# Introduction to SE Rules and Tools The Overview

Dr. Samuel Cho, PhD

NKU

# Core ideas of this module

# Core Idea 1

- SE is about rules and tools to build software products in a team.
- To be specific, SE is about *rules* and *tools* to *build* and deliver software *products* in a *team* on time and within budget.

# Core Idea 2

- It is hard to build and deliver software products on time and within budget.
- It is because of the complexity of software engineering.
- Also, everything about SE changes including requirements, schedules, people, and environments; and these add the complexity of SE.
- So, the definition of SE can be interpreted as "SE is about managing complexity using rules and tools."

1. The definition of Software Engineering in this class.
2. What are the problems of SE?
3. What are the solutions of the problems?
4. SE Tools that we are going to use in this class.

# The definition of Software Engineering in this class.

# What is Software Engineering (SE)?

- Is it about making software?
- Is it about managing people (SE engineers) to make software better?
- Is it about the philosophy of software?
- Is it about the mathematical formulation of software?

- If you ask about the definition of software engineering to hundred software engineers, it is very likely that you will get one hundred different answers.
- It is interesting because if you ask the same question to one hundred physicists, the answers will not be that different.
- It is likely the same with engineering field including mechanical engineers or electrical engineering.
- But not with software engineers. Why?

- It is because SE is a new field.
- It is because the idea of software development using programming languages is very new (the first programming language, FORTRAN, was invented in 1957 at IBM).
- It is because electronic computers were invented recently (the first fully electronic computer, ENIAC, was invented in 1946 at UPenn).
- So, we should define SE to have a common understanding of SE.
- At the same time, we need to understand why we learn SE in this course.

# The definition of SE in this class

## 👉 Definition #Software Engineering (short version)

- SE is about rules and tools to build software products in a team.

- What are the keywords that we should focus on?
- They are (1) rules, (2) tools, (3) building products, and (4) team.

# The definition of SE in this class

However, we can be **more specific** in the definition of SE.

## Definition #Software Engineering

- SE is about *rules* and *tools* to *build* and deliver software *products* in a *team* on time within budget.

- What keywords are added?
- They are (1) deliver products and (2) on time within budget.

# What are the problems of SE?

# Software Crisis

## Idea #Software Crisis

- In the 1970s and 1980's, software engineers found that most software engineering projects were over-budget and over-time.
- Even many of the delivered software products were low-quality and/or did not meet requirements.
- We call this incident **software crisis**.

- Some software engineers began to believe that these issues are inherent in software engineering, and there is no hope.
- But other software engineers began to look into the problem and tried to come up with a solution.

# What are causes of the Software Crisis?

- The main and the most important cause is complexity.
- Simply put, software engineering is just complex.
- Software engineering is the most complex entity that humans have ever experienced.

# Complexity of SE

## Idea #Complexity of SE

- Boeing 747 airplane is composed of six million parts.
- However, it is known that Windows XP had 40 million lines of code, not including source comments or any bundled third party code (like drivers), only the code compiled by Microsoft.
- Linux kernel has around 27.8 million lines of code in its Git repository excluding all the Linux tools and libraries.

# What are causes of the Software Crisis?

- The other important cause is Change.
- The users' requirements keep changing.
- The programming environment, i.e., library versions, keeps changing.
- We have new hardware, new software, and new tools are introduced every day.
- Everything about SE keeps changing, and it adds another level of complexity to the SE.

# Limitations of SE

- However, unfortunately, we have limitations in SE.
- The most important and famous rule to explain the limitations is the "No silver bullet (NSB) law."
- The other important rule that explains the limitations of SE is "Fred Brooks' law."

# No silver bullet (NSB) law

## Rule #No silver bullet (NSB) law

- There is no single management or technology development that promises an order of magnitude (tenfold) improvement in productivity within a decade by Fred Brooks (1986).

- This rule does not say there is no productivity boost in SE. This rule says that a dramatic productivity boost is slow in SE.
- It is because SE needs many management or technology tools combined to get the dramatic productivity boost, and it takes time.

