
Overproduction (Overprovisioning)	<ul style="list-style-type: none">• Gold-plating• Over elaboration	
Waiting	<ul style="list-style-type: none">• Handover delays• Critical resource bottlenecks	
Non-value added processing	<ul style="list-style-type: none">• Misguided metrics• “All here” e-mails	
Transportation	<ul style="list-style-type: none">• Traveling to fix bugs at customer site	
Excess Inventory	<ul style="list-style-type: none">• Under utilized hardware• Multiple databases	
Excess motion (Repetition)	<ul style="list-style-type: none">• Recurring problems• Ergonomic issues	
Unused employee knowledge	<ul style="list-style-type: none">• High staff turnover• Opportunity loss	

Software Process Improvement

LEAN IT: Examples of Waste in Software Dev.

Waste Type	Examples	Results
Defects	<ul style="list-style-type: none">• Coding errors• Design errors	Rework, additional costs, unreliable software
Overproduction (Overprovisioning)	<ul style="list-style-type: none">• Gold-plating• Over elaboration	Increased costs, wasted resources
Waiting	<ul style="list-style-type: none">• Handover delays• Critical resource bottlenecks	Increased costs, schedule delays, inadequate quality
Non-value added processing	<ul style="list-style-type: none">• Misguided metrics• “All here” e-mails	Miscommunication, wasted time, distractions
Transportation	<ul style="list-style-type: none">• Traveling to fix bugs at customer site	Higher operational expenses
Excess Inventory	<ul style="list-style-type: none">• Under utilized hardware• Multiple databases	Increased costs, higher energy needs, extra training
Excess motion (Repetition)	<ul style="list-style-type: none">• Recurring problems• Ergonomic issues	Lost productivity
Unused employee knowledge	<ul style="list-style-type: none">• High staff turnover• Opportunity loss	Higher training costs, continued high operational costs

Software Process Improvement

LEAN IT: Waste effects

- All of these sources of waste can have knock-on effects for product or service delivery to customers and/or the bottom line
- Sources of waste tend to ‘gang-up’ on you creating multi-faceted problems & knock-on effects
 - For instance: Trying to fix processing errors (Defects) in a software deliverable may tie up a test lab for another group trying to get in to use it (Waiting). The software eventually ships but excessive support calls start coming into the call centre from the customers (Excessive motion). Eventually engineers are dispatched to key customers to fix the problems on site (Transportation) and, a bunch of sales people are dispatched as well, even though it’s unclear what they are going to do (Transportation and Overprovisioning)

Software Process Improvement

LEAN IT: Implementation – 5 main principles

1. Understand what **value actually means**. Specify value from the end customers perspective
 - Customer can be external or internal
 - Customer seek specific items, of defined quality, at defined times, at specific price
 - Needs to be done periodically as customer change their minds



Software Process Improvement

LEAN IT: Implementation – 5 main principles

Question: What are some of the ways customers could perceive value in a computer game?

Software Process Improvement

LEAN IT: Implementation – 5 main principles

2. Understand the value stream of the product and associated information. Perform value stream mapping
 - The map is a visual representation of the steps, delays and information flows in getting the product developed and delivered to the customer
 - Analyse the maps to identify those steps or operations that do not create value and then eliminate them



Software Process Improvement

LEAN IT: Implementation – 5 main principles

3. Understand the **flow of the process**. This is the *mura* concept which refers to unevenness.
 - This is understanding the dynamic performance of the system. Focussing on the muda (waste) alone may result in missing opportunities for improvement that would result from examining the flow of the value stream.
 - This is referring to tightening up the value creating steps in the process.



Software Process Improvement

LEAN IT: Implementation – 5 main principles

4. Establish a ‘**Pull** (demand) system’. A pull system is one that is request based as opposed to a push system where a supplier tries to estimate demand up front and then manufactures to meet that demand.



Software Process Improvement

LEAN IT: Implementation – 5 main principles

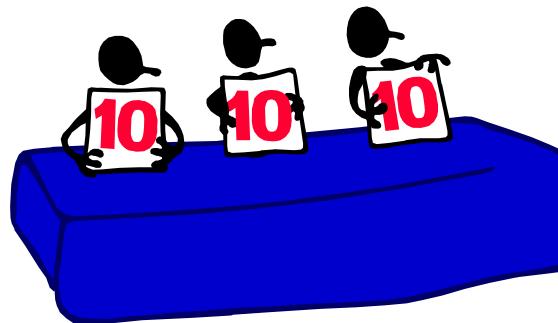
Question: What do you think are the main advantages of ‘Pull’ systems over ‘Push’ systems?

- Push systems will frequently result in large inventories of parts waiting to be used in the process. Lean, of course, regards this as waste.
- Pull systems only request components when they are required and have their source in the initial customer request. So keeping small inventories is acceptable.
- This is the basis of Just-in-Time manufacturing. Push systems are often referred to as a Just-in-Case approach.

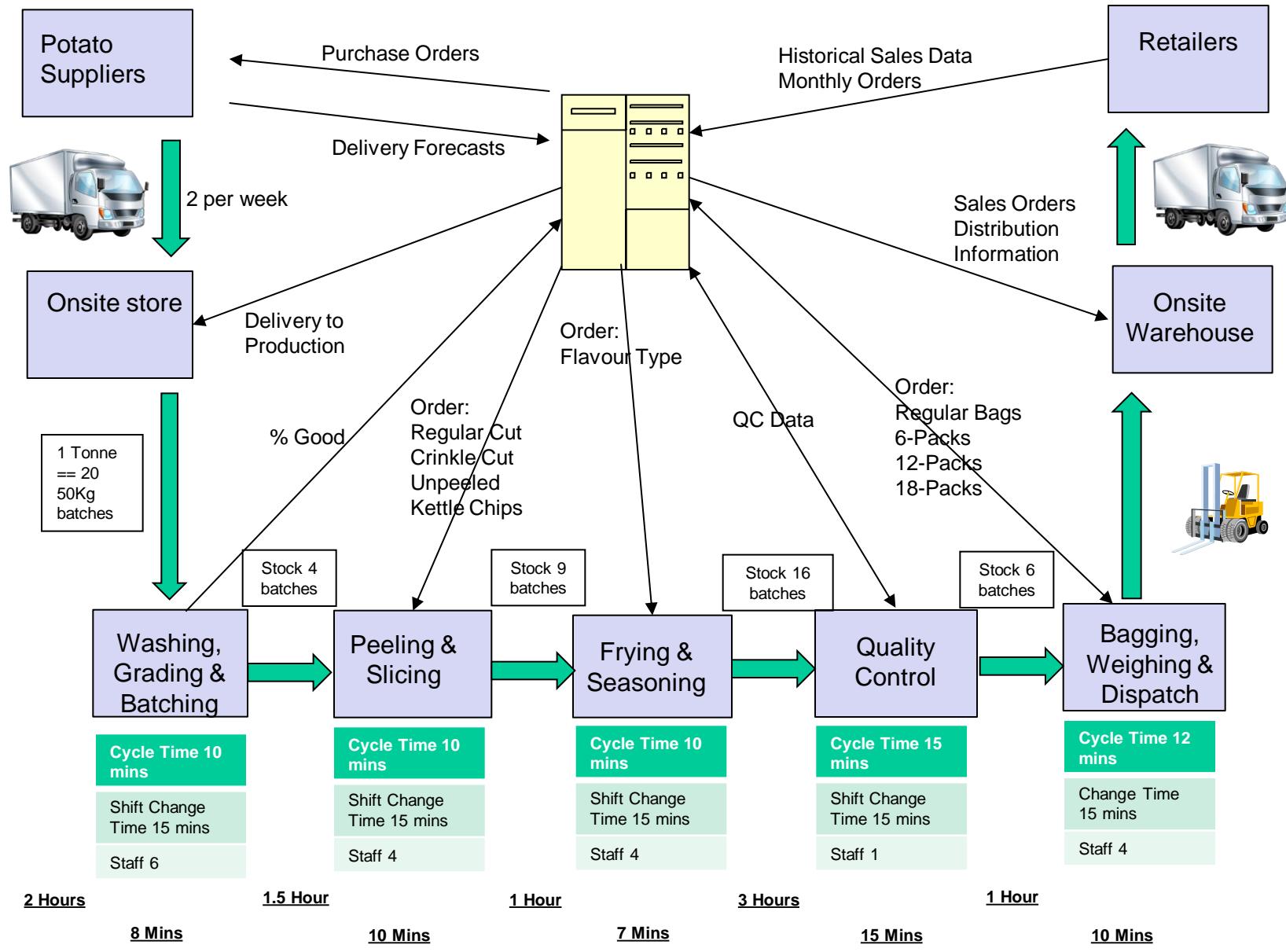
Software Process Improvement

LEAN IT: Implementation – 5 main principles

5. Seek perfection. Lean is fundamentally about continuous process improvement.
 - Having identified the value, mapped the value streams and wasted steps removed, with flow and pull systems introduced – go through the process again, seeking perfection.



Exercise: Making Crisps



Exercise: Making Crisps

Other observations:

Demand for our products is high & retailers sometimes run low at peak times e.g. Christmas, Halloween

The finished goods store is frequently full.

Produce avg. 2 tons of crisps per day.

80 % of crisps are regularly sliced, 15% are crinkle cut, 5% are unpeeled kettle chips.

Lead Time – 8.5 Hours

Processing Time - 50mins



50 Kgs



Exercise: Making Crisps

You are required to:

1. Analyse this workflow to produce findings that identify sources of waste as defined by Lean
2. Describe what actions you would suggest be taken to improve the process
3. Describe the results you would expect to achieve and where specific measurements are needed

Making crisps

Findings...

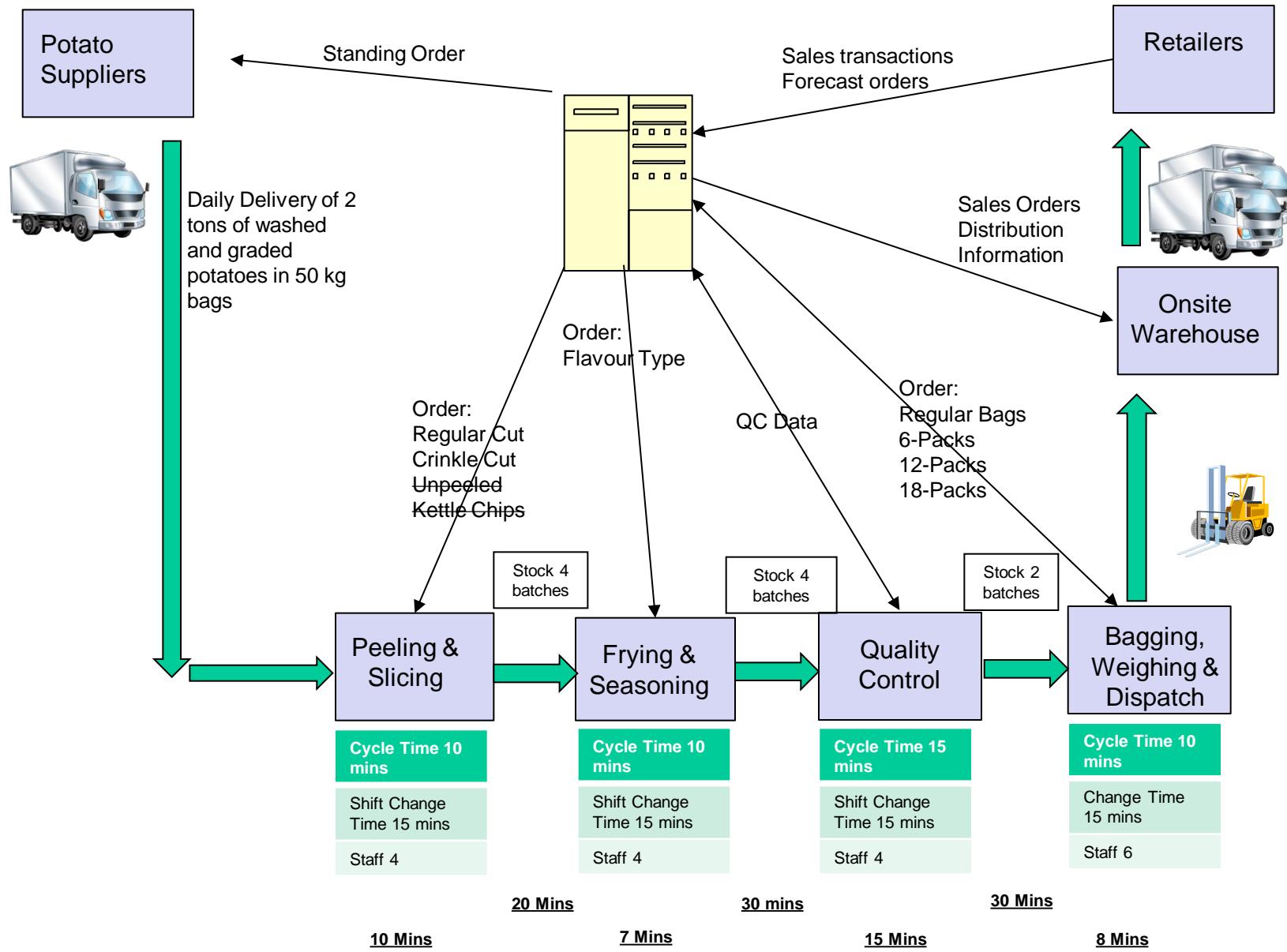
- We are spending time and money storing potatoes. We are not in the spud storage business.
- A great deal of time and effort is spent getting potatoes from the store and then washing grading and batching them.
- If we only have a certain % of potatoes that are good it means we are continually purchasing more potatoes than we actually need to compensate for waste and poor quality.
- There are significant wait times between the process steps.
- There is only one person working in quality control resulting in significant backlog of batches to be tested. The backlog is also having a knock on effect on upstream processes for frying/seasoning and peeling/slicing
- Our retailers do not attempt to forecast demand they only supply us with historical sales data that we are then using to order raw materials – there may be a great deal of variance between the two.
- If the warehouse is full it will contribute to the backup of product in the bagging station

Making crisps

Actions...

- We get our supplier to provide us with 2 tons of washed and graded potatoes that are made up in 50kg bags in a daily delivery.
- The supplier is contracted to supply the potatoes there is no issuing of individual purchase orders.
- We should consider getting rid of kettle chip production that way we could also outsource the peeling operation to our suppliers.
- We will increase the headcounts in quality control and bagging to reduce the backlog so staff will need retraining.
- We need to get our retailers to supply us with forecast data based on their actual sales and we should share this information with our suppliers to predict seasonal demand.
- We are putting on extra deliveries to reduce the inventory of finished goods, to allow our retailers better plan for peak demand, to reduce the impact on our bagging station, to deliver to end customers a fresher product.

Making Crisps



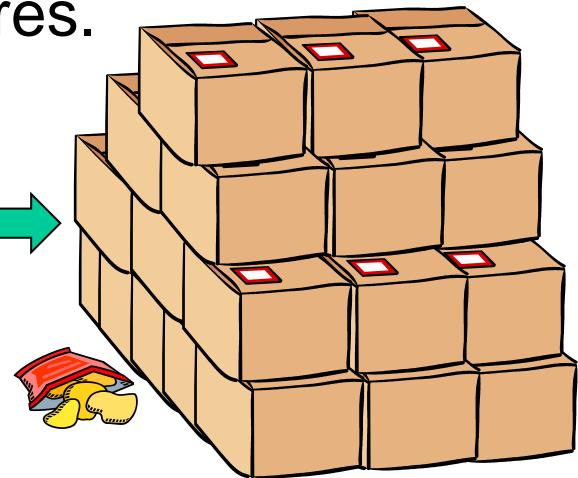
Making Crisps

Expected Results:

Forecasting the demand for our products has improved significantly - actual measures are needed
The finished goods store is better managed – actual turnover measures are needed
Lead Time has been cut to – 1Hr 20Mins.
Processing Time has been cut to - 40mins.
Overhead associated with invoice processing has been cut drastically – need measures.



50 Kgs



Lean IT

Lean in IT follows the same basic principles but it does have some significant challenges...

- Software is “unvisualizable” so seeing the product value streams can be quite difficult.
- Software lends itself to a number of different interpretations. It can be considered from a logical, or a data transformation, or a role interaction based perspective.
- Developing software is unique to each organisation and usually unique for each project so there are few reference guidelines for implementing Lean IT

Lean IT

Lean in IT follows the same basic principles but it does have some significant challenges...

- Software organisations are frequently fragmented e.g. Outsourced activities, and the value streams cannot be expressed in neat concise terms
- Resistance to change within software organisations can be huge because software departments often have very high degrees of autonomy
- Lack of support tools and technologies for Lean IT
- The software supply chain can be poorly integrated as well as being fragmented with contributions coming from many diverse sources (e.g. Off-the-shelf packages, open source software, subcontractors and consultants)



Today's Lecture...

- Software Process Improvement 2

Software Process Improvement

There are many different models & methodologies for improving the software process

We are going to take a closer look at 3 of them:

- Lean
- Maturity Models
- 6-Sigma

Software Process Improvement

Each has a distinct focus:

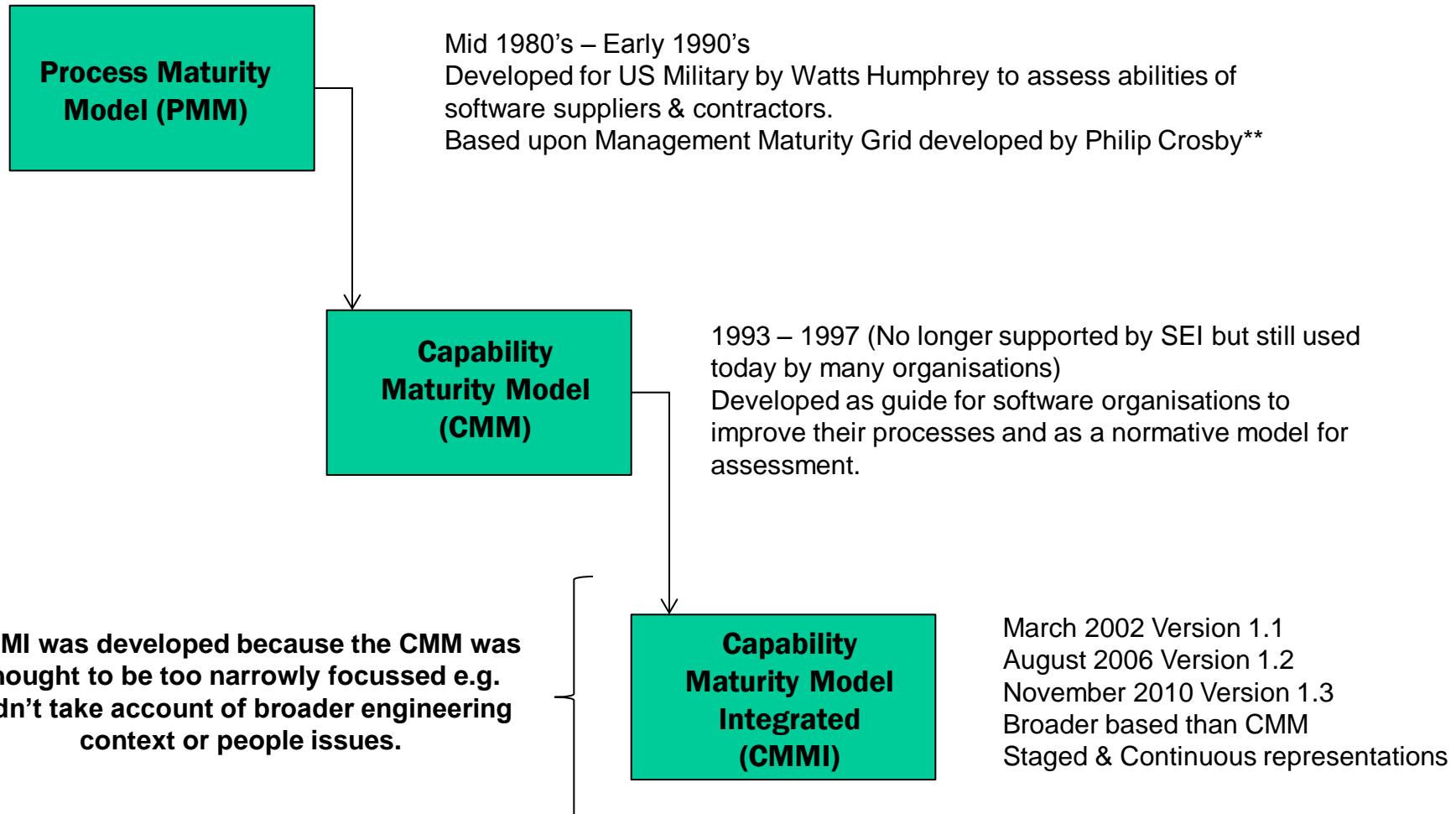
- Lean
 - The **elimination of waste**, where waste is defined as having no added value to the software product or service being provided
- Maturity Models
 - Guiding software organisations' improvement efforts by providing normative references that facilitate **continuous improvement** in stages
- 6-Sigma
 - The **removal of variability** within a process, where variability is defined in terms of exceeding statistically calculated limits

Software Process Improvement

Examination & comparison of two models:

- The Capability Maturity Model (CMM)
- The Capability Maturity Model Integrated (CMMI)

CMU SEI* Software Maturity Models



*Carnegie Mellon University Software Engineering Institute

**"Quality is Free", Crosby, P., 1980

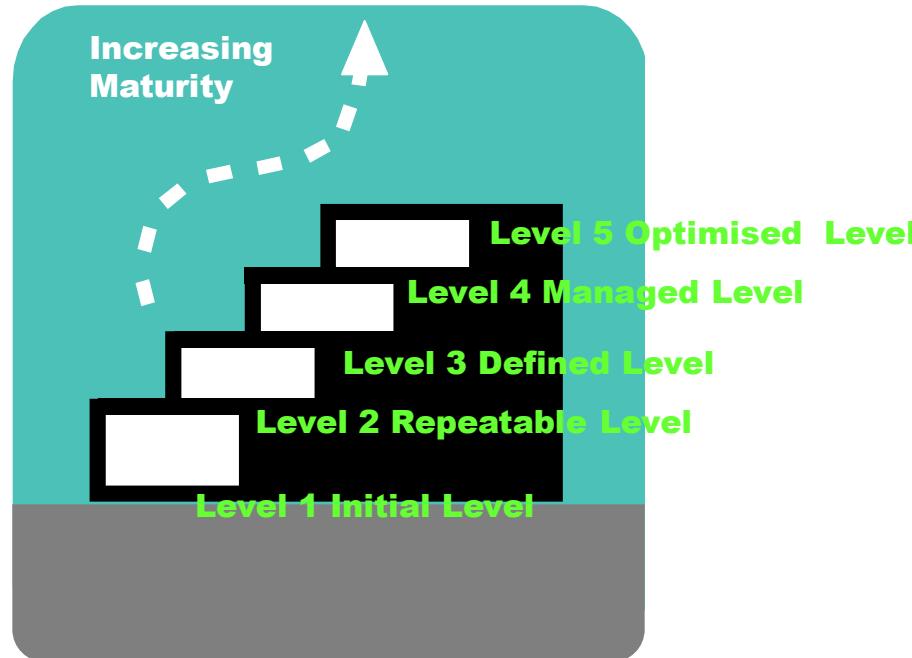
Uses of Maturity Models

- Two Principle Uses
 1. Provide a normative reference to guide organisations in continuously improving their software process
 2. Provide a means by which a software organisation's process maturity can be assessed
- Begin with the concepts that were pioneered in the CMM & then examine the extension of those concepts in CMMI
- Two representations 'staged' (CMM & CMMI) & 'continuous' (CMMI only)

Basic concepts of Capability Maturity Models

The **Staged Representation** (for both CMM & CMMI) provides:

1. A systematic, structured way to approach process improvement one stage at a time, ensuring that achieving each stage lays a foundation for the next stage
2. Prescriptive order for making improvements taking the guesswork out of what to improve next.
3. Good way to start for organisations that are new to software process improvement



Basic concepts of Capability Maturity Models

- Each Maturity Level in CMM & CMMI staged representation (except Maturity Level 1) is composed of a collection of practices known as **Process Areas** (PA's):

Process Areas for each Maturity Level		
	CMM	CMMI
Maturity Level 1	0	0
Maturity Level 2	6	7
Maturity Level 3	7	14
Maturity Level 4	2	2
Maturity Level 5	3	2
Total PA's	18	25

- The software process in Level 1 organisations is characterised as being chaotic, unpredictable, and poorly controlled hence no PA's are assigned to it.

Basic concepts of Capability Maturity Models

Major criticisms of the staged representation were:

1. Many organisations became focussed on achieving a particular maturity level – whether they were actually improving the organisation's business or not.
2. Many organisations felt constrained by the model – feeling they were obliged to work on improving what the model was dictating as opposed to what they felt was important for their business.

As a result of these concerns SEI brought out two different representations:

- 1) a CMMI **staged representation** (as before), and
- 2) a new CMMI **continuous representation**

Basic concepts of Capability Maturity Models

To facilitate a **Continuous Representation (CMMI only)** SEI introduced the concept of 6 capability levels: CL0->CL5

CL0 = Incomplete Process

CL1 = Performed Process

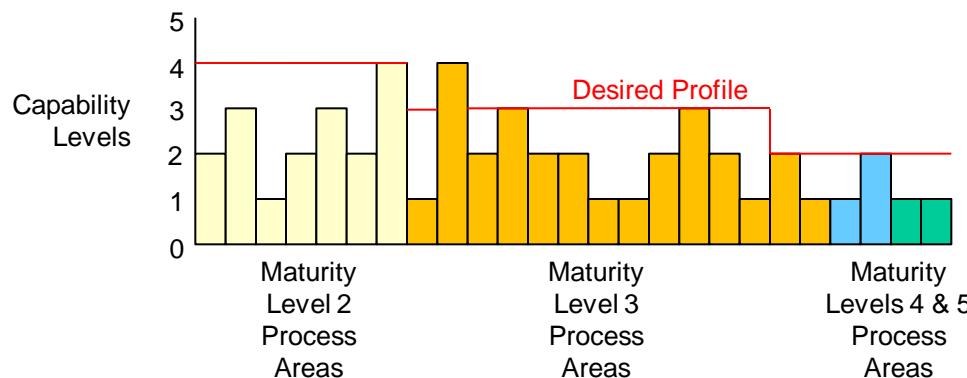
CL2 = Managed Process

CL3 = Defined Process

CL4 = Quantitatively Managed Process

CL5 = Optimizing Process

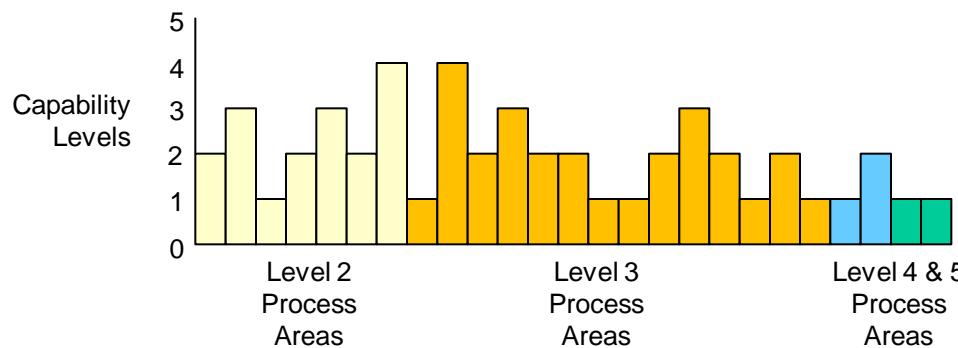
This allowed software organisations to aim for a desired process profiles for their business and through a technique called 'equivalent staging' could convert their capability level profile to a maturity level.



Basic concepts of Capability Maturity Models

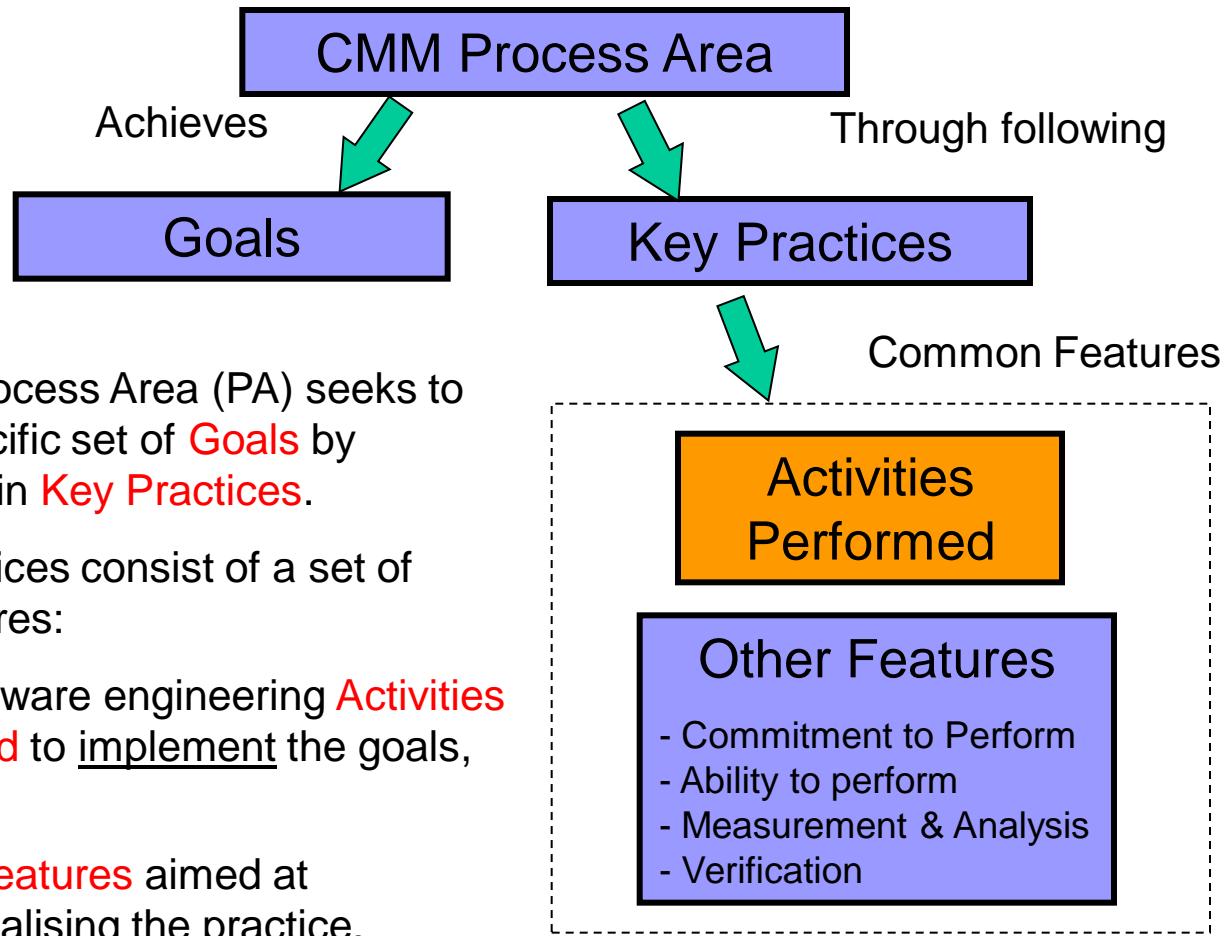
The **Continuous Representation** provides a more flexible approach to process improvement:

1. Improve the performance of a single process trouble-spot or work on process areas that are more important to the organisation's business objectives.
2. Improve different processes at different rates – but may be limited because of dependencies between process areas
3. Organisation's may strive to have different capability level in different process areas – may want to have a capability level of 2 in one area and 4 in another



Basic concepts of Capability Maturity Models

- Each CMM PA is composed of practices called **common features** which either implement the goals of the PA or institutionalise the practices of the PA.



CMM Level 1

- The software process in Level 1 organisations is characterised as being chaotic, unpredictable, and poorly controlled.
- If the software process is not managed explicitly then factors which impact the process, and are often unavoidable, will only result in increased instability of it. (e.g. technology change, strategic decisions, fads, social change)
- **But** even in undisciplined organisations, some individual software projects produce excellent results, usually due to heroics, rather than following a mature software process. So repeating results depends entirely on having the same heroes around for the next project.
- Success that rests solely on the availability of specific individuals is no basis for long-term productivity and quality improvement.

CMM Level 1 is called the **Initial** level

CMM Level 2 Process Areas

Requirements Management

Project Planning

Project Tracking & Oversight

Software Subcontract Management

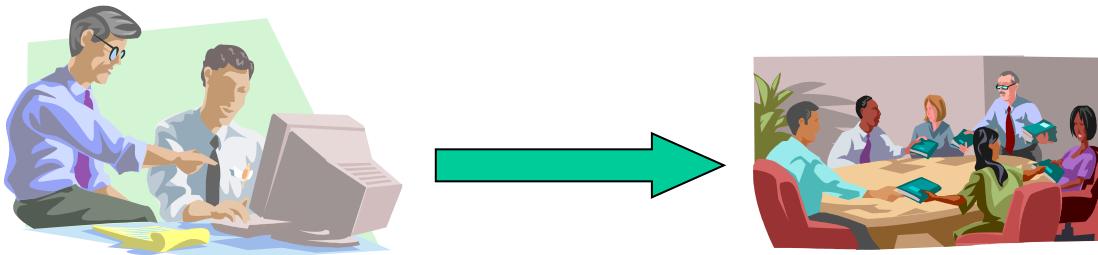
Software Quality Assurance

Software Configuration Management

The focus of CMM level 2 is on distinct management functions at the project level

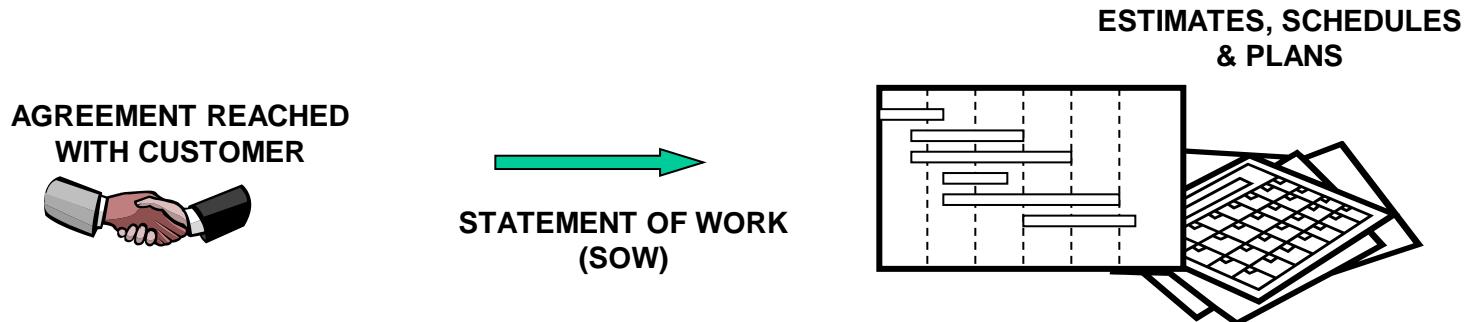
CMM Level 2 is called the **Repeatable** level.

CMM Level 2 PA's - Requirements Management



- Establishing a common understanding, between the customer and the software project, of the customer's needs that will be addressed by the software project.
- First goal is to ensure that requirements are put under control as they form the basis for the project
- Second goal is to ensure that anything we develop (documents, code, etc.) are traceable back to the requirements

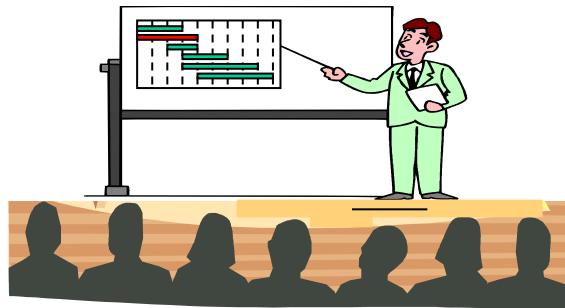
CMM Level 2 PA's - Software Project Planning



Project Planning

- Establishes reasonable plans for managing the software project.
- The agreement reached with the customer during the Requirements phase is the basis for doing the project planning.
 - First goal is that estimates are developed (based on the Statement of Work) for the work to be performed.
 - Second goal is plans are drawn up to do the project (schedules, risks, resources, etc.)
 - Third goal is that everyone understands what the project is about & their role in it

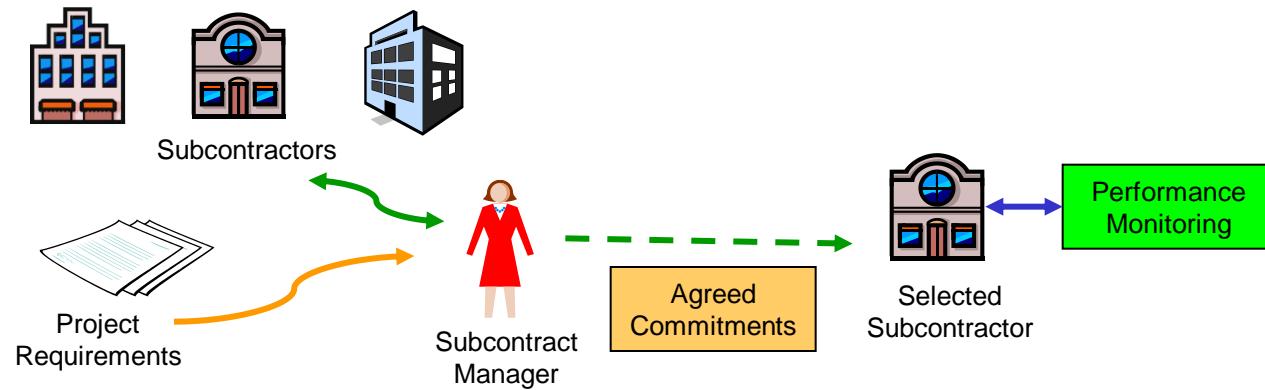
CMM Level 2 PA's - Software Project Tracking & Oversight



Project Tracking & Oversight

- Providing adequate visibility into actual progress so that management can take effective actions when the project's performance deviates significantly from the project plan.
- First goal is to make sure actual progress is tracked against plans
- Second goal is to ensure any corrective actions that are taken, when the project starts to stray from the plan are managed to closure
- Third goal is that if we change any commitments we made for the project (schedule, cost, content, etc.) are agreed to by those that are affected by the change

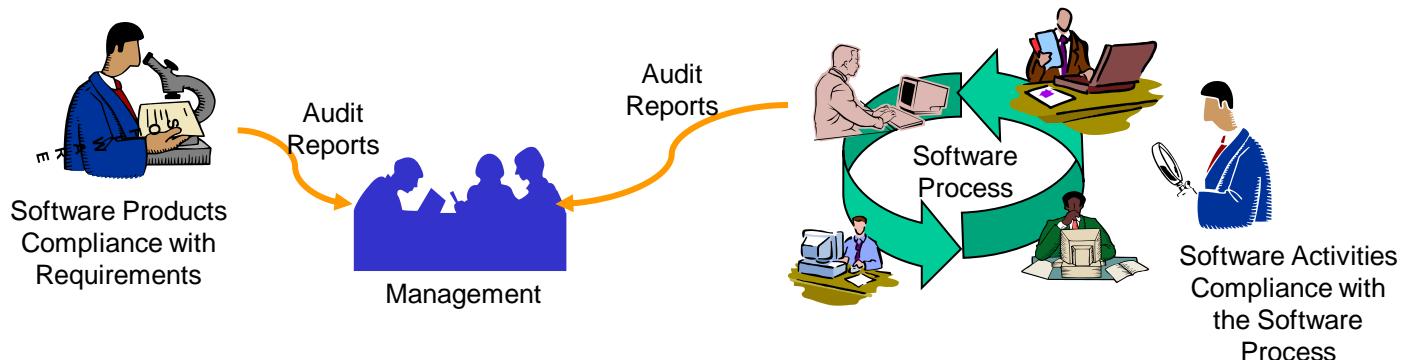
CMM Level 2 PA's - Software Subcontractor Management



Sub-Contractor Management

- First goal is to select the best qualified subcontractor to do the work that has been decided to subcontract based on the project requirements
- Second goal is for subcontract manager to agree commitments with the subcontractor about what has to be delivered plus schedule, price, etc.
- Third goal is to maintain on-going communications between the company & the subcontractor
- Fourth goal is to tracking and review the subcontractors results & performance just like any other software group

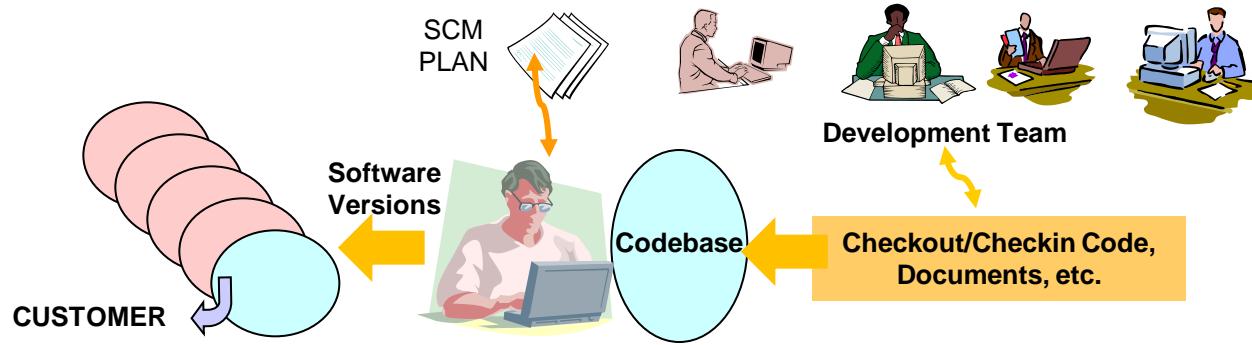
CMM Level 2 PA's - Software Quality Assurance



Software Quality Assurance

- Software Quality Assurance provides management with independent visibility into the process being used by the software project and the products being built. In many organisations Software Quality Assurance is thought of as a customer advocacy role.
- First goal is that QA activities are planned for
- Second goal is to follow the QA plan and verify the software complies with the requirements and follows the process
- Third goal is to inform people about the QA activities and issues and try to resolve them
- Fourth goal is to get management involved when issues cannot be resolved

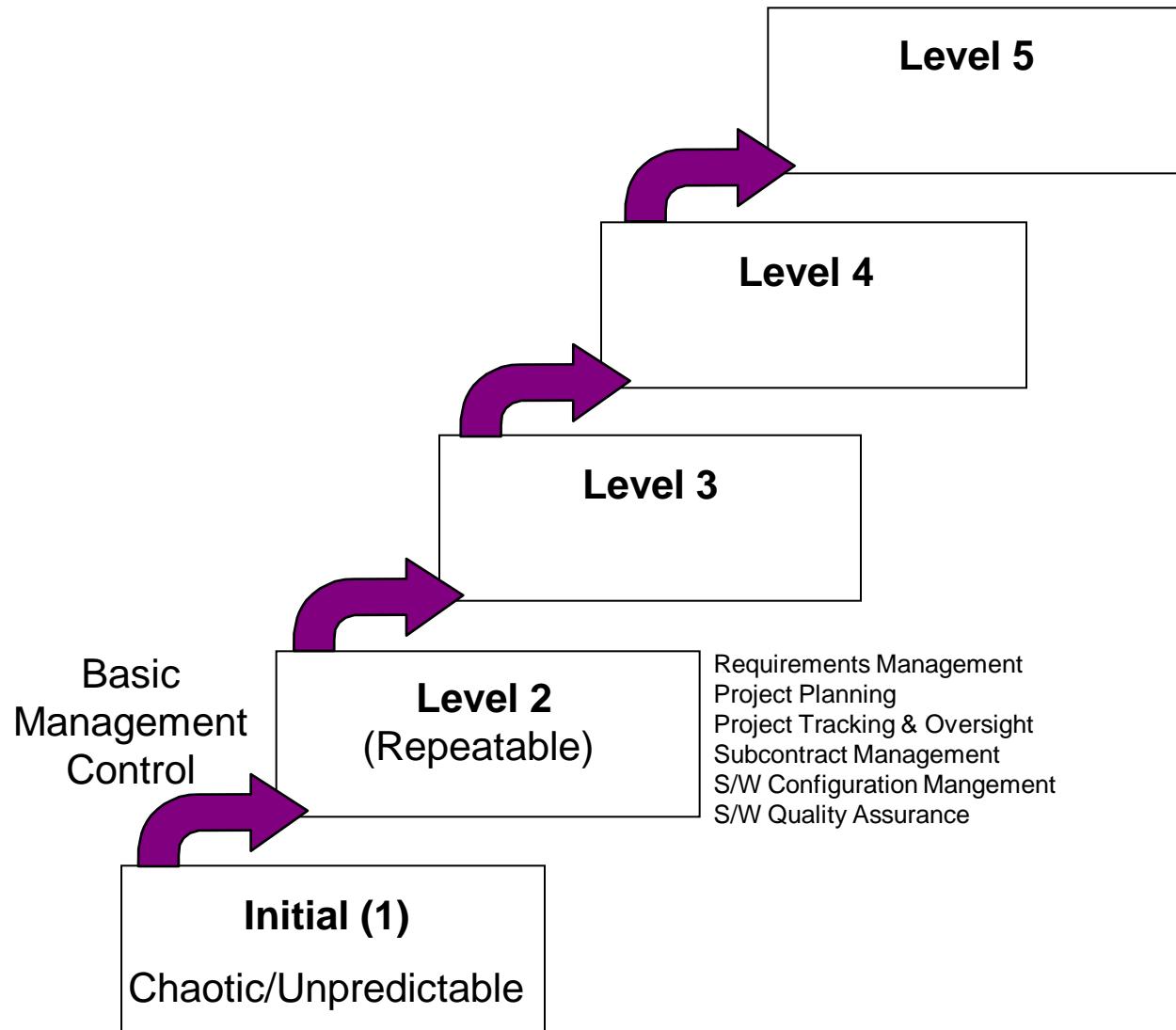
CMM Level 2 PA's - Software Configuration Management



Software Configuration Management (SCM)

- Establishes and maintains the integrity of the software products produced by the software project through version control and change control procedures
- First goal is that the SCM activities are planned
- Second goal is the work products to be put under SCM control are identified
- Third goal is to have enforcement of the change control processes and auditing for compliance
- Fourth goal is to ensure people are made aware of the status and content of the software versions

CMM Levels 1 & 2



CMM Level 3 Process Areas

Organisational Process Focus

Organisational Process Definition

Integrated Software Management

Software Product Engineering

Inter-group Co-ordination

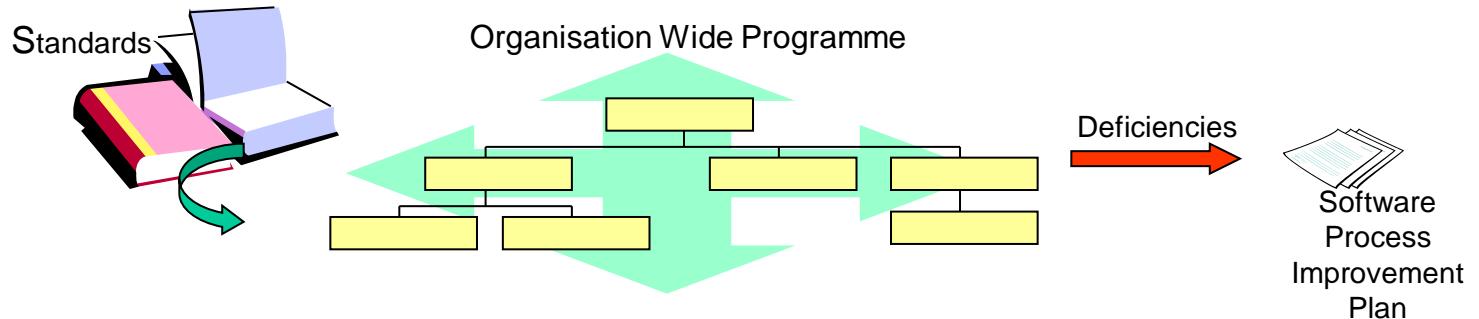
Training Plan

Peer Reviews

The focus of CMM level 3 is on managing processes at an organisational level

CMM Level 3 is called the **Defined** level.

