INTRODUCTION TO SOFTWARE ENGINEERING

- Nature of Software
- Unique Nature of Web Apps
- Software Engineering
- Software Engineering Practices
- Software Myths
- Generic Process Model

What is Software?

Software is:

(1) ***instructions*** *(computer programs) that when executed provide desired features, function, and performance;*

(2) ***data structures*** *that enable the programs to adequately manipulate information and*

(3) ***documentation*** *that describes the operation and use of the programs.*

NATURE OF SOFTWARE

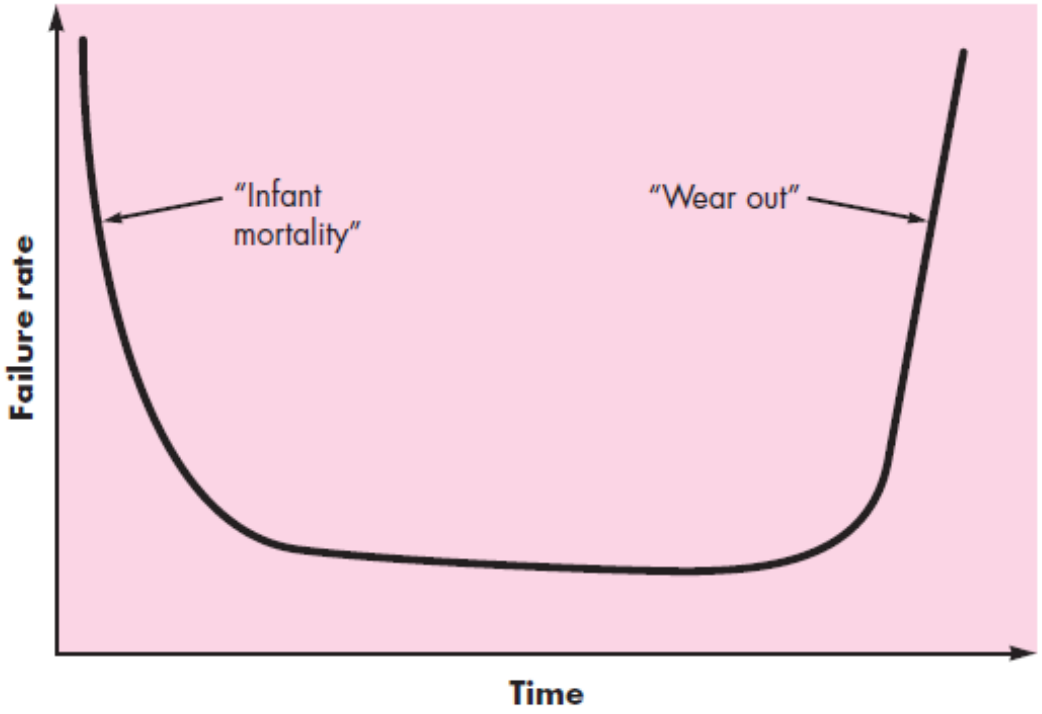
- Software is an information transformer—producing, managing, acquiring, modifying, displaying, or transmitting information
- Software delivers the most important product of our time—***information***
- questions that are asked when modern computer-based systems are built
  - Why does it take so long to get software finished?
  - Why are development costs so high?
  - Why can’t we find all errors before we give the software to our customers?
  - Why do we spend so much time and effort maintaining existing programs?
  - Why do we continue to have difficulty in measuring progress as software is being developed and maintained?

