Fundamentals of Software Engineering

Peter Lo

Evolution of Software

- The Early Years (50's – mid 60's)
  - Batch Orientation
  - Limited Distribution
  - Customization of Software

Evolution of Software

- Second Era (60's – mid 70's)
  - Growth of software houses
  - Growth of the use of multiprogramming and multi-user systems
  - Introducing of human-machine interaction
  - Software crisis begins

Evolution of Software

- Third Era (70's – mid 80's)
  - Growth and widespread use of Personal Computers (PC)
  - Microprocessors in intelligent products
  - Growth of software companies

Evolution of Software

- Fourth Era (80's – Present)
  - Increasingly powerful desktop systems
  - Object-orientation, expert systems, artificial neural networks

Increase in Software Demand

- Growth of end-user computing due to the appearance of 4 generation techniques.
- Rapidly decreasing price of hardware, making computers more affordable devices at home.
- Hardware sophistication seems to grow faster than software.
- Increasing use of computer technology

Software Crisis

- Reason of software crisis:
  - Inaccurate schedule and cost estimation
  - Productivity of software developers has not kept pace with the demand for their services
  - Inadequate quality of software
  - Little data on the software development process

Why Software Engineering?

- The questions and problems faced in the history of software development have thus been instrumental to the creation and adoption of modern software engineering practices.

Definition of Software Engineering

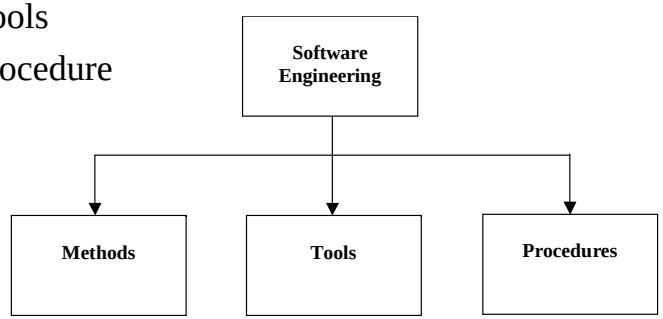
- Software Engineering is the establishment and use of sound engineering principles to economically build functional and reliable software.

Skills necessary for software engineers

- Software Skills
  - This is the nuts and bolts of actually writing the software.
  - This would typically include software construction, programming, designing the architecture of the software and so on.
- Project Management Skills
  - This refers to the how-to control the development of software (e.g. budgetary constraints, human resources) and subsequent maintenance of software.

Key Elements of Software Engineering

- Three key elements of Software Engineering:
  - Method
  - Tools
  - Procedure



Methods

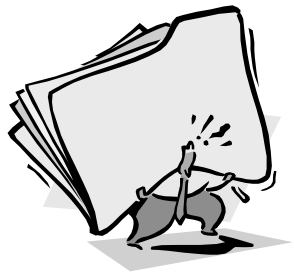
- Refer to techniques on how-to build software.
- Examples:
  - Project Planning
  - System and Software Requirement Analysis
  - Coding, Testing and Maintenance.

Tools

- Automated or semi-automated support and objects used to assist the application of methods.
- Example:
  - CASE (Computer-Aided Software Engineering)

Procedures

- Defines the sequence in which methods are applied, and makes sure that the development of software is logical and on time.



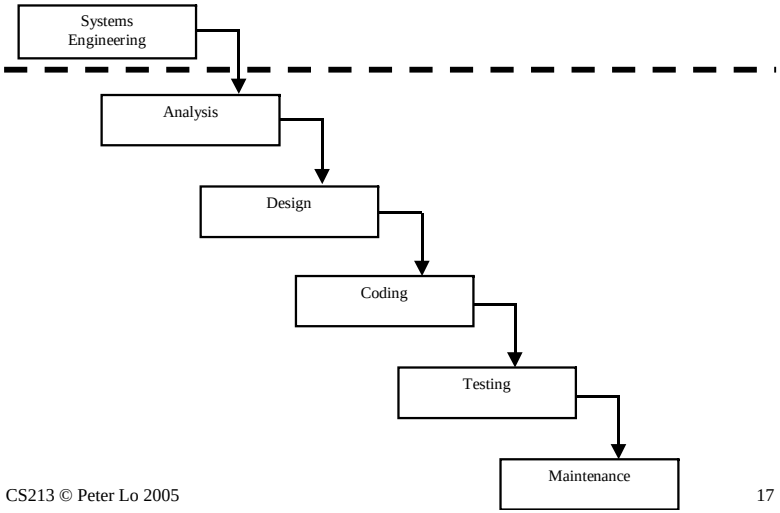
Software Engineering Paradigms

- Software Engineering comprises of a set of steps that encompass each of the three elements (Methods, Tools and Procedure).
- These steps are often referred to as **Software Engineering Paradigms**.

