SOFTWARE ENGINEERING

Introduction to Software Engineering:

- Nature of Software
- Software Engineering
- Software Process,
- Software Engineering Practice

Software Process Models:

- Linear Model
- RAD Model
- Incremental Model
- Spiral Model
- Component –based development
- Fourth Gen Techniques.

Software Evolution

Software's Dual Role

- **Software is a product**
  - Delivers computing potential
  - Produces, manages, acquires, modifies, displays, or transmits information

- **Software is a vehicle for delivering a product**
  - Supports or directly provides system functionality
  - Controls other programs (ex: an operating system)
  - Effects communications (ex: networking software)
  - Helps build other software (ex: software tools)

The early years (Before 1960):

- Batch orientation
- Limited distribution
- Custom software
- Personalized software environment
- No documentation

The second era (During mid 1960s to late 1970s):

- Multiprogramming & Multi-user
- Human machine interaction
- Real-time
- Producing outputs in milliseconds rather than minutes
- Database
- Product software

The third era (During mid 1970s to late 1980s):

- Distributed systems
- Communication between computers
- Embedded "intelligence"
- Low cost hardware
- Consumer impact

The fourth era (During mid 1980s to till today):

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

What is Software?

Is a Set of

- ***Instructions /computer programs that when executed provide desired features, function & performance;***
- ***Data structures that enable the programs to adequately manipulate information;***
- ***Documents that describe the operation and use of programs*****.**

Differences between Program and Software

PROGRAM	**SOFTWARE**
Usually small in size	Large
Author himself is sole user	Large number of users
Single developer	Team of developers
Lacks proper user interface	Well-designed interface
Lacks proper documentation	Well documented & user-manual prepared
Ad hoc development	Systematic development

•

- Software Products may be
  - Generic
  - Custom

Advantages:

- Software is intangible
- Software is easy to reproduce
- Software is easy to modify

Introduction to Software Engineering:

- Nature of Software
- Software Engineering
- Software Process,
- Software Engineering Practice

Software Process Models:

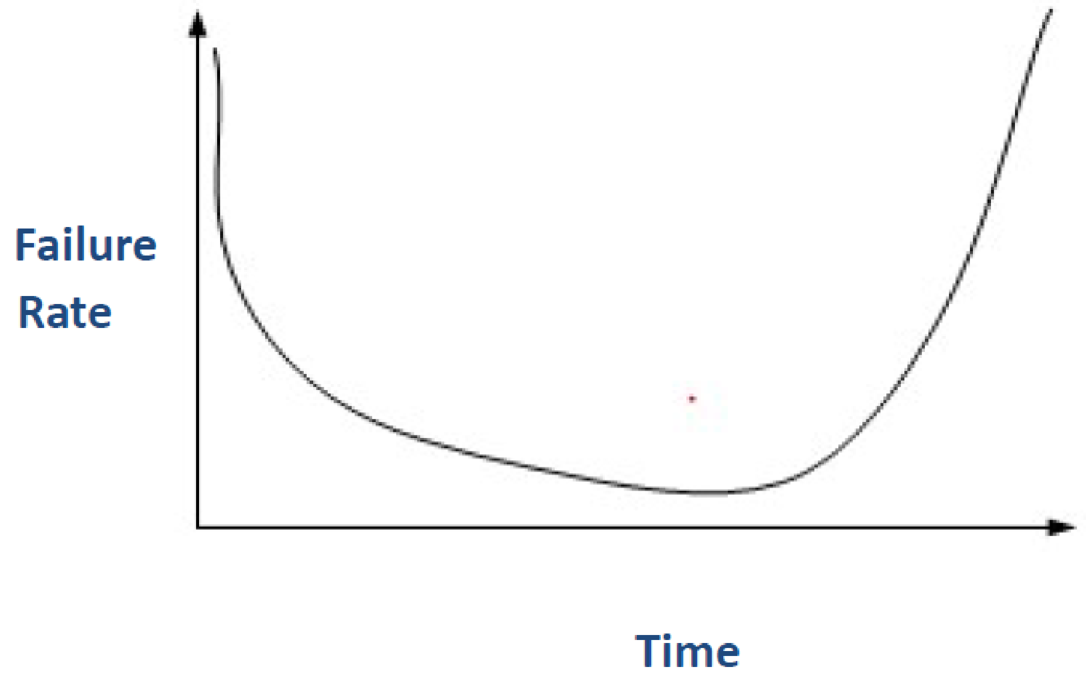
- Linear Model
- RAD Model
- Incremental Model
- Spiral Model
- Component –based development
- Fourth Gen Techniques.