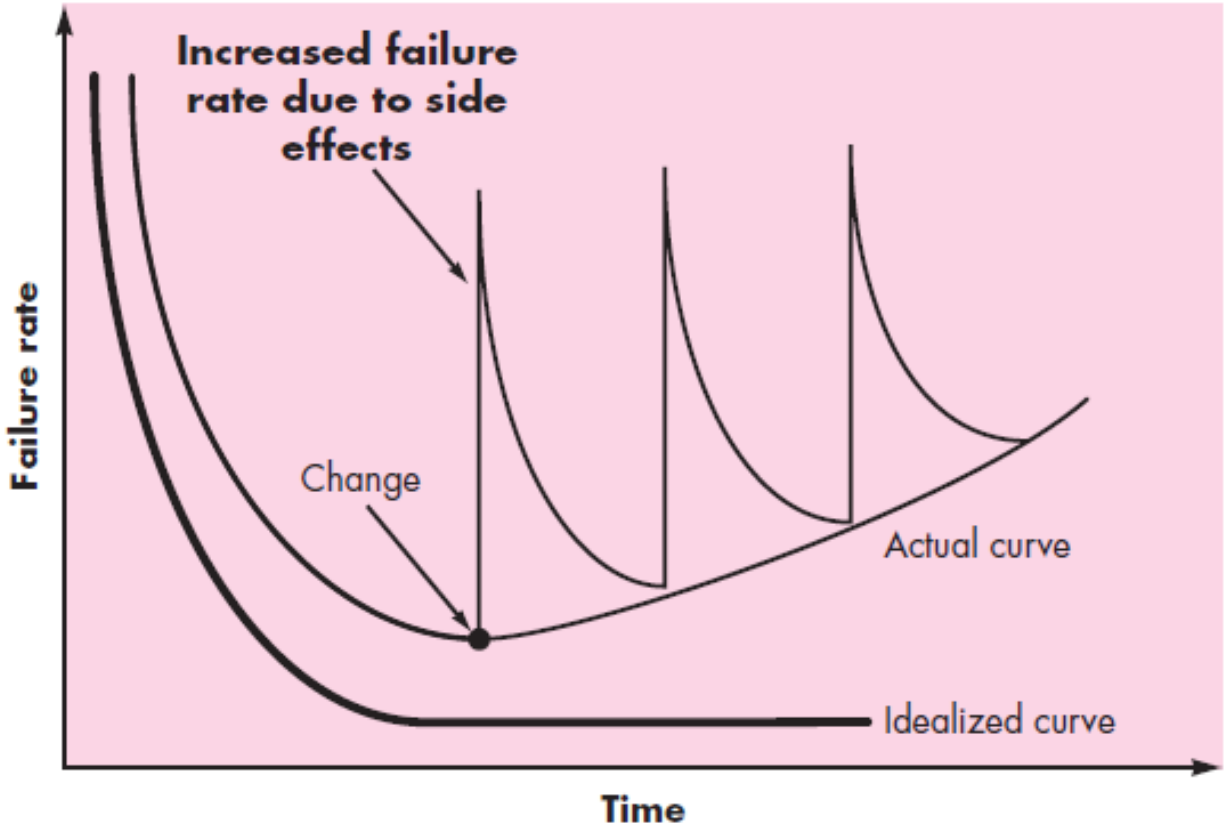
FEATURES OF SOFTWARE?

- **Features of such logical system:**

  - Software is **developed or engineered**; it is not manufactured in the classical sense which has quality problem.
  - Software **doesn't "wear out.”** but it deteriorates (due to change). Hardware has bathtub curve of failure rate (high failure rate in the beginning, then drop to steady state, then cumulative effects of dust, vibration, abuse occurs).
  - Although the industry is moving toward component-based construction (e.g. standard screws and off-the-shelf integrated circuits), most software continues to be **custom-built**. Modern reusable components encapsulate data and processing into software parts to be reused by different programs. E.g. graphical user interface, window, pull-down menus in library etc.

WEAR VS. DETERIORATION





SOFTWARE APPLICATIONS

- **System software:** to service other programs
  - E.g. Compilers, editors, file managers, OS, drivers, networking s/w etc
- **Application software:** to solve specific business need
  - E.g. Transaction Processing, Manufacturing process control etc
- **Engineering/scientific software :** to develop algorithms for real time
  - E.g. CAD, System Simulation
- **Embedded software :** Software+Hardware like interface
  - E.g. Digital functionalities in automobile, keypad for microwave
- **Product-line software:** to provide specific capability
  - E.g. Word, Multimedia applications, database etc
- **WebApps (Web applications) :** Online
- **Artificial Intelligence software :** to solve complex problems
  - E.g. Pattern Recognition, Artificial Neural Network, Game Playing etc.

