What is Software Engineering?

- Its components
  - Product - result
  - Process - how you get to the result
  - Problem - the forgotten and critical element.

- Product - Goal - a viable, merchandizable object. Note different from science where goal knowledge.
  - In software engineering the produce has related software
  - The software is new or revised and targeted to the product.
Process - various factors - management, development engineers, tools, computer aided design, development and support software, etc.

Within a time and budget environment

With minimum errors - including no critical errors

With the ability to expand, upgrade, etc., i.e., modify software to enhancing product value

Problem - a most critical element. With a poorly defined problem it is impossible to develop a good solution. No solution means a product of little, no or unrecognizable value.
What are the course objectives?

⇒ To provide a basic recognition of concepts, methods and situations. To allow you, when in a software development situations, to be able to better cope and contribute to its success.

⇒ Though study, example and problem assignments to give you a feel of software engineering and how tools and techniques apply.

⇒ The course is unable to simulate with any degree of real world ‘ism’. This is a factor of life in the educational environment.
Definition of Software Engineering

⇒ “The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is the application of engineering “principles” to software.”
⇒ It leaves out the concept that this “software” has value.

⇒ “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 machine.”
⇒ It embodies some intrinsic concepts of value but fails to address the ideas of product value. Under-emphases discipline and approach.

⇒ Orderly methodology for producing quality software
Software must evolve

Like all other products, software must evolve to keep value.

Example - Y2K problem which occasioned much consternation.

Evolution occurs due to new knowledge, better machinery, new methods, needs, complexity and systems.

Software experiences

Software disasters

Horror stories - doesn’t work, over budget, late, etc. Many were attempts by government.

Evolution creates a need for new and better software. Example networking.
Software characteristics

⇒ Software doesn’t wear out. Failures to “design” flaws.
⇒ Not susceptible to the environmental maladies like those causing hardware to “wear” out.
⇒ Is susceptible to changes which cause new “design” flaws.
⇒ Modifications (maintenance and upgrade) can create new or uncover old “design” flaws.

⇒ Software is mostly custom built.
⇒ New software need being reduced by OS, standard methods, libraries, frameworks, components, tools, etc. which provide better technology for software.
⇒ Software becomes “embedded”, e.g. Microsoft Office Upgrading as opposed to completely new development.
Software characteristics (cont’d)

⇒ Software manufacture is almost trivial
  ⇒ “Manufacture” implies reproduction to transport to the site where it is to be used.

⇒ “Manufacture” implies targeting to users’ environments which differ. Has become easier by the use of common and ‘virtual’ interfaces
Software characteristics (cont’d)

⇒ Software systems - is there a crisis?
  ⇒ Many needs now well understood - systems, business, engineering and scientific, pc, network and Internet, etc. Activity is to incorporate new technology such as wireless.

⇒ Still need for software for new problems, new environments, etc. Places like space, where weight, power, reliability, etc., are critical.

⇒ Some organizations (government), FAA and IRS are so bureaucratic, it is impossible for them to participate in successful software development to solve their problems.
Software myths - suggest you read with a grain (box) of salt.

⇒ Myths can be strawmen used by “salesmen”.

⇒ A general statement of the objectives is sufficient to begin writing programs
  ⇒ Implies a step-by-step process to discern objectives.
  ⇒ Situations where problem, solution, and experience is lacking. Apollo is a typical example.

⇒ Software creates voluminous and unnecessary documentation.
  ⇒ This can be true or false.
  ⇒ “Good” software engineering recognizes documentation is communication and “corporate” memory.
The Process

⇒ Keys to a successful process
  ⇒ Page 19 - “... Software development is a social “learning” process. ...”. Keys to successful social processes? Communication and understanding.
  ⇒ Process designed to enable communication. Team members must support this activity. All must communicate to enable success.

⇒ Layered Techniques
  ⇒ Layering - a mechanism for modularization. Layer are responsible for internal matters and an interface to others.
  ⇒ Divide and conquer - a mechanism where intractable problems are divided into smaller solvable one.
  ⇒ Software process use both layered and subdivision modelling techniques.
Modeling

⇒ A model is a mechanism for understanding, analysis and obtaining results pertinent to a problem. Like layering and subdivision it allows for problem breakdown.

