CS435: Introduction to Software Engineering

- **Software & Software Engineering**

Software Engineering: A Practitioner's Approach, 7/e
by Roger S. Pressman

Software Engineering 9/e
By Ian Sommerville

Chapter 1

What is Software?

The product that software professionals build and then support over the long term.

Software encompasses: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately store and manipulate information and (3) documentation that describes the operation and use of the programs.

Software products

- Generic products
  - Stand-alone systems that are marketed and sold to **any customer** who wishes to buy them.
  - Examples – PC software such as editing, graphics programs, project management tools; CAD software; MS Word, Photoshop.
- Customized products
  - Software that is commissioned by **a specific customer** to meet their own needs.
  - Examples – embedded control systems, air traffic control software, traffic monitoring systems.

Why Software is Important?

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled (transportation, medical, telecommunications, military, industrial, entertainment,)

Software costs

- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.

- Software costs **more to maintain** than it does to develop. For systems with a long life, maintenance costs may be several times development costs.

- Software engineering is concerned with cost-effective software development.

Characteristics /Features of Software?

- Its characteristics that make it different from other things human being build.

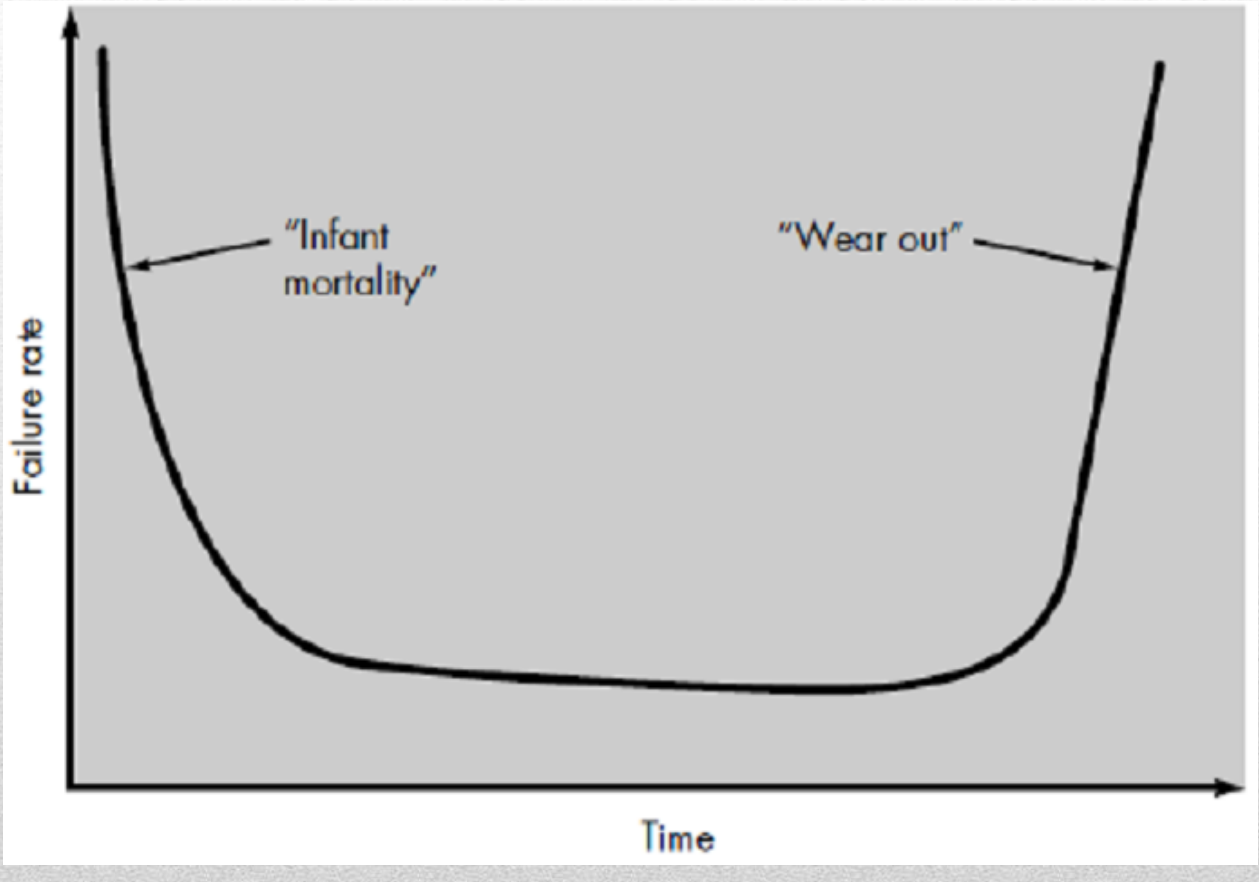
Features of such logical system:

- Software is developed or engineered, it is not manufactured in the classical sense.