Software Engineering Life-Cycles

- There are a large number of methods of development, including object-oriented models.
- Example:
  - Classical Life Cycle (Linear Sequential Model)
  - Prototyping Model
  - Fourth Generation Techniques
  - Spiral Model (Evolutionary Software Process Model)
  - A combination of several techniques

Classic Life Cycle



Systems and Information Engineering

- Software is always part of a larger system or business at work, work begins by establishing requirements for all system elements and then allocating some subset of these requirements to software.
- Essential for interfacing correctly with external components, e.g. databases, hardware

Software Requirement Analysis

- Analysis of requirements is now focused on software alone.
- Requirements for both system and the software are documented and reviewed with the customer.

Design

- The multi-step process that focuses on four distinctive attributes of the program:
  - Data structures
  - Software architecture
  - Procedural detail
  - Interface characteristics.
- The design process translates requirements into a representation of the software.

Coding

- Design is translated to machine readable form
- The software design is realized as a set of programs or program units.

Testing

- Focuses on the logical internals of the software for the intent of finding errors.
- Debugging will follow the conduct of successful testing.

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.

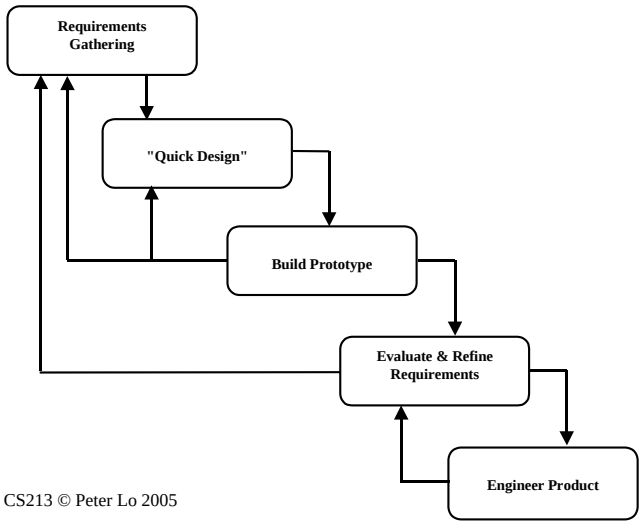
Common Problems Faced with the Classic Life-Cycle

- Real projects rarely follow sequential flow of the "waterfall" model that the model proposes.
- It is difficult to determine requirements so explicitly and so early in the development. There is always a lot of uncertainty at the start of the project.
- The customer has to wait for a really long time before he even gets a feel of the real project, A major blunder, if undetected until the working program is reviewed, can be disastrous.
- Developers often delayed unnecessarily. Due to the linear nature of the classic life cycle, “blocking states” can occur when some project team members must wait for other members of the team to complete dependent tasks.

Prototyping

- A prototype is an early, rapidly constructed working version of the proposed information system.
- **Design Prototyping** or **Throwaway Prototyping** is used to verify user requirements.

Prototyping Model



Requirements Gathering

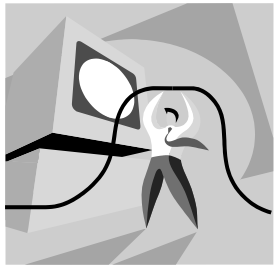
- Developer and customer meet to define overall objectives of software, identify whatever requirements are known, and outline areas where further definition is mandatory.

Quick Design

- Quick implementation of these aspects of the software that will be visible to the customer/user

Build Prototype

- Leads to construction of prototype.



Evaluate and Refine Requirements

- Prototype is evaluated by the customer/user and is used to refine requirements for the software to be developed.

