2019 级本科

软件工程

Software Engineering

张昕

zhangxin@cust.edu.cn

计算机科学技术学院软件工程系

Software Engineering

Challenges for building software

- Reality
  - Software has become deeply embedded in virtually every aspect of our lives, and as a consequence, the number of people who have an interest in the features and functions provided by a specific application has grown dramatically.
    - It follows that a concerted effort should be made to understand the problem before a software solution is developed.
  - The information technology requirements demanded by individuals, businesses, and governments grow increasing complex with each passing year. Large teams of people now create computer programs that were once built by a single individual. Sophisticated software that was once implemented in a predictable, self-contained, computing environment is now embedded inside everything from consumer electronics to medical devices to weapons systems..
    - It follows that design becomes a pivotal activity.

Challenges for building software

- Reality
  - Individuals, businesses, and governments increasingly rely on software for strategic and tactical decision making as well as day-to-day operations and control. If the software fails, people and major enterprises can experience anything from minor inconvenience to catastrophic failures.
    - It follows that software should exhibit high quality.
  - As the perceived value of a specifi c application grows, the likelihood is that its user base and longevity will also grow. As its user base and time-in-use increase, demands for adaptation and enhancement will also grow.
    - It follows that software should be maintainable.

Software in all of its forms and across all of its application domains should be engineered.

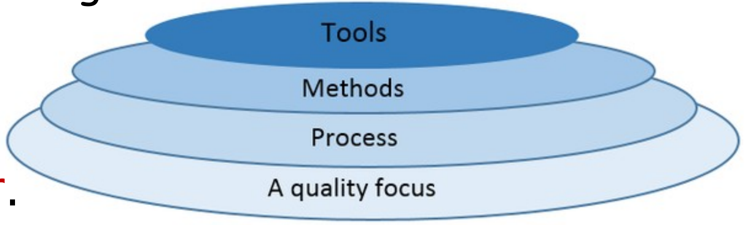
Software Engineering Discipline

- Software Engineering (by IEEE)

  ①The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

  ②The study of approaches as in ①.

- Software engineering is a layered technology.
  - Any engineering approach (including software engineering) must rest on an organizational commitment to quality.
    - The bedrock that supports software engineering is a quality focus.
  - The foundation for software engineering is the process layer.
  - The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software.
    - Process defines a framework that must be established for effective delivery of software engineering technology.
    - The software process forms the basis for management control of software projects and establishes the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured, and change is properly managed.



Software Engineering Discipline

- Software engineering *methods* provide the technical how-to's for building software.
  - Methods encompass a broad 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.
- Software engineering tools provide automated or semi-automated support for the process and the methods.
  - When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering , is established.

Software Process

- Key terms
  - 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 model).
  - A *task* focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.
  - A process is not a rigid prescription for how to build computer software. Rather, it 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 those who have sponsored its creation and those who will use it.

2019 级本科

软件工程

Software Engineering

张昕

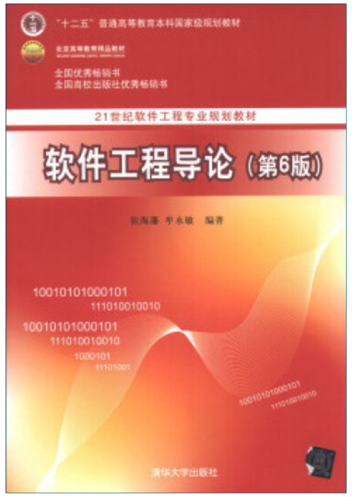
zhangxin@cust.edu.cn

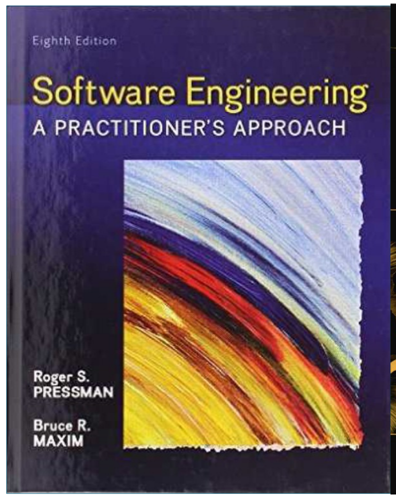
计算机科学技术学院软件工程系

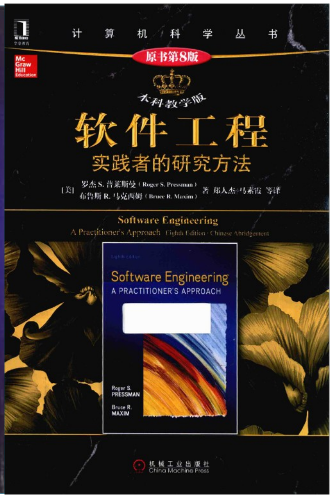
Course materials

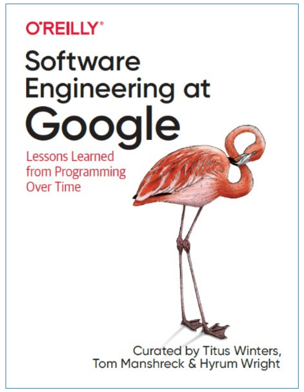
- Course Textbook: Introduction to Software Engineering
    - Good and famous book. However,
    - A little simple and fuzzy without sufficient cases
- <Software Engineering: A practitioner's approach> has been selected as complementary for strengthen your study
- Meanwhile, we also introduce two reference books
    - Composed by the experts from the industry
- Only focusing on textbook is not enough for you to master the necessary terms and knowledge

