Lecture Notes for Lecture 1 of CS 5500 (Foundations of Software Engineering) for the Spring 2021 session at the Northeastern University San Francisco Bay Area Campuses.

Introduction to Software Engineering

Philip Gust,
Clinical Instructor
Department of Computer Science

http://www.ccis.northeastern.edu/home/pgust/classes/cs5500/2021/Spring/index.html

Introduction to Software Engineering

Introduction

- In this first lecture we will begin studying the foundations of software engineering and exploring the factors that brought the field into existence.
- The purpose of this first lecture is to survey the field of software engineering to motivate the individual topics that we will discuss in more depth during the course.

Introduction to Software Engineering

Definitions of Software Engineering

- Software engineering does not have a single fixed definition. Before we study it, it may be useful to look at the ways it has been defined by people and organizations.
- In 1972, German computer pioneer Friedrich (Fritz) Bauer described software engineering as

  "The establishment and use of sound engineering principles to economically obtain software that is reliable and works on real machines efficiently."

Introduction to Software Engineering

Definitions of Software Engineering

- In 1990, IEEE Computer Society's *Software Engineering Body of Knowledge* defined software engineering as

  "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software."

Introduction to Software Engineering

Definitions of Software Engineering

- Distinguished American computer scientist Barry Bohem, a pioneer in the field said that software engineering involves

  "the practical application of scientific knowledge to the creative design and building of computer programs. It also includes associated documentation needed for developing, operating, and maintaining them."

Introduction to Software Engineering

Definitions of Software Engineering

- According to a recent definition in Wikipedia:

  "Software engineering is a profession and field of study dedicated to designing, implementing, and modifying software so that it is of higher quality, more affordable, maintainable, and faster to build."

Introduction to Software Engineering

Definitions of Software Engineering

- These definitions of software engineering include ideas that give us a starting point for getting a handle on the term.
  - systematic and disciplined
  - quantifiable
  - sound engineering principals
  - real-world applications
  - software lifecycles
  - associated documentation
  - study of the process
- As we will see, software engineering encompasses all these ideas, and others not mentioned in any of these definitions.

Introduction to Software Engineering

Historical Context

- During the 1960s, software emerged as a separate discipline with the commercialization of computers and large-scale development of business, scientific, engineering, and systems applications.
  - IBM began estimating the size of large projects in “acres of coders”, using a standardized amount of space for each.
- Many projects failed or went over-budget, and the software was unreliable and expensive to maintain as project sizes increased.
- Customer requirements and demand for new software grew faster than the ability to generate it without consistent training, reliable processes and better tools.
- The result was a crisis for the computer industry. A more systematic and disciplined approach to creating software was needed.

Introduction to Software Engineering

Historical Context

- The solution required transforming unorganized coding efforts into a software engineering discipline.
- The first conference with “software engineering” in its name was held in 1968 under the leadership of Fritz Bauer.
- Independently in 1968, Apollo Guidance Computer (AGC) software lead Margaret Hamilton first named what her team was doing “software engineering”.
- The Software Engineering Institute (SEI) opened in 1984 at Carnegie Mellon University to study software engineering processes and management.

Margaret Hamilton with AGC listing

Introduction to Software Engineering

Historical Context

- Margaret Hamilton provided details about how she came up with the term “software engineering”:

  “When I first came up with the term, no one had heard of it before, at least in our world. It was an ongoing joke. They liked to kid me about my radical ideas.

  “It was a memorable day when one of the most respected hardware gurus explained to everyone in a meeting that he agreed with me that the process of building software should also be considered an engineering discipline, just like with hardware.

  “It was memorable not because of his acceptance of the new 'term' per se, but because we had earned his and the acceptance of the others in the room as being in an engineering field in its own right.”

  https://en.wikipedia.org/wiki/Margaret_Hamilton_(software_engineer)

Hamilton receives Presidential Medal of Freedom in 2016

Introduction to Software Engineering

Historical Context

- By the late 1970s, the discipline of software engineering had matured to the point that companies were adopting its principles for their software projects.
- Some larger companies like Hewlett-Packard even formed separate corporate engineering groups to facilitate adoption by its divisions.
- The 1980s saw the use of computer-aided software engineering (CASE) tools and automation of software engineering processes.
- Beginning in the 1990s, software organizations began to emphasize management of software processes, from requirements tracking to management and automated analysis of quality metrics.
- In 2018 the 40th International Conference on Software Engineering celebrated the 50th anniversary of software engineering as a field.

