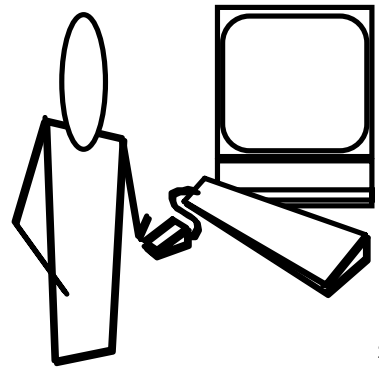
The Software Development Process

What is Software?

- Software is a set of items or objects that form a "configuration" that includes
  - Programs
  - Documents
  - Data ...



Definition of Software

- Computer programs and associated documentation such as requirements, design models and user manuals.
- Software products may be developed for a particular customer or may be developed for a general market.
- Software products may be
  - **Generic** – developed to be sold to a range of different customers e.g. PC software such as Excel or Word.
  - **Bespoke** (custom) – developed for a single customer according to their specification.
- New software can be created by developing new programs, configuring generic software systems or reusing existing software.

Why Software?

- The primary purpose of software is that of information transformer.
- Software is used to produce, manage, acquire, modify, display, and transmit information.

Software

- Software is **developed**, **NOT manufactured**.
  - Building a software product is more like constructing a design prototype.
  - Software may become deteriorate, but it does not wear out.
    - The chief reason for software deterioration is that many changes are made to a software product over its lifetime. As changes are made, defects may be inadvertently introduced to other portions of the software that interact with the portion that was changed.

Characteristics of Software

- Software has a dual role. It is a product, but also a vehicle for delivering a product.
- Software is a logical rather than a physical system element.
- Software has characteristics that differ considerably from those of hardware.
- Software is developed or engineered, it is not manufactured in the classical sense.
- Software doesn't "wear out".
- Most software is custom-built, rather than being assembled from existing components.

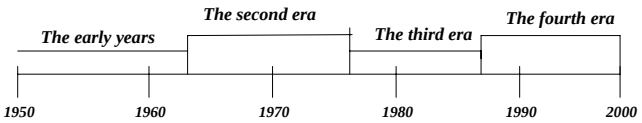
Evolution of Software

The Early Years
- Batch orientation
- Limited distribution
- Custom software

The Third Era
- Distributed systems
- Embedded "intelligence"
- Low cost hardware
- Consumer impact

The Second Era
- Multiuser
- Real-time
- Database
- Product software

The Fourth Era:
- Powerful desk-top systems
- Object-oriented technologies
- Expert systems
- Artificial neural networks
- Parallel computing
- Network computers



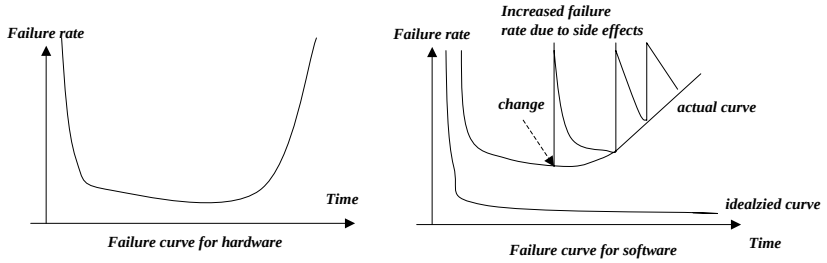
Software Applications

- **System Software** – A collection of programs written to service other programs at system level. (E.g. compiler, operating systems)
- **Real-time Software** – Programs that monitor/analyze/control real world events as they occur.
- **Business Software** – Programs that access, analyze and process business information.
- **Engineering and Scientific Software** – Software using "number crunching" algorithms for different science and applications. (E.g. System simulation, computer-aided design)
- **Embedded Software –** Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets. It has very limited and esoteric functions and control capability.
- **Artificial Intelligence (AI) Software** – Programs make use of AI techniques and methods to solve complex problems. Active areas are expert systems, pattern recognition, games
- **Internet Software** – Programs that support internet accesses and applications. (E.g. search engine, browser, e-commerce software, authoring tools)
- **Software Tools and CASE environment** – Tools and programs that help the construction of application software and systems. (E.g. test tools, version control tools)

Software Crisis

- In the software industry, we have had a "crisis" that has been with us for close to 30 years. Meaning of the word "crisis":
  - A turning point in the course of anything; decisive or crucial time, stage or event.
  - The turning point in the course of a disease, when it becomes clear whether the patient will live or die.
- What we actually have in software industry is a **Chronic Affliction**.
  - It means **lasting a long time, recurring often, continuing indefinitely**.
- Software crisis or software affliction
  - A set of problems encountered in software production.
    - Problems in developing software
    - Problems in maintain a growing volume of existing software
- Typical examples:
  - Build a wrong product, Project schedule problems, Cost estimation problems

Failure Curve for Software/Hardware



Software Myths – Management Myths

- Many causes of a software affliction can be traced to a mythology that arose during the early history of software development.
  - Software myths propagated misinformation and confusions. They had a number of attributes that made them insidious.
  - Software managers often under pressure to maintain budgets, keep schedules from slipping, and improve quality.
- Myths ➔ Misleading attitudes of people ➔ Serious problems in software production
- Management Myths:
  - We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?
  - My people do have state-of-the-art software development tools. After all, we but them the newest computers.
  - If we get behind schedule, we can add more programmers and catch up.