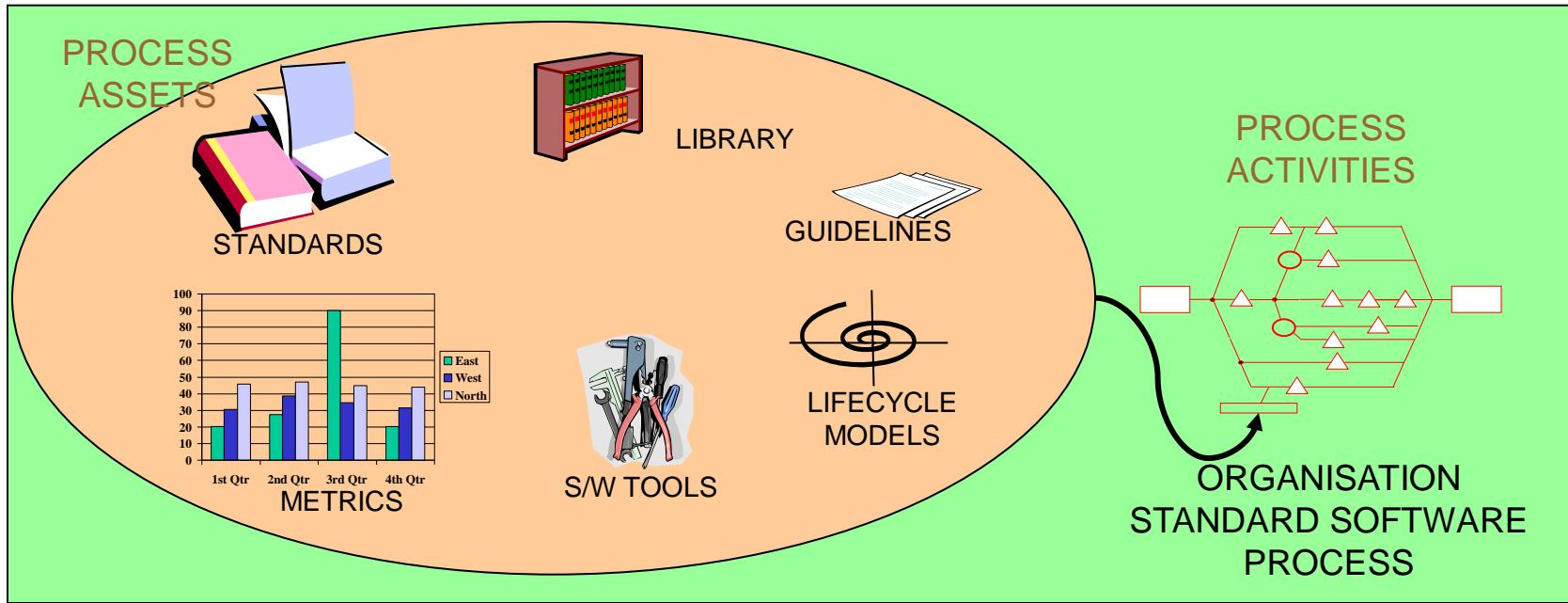
CMM Level 3 PA's – Organisation Process Focus



Organisation Process Focus

- Co-ordinating process improvement activities across the whole organisation by identifying process strengths and weaknesses and planning improvement activities
- The first goal is to ensure the activities are coordinated across the whole organisation
- The second goal is to ensure that the strengths and weaknesses of the organisations processes are identified relative to a process standard
- The third goal is to have an organisation-wide plan for the process improvement activities

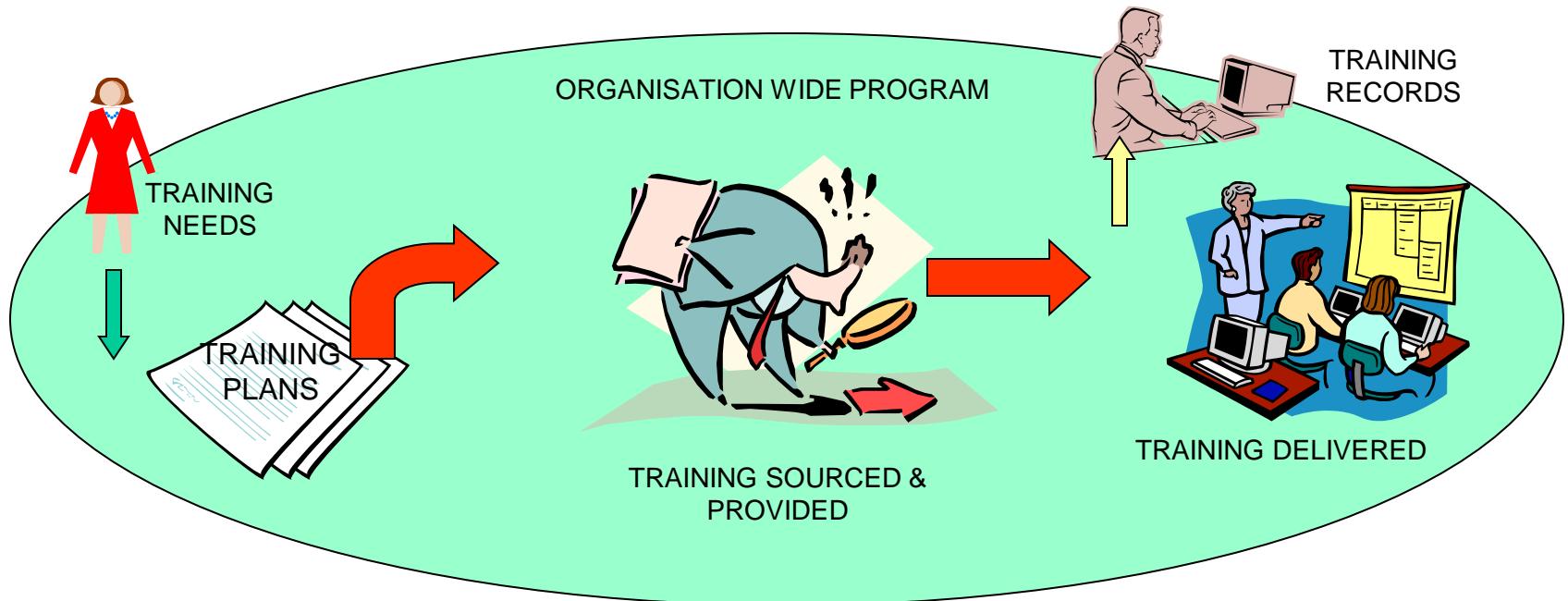
CMM Level 3 PA's – Organisation Process Definition



Organisation Process Definition

- Captures all software process activities that are performed by the organisation, along with all process assets; tools, metrics, standards, etc.
- The first goal is to have an Organisational Standard Software Process that is developed and maintained
- The second goal is to collect information about how the organisation's projects use the process and make it known

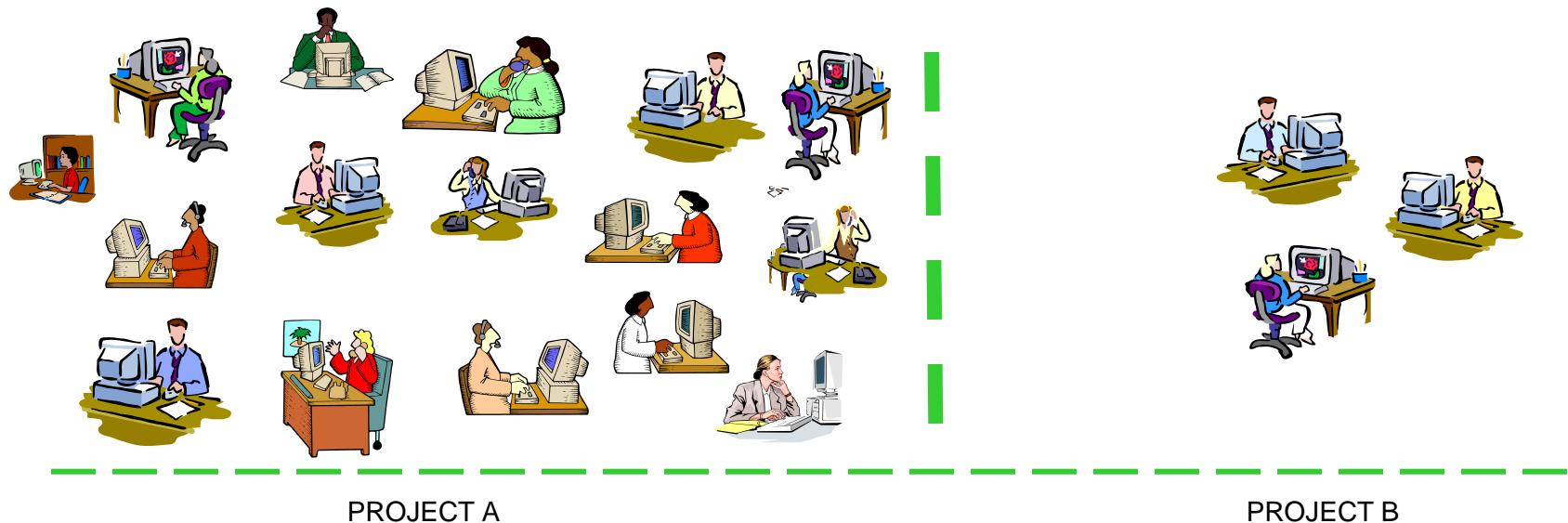
CMM Level 3 PA's – Training Program



Training Program

- Training is planned for, provided, and delivered to all organisational disciplines
- First goal is to ensure training needs are identified and planned for
- The second goal is ensuring the training is sourced and provided.
- The third goal is that training is delivered and training records are kept.

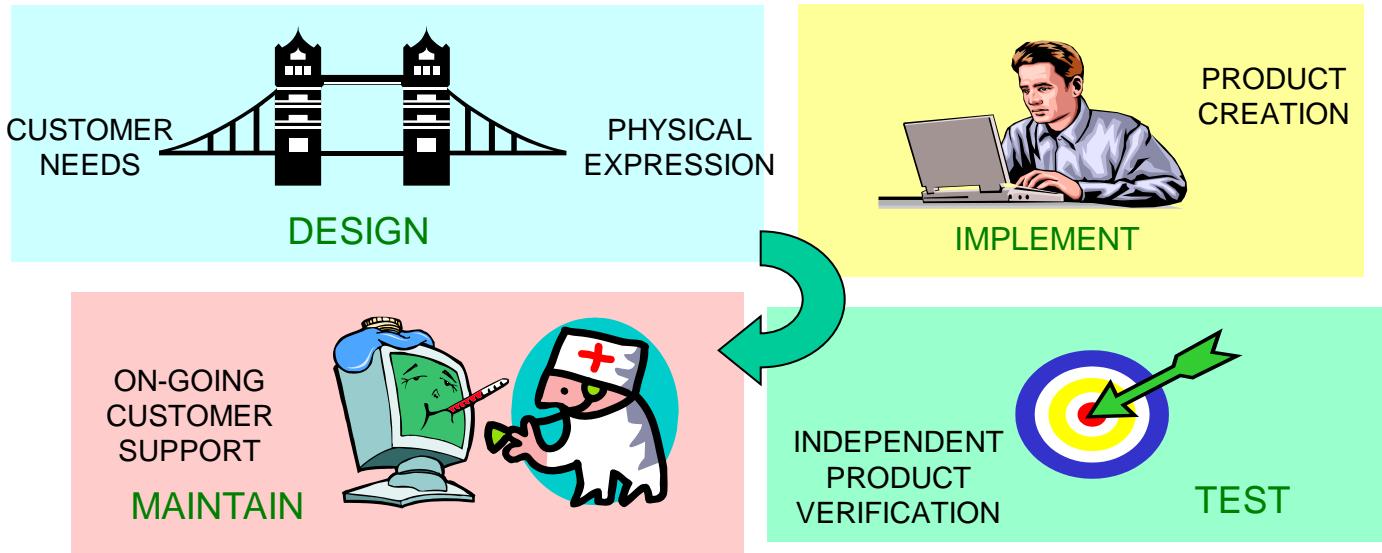
CMM Level 3 PA's – Integrated Software Management



Integrated Software Management

- Software Projects come in all sorts of shapes and sizes and the software process may not be easily used by some projects
- The first goal is to produce a tailored version of the organisations defined software process that is usable by a software project
- The second goal is that the software project is then managed according to that tailored version of the software process

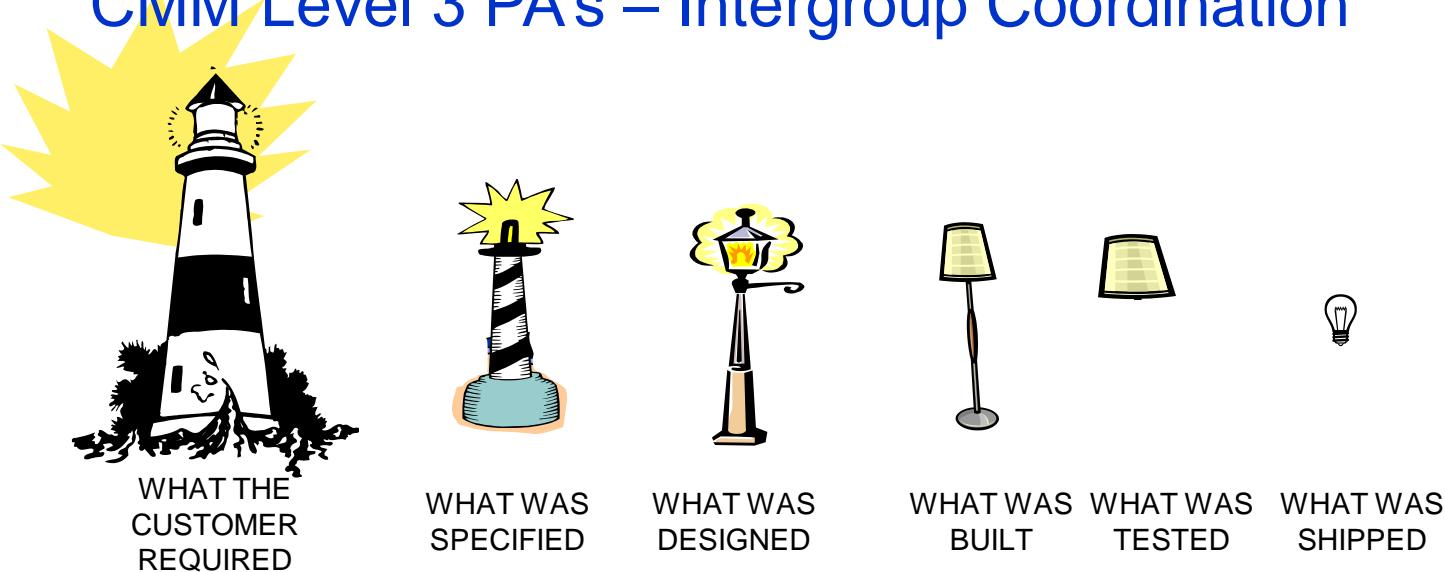
CMM Level 3 PA's – Software Product Engineering



Software Product Engineering

- Describes the engineering activities to produce software. It does not mean that software cannot be produced up to now but it does show that there are other organisational practices that need to be in place to support product engineering.
- The first goal is to ensure engineering tasks are defined, integrated, and consistently performed to produce the software
- The second goal is ensuring the software products are kept consistent with the original requirements

CMM Level 3 PA's – Intergroup Coordination



Intergroup Coordination

- Establishes a means for the software engineering group to participate actively with the other groups to address the requirements, objectives and issues of the system.
- The first goal is to ensure the customers requirements are understood by all affected groups.
- The second goal is that commitments between the engineering groups are agreed to.
- The third goal is that intergroup issues are identified tracked and resolved.

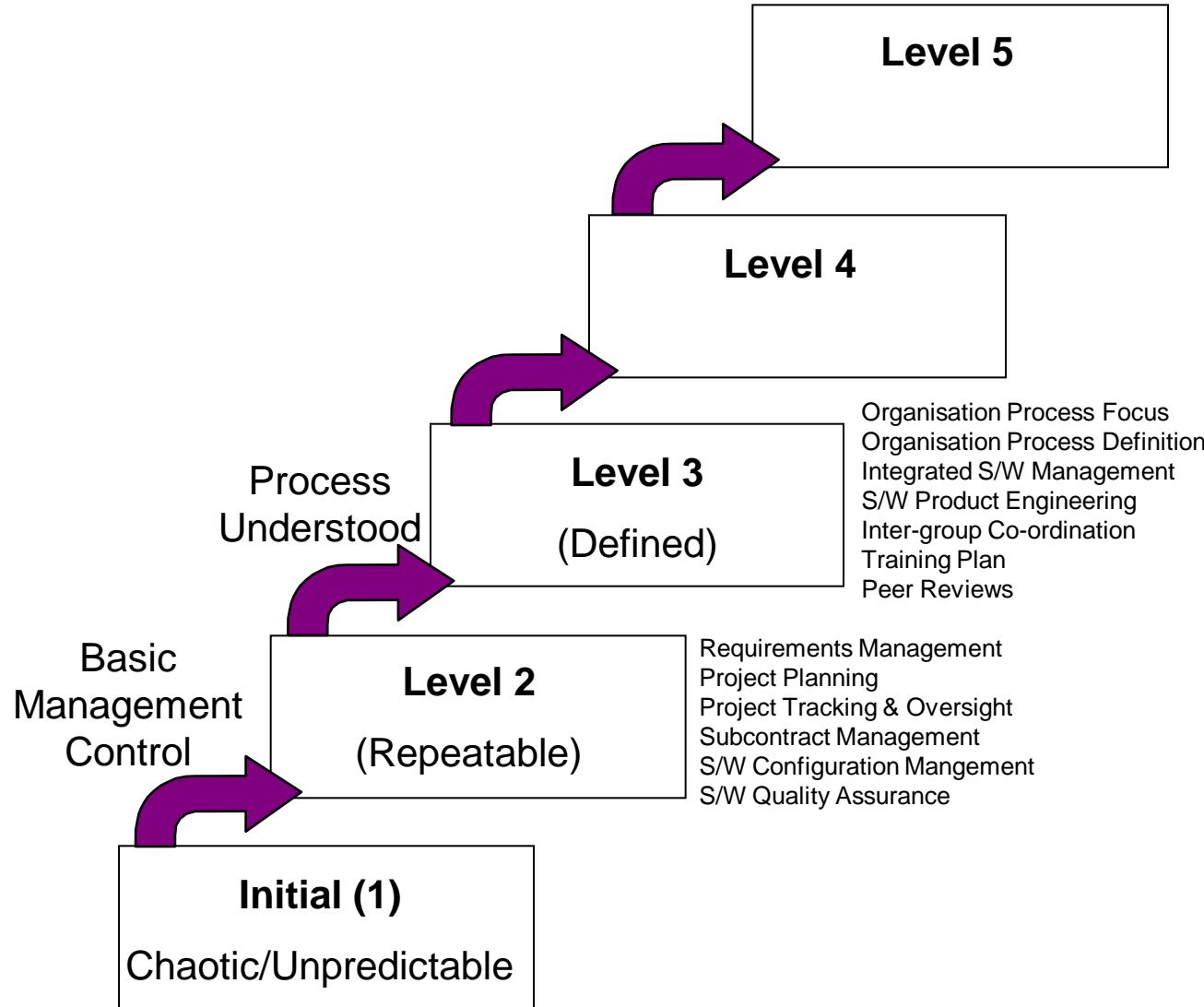
CMM Level 3 PA's – Peer Reviews



Peer Reviews

- Peer Reviews (or Formal Inspections) seek to remove defects from the products being produced, early and efficiently (ref. Software Verification Testing)
- The first goal is that the peer reviews are planned. The plans will typically identify what is to be inspected, the inspection schedule and who is involved.
- The second goal is that faults are removed from the product and records are kept.

CMM Levels 1, 2 & 3



CMM Level 4 Process Areas

Quantitative Process Management

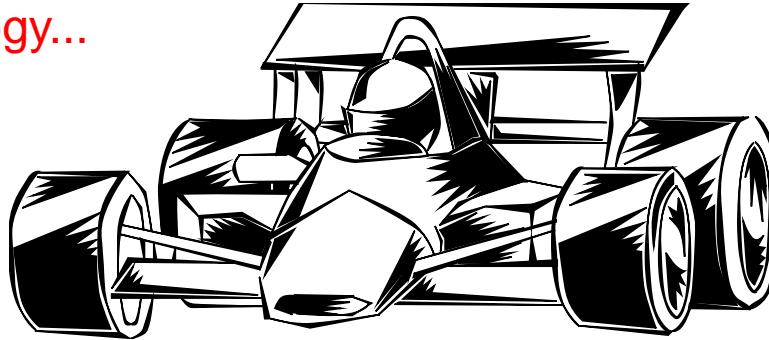
Software Quality Management

The focus of CMM level 4 is on quantitative control of the process and products

CMM Level 3 is called the **Managed** level.

Process Defined but how will it perform?

Lets take an analogy...



Just like our organisation, a racing car is a composite of various subsystems (ignition, brakes, steering, etc.) which all work together.

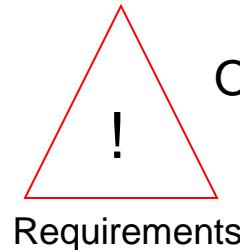
To establish how capable the car is of *performing* we need to define the operational parameters or characteristics for these various subsystems.

For example: we will determine that on **average** during a race we will have to change the tyres after 160kms, that the optimum temperature **range** for the coolant is between 85degC & 90degC and we will predict with some **degree of confidence** that the suspension will last to the end of the race.

These words - **average**, **range** and **degree of confidence** - are of course statistical terms [which we can also apply](#) to the software process

Statistical Control

We can select important software process components or product indices and seek to understand them **statistically** just as we selected variables for assessing the performance of the racing car. For example...



Our estimation accuracy should lie within some pre-determined **range** of values.

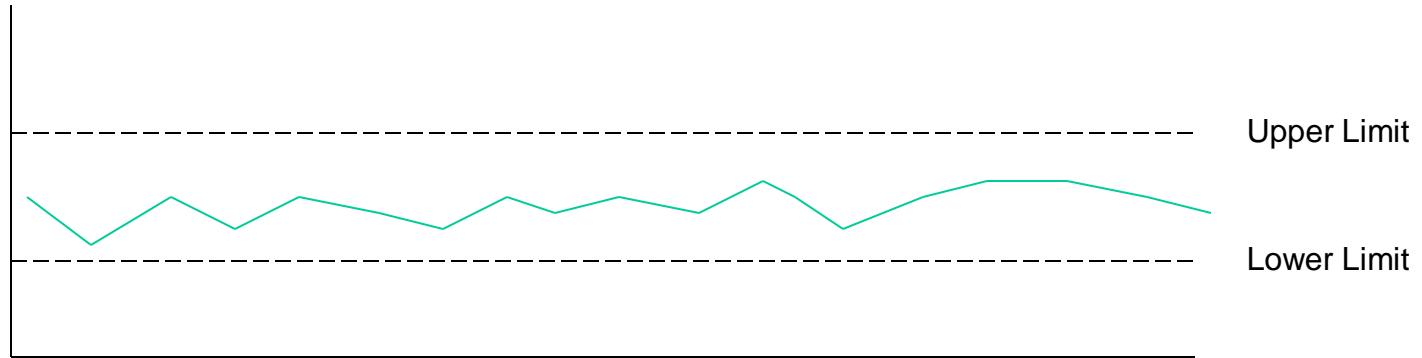
Our customer satisfaction indices should all be in the **upper quartile**.



On **average** we should be finding N major defects in 200 lines of C code



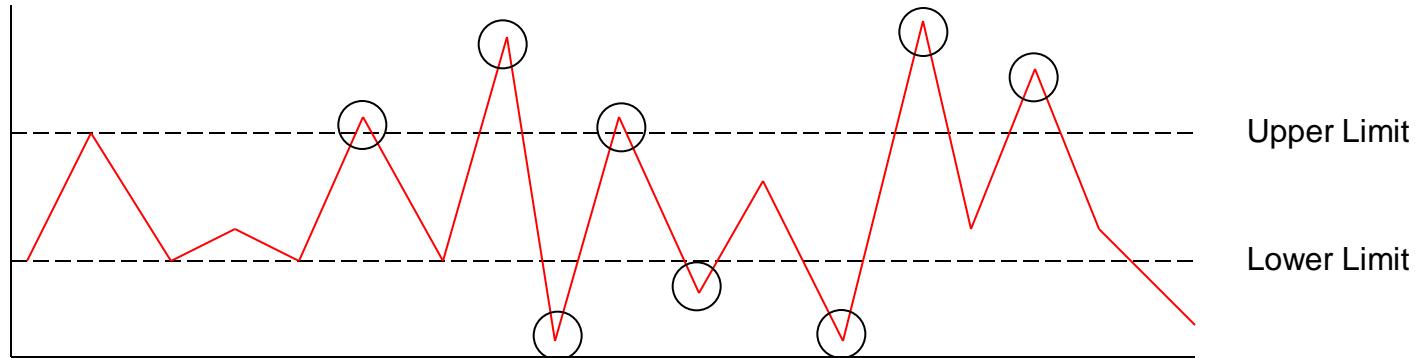
Is the software process stable or unstable?



Variable: Software Size Estimation Accuracy

In this example the Software Process is **Stable**. We know the capability of the process & we have the ability to predict quantitatively the quality of the software.

Is the software process stable or unstable?



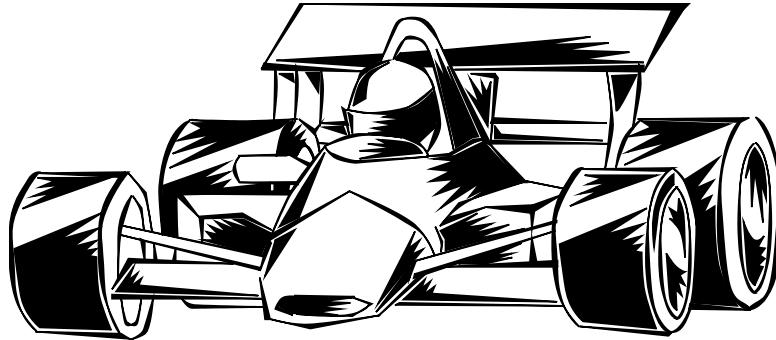
In this example the Software Process is **Unstable**. We do not know the capability of the process. We cannot predict quantitatively the quality of the software...

...because at certain points the process has exceeded the thresholds that were established - prompting us to take corrective action on these special causes of variation.

In our racing car analogy if this were tyre temperature we would want to understand this **process** - why did the tyres sometimes overheat and other times failed to reach the operating range either deviation will affect the **quality** of the cars performance.

In other words, it is only by controlling the **process** that we can hope to maintain or improve product quality.

CMM Level 4 PA's – Quantitative Process Management



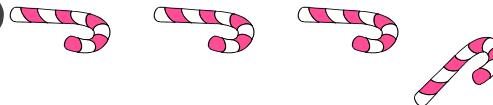
Quantitative Process Management

- Planning and controlling the organisations defined software process quantitatively and therefore knowing in quantitative terms what the process is capable of.
- The first goal is that the process management activities are planned.
- The second goal is that the process is controlled quantitatively.
- The third goal is the capability of the software process is understood in quantitative terms

Software Quality management



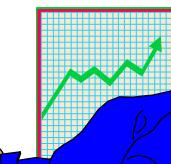
Target weight for each Candy Cane = 20 grams



The Candy Cane Process

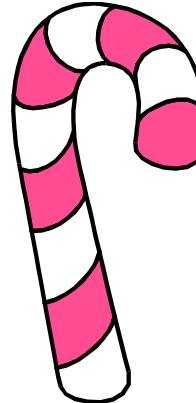


...and quality assurance verifies that each cane does weigh 20 grams



...but through quality management its found that while the canes weigh 20 grams the canes are getting longer. As a result they are too brittle. When this **quality** data is fed back the process is corrected.

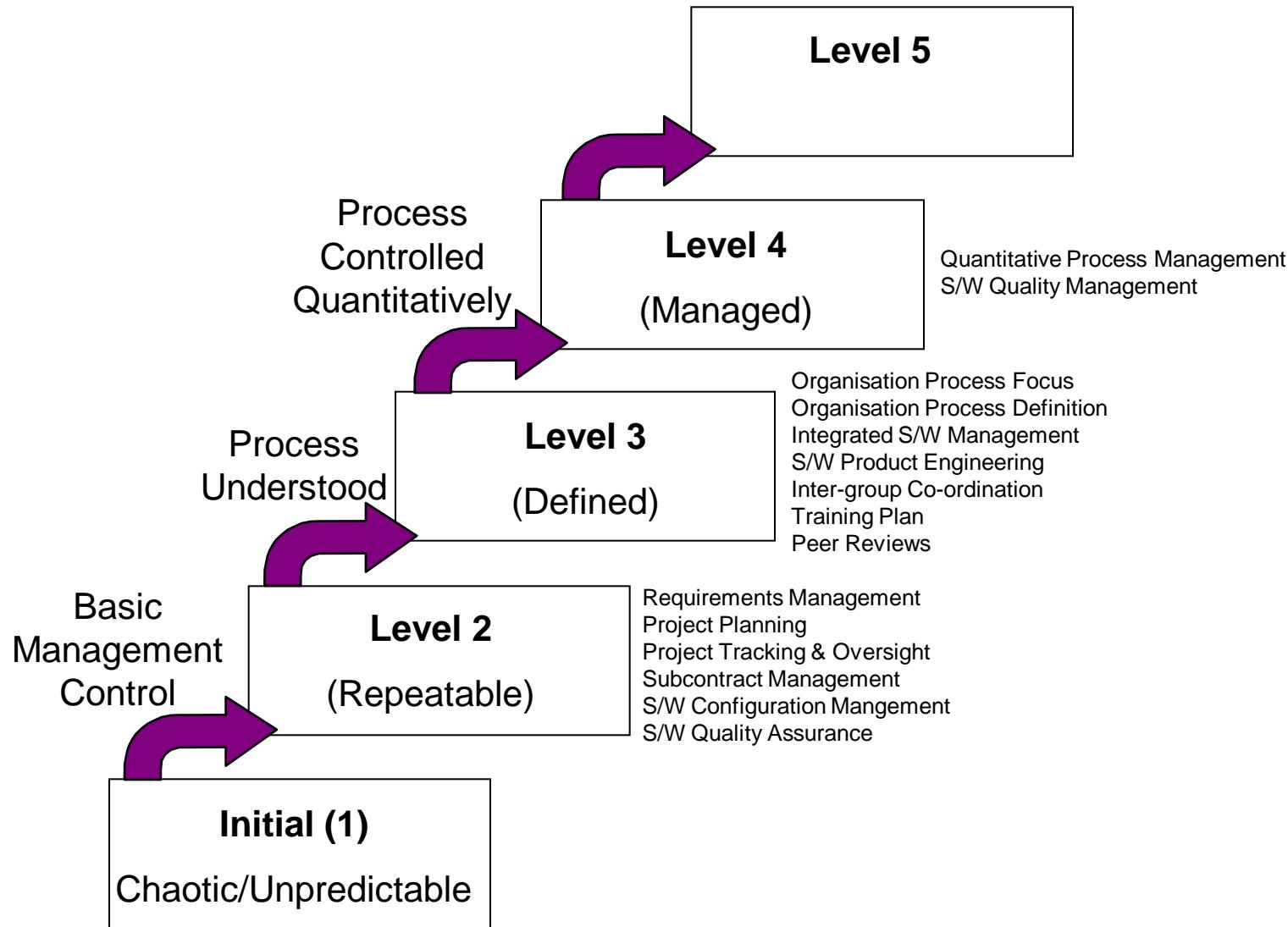
CMM Level 4 PA's – Software Quality Management



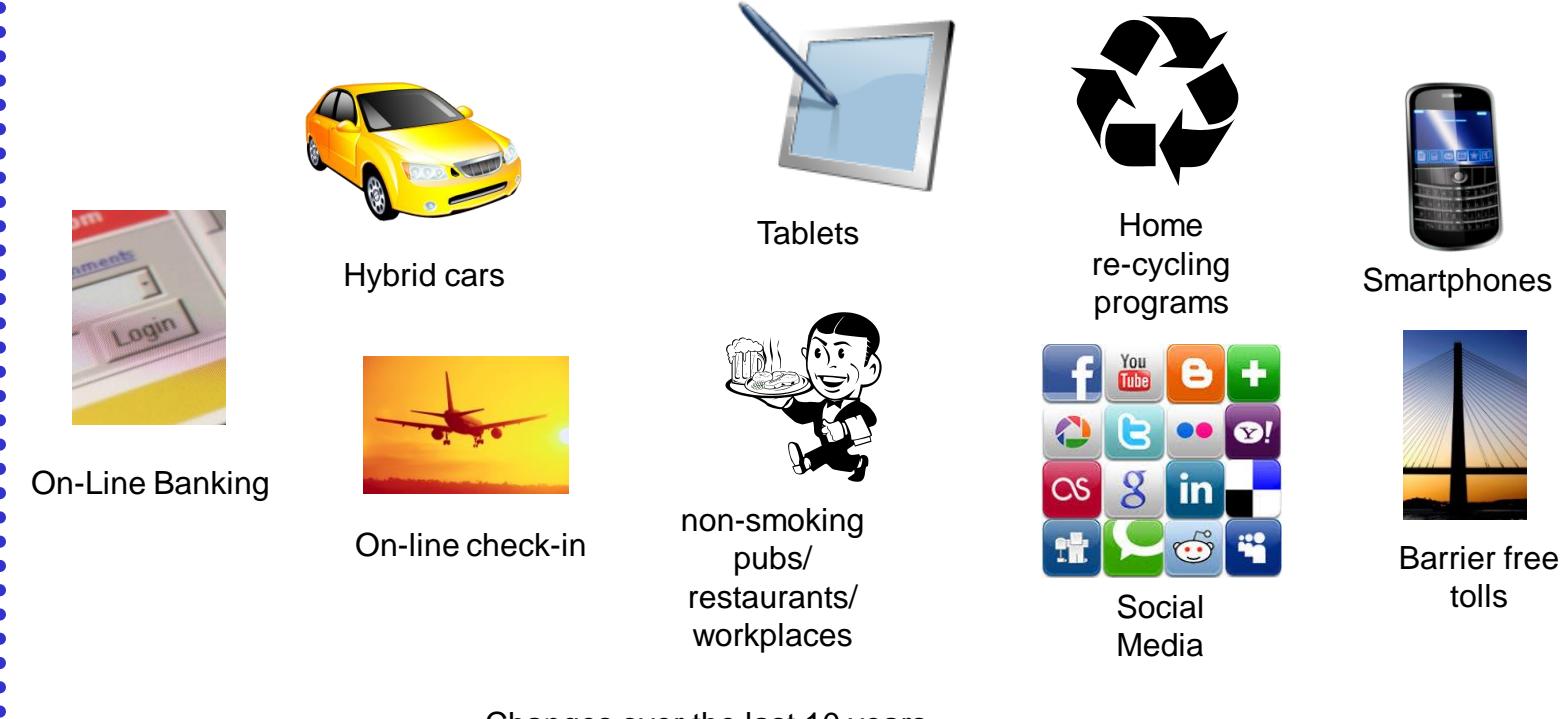
Software Quality Management

- Establishing measurable goals for software product quality and then tracking and managing progress towards meeting these quantifiable goals.
- The first goal is that the quality management activities are planned.
- The second goal is that measurable goals for product quality are defined..
- The third goal is that actual progress towards meeting the quality goals is managed

Levels 1,2,3 & 4



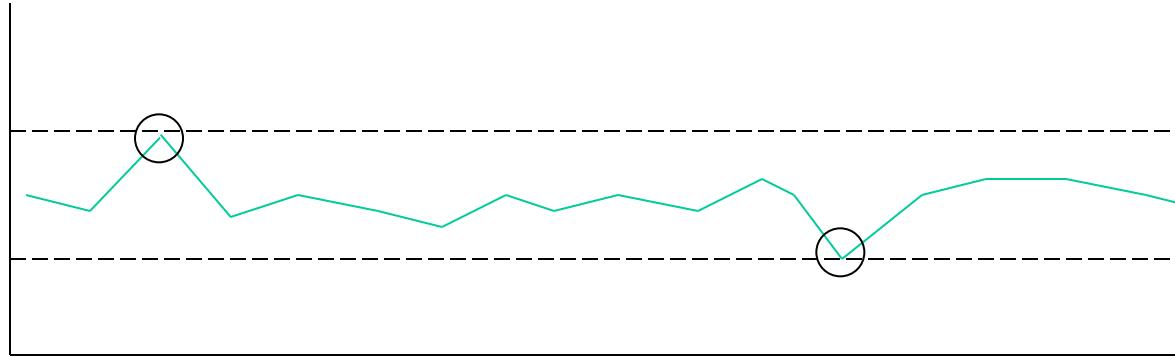
Change is Inevitable



Because organisations must deliver defect free products...because they can
realise advancement through technology changes ...or because the process
capability needs to be continually renewed and optimised...the process
needs to be changed and that change needs to be managed

Level 5 PA's - Defect Prevention

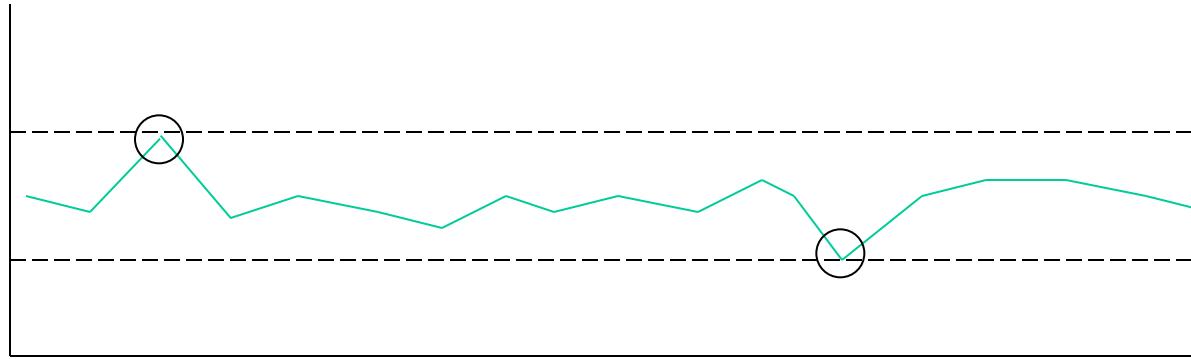
Change management is more **pro-active** than **re-active**.



When we look at this chart we see a stable process but even here there are process variations that could easily exceed our control limits. We need to **pro-actively** seek out the common causes of these variations in the process and remove them to stop them leading to problems in the future.

This can be done through a **defect prevention** program.