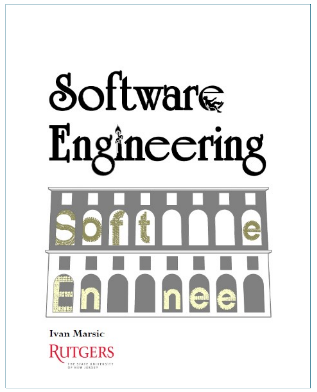
- Score
    - 10%: attendance, assignments
    - 20%: experiments
    - 70%: final exam (open book)








O'REILLY
Software Engineering at Google
Lessons Learned from Programming Over Time
Curated by Titus Winters, Tom Manshreck & Hyrum Wright



Process Framework

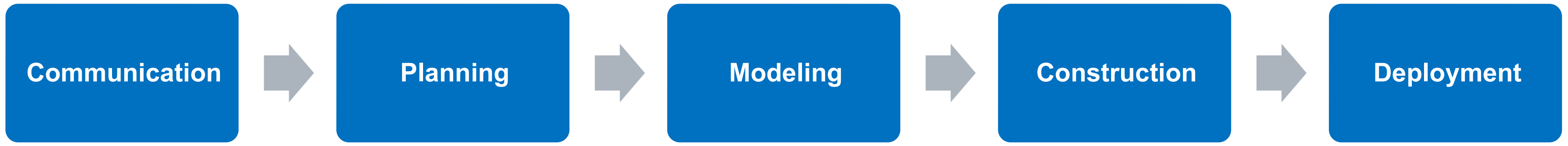
- A process framework establishes the foundation for a complete software engineering process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
- The process framework encompasses a set of umbrella activities that are applicable across the entire software process. (*during the development of small, simple programs, the creation of Web applications*)
- A generic process framework for software engineering encompasses five activities
  - Communication. The intent is to understand stakeholders' objectives for the project and to gather requirements that help define software features and functions.
  - Planning. It defines the software engineering work by describing 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.
  - Modeling. Software engineers refine complex systems by creating models to better understand software requirements and the design that will achieve those requirements.
  - Construction. This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.
  - Deployment. The software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

Umbrella Activities

- Umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk.
  - Typical umbrella activities include:
    - Software project tracking and control: allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.
    - Risk management: assesses risks that may affect the outcome of the project or the quality of the product.
    - Software quality assurance: defines and conducts the activities required to ensure software quality.
    - Technical reviews: assess software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.
    - Measurement: defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs; can be used in conjunction with all other framework and umbrella activities.
    - Software configuration management: manages the effects of change throughout the software process.
    - Reusability management: defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.
    - Work product preparation and production: encompass the activities required to create work products such as models, documents, logs, forms, and lists.

Process Framework

- A generic process framework for software engineering encompasses five activities
  - Communication. The intent is to understand stakeholders' objectives for the project and to gather requirements that help define software features and functions.
  - Planning. It defines the software engineering work by describing 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.
  - Modeling. Software engineers refine complex systems by creating models to better understand software requirements and the design that will achieve those requirements.
  - Construction. This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.
  - Deployment. The software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.



Process Adaptation

- A process adopted for one project might be significantly different than a process adopted for another project.
  - Among the differences are:
    - Overall flow of activities, actions, and tasks and the interdependencies among them.
    - Degree to which actions and tasks are defined within 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 the customer and other stakeholders are involved with the project.
    - Level of autonomy given to the software team.
    - Degree to which team organization and roles are prescribed.

Software Engineering Practice

- The essence of problem solving [How to Solve it by George Polya]
  - 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.

Software Engineering Practice

- Understand the problem → communication and analysis.
  - 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?

Software Engineering Practice

- Plan a solution → modeling and software design.
  - 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 defi ned? 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?

Software Engineering Practice

- Carry out the plan → code generation.
  - 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?

Software Engineering Practice

- Examine the result for accuracy → testing and quality assurance.
  - 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?

Software Engineering Practice

- General Principles
  - They focus on software engineering as a whole, as a specific generic framework activity, as actions or as technical tasks.
  - Help establish a mind-set for solid software engineering practice.
  - There are seven principles focusing on software as a whole proposed by David Hooker.
- The First Principle: The Reason It All Exists
  - A software system exists for one reason: to provide value to its users.
  - Keep asking "Does this add real value to the system?"
    - If the answer is no, don't do it. Allother principles support this one.
- The Second Principle: Keep It Simple and Stupid
  - *All design should be as simple as possible, but no simpler.* This facilitates having a more easily understood and easily maintained system.
  - Simple also does not mean "quick and dirty." It takes a lot of thought and work over multiple iterations to simplify. The payoff is software that is more maintainable and less error-prone.