Software Myths – Customer Myths

- Customers of a software may be:
  - An outside company that has requested software under contract
  - A person next to your desk
  - An in-house group
  - A marketing or sales group
- Customer myths ➔ Lead to false expectations (by customers) and ultimately, dissatisfaction with the developers.
- Customer Myths:
  - A general statement of objectives is sufficient to begin writing programs - we can fill in the details later.
  - Project requirements continually change, but change can be easily accommodated because software is flexible.

Software Myths – Practitioner Myths

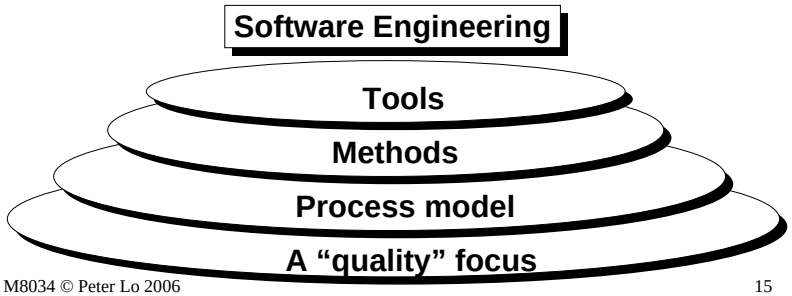
- Practitioners:
  - Planning group - System analysts, system architects
  - Development group - Software engineers
  - Verification group - Test engineers, quality assurance group
  - Support group - Customer supporters, technical supports
  - Marketing/sales - Marketing people and product sales
- Practitioner's Myths:
  - Once we write the program and get it to work, our job is done.
  - Until I get the program 'running,' I really have no way of assessing its quality.
  - The only deliverable for a successful project is the working program.
  - The major task of a software engineer is to write a program.
  - Schedule and requirements are the only important things we should concern when we write programs.

What is Software Engineering?

- Although hundreds of authors have developed personal definitions of software engineering, a definition proposed by Fritz Bauer[NAU69] provides a basis:
  - Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.
- The IEEE [IEE93] has developed a more comprehensive definition when it states:
  - Software Engineering:
    1. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
    2. The study of approaches as in (1).

Key Elements of Software Engineering

- Four key elements of Software Engineering:
  - Method, Tools, Procedure/Process Model, A Quality focus



Methods

- Refer to techniques on how-to build software and how to encompass a broad array of tasks such as requirements analysis, design, coding, testing, and maintenance
- Software engineering methods rely on a set of basic principles
- Examples:
  - Project Planning
  - System and Software Requirement Analysis
  - Coding, Testing and Maintenance

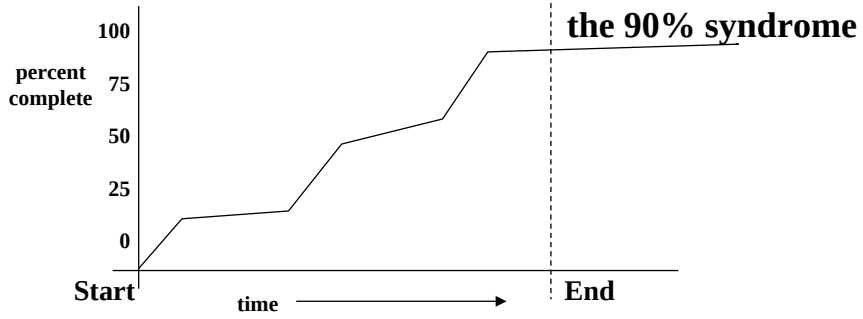
Tools

- Provide automated or semi-automated support for the process and methods.
- Support engineers to perform their tasks in a systematic and/or automatic manner.
- Example:
  - CASE (Computer-Aided Software Engineering)

Process Model / Procedures

- Defines the sequence in which methods are applied, and makes sure that the development of software is logical and on time. It is the glue that holds:
  - Technology together
  - Enables rational and timely development of computer software
- Software engineering process is a framework of a set of key process areas. It forms a basis for:
  - Project management, budget and schedule control
  - Applications of technical methods
  - Product quality control

