



# Introduction to Software Engineering

(Software Engineering I)

Software Engineering methods, Software development life-cycle (SDLC) basics, quality aspects, RAD, Formal specifications, Testing basics, Validation & verification.

This document © 2003-2008 Dr. Ernest Cachia

Ernest Cachia  
Department of Computer Science

Faculty of ICT



The intention of this study unit is to introduce students to the important aspects of software engineering approaches. Aspects like the various development paradigms – traditional and emerging. Issues pertaining to quality and soundness of software will be discussed. This unit will also introduce some issues relating to formal specification, system testing and verification and validation.

The study unit will deal with both the human and the technical dimension of software development.



### General Refresher and Introduction

This should serve as a general reminder of what you SHOULD already be well aware of as well as informing you of what lies ahead. However, the road-map for this study unit is high-level and practicality necessitates that I reserve the right to change it within limits and as needs dictate.



- Quick refresh
- About Software Engineering
- System outline
- The software crisis



## Software Representation, remember?

- Abstract notion of an algorithm
- Models to represent algorithms
- Models generally categorised as:
  - Diagrammatic
  - Textual
  - Formal
- Based on system aspects:
  - Behavioural
  - Data
  - Dynamic



## What is Software Engineering (SE)

- **Maybe you'll get a better idea after attending this course**
- In very short and using generic terms...
  - It is a way of tackling software development of considerable industrial and commercial import
  - It is not a definite tangible method
  - It is not old or new-school
  - It is not a fixed set of procedures
  - It is not the same exact activities for every organisation
- It is entirely based on historical, experiential, practical and intuitive concepts – i.e. common sense
- Useless to try to follow blindly



## Some Fundamental Points to Consider

- SE only effects software development.
- SE is not one specific method or approach but rather a way of disciplining one's actions.
- SE is not a panacea.
- SE cannot be applied blindly and can actually hinder the development process if not used in the right way.
- SE does incur substantial overheads in terms of human and organisational effort.
- SE practices have been proven valid in many large-scale projects (*in both state and private sectors*).
- Sensible application of SE practices requires formal tuition and practical experience.



## Why Software Engineering?

Maybe the answer can be deduced from some definitions of SE on the next slide...



## Some Definitions

- **Generic definition:** *“The building of software systems”* (coined in the 1960s)
- **D. L. Parnas:** *“The multi-person construction of multi-version software”* (conference 1987)

Software engineering *includes* programming but is not programming. Therefore, good programmers *are not necessarily* good software engineers!



## Fundamental SE Goals

- The final goal of SE has to be to build software systems (maybe even whole solutions) of demonstrable high quality.
- In the light of modern software systems, to attain such high aims SE must amongst other things, be able to effectively bring to focus the efforts of more than one individual (*more on this in the third year of your studies*)



## A Few More SE Aspirations

- Introduces elements of rigorous discipline to the software development process.
- Attempts to define software development techniques and guidelines which can then be standardised.
- Facilitates some of the more mechanical parts of software development.
- Strives to use software models which are portable over a wide range of environments/platforms.
- Offer a possible entry point and “common ground” for machine automation of software system development.

**So, what exactly is a software system? ...**



## A Software System

- In general, a modern ICT system is one involving co-operating software and hardware providing a solution for a specific (business or otherwise) problem.
- Therefore, a software system can be viewed as a software solution to a specific real-world problem.
  - The focus of attention and effort.
  - Generally viewed as one of the main product components of ICT development.
- So, what are the characteristics of a typical software system? ...





## An Unsustainable Situation

- **On one hand...**
  - People build software systems using no particular method
  - People build fragile software systems
  - People build inaccurate software systems
  - Software system development is expensive
  - Software system development requires considerable planning and effort
- **On the other hand...**
  - Modern software systems are ever-increasing in sophistication
  - Demands on software systems is always rising
  - Software systems are what make a computing entity
  - People consciously or unconsciously rely on software for most of their social activities



## The Software Crisis

- The tension created by the conflicting interests outlined in the previous slide gave rise to a software crisis peaking in the late 70s / early 80s.
- The software crisis threatened to cripple the progress of computing as a whole.
- Effects of the crisis still linger in the form of software development always playing “catch-up” to hardware development.



## An “unhealthy” state of affairs

- **Monolithic development is not effective for modern system development.**
    - No process control
    - No product or process guarantees
    - No true management
    - No client confidence
    - No process visibility / traceability
    - No metrication
    - No communication
- => no quality!**



## The Software Crisis

Tools



+ Methods





## Summary (Session 1)

- What is SE? What is its scope? What is a system?
- Factors leading to and influencing the software crisis.
- What needs to be done.



## Session 2

### Software Quality

This should serve as a good introduction to the notion of quality in general and in particular software quality expressed as a set of stated attributes. We will also look at the attainment of quality in view of the various system types generally encountered.



## Session Aims

The main aim of this session is to introduce you to the notion of quality in software process and product. The appreciation of a scientific approach to quality understanding through the determination of distinguishing criteria and their classification

- To instil the need to have a clear understanding of what quality means in terms of software development
- To analyse software development in terms of its quality aspects
- To introduce the various types of product (systems)
- To present and outline a generic quality profile.



## Session Contents

- Definition of quality
- Software quality
- Types of system
- Quality attributes



## The Meaning of Quality (Is there one?)

### In general:

“A distinctive attribute or characteristic”...“the degree of excellence of something as measured against other similar things” – *Oxford Online*.