- 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).

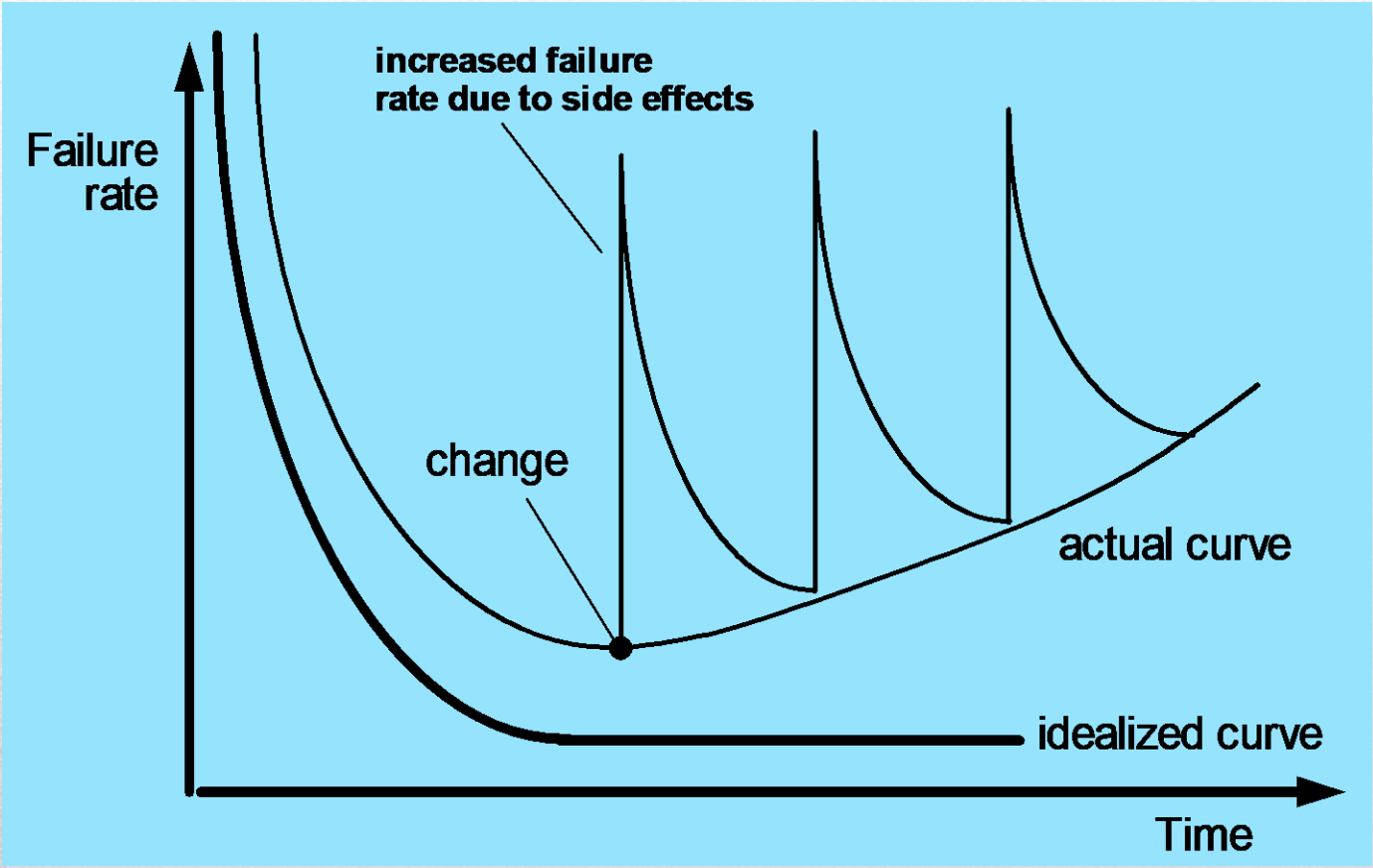
- Although the industry is moving toward component-based construction (e.g. standard screws and off-the-shelf integrated circuits), most software continues to be custom-built. Modern reusable components encapsulate data and processing into software parts to be reused by different programs. E.g. graphical user interface, window, pull-down menus in library etc.