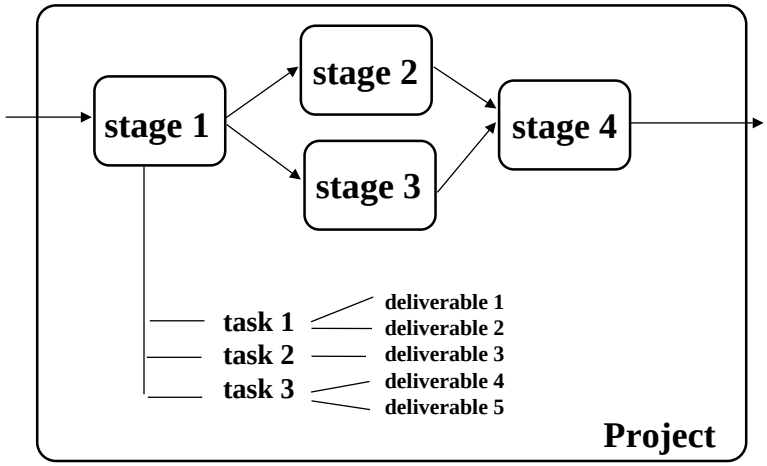
Unstructured Development



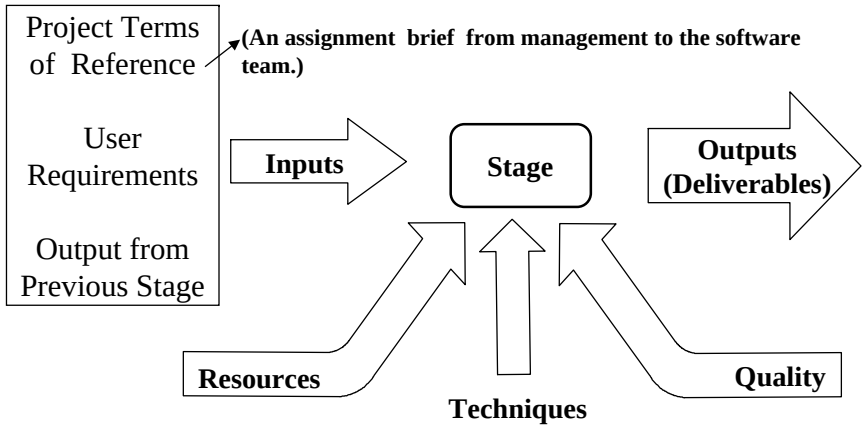
- **Quick start to coding**
- **Concurrent analysis, design, coding**