“A peculiar and essential character”...“An intelligible feature by which a thing may be identified” – *Merriam Webster Online*.

### Software quality:

“Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software” – *Pressman, R.*



## Software Quality

### Software Quality Considerations

- Product (solution) quality is dependent on process quality
- Quality control must be an ongoing process
- Quality is a direct result of good (i.e. correct) management
- Quality depends on the type of system in question
- Quality can be viewed as an internal or external property



## Attempting to qualify software

### **This is not a straightforward issue due to such facts as:**

- Very difficult if approached un-scientifically
- Akin to qualifying human thought/reasoning
- Various factors effect it (i.e. must be considered)
- There are multiple facets to a s/w product
- Very easy to qualify “incorrectly”
- There is considerable subjectivity in a s/w product
- A s/w product may be prone to changes



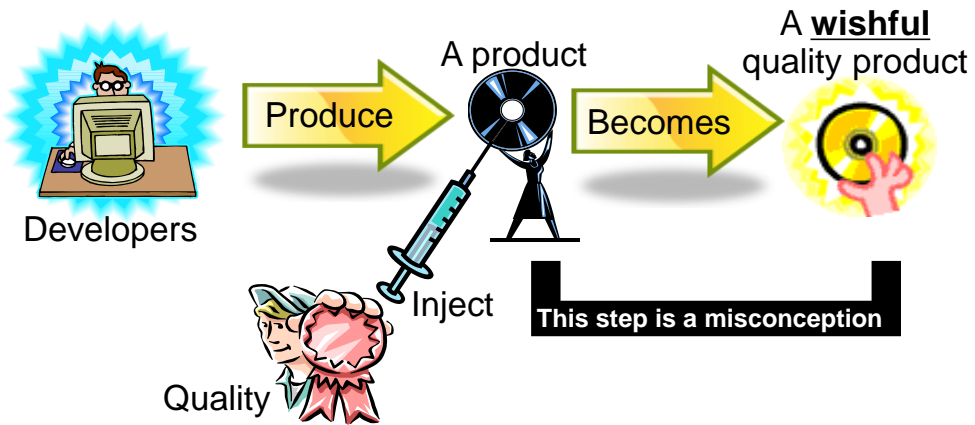
## Some Generic Software Aspects

### **Software quality can be characterised by its:**

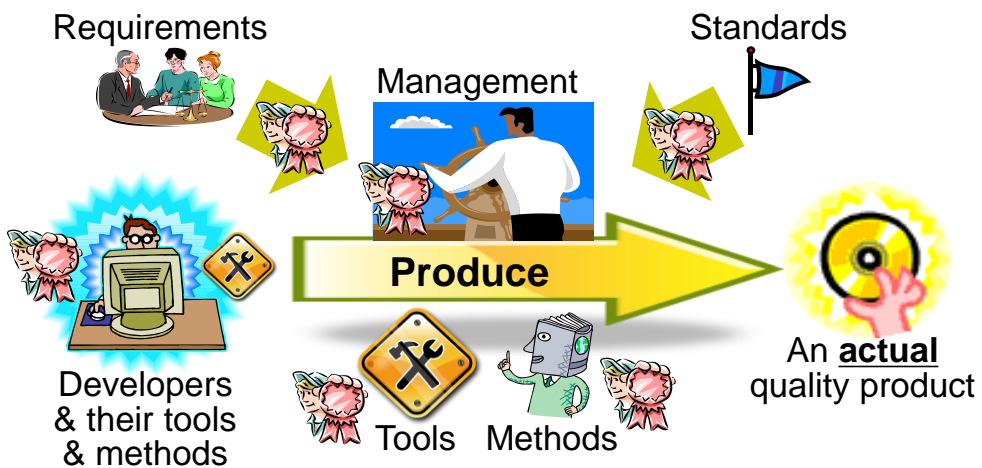
- The development process
- End-products (with all their representative attributes)
- Interaction with humans (users, developers, process managers, vendors, etc.)
- Relation to the real-world process it is to implement
- Reviews and/or feedback (relating to the product).



# Quality cannot be an “add-on”



# Quality Must Permeate





## The Actual Meaning of Quality

- **Quality means different things to different people. Consider the following:**
  - **For the user** (reliable, efficient, user-friendly, etc.)
  - **For the producer** (maintainable, verifiable, portable, etc.)
  - **For the manager** (measurable, visible, controllable, etc.)
  - **For the client** (cost-effective, interoperable, effective, etc)
  - **For the vendor** (value-for-money, relevant, timely, etc)
- **The main two categories of s/w qualities are:**
  - **External** (qualities observable from “outside” the system)
  - **Internal** (qualities pertaining to internal system aspects)



## The Process Makes the Product

- **Process:** “A series of activities undertaken to achieve a stipulated entity”
- **Product:** “An entity resulting from a given process”
- Quality applies to both process and product
- A software product can (typically) consist of:
  - The executable system
  - Installation guides
  - User manuals
  - Application documentation
  - User training plan/requirements
  - System cross-over and data migration strategies
  - Testing justification and logs
  - Etc.



### A general system classification could be as follows:

- Information/Business (Batch-Processing)
- On-line
- Real-time
- Embedded
- Distributed

Quality for each of the above can have a different meaning.