Software Characteristics

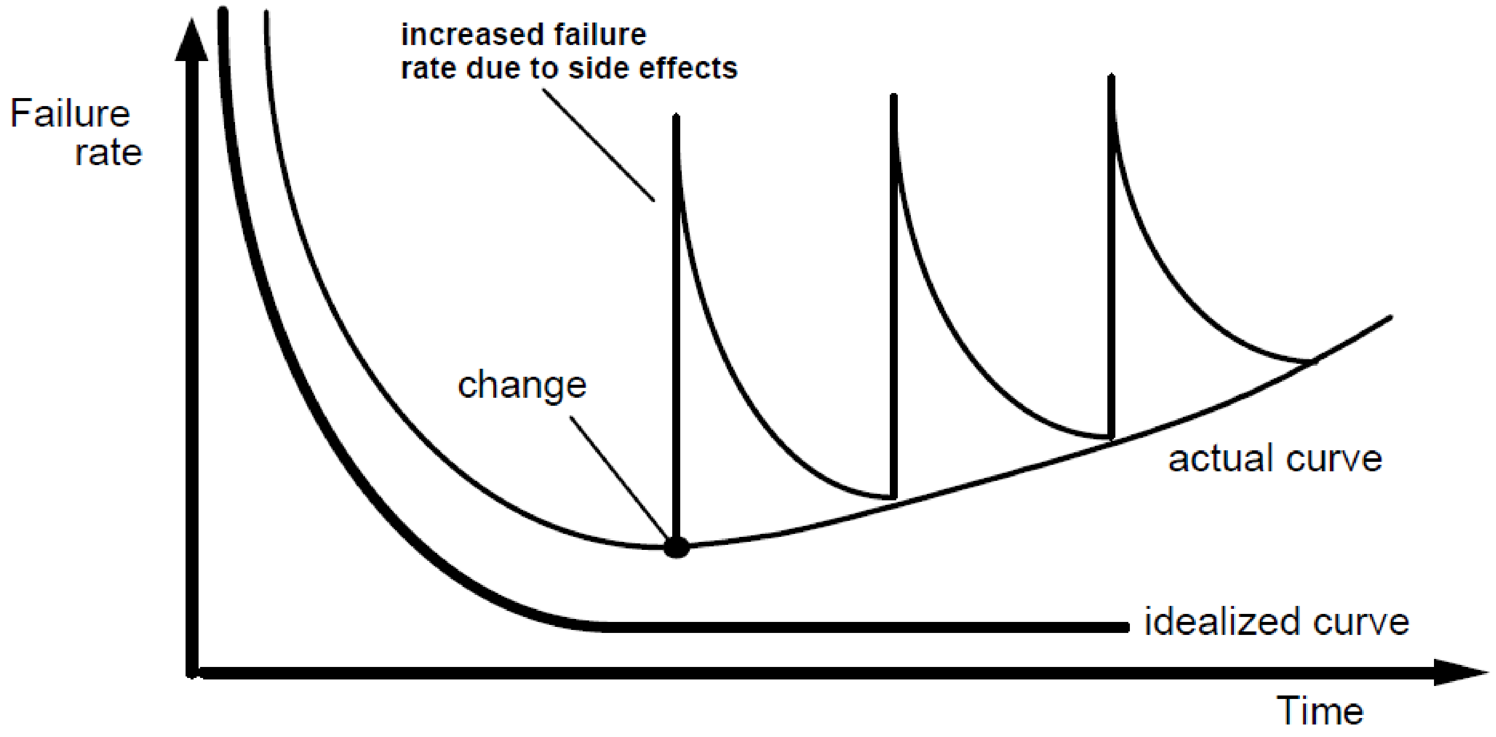
- *It is developed or engineered; it is not manufactured in the classical sense.*
- *Software does not "wear out".*
- *Although the industry is moving toward component-based construction, most software continues to be custom built.*

Software does not "wear out".

Failure Curve for Hardware (Ware out)



Wear vs. Deterioration



Actual failure curve for software

Software Application Domains

- **System software**
  - Is a collection of programs written to service other programs.