Hardware Wear out



The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life (design or manufacturing defects); defects are corrected and the failure rate drops to a steady-state level for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to wear out.

Wear vs. Deterioration



The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life (design or manufacturing defects); defects are corrected and the failure rate drops to a steady-state level for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to wear out. Software is not susceptible to the environmental maladies that cause hardware to wear out. But it does deteriorate!

Software Applications

- 1. System software: such as operating system components, drivers, networking software, compilers, editors, file management utilities
- 2. Application software: stand-alone programs for specific needs such as MS Word, VLC.
- 3. Engineering/scientific software: Characterized by "number crunching"algorithms. such as astronomy, automotive stress analysis, molecular biology, orbital dynamics, system simulation etc
- 4. Embedded software resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)
- 5. Product-line software designed to provide a specific capability for use by many different customers. It focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)
- 6. WebApps (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.
- 7. AI software uses non-numerical algorithm to solve complex problem. Robotics, expert system, pattern recognition (image and voice), artificial neural networks, and game playing

Software—New Categories

- Open world computing—pervasive, ubiquitous, distributed computing due to wireless networking. How to allow mobile devices, personal computer, enterprise system to **communicate across vast network**.
- Netsourcing—the Web as a computing engine. How to architect simple and sophisticated applications to target end-users worldwide.
- Open source—”free” source code open to the computing community (a blessing, but also a potential curse!)
- Also … (see Chapter 31)
  - Data mining
  - Grid computing
  - Cognitive machines
  - Software for nanotechnologies

Software Engineering Definition

The seminal definition:

> *[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 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 in (1).*

Importance of Software Engineering

- More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.

- It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

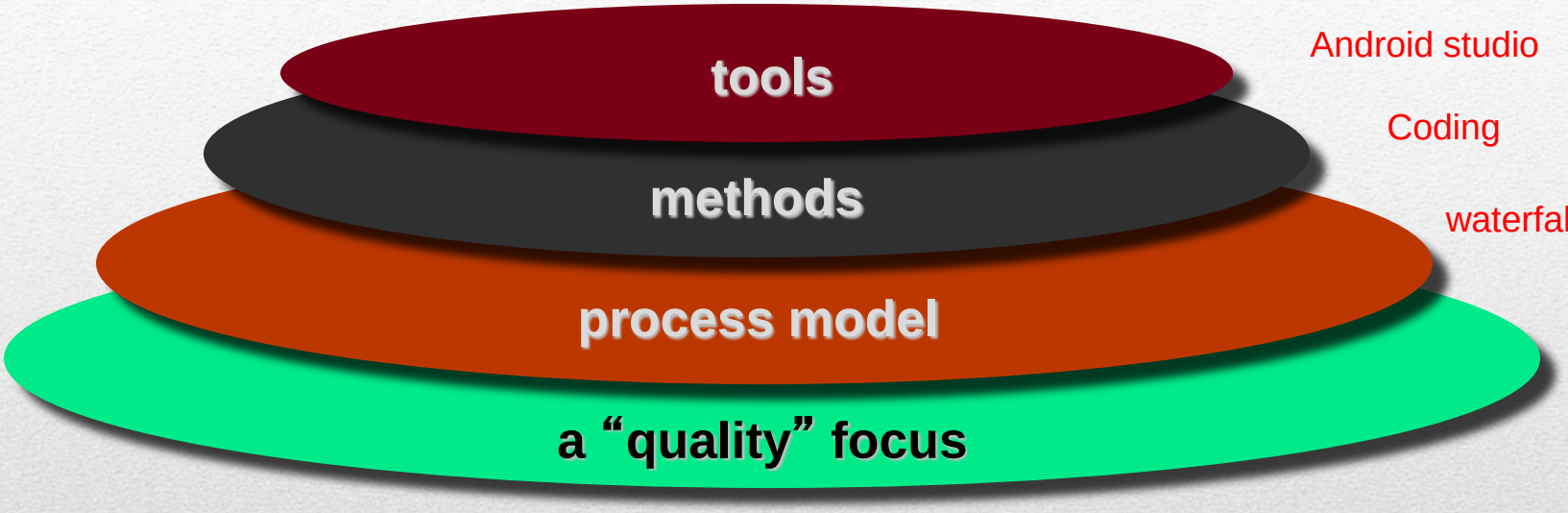
FAQ about software engineering

Question	Answer
What is software?	Computer programs, data structures and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.