- **Lack of stages**
- **Lack of control**
- **90% syndrome**

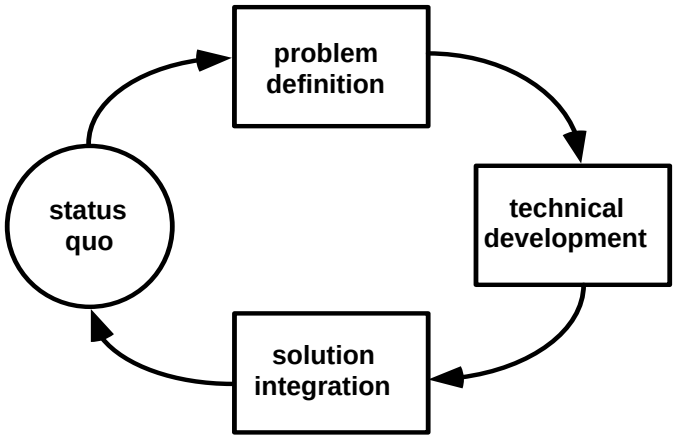
Structured Development



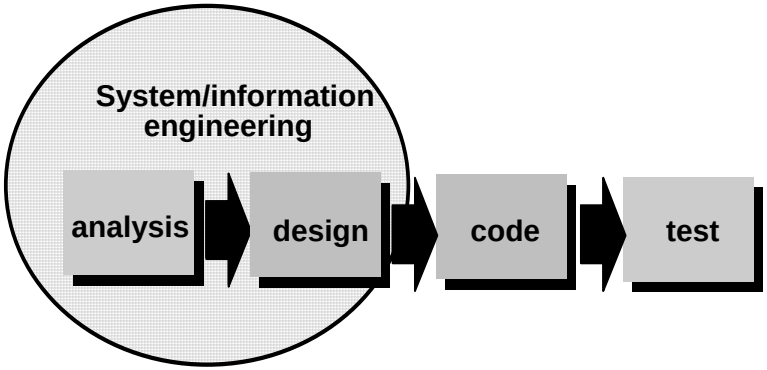
Development Stages



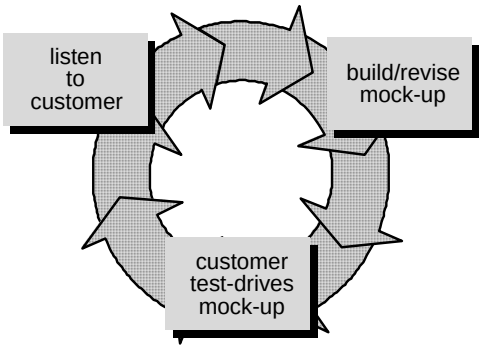
Process as Problem Solving



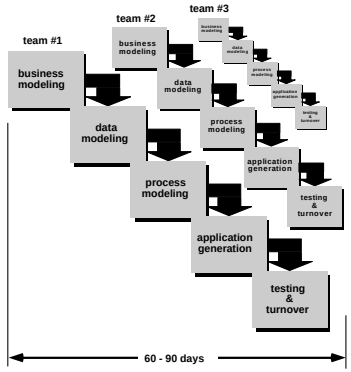
The Linear Model



Iterative Models

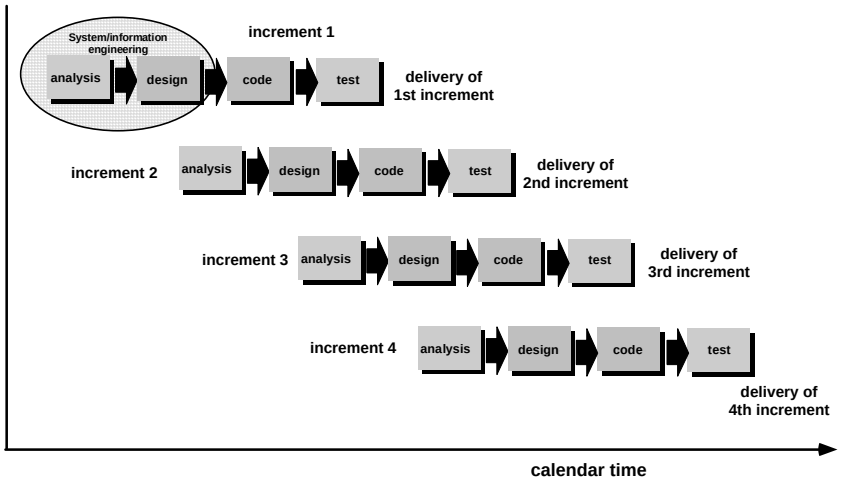


Prototyping

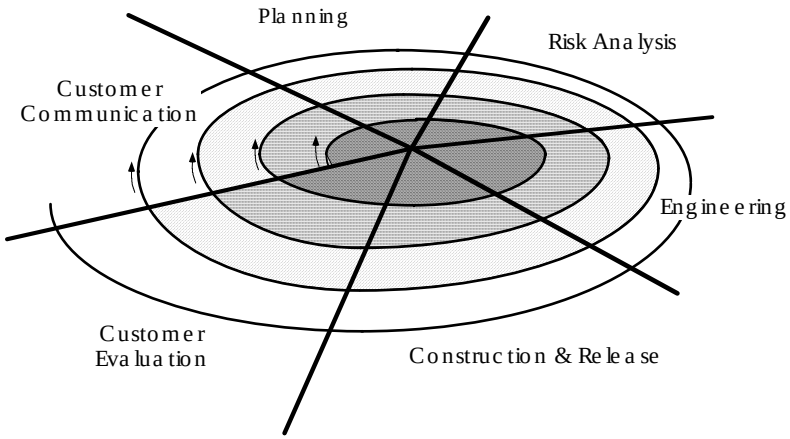


RAD

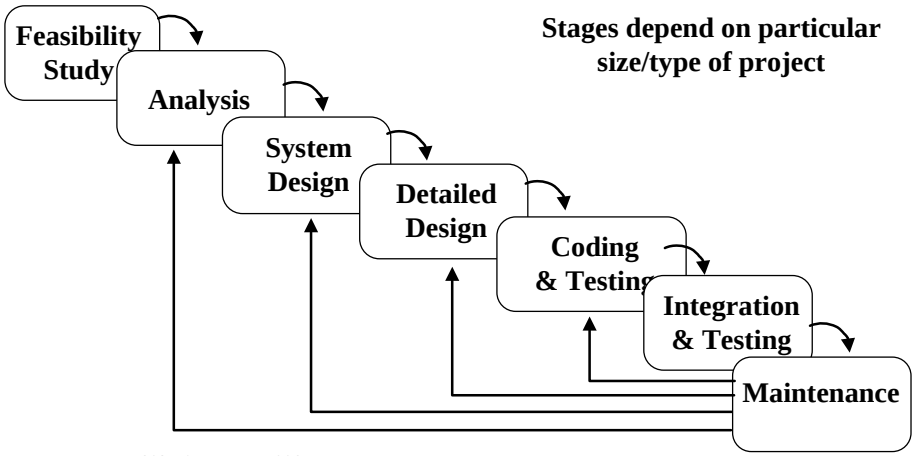
The Incremental Model



An Evolutionary (Spiral) Model



The Waterfall Model (The Linear Sequential Model)



The Waterfall Model

- The Waterfall model (so called because the stages appear to flow down into each other) is one of the most widely documented models of software development.
- The diagram shows some small overlap between the stages (analysis can start before the feasibility stage is entirely completed) but it is a broadly linear process with, for example, all analysis being completed before any design is done.
- The arrows flowing ‘up’ the waterfall are a recognition of the fact that in course of maintenance it may be necessary to repeat some of all of the process.