Introduction to Software Engineering

Historical Context

- In the 2020s, organizations will begin experimenting with automation and engineering process management that goes beyond imperative algorithms and simple rules.
- Using large-scale expert-systems and ML techniques, organizations can begin optimizing software engineering management by
  - Understanding and tracking requirements and specifications against software under development.
  - Automatically reorganizing tasks and assignments among members of a team.
  - Analyzing code and predicting potential errors to make the testing process more effective and reliable.
  - Leading reviews of specifications and code and tracking outcomes to ensure more uniform results.

Introduction to Software Engineering

Motivation for Software Engineering

- The motivation for software engineering arises because of changing environments and requirements in which software runs.
  - **Large software:** As the size of the software becomes large, a software engineering process provides the tools to create it.
  - **Scalability:** If a software process is not based on engineering concepts, it might be easier to re-create new software than scale an existing one.
  - **Adaptability**: if a software development process is not based on engineering concepts, it might be easier to create new software than adapt an existing one.

Ref: https://www.guru99.com/what-is-software-engineering.html

Introduction to Software Engineering

Motivation for Software Engineering

- The motivation for software engineering arises because of changing environments and requirements in which software runs.
  - **Cost:** As the cost of hardware continues to decrease, the cost of software will reman high if software engineering principles are not followed.
  - **Dynamic Nature:** As conditions change, requirements change too and a flexible software engineering process can help adapt the software to meet the needs.
  - **Quality Management:** A better software engineering process provides a better and higher quality software product.

Ref: https://www.guru99.com/what-is-software-engineering.html

Introduction to Software Engineering

Relationship With Other Disciplines

- Software engineering overlaps with a number of other disciplines.
  - **Computer Science**: Gives the scientific foundation for the software as electrical engineering mainly depends on physics.
  - **Management Science:** Software engineering is labor-intensive work which demands both technical and managerial control. Therefore, it is widely used in management science.
  - **Operations Research:** Software engineering helps to optimize a process by quickly examining options and determining which one is best.
  - **Economics:** In this sector, software engineering helps you in resource estimation and cost control. Computing system must be developed, and data should be maintained regularly within a given budget.
  - **System Engineering:** Most software is part of a much larger system (e.g. software in a reactor, aircraft flight software). Software engineering methods can be applied to the study of these types of systems.

Introduction to Software Engineering

Challenges of Software Engineering

- Some challenges for software engineering include.
  - In safety-critical areas such as space, aviation, nuclear power, etc. the cost of software failure can be massive because lives are at risk.
  - Market demands for fast turnaround time.
  - Increased complexity of software needed for new applications.
  - Rapidly changing conditions requiring software to rapidly adapt as well.
  - Diversity of software systems that need to communicate with each other.

Introduction to Software Engineering

Examples of Software Engineering Failures

- Therac-25 (1985)

  “Between June 1985 and January 1987, a computer-controlled radiation therapy machine called Therac-25 massively overdosed six people. These accidents have been described as the worst in the 35-year history of medical accelerators”

https://www.computer.org/csdl/magazine/co/2017/11/mco2017110008/13RUxAStVR

Introduction to Software Engineering

Examples of Software Engineering Failures

- Ariane 5 Explosion (1997)

  "It took the European Space Agency 10 years and $7 billion to produce Ariane 5 a giant rocket capable of hurling a pair of three-ton satellites into orbit with each launch, and intended to give Europe overwhelming supremacy in the commercial space business.

  "All it took to explode that rocket less than a minute into its maiden voyage last June, scattering fiery rubble across the mangrove swamps of French Guiana, was a small computer program trying to stuff a 64-bit number into a 16-bit space"

  https://iansommerville.com/software-engineering-book/case-studies/ariane5/

Introduction to Software Engineering

Examples of Software Engineering Failures

- Mars Climate Orbiter (1999)

  "For nine months, the Mars Climate Orbiter was speeding through space and speaking to NASA in metric. But the engineers on the ground were replying in non-metric English.