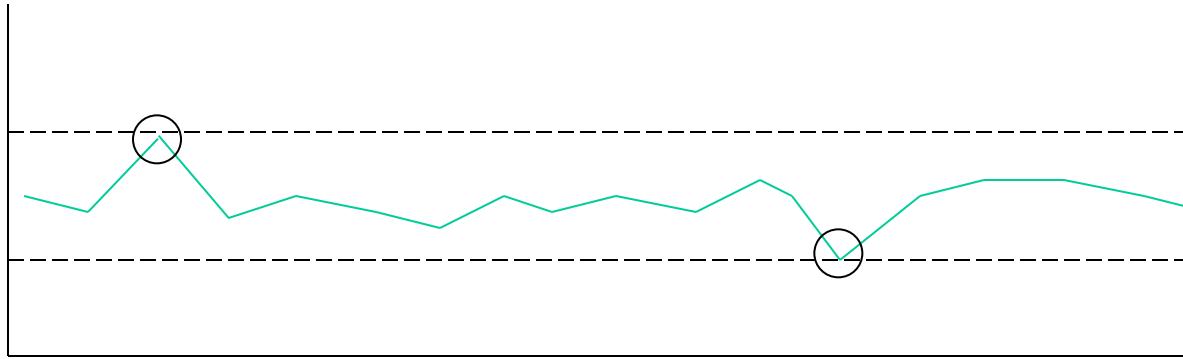
CMM Level 5 PA's – Defect Prevention



Defect Prevention

- Planning for and following activities that will seek out common causes of defects, will prioritise them and eliminate them.
- The first goal is that the defect prevention activities are planned.
- The second goal is that common causes of defects are found and identified
- The third goal is that common causes of defecets are systematically eliminated

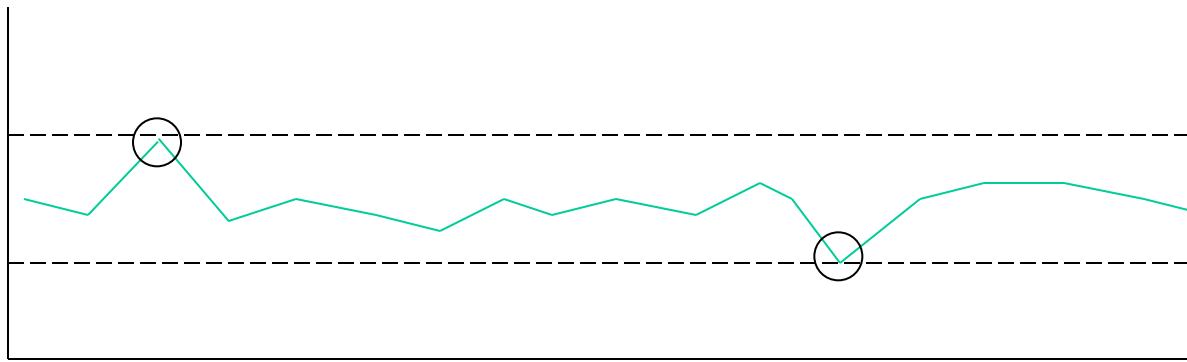
CMM Level 5 PA's – Technology Change Management



Technology Change Management

- Planning for and then evaluating new technologies to determine their effect on quality and productivity before transferring the new technology into normal practice in the organisation
- The first goal is that the technology change activities are planned.
- The second goal is that new technologies are evaluated to assess their effect on quality & productivity
- The third goal is that new technologies are transferred into normal practice across the organisation.

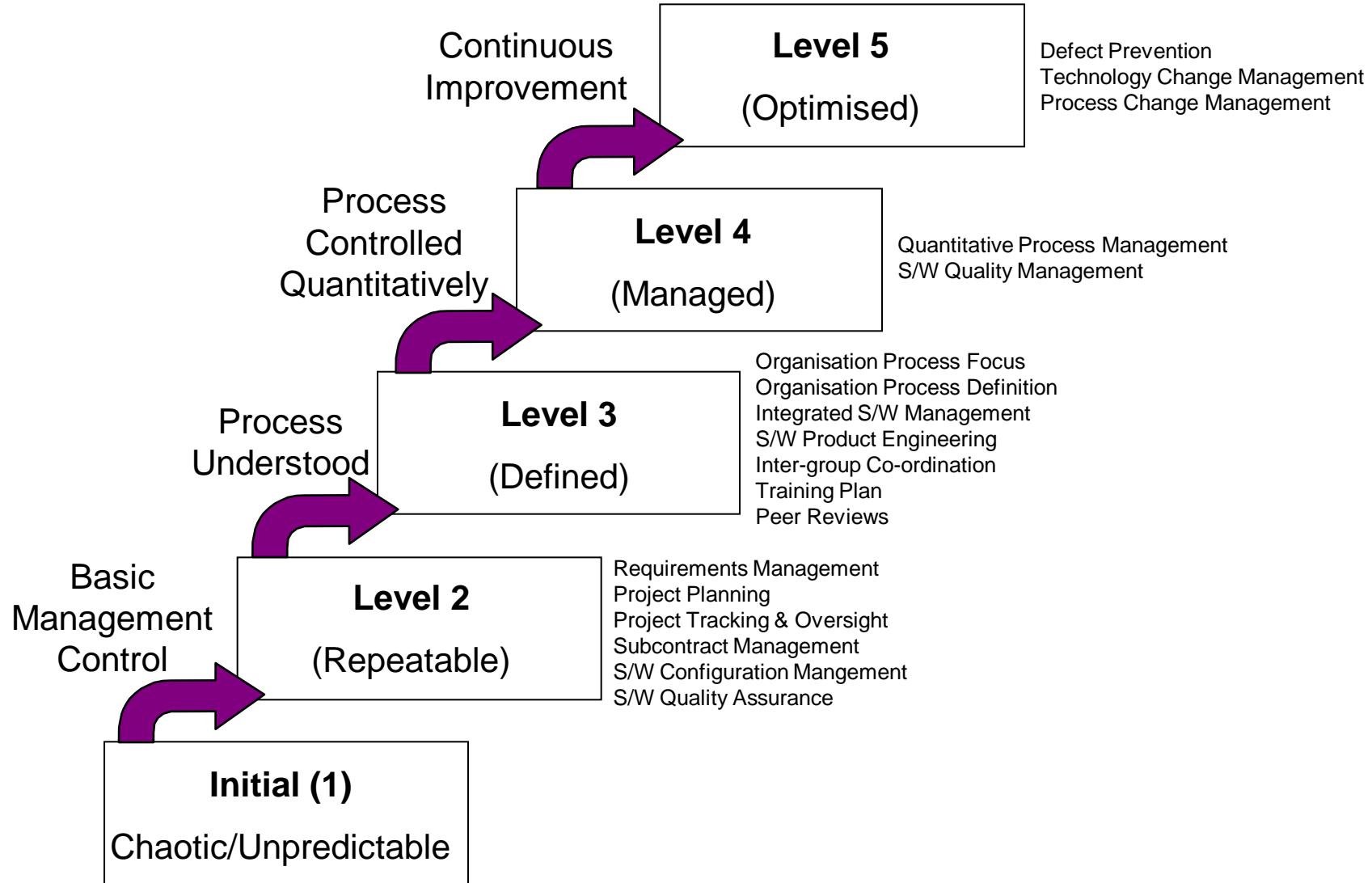
CMM Level 5 PA's – Process Change Management



Process Change Management

- Planning for change and ensuring it is continuous, organisation wide and pro-active
- The first goal is that the process change activities are planned.
- The second goal is that participation is organisation wide
- The third goal is that processes are improved continuously

All CMM Levels



Today's Lecture...

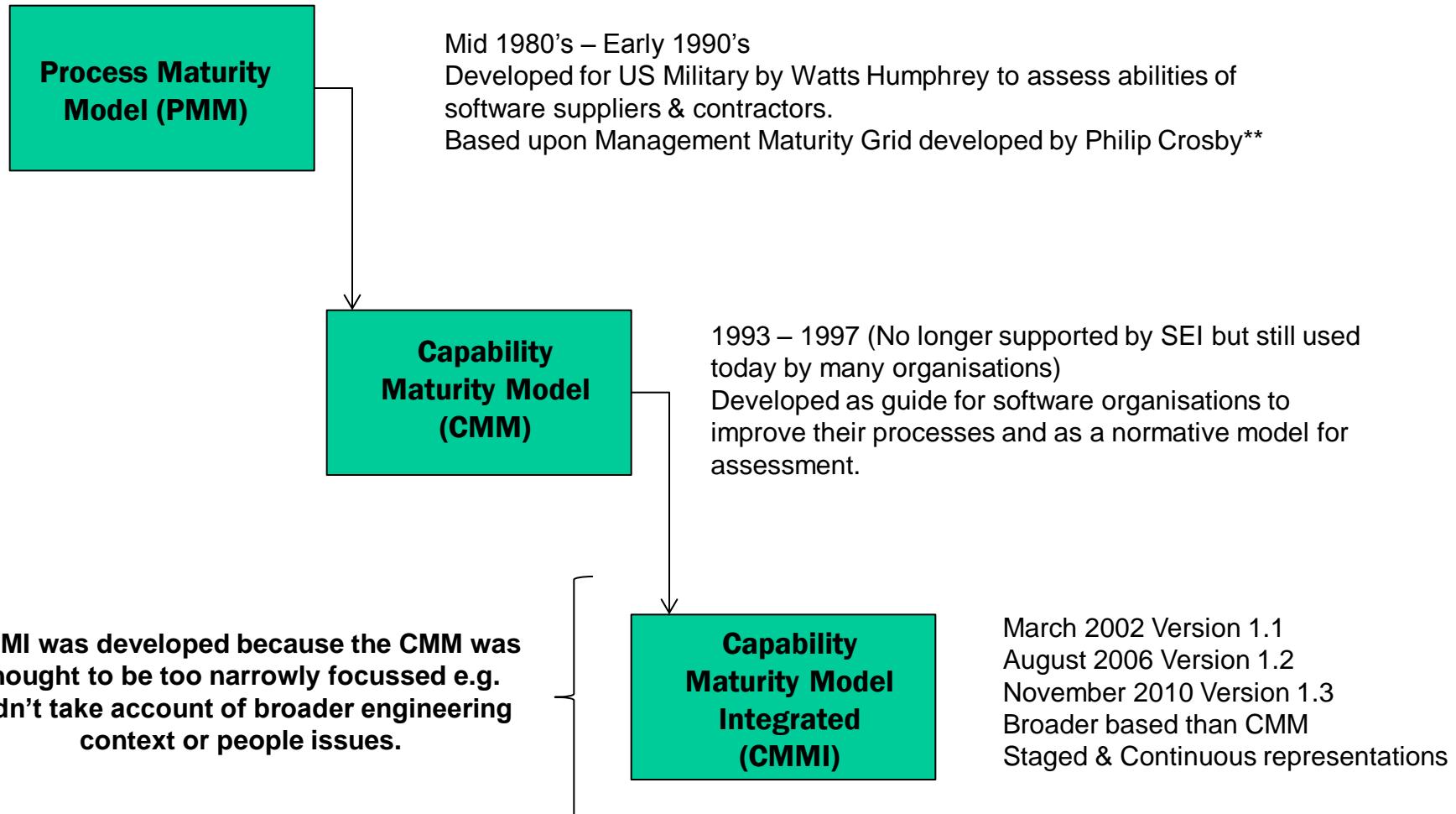
- Software Process Improvement 3

Software Process Improvement

Examination & comparison of two models:

- The Capability Maturity Model (CMM)
- The Capability Maturity Model Integrated (CMMI)

CMU SEI* Software Maturity Models



*Carnegie Mellon University Software Engineering Institute

**"Quality is Free", Crosby, P., 1980

CMMI Background

CMMI Background:

CMMI (Capability Maturity Model Integrated) came about because the SEI (Software Engineering Institute) realised the original CMM had some shortcomings: Examples -

1. CMM was “discipline specific” – i.e. just focussed on software engineering processes – whereas what was needed was enterprise-wide software process improvement
2. CMM didn’t explicitly recognise the associations and linkages between key process areas. For example the linkage between Requirements Management and Project Planning, or the linkages between Project Planning and Software Configuration Management are inferred – not explicit.
3. CMM was criticised for hampering innovation – continuous improvement of existing processes
4. CMM didn’t adequately address the impact that teams and teaming had on software product development
5. CMM didn’t adequately address the issue of software suppliers
6. CMM didn’t give sufficient recognition to topics such as risk management and software testing

CMMI Background

CMMI Evolution:

As a result of these concerns the CMMI (Capability Maturity Model Integrated) came into being. It was an integration of 4 disciplines.

1. **Systems Engineering:** Covering the development of total systems - which may or may not include software or software development
2. **Software Engineering:** Covering the development of software systems – which the original CMM was largely addressing
3. **Integrated Product and Process Development:** Covering the collaboration of relevant stakeholders throughout the life of the product – which addresses teaming and the development environment
4. **Supplier Sourcing:** Covering the existing management of software subcontractors but extended to software acquisition

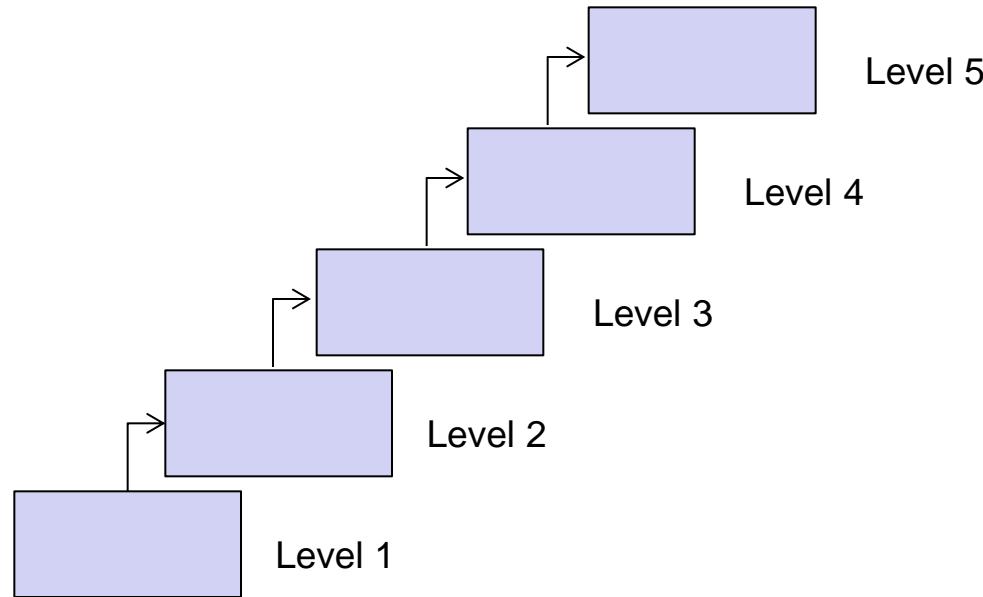
This broadened the scope of the model and hence the number of process areas rose from 18 in the original CMM to 25 in CMMI

As mentioned previously CMMI has two representations: The ‘staged representation’ and The ‘continuous representation’

CMMI Staged Representation

The **Staged Representation** (as with CMM) provides:

1. A systematic, structured way to approach process improvement one stage at a time, ensuring that achieving each stage lays a foundation for the next stage
2. Prescriptive order for making improvements taking the guesswork out of what to improve next.
3. Good way to start for organisations that are new to software process improvement



Staged Representation Criticisms

Major criticisms of the staged representation were:

1. Many organisations became focussed on achieving a particular maturity level – whether they were actually improving the organisation's business or not.
2. Many organisations felt constrained by the model – feeling they were obliged to work on improving what the model was dictating as opposed to what they felt was important for their business.

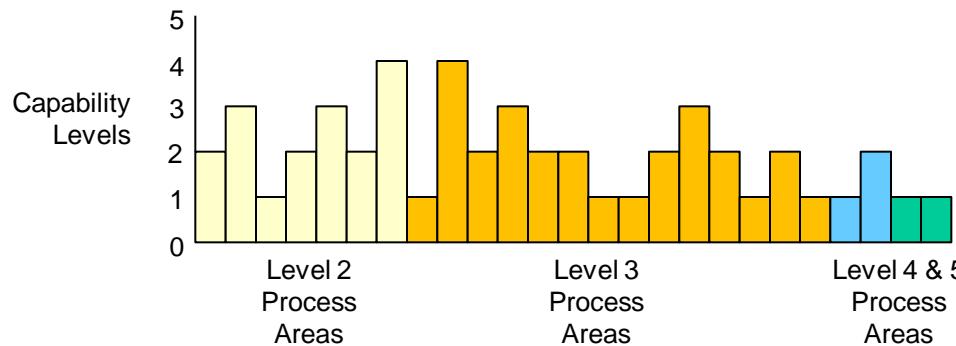
As a result of these concerns SEI supports two different representations:

- 1) a CMMI **staged representation** (as before), and
- 2) a new CMMI **continuous representation**

CMMI Continuous Representation

The **Continuous Representation** provides a more flexible approach to process improvement:

1. Improve the performance of a single process trouble-spot or work on process areas that are more important to the organisation's business objectives.
2. Improve different processes at different rates – but may be limited because of dependencies between process areas
3. Organisation's may strive to have different capability level in different process areas – may want to have a capability level of 2 in one area and 4 in another



CMMI Continuous Representation

To facilitate a **Continuous Representation** SEI introduced the concept of 6 capability levels: CL0->CL5

CL0 = Incomplete Process

CL1 = Performed Process

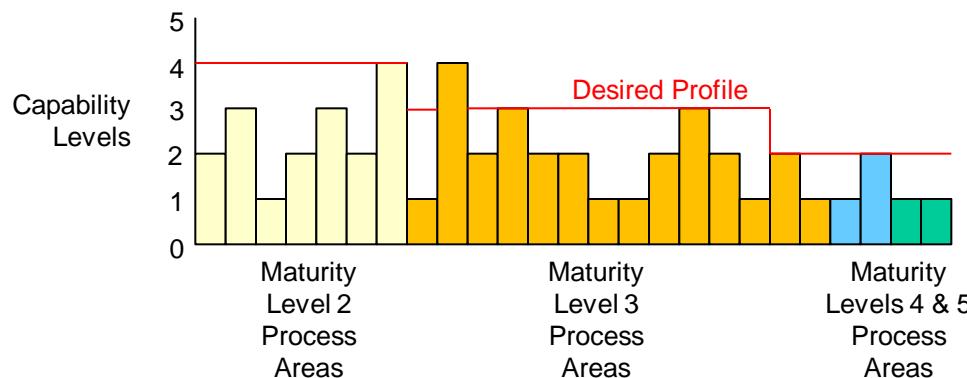
CL2 = Managed Process

CL3 = Defined Process

CL4 = Quantitatively Managed Process

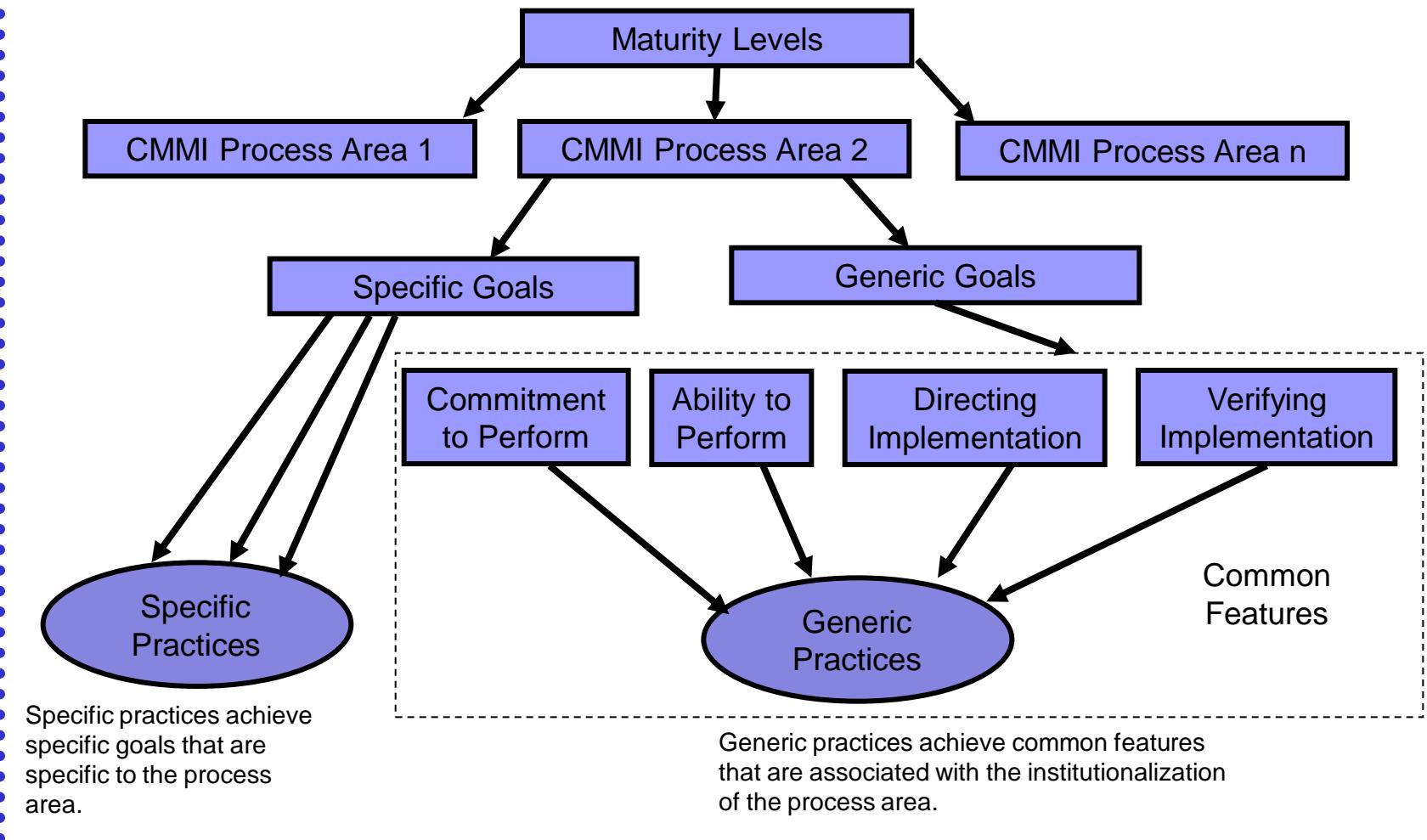
CL5 = Optimizing Process

This allowed software organisations to aim for a desired process profiles for their business and through a technique called 'equivalent staging' could convert their capability level profile to a maturity level.



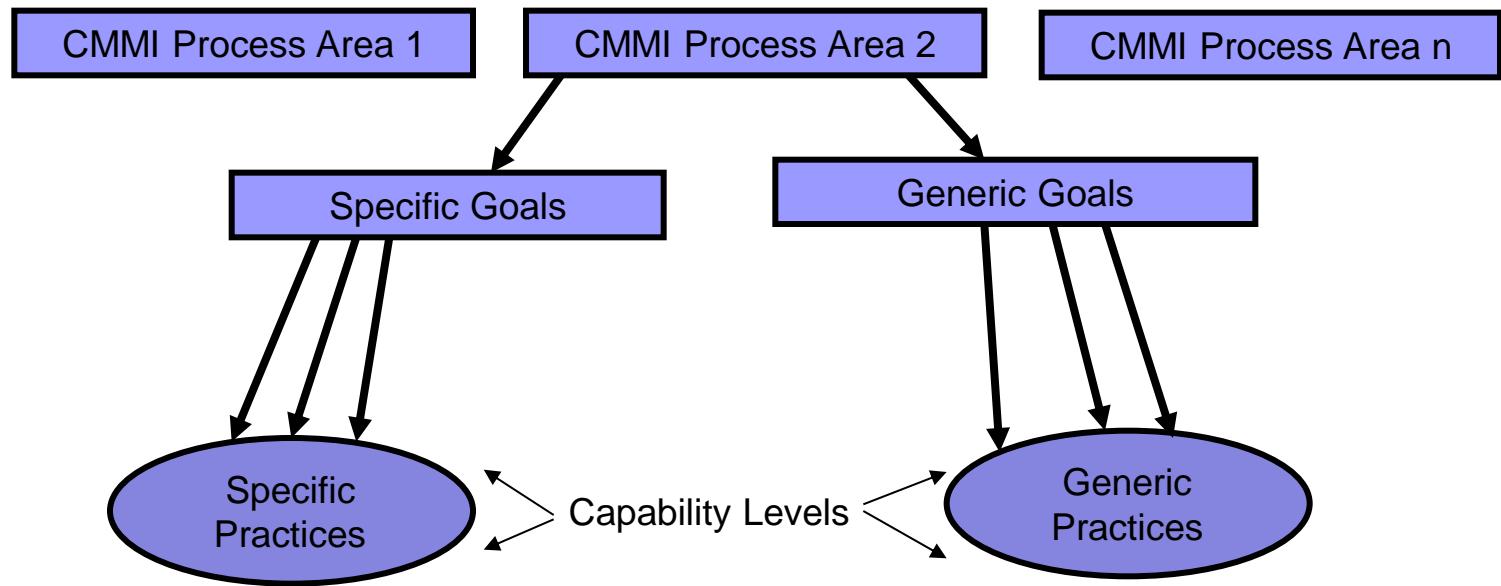
CMMI PA Structure – Staged Representation

- Very similar to the CMM structure. Each maturity level (which is organisational) is made up of a number of process areas.



CMMI PA Structure – Continuous Representation

- Similar to the staged representation but the continuous representation focusses on capability levels within the individual PA's.



- The CMMI continuous representation specifies for each PA the degree of capability that has to be achieved by specific practices and generic practices if the PA is to be appraised at a particular maturity level.

Equivalent Staging

- Even though an organisation may want to pursue CMMI continuous representation there could be a very real need to be able to draw a comparison with another group that is following the staged representation.
- So how do we compare the achievement of capability levels with achieving maturity levels?
- The answer is equivalent staging.

Name	Abbr	ML	CL1	CL2	CL3	CL4	CL5
Requirements Management	REQM	2					
Measurement and Analysis	MA	2					
Project Monitoring and Control	PMC	2					
Project Planning	PP	2					
Process and Product Quality Assurance	PPQA	2					
Supplier Agreement Management	SAM	2					
Configuration Management	CM	2					
Decision Analysis and Resolution	DAR	3					
Product Integration	PI	3					
Requirements Development	RD	3					
Technical Solution	TS	3					
Validation	VAL	3					
Verification	VER	3					
Organizational Process Definition	OPD	3					
Organizational Process Focus	OPF	3					
Integrated Project Management (IPPD)	IPM	3					
Risk Management	RSKM	3					
Integrated Supplier Management	ISM	3					
Organizational Training	OT	3					
Integrated Teaming	IT	3					
Organizational Environment for Integration	OEI	3					
Organizational Process Performance	OPP	4					
Quantitative Project Management	QPM	4					
Organizational Innovation and Deployment	OID	5					
Causal Analysis and Resolution	CAR	5					

CMMI Maturity Level 2 Process Areas

Requirements Management

Measurement and Analysis

Project Monitoring & Control

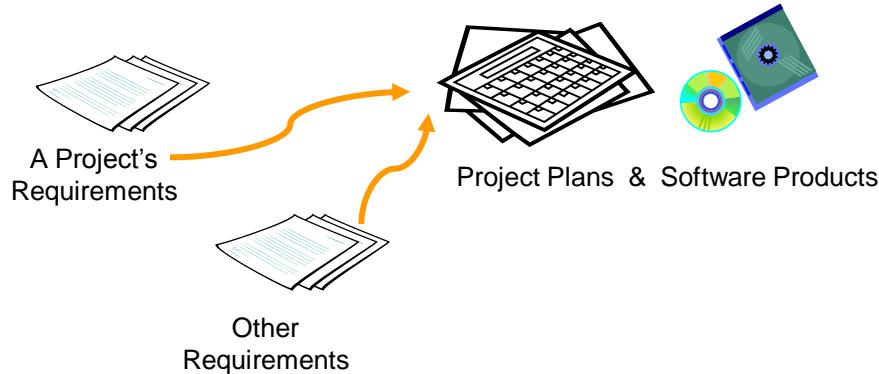
Project Planning

Supplier Agreement Management

Configuration Management

Process & Product Quality Assurance

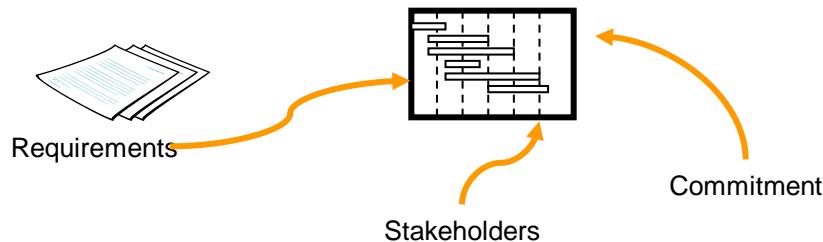
CMMI Level 2 PA's - Requirements Management



Requirements Management (REQM)

- Similar to CMM Requirements Management PA but considers an additional factor.
- To manage the requirements of a project's products and product components and to identify inconsistencies between those requirements and the projects plans and work products.
- Will also take into account other requirements levied by the organisation as it may be trying to turn some of the features being developed for individual customers into standard products that it can sell.
- Specific Goal:
 - Manage the Requirements

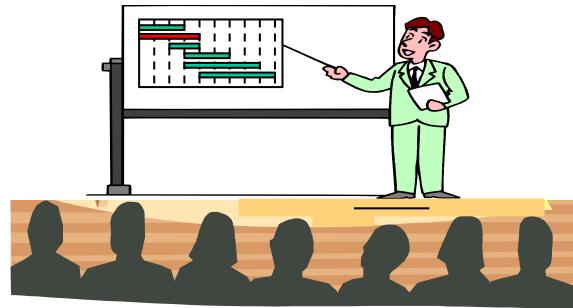
CMMI Level 2 PA's - Project Planning



Project Planning (PP)

- Similar to CMM Project Planning PA.
- To establish and maintain plans that define project activities. Involves developing the project plan, interacting with stakeholders, getting commitment to the plan, and maintaining the plan.
- Specific Goals:
 - Establish Estimates
 - Develop a Project Plan
 - Obtain Commitment to the Plan

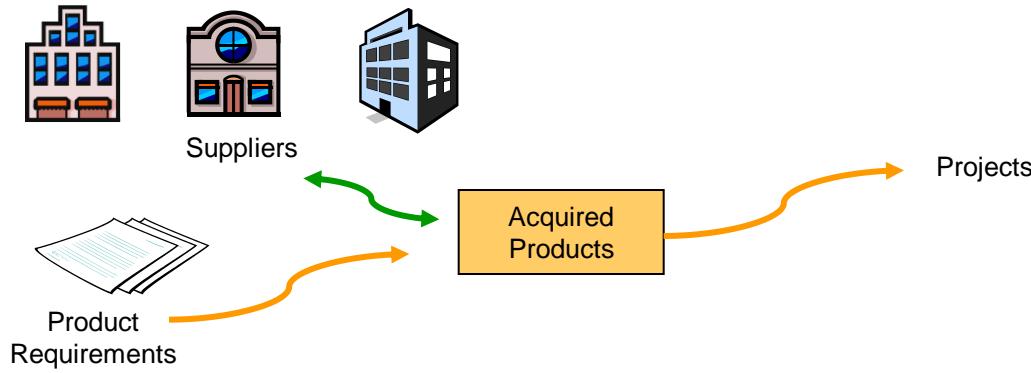
CMMI Level 2 PA's - Project Monitoring & Control



Project Monitoring and Control (PMC)

- Similar to CMM Project Tracking & Oversight PA.
- To provide an understanding of the project's progress so that appropriate corrective actions can be taken when the projects performance deviates significantly from plan.
- Specific Goals:
 - Monitor the Project Against Plan
 - Manage Corrective Actions to Closure

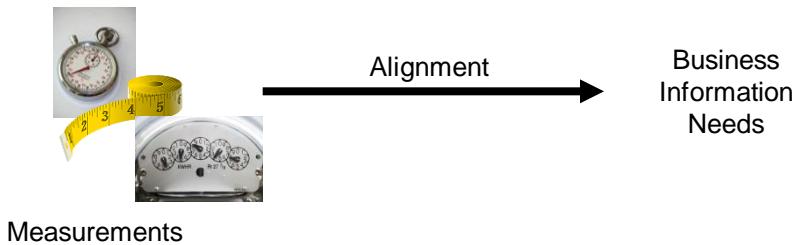
CMMI Level 2 PA's – Supplier Agreement Management



Supplier Agreement Management (SAM)

- Goes beyond the CMM Subcontractor Management PA to include all sorts of suppliers.
- Manage the acquisition of products from suppliers for which there exists a formal agreement. Involves determining the type of acquisition that will be used for the products to be acquired, selecting suppliers, establishing and maintaining agreements with suppliers, executing the supplier agreement, accepting delivery of acquired products, and transitioning the acquired products to the project.
- Specific Goals:
 - Establish Supplier Agreements
 - Satisfy Supplier Agreements

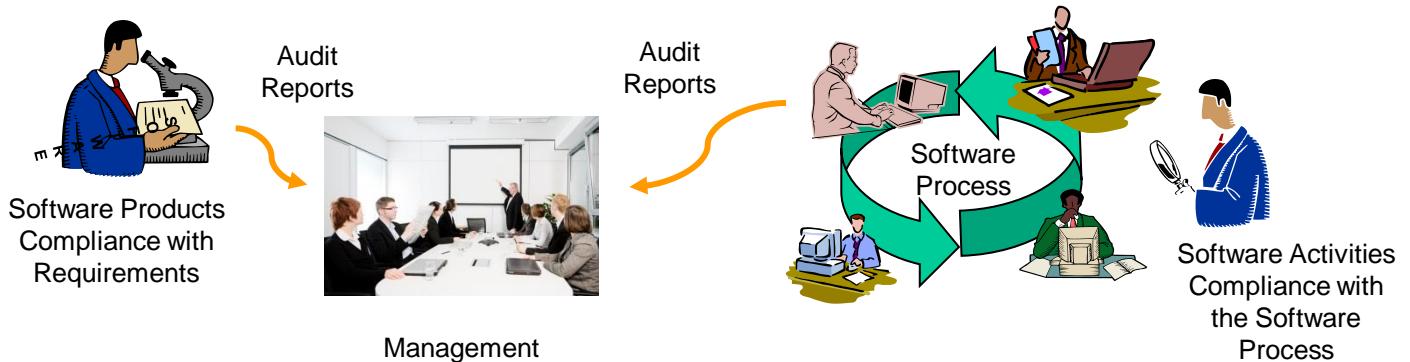
CMMI Level 2 PA's – Measurement & Analysis



Measurement and Analysis (MA)

- In CMM Measurement & Analysis was one of the common features for each PA. In CMMI it has been deliberately made a PA in its own right.
- Purpose: To develop and sustain a measurement capability that is used to support MIS needs. Involves specifying the objectives of measurement and analysis such that they are aligned with identified information needs and objectives, specifying the measurement data collection and analysis mechanisms, specifying the reporting mechanisms, implementing the data collection and analysis, and providing objective results for making informed decisions.
- Specific Goals:
 - Align Measurement & Analysis Activities
 - Provide Measurement Results

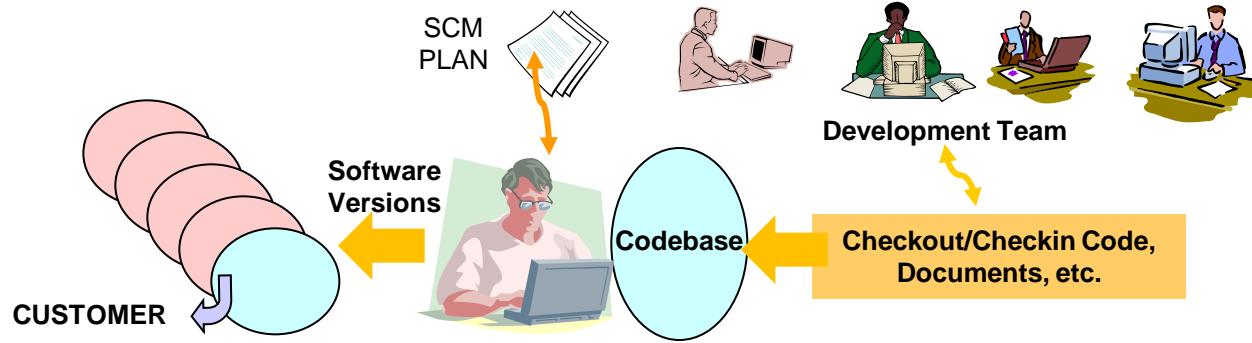
CMMI Level 2 PA's – Process & Product Quality Assurance



Process and Product Quality Assurance (PPQA)

- Practically the same as in CMM Software Quality Assurance PA.
- Purpose: To provide staff and management with objective insight into processes and associated work products. Involves evaluation of processes and products against applicable descriptions, identifying non-compliance issues, providing feedback to staff and managers, and ensuring non-compliance issues are addressed.
- Specific Goals:
 - Objectively Evaluate Processes and Work Products
 - Provide Objective Insight

CMMI Level 2 PA's - Configuration Management



Configuration Management (CM)

- Practically the same as CMM Software Configuration Management PA.
- Purpose: To establish and maintain the integrity of work products using configuration identification, configuration control, configuration status accounting, and configuration audits. Involves identifying the work products that compose the baselines at given points, controlling changes to the baselines, building the work products, maintaining the integrity of the baselines, and providing accurate status information to stakeholders.
- Specific Goals:
 - Establish Baselines
 - Track & Control Changes
 - Establish Integrity

CMMI Maturity Level 3 Process Areas

Requirements Development

Technical Solution

Product Integration

Verification

Validation

Organizational Process Focus

Organizational Process Definition

Organizational Training

Integrated Project Management

Integrated Product and Process Development

Risk Management

Integrated Teaming

Integrated Supplier Management

Decision Analysis and Resolution

Organizational Environment for Integration

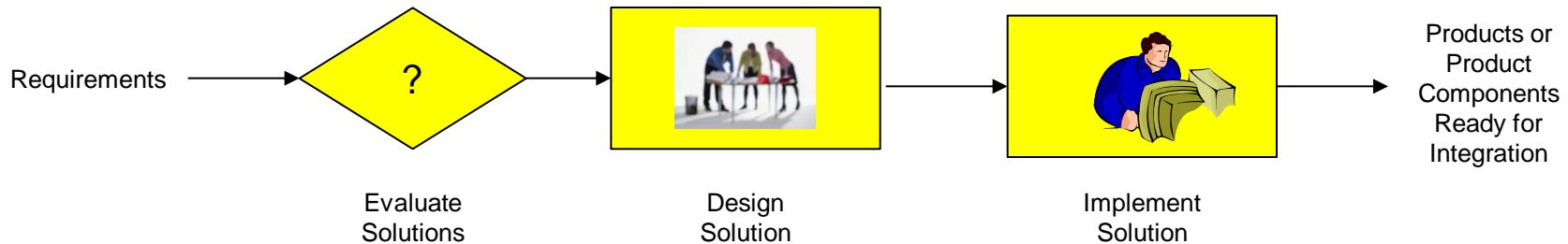
CMMI Level 3 PA's - Requirements Development



Requirements Development (RD)

- New – didn't feature in CMM
- Produce and analyse all customer requirements which are refined into product requirements, and product component requirements, rather than just product-level requirements for a single project instance (L2).
- Specific Goals:
 - Develop Customer Requirements
 - Develop Product Requirements
 - Analyse & Validate Requirements

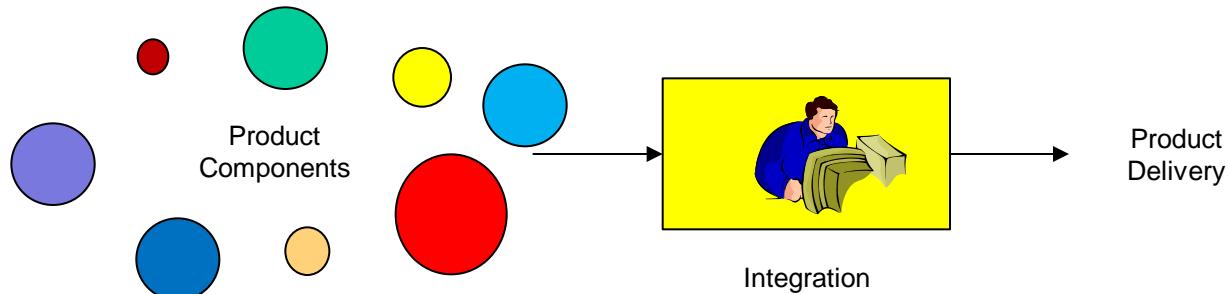
CMMI Level 3 PA's – Technical Solution



Technical Solution (TS)

- In CMM would be associated with Software Product Engineering PA but extends to the design/implementation of processes and services.
- Design, develop and implement solutions to the requirements. Solutions, designs, and implementations, includes products, product components, and product-related life-cycle processes including those for services.
- Specific Goals:
 - Select Product-Component Solutions
 - Develop the Design
 - Implement the Product Design

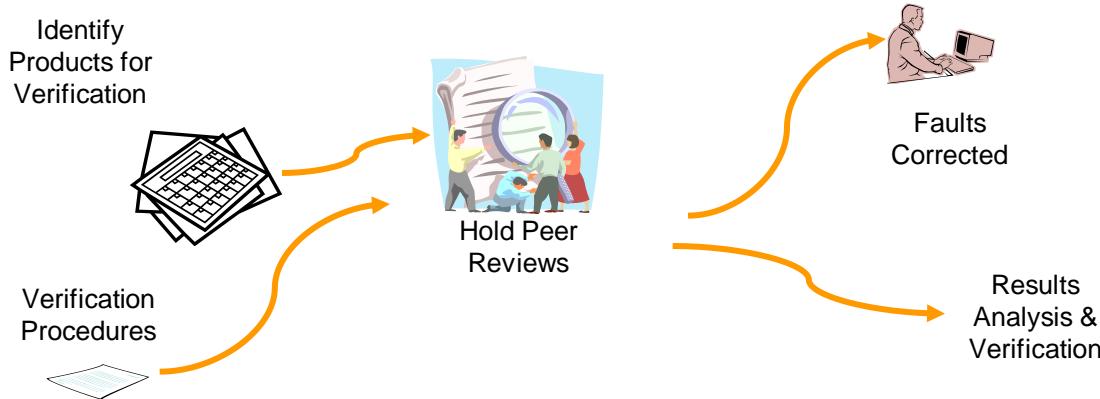
CMMI Level 3 PA's – Product Integration



Product Integration (PI)

- New. In CMM would be associated with Software Product Engineering & Software Configuration Management but this is more specific.
- Assemble the product from the product components, ensure that the product, as integrated, functions properly, and deliver the product. Particular attention is paid to the interfaces between product components.
- Specific Goals:
 - Prepare for Product Integration
 - Ensure Interface Compatibility
 - Assemble Product Components and Deliver the Product

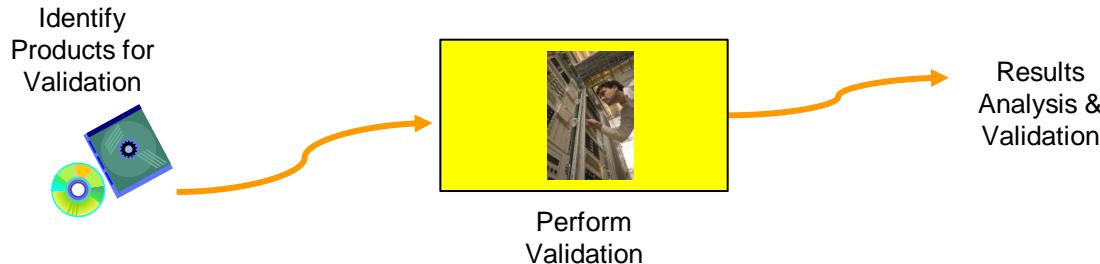
CMMI Level 3 PA's – Verification



Verification (VER)

- New. In CMM this was in Peer Reviews PA and Software Product Engineering PA. In CMMI this testing technique is made explicit.
- Ensure that selected work products meet their specification. Involves verification preparation, verification performance, and the identification of corrective actions. Ensures the product was ‘built the right way’ – peer reviews play a significant part in verification.
- Specific Goals:
 - Prepare for Verification
 - Perform Peer Reviews
 - Verify Selected Work Products

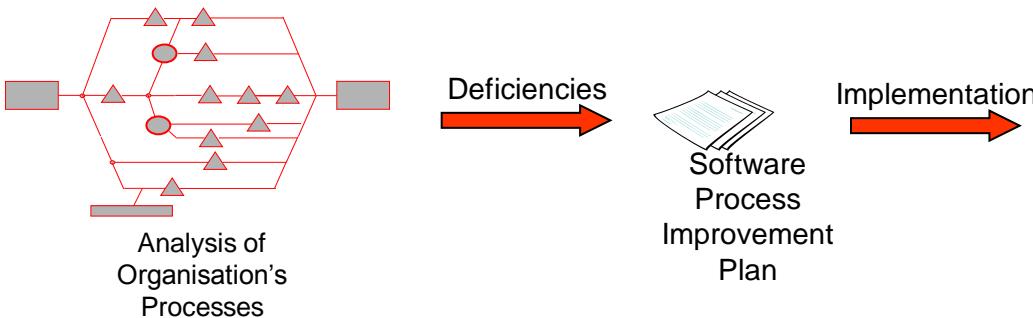
CMMI Level 3 PA's – Validation



Validation (VAL)