General Stage in the Waterfall Model

- Feasibility Study
- Analysis
- Design
- Code generation
- Testing
- Maintenance

Number of stages depend on particular size/type of project!

Stage in the Waterfall Model - Feasibility Study

- A study to determine the feasibility of a request for the new software

Stage in the Waterfall Model - Analysis

- Analysis of requirements for the software.
- The software engineer must understand all required functions, behavior, performance and interfacing.
- The requirements should be documented and reviewed with the customer.

Stage in the Waterfall Model – Design

- Software design is actually a multi-step process that focuses on four distinct attributes of a program :
  - Data structure
  - Software architecture
  - Interface representations
  - Procedural detail
- The design process translates requirements into a representation of a the software that can be assessed for quality.
- The design also should be documented.

Stage in the Waterfall Model – Code Generation

- The design must be translated into a machine readable form.
- The software design is realized as a set of programs or program units.

Stage in the Waterfall Model – Testing

- Focuses on the logical internals of the software for the intent of finding errors.
- Debugging will follow the conduct of successful testing.
- All statements should be tested to uncover errors.

Stage in the Waterfall Model - Maintenance

- After the installation and implementation of the software in the actual environment, errors/changes will invariably occur because software must accommodate changes in the real and external environment.

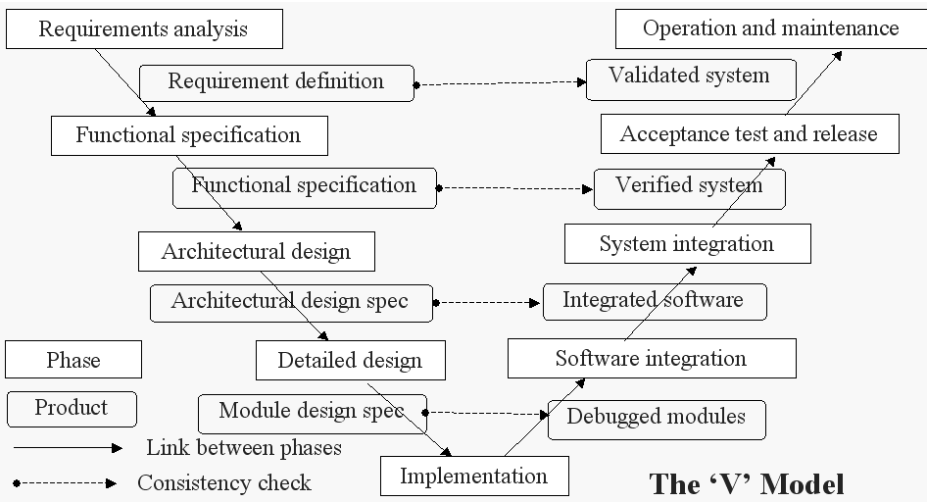
Limitations of Waterfall Model

- Difficult for the customer to state all requirements explicitly.
- A working version of the program will be available until late in the project time-span. A major blunder, if undetected until the working program is reviewed, can be disastrous.
- Developers are often delayed unnecessarily. Some project team members must wait for other members of the team to complete dependent tasks.

When to use the Waterfall Model?

- Straight forward (low risk) applications
- Experienced staff
- Clear requirements
- Requirements unlikely to change
- Stable technology

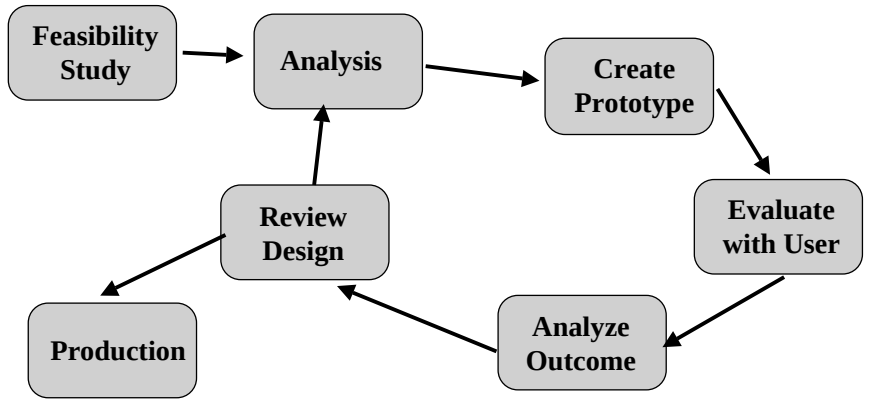
The V Model



Objective of V model

- It presents the same phases as the waterfall model, with each development phase matched by a corresponding verification or validation phase.
  - Verification – Are we building the product right?
  - Validation – Are we building the right product?