- System software: This type of software is designed to provide essential services and functionality to other programs. It includes compilers, editors, file managers, operating systems (OS), drivers, networking software, and more. System software helps manage the computer's resources and provides a platform for other software to run on.
- Application software: This type of software is developed to solve specific business needs or perform specific tasks. Examples include transaction processing software, manufacturing process control software, and other software applications that address specific tasks or operations within a business or organization.
- Engineering/Scientific software: This type of software is used for developing algorithms and simulations for real-time applications. Examples include computer-aided design (CAD) software and system simulation software, which are used in engineering and scientific fields.
- Embedded software: This type of software is a combination of software and hardware interfaces. It is typically used in specialized devices where software is embedded in the hardware to provide specific functionality. Examples include digital functionalities in automobiles or the keypad interface for a microwave.
- Product-line software: This type of software is designed to provide specific capabilities or functionality to users. Examples include word processing software, multimedia applications, and database management systems.
- WebApps (Web applications): These are online applications that run on web browsers and are accessed over the internet. They provide services or functionality to users through a web-based interface.
- Artificial Intelligence software: This type of software is designed to solve complex problems using artificial intelligence techniques. Examples include pattern recognition software, artificial neural network algorithms, and software for game playing.

SOFTWARE—NEW CATEGORIES

- **Open world computing**—pervasive, distributed computing
- **Ubiquitous computing**— wireless networks
- **Net-sourcing**—the Web as a computing engine
- **Open source**—"free" source code open to the computing community (a blessing, but also a potential curse!)
- Also …
  - **Data mining**
  - **Grid computing**
  - **Cognitive machines**
  - **Software for nanotechnologies**

LEGACY SOFTWARE : OLDER PROGRAMS

- Software developed **decades ago** and continually modified to meet changes in business requirements and computing platforms
- Software **difficult and costly to maintain**, risky to evolve

THE QUALITY OF LEGACY SOFTWARE

Why must it change?

- software must be **adapted** to meet the needs of new computing environments or technology.
- software must be **enhanced** to implement new business requirements.
- software must be **extended to make it interoperable** with other more modern systems or databases.
- software must be **re-architected** to make it viable within a network environment.

UNIQUE NATURE OF WEB APPS

- The following attributes are encountered in the vast majority of Web Apps:
  - Network intensiveness
  - Concurrency
  - Unpredictable load
  - Performance
  - Availability
  - Data driven
  - Content sensitive
  - Continuous evolution
  - Immediacy
  - Security
  - Aesthetics

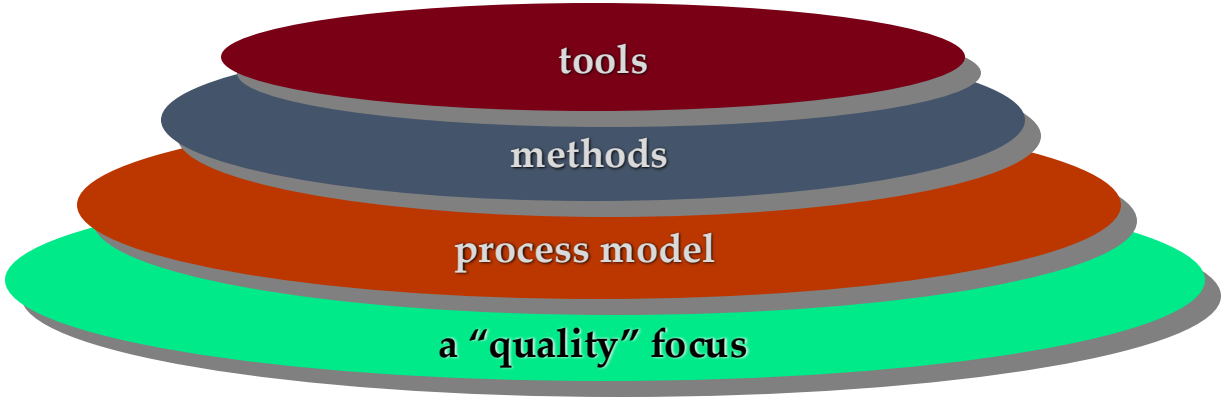
KEY POINTS OF SOFTWARE ENGINEERING

- It follows that a concerted effort should be made to **understand the problem before a software solution** is developed.
- It follows that **design becomes a pivotal activity**.
- It follows that **software should exhibit high quality**.
- It follows that **software should be maintainable**.
- Software in all of its forms and across all of its application domains **should be engineered.**