- In CMM was buried within Software Product Engineering PA. In CMMI explicitly recognised as part of the testing discipline
- Demonstrate that a product or product component fulfils its intended use when placed in its intended environment. Ensures 'you built the right thing' – uses many of the test techniques discussed earlier in the course.
- Specific Goals:
 - Prepare for Validation
 - Validate the Product or Product Components

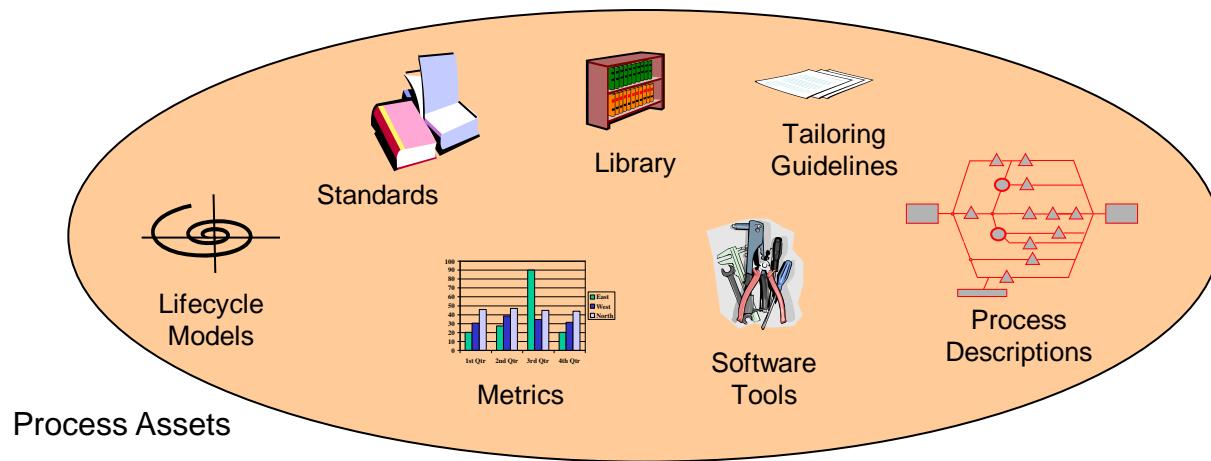
CMMI Level 3 PA's – Organisation Process Focus



Organizational Process Focus (OPF)

- Largely unchanged from CMMI PA of same name.
- Plan and implement organizational process improvement based on a thorough understanding of the current strengths and weaknesses of the organisation's processes and process assets.
- Specific Goals:
 - Determine Process Improvement Opportunities
 - Plan & Implement Process Improvement Activities

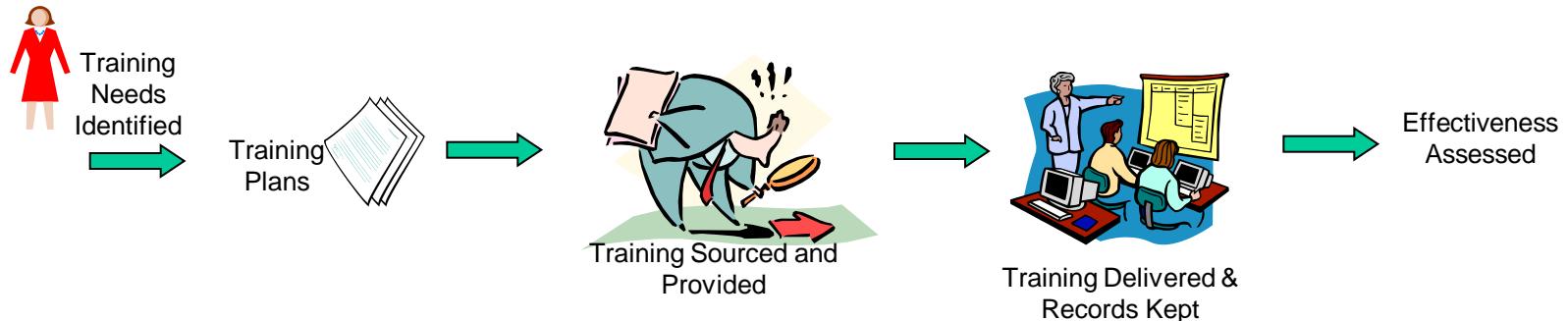
CMMI Level 3 PA's – Organisation Process Definition



Organizational Process Definition (OPD)

- Largely unchanged from CMM PA of same name.
- Establish and maintain a usable set of organizational process assets. These enable consistent process performance across the organization and provide a basis for cumulative, long-term benefits to the organization.
- Specific Goal:
 - Establish Organizational Process Assets

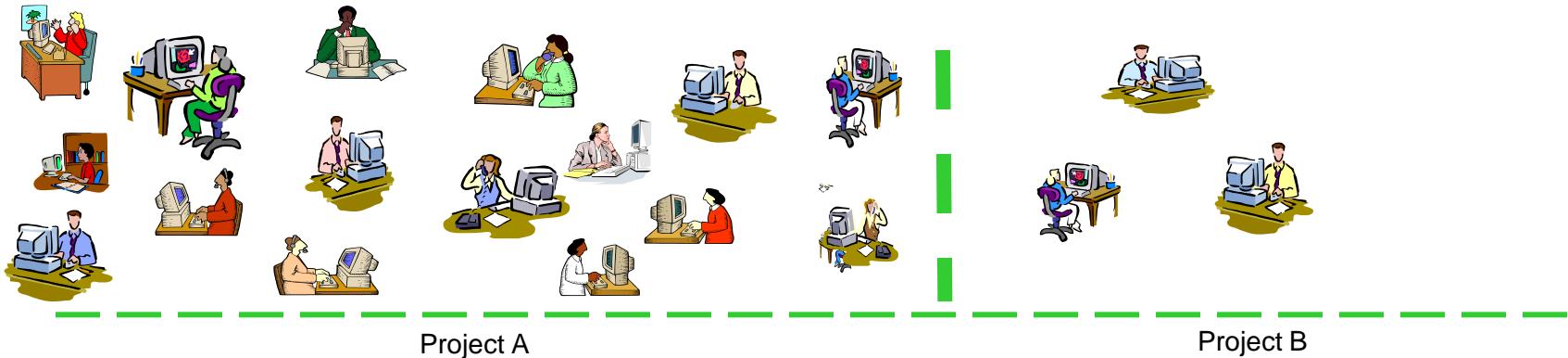
CMMI Level 3 PA's – Organizational Training



Organizational Training (OT)

- Largely unchanged from CMM Training Plan PA but name now better reflects organisational focus.
- Develop the skills and knowledge of people so they can perform their roles effectively and efficiently. Involves identifying the training that is needed, sourcing and providing the training, maintaining the training capability, maintaining training records, assessing training effectiveness.
- Specific Goals:
 - Establish an Organizational Training Capability
 - Provide Necessary Training

CMMI Level 3 PA's – Integrated Project Management



Integrated Project Management (IPM)

- Practically unchanged from CMM Integrated Software Management PA but through incorporation of **Integrated Product and Process Development (IPPD)** it has greater emphasis on team structure & team integration.
- Establish and manage the project and the involvement of the relevant stakeholders according to an integrated and defined process. This process is tailored from the organization's set of standard processes.
- Specific Goals:
 - Use the Project's Defined Process
 - Coordinate & Collaborate with Relevant Stakeholders
 - Use the Project's Shared Vision for IPPD
 - Organize Integrated Teams for IPPD

CMMI Level 3 PA's – Integrated Product & Process Development



Integrated Product and Process Development (IPPD)

- Teams and teaming structures did not feature strongly in CMM
- Integrated Product and Process Development (IPPD) is not regarded as a process area in its own right but is regarded as a augmentation of Integrated Project Management PA and relies upon the Organizational Environment for Integration PA for it's supporting infrastructure.
- Purpose: To establish a shared vision for the project supporting collaboration between all stakeholders and structures for integrated approaches that will carry out the objectives of the project.

CMMI Level 3 PA's – Risk Management



Risk Management (RSKM)

- In CMM risk management was buried within the Project Planning and Project Tracking & Oversight PA's. In CMMI risk is explicitly managed.
- Identify potential problems before they occur so that risk-handling activities can be planned and invoked as needed across the life of the product or project to mitigate adverse impacts on achieving objectives.
- Specific goals:
 - Prepare for Risk Management
 - Identify and Analyse Risks
 - Mitigate Risks

CMMI Level 3 PA's – Integrated Teaming



Integrated Teaming (IT)

- New. Teaming didn't feature in CMM.
- Form and sustain an integrated team for the development of work products. The integrated team is composed of team members who generate and implement decisions for the work product being developed and are collectively responsible for its delivery.
- Specific Goals:
 - Establish Team Composition
 - Govern Team Operation

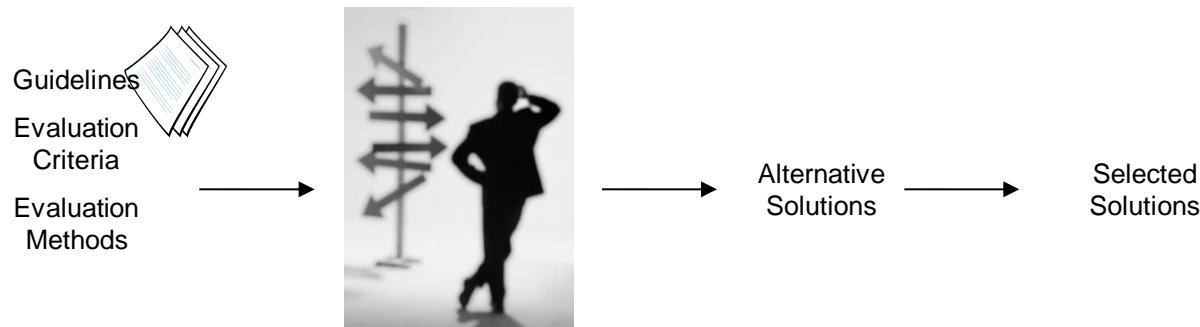
CMMI Level 3 PA's – Integrated Supplier Management



Integrated Supplier Management (ISM)

- New. In CMM Subcontractor Management PA was at Level 2 and reactive.
- More proactive than Supplier Agreement Management PA to identify sources of products that may be used to satisfy the projects requirements and to manage selected suppliers while building, maintaining and emphasizing a cooperative project-supplier relationship.
- Specific Goals:
 - Analyse and Select Sources of Products
 - Coordinate Work with Suppliers

CMMI Level 3 PA's – Decision Analysis and Resolution



Decision Analysis and Resolution (DAR)

- New. Decision making processes did not feature in CMM.
- Analyse possible decisions using a formal evaluation process that evaluates identified alternatives against established criteria. Involves establishing guidelines to determine which issues should be subjected to a formal evaluation process and then applying a formal evaluation process to these issues. A formal evaluation process involves establishing the criteria for evaluating alternatives, identifying the alternative solutions, selecting methods for evaluating alternatives, performing the evaluation, and selecting recommended solutions from the alternatives.
- Specific Goals:
 - Evaluate Alternatives

CMMI Level 3 PA's – Organizational Environment for Integration



Organizational Environment for Integration (OEI)

- Did not feature in CMM.
- Provide an **Integrated Product and Process Development (IPPD)** infrastructure, and manage people for integration. Successful integration of business and technical elements in projects is dependent on substantive and proactive organisational processes and guidelines. The organisation must raise performance expectations from all projects while providing mechanisms that stimulate both team and individual excellence.
- Specific Goals:
 - Provide IPPD infrastructure
 - Manage People for Leadership

CMMI Maturity Level 4 Process Areas

Organization Process Performance

Quantitative Project Management

CMMI Level 4 PA's – Organization Process Performance



Organizational Process Performance (OPP)

- In CMM both the Quantitative Process Management and Software Quality Management PA's combined to address this CMMI PA which is at the organisational level.
- Establish and maintain a quantitative understanding of the performance of the organisation's set of standard processes in support of quality and process performance objectives, and to provide the process performance data, baselines, and models to quantitatively manage the organisation's projects. Measures are composed of both process and product measures.
- Specific Goals:
 - Establish Performance Baselines and Models

CMMI Level 4 PA's – Quantitative Project Management

$$\begin{aligned} \frac{1}{V} \int z dV &= \frac{\pi r^2}{VH^2} \int_0^h (z^3 - 2z^2H + zH^2) dz \\ &= \frac{\pi r^2}{VH^2} \left[\frac{z^4}{4} - \frac{2z^3H}{3} + \frac{z^2H^2}{2} \right]_0^h \\ &= \frac{\pi r^2 h^4}{VH^2} \left[\frac{1}{4} - \frac{2H}{3h} + \frac{H^2}{2h^2} \right]. \end{aligned}$$

Quantitative Project Management (QPM)

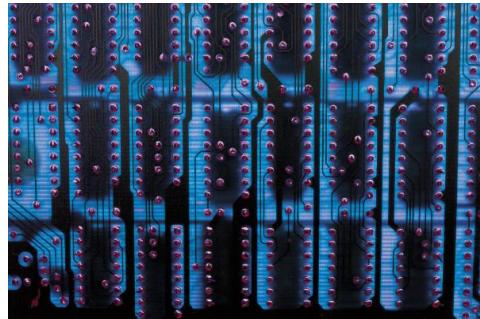
- In CMM baselines were established at the organisational level using Quantitative Process Management and Software Quality Management PA's but not at the project level as is the case here.
- Quantitatively manage the project's defined process to achieve the project's established quality and process-performance objectives. Done by applying suitable statistical and analytical techniques.
- Specific Goals:
 - Quantitatively Manage the Project
 - Statistically Manage Sub-process Performance

CMMI Maturity Level 4 Process Areas

Organizational Innovation and Deployment

Causal Analysis and Resolution

CMMI Level 5 PA's – Organizational Innovation and Deployment



Organizational Innovation and Deployment (OID)

- In CMM this PA would have been covered by the Process Change Management and Technology Change Management PA's.
- Select and deploy incremental and innovative improvements that measurably improve the organisation's processes and technologies, enhancing the organisation's ability to meet its quality and process-performance objectives.
- Specific Goals:
 - Select Improvements
 - Deploy Improvements

CMMI Level 5 PA's – Causal Analysis and Resolution



Causal Analysis and Resolution (CAR)

- In the CMM this PA was covered by Defect Prevention PA
- Identify causes of defects and other problems and take action to prevent them occurring in the future. Involves identifying and analyzing causes of defects and other problems using quantitative measures, and taking specific actions that demonstrably improve products and processes through the prevention of such defects.
- Specific Goals:
 - Determine Causes of Defects
 - Address Causes of Defects

CMMI PA's – Summary

Name	Abbr	ML	CL1	CL2	CL3	CL4	CL5
Requirements Management	REQM	2					
Measurement and Analysis	MA	2					
Project Monitoring and Control	PMC	2					
Project Planning	PP	2					
Process and Product Quality Assurance	PPQA	2					
Supplier Agreement Management	SAM	2					
Configuration Management	CM	2					
Decision Analysis and Resolution	DAR	3					
Product Integration	PI	3					
Requirements Development	RD	3					
Technical Solution	TS	3					
Validation	VAL	3					
Verification	VER	3					
Organizational Process Definition	OPD	3					
Organizational Process Focus	OPF	3					
Integrated Project Management (IPPD)	IPM	3					
Risk Management	RSKM	3					
Integrated Supplier Management	ISM	3					
Organizational Training	OT	3					
Integrated Teaming	IT	3					
Organizational Environment for Integration	OEI	3					
Organizational Process Performance	OPP	4					
Quantitative Project Management	QPM	4					
Organizational Innovation and Deployment	OID	5					
Causal Analysis and Resolution	CAR	5					



Today's Lecture...

- Software Process Improvement 4

Software Process Improvement

There are many different models & methodologies for improving the software process

We are going to take a closer look at 3 of them:

- Lean
- Maturity Models
- 6-Sigma

Software Process Improvement

Each has a distinct focus:

- Lean
 - The **elimination of waste**, where waste is defined as having no added value to the software product or service being provided
- Maturity Models
 - Guiding software organisations' improvement efforts by providing normative references that facilitate **continuous improvement** in stages
- 6-Sigma
 - The **removal of variability** within a process, where variability is defined in terms of exceeding statistically calculated limits

Software Process Improvement

Sigma: A measure of goodness

- Sigma: How many times a customer's requirements were not met (a defect), given a million opportunities



- With each rise in sigma level (1,2,3, etc), the number of defects is reduced exponentially.

Software Process Improvement

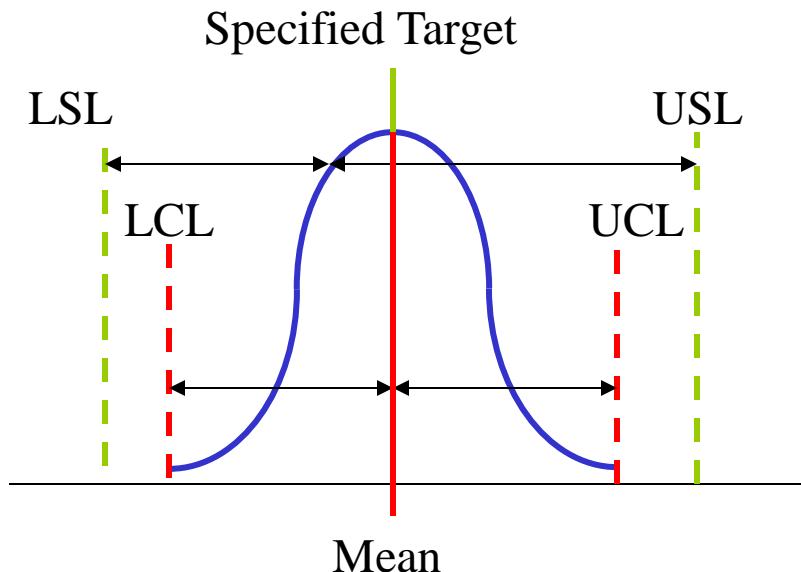
6-Sigma: Origins

- Originally developed by Motorola in 1986 it has now made its way into many different domains including software.
- Kicked off in 1979 when Art Sundry, a Motorola sales manager, wrote himself into the history books by telling Bob Galvin, the CEO and son of the founder, “Our quality stinks”.
- Galvin went on the road and visited Motorola manufacturing plants and users all over the world and confirmed what Sundry had said – their quality was lousy!
- Galvin realised Motorola would continue its downward trend unless something was done. He also believed that business performance improvement would follow quality improvement – so he instituted the 10x programme
- Which in 1986 was made an official programme called 6-Sigma

Software Process Improvement

- Process Capability

❖ If the process variation is less than the specification limits then the process is capable of meeting customer specifications but again there could still be defective product if the process average is not centred on the specification target.



Software Process Improvement

6-Sigma: Origins

- In business, variation often translates into products or services that do not meet customer specifications i.e. waste (ref: USL & LSL)
- Remember! customers may be external or internal to a business
- 6-Sigma initiatives seek to reduce variation in business processes by identifying and then removing the causes of defects.
- A defect in 6-sigma terminology is any output that does not meet customer specifications or could lead to an output that does not meet customer specifications
- 6-Sigma focuses on measuring the quality of a process in terms of the number of defects it produces
- The higher the sigma value the lower the probability of a defect occurring.

Software Process Improvement

6-Sigma: Who's using it?

3M, Alcoa, GE , Abbott Pharmaceuticals, Johnson Controls , Ford, Pratt & Whitney , Northrop Grumman , BP, Air France, Lufthansa, Conoco, Halliburton, Ferrari , Morelli, Boeing, Motorola, TRW, Wipro, Tata Group, Singapore Airlines, Honeywell, Sun Microsystems, Citigroup, Jaguar, Rolls Royce, Bombardier, Home Depot , Amazon.com, Sprint, Apple Computer, IBM, Sony, Ericsson, Nokia, Canon, Hitachi , Maytag, Polaroid, Lockheed Martin, Dupont
+ lots and lots of other companies worldwide.

Software Process Improvement

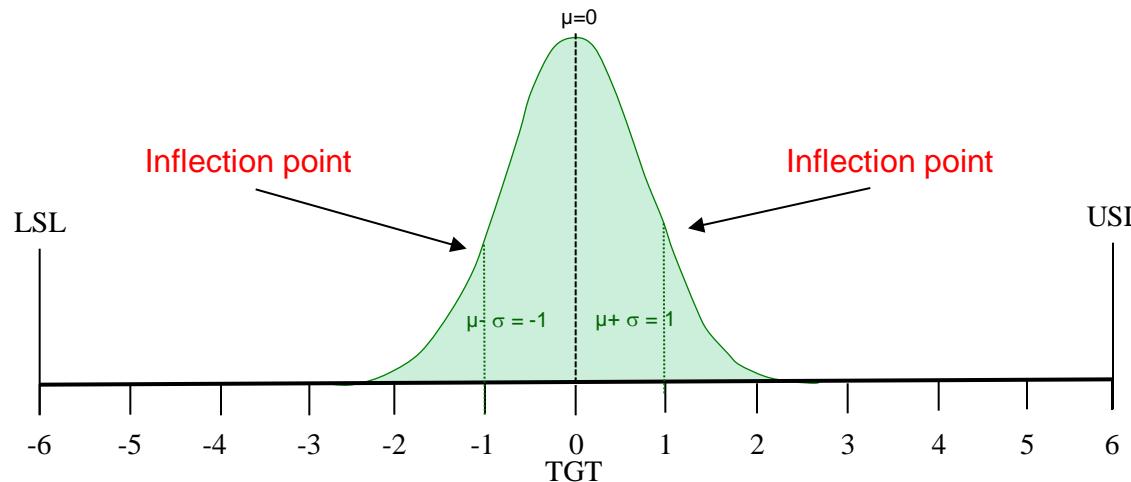
6-Sigma: What are they saving?

Company	Annual Savings
General Electric	\$2.0+ billion
JP Morgan Chase	\$1.5 billion
Texas Instruments	\$600 million
Johnson & Johnson	\$500 million
Honeywell	\$600 million

Software Process Improvement

6-Sigma: Theory

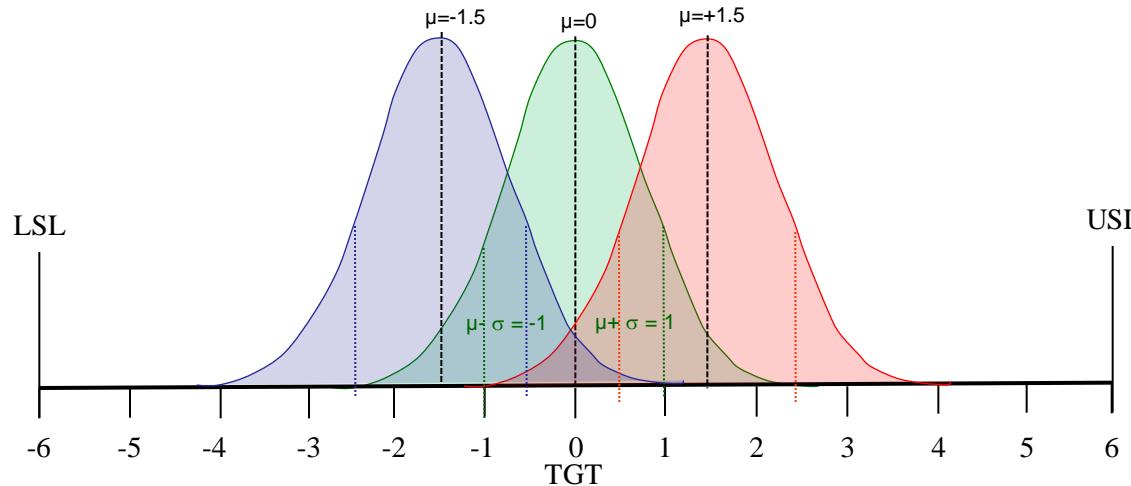
- Sigma (σ) is a Greek letter used in statistics to denote the standard deviation i.e. variability from the mean or average
- Calculated as the distance on the horizontal axis between the mean, μ , and the curve's inflection point. The greater this distance, the greater is the spread of values encountered.



- Theory is that if one has six standard deviations between the process mean and the nearest specification limit, practically no items will fail to meet specifications

Software Process Improvement

6-Sigma: Basics



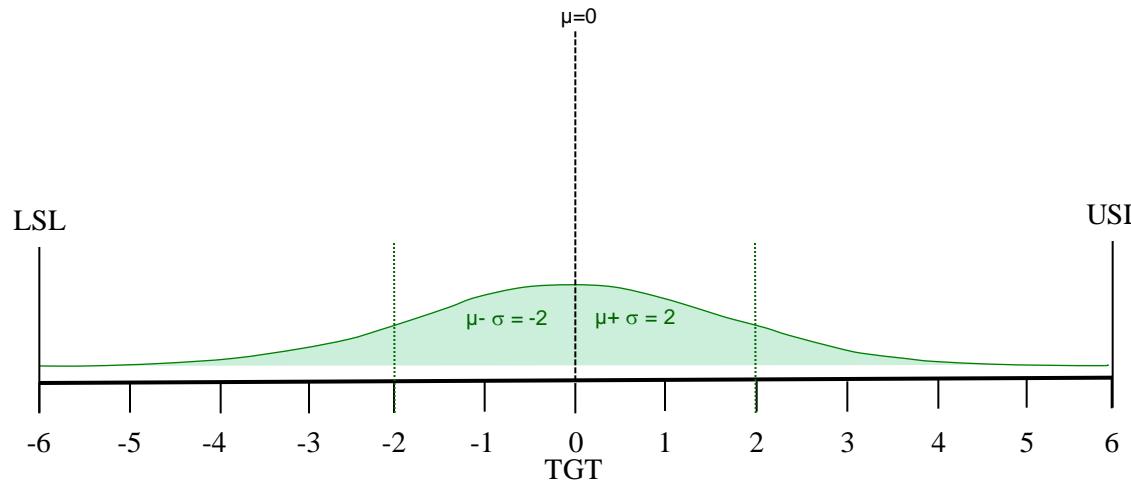
For the green curve shown above, $\mu = 0$ and $\sigma = 1$. The upper and lower specification limits (USL and LSL, respectively) are at a distance of 6σ from the mean.

Even if the mean were to move right or left by 1.5σ at some point in the future (1.5 sigma shift, coloured red and blue), still process $\sigma = 1$, there is still a good safety cushion (4.5 sigma for the red and blue curves).

But if the mean of the process moves further away from the specification target, fewer standard deviations will fit between the mean and the nearest specification limit increasing the likelihood of items outside specification.

Software Process Improvement

6-Sigma: Basics



Similarly if the process standard deviation (σ) goes up (greater process spread) variability in the process grows too. For the green curve shown above, $\mu = 0$ and $\sigma = 2$. Even though the upper and lower specification limits are still at a distance of 6σ from the mean ($\mu = 0$ relative to the target) three process standard deviations will only just fit between the mean and specified limits so this process is at 3 sigma.

A shift away from the target ($\mu > 0$ or $\mu < 0$) coupled with increasing process spread will only make things much worse.

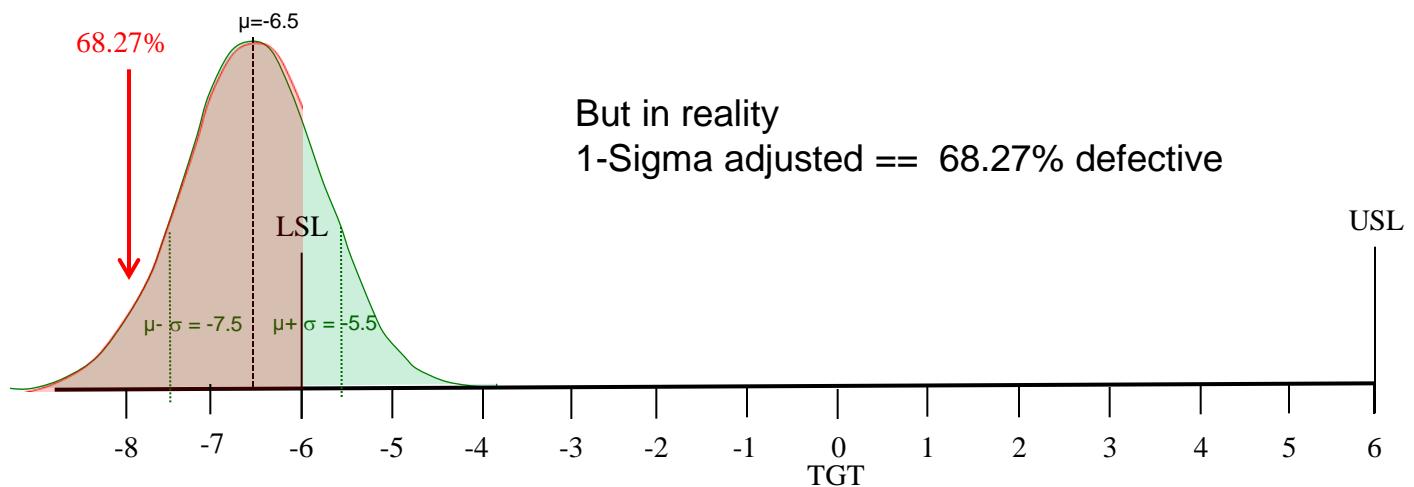
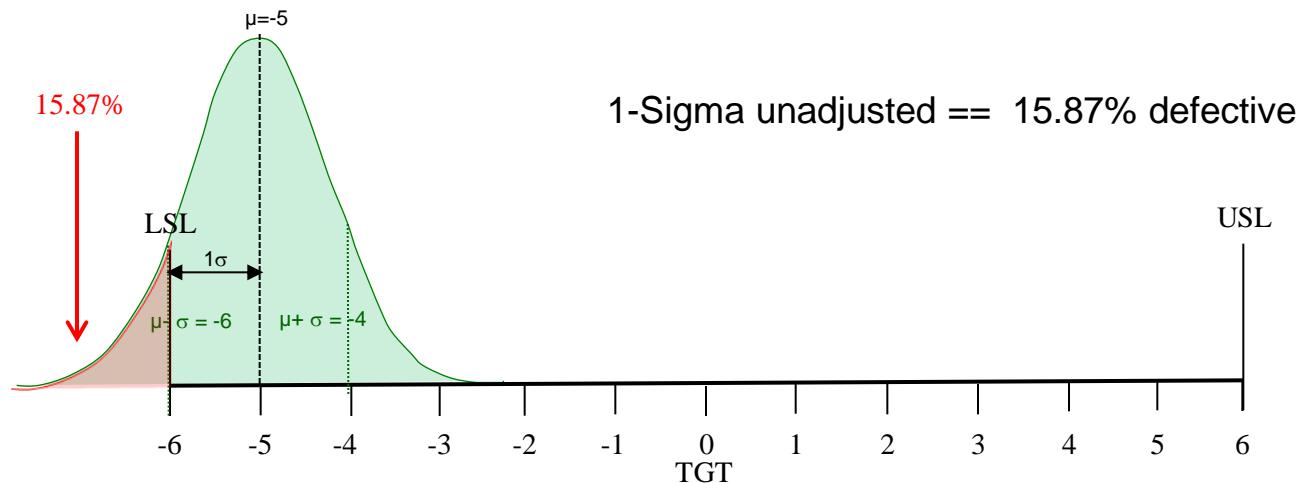
Software Process Improvement

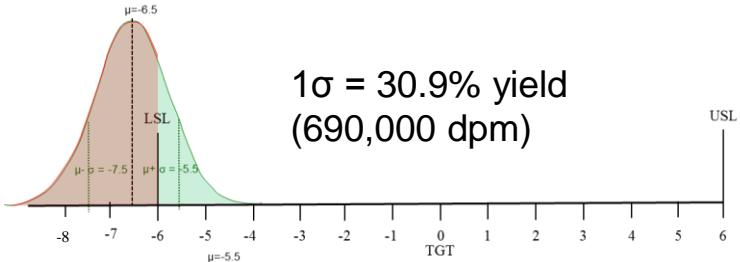
6-Sigma: Basics

- But 6 sigma actually translates to about 2 defects per billion opportunities. The 3.4 defects per million opportunities, which we normally define as 6 sigma, really corresponds to a sigma value of 4.5! Why is this?
- Experience shows that over the longer term process variation will increase (more special causes of variation). So a 1.5 sigma shift is introduced to take account of this hence a process that fits 6 sigma between the process mean and the nearest specification limit in a short-term study will in the long term fit only 4.5 sigma.

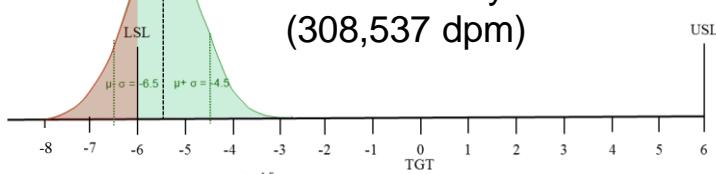
Software Process Improvement

6-Sigma: Basics

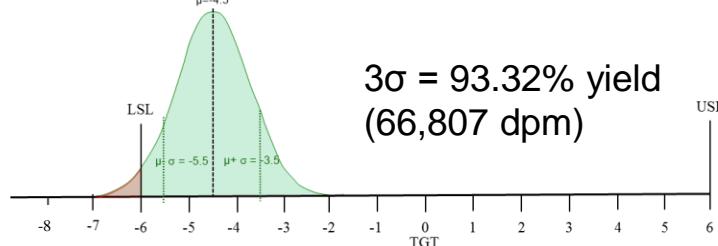




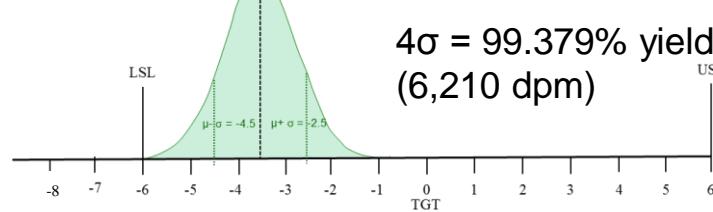
$1\sigma = 30.9\%$ yield
(690,000 dpm)



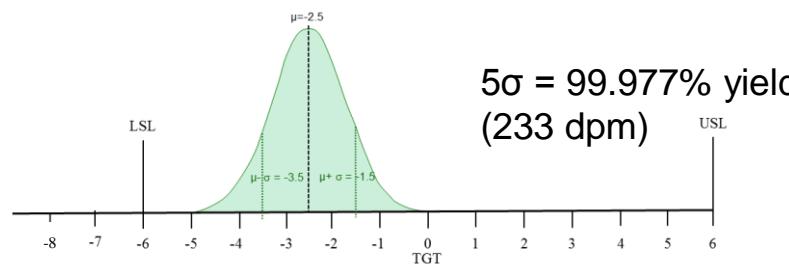
$2\sigma = 69.1\%$ yield
(308,537 dpm)



$3\sigma = 93.32\%$ yield
(66,807 dpm)



$4\sigma = 99.379\%$ yield
(6,210 dpm)

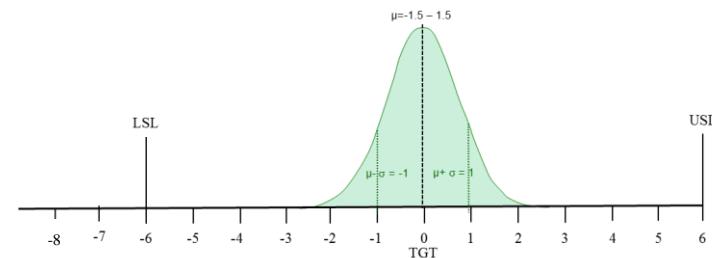


$5\sigma = 99.977\%$ yield
(233 dpm)

6-Sigma: Basics

Sigma Values – Their yield in terms of % good product (or service) & number of defects per million

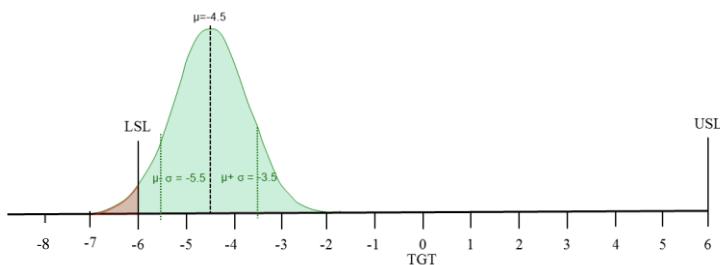
$6\sigma = 99.9997\%$ yield (3.4 dpm)



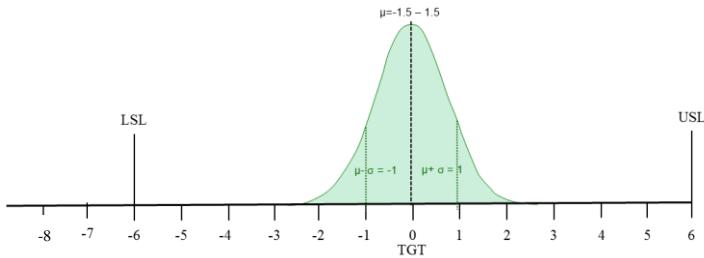
Software Process Improvement

6-Sigma: In Real Terms

$3\sigma = 93.32\% \text{ yield (66,807 dpm)}$



$6\sigma = 99.9997\% \text{ yield (3.4 dpm)}$



Comparison	
3σ	6σ
1,350 poorly performed surgical operations in 1 week	1 poorly performed surgical operation in 20 years
Over 40,500 new-born babies dropped by doctors or nurses each year	3 new-born babies dropped by doctors or nurses in 100 years
5 missed landings at any major US airport each day	1 missed landing in 10 years at all US airports
54,000 lost pieces of mail per hour	35 lost pieces of mail per year

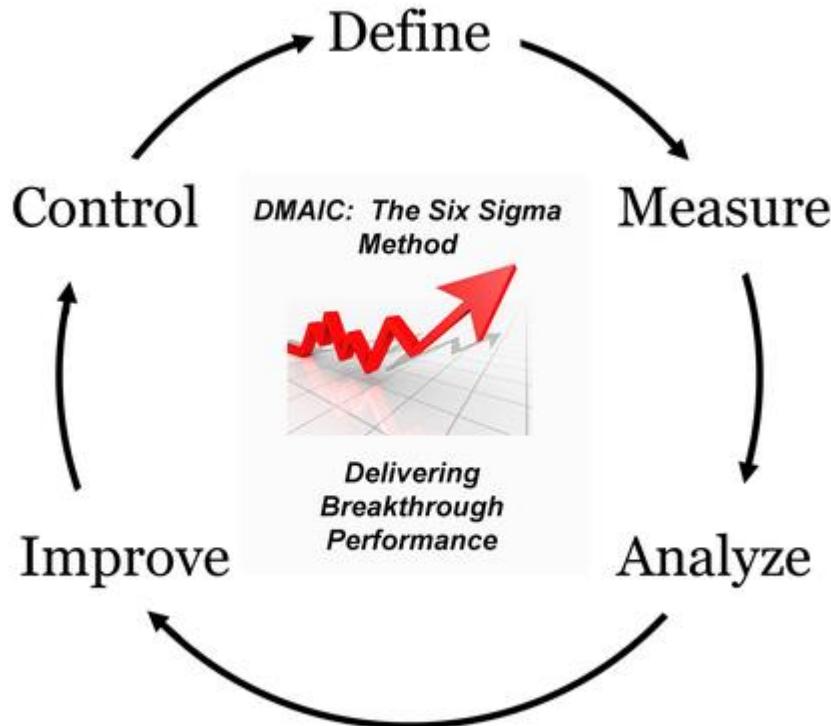
Software Process Improvement

- There are two basic frameworks that were developed by Motorola based upon the Shewart cycle (plan-do-check-act): DMAIC and DMADV
 - DMAIC: Define, Measure, Analyze, Improve, Control & is typically used to improve existing processes
 - DMADV: Define, Measure, Analyze, Design, Verify & is typically used for the development of new processes

We will look at DMAIC which is the most widely known of the two

Software Process Improvement

6-Sigma DMAIC



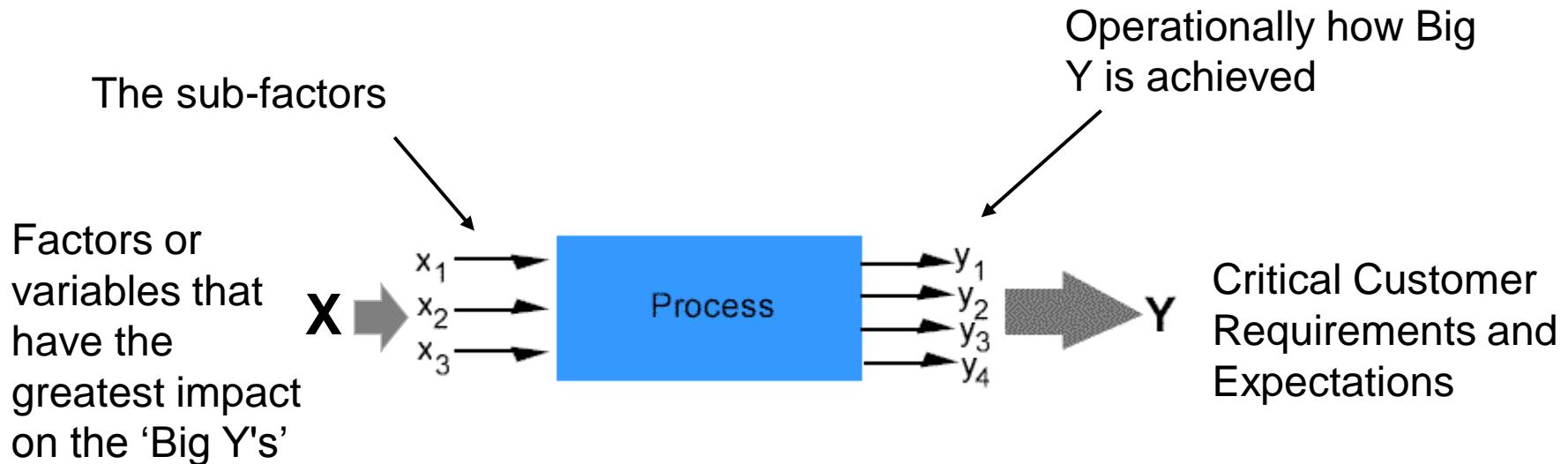
Software Process Improvement

6 Sigma: X's and Y's

- *Big Y's* == The most important business results and measures that are linked to critical customer requirements and expectations.
- *Little Y's* == Operational objectives that must be improved to achieve Big Y improvements. In a restaurant the Big Y might be to improve diner satisfaction. The Little y could be time to seat the customers, availability of menu choices etc.
- *Big X's* == Factors or variables that have the greatest impact on the 'Big Y's'
- Little X's == Sub-factors that influence a 'Big X'

Software Process Improvement

6 Sigma: X's and Y's

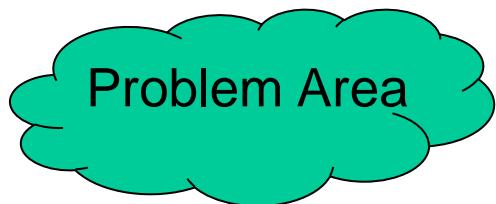


Software Process Improvement

6 Sigma: Define

- *Define* the problem, the voice of the customer, and the project goals.
 - Poorly performing areas are identified and prioritized through use of data
 - Gather & prioritize customer wants & needs (their voice)
 - Make a business case for improvement
 - Form teams & issue charter.
 - Map the process

Software Process Improvement



Business Case

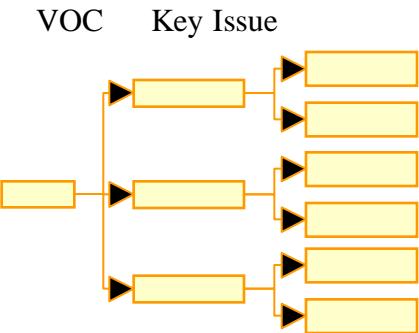
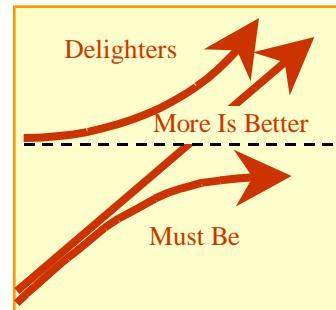


Project Charter & Teams

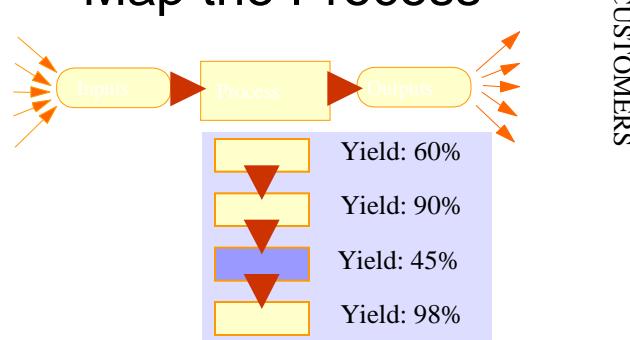
Problem Statement:	_____
Goal:	_____
Business Case:	_____
Scope:	_____
Cost Benefit Projection:	_____
Milestones:	_____



Voice of the Customer:
Prioritized customer's expectations,
preferences and aversions



Map the Process



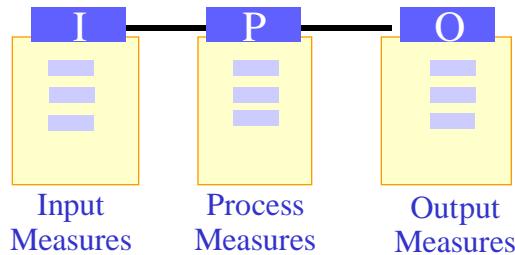
Software Process Improvement

6 Sigma: Measure

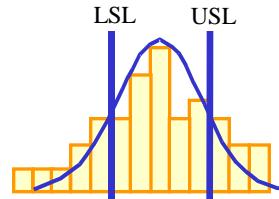
- *Measure* key aspects of the current process and collect relevant data
 - Identify the set of metrics that need to be collected to characterise the process & Prioritise them
 - Identify how the data will be collected & How it will be validated
 - Make use of quality control tools like Check sheets, Pareto charts, Cause & Effect (Ishikawa)diagrams, Histograms

Software Process Improvement

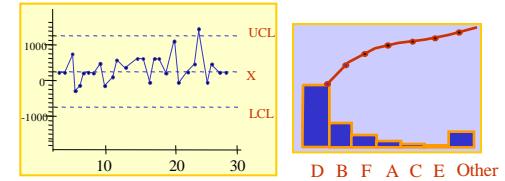
Identify the Metrics



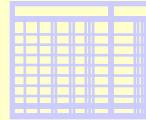
Identify Process Capability



Display Data



Prioritize the Metrics



Data Collection Plan

Data Collection Plan				
What questions do you want to answer?				
	Operational Definition and Procedures			
What	Measure type/ Data type	How measured	Related conditions	Sampling notes
				How/where
	How will you ensure consistency and stability?		What is your plan for starting data collection?	
			How will the data be displayed?	

