CS2212
Introduction to Software Engineering

Software and Software Engineering

Ask Questions Live
cs1.ca/ask

What is Software Engineering?

1. It is an area of Computer Science which relates to **techniques**, **methods**, **practices** and **tools** for 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) in order to develop **high quality** products **within the specified budget and time**.

[IEEE Standard 610.12]

The Nature of Software - Definitions

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.
3. Documentation that describes the operation and use of the programs. => user guides, etc

The Nature of Software – What is Software?

- Software is **developed or engineered**; it is not manufactured in the classical sense like physical objects (like hardware)
- Software **doesn't "wear out"** over time from use or environmental conditions
- Although the industry is moving toward component-based construction, most software continues to be custom-built without readily swappable parts

Hardware Failure Curve

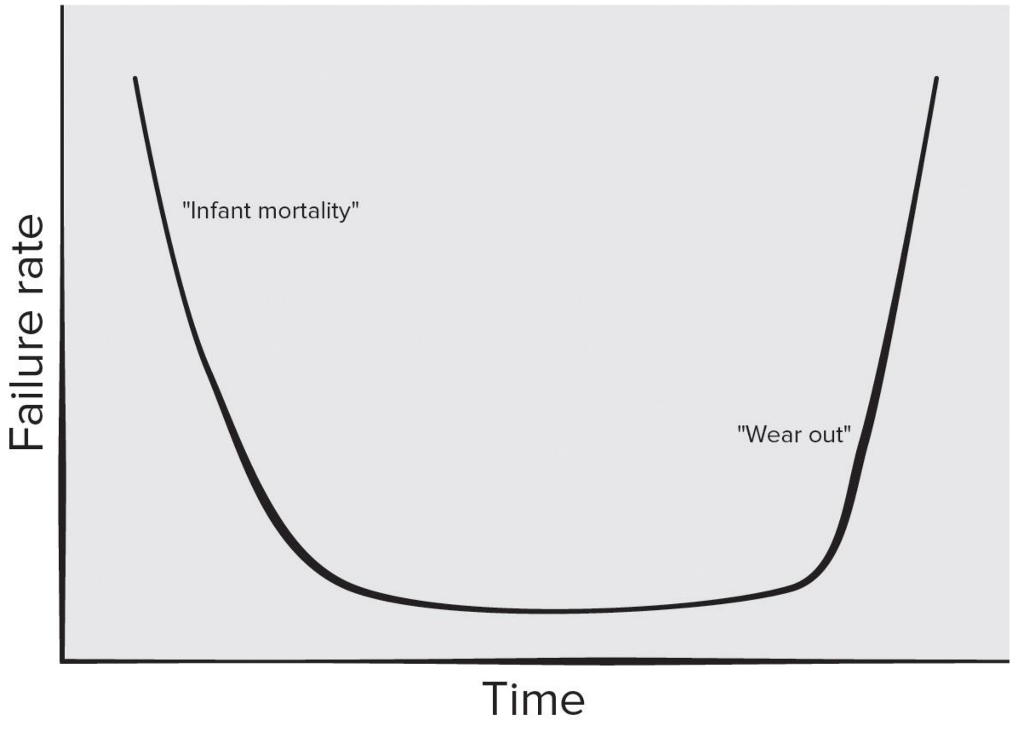


Figure 1.1 from your textbook.

The Nature of Software – What is Software?

- Software is **developed or engineered**; it is not manufactured in the classical sense like physical objects (like hardware)
- Software **doesn't "wear out"** over time from use or environmental conditions
- Although the industry is moving toward component-based construction, most software continues to be custom-built without readily swappable parts