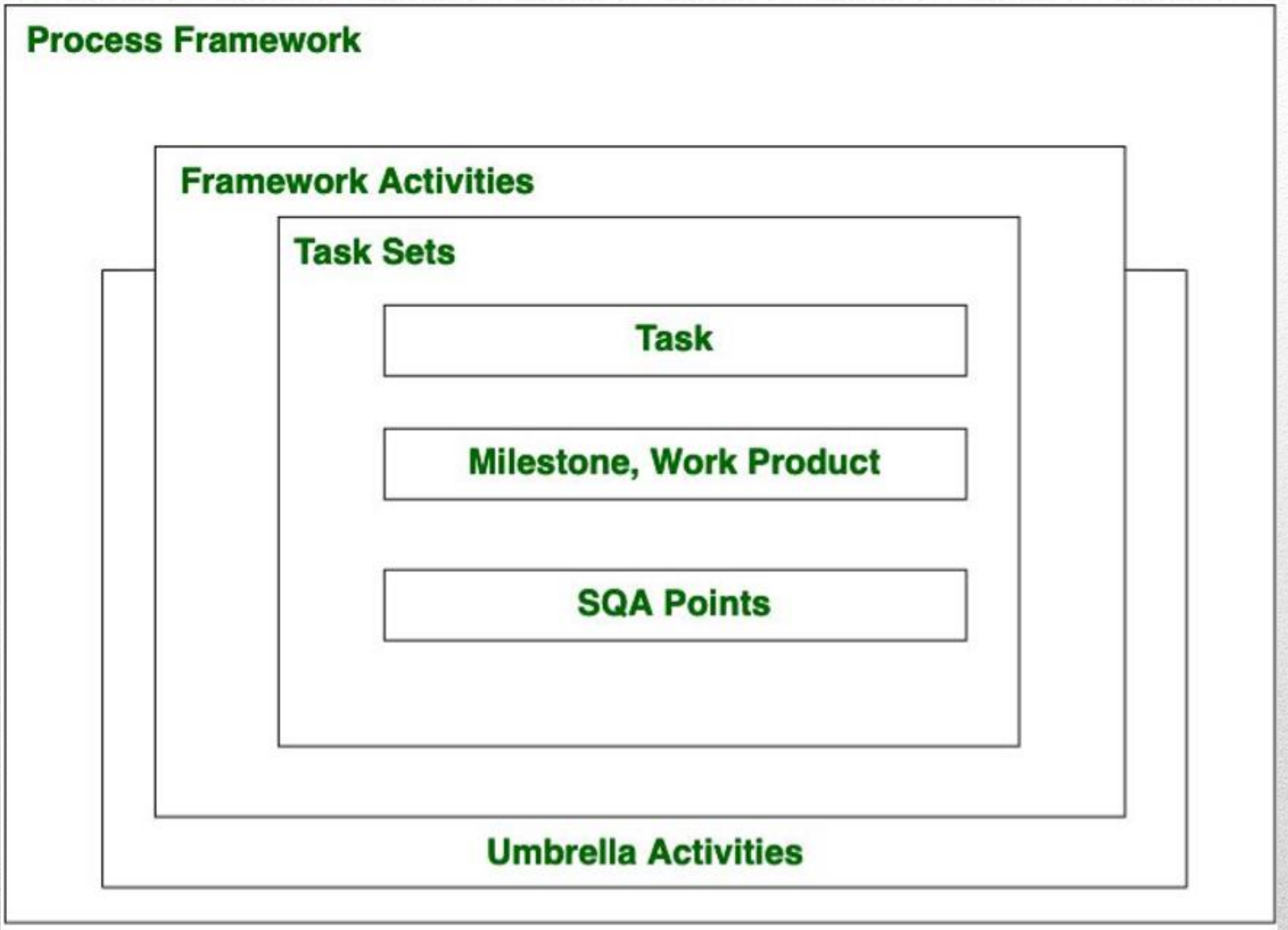
Essential attributes of good software

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

A Layered Technology



- Any engineering approach must rest on organizational commitment to **quality** which fosters a continuous process improvement culture.
- **Process** layer as the foundation defines a framework with activities for effective delivery of software engineering technology. Establish the context where products (model, data, report, and forms) are produced, milestone are established, quality is ensured and change is managed.
- **Method** provides technical how-to’s for building software. It encompasses many tasks including communication, requirement analysis, design modeling, program construction, testing and support.
- **Tools** provide automated or semi-automated support for the process and methods.