Data Validation



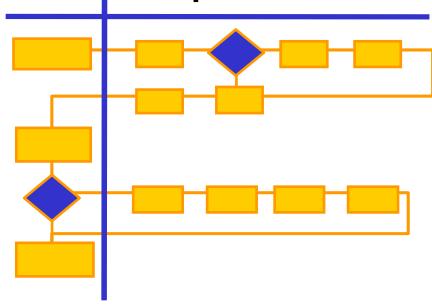
Software Process Improvement

6 Sigma: Analyse

- *Analyse the data to investigate and identify causes and effects*
 - When, where and why do defects occur i.e. understand exactly what is happening within a process and why defects are occurring
 - Look from both process and data perspectives
 - Use appropriate statistical tools
 - Characterise the defect
 - Go after root cause

Software Process Improvement

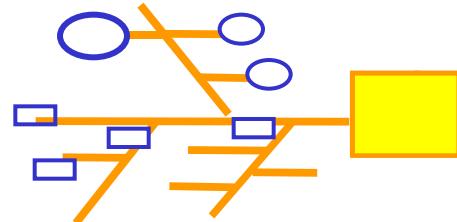
Analysis from a
Process
Perspective



Hypothesis-Testing
for statistical
significance



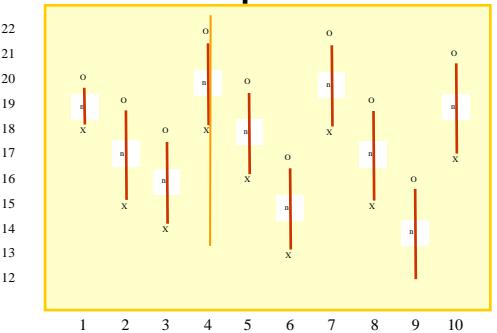
Cause & Effect



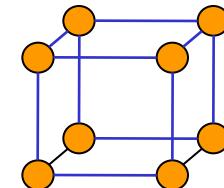
Regression Analysis



Analysis from a
Data
Perspective



Controlled Experiments



Software Process Improvement

6 Sigma: Improve

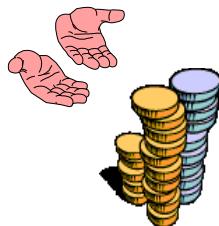
- *Improve or optimize the current process*
 - Generate solutions or mistake proof processes (poka yoke in LEAN)
 - Weigh up the costs & benefits
 - Select the best solution
 - Understand the risks
 - Pilot the new process & if successful -> plan for implementation

Software Process Improvement

Generate Solutions



Perform Cost-Benefit
Analysis



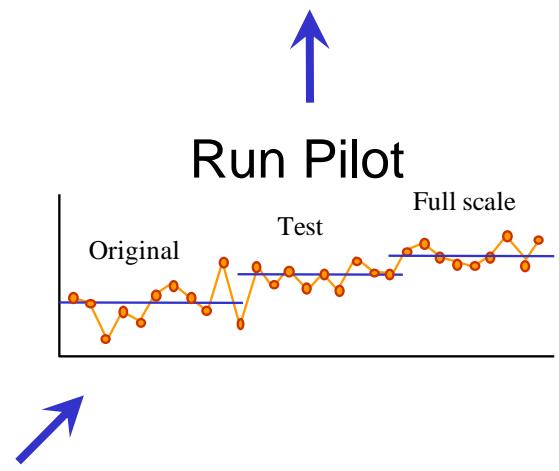
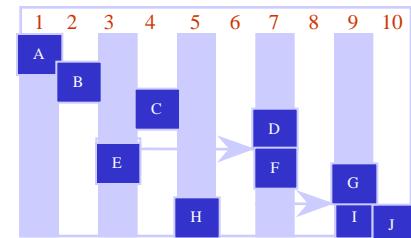
Select the Solution



Assess Risks



Plan
Implementation



Run Pilot

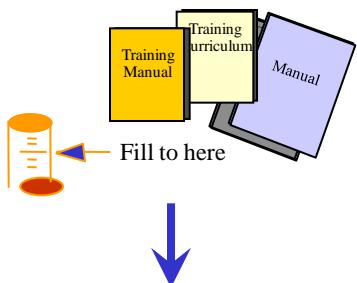
Software Process Improvement

6 Sigma: Control

- *Control* the new process to ensure that any deviations from target are corrected before they result in defects
 - Assign a process owner
 - Document the process
 - Use statistical process control (SPC) tools to control the process
 - Continuously monitor the process
 - Seek process change for further improvements
 - Monitor the results

Software Process Improvement

Document the process



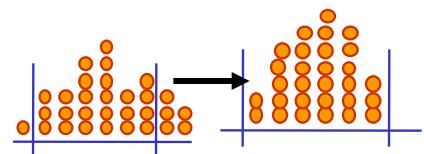
Assign Process Owner



Ongoing Monitoring

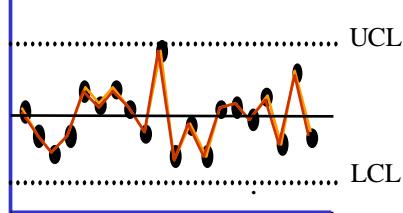
Product Name	Date of Release	Facility	Approved by	Signature
Process Name	Revision Date	Reason		
Process Code #				
Flowchart	Work Instructions	Code #	Control Check Points	Response to Abnormalities
Flowchart	Work Instructions	Code #	Change Control Method	Reinforced
Flowchart	Work Instructions	Code #	Review Period	Permanent
Flowchart	Work Instructions	Code #	Notes	Notes

Evaluate Results

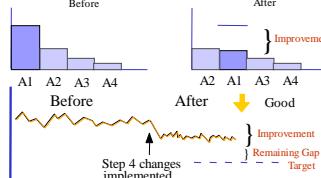


$s = 3.7$ $C_p = 1.4$ LSL $s = 2.7$ $C_p = 0.4$ USL

Use SPC techniques



Process Change



Software Process Improvement

6 Sigma: The Key Roles

- Project Champions
- Master Black Belts
- Black Belts
- Green Belts
- Yellow Belts
- White Belts

Software Process Improvement

6 Sigma: The Key Roles

- *Project Champions* are involved in:
 - Selecting projects
 - Identifying Black and Green Belt candidates
 - Setting improvement target
 - Providing resources
 - Reviewing the projects on regular basis
 - Removing any road blocks to the programs success.

Software Process Improvement

6 Sigma: The Key Roles

- *Master Black Belts* are involved in:
 - Technical leaders of Six Sigma;
 - Serve as instructors for Black & Green Belts;
 - Provide ongoing coaching and support to project teams to assure the appropriate application of statistics
 - Provide assistance to Project Champions;
 - Deploy Six Sigma program.

Software Process Improvement

6 Sigma: The Key Roles

- *Black Belts* are:
 - Backbone of Six Sigma deployment
 - Highly qualified
 - Lead teams
 - Attack chronic problems
 - Manage projects
 - “Drive” teams for solutions that work
 - Responsible for bottom line results.

Software Process Improvement

6 Sigma: The Key Roles

- *Green Belts:*
 - Provide team support to Black Belts
 - Assist in data collection, input
 - Analyze data using software
 - Prepare reports for management.

Software Process Improvement

6 Sigma: The Key Roles

- *Yellow Belts:*
 - Represent large percentage of work force
 - Trained with basic skills
 - Assist GB & BB on large projects
 - Assist in build and sustain Six Sigma culture

Software Process Improvement

6 Sigma: The Key Roles

- *White Belts:*
 - Foot-soldiers in a project work force
 - Basic 6-sigma overview training
 - Not regarded as a core competency for them
 - Work under direction

Software Process Improvement

6 Sigma: Criticism

- Lack of originality – “Nothing new” Juran
- Overselling by consultancies
- Arbitrary standards – other products/services may need higher levels than 6 sigma – for instance cardiac pacemakers, autopilot software.
- Stifling creativity

Software Process Improvement

6 Sigma: Exercise

- Go through the ‘Define’ Phase
- Identify the product you create/the service you provide
- Identify the customer and the voice of the customer
- Identify your needs (to satisfy the customer)
- Define the process
- Suggest how the process may be mistake-proofed

Software Process Improvement

6 Sigma: Summary

- *Summary:*
- Defined Focus – Reduce variability
- Defined Statistical Basis – Sigma spread
- Defined Process – DMAIC,DMADV etc.
- Defined Roles – Belts...

IS4415

Software Metrics & Statistical Techniques Part 1

- Part 1
 - Metrics?
 - Causal Analysis
 - Some Causal Analysis Techniques
- Part 2
 - Statistical Techniques
 - Histograms & Data Distributions
 - Control Charts
 - Process Capability

Software Metrics & Statistical Techniques (1)

- Metric -To measure [Gk. = *metron*]
- Software Metrics - ways of quantifying important indices that will give an insight into the efficiency and effectiveness of the software process, products, or project performance.

Software Metrics & Statistical Techniques (1)

- Three principle types of software metrics:
- *Software Process Metrics*
 - Measurements that will give information about the effectiveness of software process
- *Software Product Metrics*
 - Measurements that will give information about the quality of the software being produced
- *Software Project Metrics*
 - Measurements that will give information about project performance

Software Metrics & Statistical Techniques (1)

- *Software Process Metrics*

- Measurements that will give information about the effectiveness of software productivity e.g.:
 - Phase cycle times - time spent, costs incurred, etc
 - Defect arrival rates – from test phase, from customers, etc
 - Defects by phase – how many are escaping detection in-phase
 - Test execution rate – delays due to excessive faults found, etc.

Software Metrics & Statistical Techniques (1)

- *Software Product Metrics*

- Measurements that will give information about the quality of the software product being developed. E.g.:
 - Customer found defects – type of defect, severity, etc.
 - Customer satisfaction indices – why high?, why low?, etc.
 - Customer calls to helpdesk – usability issues, navigation, etc.
 - Total downtime of customer system – impacts, costs, etc.
- Barry Boehm referred to these metrics as the ‘ilities’ – reliability, usability, availability, maintainability, portability and so on...

Software Metrics & Statistical Techniques (1)

- *Software Project Metrics*

- Measurements that will give information about the performance of the current project. E.g.:
 - Requirements changes – schedule impact, scope impact, etc
 - Tasks scheduled v's tasks completed – estimate accuracy, etc.
 - Budget under-run/over-run – remedial actions, scope, etc.
 - Training hours planned v's taken – budget impact, quality impact

Software Metrics & Statistical Techniques (1)

Premise

“When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science.”

- Lord Kelvin

But producing software is complex – lots of variables, lots of interaction = lots of causes of process and product variation!

Software Metrics & Statistical Techniques (1)

The BIG Question is:

How do we decide what to measure?

Answer: We must conduct a thorough causal analysis to identify what are the KEY

- Process
- Product
- Project

Variables that will give us the essential information we need to manage the business of developing software products.

A step that is often done poorly or overlooked entirely!

Software Metrics & Statistical Techniques (1)

- Causal analysis is a major part of the process - but one that is often omitted or decided arbitrarily.
- Often ignored as organisations rush to achieve Level 4 CMM/CMMI certification and start collecting data on everything in sight.
- Net result is organisations spending vast sums of money and energy measuring and controlling process, product and project variables that matter little to the business.
- A pity as some causal analysis techniques are very simple!

Software Metrics & Statistical Techniques (1)

What are some of the techniques we can use for causal analysis?

SOME CAUSAL ANALYSIS TECHNIQUES

- Brainstorming
- Goal-Question-Metric (GQM)
- Pareto Charts
- Ishikawa (Fishbone) Diagrams

Software Metrics & Statistical Techniques (1)

Brainstorming:

- Typically a free format open or structured discussion of a problem conducted by a team
 - Try to generate ideas on a topic
 - Free of criticism
 - Free of judgement
-
- But is not just “shooting the breeze”..... there is a process for conducting good and effective brainstorming sessions....

Software Metrics & Statistical Techniques (1)

Brainstorming Process:

1. State and agree the problem

“We are all agreed we are delivering projects late” – Sets the agenda

2. Record the problem (whiteboards, blackboards, flipcharts) – keeps the meeting on topic & provides a focus

3. Team members supply ideas – never criticised – keep an open mind – too many good ideas get lost because people don't speak up!

4. Record each idea – don't trust to memory – have someone act as scribe

5. When all are recorded review list for duplicates & clarify – there will be overlaps and improvements of ideas – refine what's been said

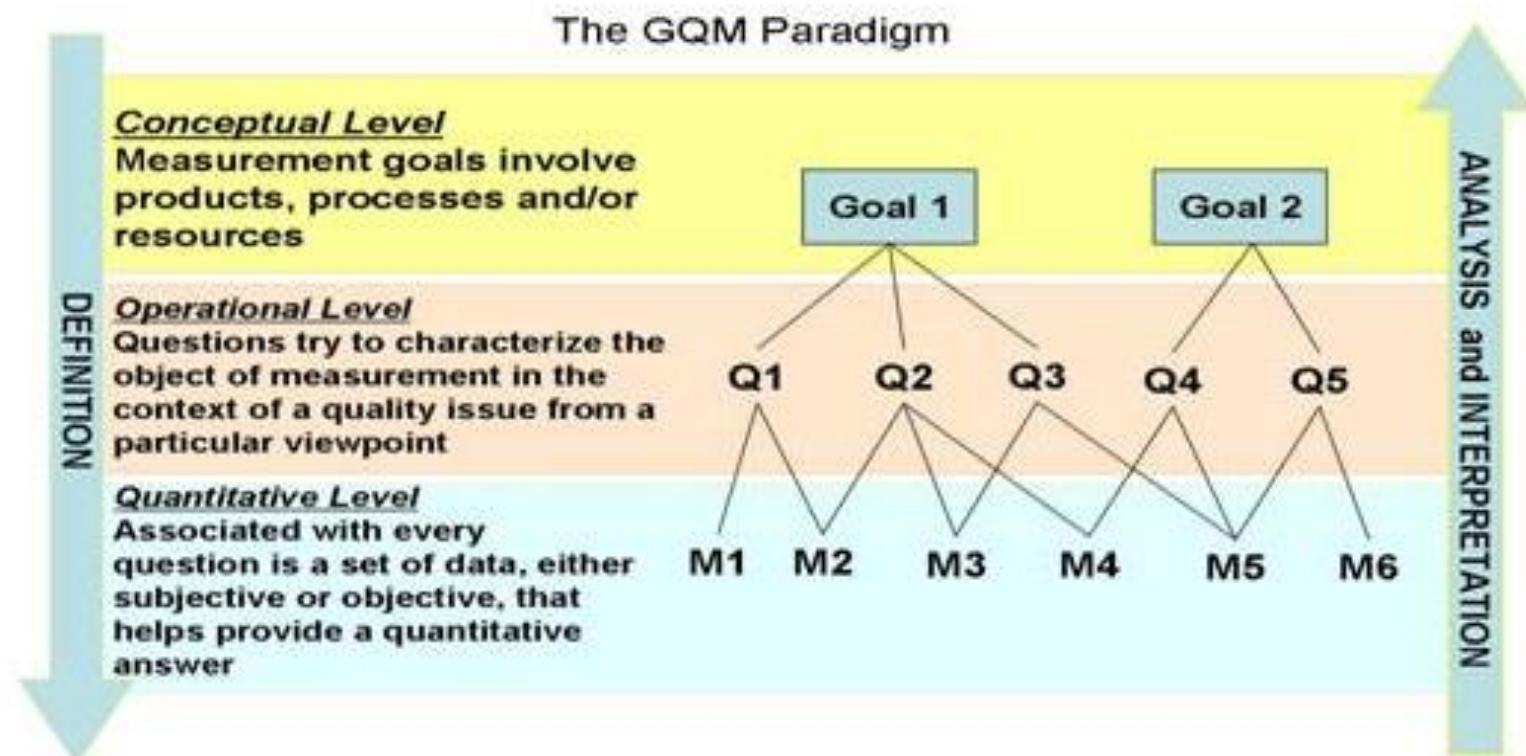
6. Record set of possible causes – keep an open mind – there could be more than one cause for a problem...

Software Metrics & Statistical Techniques (1)

- A useful technique that is used to develop quality indices is the **Goal-Question-Metric (GQM) method** Basili, 1984
- Began at NASA / Goddard Space Flight Centre as a way to collect valid software engineering data – became GQM
- Top-Down
 - What are the goals the business wants to accomplish? – what matters to it, what is strategic, etc.
 - What questions are needed to define/refine those goals? – how do we make these conceptual goals real, and tangible, etc.
 - What metrics are required to answer the questions and determine if a goal has been achieved – base decision making on factual data

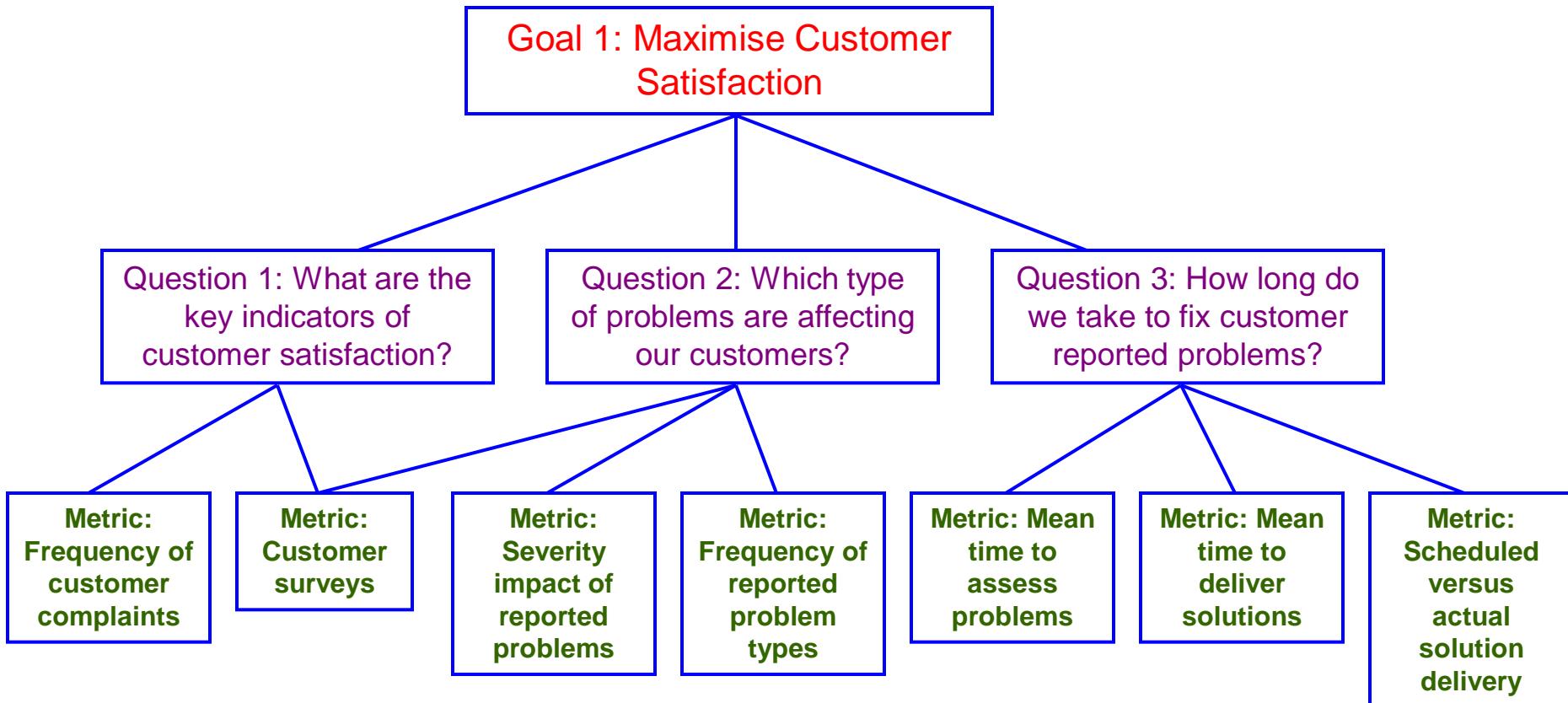
Software Metrics & Statistical Techniques (1)

- The GQM model – three levels...



Software Metrics & Statistical Techniques (1)

- For example:



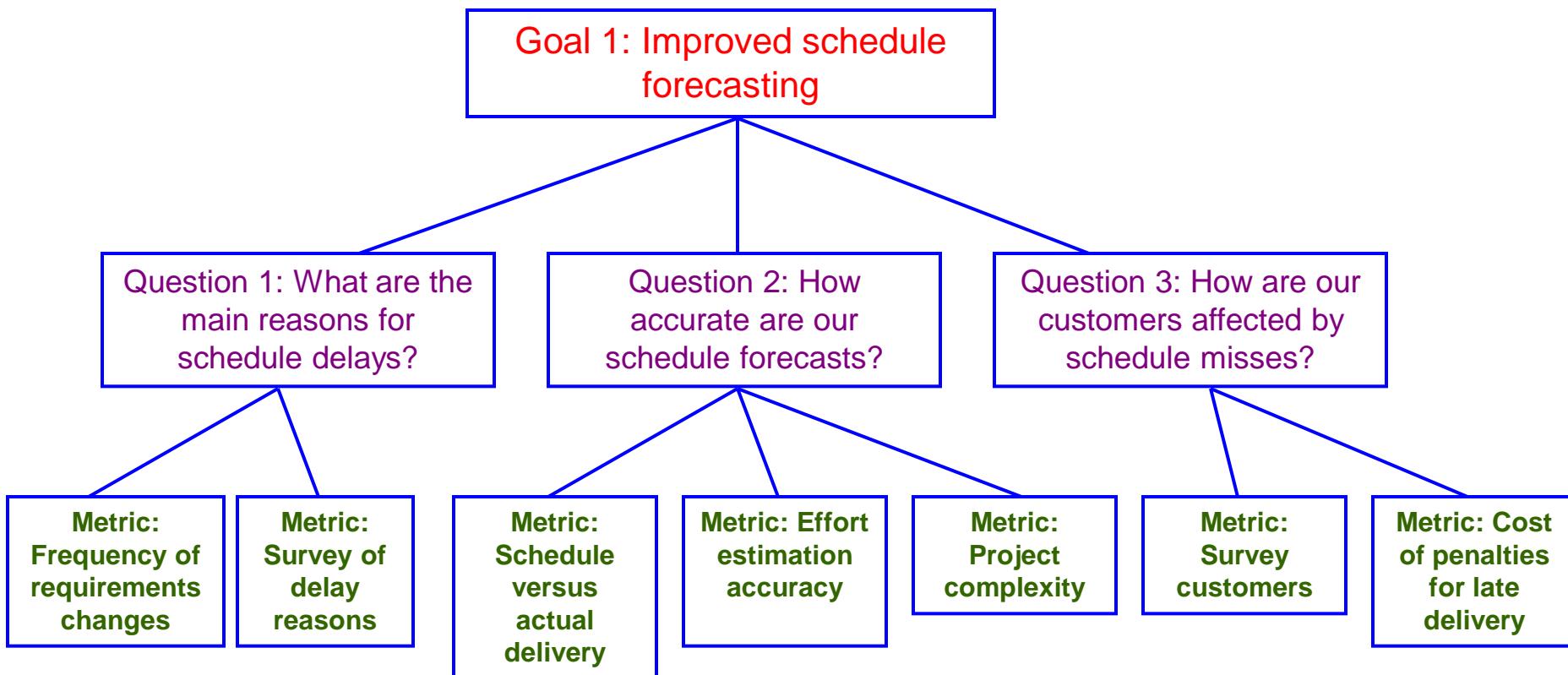
Software Metrics & Statistical Techniques (1)

- Exercise

Goal 1: Improved schedule
forecasting

Software Metrics & Statistical Techniques (1)

- Exercise – Suggested Questions and Metrics



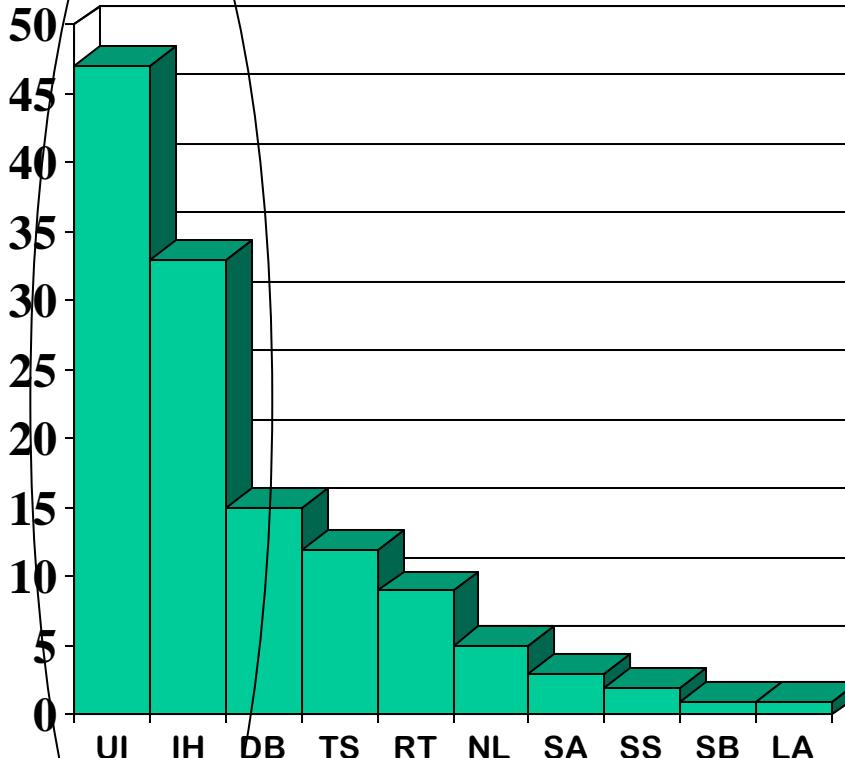
Software Metrics & Statistical Techniques (1)

Pareto Charts:

- Focus attention on the problems that offer the greatest potential for improvement by showing their relative frequency or size in a descending bar graph
- Helps organisations to focus on those causes that will have the greatest impact if solved
- Based on the proven 80/20 rule – 20% of the sources cause 80% of the problems
- Displays relative importance of problems simply
- Progress can be measured longitudinally (over time)

Software Metrics & Statistical Techniques (1)

Pareto Charts Example:



■ System faults

UI = User Interface
IH = Interrupt Handling
DB = Database Faults
TS = Time Sequencing
RT = Response Timing
NL = Network Loading
SA = Security Access
SS = Signal Strength
SB = System Backup
LA = Line Attenuation

20% of sources causing
80% of problems

Software Metrics & Statistical Techniques (1)

Pareto Chart Process:

1. **Decide which problem you need to know more about** – business impact, strategic importance, etc.
2. **Choose the problem categories that will be monitored** – use brainstorming, examine historical data, etc.
3. **Choose the most meaningful unit of measurement** – frequency of occurrence, unit cost, etc.
4. **Choose the time period for data collection** – daily, weekly, etc.
5. **Gather the data** – in “real-time”, review historical sources, etc.
6. **Identify relative frequency for each problem category** – highest range of results, etc.
7. **List the problem categories** (starting with the largest) on the horizontal axis and the frequency on the vertical axis
8. **Interpret the results** - what is the distribution telling you?

Software Metrics & Statistical Techniques (1)

Pareto Charts Example Scenario:

1. Getting an excessive number of calls to a customer helpdesk
2. Brainstorm to identify why people are calling the helpdesk - what are typical problem types that users report to the helpdesk
3. Do we have historical data of the problem types that were recorded?
4. Most important to measure the frequency of the calls
5. We decide to review the helpdesk calls for 10 weeks (May 22 – August 4)
6. Gather the historic data
7. What's it telling us about the software product?

Software Metrics & Statistical Techniques (1)

Pareto Charts Exercise

1. Compare relative frequency for each problem category

Problem Category	Frequency
Bad Configuration	3
Boot Problems	68
File Problems	8
Connectivity Issues	20
Print Problems	16
System Configuration	24
Monitor Faults	11
LAN Problems	6
WAN Connection	16
Server Problems	19
Others	15
Total	206

2. List the problem categories (starting with the largest) on the horizontal axis and the frequency on the vertical axis
3. Interpret the results

Software Metrics & Statistical Techniques (1)

Ishikawa Diagrams:

- Allow a team to identify, explore and graphically display, in increasing detail all of the possible causes related to a problem or condition to discover its root cause.
- Enables the team to focus on the content of the problem rather than on the history of the problem or personal interests.
- Builds support and consensus for the resulting solutions
- Focuses on causes, not symptoms

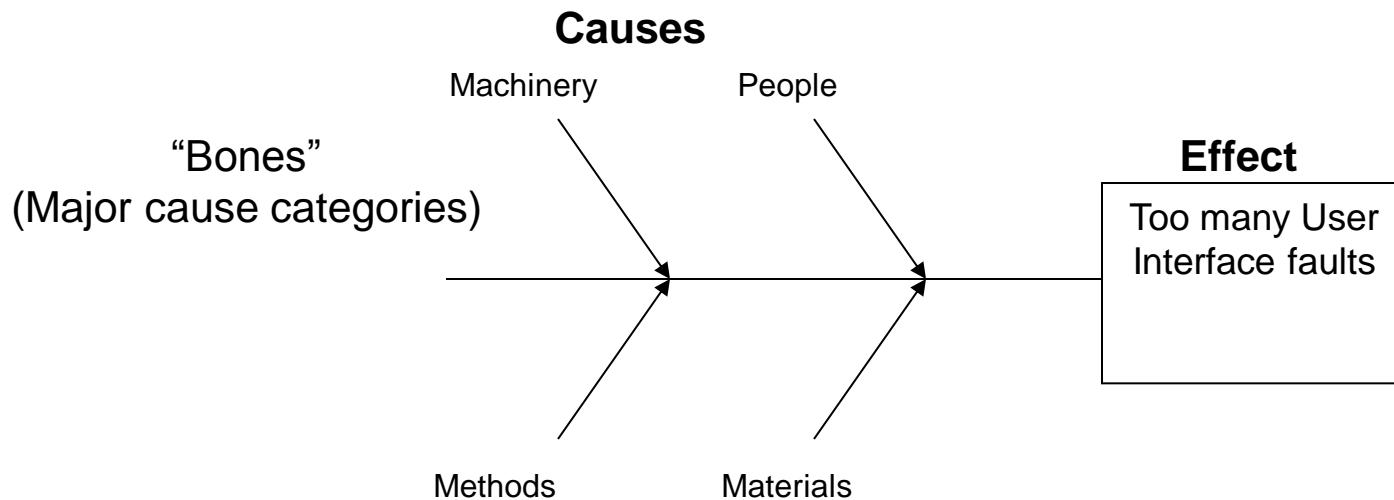
Software Metrics & Statistical Techniques (1)

Ishikawa Diagrams:

Two different types: Dispersion Analysis & Process Classification

Dispersion Analysis Type

Constructed by placing individual causes within a “Major” cause category and then asking ‘why does this cause happen?’ This question is repeated for the next level of detail until the team runs out of causes.



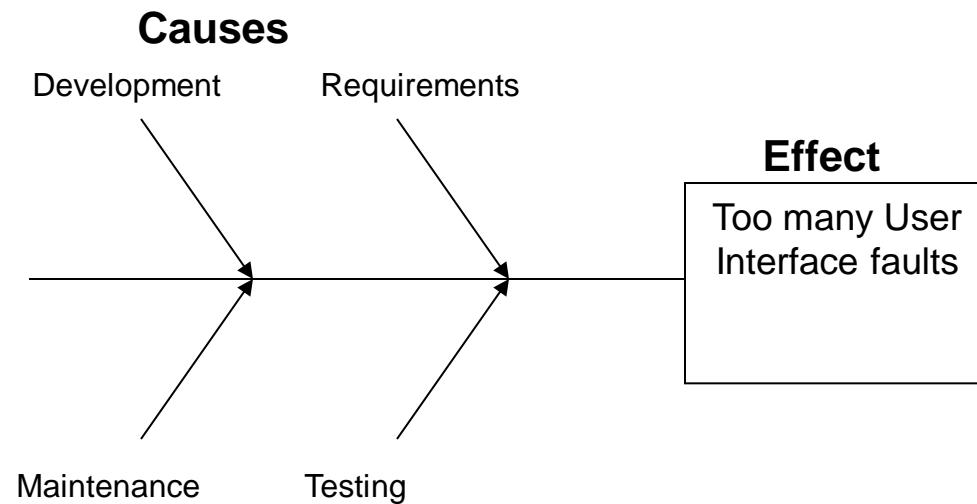
Software Metrics & Statistical Techniques (1)

Ishikawa Diagrams:

Two different types: Dispersion Analysis & Process Classification

Process Classification Type

Constructed by placing the major steps in the process in place of major cause categories. The questioning then follows same as for the Dispersion Analysis Type above.

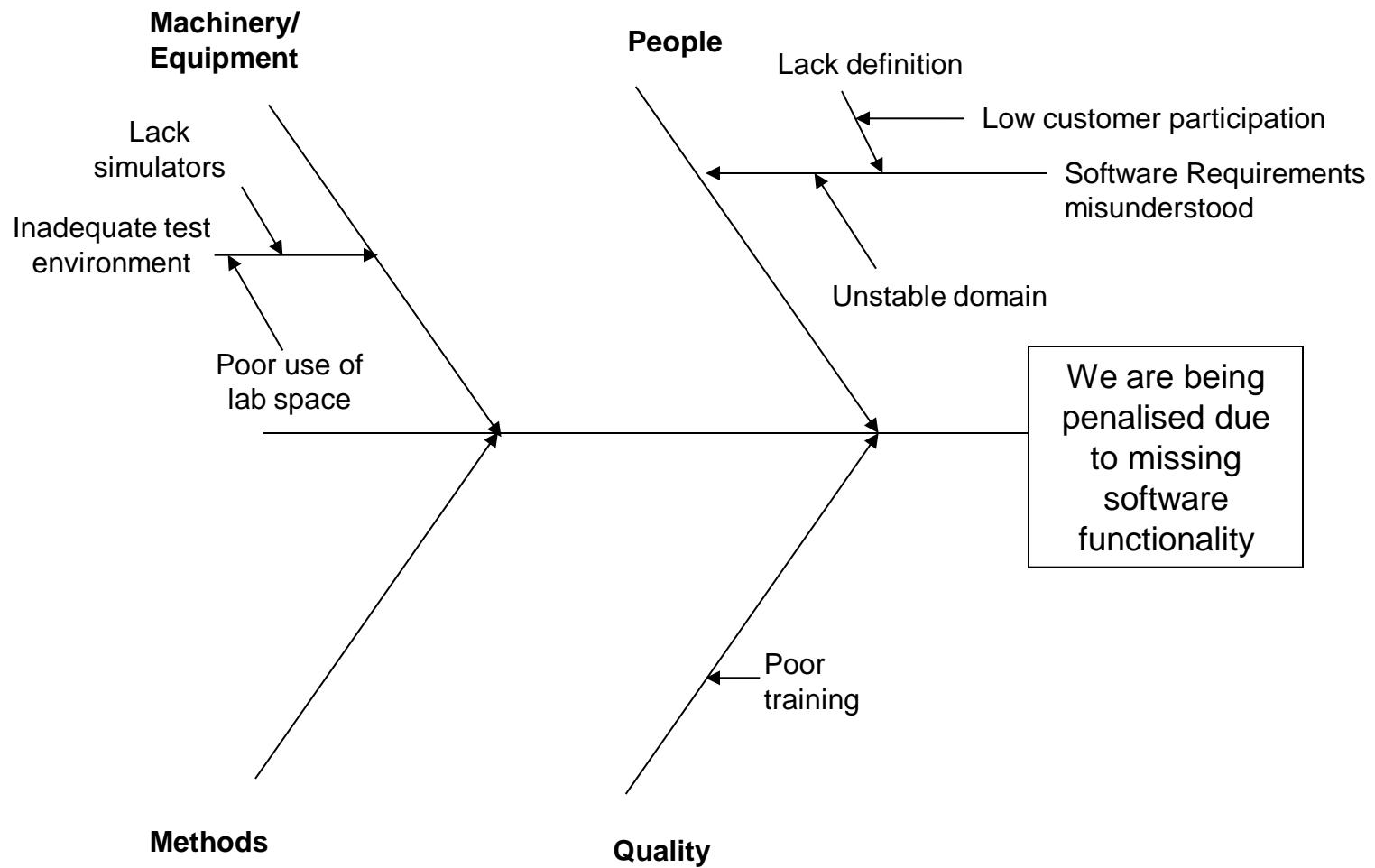


Software Metrics & Statistical Techniques (1)

Ishikawa Diagrams Process:

1. **Select the most appropriate format type**
2. **Generate the causes needed for entry under the major cause categories – brainstorm, use recorded data, etc.**
3. **Construct the diagram**
 - Record the problem statement into the effect box and ensure everyone agrees with it
 - Draw in the major cause categories
 - Place the brainstormed or database causes in the appropriate category
4. **For each “deeper”cause continue to push for deeper understanding until cause is obvious**
5. **Interpret or test for root causes**
 - Look for causes that appear repeatedly within or across major cause categories
 - Select most likely cause
6. **Gather statistical data on the root causes - determine frequency**

Software Metrics & Statistical Techniques (1)



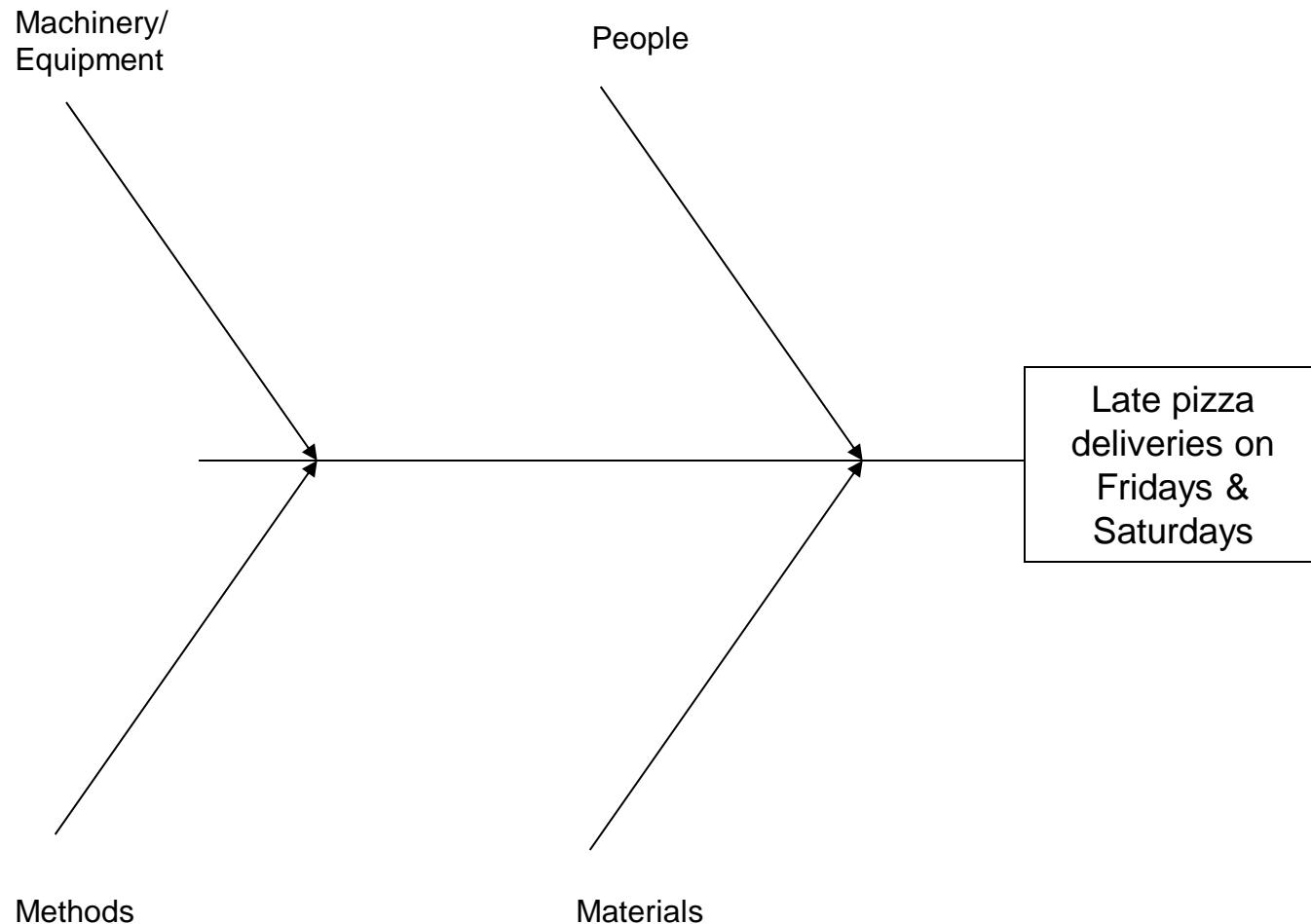
Software Metrics & Statistical Techniques (1)

Exercise:

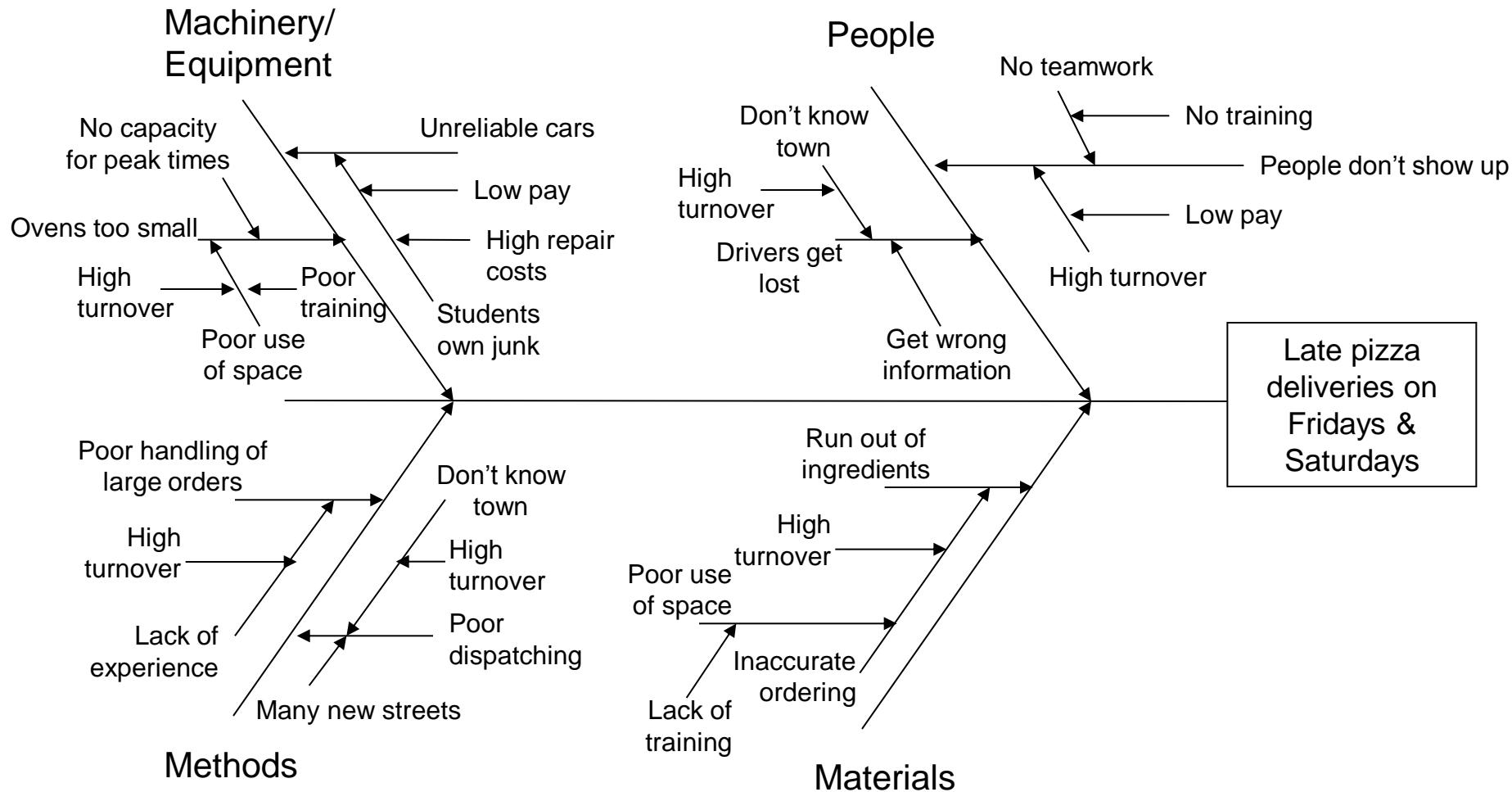
- I have a moderately successful pizza business – I say moderately because at peak times – Friday and Saturday nights – my deliveries start to get all messed up. Really big orders can come in, can really put the kitchen under pressure.
- At weekends I employ students to make the deliveries they have to use their own transport to do this. Most of them are from out of town and are available ‘most’ weekends, but they don’t know each other so I don’t have a single point of contact for them.
- My pizzas are really good and I need to remember to have extra ingredients in for the weekends – and find somewhere to put it!
- My problem is we end up delivering late and now I’m starting to lose customers....