Software Failure Curve

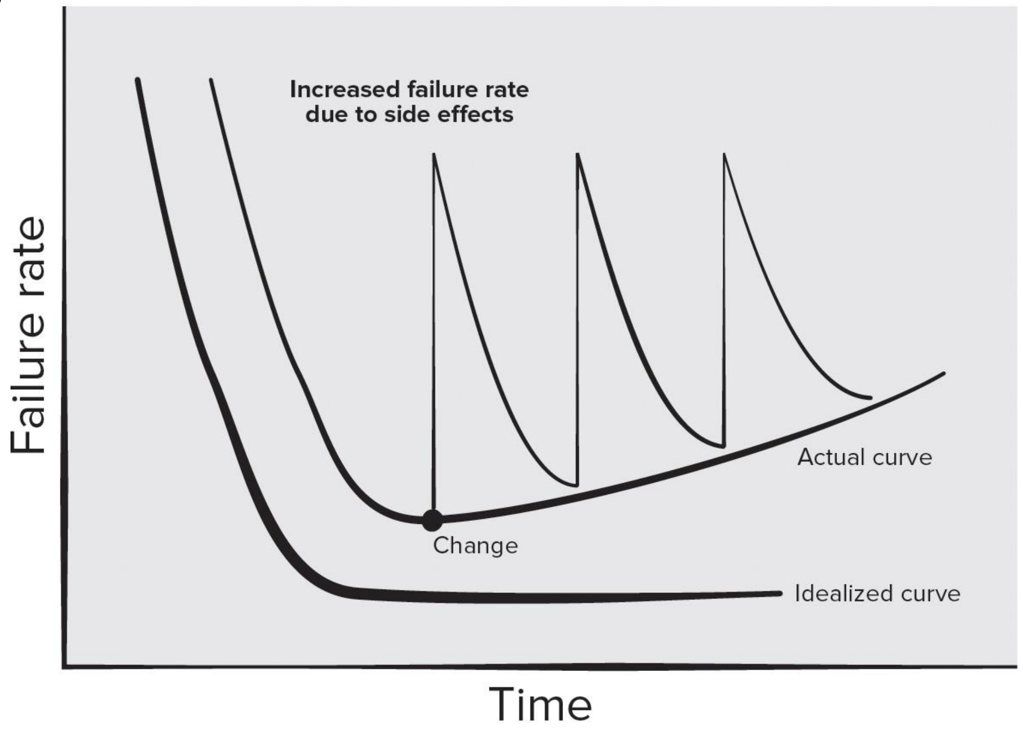


Figure 1.2 from your textbook.

Types of Software

Seven broad categories of Software Applications:

- System Software
- Application Software
- Engineering/Scientific Software
- Embedded Software
- Product-line Software
- Web/Mobile Applications
- Artificial Intelligence Software

memory process

[illegible]

1. the software development have to go through coding, debugging, adjusting

2. the developers have to consider many different cases and make the software adapt to these cases. So they have to spend a lot of time ~~[illegible]~~ testing the software. Also, they might try different ways of implementation.

3. because our costomers might think in a different way. and might use the software not in our expected way

4. thing are changing and developing: we could adjust and better our software. i.e. fixing bugs, adding features, or increasing running speed.

Types of Software

Seven broad categories of Software Applications:

- System Software
- Application Software
- Engineering/Scientific Software
- Embedded Software
- Product-line Software
- Web/Mobile Applications
- Artificial Intelligence Software

System Software

- Collections of programs written to service other programs.
- E.g. Compilers, editors, file management utilities, operating system components, drivers, networking software, telecommunications software, and more.

macOS

Windows 11

Types of Software

Seven broad categories of Software Applications:

- System Software
- **Application Software**
- Engineering/Scientific Software
- Embedded Software
- Product-line Software
- Web/Mobile Applications
- Artificial Intelligence Software

Application Software

- Stand-alone programs that solve a specific business need.
- Process business or technical data in a way that facilitates business operations or decision making.

Types of Software

Seven broad categories of Software Applications:

- System Software
- Application Software
- **Engineering/Scientific Software**
- Embedded Software
- Product-line Software
- Web/Mobile Applications
- Artificial Intelligence Software

Engineering/Scientific Software

- Broad array of “number-crunching” or data science programs.
- **Example uses:** astronomy, volcanology, automotive stress analysis, orbital dynamics, computer-aided design, modeling consumer spending habits, genetic analysis, meteorology, and much more.