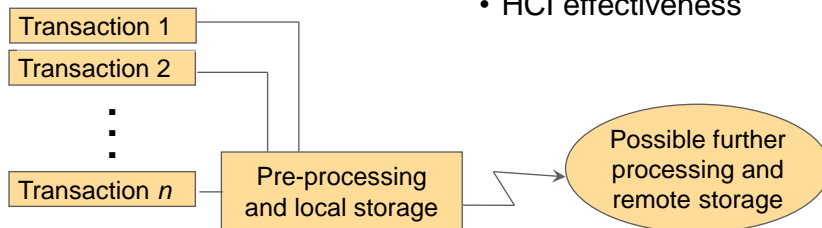


### • Main elements:

- Data
- Database
- Transaction
- Security

### • Important aspects:

- Data integrity
- Data availability
- Data security
- Transaction efficiency
- HCI effectiveness





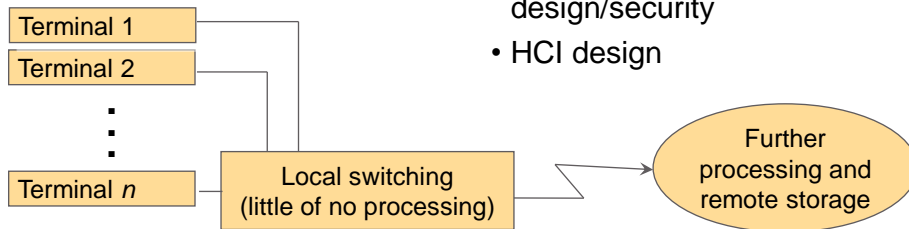
## On-line Systems

- **Main elements:**

- Result time limits
- Time-slicing
- Security

- **Important aspects:**

- Response time range
- Stimuli to results relationships
- Communication design/security
- HCI design



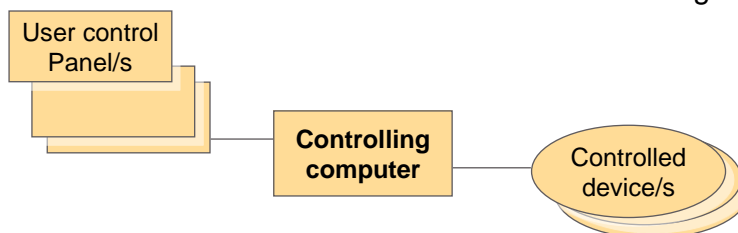
## Real-Time Systems

- **Main elements:**

- Stringent timing
- Control
- I/O specifications
- Safety

- **Important aspects:**

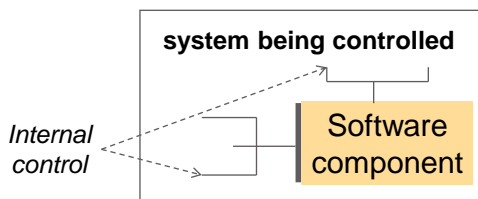
- Response time
- System size & complexity
- Control protocol design
- Safety mechanisms
- HCI and MMI design





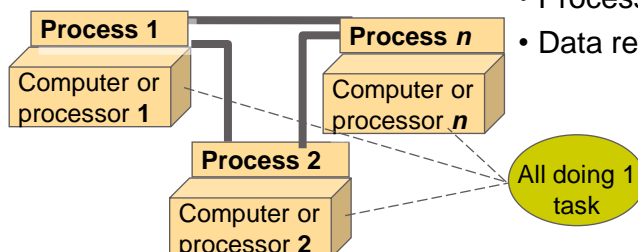
## Embedded Systems

- **Main elements:**
  - Stringent timing
  - Control
  - I/O specifications
  - System dependency
  - Safety
- **Important aspects:**
  - Response time
  - System size & complexity
  - Control protocol design
  - Safety mechanisms
  - Sensory feedback loops
  - MMI design



## Distributed Systems

- **Main elements:**
  - Process communication
  - Process distribution
  - Data distribution
  - Network links
- **Important aspects:**
  - Communication protocols & bandwidth
  - Logical to physical process (and data) mapping
  - Process dependency & sync.
  - Data redundancy





## Some Software Process & Product Quality Attributes

- Correctness (*int/prd*)
- Security (*ext/prd*)
- Reliability (*ext/prd/prc*)
- Robustness (*ext/prd/prc*)
- Efficiency (*ext/prd/prc*)
- User friendliness (*ext/prd*)
- Verifiability (*int/prd*)
- Maintainability (*int/prd/prc*)
- Reusability (*int/prd*)
- Portability (*ext/prd*)
- Understandability (*int/prd*)
- Interoperability (*ext/prd*)
- Productivity (*prc*)
- Timeliness (*prc*)
- Visibility (*prc*)
- Manageability (*prc*)

*“ext” - external quality; “int” - internal quality  
“prd” - product quality; “prc” - process quality*



## Briefly on Each Attribute (1/3)

- **Correctness**  
Does the system do the right thing?
- **Security**  
Can the system be trusted?
- **Reliability**  
Is the system dependable? Is its production consistently acceptable?
- **Robustness**  
Does the system have a margin of tolerance? Is its production flexible?
- **Efficiency**  
Is the system helpful? Is its production well controlled?
- **User friendliness**  
Is the system pleasant and conducive to use?



## Briefly on Each Attribute (2/3)

- **Verifiability**  
Is the system easy to verify in correctness and in required function?
- **Maintainability**  
Is the system easy to alter and/or upgrade? Is its production adaptable?
- **Reusability**  
Has the system been built based on past experience/effort?
- **Portability**  
Can the system work on other computing platforms/environments?
- **Understandability**  
Is the way the system was built and its logic easy to understand?