SOFTWARE ENGINEERING

- The IEEE definition:
  - *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 mentioned in (1).*

A LAYERED TECHNOLOGY

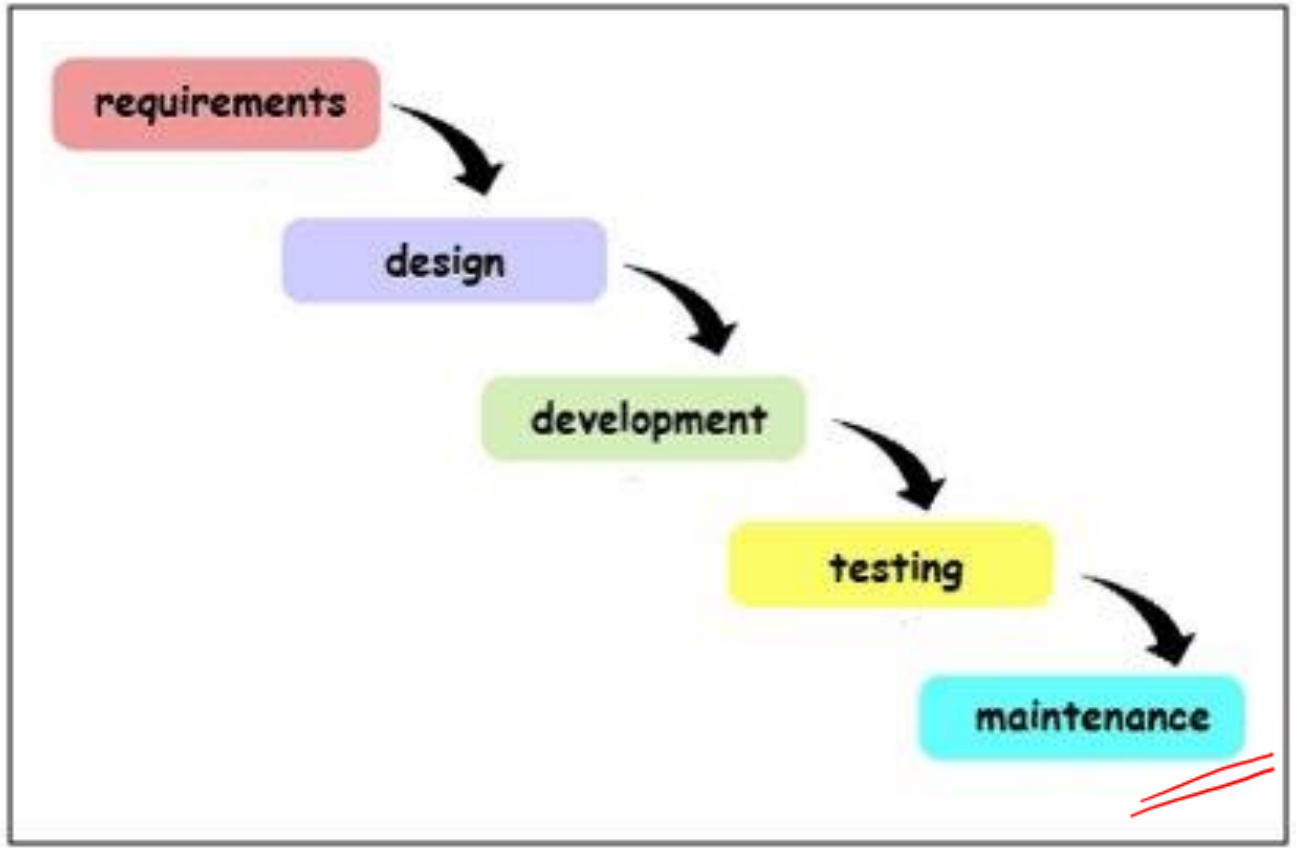


Software Engineering

SOFTWARE ENGINEERING PRACTICE

- Essence of practice
  - *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).
- Hooker's General Principles :
  - **The First Principle: *The Reason It All Exists***
  - **The Second Principle: *KISS (Keep It Simple, Stupid!)***
  - **The Third Principle: *Maintain the Vision***
  - **The Fourth Principle: *What You Produce, Others Will Consume***
  - **The Fifth Principle: *Be Open to the Future***
  - **The Sixth Principle: *Plan Ahead for Reuse***
  - **The Seventh principle: *Think!***

PHASES IN SOFTWARE DEVELOPMENT



THE PRIMARY GOAL OF ANY SOFTWARE PROCESS: HIGH QUALITY

Remember:

High quality ⇒ project timeliness

Why?

Less rework!

SOFTWARE MYTHS

- **Software Management Myths**
- **Software Customer Myths**
- **Developer Myths**

SOFTWARE MANAGEMENT MYTHS

- **Myth:** *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?*

- **Reality:** The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it adaptable? Is it streamlined to improve time-to-delivery while still maintaining a focus on quality? In many cases, the answer to all of these questions is "no."

SOFTWARE MANAGEMENT MYTHS

- **Myth:** *If we get behind schedule, we can add more programmers and catch up (sometimes called the "Mongolian horde" concept).*

- **Reality:** Software development is not a mechanistic process like manufacturing. In the words of Brooks [Bro95]: "adding people to a late software project makes it later." At first, this statement may seem counterintuitive. However, as new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort. People can be added but only in a planned and well coordinated manner.

SOFTWARE MANAGEMENT MYTHS

- **Myth:** *If I decide to outsource the software project to a third party, I can just relax and let that firm build it.*

- **Reality:** If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

CUSTOMER MYTHS

- **Myth:** *A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.*

- **Reality:** Although a comprehensive and stable statement of requirements is not always possible, an ambiguous “statement of objectives” is a recipe for disaster. Unambiguous requirements (usually derived iteratively) are developed only through effective and continuous communication between customer and developer