  "It was a mathematical mismatch that was not caught until after the $125 million spacecraft, a key par of NASA's Mars exploration program, was sent crashing too low and too fast into the Martian atmosphere The craft has not been heard from since."

  https://mars.jpl.nasa.gov/msp98/news/mco991110.htm

Introduction to Software Engineering

Example of a Software Engineering Success

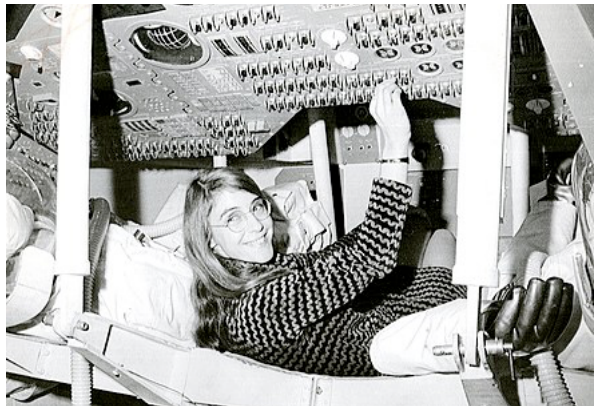
- Apollo 11 Lunar Module (1969)

  "In one of the critical moments of the Apollo 11 mission, the Apollo Guidance Computer and on-board flight software averted an abort of the Moon landing.

  "Three minutes before the lunar lander reached the Moon's surface, several computer alarms were triggered. The on-board flight software captured these 'never supposed to happen' displays. Software lead Margret Hamilton had prepared the AGC software for this situation years before.

  "If the AGC hadn't recognized this problem and taken recovery actions, Apollo 11 would not have successfully landed on the moon."

  https://en.wikipedia.org/wiki/Margaret_Hamilton_(software_engineer)



Hamilton in Apollo Command Module

Introduction to Software Engineering

Attributes of Software Products

- The characteristics of any software product include attributes displayed by the product when it is installed and put in use.
- These are not the services that are provided by the product. Instead, they are related to the product dynamic behavior and the use made of the product.
- Examples of these attributes are:
  - maintainability
  - dependability
  - efficiency
  - usability

Introduction to Software Engineering

Attributes of Software Products

- The relative importance of these attributes varies from one software system to another. Optimizing them is very challenging. For example, offering a better UI can reduce system efficiency.

Product Characteristics	Description
Maintainability	The software should evolve to meet the changing demands of the clients.
Dependability	Dependability includes various characteristics. Dependable software should never cause any physical or economic damage at the time of system failure.
Efficiency	The software application should not overuse system resources like memory and processor cycle.
Usability	The software application should have specific UI and documentation.

Introduction to Software Engineering

Characteristics of Good Software

- Any software should be judged by what it offers and what are the methods which help you to use it. The software must satisfy the following characteristics:
  - Operational
  - Transitional
  - Maintenance
- We will look at each of them in turn to discover important characteristics of good software developed by software professionals using sound software engineering practices.

Introduction to Software Engineering

Characteristics of Good Software

- Operational
  - This characteristic lets us know about how well software works in the operations which can be measured on:
    - Cost
    - Efficiency
    - Usability
    - Dependability
    - Correctness
    - Functionality
    - Safety
    - Security

Introduction to Software Engineering

Characteristics of Good Software

- Transitional
  - This is an essential characteristic when the software is moved from one platform to another:
    - Interoperability
    - Reusability
    - Portability
    - Adaptability