MATLAB AUTOCAD PSpice®

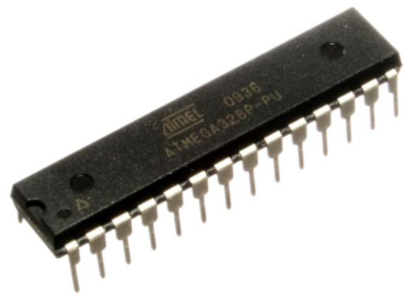
Types of Software

Seven broad categories of Software Applications:

- System Software
- Application Software
- Engineering/Scientific Software
- **Embedded Software**
- Product-line Software
- Web/Mobile Applications
- Artificial Intelligence Software

Embedded Software

- Resides within a product or system and is used to implement and control features and functions for the end user and the system itself.
- End user rarely interacts with it directly.
- **Examples:** key pad control for a microwave, digital functions in an automobile (dashboard, fuel control, lights), tv remote control, drone control, etc.



Types of Software

Seven broad categories of Software Applications:

- System Software
- Application Software
- Engineering/Scientific Software
- Embedded Software
- **Product-line Software**
- Web/Mobile Applications
- Artificial Intelligence Software

Product-line Software

- Composed of reusable components and designed to provide specific capabilities for use by many different customers.
- Collection of similar software systems from a shared set of software assets.

Office 365

Architecture, Engineering & Construction Collection

Adobe® Creative Cloud™

Types of Software

Seven broad categories of Software Applications:

- System Software
- Application Software
- Engineering/Scientific Software
- Embedded Software
- Product-line Software
- **Web/Mobile Applications**
- Artificial Intelligence Software

Web/Mobile Applications

- **Web Applications:** Software-as-a-service that is delivered through your web browser.
- **Mobile Applications:** Application software for mobile devices (e.g. smart phones).

Types of Software

Seven broad categories of Software Applications:

- System Software
- Application Software
- Engineering/Scientific Software
- Embedded Software
- Product-line Software
- Web/Mobile Applications
- **Artificial Intelligence Software**

Artificial Intelligence Software

- Makes use of heuristics to solve complex problems that are not amenable to regular computation or straightforward analysis.
- Often based around classifying new instances of data based on past examples (training data).
- **Applications:** robotics, decision-making systems, pattern recognition (image/voice/text), machine learning, theorem proving, and game playing.

How Do We Make “Good” Software?

By applying a systematic, disciplined, and quantifiable approach to its development. That is applying Engineering principles to the discipline.