Prototyping

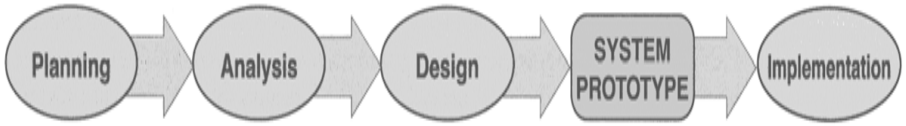


Prototyping Model

- This model is good to use when the customer has legitimate needs, but is not able to articulate the details at the start of the project.
- A small mock-up of a working system is developed and presented to the customer. Sometimes this first system is discarded and sometimes it is extended based on the customer's feedback.
- **Evolutionary Prototyping** or **Throwaway Prototyping** is used to verify user requirements.

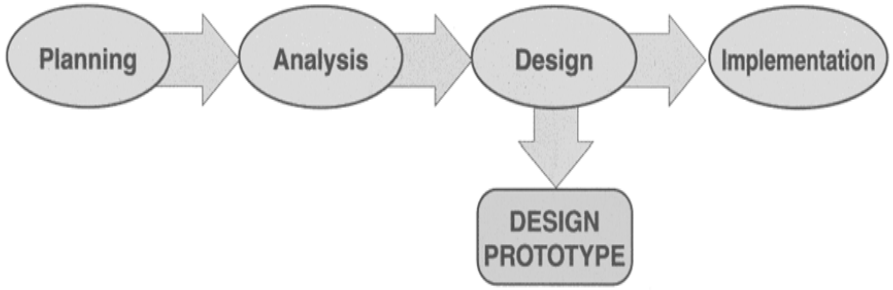
Evolutionary Prototyping

- The prototype surfaces out as the final product and there is no Product Engineering stage. To achieve this, analyze, design, code and test of prototype must be done carefully and much more time is required.



Throwaway Prototyping

- The role of prototype is to perfect requirements, and must be thrown away at the end. Don't try to deliver prototype to customer because it has not been properly designed, coded and tested.



Benefits of Prototyping

- Avoid misunderstanding.
- Create accurate specification.
- Evaluate a working model more effectively.
- Develop testing and training procedures before the finished system is available.
- Reduces the risk and potential financial exposure that occur when a finished system fails to support business needs

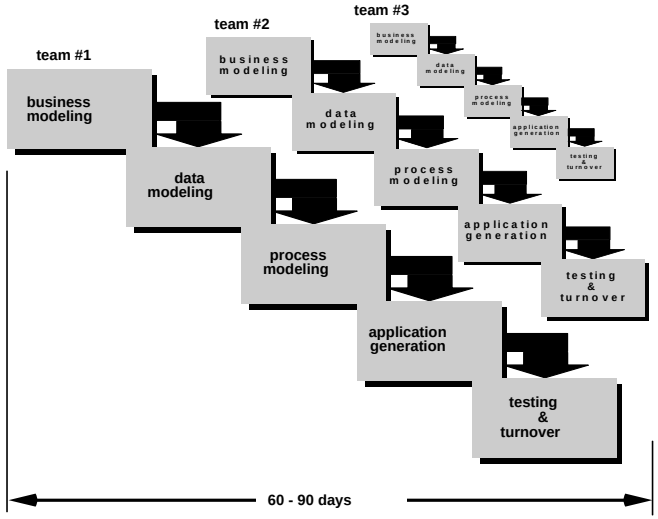
Problems in Prototyping

- Prototype is often rushed, and long term aspects of overall software quality and maintainability not considered.
- Other system requirements, such as reliability and maintainability, cannot adequately be tested using a prototype.
- In very complex systems, the prototype becomes unwieldy and difficult to manage.
- Overall software quality and maintainability not considered.
- Developer often makes implementation promises in order to get prototype working quickly.
- The uncontrolled iteration of continual creation and modification of prototype will lead to rapidly increasing costs in terms of time and manpower.