    It is characterized by
    - Heavy interaction with computer hardware
    - Heavy usage by multiple users.
- **Application software**
  - Application s/w consists of standalone programs that solve a specific business need
    - Ex: Point-of-sale transaction processing
- **Engineering/scientific software**
  - It has been characterized by **“ number crunching”** algorithms

Contd….

Software Application Domains

- **Embedded software**
  - Embedded software resides in read only memory and is used to control products and systems for the consumer and industrial markets.
  - Embedded software can perform very limited and esoteric functions.
    - Ex: Micro woven, digital functions in an automobile such as fuel control, Traffic signal system.
- **Product-line software**
  - It can focus on a limited and esoteric market place.
    - Ex: Inventory control products

Contd….

Software Application Domains

- **Web Apps (Web applications)**
  - " Web apps" span a wide array of applications.
  - It can be little more than a set of linked hypertext files that resent information using text and limited graphics.
  - It provides stand alone features.
    - Ex: e-commerce , e-seva …..etc
- **AI software**
  - AI s/w makes use of nonnumerical algorithms to solve complex problems that are not amenable to computation or straight forward analysis.
    - Ex: Robotics, expert systems, pattern recognition systems, Theorem proving, Game planning ….etc
- **Software—New Categories**
  - **Ubiquitous computing** — wireless networks
  - **Netsourcing** — The Web as a computing engine
  - **Open source** — "free" source code open to the computing community

Legacy Software

- Support **core business** functions
- Have **longevity** and **business criticality**
- Exhibit **poor quality**
  - Poor documentation, poor testing, poor change management.

Reasons for evolving Legacy software

- **Adaptive**
  - Must be adapted to meet the needs of new computing environments or more modern systems, databases or networks.
- **Perfective**
  - Must be enhanced to implement new business requirements.
- **Corrective**
  - Must be changed because of errors found in the specification, design and implementation.

Introduction to Software Engineering:

- Nature of Software
- Software Engineering
- Software Process,
- Software Engineering Practice

Software Process Models:

- Linear Model
- RAD Model
- Incremental Model
- Spiral Model
- Component –based development
- Fourth Gen Techniques.

What is Software Engineering?

IEEE has developed a more comprehensive definition:-

The application of a systematic, disciplined, quantifiable approach to the development, operation, & maintenance of software; that is, the application of engineering to software.

Software Engineering

- **What software features and functions should be delivered.**
- ***It follows that a concerted effort should be made to understand the problem before a software solution is developed.***
- ***Software design becomes a pivotal activity.***
- ***Software should exhibit high quality.***
- ***software should be maintainable***

Introduction to Software Engineering:

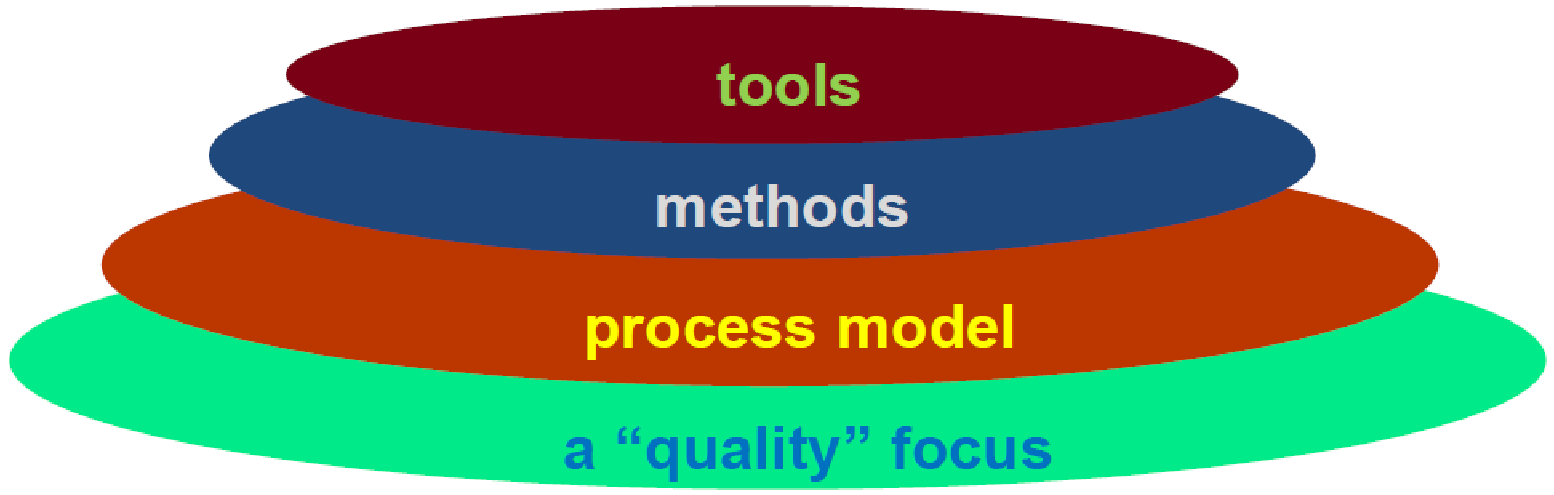
- Nature of Software
- Software Engineering
- Software Process,
- Software Engineering Practice