Software Process Framework

Software Process

- A process is a collection of activities, actions and tasks that are performed when some work product is to be created. It is **not a rigid prescription** for how to build computer software. Rather, it is an adaptable approach that enables the people doing the work to pick and choose the **appropriate set of work actions** and tasks.

- Purpose of process is to deliver software in a timely manner and with sufficient quality to satisfy those who have sponsored its creation and those who will use it.

Five Activities of a Generic Process framework

- Communication: communicate with customer to understand objectives and gather requirements
- Planning: creates a “map” defines the work by describing the tasks, risks and resources, work products and work schedule.
- Modeling: Create a “sketch”, what it looks like architecturally, how the constituent parts fit together and other characteristics.
- Construction: code generation and the testing.
- Deployment: Delivered to the customer who evaluates the products and provides feedback based on the evaluation.

- These five framework activities can be used to all software development regardless of the application domain, size of the project, complexity of the efforts etc, though the details will be different in each case.

- For many software projects, these framework activities are applied **iteratively** as a project progresses. Each iteration produces a software increment that provides a subset of overall software features and functionality.

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.

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 manner in which 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.
- Unfortunately, there have been times when these objectives were not achieved. If prescriptive models are applied dogmatically and without adaptation, they can increase the level of bureaucracy.

- **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.

The Essence of Practice

- How does the practice of software engineering fit in the process activities mentioned above? Namely, communication, planning, modeling, construction and deployment.
- George Polya outlines the essence of problem solving, suggests:

SRS — 1. *Understand the problem* (communication and analysis). — Analyst

DD — 2. *Plan a solution* (modeling and software design). — UML – Unified Modeling Language Designer

3. *Carry out the plan* (code generation). — developer

tester — 4. *Examine the result for accuracy* (testing and quality assurance).

Understand the Problem

- *Who has a stake in the solution to the problem?* That is, 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 solutions 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 for Software Engineering Practice: important underlying law

Help you establish mind-set for solid software engineering practice (David Hooker 96).

- 1: *The Reason It All Exists: provide values to users*
- 2: *KISS (Keep It Simple, Stupid! As simple as possible)*
- 3: *Maintain the Vision (otherwise, incompatible design)*
- 4: *What You Produce, Others Will Consume* (code with concern for those that must maintain and extend the system)
- 5: *Be Open to the Future* (never design yourself into a corner as specification and hardware changes)
- 6: *Plan Ahead for Reuse*
- 7*: Think! Place clear complete thought before action produces better results.*

Software Myths

Erroneous beliefs about software and the process that is used to build it.

- Affect managers, customers (and other non-technical stakeholders) and practitioners
- Are believable because they often have elements of truth,

but …

- Invariably lead to bad decisions,

therefore …

- Insist on reality as you navigate your way through software engineering

Software Myths - Programmers

- Myth 1: Once we write the program and get it to work, our job is done.
- Reality: the sooner you begin writing code, the longer it will take you to get done. 60% to 80% of all efforts are spent after software is delivered to the customer for the first time.

- Myth 2: Until I get the program running, I have no way of assessing its quality.
- Reality: technical review are a quality filter that can be used to find certain classes of software defects from the inception of a project.

- Myth 3: software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.
- Reality: it is not about creating documents. It is about creating a quality product. Better quality leads to a reduced rework. Reduced work results in faster delivery times.

- Many people recognize the fallacy of the myths. Regrettably, habitual attitudes and methods foster poor management and technical practices, even when reality dictates a better approach.

Software Myths - Managers

- Myth 1: 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: They may exist but are they used and reflect modern engineering.

- Myth 2: If we add more programmers, we can catch up with the schedule.
- Reality: Software Engineering is not a manufacturing process. It makes things worse.

- Myth 3: We can outsource the project to a third party and relax.
- Reality: If you do not know how to manage and control software projects, you will struggle when you outsource them.

Software Myths - Customer

- Myth 1: 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.

- Myth 2: 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.16 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 modification.

How It all Starts

- *SafeHome:*
  - Every software project is precipitated by some business need—
    - the need to correct a defect in an existing application;
    - the need to the need to adapt a ‘legacy system’ to a changing business environment;
    - the need to extend the functions and features of an existing application, or
    - the need to create a new product, service, or system.