Using Prototyping

- Faster system development
- Facilitates end-user involvement
- Caters for uncertain requirements
- Appropriate exploratory approach for new technology
- Harder to control budgets / resource use
- Danger of 'design drift'

Rapid Application Development (RAD) Model



Rapid Application Development (RAD) Model

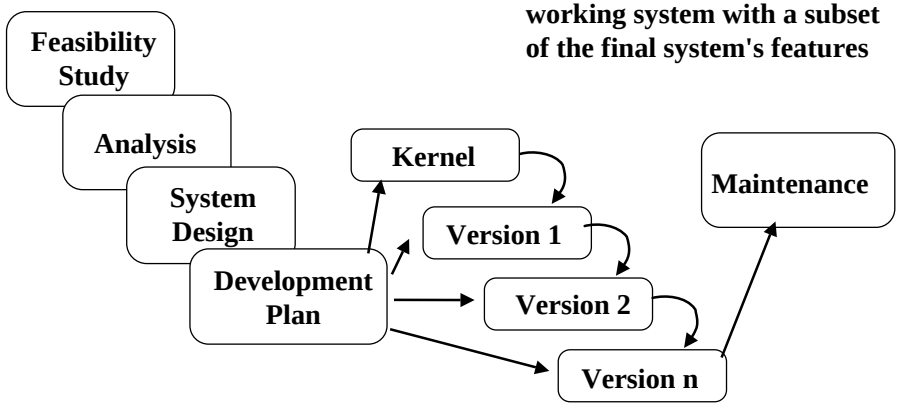
- It is a linear sequential software development process model that emphasizes an extremely short development cycle.
- This model is a “high-speed” adaptation of the linear sequential model in which rapid development is achieved by using a component-based construction approach.
- For large projects, RAD requires sufficient human resources to create the right number of RAD teams.

General Stages in Rapid Application Development

- Business Modeling
  - The information flow among business functions is modeled
- Data Modeling
  - The information flow is refined into a set of data objects that are needed to support the business
- Process Modeling
  - The data objects are transformed to achieve the information flow necessary to implement a business function. Process descriptions are created for adding, modifying, deleting, or retrieving a data objects
- Application Generation
  - Reuse existing program components or create reusable components
- Testing and Turnover
  - Test new components and interfaces.

Evolutionary Development

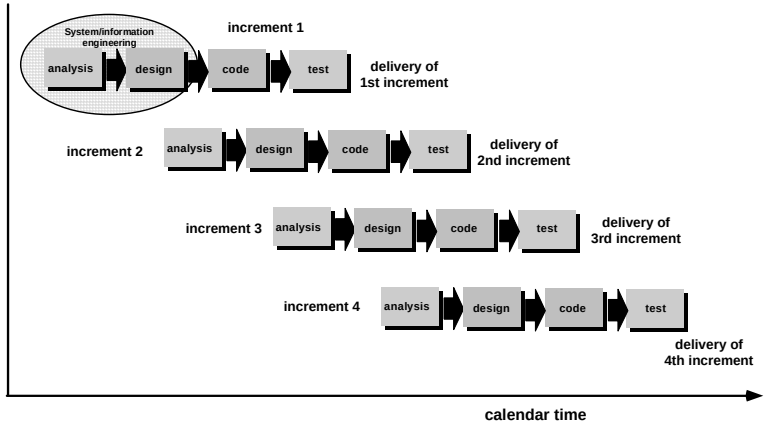
Each version is a complete working system with a subset of the final system's features



Evolutionary Development

- Evolutionary development breaks the construction and testing of the system into several phases or versions.
- A kernel version of the system is produced which possesses the minimum facilities of a workable system.
- The user can use the kernel version to gain experience of the software and feed back problems/bugs etc. to the designer.
- Each new version adds extra features.
- Evolutionary Software Process Models are iterative.
  - The Incremental Model
  - The Spiral Model

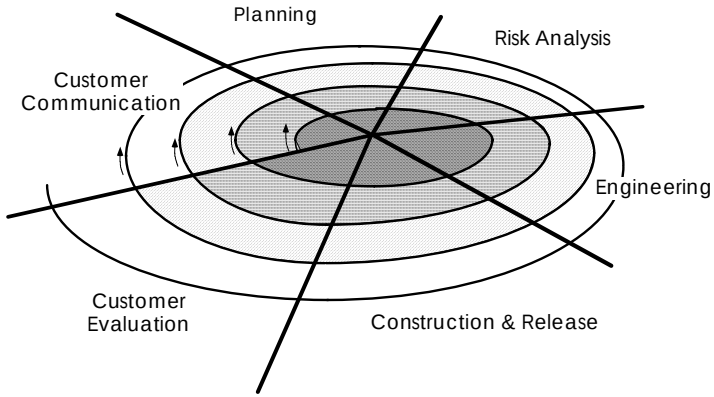
The Incremental Model



The Incremental Model

- This combines elements of the waterfall model (applied repetitively) with the iterative philosophy of prototyping. Each waterfall sequence produces a deliverable "increment (version)" of the software.
- The first increment is known as core product (Kernel). Here all basic requirements are addressed, but many supplementary features remain undelivered. This process is repeated until the complete product is produced.
- This model focuses on the delivery of an operational product with each increment (not like prototyping).
- Incremental model is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project.