Engineer Product

- With a confirmed set of requirements based on the prototype, actual development of the final software can begin.

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.
- Developer often makes implementation promises in order to get prototype working quickly.
- If prototyping itself is not controlled and carried out with some discipline, there is a possibility of “feature creep”.
- 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.

Fourth Generation Techniques

- 4th Generation Techniques actually encompasses a broad array of software tools which have one thing in common:
  - Each of them enables the software developer to specify some characteristic of software at a high level.
- The tool then automatically generates source code based on the developer specification.

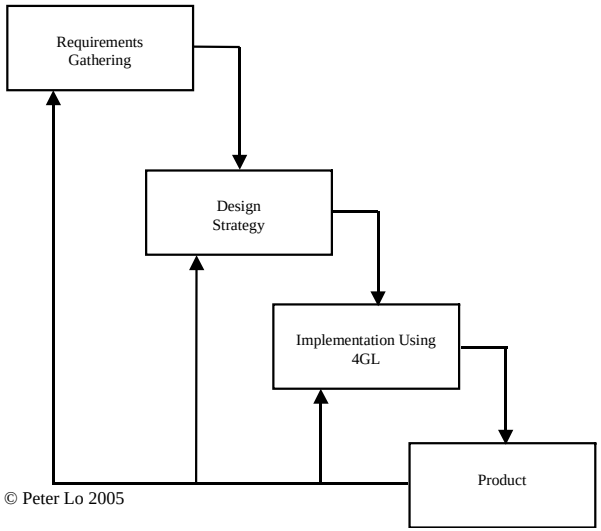
Fourth Generation Techniques

- 4th Generation Techniques are often regarded as “easy to use” for several reasons.
  - They combine both procedural and non-procedural features into the method of development.
  - The method of development is often intuitive, e.g. through graphical user interfaces.

Some features of 4L Tools

- Nonprocedural languages for database query
- Report generation
- Data manipulation
- Screen interaction and definition
- Code generation
- High-level graphics development

Fourth Generation Techniques



Requirements Gathering

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

Design Strategy

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

Implementation Using 4GL

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

Product

- The final version of the software

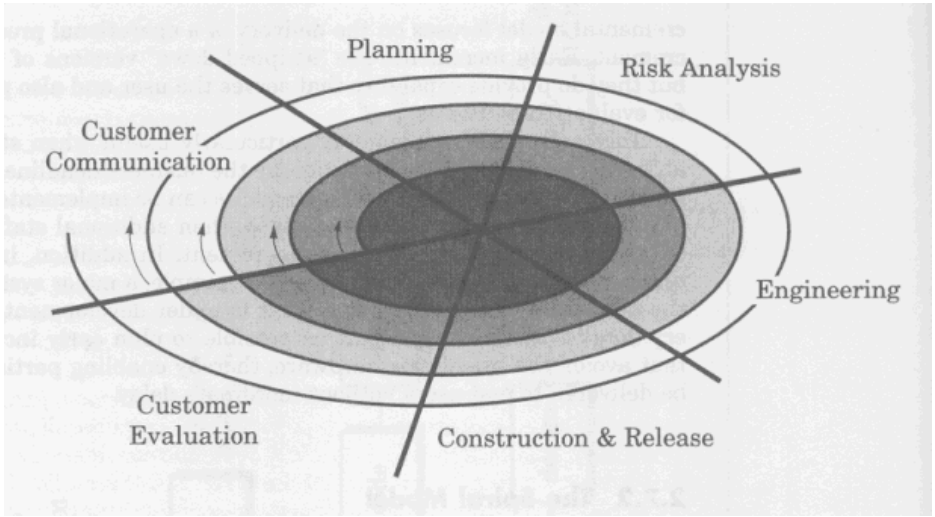
Possible Problems

- Current application domain for 4GL is limited to business information systems, for example information analysis and reporting that is keyed to large databases.
- Rapid system development is true only for small systems.

Spiral Model

- Spiral Models are a evolutionary models
- They are iterative.
- They are characterized by a manner that enables software engineers to develop increasingly more complete versions of the software.

Spiral Model



Spiral 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.

Customer Communication

- Tasks required to establish effective communication between developer and customer

Planning

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

Risk Analysis

- Tasks required to assess both technical and management risks.