Software Engineering Layers

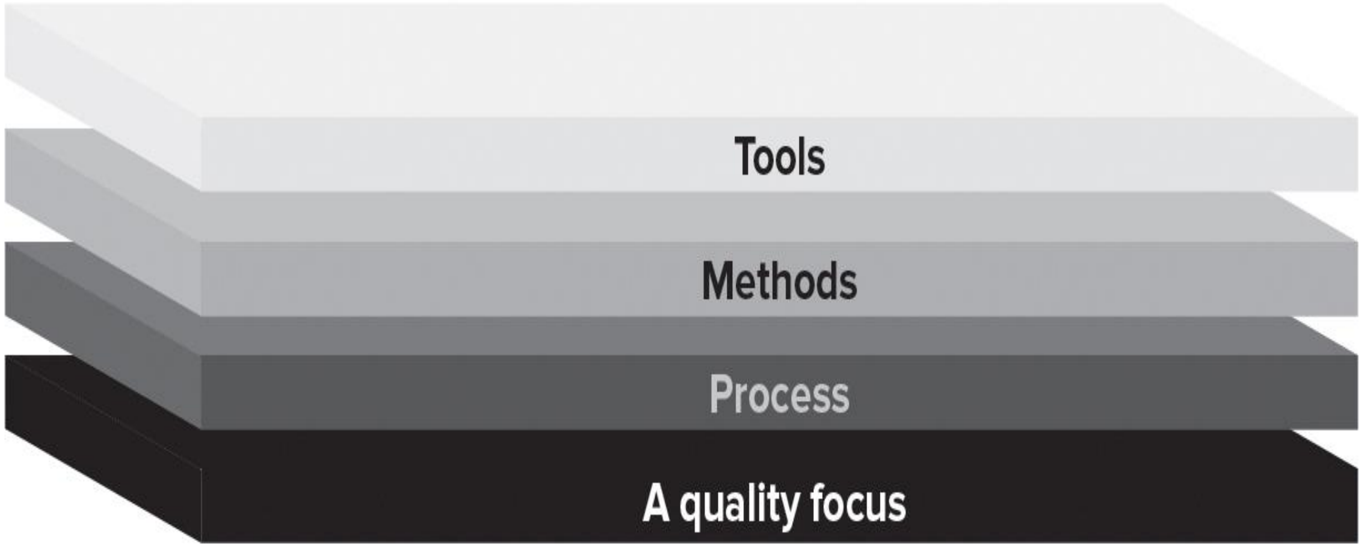
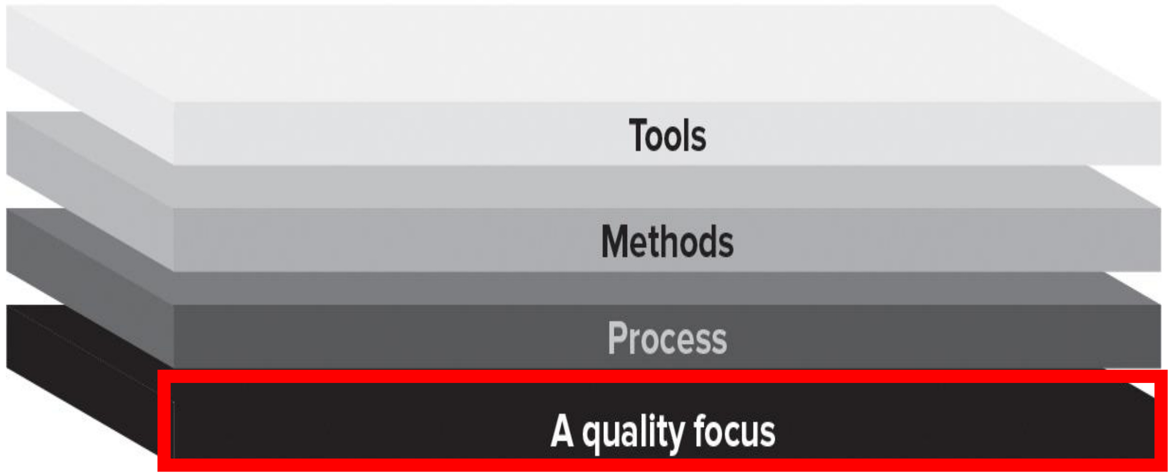


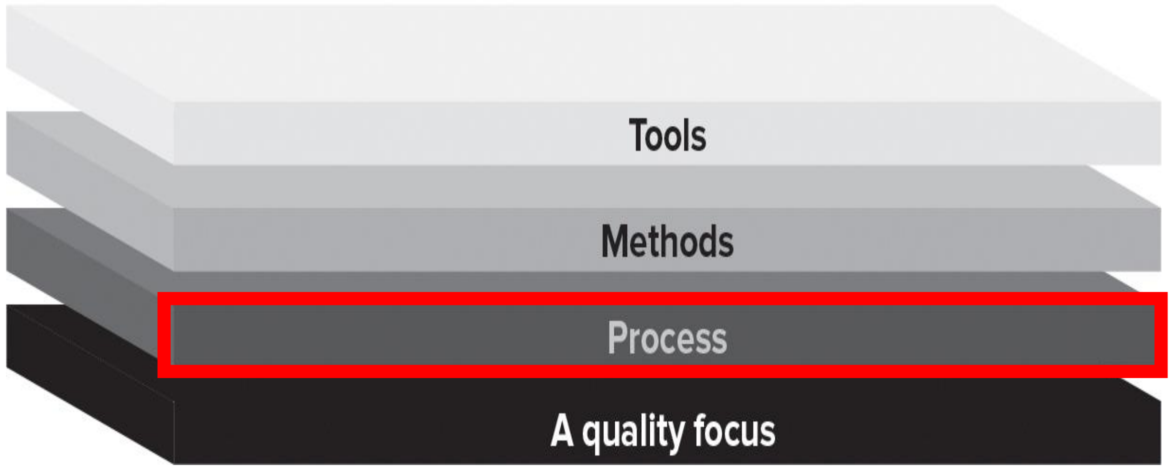
Figure 1.3 from your textbook.