## Briefly on Each Attribute (3/3)

- **Interoperability**  
Can the system work well in conjunction with other systems?
- **Productivity**  
How stabilised and competitive is the production?
- **Timeliness**  
How mature and controlled is the production to ensure timely delivery?
- **Visibility (prc)**  
How clearly is the production life-cycle structured?
- **Manageability (prc)**  
How well defined are all the stages and meta-products in the life-cycle?



## Activity “CSA2170-A”

For this activity you are to read up on **McCall’s Quality Factors** and compare these to the quality factors according to the **ISO9126 standard**. Give a (approximately one-page) conclusion **in your own words** based on what you discover.

You should compare the above on the basis of:

- 1) coverage;
- 2) Relevance;
- 3) practical applicability.



## Summary (Session 2)

- What is quality? How can it be gauged? Where should it be considered?
- The quality of software and how this relates to the various types of system.
- An overview of the various quality attributes relating to software product and process.



### Phased Development and Life-Cycles

This session should provide an clear insight into what phased software development is, what it aims to achieve, and how it achieves it. Furthermore, various “prominent” and representative software development life-cycles (SDLCs) are introduced in preparation for further discussion.



The scope of this session is to instil in you an appreciation for the need of fragmenting the development process and precisely defining each part. You will also learn to understand the composition and application of various SDLCs. The concepts behind the use of prototypes will also be tackled.



## Session Contents

- Phased development
- Milestones
- Software development life-cycles
- Prototyping

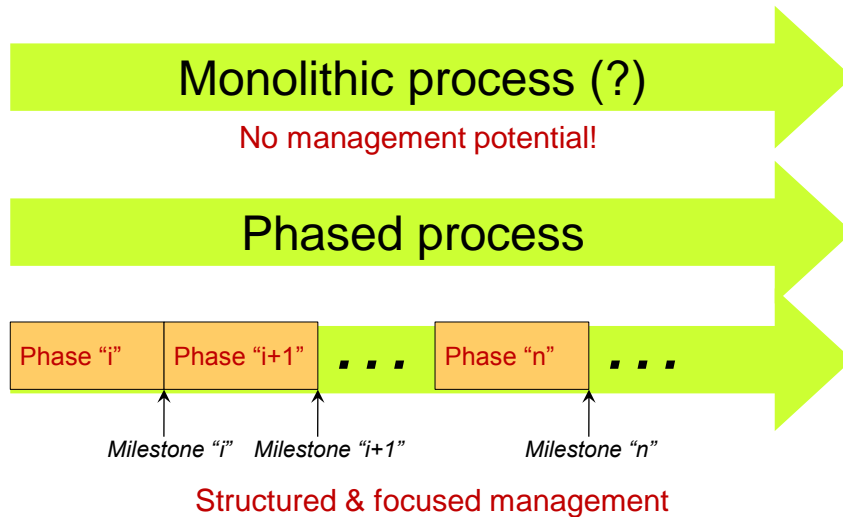


## Breaking the Monolithic Model

- Done by introducing “steps” into the software development process.
- Steps in the development process are called “phases” (or “stages”).
- Phases must be self contained and pre-defined.
- Phases should decrease abstraction as they progress.



## Phased Development



## A Software Development Phase

### A software development phase:

- Is a **delimited period** of time within the process of development of a software system.
- Has a **definite starting set of data** and a **definite set of results**.
- Is based on the results of earlier phases.



## Advantages of Phased Development

### Phased development

- Offers benchmarking
- Offers insight
- Offers mile-stoning niches
- Offers a documentation-building framework
- Offers a definite progression sequence
- Offers possibilities for prototyping
- Allows end-user and client participation
- Offers possibilities for better testing strategies