Engineering

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

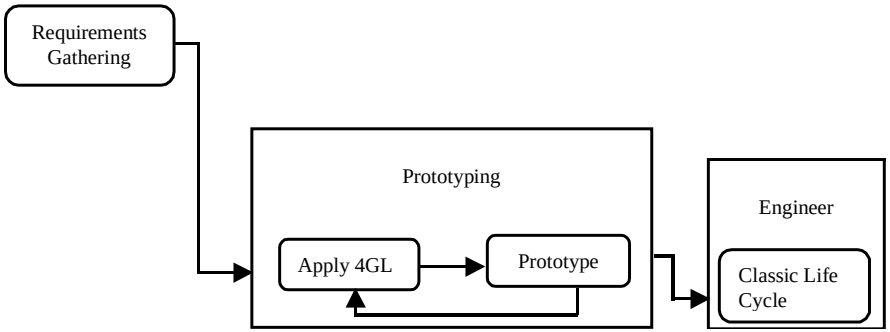
Construction and Release

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

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.

Combining Paradigms



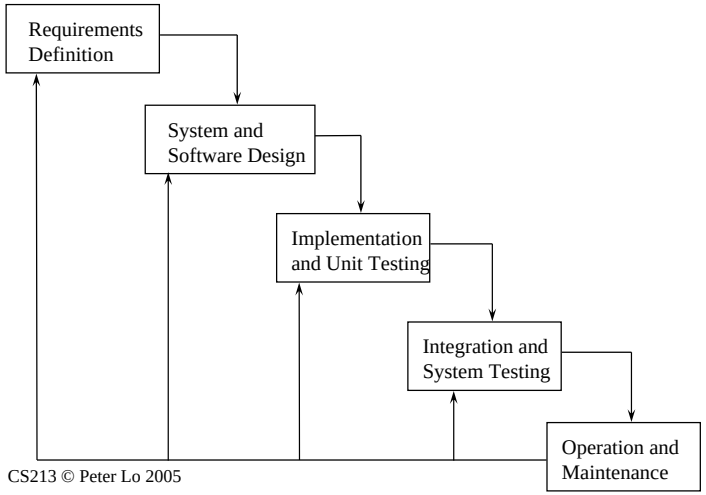
Combining Paradigms

- In many cases, paradigms can be combined and the strengths of each can be utilized in a single project.
  - Requirements gathering is still essential, and interaction between the developer and customer must still occur.
  - To create the prototype, 4GL can be applied to develop the prototype quickly.
  - The prototype will then be evaluated and refined, and once requirements are finally established;
  - Design and implementation steps of the classic life cycle can be applied to engineer the software formally.

Appendix

Waterfall Model

Waterfall Model



Waterfall Model

- Requirements analysis and definition
  - The system's services, constraints and goals are established by consultation with system users.
  - They are then defined in a manner which is understandable by both users and development staff.
- System and software design
  - This process partitions the requirements to either software or hardware systems.
  - It establishes an overall system architecture.
  - Software design involves representing the software system functions in a form that may be transformed into one or more executable programs.

Waterfall Model

- Implementation and unit testing.
  - The software design is realized as a set of programs or program units.
  - Unit testing involves verifying that each unit meets its specification.
- Integration and system testing.
  - The individual program units or programs are integrated and tested as a complete system to ensure that the software requirement have been met.
  - After testing, the software system is delivered to the customer.

Waterfall Model

- Operation and maintenance.
  - The system is installed and put into practical use.
  - Maintenance involves correcting errors which were not discovered earlier, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

Waterfall Model

- The waterfall model is a documentation-driven model and offers a means of making the development process more visible.
- The waterfall model is a dynamic model, and the feedback loops play an important role.
- It was the only widely accepted model until the early 1980.
- The model is also called **Software Life Cycle**.

Advantages and Disadvantages

- Advantages:
  - Enforces disciplined approach - the stipulation that document be provided at each phase.
  - The requirement that all the products of each phase (including the documentation) be carefully checked by Software Quality Assurance (SQA).
  - Inherent in every phase of the model is testing.
- Disadvantages
  - Inflexible partitioning of the project into distinct stages
  - Delivered systems are sometimes unusable as they do not meet the customer's real requirements.