CUSTOMER MYTHS

- **Myth:** *Software requirements continually change, but change can be easily accommodated because software is flexible.*

- **Reality:** It is true that software requirements change, but the impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has been started), the cost impact is relatively small. However, as time passes, the cost impact grows rapidly—resources have been committed, a design framework has been established, and change can cause upheaval that requires additional resources and major design modifications.

PRACTITIONER'S MYTHS

- **Myth:** *Once we write the program and get it to work, our job is done.*

- **Reality:** Someone once said that "the sooner you begin 'writing code,' the longer it'll take you to get done." Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

PRACTITIONER'S MYTHS

- **Myth:** *Until I get the program "running" I have no way of assessing its quality.*

- **Reality:** One of the most effective software quality assurance mechanisms can be applied from the inception of a project—*the technical review.* Software reviews (described in Chapter 15) are a "quality filter" that have been found to be more effective than testing for finding certain classes of software defects.

PRACTITIONER'S MYTHS

- **Myth:** *The only deliverable work product for a successful project is the working program.*

- **Reality:** A working program is only one part of a software configuration that includes many elements. A variety of work products (e.g., models, documents, plans) provide a foundation for successful engineering and, more important, guidance for software support.

PRACTITIONER'S MYTHS

- **Myth:** *Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.*

- **Reality:** Software engineering is not about creating documents. It is about creating a quality product. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

DEFINITION OF SOFTWARE PROCESS

- A ***process*** is a collection of activities, actions, and tasks that are performed when some work product is to be created
- An ***activity*** strives to achieve a broad objective (e.g., communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.
- An ***action*** (e.g., architectural design) encompasses a set of tasks that produce a major work product (e.g., an architectural design model).
- A ***task*** focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