Software Engineering Layers



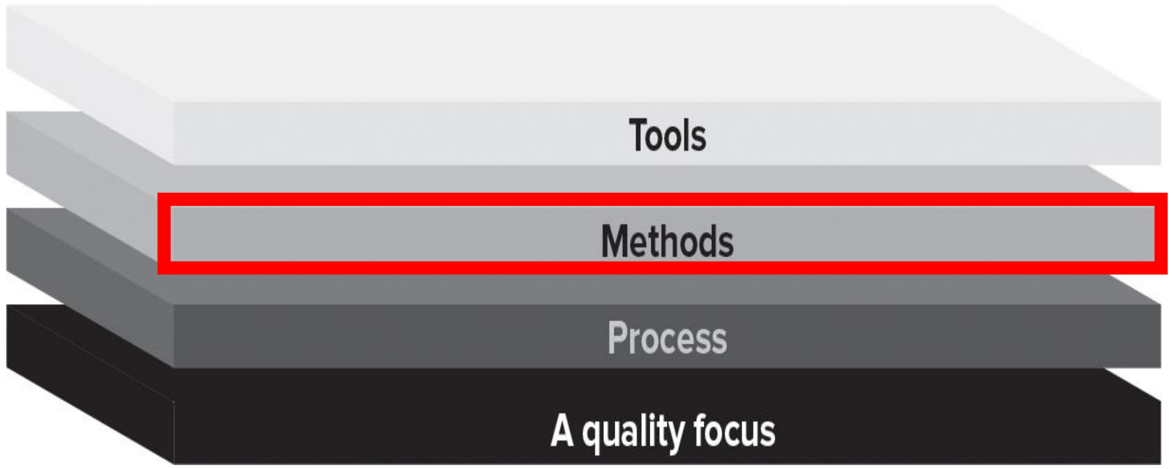
- Software Engineering requires a commitment to quality.
- Will only work if the team or organization applying it commits to a culture of continuous improvement and focus on quality.
- This focus is the bedrock of software engineering.

Software Engineering Layers



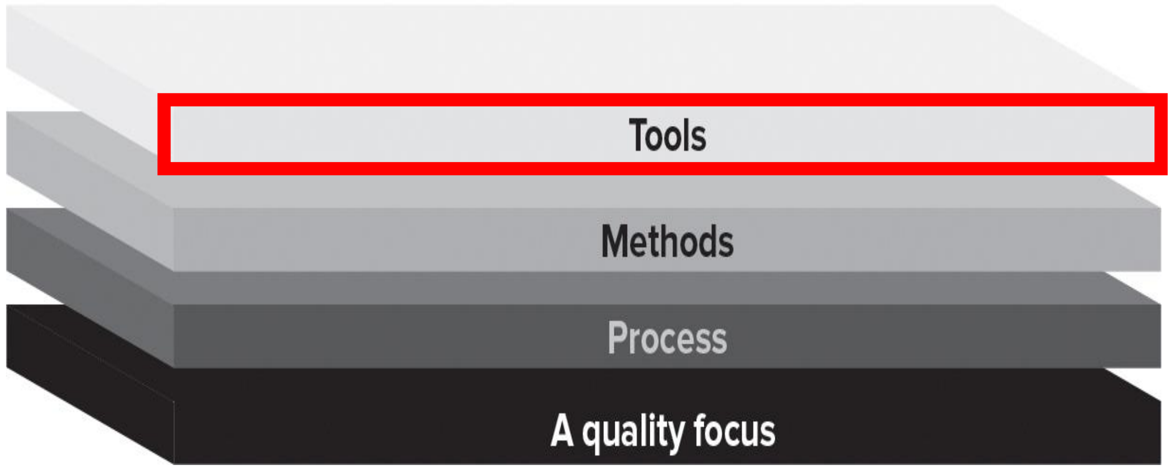
- Process is the foundational Layer of software engineering and holds everything together.
- Process defines a framework for the effective and timely delivery of software engineering technology.
- Software processes enable control of software projects and guide the technical methods applied, the work products produced, ensures quality, and helps manage change.