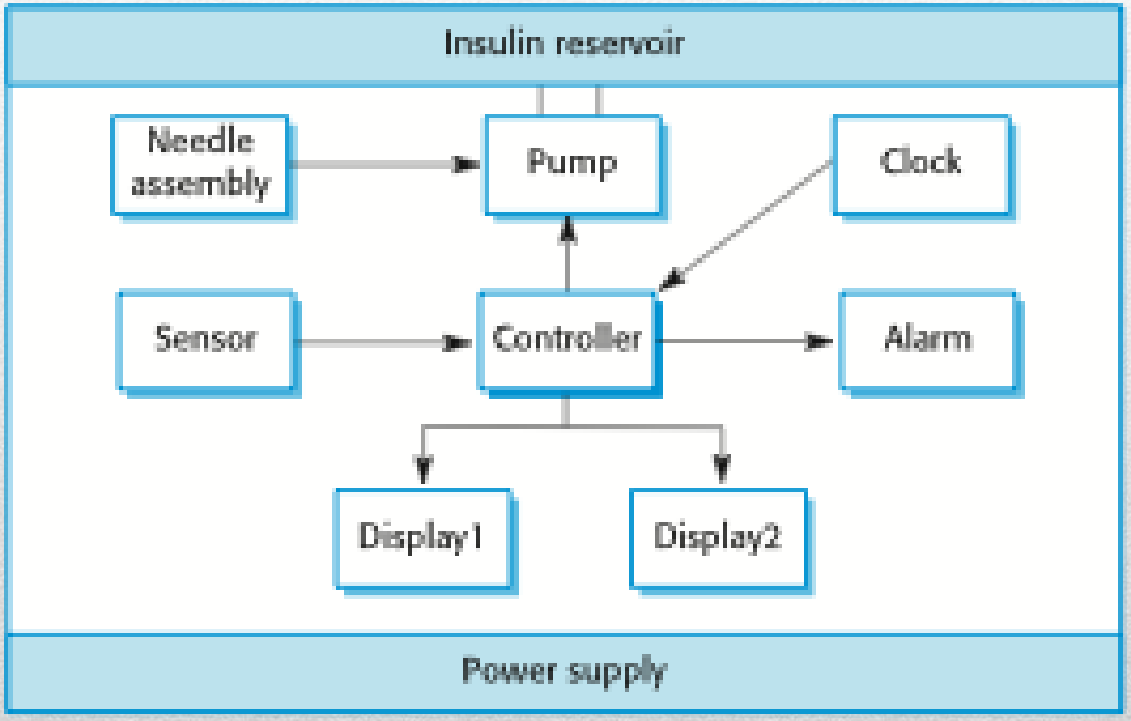
Case studies

- A personal insulin pump
  - An embedded system in an insulin pump used by diabetics to maintain blood glucose control.
- A mental health case patient management system
  - A system used to maintain records of people receiving care for mental health problems.
- A wilderness weather station
  - A data collection system that collects data about weather conditions in remote areas.