Software Metrics & Statistical Techniques (1)

Exercise:



Late Pizza Delivery Exercise



- Software Metrics & Statistical Techniques Part 2

Software Metrics & Statistical Techniques (2)

- Part 1
 - Metrics?
 - Causal Analysis
 - Some Causal Analysis Techniques
- Part 2
 - Statistical Techniques
 - Histograms & Data Distributions
 - Control Charts
 - Process Capability

Software Metrics & Statistical Techniques (2)

- Software metrics & statistical techniques:
 - Fundamentally about better decision making
 - Previously examined how to identify what should be measured:-
 - Brainstorming
 - Goal-Question-Metric technique
 - Pareto Charts
 - Ishikawa Diagrams
 - Now will examine some basic but useful methods that are commonly used to inform decision making

Software Metrics & Statistical Techniques (2)

- Three principle types of software metrics:
 - Software Process Metrics
 - Software Product Metrics
 - Software Project Metrics

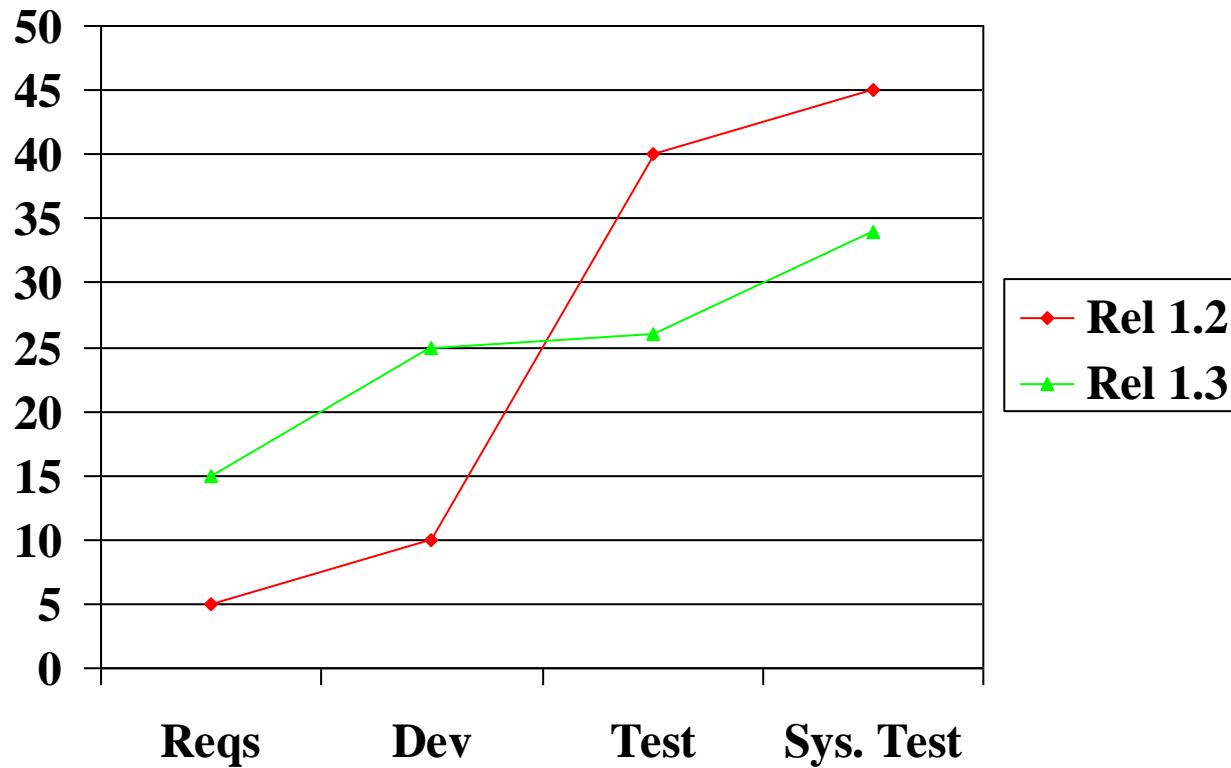
Software Metrics & Statistical Techniques (2)

- Three principle types of software metrics:
 - Software Process Metrics
 - Measurements that will give information about the effectiveness of the process in terms of **software productivity**.
- For instance
- Phase cycle times
 - Defect arrival rates
 - Defects by phase
 - Test execution rate

Software Metrics & Statistical Techniques (2)

- Process Metrics

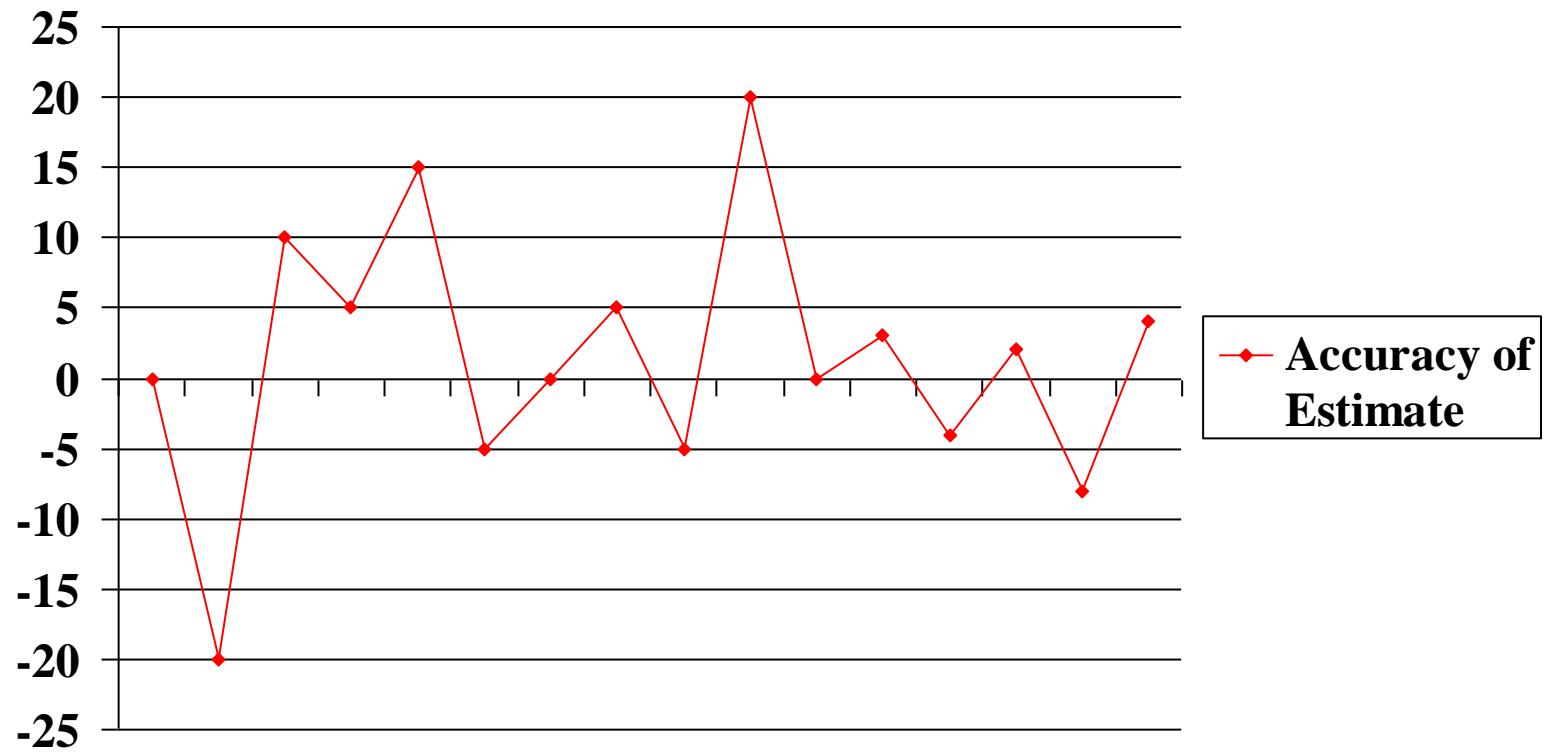
- Example 1: Cumulative # of major defects found per phase



Software Metrics & Statistical Techniques (2)

- Process Metrics

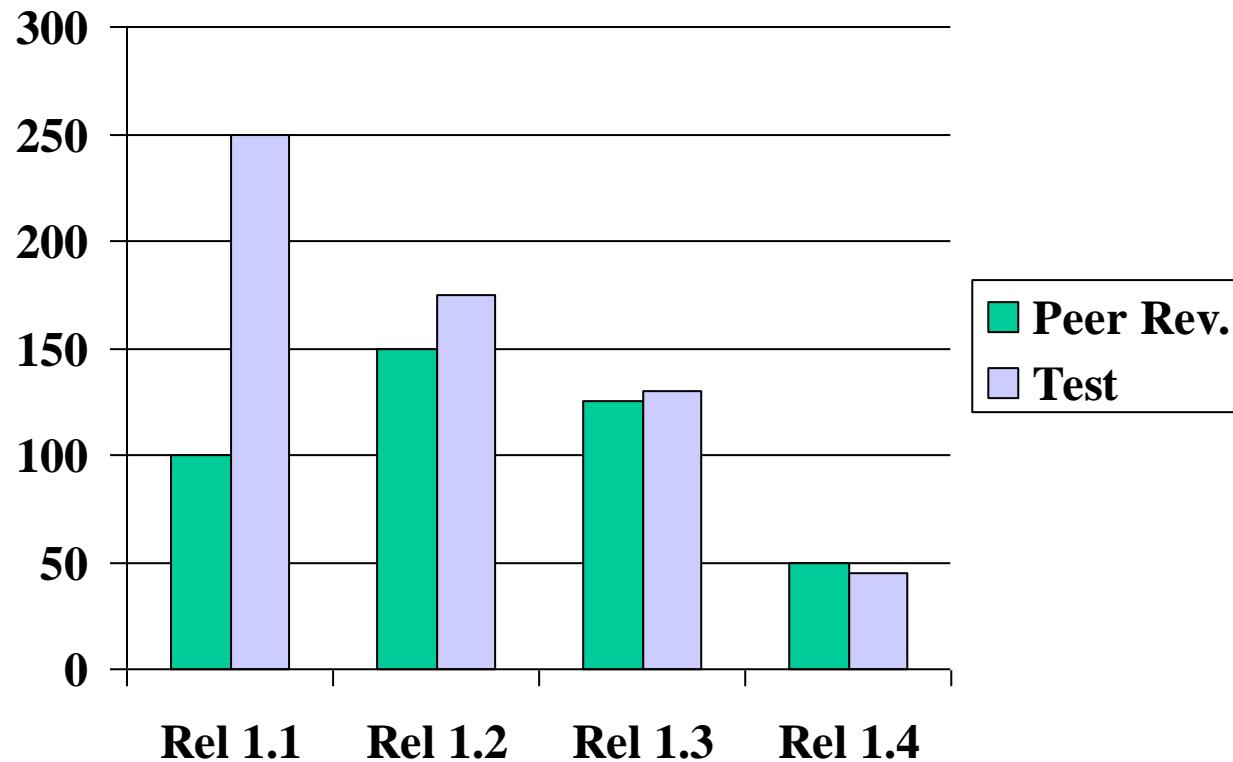
- Example 2: Estimation Accuracy: % over or under v's estimated



Software Metrics & Statistical Techniques (2)

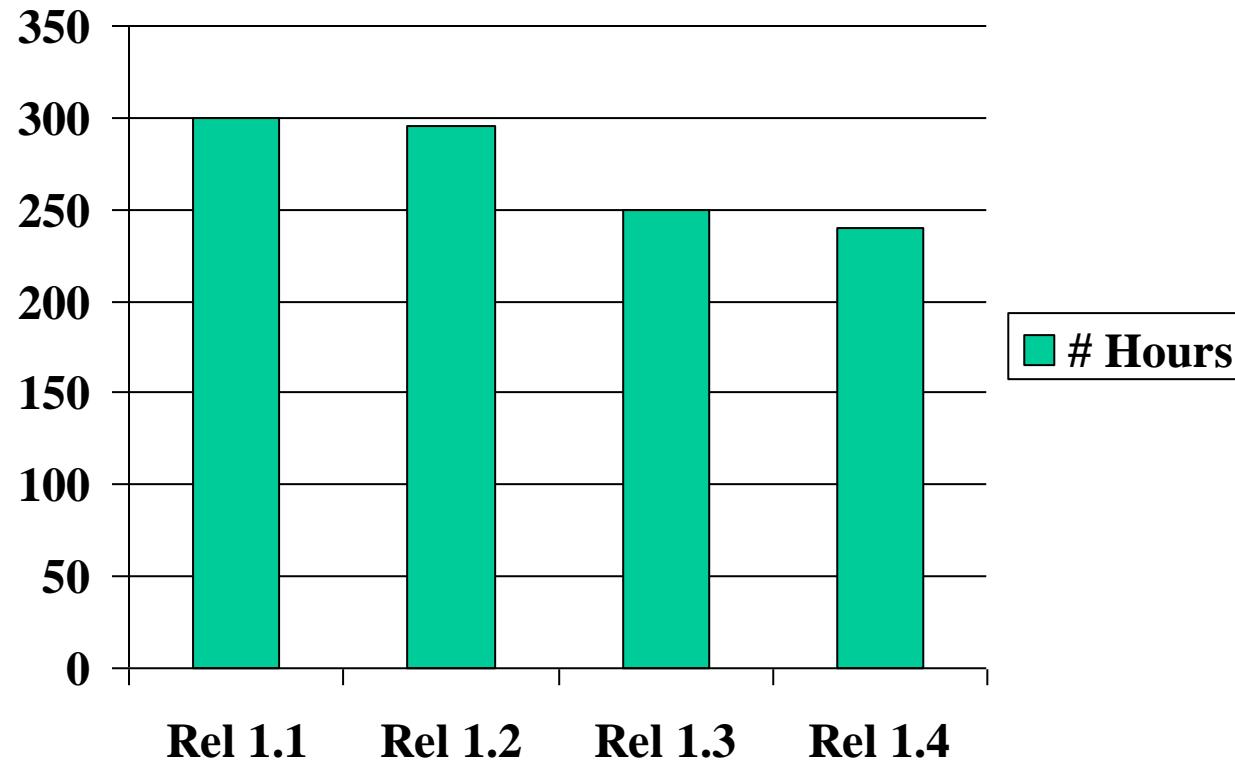
- Process Metrics

- Example 3: Defect detection - Faults found by peer review v's faults found in test



Software Metrics & Statistical Techniques (2)

- Process Metrics
 - Example 4: Test cycle time per release (normalized for release size)

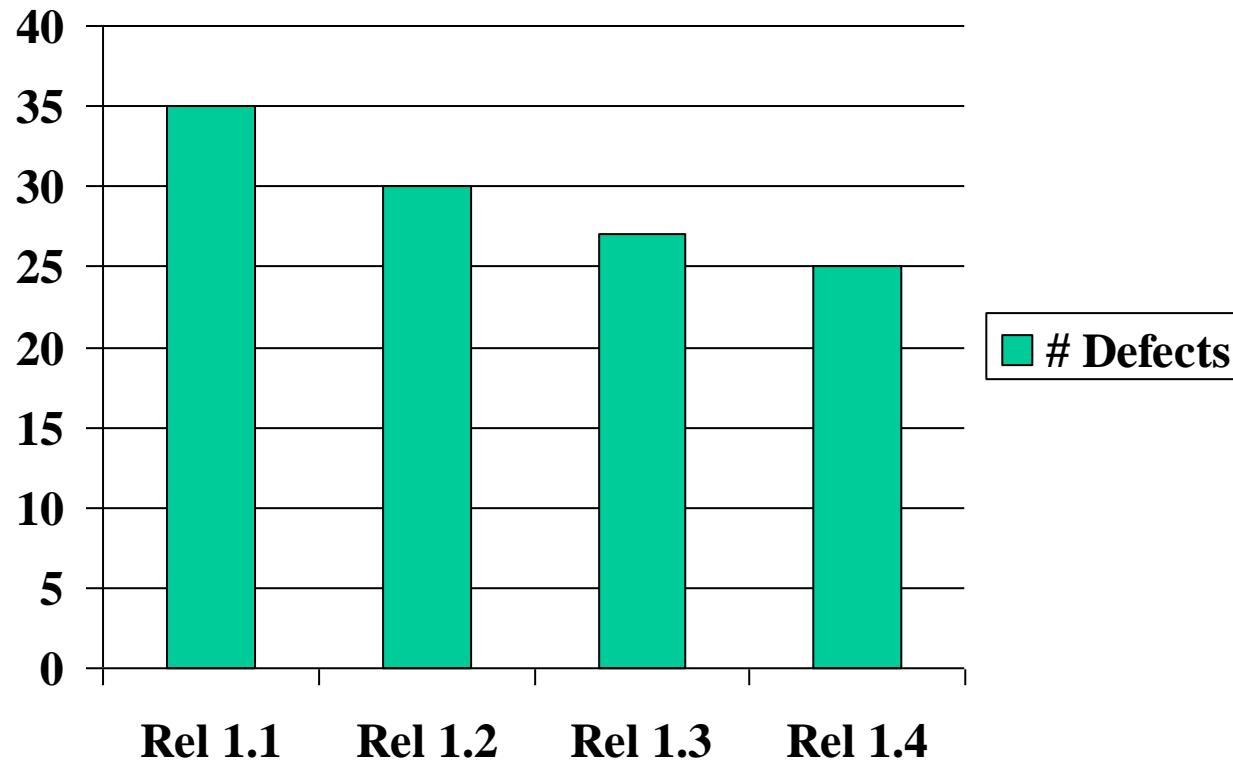


Software Metrics & Statistical Techniques (2)

- Three principle types of software metrics (contd):
- Software Product Metrics
 - Measurements that will give information about the **quality of the software product** being developed. For instance:
 - Customer found defects
 - Customer satisfaction indices
 - Customer calls to helpdesk
 - Total downtime of customer system
 - Barry Boehm referred to these metrics as the ‘ilities’ – reliability, usability, availability, maintainability, portability and so on...

Software Metrics & Statistical Techniques (2)

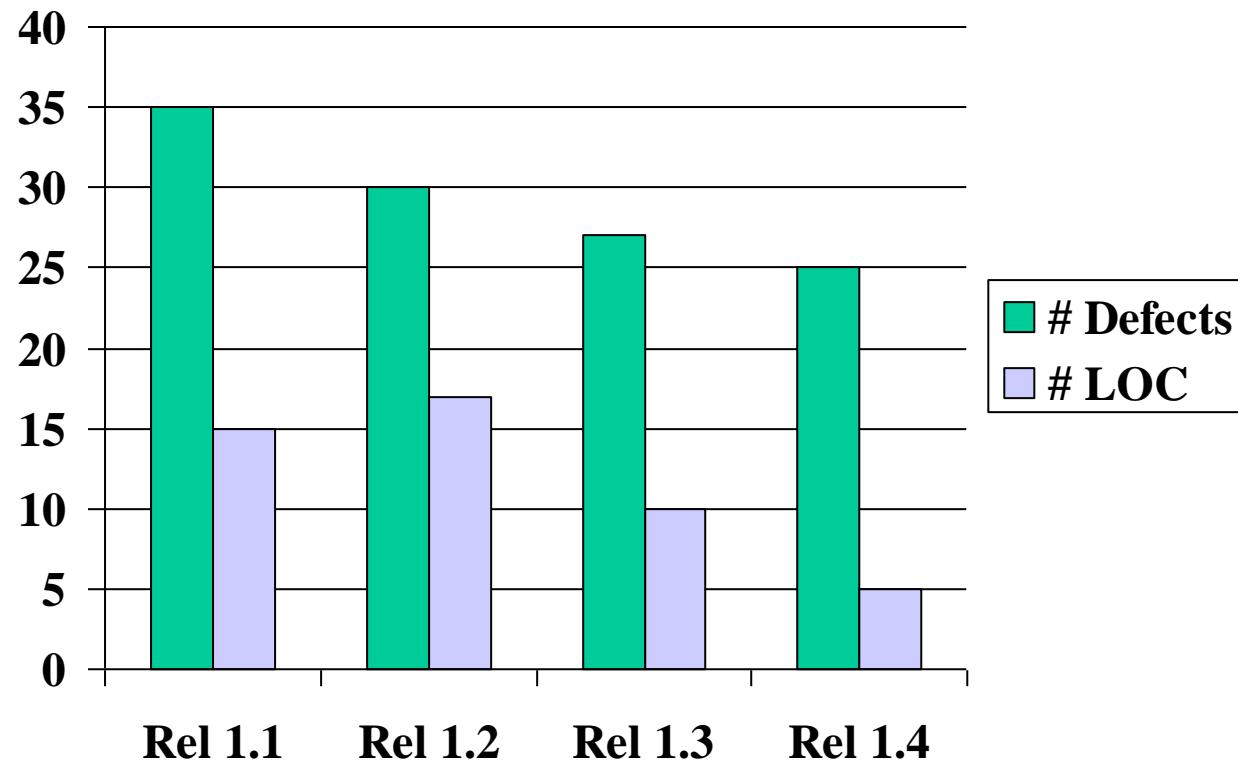
- Product Metrics
 - Example 1: Customer reported defects/release



Software Metrics & Statistical Techniques (2)

- Product Metrics

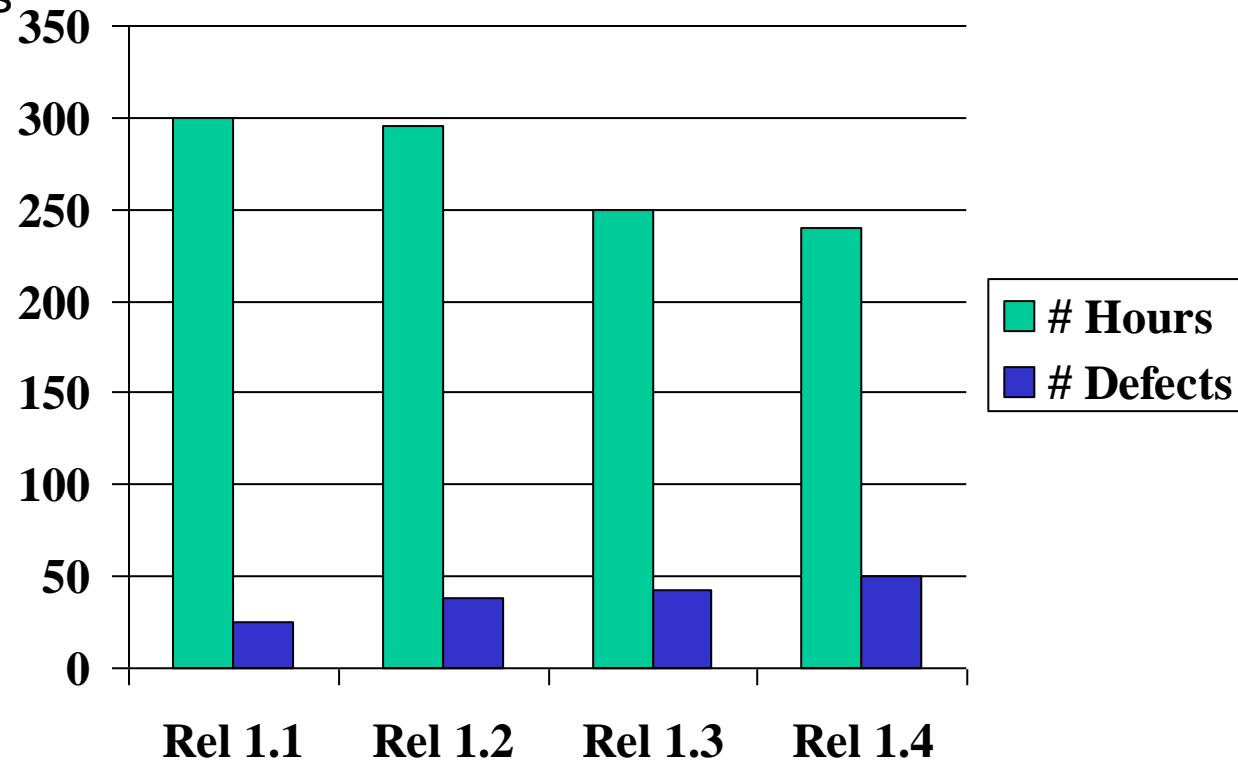
- Example 2: Customer reported defects/release v's Lines of code (k)



Software Metrics & Statistical Techniques (2)

- Product Metrics

- Example 3: Average cycle time per release v's Customer found defects



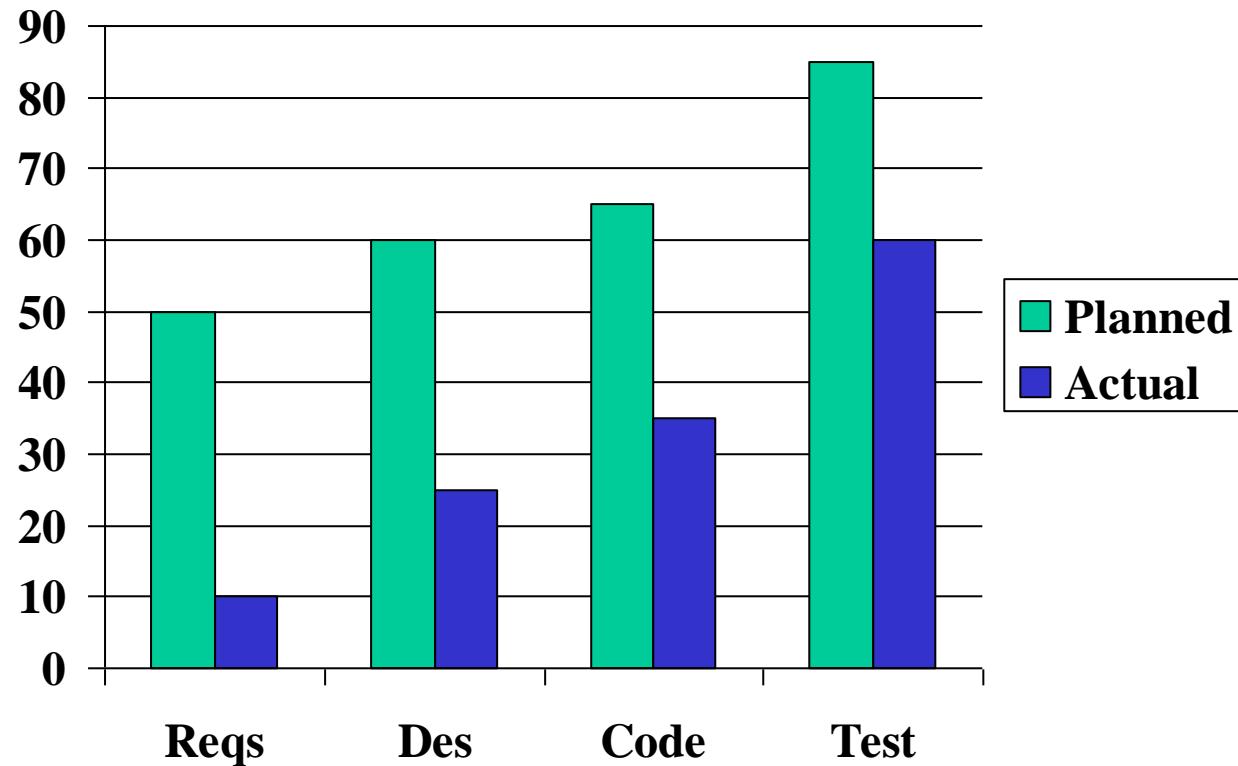
Software Metrics & Statistical Techniques (2)

- Three principle types of software metrics (contd):
- Software Project Metrics
 - Measurements that will give information about the performance of the current project. For instance:
 - Number of requirements changes
 - Tasks scheduled v's tasks completed
 - Budget under-run/over-run
 - Training hours planned v's taken

Software Metrics & Statistical Techniques (2)

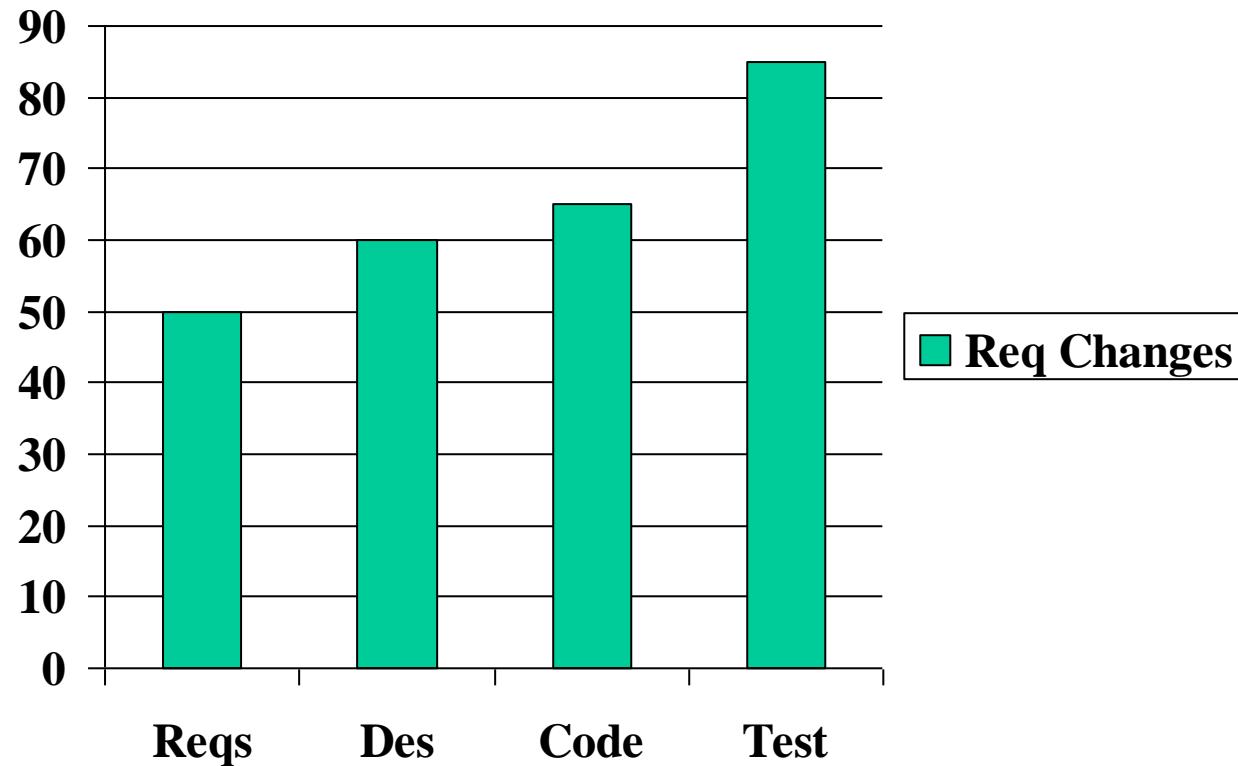
- Project Metrics

- Example 1: Planned v's actual budget spend

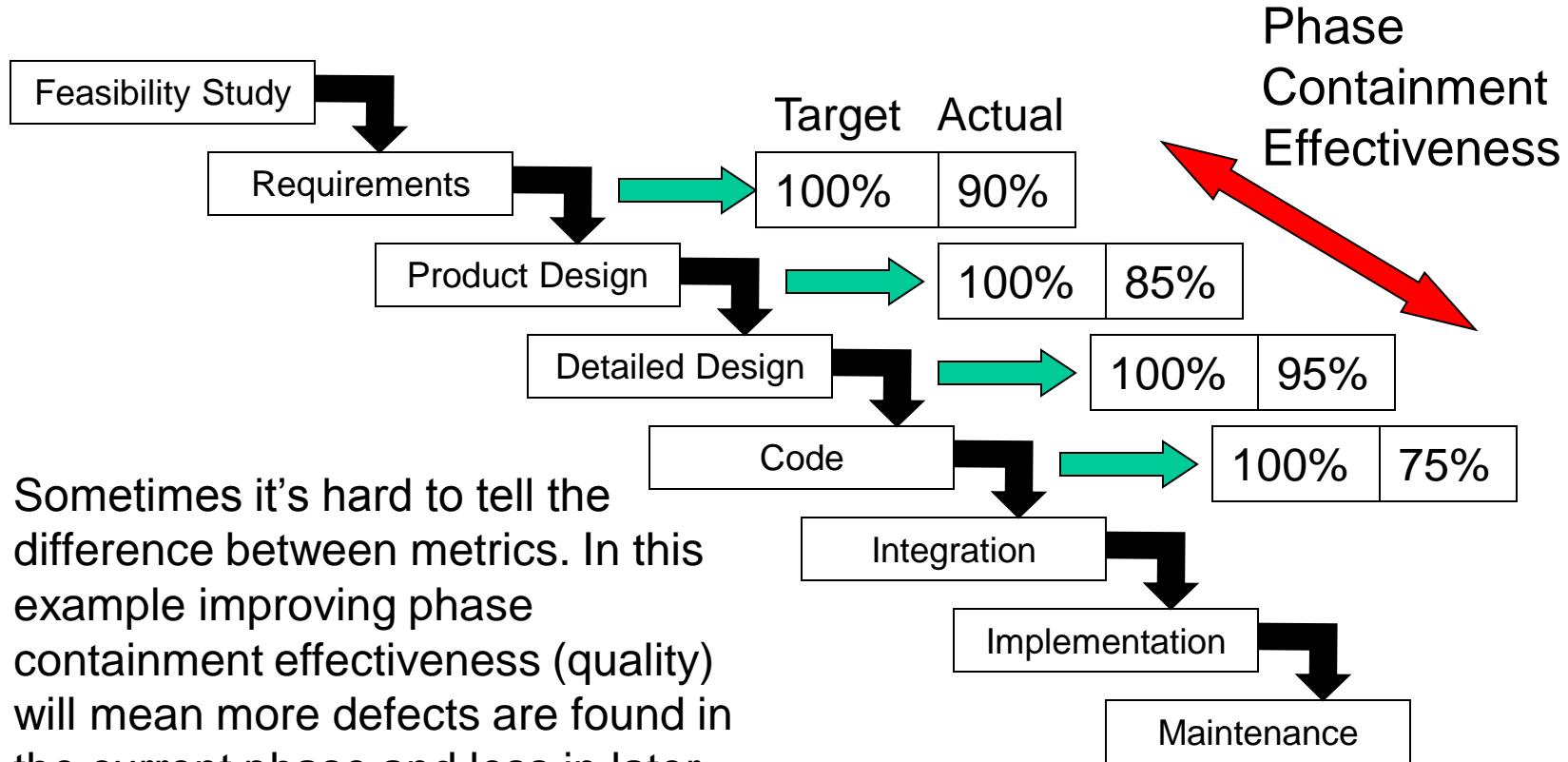


Software Metrics & Statistical Techniques (2)

- Project Metrics
 - Example 2: Requirements changes by phase



Software Metrics & Statistical Techniques (2)



Sometimes it's hard to tell the difference between metrics. In this example improving phase containment effectiveness (quality) will mean more defects are found in the current phase and less in later phases. So time spent in test phase will be less (productivity).

Software Metrics & Statistical Techniques (2)

- Part 1
 - Metrics?
 - Causal Analysis
 - Some Causal Analysis Techniques
- Part 2
 - Statistical Techniques
 - Histograms & Data Distributions
 - Control Charts
 - Process Capability

Software Metrics & Statistical Techniques (2)

- Histograms

- ❖ Display large amounts of data that are difficult to interpret in tabular form.
- ❖ Shows relative frequency of data values.
- ❖ Reveals the shape and the distribution of the data.
- ❖ Gives insight into future process performance.
- ❖ Indicates a change in the process.
- ❖ Helps in deciding if the process is capable of meeting customer requirements.

Software Metrics & Statistical Techniques (2)

- Developing Histograms

❖ Using data analysis techniques identify a process variable and then collect 50 – 100 data points over a sample time period or from historical data.

Problem Report Assessment Time

117	65	29	77	61	94	67	108	43	118
133	83	138	87	78	88	114	136	119	67
127	107	82	63	89	72	65	60	107	45
1	94	98	95	47	77	75	98	91	58
10	76	67	133	76	91	32	58	66	106
24	52	94	121	82	44	147	61	102	44
112	48	90	96	57	111	114	78	92	82
107	80	30	108	122	100	52	76	71	50
18	42	62	53	107	53	42	74	33	61
22	136	76	94	91	85	102	85	128	93

Software Metrics & Statistical Techniques (2)

- Developing Histograms

- ❖ Find the number of class intervals (k). $k = \text{sqrt } n$ rounded to the nearest whole number. Where n = number of data points.
- ❖ Determine the range (R). $R = X_{MAX} - X_{MIN}$
- ❖ Find the class width (H). $H = R/k$ rounded to the same number of decimal places as the original sample.
- ❖ Determine the class boundaries. Starting with the smallest data point i.e. the lower end point for the first class, add the class width (H) giving the lower end point for the next class. Repeat until all (k) class intervals are obtained.
- ❖ Construct the frequency table

Software Metrics & Statistical Techniques (2)

- Developing Histograms

- ❖ Count the number of data points (n)
- ❖ Determine the range (R). $R = X_{MAX} - X_{MIN}$

117	65	29	77	61	94	67	108	43	118
133	83	138	87	78	88	114	136	119	67
127	107	82	113	89	72	65	60	107	45
0	94	98	95	47	77	75	98	91	58
10	76	67	133	76	91	32	58	66	106
24	52	94	121	82	44	147	81	102	44
112	48	90	96	57	111	114	78	92	82
107	80	30	108	122	100	52	76	71	50
18	42	62	53	107	53	42	74	33	61
22	136	76	94	91	85	102	85	128	93

Software Metrics & Statistical Techniques (2)

- Developing Histograms

Class	Class Boundaries	Mid-point	Frequency	Total
1	0 - 14	7		2
2	15- 29	22		4
3	30 – 44	37		8
4	45 – 59	52		11
5	60 – 74	67		13
6	75 – 89	82		20
7	90 – 104	97		17
8	105– 119	112		15
9	120 – 134	127		6
10	135 – 149	142		4

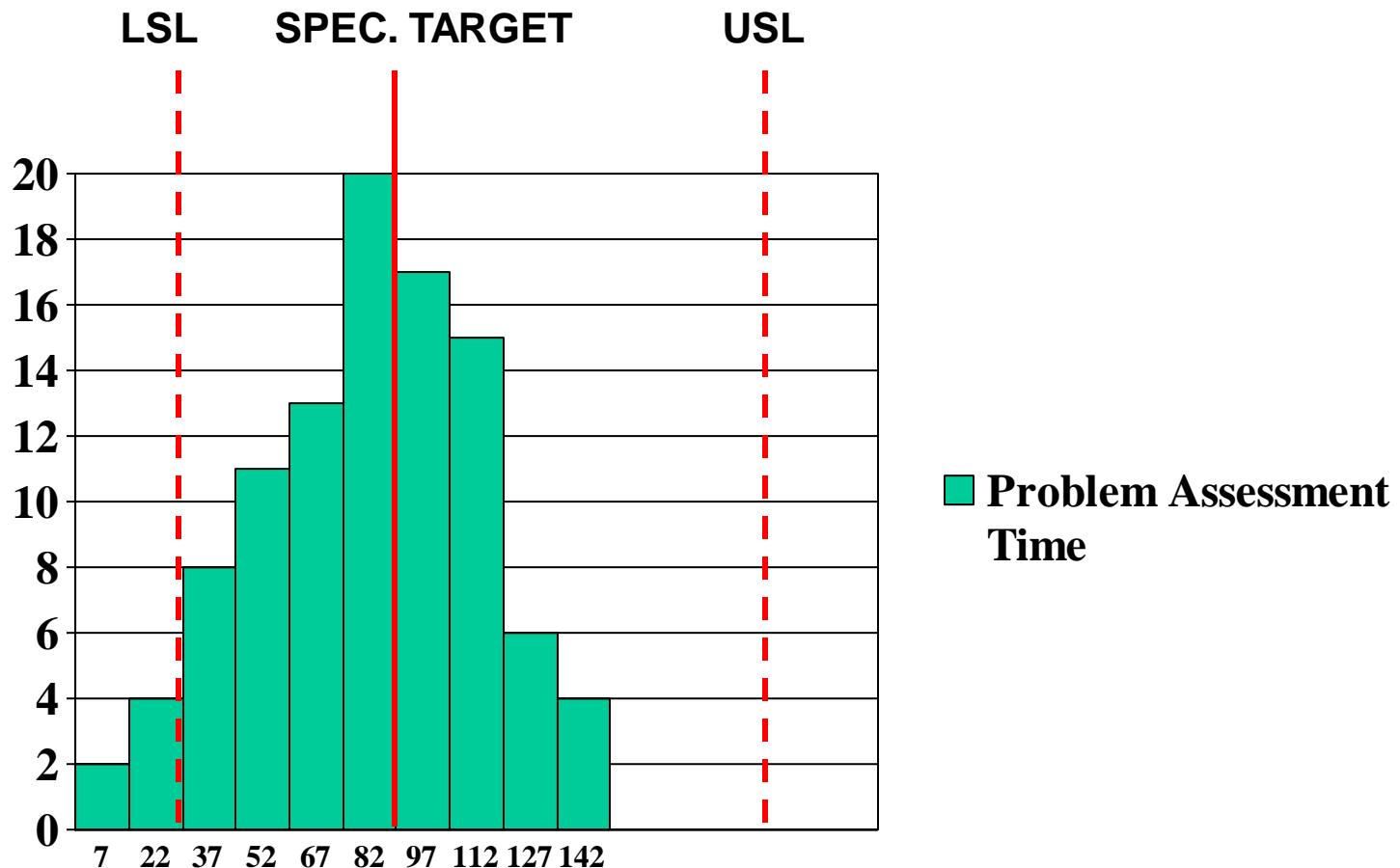
Software Metrics & Statistical Techniques (2)

- Developing Histograms

- ❖ Draw the histogram from the frequency table
- ❖ Draw in the specified target (Spec. Target) and upper and lower specification limits (USL & LSL). These are supplied (for example by a customer) they do not represent the capability of the process.
- ❖ Interpret the histogram – examine its centring, spread & shape as indicators of process capability.