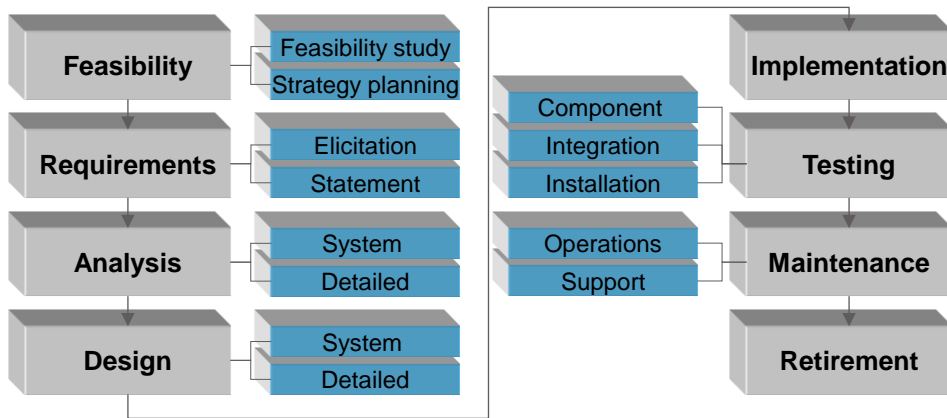


## A Development Milestone

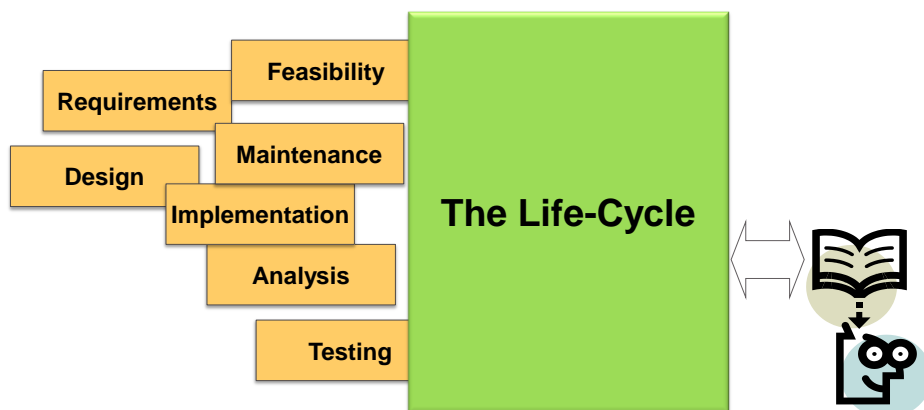
- **A software development milestone is a scheduled event...**
  - for which some project member or manager is accountable.
  - is used to measure progress.
- **A milestone typically includes:**
  - A formal review.
  - The issuance of documents.
  - The delivery of a (sometimes intermediate) product.



## Typical Software Development Phases



## Phases are Neutral and Composable



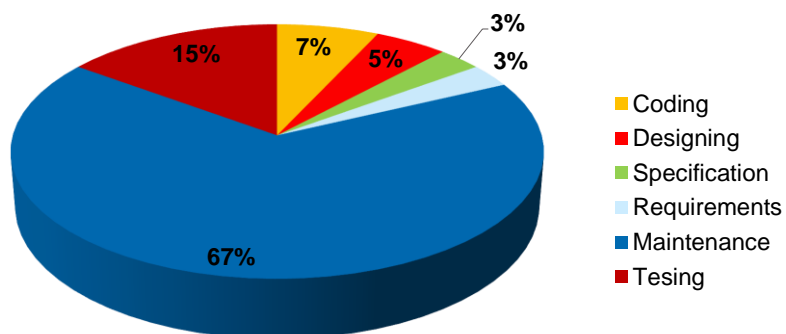


## Activities During Phases

- **Requirements:** establish the customer's needs
- **System Design:** develop the system's structure
- **Detailed Design:** develop module structures
- **Implementation:** code or otherwise
- **Testing:** check what's been developed
- **Installation:** bring the system into production
- **Maintenance:** correct, adapt, improve



## Traditional software development effort spread





## The Development Life-Cycle

*(Also known as the Software Development Process)*

- A project is a set of activities, interactions and results.
- A life-cycle or a software process is the organisational framework for a project.



## Life Cycle *(ANSI / IEEE Standard 729-1983)*

- **A life-cycle...**
  - is a finite and definite period of time.
  - starts when a software product is conceived.
  - ends when the product is no longer available or effective for use.
- **Any life-cycle is organised in (composed of) phases**



## Types of SDLC

- More than one possible life cycle can be adopted to attain a particular goal.
- The type of SDLC is defined by the way it links its component development phases.
- In theory, any combination of phases is possible, however, in practice only ones that lead to a visible and controllable development process are useful.
- A type of SDLC is also known as a Development Model.



## The Nature of an Effective DM

### **An effective DM is one that:**

- Effectively links the phases it includes
- Focuses phases towards a definite goal
- Provides mechanisms for the controlled decrease of system abstraction
- Includes definite milestones
- Is transparent
- Is traceable between adjacent phases



## Some Representative Life-Cycles

**Development model definition (personal):** A particular interaction configuration of development phases leading to a final software product.

- Waterfall (and Enhanced Waterfall)
- V-model
- Evolutionary Prototyping (aka Incremental)
- Throw-away Prototyping (aka Rapid)
- Rapid Application Development (RAD)
- Spiral
- Reuse-oriented
- Formal (a.k.a. Transformational)

*These will be outlined in the next slides.*



## The Waterfall Model

- Is the root of all other models
- Still prevalent in general
- Exists in many versions
- Supported by many methods/techniques

Has many drawbacks!



## Fundamental Waterfall Model Properties

- **Sequential nature**
  - All phases must happen in a predefined sequence.
- **Transformability**
  - Any given phase can be directly attributed to the preceding one.
- **Completeness**
  - Any given phase must fully completed before the next phase can be started.

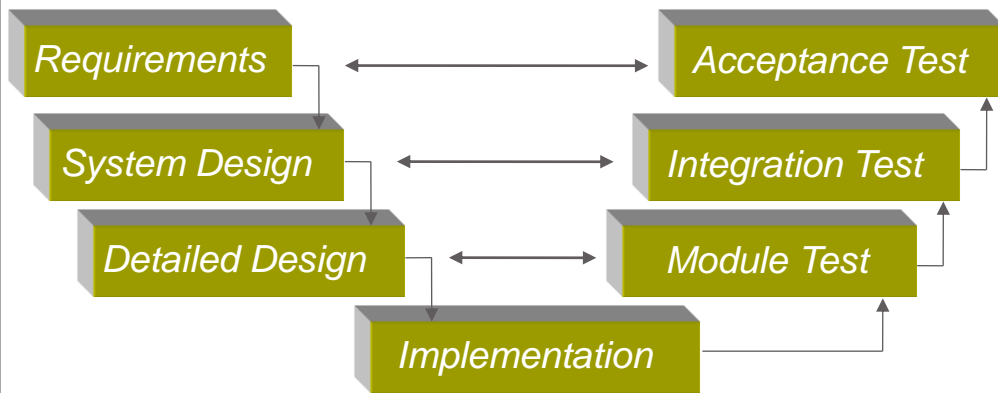


## Waterfall Model Drawbacks

- Sequential nature
- Late tangible product maturity
  - Late feedback to both customer and developer
  - Minimal risk management for both customer and developer
- Late testing accuracy
- Late client acceptance confirmation
- Inflexible
- Can be overly time-consuming