Software Process Models:

- Linear Model
- RAD Model
- Incremental Model
- Spiral Model
- Component –based development
- Fourth Gen Techniques.

A Generic view of process:

- Software Engineering - A Layered Technology



S/W Engineering Layers

- S/W engineering is an outgrowth of hardware and system engineering.

- It encompasses a set of three key elements.

    - **Tools**
    - **Methods**
    - **Process/ Procedures**

Process

- The foundation for software engineering is the ***process*** **layer**.
- Process defines a framework for a set of ***key process areas*** **(KPAs)** that must be established for effective delivery of software engineering technology.
- The key process areas form the basis for management control of software projects and establish the context in which technical methods are applied, **work products** (models, documents, data, reports, forms, etc.) are produced, **milestones** are established, **quality** is ensured, and **change** is properly managed

Methods

- It provides the **technical how-to's** for building software.
- It includes a broad array of tasks that include **project planning & estimation, system & software requirements analysis**, **design of data structures**, **program architecture & algorithm procedure**, coding, testing, and support.
- It often introduces a special language – oriented or graphical notation and introduce a set of criteria for software quality.

Tools

- It provides automated or semi-automated support for the process and the methods.
- When tools are integrated so that information created by one tool can be used by another, a system for the support of software development called ***computer-aided software engineering (CASE),*** is established.
- CASE combines software, hardware, and a software engineering database (a repository containing important information about analysis, design, program construction, and testing) to create a software engineering environment analogous to CAD/CAE (computer-aided design / engineering) for hardware.

Generic View of Software Engineering

Engineering is the analysis, design, construction, verification and management of technical entities. Regardless of the entity to be engineered, the following questions must be asked and answered.

- What is the problem to be solved?
- What characteristics of the entity are used to solve the problem?
- How will the entity be realized?
- How will the entity be constructed?
- What approach will be used to uncover errors, that were made the design and construction of the entity?
- How will the entity be supported over the long term, when corrections, adaptations, and enhancements are requested by users of the entity?

To work associated with s/w engineering can be categorized into three generic phases, regardless of application area, project size or complexity. Each phase addresses one or more of the questions. Three phases are:

- **Definition: What**
  - What information is to be processed?
  - What function and performance are desired?
  - What system behavior can be expected?
  - What validations are required to define a successful system?
- **Development: How**
  - How data are to be structured?
  - How function is to be implemented within a s/w architecture?
  - How procedural details are to be implemented?
  - How the design will be translated into a programming language?
  - How testing will be performed?
- **Maintenance/Support : Changes**
  - Four types of changes are encountered during the support phase
    - Correction
    - Adaptation
    - Enhancement
    - Prevention

A Process Framework

- It establishes the foundation for a complete software process for all projects regardless of their size or complexity.
- This model stresses detailed definition, identification and application of process activities.
- Process framework
  - **Framework activities**
    - work tasks
    - work products
    - milestones & deliverables
    - QA checkpoints
  - **Umbrella Activities**