GENERIC PROCESS MODEL

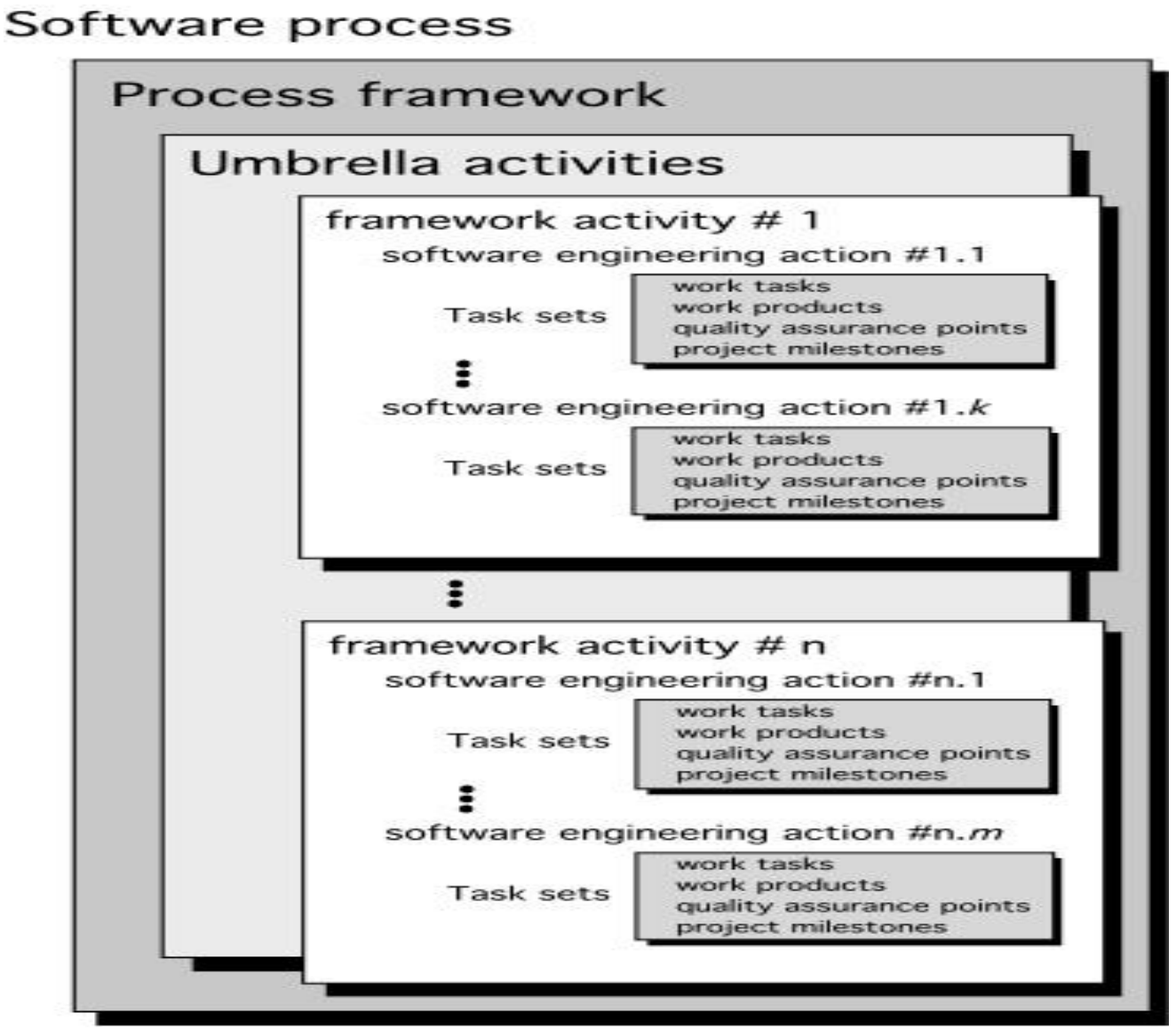
- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
- Construction
  - Code generation
  - Testing
- Deployment

UMBRELLA ACTIVITIES

Complement the five process framework activities and help team manage and control progress, quality, change, and risk.