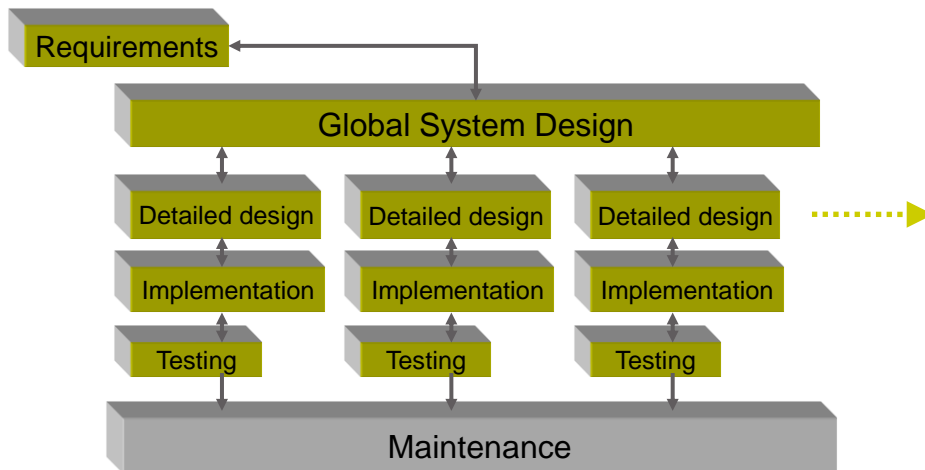
## The V-Model



## Analysis of the V-Model

- Improves testing strategies
- Does not particularly improve:
  - Sequential nature
  - Feedback
  - Developmental risk management

## The Incremental Model



## Analysis of The Incremental Model

- Assumes independent sub-systems.
- Improves (by delivering smaller units):
  - Feedback (in steps)
  - Testing
- Avoids the production of a monolithic product.
- Does not particularly improve:
  - Developmental risk management
  - Sequential nature (still present in sub-systems)



## The Prototyping Model

### Goals:

- to break away from the sequential nature.
- to speed up feedback.
- to minimise risks (*for both customer and developer*)
- to be incomplete but executable.
- to be cheap and fast.



## What is Prototyping?

### A definition (*A. Davis*):

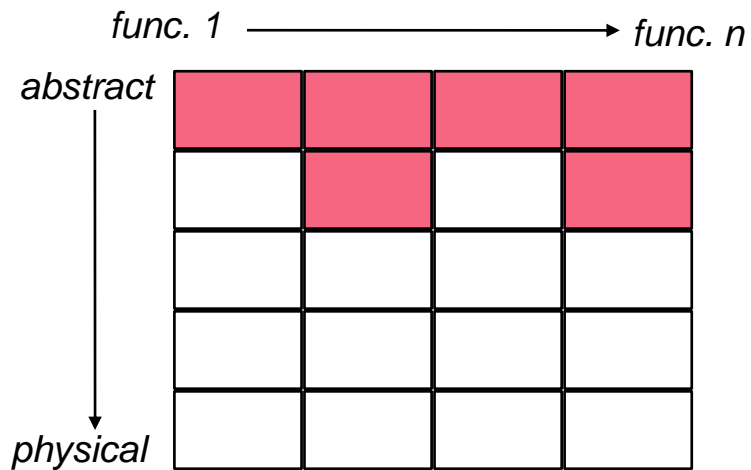
A prototype is a partial implementation of a system, constructed primarily to enable customer, end-user, or developer to learn about the problem and/or its solution.

### Types:

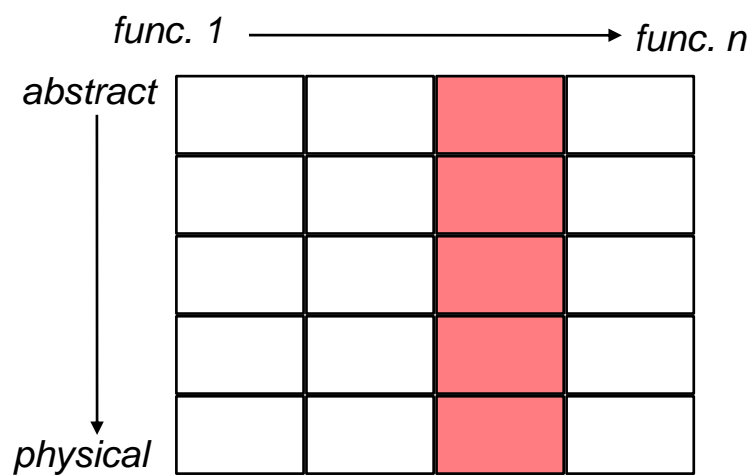
- evolutionary / throw-away
- horizontal / vertical