Framework Activities

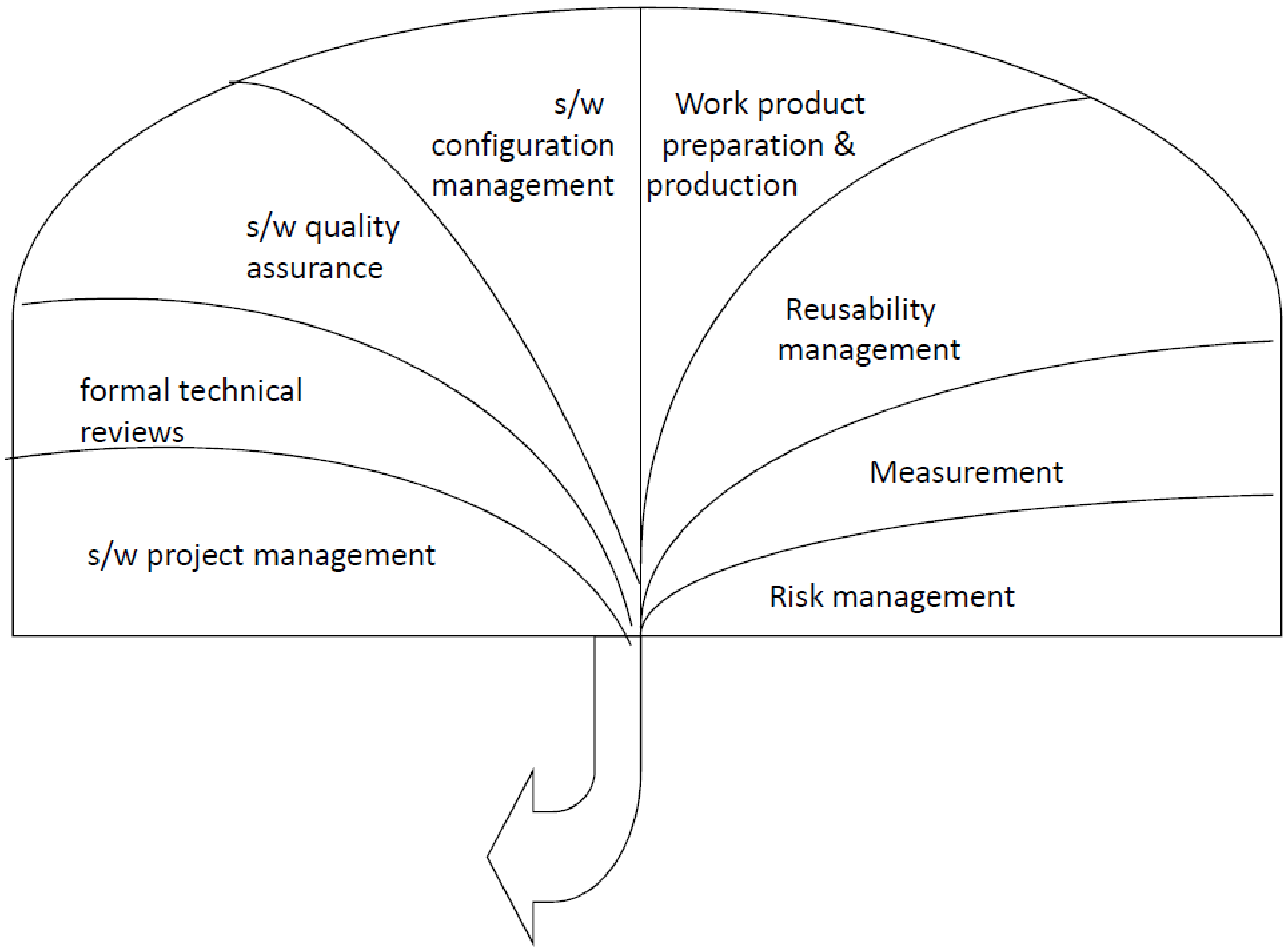
The following generic process framework is applicable to the vast majority of software projects:-

- **Communication**
  - Involves heavy communication & collaboration with customer and includes requirements gathering & other related activities.
- **Planning**
  - It describes technical tasks to be conducted, risks that are likely, resources that will be required, work products to be produced and a work schedule.
- **Modeling**
  - Allow developer & customer to better understand software requirements
  - Design
- **Construction**
  - Code generation
  - Testing
- **Deployment**
  - Usage & enhancement

Above 5 generic frame activities can be used during the development of small or large programs.

Umbrella Activities

- Software project management
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Work product preparation and production
- Reusability management
- Measurement
- Risk management


s/w configuration management
Work product preparation & production
s/w quality assurance
Reusability management
formal technical reviews
Measurement
s/w project management
Risk management

Software project management

- software project tracking + control.
- It allows the software team to review progress against the project plan and take necessary action to maintain schedule.

Formal technical reviews

- It reviews software engineering work products in an effort to uncover & remove errors before they are propagated to the next action or activity.