- **Software project tracking and control:** assess progress against the plan and take actions to maintain the schedule.
- **Risk management:** assesses risks that may affect the outcome and quality.
- **Software quality assurance:** defines and conduct activities to ensure quality.
- **Technical reviews:** assesses work products to uncover and remove errors before going to the next activity.
- **Measurement:** define and collects process, project, and product measures to ensure stakeholder's needs are met.
- **Software configuration management:** manage the effects of change throughout the software process.
- **Reusability management:** defines criteria for work product reuse and establishes mechanism to achieve reusable components.
- **Work product preparation and production:** create work products such as models, documents, logs, forms and lists.

A PROCESS FRAMEWORK



IDENTIFYING A TASK SET

- Before you can proceed with the process model, a key question: what **actions** are appropriate for a framework activity given the nature of the problem, the characteristics of the people and the stakeholders?
- A task set defines the actual work to be done to accomplish the objectives of a software engineering action.
  - **A list of the task to be accomplished**
  - **A list of the work products to be produced**
  - **A list of the quality assurance filters to be applied**

IDENTIFYING A TASK SET

- For example, a small software project requested by one person with simple requirements, the communication activity might encompass little more than a phone all with the stakeholder. Therefore, the only necessary action is phone conversation, the work tasks of this action are:

  1. Make the contact with stakeholder via telephone.

  2. Discuss requirements and take notes.

  3.Organize notes into a brief written statement of requirements.

  4. E-mail to stakeholder for review and approval.

EXAMPLE OF A TASK SET FOR ELICITATION

- The task sets for Requirements gathering action for a **simple** project may include:

1. Make a list of stakeholders for the project.
2. Invite all stakeholders to an informal meeting.
3. Ask each stakeholder to make a list of features and functions required.
4. Discuss requirements and build a final list.
5. Prioritize requirements.
6. Note areas of uncertainty.

EXAMPLE OF A TASK SET FOR ELICITATION

The task sets for Requirements gathering action for a **big** project may include:

1. Make a list of stakeholders for the project.
2. Interview each stakeholders separately to determine overall wants and needs.
3. Build a preliminary list of functions and features based on stakeholder input.
4. Schedule a series of facilitated application specification meetings.
5. Conduct meetings.
6. Produce informal user scenarios as part of each meeting.
7. Refine user scenarios based on stakeholder feedback.
8. Build a revised list of stakeholder requirements.
9. Use quality function deployment techniques to prioritize requirements.
10. Package requirements so that they can be delivered incrementally.
11. Note constraints and restrictions that will be placed on the system.
12. Discuss methods for validating the system.

Both do the same work with different depth and formality. Choose the task sets that achieve the goal and still maintain quality and agility.

PROCESS MODELS

- Prescribe a set of process elements
  - Framework activities
  - Software engineering actions
  - Tasks
  - Work products
  - Quality assurance and
  - Change control mechanisms for each project
- There are a number of process models in operation
- Each process model also prescribes a *workflow*
- Various process models differ in their emphasis on different activities and workflow

ADAPTING A PROCESS MODEL

The process should be **agile and adaptable** to problems. Process adopted for one project might be significantly different than a process adopted from another project (**to the problem, the project, the team, organizational culture**). Among the differences are:

- the overall flow of activities, actions, and tasks and the interdependencies among them
- the degree to which actions and tasks are defined within each framework activity
- the degree to which work products are identified and required
- the manner which quality assurance activities are applied
- the way project tracking and control activities are applied
- the overall degree of detail and rigor with which the process is described
- the degree to which the customer and other stakeholders are involved with the project
- the level of autonomy given to the software team
- the degree to which team organization and roles are prescribed

PRESCRIPTIVE AND AGILE PROCESS MODELS

- The **prescriptive process** models stress detailed definition, identification, and application of process activates and tasks. Intent is to improve system quality, make projects more manageable, make delivery dates and costs more predictable, and guide teams of software engineers as they perform the work required to build a system.

- **Agile process models** emphasize project “agility” and follow a set of principles that lead to a more informal approach to software process. It emphasizes maneuverability and adaptability. It is particularly useful when Web applications are engineered.