## Horizontal Prototyping

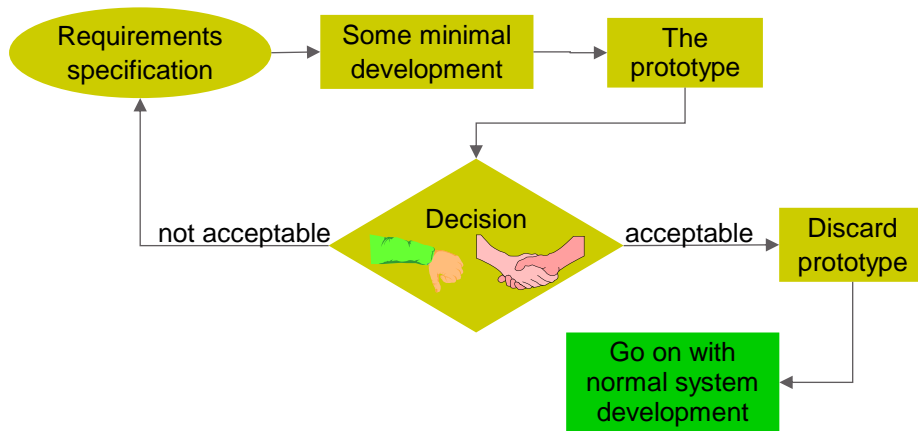


## Vertical Prototyping

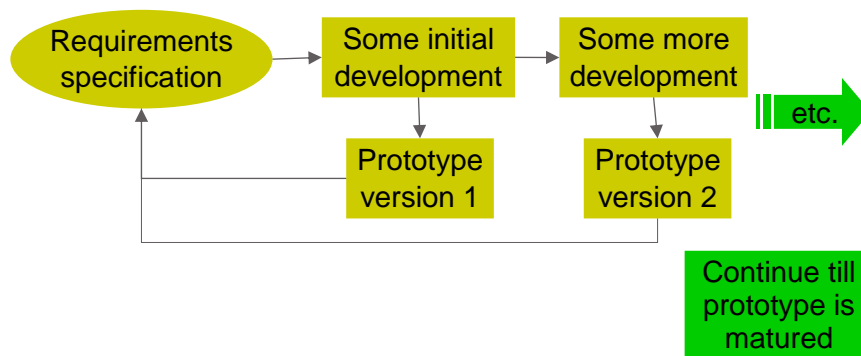




## Throwaway Prototyping Visual

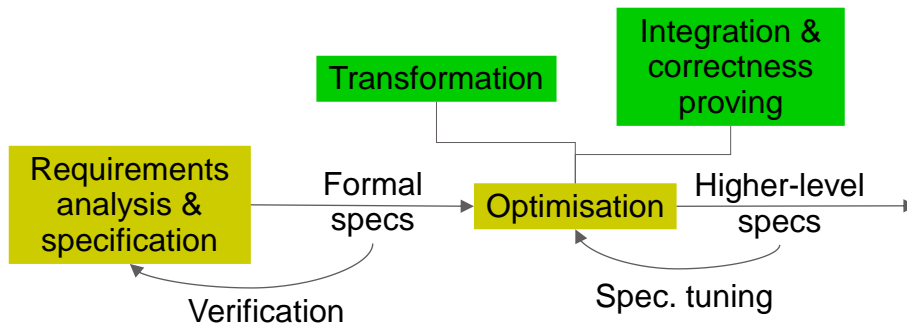


## Evolutionary Prototyping Visual





## Formal (Transformational) Model



## Analysis of The Prototyping Model

- Improves:
  - breaks the sequential nature.
  - supports fast feedback.
  - offers an opportunity for risk management.
- Problem:
  - has no definite (i.e. strictly defined) organisational structure.

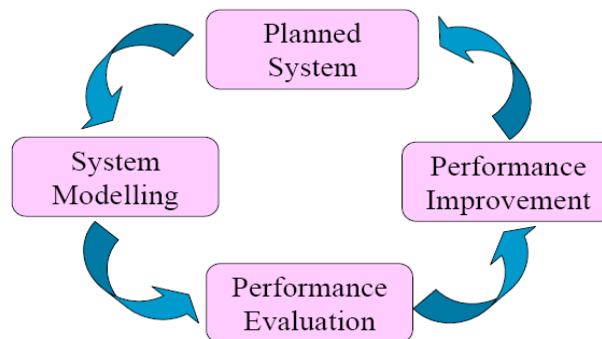


## Characteristics of the Spiral Model

- Placeholder (“framework” or “meta-model”) for other, less elaborate, development models
- Iterative by nature
- Prototype-oriented
- Starts with planning and ends with customer evaluation
- Lowers risk (by implicitly enforcing risk analysis)



## Basic Concepts of Spiral Development



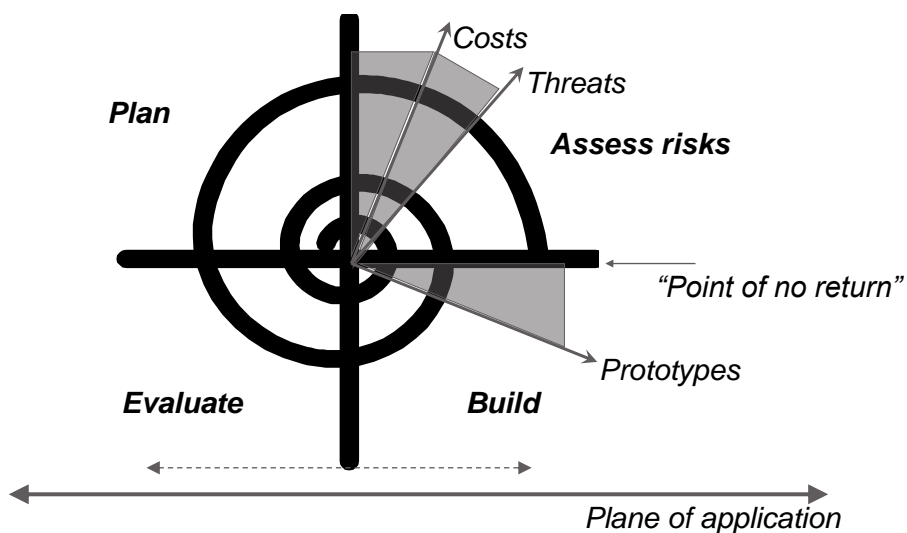


## Spiral “sections”

- **Planning**
  - Getting requirements
  - Project planning (based on initial reqs.)
  - Project planning (based on customer eval.)
- **Risk analysis**
  - Cost/Benefit and threats/opportunities analysis
  - Based on initial reqs. and later on customer feedback
- **Engineering**
  - Preview it
  - Do it
- **Customer evaluation**