Software Engineering Layers



- Methods provide the technical “how-to’s” for building software.
- Encompass a broad array of tasks, including communication, requirements analysis, design modeling, program construction, testing and support.
- Rely on basic principles that govern each area of the technology.

Software Engineering Layers



- Tools aid in automation and support of software engineering processes and methods.
- Help ensures quality and enables effective communication.
- When tools are used to support software development it is called: ***computer-aided software engineering***

The Process Framework - Definitions

- **Process:** a collection of **activities**, **actions**, and **tasks** that are performed when some work product is to be created.
- **Activity:** strives to achieve a broad objective (e.g. communication with stakeholders) and is applied regardless of the application domain, size, complexity, etc.
- **Action:** encompasses a set of **tasks** (e.g. architectural design) that produce a major work product (e.g. an architectural model).
- **Tasks**: focuses on small, but well-defined objectives (e.g. conducting a unit test) that produces a tangible outcome.

The Process Framework

- A process framework establishes the foundation for a complete **process** by identifying a small number of **framework activities** that are applicable to **all software projects**.
- The process framework also contains a set of **umbrella activities** that complement the **framework activities** and are applicable across the entire software process.

The Process Framework

There are five key activities applicable to all software projects:

- Communication
- Planning
- Modeling
- Construction
- Deployment

The Process Framework

There are five key activities applicable to all software projects:

- Communication
- Planning
- Modeling
- Construction
- Deployment

Before technical work can commence, it is important to communicate and collaborate with customers and other **stakeholders** to understand their objectives and to identify what they need and want as outcomes.

The Process Framework

There are five key activities applicable to all software projects:

- Communication
- **Planning**
- Modeling
- Construction
- Deployment

Software projects are complex so planning is required to create a “map” to guide the team.

This plan describes the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

The Process Framework

There are five key activities applicable to all software projects:

- Communication
- Planning
- **Modeling**
- Construction
- Deployment

To better understand the problem and how it is going to be solved, a software engineer creates models of the software's requirements and the design that will achieve those requirements.

The Process Framework

There are five key activities applicable to all software projects:

- Communication
- Planning
- Modeling
- Construction
- Deployment

What is designed must be built!

Construction entails both code generation (**either manual or automated**) as well as the **testing** required to uncover errors in the code.

The Process Framework

There are five key activities applicable to all software projects:

- Communication
- Planning
- Modeling
- Construction
- **Deployment**

The software is delivered to the customer who evaluates the delivered product, provides feedback based on the evaluation, and (assuming everything is acceptable) puts the software into use.

Umbrella Activities

In addition to framework activities, **umbrella activities** are also applied throughout a software project. Typical **umbrella activities** include:

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

Umbrella Activities

In addition to framework activities, **umbrella activities** are also applied throughout a software project. Typical **umbrella activities** include:

- **Software project tracking and control**
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

Allows the team to assess progress against the project plan and take necessary actions to maintain the schedule.

Umbrella Activities

In addition to framework activities, **umbrella activities** are also applied throughout a software project. Typical **umbrella activities** include:

- Software project tracking and control
- **Risk management**
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

Assess risks that may affect the output of the project or the quality of the product, and make plans to mitigate those risks.