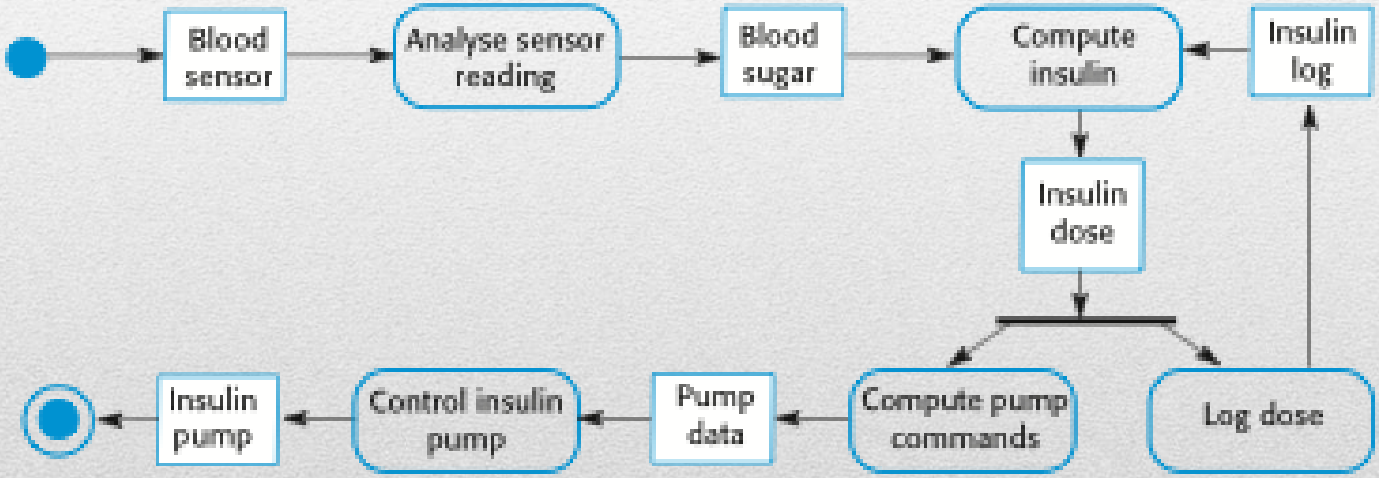
Insulin pump control system

- Collects data from a blood sugar sensor and calculates the amount of insulin required to be injected.
- Calculation based on the rate of change of blood sugar levels.
- Sends signals to a micro-pump to deliver the correct dose of insulin.
- Safety-critical system as low blood sugars can lead to brain malfunctioning, coma and death; high-blood sugar levels have long-term consequences such as eye and kidney damage.

Insulin pump hardware architecture



Activity model of the insulin pump



Essential high-level requirements

- The system shall be available to deliver insulin when required.
- The system shall perform reliably and deliver the correct amount of insulin to counteract the current level of blood sugar.
- The system must therefore be designed and implemented to ensure that the system always meets these requirements.

A patient information system for mental health care

- A patient information system to support mental health care is a medical information system that maintains information about patients suffering from mental health problems and the treatments that they have received.
- Most mental health patients do not require dedicated hospital treatment but need to attend specialist clinics regularly where they can meet a doctor who has detailed knowledge of their problems.
- To make it easier for patients to attend, these clinics are not just run in hospitals. They may also be held in local medical practices or community centres.

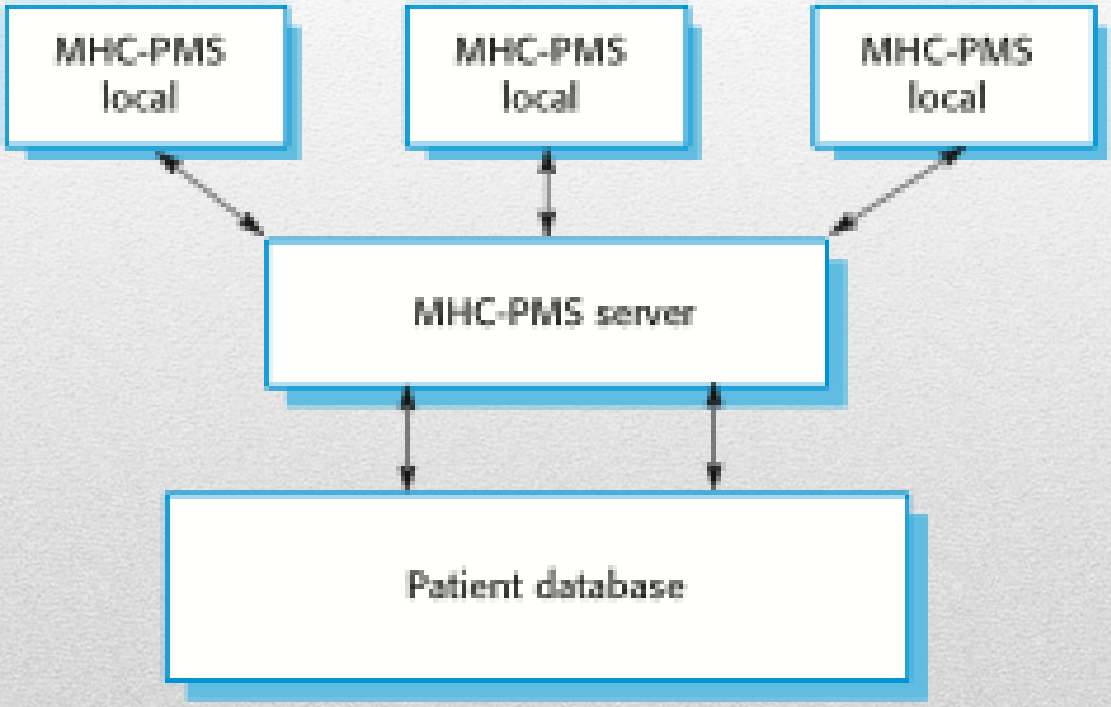
MHC-PMS

- The MHC-PMS (Mental Health Care-Patient Management System) is an information system that is intended for use in clinics.
- It makes use of a centralized database of patient information but has also been designed to run on a PC, so that it may be accessed and used from sites that do not have secure network connectivity.
- When the local systems have secure network access, they use patient information in the database but they can download and use local copies of patient records when they are disconnected.