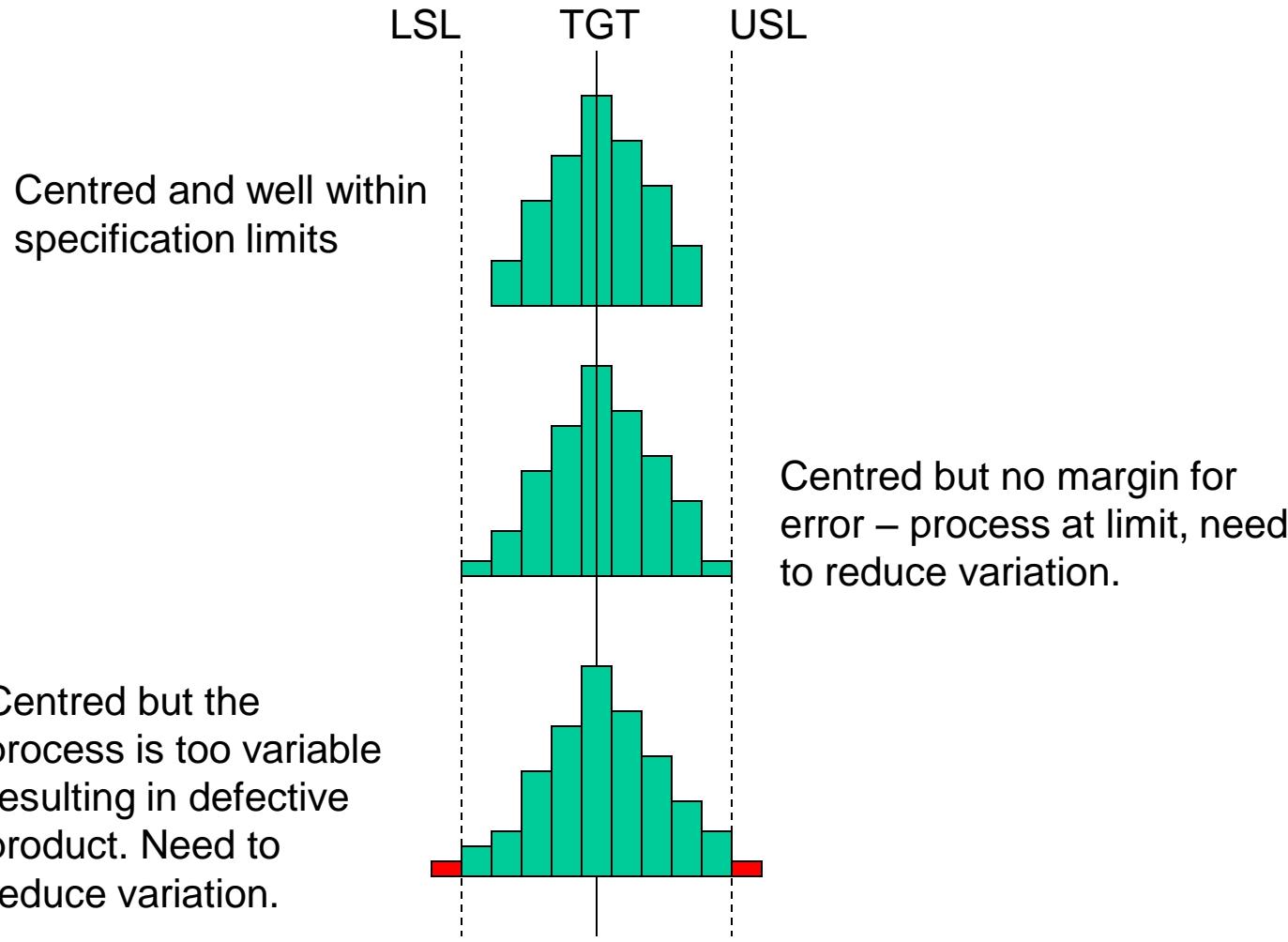
Software Metrics & Statistical Techniques (2)

- Developing Histograms



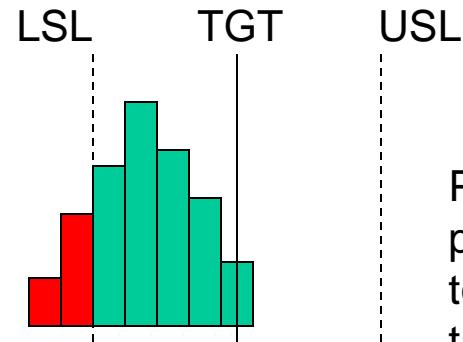
Software Metrics & Statistical Techniques (2)

- Data distributions:

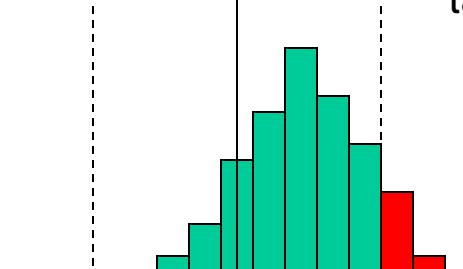


Software Metrics & Statistical Techniques (2)

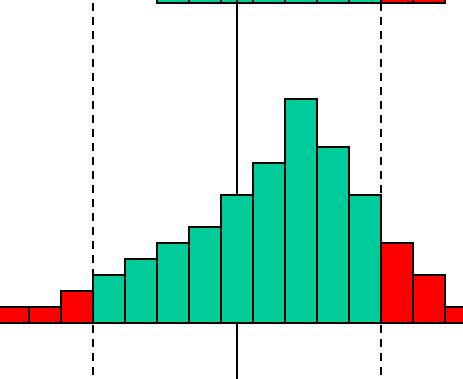
- Data distributions:



Process running low defective product is being made. Need to bring average closer to target



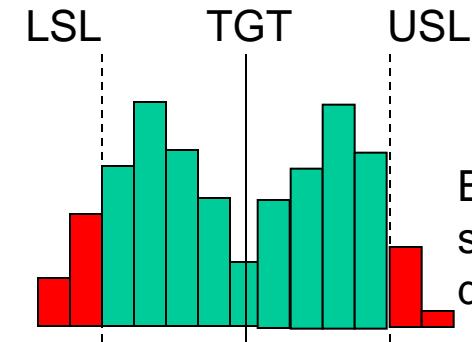
Process running high defective product is being made. Need to bring average closer to target



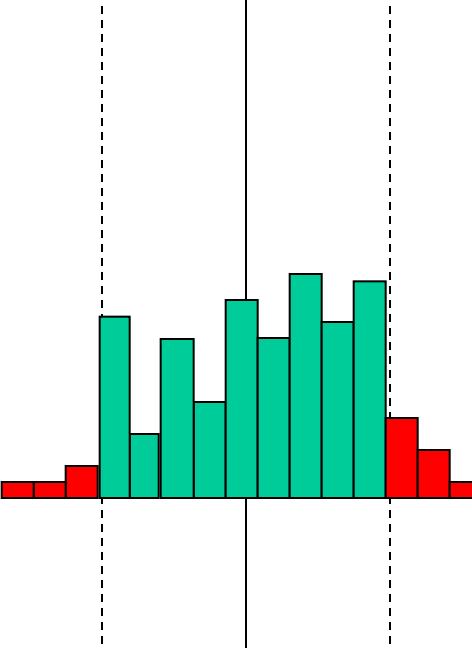
Process is negatively skewed & too variable. Need to bring average closer to target and reduce variation.

Software Metrics & Statistical Techniques (2)

- Data distributions:



Bi-modal distribution - often a sign of two different sources of data



Multi modal – A sign of multiple sources of data not necessarily following the process

Software Metrics & Statistical Techniques (2)

- Exercise

Software Metrics & Statistical Techniques (2)

- Control Charts

- ❖ Focuses on detection and monitoring of process variation over time.
- ❖ Distinguishes special from common causes of variation.
- ❖ Tool for quantitative process control.
- ❖ Helps improve a process to perform consistently and predictably for lower cost, improved quality and higher capacity.

Software Metrics & Statistical Techniques (2)

- Control Charts

- ❖ Based on the type of data & sample size the appropriate control chart is selected:
 - ❖ *Variable Data* can be measured and plotted on a continuous basis e.g. Call centre response time
Example chart types: Average & Range, Average and Standard Deviation
 - ❖ *Attribute Data* can be counted and plotted as discrete events e.g. Number of tests that fail
Example chart types: Fraction Defective, Number of Defects per Unit

Software Metrics & Statistical Techniques (2)

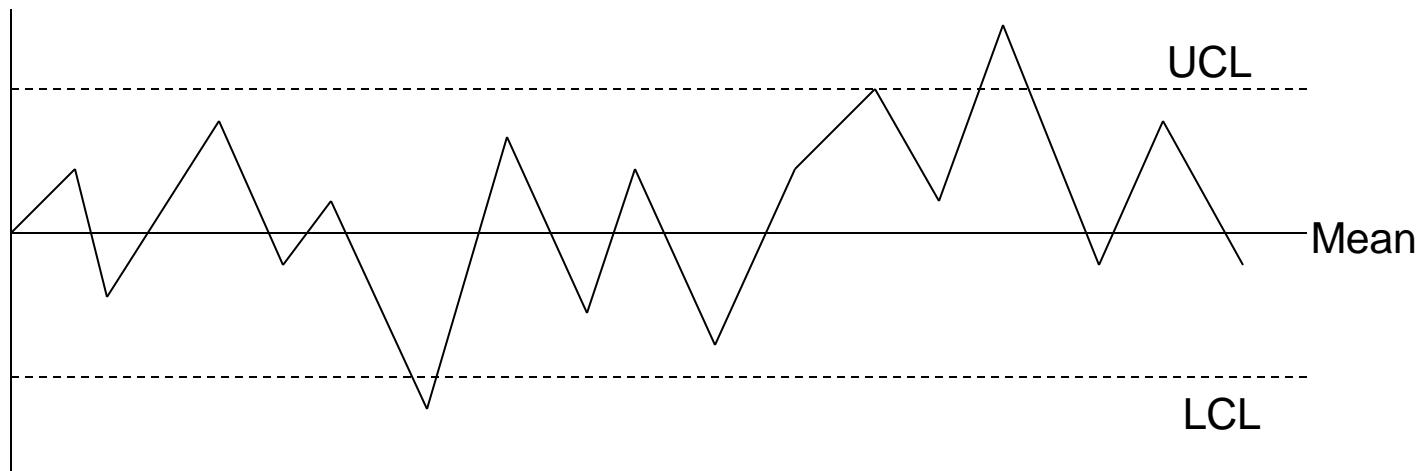
- Control Charts

- ❖ Make sure collected samples are random.
- ❖ Collect samples under similar conditions.
- ❖ Collection frequency depends on the process to be charted.
- ❖ Collect 20-25 sets of samples before calculating statistics and control limits.
- ❖ Calculate the mean and then calculate the upper (UCL) and lower (LCL) control limits as per the formula pertaining to the control chart type that has been selected.

Software Metrics & Statistical Techniques (2)

- Control Charts

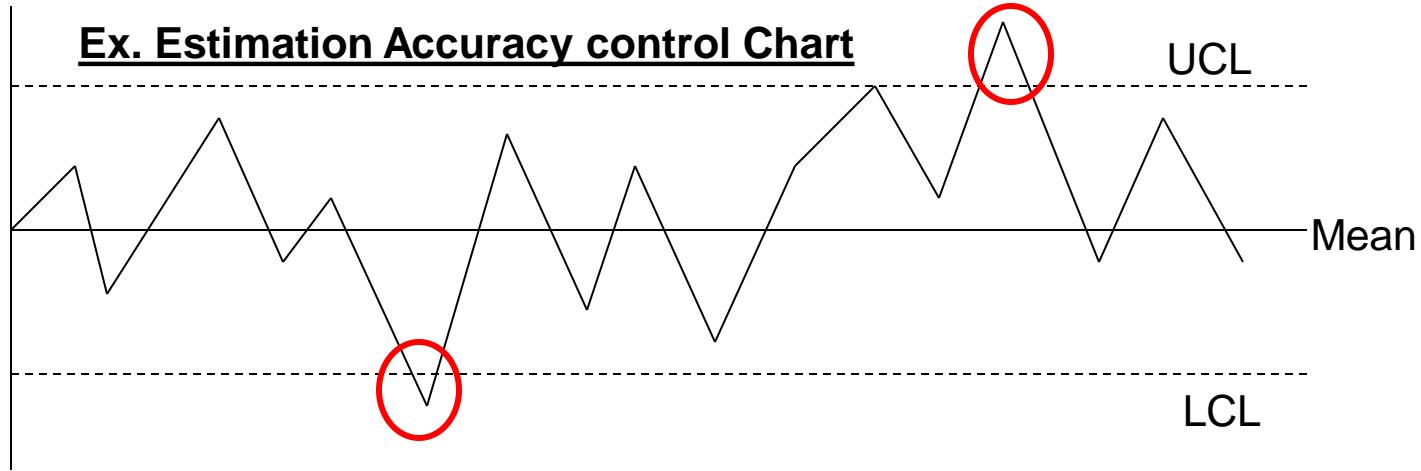
- ❖ Construct the control chart & plot the data. One chart if attribute data e.g. number defective, two if variable data e.g. average and range.



Software Metrics & Statistical Techniques (2)

- Control Charts

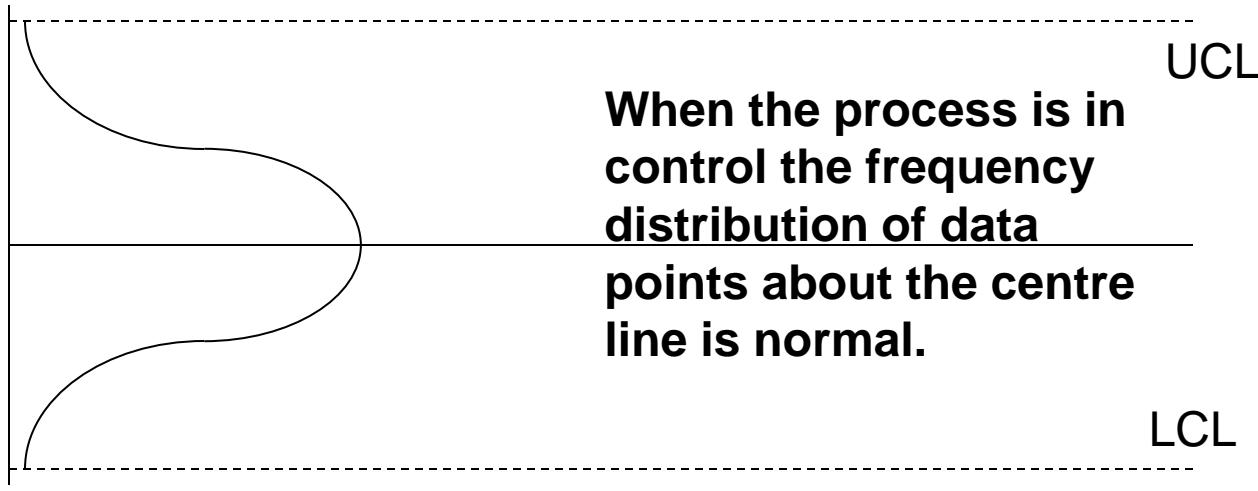
❖ Determine if the Mean is acceptable relative to the Specified Target (ref: histogram), then analyse the data relative to the control limits. Variations within the limits are due to normal variation within the process and are called Common Causes of Variation. Points outside the limits are out-of-control conditions and are called Special Causes of Variation.



Software Metrics & Statistical Techniques (2)

- Control Charts

- ❖ Special causes (unplanned events, human errors, etc.) must be eliminated before the process can be deemed to be ‘in control’
- ❖ When the process is in control this does not necessarily mean that it is capable of meeting specification limits (ref. histogram).



Software Metrics & Statistical Techniques (2)

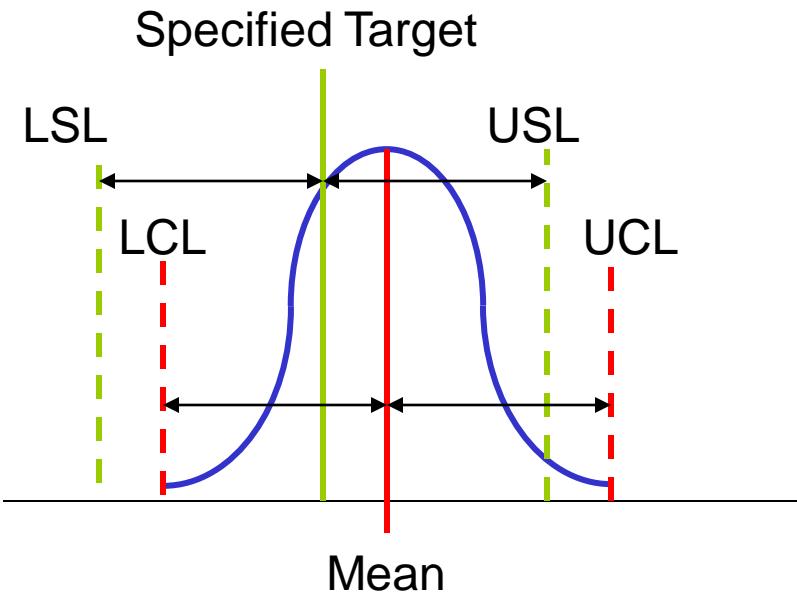
- Process Capability

- ❖ Answers the question “Is the process, given its natural variation, capable of meeting customer specification limits?”
- ❖ Indicates if there has been a change in the process.
- ❖ Helps identify the percent of product or service not meeting customer requirements.

Software Metrics & Statistical Techniques (2)

- Process Capability

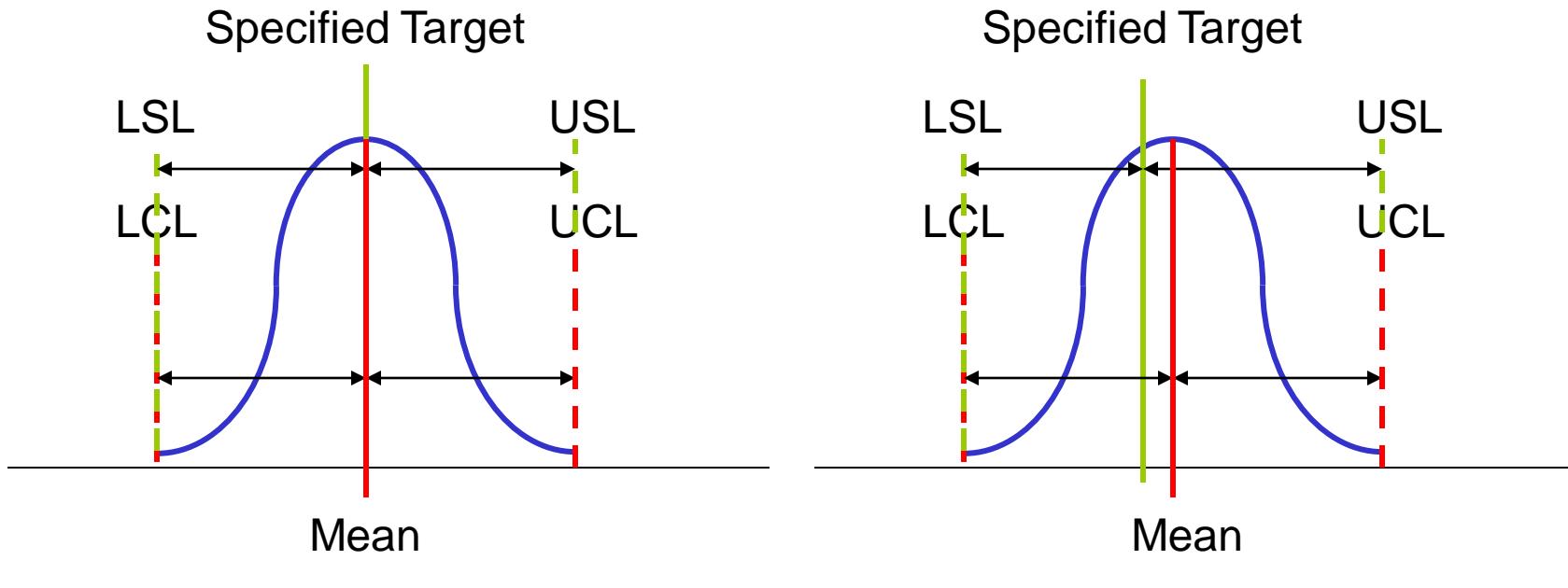
❖ If the process variation exceeds the specification limits then defective product is being produced and the process is therefore incapable of meeting the (customers) specifications.



Software Metrics & Statistical Techniques (2)

- Process Capability

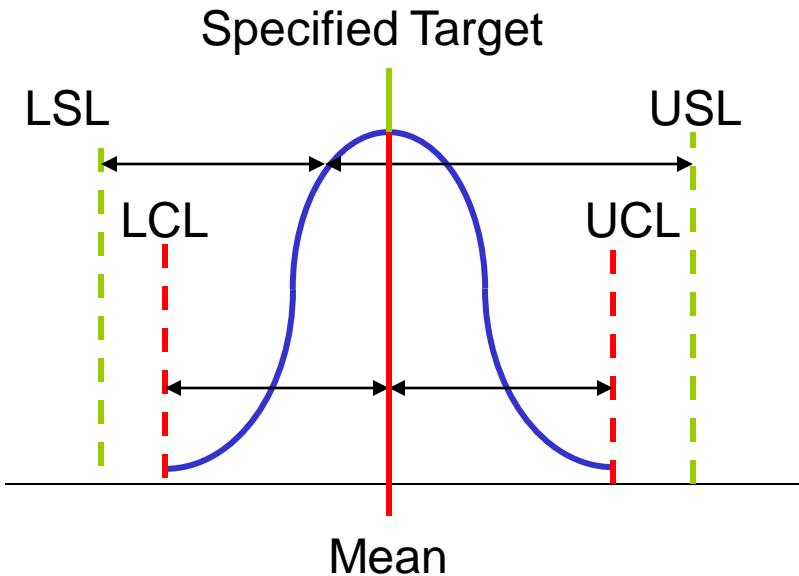
❖ If the process variation just meets the specification minimum then .3% of product is defective – more if the process average (Mean) is not centred on the Specified Target



Software Metrics & Statistical Techniques (2)

- Process Capability

❖ If the process variation is less than the specification limits then the process is capable of meeting customer specifications but again there could still be defective product if the process average is not centred on the specification target.



Software Metrics & Statistical Techniques (2)

- Process Capability

- ❖ If the process is not capable of meeting the specifications then identify and correct the common causes of variation – i.e. take actions to reduce the natural variation in the process.

Software Metrics & Statistical Techniques (2)

- Summary

- ❖ Looked at some basic Process, Product & Project metrics
- ❖ Examined histograms & data distributions
- ❖ Introduced control charts & statistical control
(ref: CMM/CMMI Level 4 & 5)
- ❖ Discussed process capability

- IS Management Models & Techniques

IS Management Models & Techniques

- Previously
 - Examined structured techniques to identify meaningful business metrics
 - Examined different types of metrics for process, product and project
 - Examined histograms and data distributions
 - Examined the idea of control charts and process capability

IS Management Models & Techniques

- Now want to introduce a selection of other management models and techniques that would be supported by these metrics and analysis
 - Balanced Scorecard
 - Customer Relationship Management (CRM)
 - Maslow's Theory of Needs
 - SWOT Analysis
 - The Value Chain

IS Management Models & Techniques

- As we will see the models differ somewhat in their primary focus:
 - Some it could be said are more focussed on **people** (e.g. Maslow's theory of needs) while others are more concerned with **strategy** (e.g. SWOT analysis)
 - Some may be more concerned with **functional processes** (e.g. CMMI or CRM) while others are more **organisational** (e.g. The Balanced Scorecard)
 - Some are a combination of things (e.g. The value chain) involves **strategy and organisation**

IS Management Models & Techniques

- **Balanced Scorecard**

- The BIG idea:

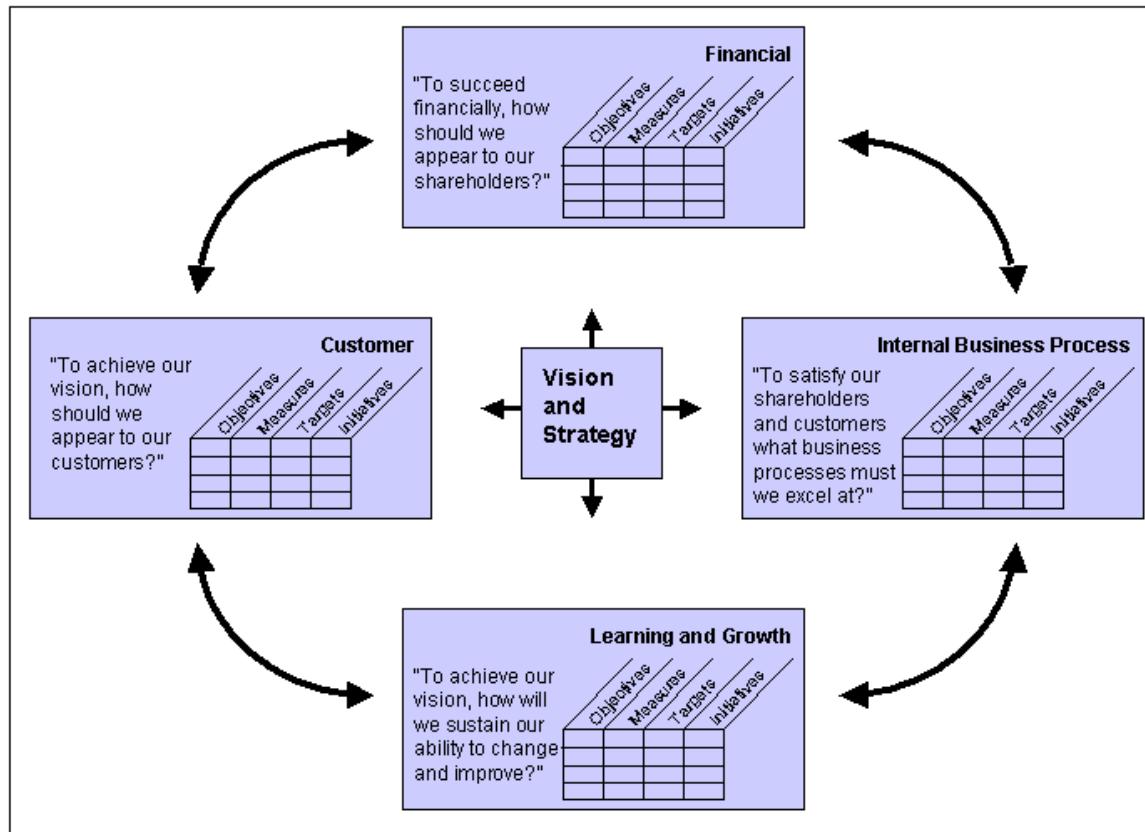
- The Balanced Scorecard seeks to get managers to think about more than just the financial perspective:

- Customer Perspective
 - Business Processes
 - Organisation's Innovation & Learning Ability

and how these other perspectives result in financial consequences for the business

IS Management Models & Techniques

- Balanced Scorecard



IS Management Models & Techniques

- Balanced Scorecard

- From the **financial** perspective:
 - Is the company's choice of strategy, its implementation and its execution contributing to the bottom line?
 - E.G. are we seeing positive change in:
 - operating income
 - sales and revenue growth
 - repeat sales as a % of total sales volume
 - revenue per unit/customer/employee

IS Management Models & Techniques

- Balanced Scorecard

- From the **customer** perspective:
 - How are our efforts as regards service and customer satisfaction affecting our top and bottom line?
 - E.G. are we seeing positive change in:
 - market share in target segments
 - on-time delivery
 - customer complaints frequency
 - existing customer business development

IS Management Models & Techniques

- Balanced Scorecard

- From the business process perspective:
 - How successful is the company at setting up and managing business processes to meet (future) customer demands and deliver service?
 - E.G. are we seeing positive change in:
 - meeting product introduction goals
 - cycle times
 - reported problem severity
 - estimation accuracy

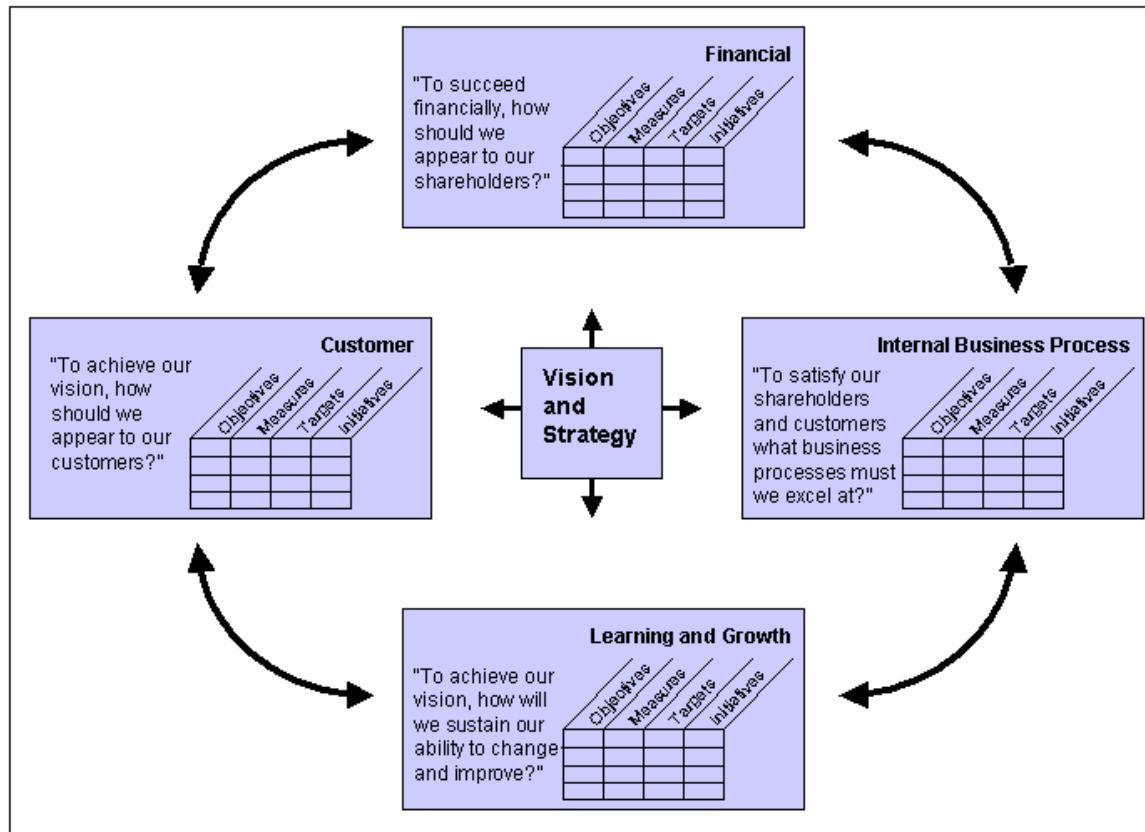
IS Management Models & Techniques

- Balanced Scorecard

- From the learning and growth perspective:
 - Are we successfully managing, developing, and retaining human resources, knowledge and systems?
 - E.G. are we seeing positive change in:
 - employee satisfaction and retention
 - new product ideas emerging
 - process improvement suggestions
 - increased revenue and/or value added per employee/process

IS Management Models & Techniques

- Balanced Scorecard



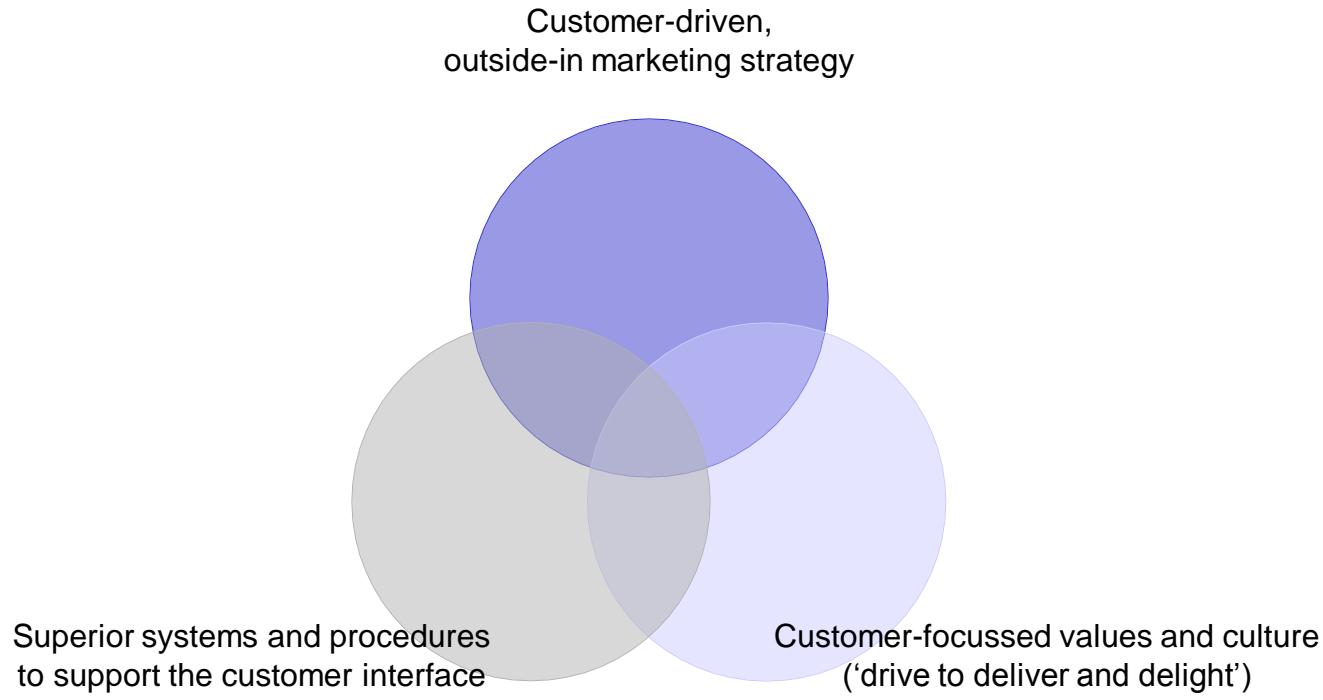
IS Management Models & Techniques

- Customer Relationship Management (CRM)
 - The BIG idea:
 - There is more value to your customer than the current sale. If you can successfully:
 - identify your most valuable customers
 - acquire them
 - keep them
 - grow their purchases
 - you will generate significantly more value than having a one-size-fits-all approach

IS Management Models & Techniques

- Customer Relationship Management (CRM)

- Many support tools exist for CRM but fundamentally its about your customers perceiving they are being treated very well. Done by strategically managing all customer interactions through:

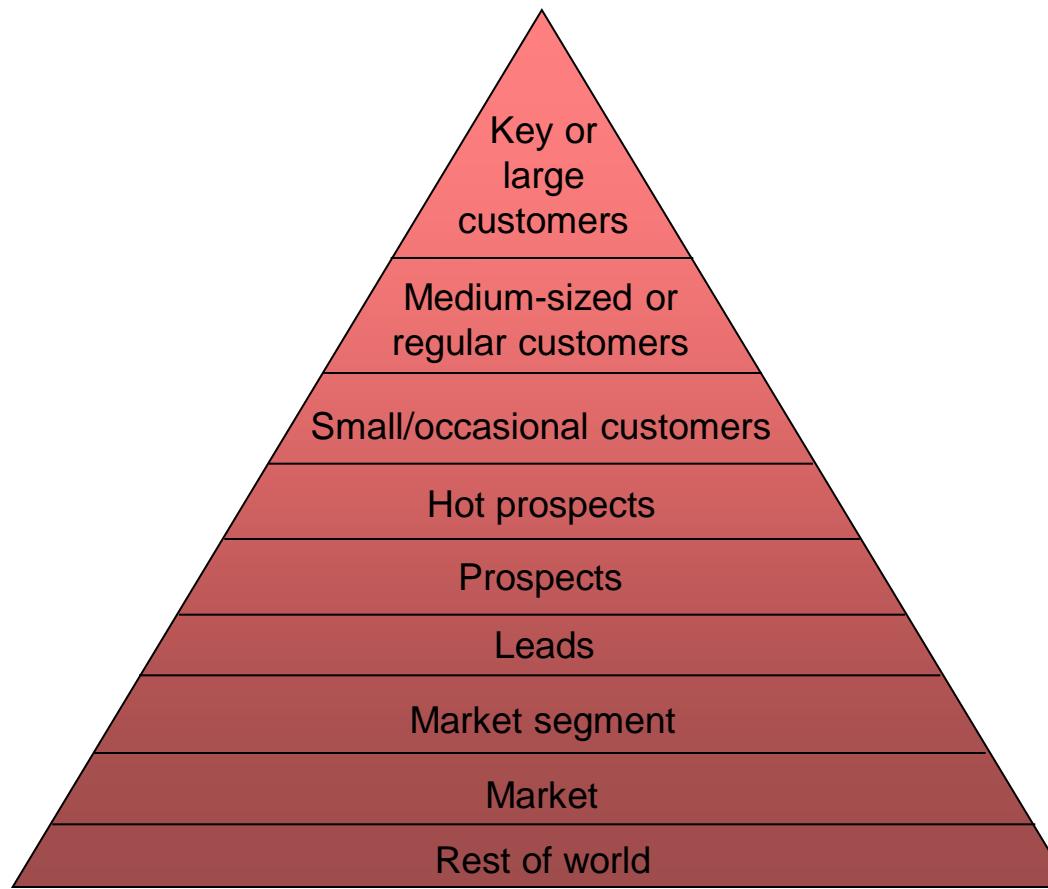


IS Management Models & Techniques

- Customer Relationship Management (CRM)
 - The (pareto) 80/20 rule often applies here – 20% of your customers account for 80% of your profit
 - Once you know who your most (potential) customer is:
 - **Gather** as much relevant information about them as you can
 - **Analyse** the information, re-designing information needs as you go
 - **Create** targets to improve the customer's experience in dealing with you
 - **Choose** the best media, systems and content for your customer communication and interaction
 - **Develop** rules of engagement and 'packages' for customers
 - **Embed** a customer focussed culture in the company
 - **Store and analyse** data on your customers (difficult!)
 - **Develop** your CRM system as you learn more

IS Management Models & Techniques

- Customer Relationship Management (CRM)
 - Key lies in developing a customer hierarchy:



IS Management Models & Techniques

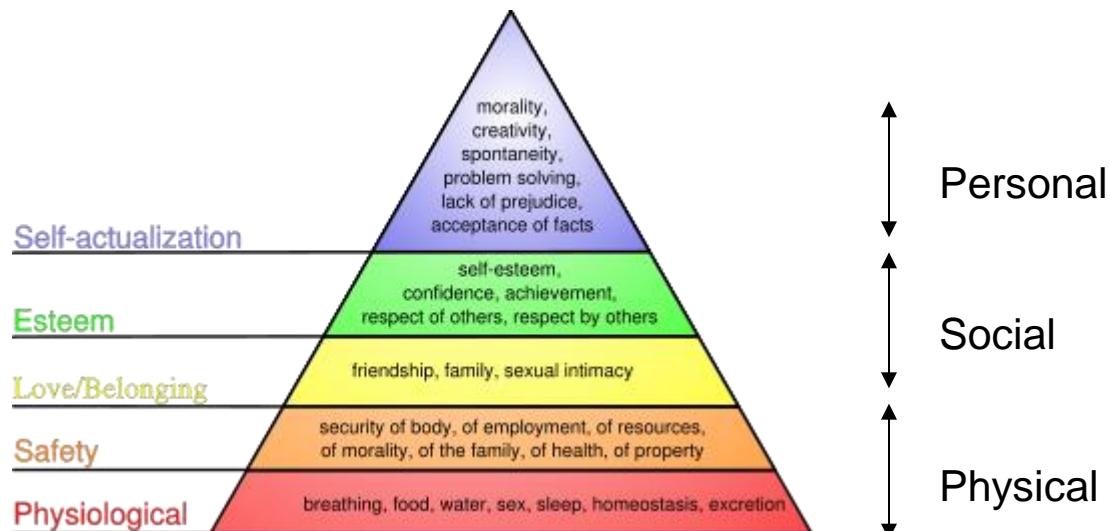
- Maslow: Theory of Needs
 - The BIG idea:
 - Trying to understand what motivates people Maslow developed a hierarchy of needs:
 1. physiological needs (most basic)
 2. safety/certainty
 3. love/belonging/social acceptance
 4. esteem/appreciation
 5. self-actualization (most abstract)

no sooner are the needs at one level met than people start to think about the next level.