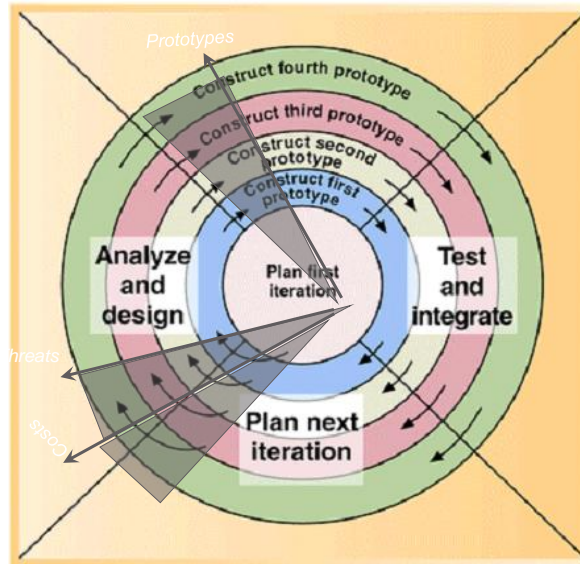


## Diagrammatically said...





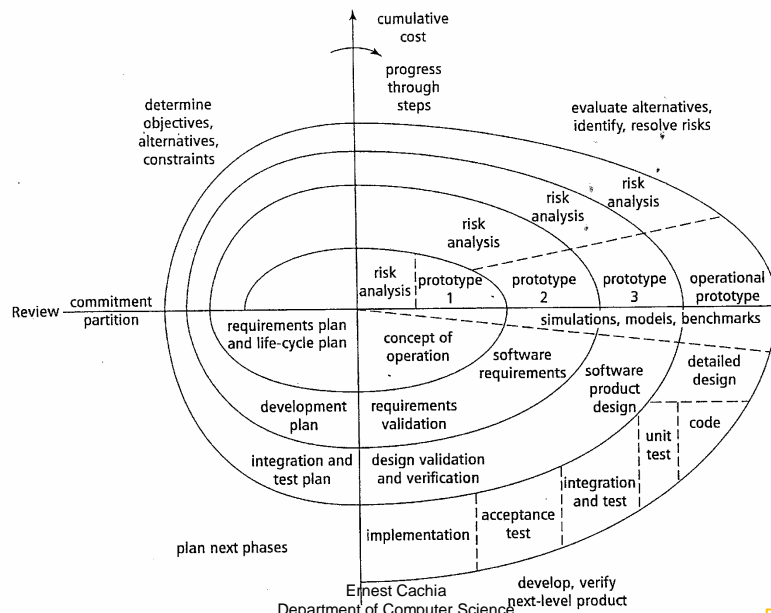
# Another view



By Prof. Peter Khaiter



# Details of the Spiral SDLC





## The Spiral's Steps

- 1) Define requirements
  - Through user involvement and analysis of existing system
- 2) Initial new system design
- 3) Construct and evaluate an initial prototype
  - Rough (skeletal) system attribute framework
- 4) Construct a further (refined) prototype
  - Basing it on evaluation of initial prototype
  - Defining its scope
  - Planning its development
  - Implementing it
- 5) Overall (system-wide) risk assessment
- 6) Prototype assessment (as per step 4) and possible development of further prototypes
- 7) Repeat steps 1-5 until refined prototype meets user expectations
- 8) Construct the system (based on final refined prototype)
- 9) Test and maintain the system



## The "Win-Win" variant of the Spiral

Simply put...

Make everyone involved happy and you're practically guaranteed project success.

How can you try and do that?

Make sure every phase in the spiral starts off with:

- Understanding who "everyone" is
- Understanding what everyone wants
- Reconciling everyone's needs



## Some Points to Ponder

1. What is Software Engineering?
2. What are the attributes of a good software?
3. What are the key challenges facing software engineering today?
4. Give examples (one in each case) of physical and abstract systems.
5. Explain why, in terms of modelling, physical and abstract systems are not differentiated.
6. Outline the role of documentation as a system component.
7. Why is communication so vital in software development? Give two concrete examples of what can happen in the case of communication breakdown.



## Some *More* Points to Ponder

1. Differentiate between a software development life-cycle and a software development model.
2. List the main advantages and disadvantages of the Waterfall Development Model.
3. Why are most modern systems built according to one of the prototyping models?
4. Why do you think formal development models are not as popular as other less formal ones?
5. Can you propose a way to improve risk management in software development?



## Summary (Session 3)

- Development broken down into well defined phases, with milestones and clear I/O
- The nature of SDLCs as collections of phases linked together in specific ways
- SDLC advantages and drawbacks – making them suitable for particular types of system development.
- An overview of the various SDLCs.