Principle: an important underlying law or assumption required in a system of thought.

Software Engineering Practice

- The Third Principle: Maintain the Vision
  - *A clear vision is essential to the success of a software project.*
  - Compromising the architectural vision of a software system weakens and will eventually break even the well-designed systems.
  - Having an empowered architect who can hold the vision and enforce compliance helps ensure a very successful software project.
- The Fourth Principle: What You Produce, Others Will Consume
  - *Always specify, design, and implement knowing someone else will have to understand what you are doing.*
  - Specify with an eye to the users. Design, keeping the implementers in mind. Code with concern for those that must maintain and extend the system.
  - Making their job easier adds value to the system.

Software Engineering Practice

- The Fifth Principle: Be Open to the Future
  - A system with a long lifetime has more value.
  - True "industrial-strength" software systems must endure far longer. To do this successfully, these systems must be ready to adapt to these and other changes.
  - *Never design yourself into a corner.* Always ask "what if," and prepare for all possible answers by creating systems that solve the general problem, not just the specific one. This could very possibly lead to the reuse of an entire system.
- The Sixth Principle: Plan Ahead for Reuse
  - Reuse saves time and effort.
  - Achieving a high level of reuse is arguably the hardest goal to accomplish in developing a software system.
  - *Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated.*

Software Engineering Practice

- The Seventh Principle: 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.
  - If you do think about something and still do it wrong, it becomes a valuable experience.
    - A side effect of thinking is learning to recognize when you don’t know something, at which point you can research the answer.
  - When clear thought has gone into a system, value comes out.

Software Development Myths

- Software development myths—erroneous beliefs about software and the process that is used to build it—can be traced to the earliest days of computing.
  - They appear to be reasonable statements of fact (sometimes containing elements of truth), they have an intuitive feel, and they are often promulgated by experienced practitioners who "know the score."
  - Management myths.
    - Managers with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality.
    - Like a drowning person who grasps at a straw, a software manager often grasps at belief in a software myth, if that belief will lessen the pressure (even temporarily).

Software Development 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 refl ect 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 Development 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 fi rst, 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 Development Myths

- Myth
  - If I decide to outsource the software project to a third party, I can just relax and let that fi rm 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.

Software Development Myths

- Customer myths.
  - A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing/sales department, or an outside company that has requested software under contract.
  - In many cases, the customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths lead to false expectations (by the customer) and, ultimately, dissatisfaction with the developer.

Software Development Myths

- Myth
  - A general statement of objectives is suffi cient to begin writing programs—we can fi ll 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.

Software Development Myths

- Myth
  - Software requirements continually change, but change can be easily accommodated because software is fl exible.

- 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 modification.

Software Development Myths

- Practitioner's myths.
    - Myths that are still believed by software practitioners have been fostered by over 60 years of programming culture.
    - During the early days, programming was viewed as an art form. Old ways and attitudes die hard.

Software Development 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.

Software Development 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 20) are a “quality filter” that have been found to be more effective than testing for finding certain classes of software defects.

Software Development 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.

Software Development 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.

How it all starts

How a Project Starts

The scene: Meeting room at CPI Corporation, a (fictional) company that makes consumer products for home and commercial use.

The players: Mal Golden, senior manager, product development; Lisa Perez, marketing manager; Lee Warren, engineering manager; Joe Camalleri, executive vice president, business development

The conversation:

Joe: Okay, Lee, what's this I hear about your folks developing a what? A generic universal wireless box?

Lee: It's pretty cool . . . about the size of a small matchbook . . . we can attach it to sensors of all kinds, a digital camera, just about anything. Using the 802.11n wireless protocol. It allows us to access the device's output without wires. We think it'll lead to a whole new generation of products.

Joe: You agree, Mal?

Mal: I do. In fact, with sales as flat as they've been this year, we need something new. Lisa and I have been doing a little market research, and we think we've got a line of products that could be big.

Joe: How big . . . bottom line big?

Mal (avoiding a direct commitment): Tell him about our idea, Lisa.

Lisa: It's a whole new generation of what we call "home management products." We call 'em *SafeHome*. They use the new wireless interface, provide homeowners or small-businesspeople with a system that's controlled by their PC—home security, home surveillance, appliance and device control—you know, turn down the home air conditioner while you're driving home, that sort of thing.

Lee (jumping in): Engineering's done a technical feasibility study of this idea, Joe. It's doable at low manufacturing cost. Most hardware is off the shelf. Software is an issue, but it's nothing that we can't do.

Joe: Interesting. Now, I asked about the bottom line.

Mal: PCs and tablets have penetrated over 70 percent of all households in the USA. If we could price this thing right, it could be a killer app. Nobody else has our wireless box . . . it's proprietary. We'll have a 2-year jump on the competition. Revenue? Maybe as much as $30 to $40 million in the second year.

Joe (smiling): Let's take this to the next level. I'm interested.