Umbrella Activities

In addition to framework activities, **umbrella activities** are also applied throughout a software project. Typical **umbrella activities** include:

- Software project tracking and control
- Risk management
- **Software quality assurance**
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

Defines and conducts the activities necessary to ensure software quality.

Umbrella Activities

In addition to framework activities, **umbrella activities** are also applied throughout a software project. Typical **umbrella activities** include:

- Software project tracking and control
- Risk management
- Software quality assurance
- **Technical reviews**
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

Assess software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.

Umbrella Activities

In addition to framework activities, **umbrella activities** are also applied throughout a software project. Typical **umbrella activities** include:

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- **Measurement**
- Software configuration management
- Reusability management
- Work product preparation and production

Defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs.

Umbrella Activities

In addition to framework activities, **umbrella activities** are also applied throughout a software project. Typical **umbrella activities** include:

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
- **Software configuration management**
- Reusability management
- Work product preparation and production

Manages the effects of change throughout the software process.

Umbrella Activities

In addition to framework activities, **umbrella activities** are also applied throughout a software project. Typical **umbrella activities** include:

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- **Reusability management**
- Work product preparation and production

Defines criteria for work product reuse and establishes mechanisms to achieve reusable components.

Umbrella Activities

In addition to framework activities, **umbrella activities** are also applied throughout a software project. Typical **umbrella activities** include:

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- **Work product preparation and production**

Encompasses the activities required to create work products such as models, documents, logs, forms, lists, and so on.

Process Adaptation

- Software engineering processes should not be a rigid prescription that must be followed dogmatically.
- Rather, it should be **agile** and adaptable.
- A process for one project might be significantly different than a process adopted for another problem, project, team, or organization.
- Processes should always be tailored to the current situation. The processes we discuss in this course are not meant to be unbreakable rules, but rather should be flexible and adaptable.

Software Engineering Practice

The Essence of Problem Solving

1. Understand the Problem
   (communication and analysis)
2. Plan a Solution
   (modeling and software design)
3. Carry Out the Plan
   (code generation)
4. Examine the Result for Accuracy
   (testing and quality assurance)

Software Engineering Practice

The Essence of Problem Solving

1. **Understand the Problem**
   (communication and analysis)
2. Plan a Solution
   (modeling and software design)
3. Carry Out the Plan
   (code generation)
4. Examine the Result for Accuracy
   (testing and quality assurance)

Understand the Problem

- We seldom spend enough time truly understanding a problem before attempting to solve it.
- Need to take some time to reflect and answer some questions first:
  - *Who has a stake in the solution to the problem?*
  - *What are the unknowns?*
  - *Can the problem be compartmentalized?*
  - *Can the problem be represented graphically?*

Software Engineering Practice

The Essence of Problem Solving

1. Understand the Problem
   (communication and analysis)
2. **Plan a Solution**
   (modeling and software design)
3. Carry Out the Plan
   (code generation)
4. Examine the Result for Accuracy
   (testing and quality assurance)

Plan a Solution

- Even with better understanding of the problem, we should not be tempted to start coding.
- Need to do some planning beforehand and reflect on some questions first:
  - *Have you seen similar problems before?*
  - *Has a similar problem been solved?*
  - *Can subproblems be defined?*
  - *Can you represent a solution in a manner that leads to effective implementation?*

Software Engineering Practice

The Essence of Problem Solving

1. Understand the Problem
   (communication and analysis)
2. Plan a Solution
   (modeling and software design)
3. **Carry Out the Plan**
   (code generation)
4. Examine the Result for Accuracy
   (testing and quality assurance)

Carry Out the Plan

- With a plan, we can now move on to building the software.
- Along the way, there are more questions to answer:
  - *Does the solution conform to the plan?*
  - *Is each component part of the solution provably correct?*

Software Engineering Practice

The Essence of Problem Solving

1. Understand the Problem
   (communication and analysis)
2. Plan a Solution
   (modeling and software design)
3. Carry Out the Plan
   (code generation)
4. Examine the Result for Accuracy
   (testing and quality assurance)