MHC-PMS goals

- To generate management information that allows health service managers to assess performance against local and government targets.
- To provide medical staff with timely information to support the treatment of patients.

The organization of the MHC-PMS



MHC-PMS key features

- Individual care management
  - Clinicians can create records for patients, edit the information in the system, view patient history, etc. The system supports data summaries so that doctors can quickly learn about the key problems and treatments that have been prescribed.
- Patient monitoring
  - The system monitors the records of patients that are involved in treatment and issues warnings if possible problems are detected.
- Administrative reporting
  - The system generates monthly management reports showing the number of patients treated at each clinic, the number of patients who have entered and left the care system, number of patients sectioned, the drugs prescribed and their costs, etc.

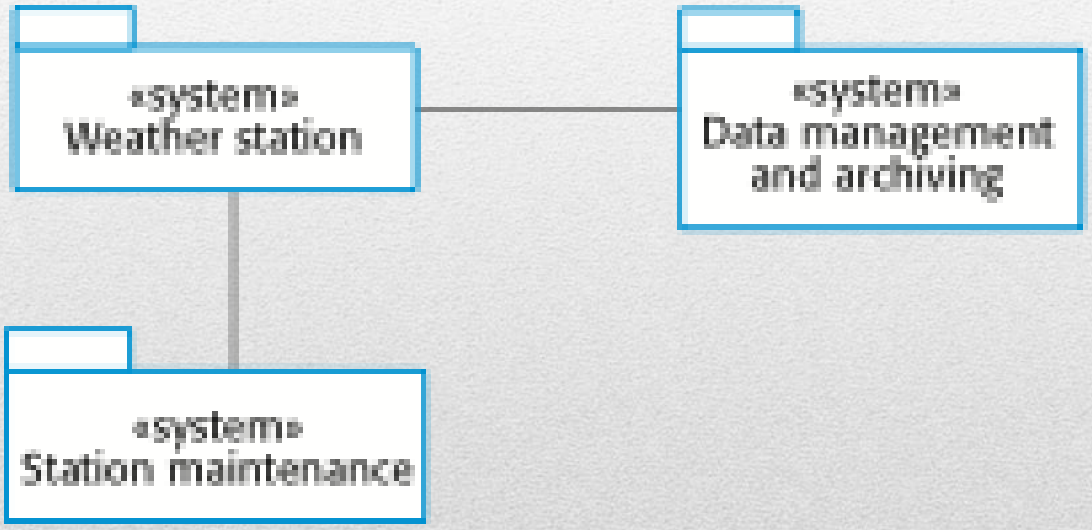
MHC-PMS concerns

- Privacy
  - It is essential that patient information is confidential and is never disclosed to anyone apart from authorised medical staff and the patient themselves.
- Safety
  - Some mental illnesses cause patients to become suicidal or a danger to other people. Wherever possible, the system should warn medical staff about potentially suicidal or dangerous patients.
  - The system must be available when needed otherwise safety may be compromised and it may be impossible to prescribe the correct medication to patients.

Wilderness weather station

- The government of a country with large areas of wilderness decides to deploy several hundred weather stations in remote areas.
- Weather stations collect data from a set of instruments that measure temperature and pressure, sunshine, rainfall, wind speed and wind direction.
  - The weather station includes a number of instruments that measure weather parameters such as the wind speed and direction, the ground and air temperatures, the barometric pressure and the rainfall over a 24-hour period. Each of these instruments is controlled by a software system that takes parameter readings periodically and manages the data collected from the instruments.
-

The weather station's environment



Weather information system

- The weather station system
  - This is responsible for collecting weather data, carrying out some initial data processing and transmitting it to the data management system.
- The data management and archiving system
  - This system collects the data from all of the wilderness weather stations, carries out data processing and analysis and archives the data.
- The station maintenance system
  - This system can communicate by satellite with all wilderness weather stations to monitor the health of these systems and provide reports of problems.

Additional software functionality

- Monitor the instruments, power and communication hardware and report faults to the management system.
- Manage the system power, ensuring that batteries are charged whenever the environmental conditions permit but also that generators are shut down in potentially damaging weather conditions, such as high wind.
- Support dynamic reconfiguration where parts of the software are replaced with new versions and where backup instruments are switched into the system in the event of system failure.