IS Management Models & Techniques

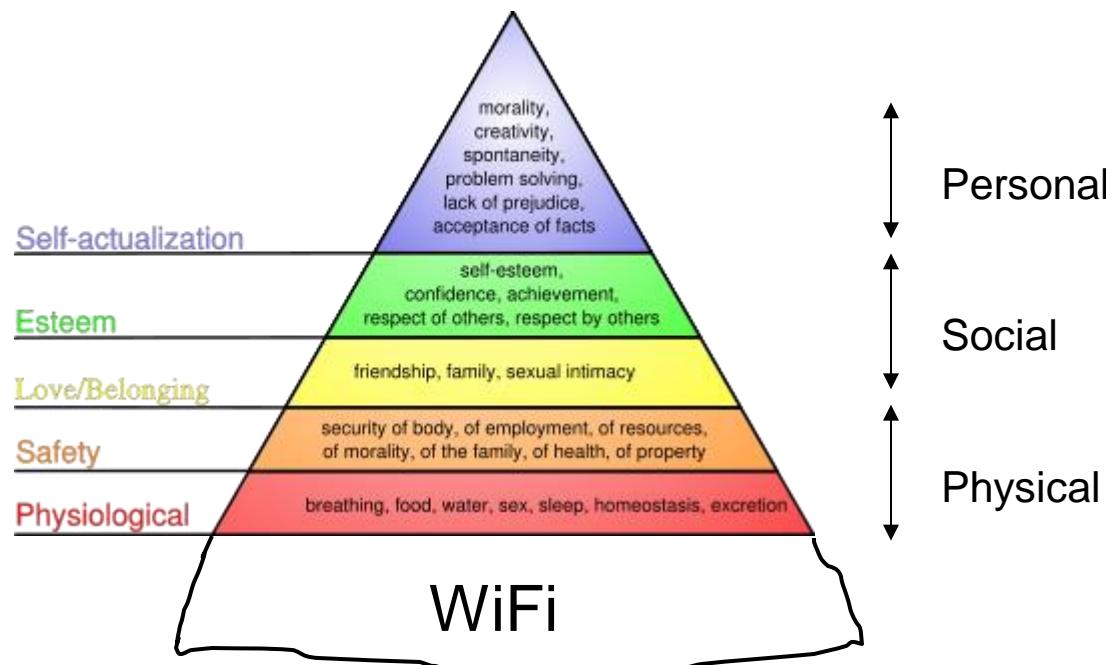
- Maslow: Theory of Needs

- Usually represented as a pyramid with the more primitive physiological needs (food, water, sleep, etc.) at the lowest level, social needs (friendship, achievement, respect) at the mid-level, higher 'self-actualization' needs (morality, creativity, problem solving) at the top



IS Management Models & Techniques

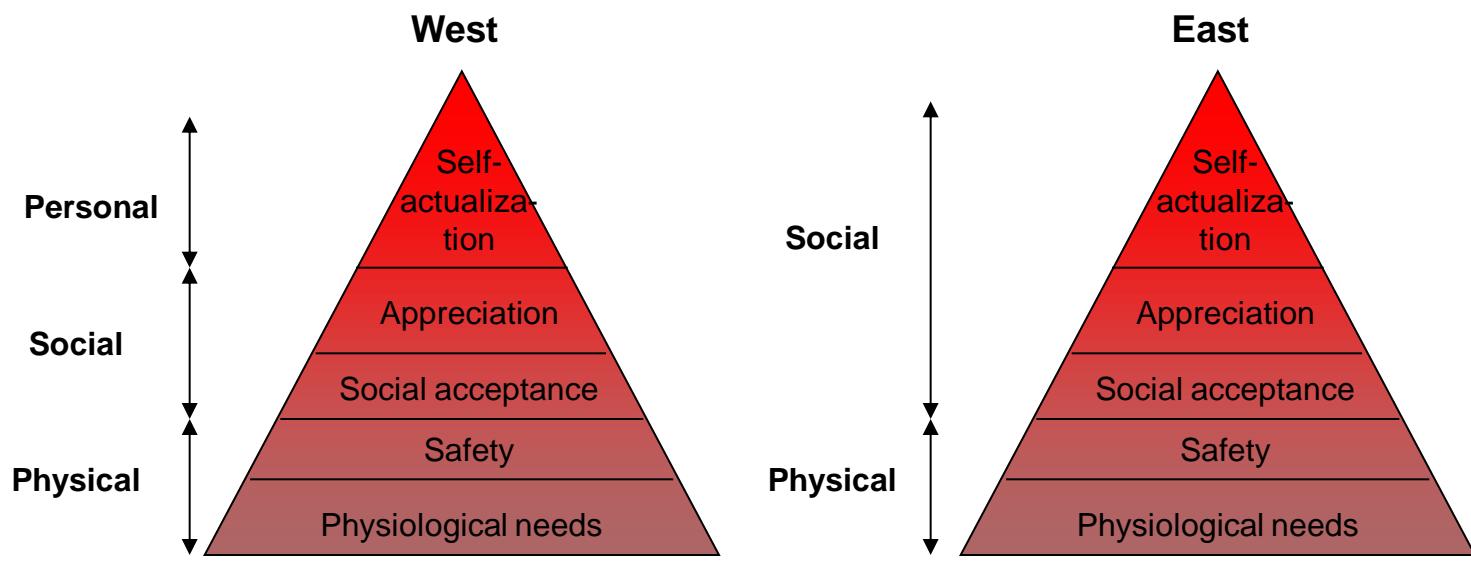
- Maslow: Theory of Needs – A Modern Update



IS Management Models & Techniques

- Maslow: Theory of Needs

- So the theory goes that what motivates you as a customer or an employee will depend on where you are on the pyramid at a particular point in time
- But we need to remember that Maslow based his needs pyramid very much on American/western European norms and values – in Asia, for example, ‘social’ needs are much more important than ‘personal’ needs



IS Management Models & Techniques

- SWOT Analysis
 - The BIG idea:
 - Any company undertaking strategic planning will at some stage wish to evaluate its strengths and weaknesses. When combined with an inventory of opportunities and threats in the external environment the company is conducting a SWOT analysis.
 - Strengths
 - Weaknesses
 - Opportunities
 - Threats

IS Management Models & Techniques

- SWOT Analysis

- Strengths:

- What is the company really good at?
 - E.G. can we identify strengths in:
 - our sales force
 - our engineering expertise
 - our brand
 - our process capability

Note: that strengths are *not*: a growing market, new products, etc.

IS Management Models & Techniques

- SWOT Analysis

- Weaknesses:

- Where is the company not so strong?
 - E.G. we may identify weaknesses in:
 - our sales force
 - our engineering expertise
 - our brand
 - our process capability

Note: that weaknesses are not the inverse of strengths. It's a relative term. We may have a particular weakness but our competitors may also have the same weakness or worse

IS Management Models & Techniques

- SWOT Analysis

- Strengths and Weaknesses we can identify by:

- auditing ourselves using available standards

- or by benchmarking our company against:

- competitors

- the broader industry

- competing units within our own business

- unrelated industries but sharing some common trait (e.g. same distribution channels)

IS Management Models & Techniques

- SWOT Analysis

- Opportunities and Threats however occur as a result of external forces. For instance:

- demographic shifts
 - economic change
 - technological developments
 - political change
 - social and cultural dynamics
 - industry specific forces (customers, competitors, distribution channels, suppliers)

IS Management Models & Techniques

- SWOT Analysis

- Opportunities:

- Can we, for example, leverage off:
 - technological developments
 - demographic shifts
 - a successful partnership
 - new markets we could get into
 - licensing our technology
 - selling patents

The level of detail and the (perceived) degree of realism determine the extent of the opportunity

IS Management Models & Techniques

- SWOT Analysis

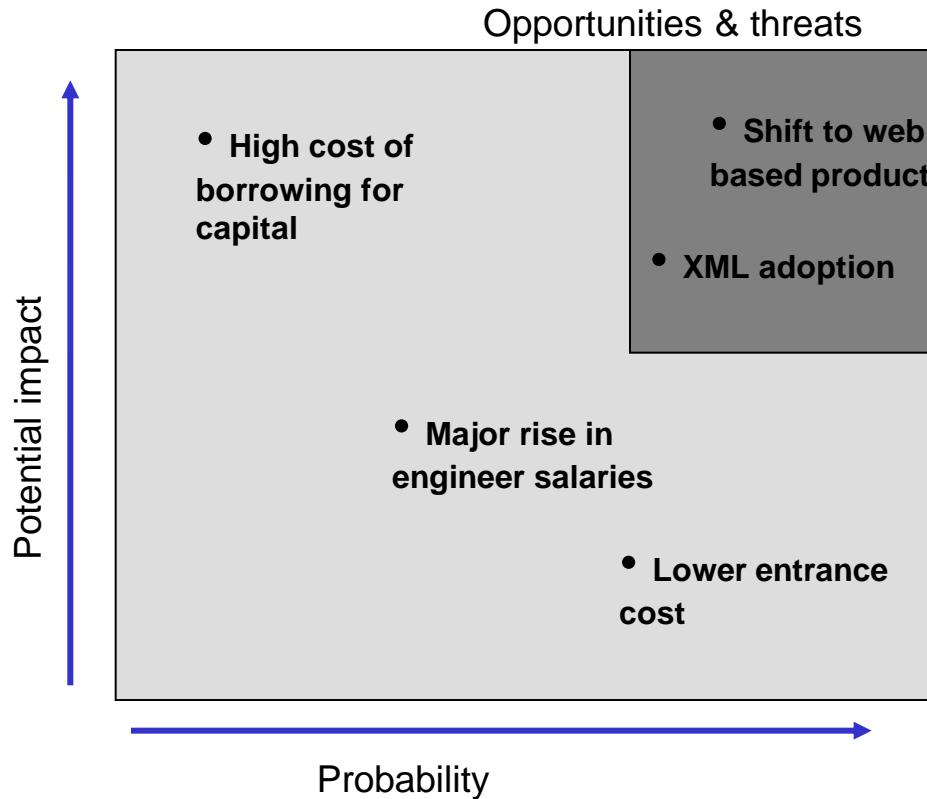
- Threats:

- Are we likely to be adversely affected by:
 - technological developments
 - demographic shifts
 - changes in regulations
 - cheaper substitutes
 - our competitors opportunity
 - commercial dangers (lower sales, higher operational costs, interest rate rises, shrinking margins, etc.)

Both opportunities and threats can be classified according to their potential impact and the actual probability

IS Management Models & Techniques

- SWOT Analysis



- Analysing SWOT's can be very difficult but the next step **deciding on what actions** need to be taken is even more difficult:

IS Management Models & Techniques

- SWOT Action taking

	Strengths (S)	Weaknesses (W)
Opportunities (O)	<ul style="list-style-type: none">• SO Strategies Use strengths to take advantage of opportunities	<ul style="list-style-type: none">• WO Strategies Take advantage of opportunities by overcoming weaknesses
Threats (T)	<ul style="list-style-type: none">• ST Strategies Use strengths to avoid threats	<ul style="list-style-type: none">• WT Strategies Minimize weaknesses and avoid threats

- ‘SO’ and ‘WT’ strategies are obvious. A company should do what it’s good at when the opportunity arises & avoid businesses or applications that it does not have the competencies for.

IS Management Models & Techniques

- SWOT Action taking

	Strengths (S)	Weaknesses (W)
Opportunities (O)	<ul style="list-style-type: none">• SO Strategies Use strengths to take advantage of opportunities	<ul style="list-style-type: none">• WO Strategies Take advantage of opportunities by overcoming weaknesses
Threats (T)	<ul style="list-style-type: none">• ST Strategies Use strengths to avoid threats	<ul style="list-style-type: none">• WT Strategies Minimize weaknesses and avoid threats

- ‘WO’ and ‘ST’ strategies are less obvious and more daring.
- Taking on an opportunity without having the necessary strengths, in ‘WO’, means those strengths must be developed, bought/borrowed, or the company must somehow outmanoeuvre the competition
- Companies using ‘ST’ strategies essentially ‘buy or bust’ their way out of trouble. Often seen when a large company undercuts a smaller rival & there is a price war, huge marketing campaigns, or promotional offers

IS Management Models & Techniques

- SWOT Analysis

- Some further comments
- SWOT analysis is a very powerful tool but can be difficult to do well especially if the company is not completely honest with itself regarding its strengths and weaknesses
- Assessing the impacts from opportunities or threats is also a difficult task and again can be adversely affected usually by too much optimism
- SWOT analysis cannot help with translating analysis into action taking. Often, presented with a range of alternatives, managers will dither and valuable time may be lost

IS Management Models & Techniques

- SWOT Analysis Exercise

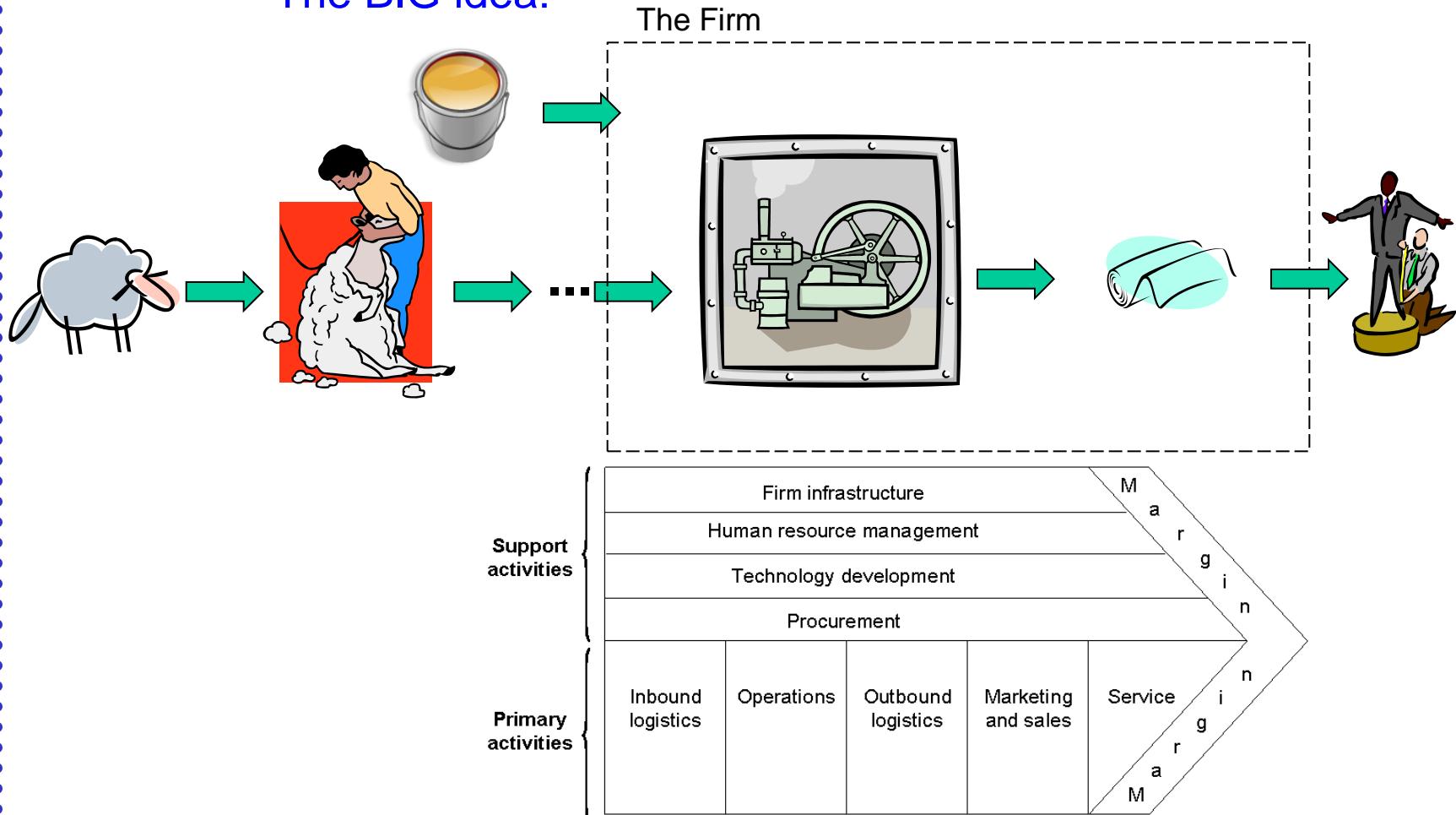
- Our company manufactures CD's & DVD's & they are of high quality. Of the blank CD's/DVD's 50% are for our own use and the other 50% we sell for home use – our stock management system won an efficiency award
- We get the master recordings directly from recording studios or game manufacturers and we then mass produce. We directly print any artwork onto the CD's/DVD's which can be expensive but its very good. We buy in plastic jewel cases and the price has been rising due to raw materials costs which is affecting all suppliers. Card inserts are printed and supplied to us from any number of print shops
- Lately our management has become concerned about the increased use of USB keys and the rise of direct downloads for music and film we also have a very aggressive competitor who manufactures their own jewel cases and prints the card inserts - but they look cheap
- We are to conduct a SWOT analysis to help forecast our strategy

IS Management Models & Techniques

- The Value Chain
 - The BIG idea:
 - Competitive advantage can be understood only by looking at a business as a whole.
 - Cost advantages and successful differentiation are found throughout the chain of activities that a firm performs to add value to its customers.
 - Advantage or disadvantage can occur at any one of 5 primary and 4 secondary activities. Together they form the value chain

IS Management Models & Techniques

- The Value Chain
 - The BIG idea:



IS Management Models & Techniques

- The Value Chain
 - Primary Activities: Inbound Logistics
 - Inbound logistics activities can include:
 - receiving
 - storing
 - listing
 - approving and
 - grouping inputs to the product
 - and includes functions such as:
 - materials handling
 - warehousing
 - inventory management
 - supplier management
 - transportation scheduling

IS Management Models & Techniques

- The Value Chain
 - Primary Activities: Operations
 - Operations activities can include:
 - machining
 - packaging
 - assembling
 - equipment maintenance
 - product testing
 - In a software situation we could also add:
 - requirements management
 - design
 - coding
 - subcontractor management
 - software testing in its many and varied forms

IS Management Models & Techniques

- The Value Chain
 - Primary Activities: Outbound Logistics
 - Outbound logistics activities can include:
 - order processing
 - warehousing
 - scheduling transportation
 - distribution management
 - In a software situation we could also add:
 - software release management
 - configuration management

IS Management Models & Techniques

- The Value Chain

- Primary Activities: Marketing & Sales

- Marketing & Sales activities make (or convince) buyers go for your product or service these can include:

- advertising
 - promotion
 - selling
 - channel selection
 - retail management

- In the software situation quite often things such as capability maturity ratings or compliance with certain quality standards are also used in advertising and in promoting the company

IS Management Models & Techniques

- The Value Chain

- Primary Activities: Service

- Service activities maintain the product after sales these can include:

- installation
 - training
 - servicing
 - help-desk management
 - customer care
 - maintenance release management

IS Management Models & Techniques

- The Value Chain
 - Support Activities
 - These are activities which facilitate the primary value chain activities.
 - They include general management activities
 - Human Resource management
 - Research and Development
 - Procurement

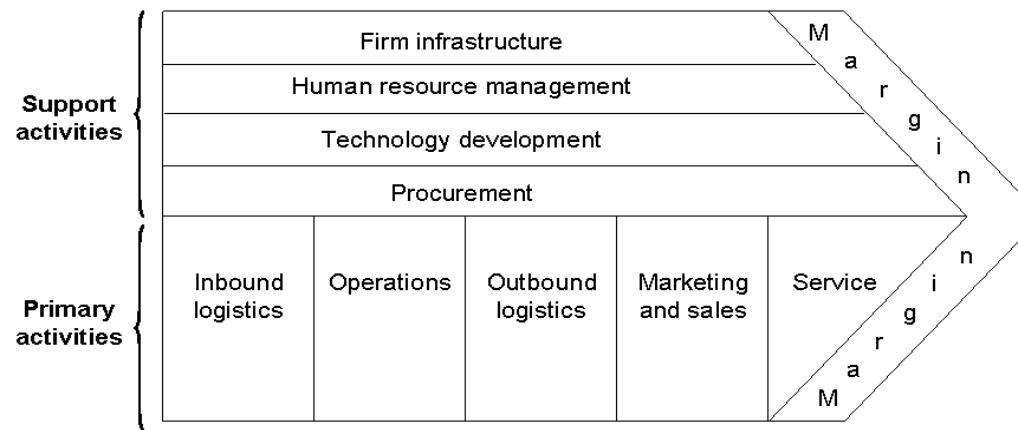
IS Management Models & Techniques

- The Value Chain

- Support Activities: Procurement

- Procurement activities include negotiating and purchasing:

- raw materials
 - supplies
 - services
 - contract negotiation with suppliers and sub-contractors
 - leases
 - licences



IS Management Models & Techniques

- The Value Chain
 - Support Activities: Technology Development
 - Technology development activities can include:
 - research and development
 - product improvement
 - process improvement
 - new service development
 - (re) design

IS Management Models & Techniques

- The Value Chain

- Support Activities: Human Resource Management

- Human resource management activities can include:

- recruitment
 - education
 - compensation
 - retention
 - career planning

IS Management Models & Techniques

- The Value Chain
 - Support Activities: Firm Infrastructure
 - Firm infrastructure activities can include:
 - general management
 - planning procedures
 - finance
 - accounting
 - public relations
 - Quality management

IS Management Models & Techniques

- The Value Chain

- To analyse a firm's competitive advantage (or lack of) the value chain is used to separate the firm's activities into detailed discrete activities which allow the relative performance of the business units to be determined
- Based upon this analysis and breakdown appropriate strategies can be selected and we can determine areas of competitive advantage the firm has or needs to develop
- Strategic planners and consultants have used the value chain extensively for:
 - strategic alliances
 - mergers and acquisitions
 - developing quantitative indices for assessment

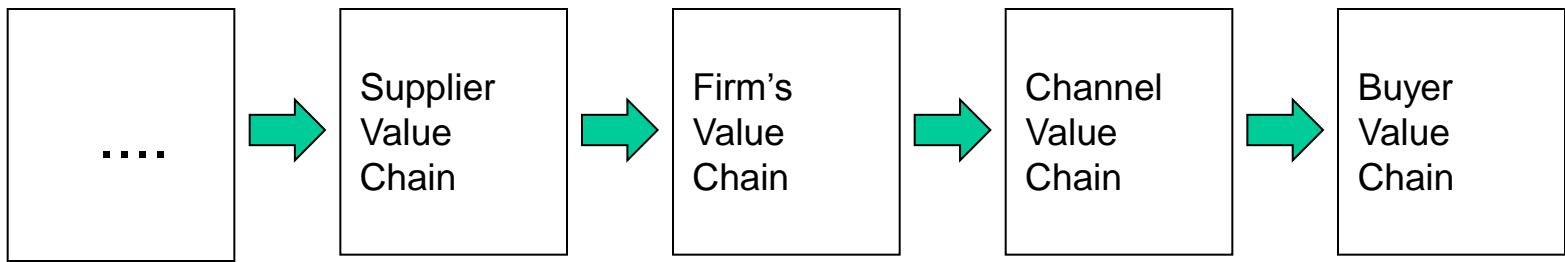
IS Management Models & Techniques

•The Value System

- A firm's value chain is part of a larger system that includes the value chains of upstream suppliers and downstream channels and customers.

Porter calls this series of value chains the *value system*.

The Value System



Linkages exist not only in a firm's value chain, but also between value chains.

IS Management Models & Techniques

- Summary

- Examined a range of different tools and methods that have different emphasis (strategic, organisational, and so on) which are commonly used for IS business improvement:
 - Balanced Scorecard
 - Customer Relationship Management (CRM)
 - Maslow's Theory of Needs
 - SWOT Analysis
 - The Value Chain

Information Systems

Where were we?

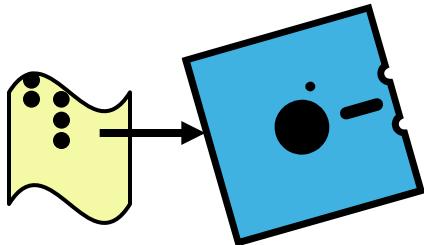
Where are we now?

Where do we need to go?

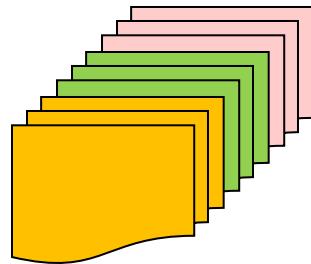
Into the white space

Computing: where were we?

25/30 years ago:



Paper media (punch tape & cards) was slowly giving way to Magnetic media (Cassette tapes & 8" floppy disks)



Batch jobs pre-dominated



P.C.'s were in their infancy



Very large rooms full of not very reliable but expensive computing machinery & huge air conditioning units



People spoke of Kilobytes & 16-bit processors



Huge numbers of support staff working 24 hour shifts:
Data entry clerks
Batch Controllers
Computer Operators
Paper handling staff
Technical support staff
Operations management staff

A trip back in time...

2nd & 3rd generation software languages dominated: Cobol, Algol, Fortran, PL1, Basic,& Assembler



Computer networking was in its infancy – Some universities were experimenting with Lisp & APL

Processes for the development of software were either non-existent or poorly understood - what did exist was borrowed from civil engineering

Software people came strictly from engineering, mathematics, or science backgrounds

A trip back in time...

Software screw up's were commonplace in many industries and frequently spectacular if:

- Large numbers of customers or their money was affected (banks, reservation systems, billing systems)
- The wrong chemicals or their proportions got mixed up (pharmaceuticals, chemical industries)
- An industrial valve that should have been open but was actually closed meant that high pressure gases had nowhere to go (industrial processes)
- When chunks of metal took to the air - and that wasn't the original intention (high profile accidents)

Why were we screwing up?

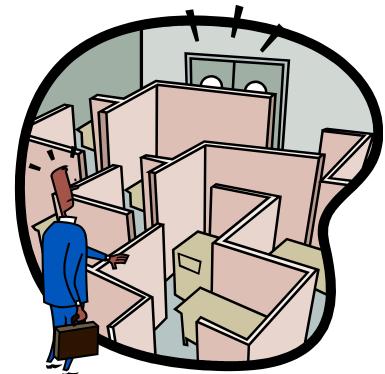
The truth was that we didn't worry too much about:



Documentation



User interfaces



Processes

Because the users were expected to be people just like us: engineers, scientists & technologists

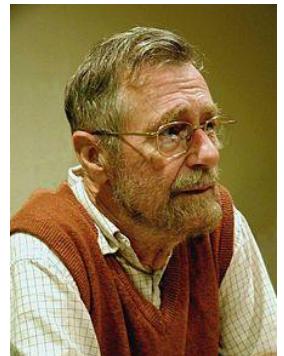
Quite often software was not regarded as a commodity in its own right - frequently we gave the software away for free just to sell hardware

Why we were screwing up!

But gradually we came to realise:

1. We needed processes for producing good software just as there were processes for producing good cars, good food, or drugs
 2. The majority of software users were not techies
 3. That good documentation was really important
- but the screw up's continued...

Leading Edsger Dijkstra to be first to refer to
“The Software Crisis”



...and then all of a sudden!

.. the technology took off

Paper... Magnetic media....Optical media....Solid state

Batch jobs

Dumb mainframe terminals

Workstations & Servers

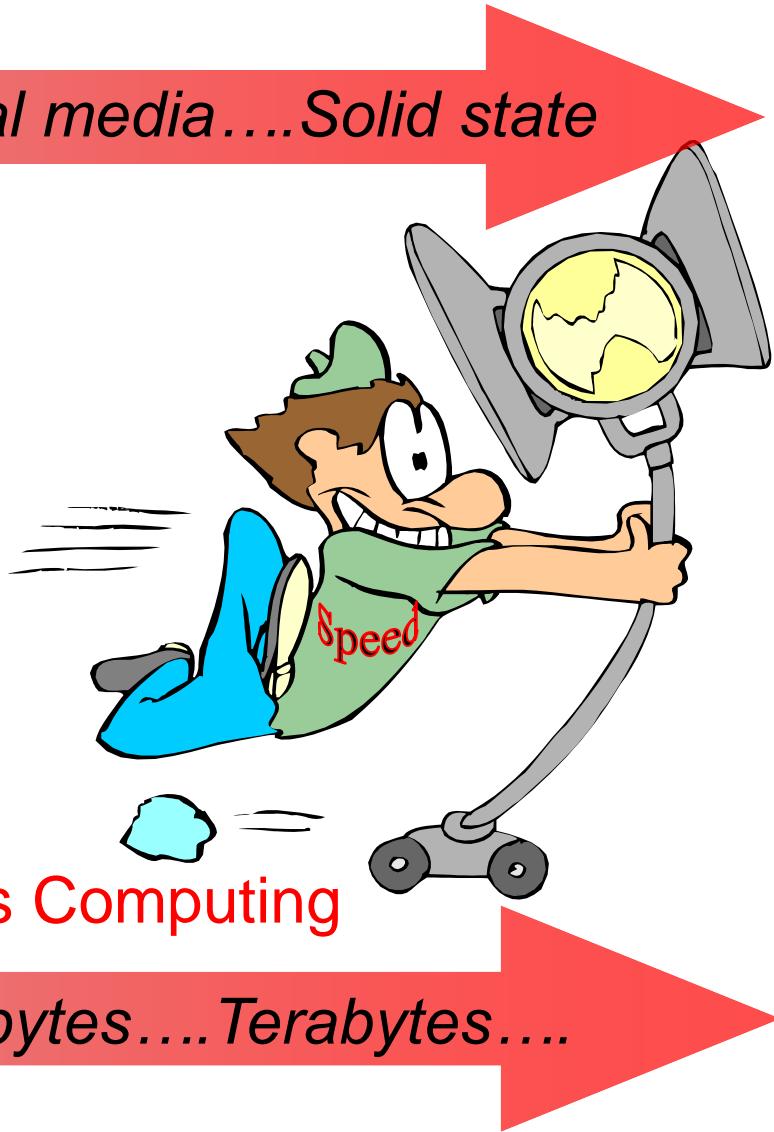
Desktop systems

Laptops

Wi-Fi

Ubiquitous Computing

Kilobytes....Megabytes....Gigabytes....Terabytes....



& software was everywhere...

The Internet



Cash machines
Chip & PIN cards
Internet banking
NFC

Pervasive Software



Intelligent TV's,
Cookers, Washing
Machines..
Satellite Television
CD's & DVD's
WIFI

GPS



Cameras
Games
Exercise

Email

Healthcare

Sports

Travel

Education

Agriculture

Transport

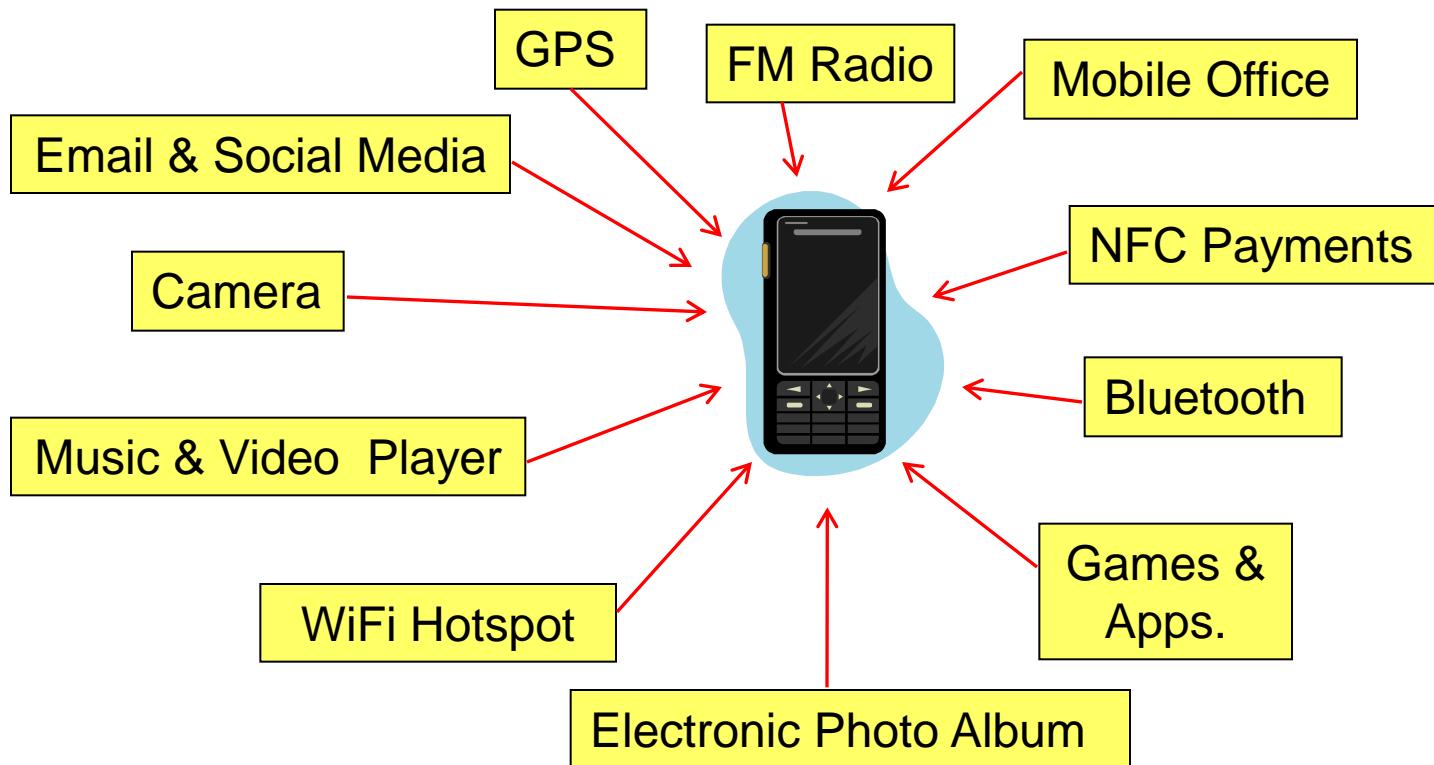
Manufacturing

Security

+

Today...

We see technology converging - in phones for example:



Today

Software is centre stage because:

- Huge advances in underlying hardware technology
- Pervasive we have absolute dependence upon it
- Part of peoples lives we grow up with it we expect it
- Can't do certain work functions anymore without it

And yet....

Today

We still screw up...

- Heathrow T5 Baggage Handling
- UK Passport Agency
- Toyota Prius Recall
- Identity Theft
- Mars Climate Orbiter
- Quantas Flight 72
- **Many, many more:** Taurus share dealing system, NASA Mars Rover, AT&T Telephone network, etc.



A home-grown example

PPARS (Personnel, Payroll & Related Systems) for HSE:

HSE figures:

- Project estimates: €9m (2002)
- Actual cost: **€220m (25x over estimate)**
- Schedule: **Over 4 years behind**
- Content/Functionality: **Still has major flaws, “can only pay 30,000 of 136,000 HSE staff”**
- System Deployment: **“Major problems with adoption & rollout of system”**
 - Comptroller & Auditor General
- Conclusion: **“needs replacing”.**

Not a whole lot has changed!

The **essential** difficulties with software development
(**invisible, complex, conformity, and changeability**),
identified 25 years ago!!, are still there (Brooks, No Silver Bullet, April 1987)

and...

“Despite numerous initiatives over the years nothing has
***essentially* changed**”. (Boehm, 2006)

We're still screwing up!

Ask again why?

What is wrong with...

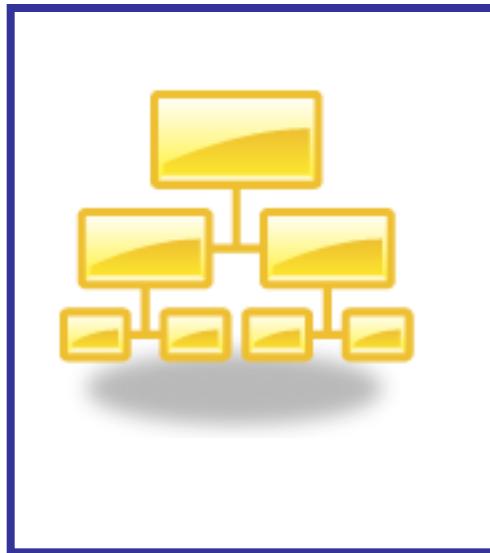
- the way we develop software?
- control software projects?
- describe software work processes?
- understand software organisations & teams?
- visualise the flow of information in software projects?

Because we have done this for loads of other industries!

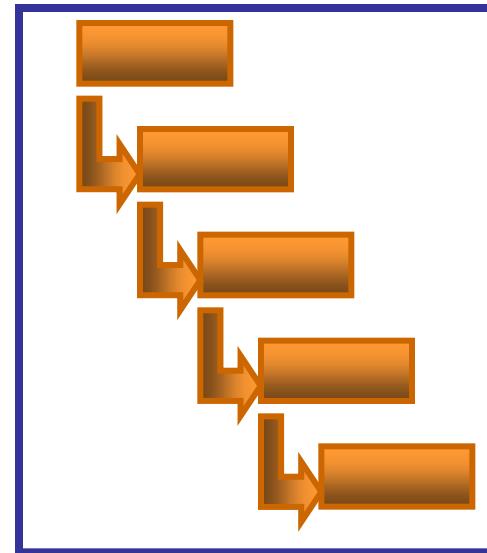
Why doesn't it work or software?

We use the same Models & Abstractions!

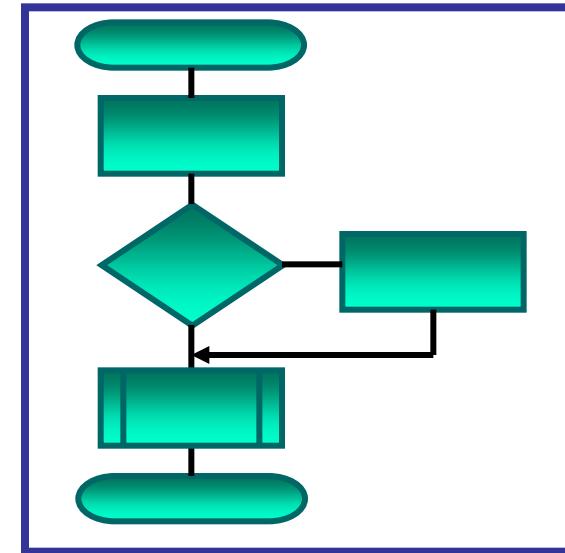
Org. Charts



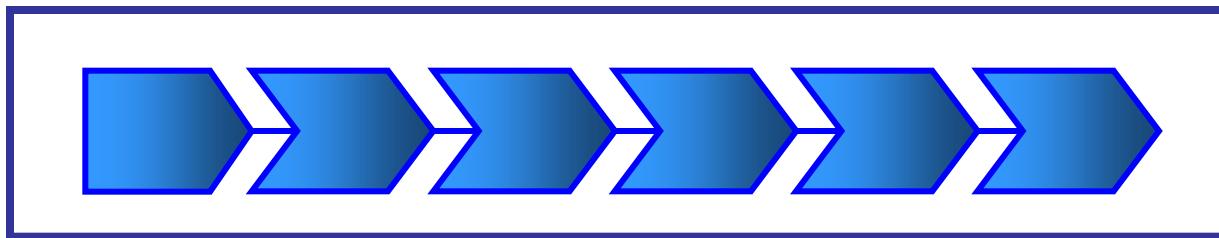
Process Maps



Flowcharts



Value Chains



Development Models

Traditional: Waterfall, V-Model

Agile: XP, DSDM, SCRUM

Accurate Descriptions?

Do these models truly reflect what goes on in software development projects?

Have we talked ourselves into believing our mechanistic, sterile descriptions of software development processes and organisations actually represent reality?

Sterile descriptions?

‘... fictions to present an image of control or to provide a symbolic status’
(Nandhakumar, J. & Avison, J., 1999)

Or

“...unattainable ideals and hypothetical “straw men” that provide normative guidance to utopian development situations?” (Truex, D., Baskerville, R. & Travis, J., 2000)

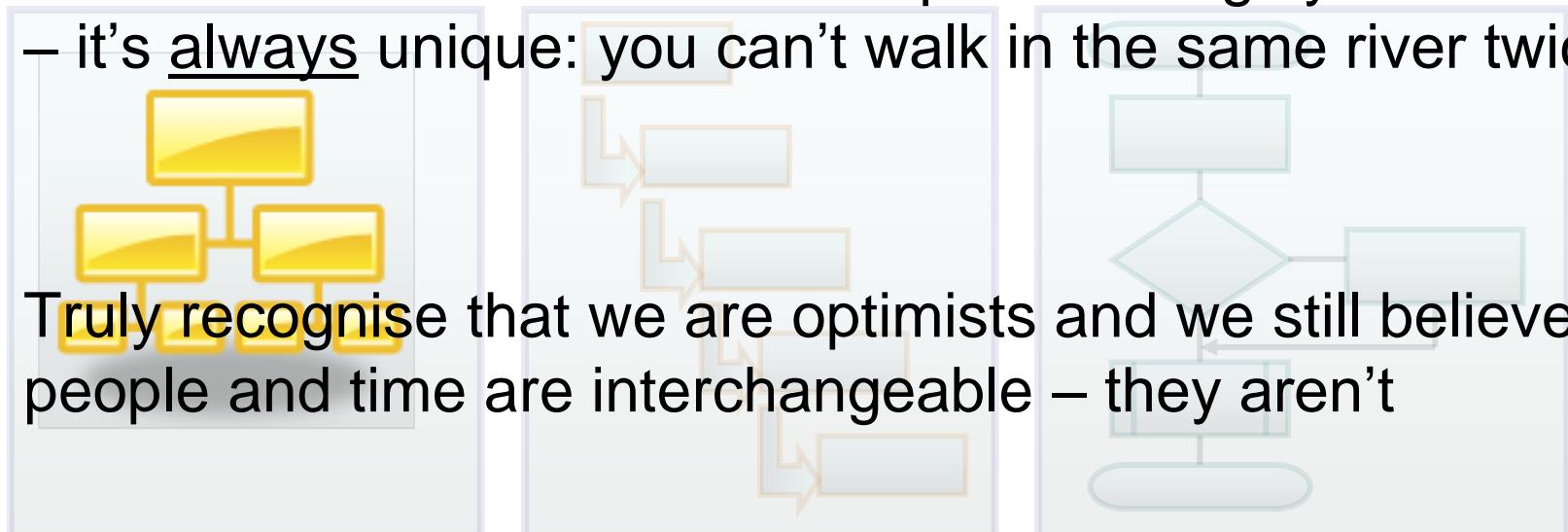
So where do we need to go if we want to truly understand what is going on & why we continue to screw up something so central and important to our lives?

Where do we need to go?

Into the ‘white space’ between the boxes & symbols in our carefully constructed models:

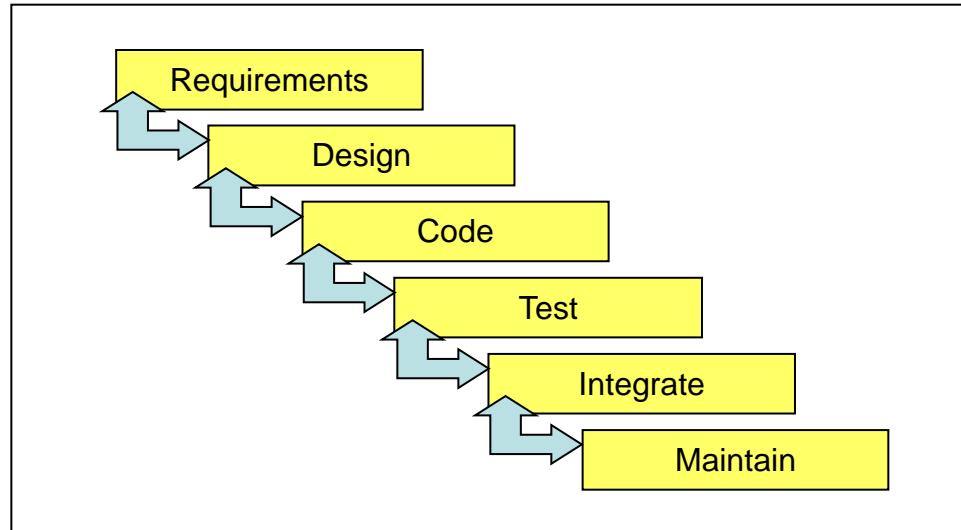
- Begin with the unconditional recognition that software development, is of its very essence a human activity: asynchronous, spontaneous, autonomous - the most difficult thing we can construct
- Recognise that we must build communication, motivation, trust and all those things that make good teams & partners

Into the white space

- Understand that software development is highly situational
 - it's always unique: you can't walk in the same river twice
- Truly recognise that we are optimists and we still believe people and time are interchangeable – they aren't
- Understand that the one size fits all mentality we have for methodological development may be useful but it is seriously flawed

Into the white space

Above all we must understand that there is just as much, if not more, happening in the white space between & around the boxes in our carefully crafted process, architectural, and organisational models as goes on within them



Into the white space

It's in the white space where:



- misunderstandings arise
- important information is lost
- verbal promises get broken or forgotten
- things don't get documented or recorded
- disastrous arbitrary decisions are made
- project scope is allowed to creep
- and stupidity & optimism run rampant

We either learn from past mistakes or else we'll continue to screw up!

Remember

“Those who cannot remember the past are condemned to repeat it” Santayana