Examine the Result for Accuracy

- While you cannot be sure that your solution is perfect, you can still design sufficient testing to uncover as many issues as possible.
- Even more questions to address here:
  - *Is it possible to test each component part of the solution?*
  - *Does the solution produce results that conform to the data, functions, and features that are required?*

General Principles

Seven principles that focus on software engineering practice as a whole:

1. The Reason It All Exists
2. KISS (Keep It Simple, Stupid!)
3. Maintain the Vision
4. What You Produce, Others Will Consume
5. Be Open to the Future
6. Plan Ahead for Reuse
7. Think!

General Principles

Seven principles that focus on software engineering practice as a whole:

1. **The Reason It All Exists**
2. KISS (Keep It Simple, Stupid!)
3. Maintain the Vision
4. What You Produce, Others Will Consume
5. Be Open to the Future
6. Plan Ahead for Reuse
7. Think!

A software system exists for one reason: ***to provide value to its users***. All decisions throughout a project should be made with this in mind.

General Principles

Seven principles that focus on software engineering practice as a whole:

1. The Reason It All Exists
2. **KISS (Keep It Simple, Stupid!)**
3. Maintain the Vision
4. What You Produce, Others Will Consume
5. Be Open to the Future
6. Plan Ahead for Reuse
7. Think!

Simplicity facilitates having a more easily understood and easily maintained software system. Simple does not mean *"quick and dirty"*; in fact it often takes a lot of thought and work over time to simplify. The more elegant (and better) designs are usually the more simple ones.

General Principles

Seven principles that focus on software engineering practice as a whole:

1. The Reason It All Exists
2. KISS (Keep It Simple, Stupid!)
3. **Maintain the Vision**
4. What You Produce, Others Will Consume
5. Be Open to the Future
6. Plan Ahead for Reuse
7. Think!

A clear vision is essential to the success of a software project. Without one, a project can become a patchwork of incompatible designs that are barely held together … and such things more often than not break in time.

General Principles

Seven principles that focus on software engineering practice as a whole:

1. The Reason It All Exists
2. KISS (Keep It Simple, Stupid!)
3. Maintain the Vision
4. **What You Produce, Others Will Consume**
5. Be Open to the Future
6. Plan Ahead for Reuse
7. Think!

Software is seldom built and used in a vacuum. Someone else will use, maintain, document, or otherwise depend on being able to understand your work. So, always be sure to specify, design, and implement knowing that someone else will have to understand what you are doing.

General Principles

Seven principles that focus on software engineering practice as a whole:

1. The Reason It All Exists
2. KISS (Keep It Simple, Stupid!)
3. Maintain the Vision
4. What You Produce, Others Will Consume
5. **Be Open to the Future**
6. Plan Ahead for Reuse
7. Think!

A system with a long lifetime has more value. True "industrial strength" software systems must endure, and so these systems must be ready to adapt to change. Never design yourself into a corner that puts limits on what you do; instead, always ask "what if" and prepare for future possibilities.

General Principles

Seven principles that focus on software engineering practice as a whole:

1. The Reason It All Exists
2. KISS (Keep It Simple, Stupid!)
3. Maintain the Vision
4. What You Produce, Others Will Consume
5. Be Open to the Future
6. **Plan Ahead for Reuse**
7. Think!

Reuse of designs and code saves time and effort, but it requires investment to pay off in the future. Reusability is not automatic and needs time and effort to achieve through forethought, consideration, and planning.

General Principles

Seven principles that focus on software engineering practice as a whole:

1. The Reason It All Exists
2. KISS (Keep It Simple, Stupid!)
3. Maintain the Vision
4. What You Produce, Others Will Consume
5. Be Open to the Future
6. Plan Ahead for Reuse
7. **Think!**

Placing clear, complete thought before action almost always produces better results. When you think about something, you are more likely to do it right. Even if something goes wrong along the way, it becomes a valuable learning experience. **When clear thought has gone into a system value tends to come out.**