⇒ Looking at the problem from different directions. Important in its overall understanding

⇒ Modeling - The word model implies that it is “an approximation” to the real world. As a result you know that, in total, some results are “inaccurate and/or wrong”. Hence, care must be taken to understand what results are useful. Problems occur due approximations in simulations, mathematical modeling, etc.
CS 451 Software Eng. Survey

Process Models

Task structure diagram
Task Description

⇒ Tasks, Deliverables, Quality assurance

⇒ Tasks - work broken up into recognizable, measurable units, milestones, subtasks, etc. Mapped into units with time and costs assigned.

⇒ Deliverables - items represent the successful completion of the tasks. May be software units, demonstrations, test outcomes, documents, user manuals, etc.

⇒ Quality assurance - management and measurement which reflects the “correctness” and “effectiveness” of delivered items.

⇒ Umbrella activities - overseeing activities.
Process Models - Maturity models

- Maturity models - won’t be cover

- Attempt to measure an organization’s infrastructure to ascertain to what degree they have the technology and management in place to produce quality software.

- Probably similar to the quality assurance programs ISO 900* - which aims for 6 sigma defects. Has been applied to manufacturing, software engineering, etc.
Diagrams show process flow - applicable to all subproblem levels
Process staging models

Problem solving loops, prototyping paradigm are examples of closed loop models.

⇒ They illustrate the feedback nature needed in the software process

⇒ They omit the evolutionary linear progression with distinct states.
Process Models - staging (cont’d)

⇒ Linear sequential models. - Incremental model, spiral model, concurrent development model

⇒ Linear stages - Problem definition & analysis, design, development and coding, test, support and upgrade.

⇒ In the spiral model - three axis models which go through most of the linear stages but a number of times.
  ⇒ They add linear stages of customer evaluation and customer feedback
  ⇒ They also add project steps in a third axis, steps such as concept, product, enhancement and maintenance phases
Linear Sequential Models

- Analysis
- Design
- Code
- Test

Data

Process

Appl.

Test & turnover

Waterfall model - Rapid App. Devel.

Note: I have added feedback at stages
Process Models (cont’d)

Spiral Diagrams

Customer Communications

Maint. Loop

Test & Evaluation

Planning

Risk Analysis

Development & Coding

Concept Loop

Dev. Loop

Enhance. Loop
Boehm’s WINWIN model adds major identification of the customer’s activities early in the project.

Probably the best in integrating customer activity.
Process Models - concurrent

⇒ Model shows software activity
⇒ Incorporates the activities of subproblems.
⇒ Active tasks as opposed to stages
In general they omit

- The need for feedback - communication

- That backward steps can occur; i.e., progression is not always strictly forward

- That difference subproblems may be in distinctly different stages

- In some projects, some stages may not exist.
Process - Tools

⇒ Development Tools

⇒ Components - modular (and modifiable) objects that can readily added during the development (design and coding stages).

⇒ Based on object paradigm like C++ and Java

⇒ Had the original name of Frameworks

⇒ Examples include Visual C, Borland Builder, Microsoft Active Objects, Kylix, etc.

⇒ Can be packages, dynamically linked libraries, object linking & embedding, COMs, etc.
Process - Tools

⇒ Formal Methods
  ⇒ “Languages”, graphs, charts, etc. - mathematical specification techniques which have sufficient formalism that they can be mapped directly into code

⇒ Difficult to use since they require a great deal of mathematical insight into the problem. Not every thing can be described symbolically

⇒ We will treat these ideas in much greater detail later on in the course to show exactly how they fit in the development scheme.
The process provides the “glue” to carry out the project. It integrates project tasks with project members and management, provides the resources and “tool”, through monitoring maintains schedules and delivers the product and specified (possibly modified) output.

Process models illustrate the organization and flow occurring in typical software engineering processes.