Introduction to Software Engineering

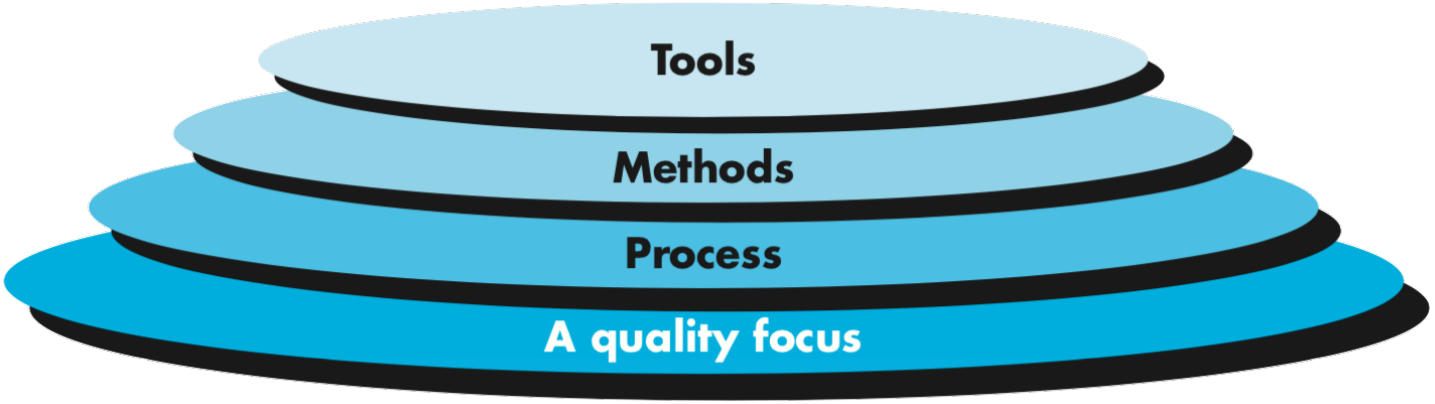
Characteristics of Good Software

- Maintenance
  - This characteristic talks about how well software has the capabilities to adapt itself in the quickly changing environment:
    - Flexibility
    - Maintainability
    - Modularity
    - Scalability

Introduction to Software Engineering

Nature of Software Engineering

- Software engineering is a layered technology.



Source: Software Engineering A Practitioner's Approach 8th edition

Introduction to Software Engineering

Nature of Software Engineering

- Quality.
  - Any engineering approach, including software engineering, must rest on a bedrock of quality that supports the discipline.
  - Total quality management, Six Sigma, and similar philosophies foster a continuous process improvement culture.
  - It is this culture that ultimately leads to the development of increasingly more effective approaches to software engineering.

Introduction to Software Engineering

Nature of Software Engineering

- Process.
  - The foundation for software engineering is the process layer. This layer holds the technology layers together and enables rational and timely development of computer software.
  - Process defines a framework for effective delivery of software engineering technology. It forms the basis for management control of software projects.
  - Process establishes the context where technical methods are applied, work products (models, documents, data, reports, etc.) are produced, milestones established, quality ensured, and change managed.

Introduction to Software Engineering

Nature of Software Engineering

- Methods.
  - Software engineering methods provide the technical how-to's for building software.
  - Methods encompass an array of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support.
  - Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

Introduction to Software Engineering

Nature of Software Engineering

- Tools.
  - Tools provide the capabilities required to implement software engineering methods, manage software engineering processes, and facilitate and track software quality
  - Tools embed best practices for creating software and help automate the process to create software more consistently, more reliably, and more rapidly in response to requirements.
  - Tools that can be applied across multiple phases of a software development process are especially useful because they can integrate information across the phases and provide more useful feedback.

Introduction to Software Engineering

Software Engineering Process

- A *process* is a collection of *activities*, *actions*, and *tasks* that are performed when some work product is to be created. That is, a process defines who is doing what, when, and how to reach a goal.
- An *activity* seeks to achieve a broad objective (e.g., communication with customers and stakeholders).
  - It 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* encompasses a set of tasks that produce a major work product (e.g., architectural model).
- A *task* focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

Introduction to Software Engineering

Software Engineering Process

- A process is not a rigid prescription for how to build computer software.
- Rather, a process is an adaptable approach that enables the people doing the work (the software team) to pick and choose the appropriate set of work actions and tasks.
- The intent is always to deliver software in a timely manner and with sufficient quality to satisfy the stakeholders who have sponsored its creation and customers who will use it.