An Evolutionary (Spiral) Model



The Spiral Model

- This is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic approach of waterfall model.
- Software is developed in a series of incremental releases.
  - During early iterations, the incremental release might be a paper model or prototype.
  - During later iterations, increasingly more complete versions of the engineered systems are produced.
- It provides the rapid development of incremental versions of software.
- Here, during early iterations, the incremental release might be paper model or prototype. During later iterations, more complete versions of the engineered system are produced.

Spiral Model

- Each cycle represent a project itself. E.g.
  - Concept Development
  - Product Development
  - Product Enhancement
  - Product Maintenance
- It is divided into Framework activities, usually 3-6. E.g.
  - Customer Communication - establish effective communication
  - Planning - define resources, timeline etc.
  - Risk Analysis - assess technical & management risks
  - Engineering - prototype
  - Construction & Release - code, test & release
  - Customer Evaluation - obtain customer feedback

General Stages of Spiral Model - Customer Communication

- Tasks required to establish effective communication between developer and customer

General Stages of Spiral Model - Planning

- Tasks required to define resources, timelines, and other project related information.

General Stages of Spiral Model - Risk Analysis

- Tasks required to assess both technical and management risks.

General Stages of Spiral Model - Engineering

- Tasks required to build one or more representations of the application.

General Stages of Spiral Model - Construction and Release

- Tasks required to construct, test, install and provide user support (e.g. documentation and training).

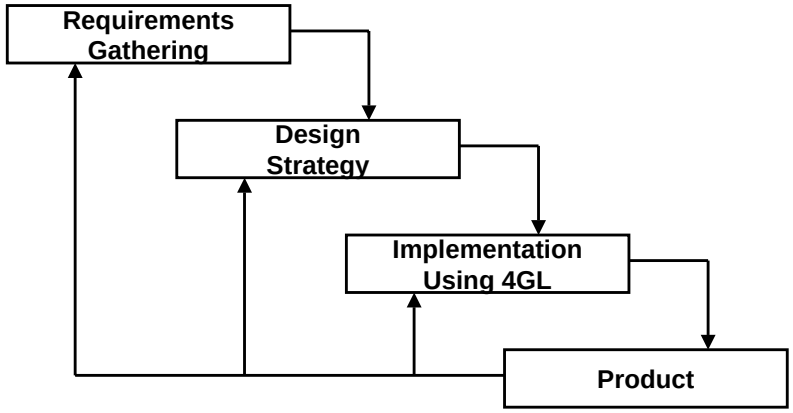
General Stages of Spiral Model - Customer Evaluation

- Tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.

Using Evolutionary Development

- Early first delivery and use
- Design can evolve with changed requirements
- “Open” approach to design needed (allow for change)
- Feed-back from users
- Low 'per step' risk
- Greater design effort balanced by timeliness

Fourth Generation Techniques



Fourth Generation Techniques

- Fourth Generation Techniques (4GT) encompasses a broad array of software tools, each of them enables the software developer to specify some characteristic of software at a high level.
- Example
  - Non procedural database query language
  - Report generation
  - Data manipulation
  - Screen interaction
  - Code generation
  - Graphics / Spreadsheets

General Stages of 4GT - Requirements Gathering

- Customer could actually describe the requirements and these would be directly translated into an operational prototype.

General Stages of 4GT – Design Strategy

- Possible to move directly from the requirements gathering step to implementation for small systems.

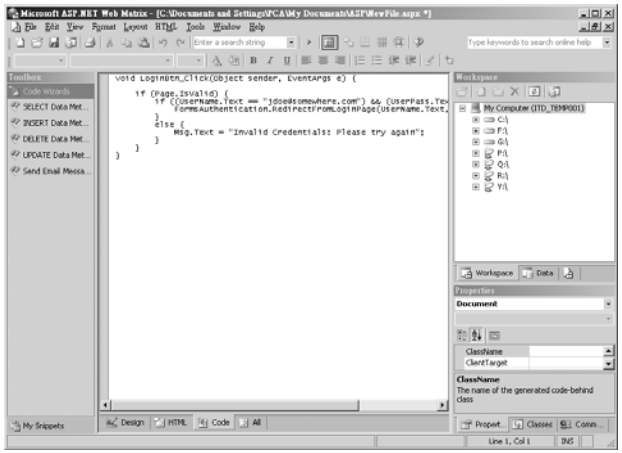
General Stages of 4GT - Implementation Using 4GL

- Typically, code can be generated based on some specification, e.g. input and output formats.

General Stages of 4GT – Product

- The final version of the software

Example of 4GT – Application Generators



Problems of 4GT

- Current Application domain for 4GT is limited to business information systems.
- Rapid system development is true only for small systems.
- The use of 4GL without design (for large projects) will cause the same difficulties that have been encountered when developing software using ad hoc approaches

Conclusion - Choosing a Model

- Before a project starts, the stages through which the project will go have to be decided upon.
- The type of project, end users, application area all affect the decision.