Software quality assurance

- It defines & conducts the activities required to ensure software quality.

Software configuration management

- It manages the effects of change throughout the software process.

Work product preparation & production

- It includes the activities required to create work products such as Models, Documents, Logs, forms & lists.

Reusability management

- It defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.

Measurement

- It defines & collects process, project & product measures that assist the team in delivering software that meets customer’s needs; can be used in combination with all other frame work & umbrella activities.

Risk management

- It assesses risks that may effect the outcome of the project or the quality of the product.

Process models differ fundamentally in:

1. The overall flow of activities & tasks and interdependencies among activities & tasks.
2. The degree to which work tasks are defined within each framework activity.
3. The degree to which work products are identified & required.
4. The manner which quality assurance activities are applied.
5. The manner in which project tracking & control activities are applied.
6. The overall degree of detail & rigor with which the process is described.
7. The degree to which customer & the other stakeholders are involved with the project.
8. The level of autonomy given to the software project team.
9. The degree to which team organization & roles are prescribed.

Introduction to Software Engineering:

- Nature of Software
- Software Engineering
- Software Process,
- Software Engineering Practice

Software Process Models:

- Linear Model
- RAD Model
- Incremental Model
- Spiral Model
- Component –based development
- Fourth Gen Techniques.

SOFTWARE ENGINEERING PRACTICE

The Essence of Practice

Polya suggests:

- ***Understand the problem*** (communication and analysis).
- ***Plan a solution*** (modeling and software design).
- ***Carry out the plan*** (code generation).
- ***Examine the result for accuracy*** (testing and quality assurance).

Understand the Problem

- *Who has a stake in the solution to the problem?*
  - who are the stakeholders?
- *What are the unknowns?*
  - What data, functions, and features are required to properly solve the problem?
- *Can the problem be compartmentalized?*
  - Is it possible to represent smaller problems that may be easier to understand?
- *Can the problem be represented graphically?*
  - Can an analysis model be created?

Plan the Solution

- ***Have you seen similar problems before?***
  - Are there patterns that are recognizable in a potential solution?
  - Is there existing software that implements the data, functions, and features that are required?
- ***Has a similar problem been solved?***
  - If so, are elements of the solution reusable?
- ***Can subproblems be defined?***
  - If so, are solutions readily apparent for the subproblems?
- ***Can you represent a solution in a manner that leads to effective implementation?***
  - Can a design model be created?

Carry Out the Plan

- *Does the solution conform to the plan?*
  - Is source code traceable to the design model?
- *Is each component part of the solution provably correct?*
  - Has the design and code been reviewed, or better, have correctness proofs been applied to algorithm?

Examine the Result

- ***Is it possible to test each component part of the solution?***
  - Has a reasonable testing strategy been implemented?

- ***Does the solution produce results that conform to the data, functions, and features that are required?***
  - Has the software been validated against all stakeholder requirements?

Hooker’s General Principles

- *The Reason It All Exists*
- *KISS (Keep It Simple, Stupid!)*
- *Maintain the Vision*
- *What You Produce, Others Will Consume*
- *Be Open to the Future*
- *Plan Ahead for Reuse*
- *Think!*

SUMMARY

- s/w has become the key element in the evolution of computer-based systems.

- The intent of s/w engineering is to provide a framework for building higher quality s/w.

- S/W is collection of programs or set of instructions that when executed provide desired function.

Summary (contd….):

S/W evolutions.

The Law of Continuing Change (1974): E-type systems must be continually adapted else they become progressively less satisfactory.

The Law of Increasing Complexity (1974): As an E-type system evolves its complexity increases unless work is done to maintain or reduce it.

The Law of Self Regulation (1974): The E-type system evolution process is self-regulating with distribution of product and process measures close to normal.

The Law of Conservation of Organizational Stability (1980): The average effective global activity rate in an evolving E-type system is invariant over product lifetime.

The Law of Conservation of Familiarity (1980): As an E-type system evolves all associated with it, developers, sales personnel, users, for example, must maintain mastery of its content and behavior to achieve satisfactory evolution.

The Law of Continuing Growth (1980): The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime.

The Law of Declining Quality (1996): The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.

The Feedback System Law (1996): E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base.

The Primary Goal of Any Software Process: ***High Quality***

- Remember:

High quality = project timeliness

- Why?

Less rework!