Introduction to Software Engineering

Software Engineering Process

- A *process framework* establishes the foundation for a complete software engineering process.
- The framework identifies a small number of *framework activities* that are applicable to all software projects, regardless of their size or complexity.
- A *generic process framework* for software engineering includes five generic framework activities:
  - Communication
  - Planning
  - Modeling
  - Construction
  - Deployment

Introduction to Software Engineering

Software Engineering Process

- Here is a list of generic framework activities and their purposes.

framework activity	description
Communication	Before any technical work can commence, communicate and collaborate to understand customer's and stakeholders' objectives and gather requirements
Planning	Create a software project plan that describes the technical tasks, the risks, the resources, the work products, and a work schedule.
Modeling	Create a preliminary sketch and refine it in greater detail to understand the requirements and the design.
Construction	Generate code (either manual or automated) and test to uncover errors in the code.
Deployment	Deliver software to the customer, who evaluates the delivered product and provides feedback.

Introduction to Software Engineering

Software Engineering Process

- The five generic framework activities can be used during the creation of simple programs, web applications, or large systems. Details will be different, but the framework activities are the same.
- For many software projects, framework activities are applied iteratively as a project progresses.
- Each iteration produces a software *increment* that provides customers and stakeholders with a subset of overall software features and functionality.
- As each increment is produced, the software becomes more and more complete.

Introduction to Software Engineering

Software Engineering Process

- *Umbrella framework activities* are applied throughout a project to help a team manage and control progress quality, change and risk.

umbrella activity	description
Tracking and control	Assess progress against project plan and take necessary action to maintain schedule.
Risk management	Assess risks that may affect project outcome or quality.
Quality assurance	Defines and conduct activities to ensure software quality.
Technical reviews	Assess work products to uncover and remove errors.
Measurement	Define and collect measures that assist team in delivering software that meets customer's and stakeholders' needs.
Configuration mgmt.	Manages effects of change throughout software process.
Reusability mgmt.	Define criteria and mechanisms for work product reuse.
Prep. and production	Create work products such as models, documents, etc..

Introduction to Software Engineering

Software Engineering Process

- A software engineering process is not rigid; it should be agile and adaptable. A process adapted for one project may be significantly different than one adapted for another. Difference may include:
  - Overall flow of activities, actions, and tasks and inter-dependencies.
  - Degree to which actions and tasks are defined in each framework activity.
  - Degree to which work products are identified and required.
  - Manner in which quality assurance activities are applied.
  - Manner in which project tracking and control activities are applied.
  - Overall degree of detail and rigor with which the process is described.
  - Degree to which customers and stakeholders are involved with the project.
  - Level of autonomy given to the software team.
  - Degree to which team organization and roles are prescribed.

Introduction to Software Engineering

Software Engineering Practice

- Once we have created a software engineering process, here are the steps required to put it into 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).
- These set of steps lead to a series of essential questions.

Introduction to Software Engineering

Software Engineering Practice

- 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?

Introduction to Software Engineering

Software Engineering Practice

- 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 required data, functions, and features?
  - 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?

Introduction to Software Engineering

Software Engineering Practice

- 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 the algorithm?

Introduction to Software Engineering

Software Engineering Practice

- 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?

Introduction to Software Engineering

Software Engineering Principles

- This set of seven principles first proposed by computer scientist David Hooker focuses on engineering practices as a whole.
  1. **The reason it all exists:** Does the change add real value?
  2. **KISS (Keep It Simple Stupid!):** Keep it as simple as possible, but no simpler.
  3. **Maintain the Vision:** Do not compromise the architectural vision.
  4. **What you produce, others consume:** What you do must be understandable to others.
  5. **Be open to the future:** Never design yourself into a corner.
  6. **Plan ahead for reuse:** Difficult to do, but reduces cost and adds value.
  7. **Think!:** Give it a clear, complete thought before acting.

Introduction to Software Engineering

Summary

- Prior to the late 1960s, the field of software engineering did not exist. The growth in the computer market and the demand for more advanced software created a crisis in software development.
- Starting in the late 1960s, several computer pioneers including Fritz Bauer and Margaret Hamilton advocated for software engineering as a separate engineering discipline.
- Software engineering encompasses the following:
  - systematic and disciplined
  - quantifiable
  - sound engineering principals
  - real-world applications
  - software lifecycles
  - associated documentation
  - study of the process