# No silver bullet (NSB) law

## Rule #No silver bullet (NSB) law

- There is no single management or technology development that promises an order of magnitude (tenfold) improvement in productivity within a decade by Fred Brooks (1986).

- This rule does not say there is no productivity boost in SE. This rule says that a dramatic productivity boost is slow in SE.
- It is because SE needs many management or technology tools combined to get the dramatic productivity boost, and it takes time.

# Fred Brooks' (FB) law

## Rule #Fred Brooks' (FB) law

- Adding more manpower to a late project makes it later by Fred Brooks.

- Software engineering requires some knowledge to play the role of a software engineer.
- So, when new members join a project, existing members should train (complexity) the new members; we have added communication channels (complexity) with added members in a team.
- All these added complexities delay the late project even more.

# So, let's simplify everything we have discussed so far

- SE is just complex and keeps changing.
- We cannot have innovation that can boost productivity in a short amount of time.
- We cannot just hire people to solve SE problems because of training and added complexity.
- It is not surprising that we had "Software Crisis."

However, fortunately, we could manage (not solve) some of SE's complexity to overcome the "Software Crisis."

# What are the solutions of the problems?

# The limitations of SE, again

- We should admit that we cannot solve the Complexity issue of SE.
- It is very likely we cannot solve this Complexity problem forever (at least quite a while.)
- So, we can focus only on "managing complexity".
- When we talk about "solutions," we are talking about the solutions to "manage SE complexity" or "solving some of the problems caused by the complexity", not "solving SE complexity."

# SE Rules, Tools, and Team, again

- We have accumulated SE rules that can manage the SE complexity.
- We have developed SE tools that can manage the SE complexity.
- Nowadays, it is normal to have a team to develop software products. We can join our workforce together to manage the SE complexity.

# What is the most important SE rule?

## Rule #KISS

Keep it Simple and Stupid.

- Simplicity is the most powerful rule and tool for managing complexity.
- Keep everything just simple, and if necessary, absurdly stupid.
- If you have to choose between any type of choices in SE, always choose the simplest one.

# What is the most important SE tool?

## Definition #Software Design

Software Design is about "modules" and "interfaces" of software.

- Software design is not about GUI (Graphical User Interface) or anything about software graphics. From now on, when we talk about "Design" it is about "Software Design."
- It is a rather conceptual tool, but we use the software engineering term "modules" and "interfaces" to explain the idea.
- We use circles or rectangles to express the "modules" and arrows or connections to express the "interfaces."

# What is the most important Team rule?

## Rule #No Surprises Rule

When you work as a member of a team, never surprise anyone in your team with your work.

- Always let others know your plans and what you are doing now.
- When there is any change, you should let them know ASAP.

- It is OK to make a mistake, but it is never OK to hide the mistake.

## Idea #No Surprises rule in programming languages

- Programming language designers do their best not to surprise users (programmers) by applying the "No surprise" rule.
- When they introduce the 'if' statement, it implements the selection feature, and the same for the 'for' statement to implement the loop feature.

This is from Matz, who designed and implemented the Ruby Programming Language. *"I designed Ruby to minimize my surprise. I wanted to minimize my frustration during programming, so I want to minimize my effort in programming."*

# Other important SE rules

These rules and software design are the basis of many other important SE rules and tools.

## Rule #Divide and Conquer Rule

When we deal with the SE problem, we should divide the problem into small pieces and solve the problem one by one instead of solving the problem as a whole.

## Rule #Only Fools Rush In Rule

Think before you act. Design it before you implement it.

- To manage complexity, we should follow the "Only Fools Rush in" rule, and we should "Design" before we "Implement."
- To manage complexity, we should "Divide and Conquer" rule, and we divide the big task into small ones so that each team member finishes their tasks.
- We need to follow SE rules to manage complexity. To do that, we (SE engineers) should find and understand the meaning of each SE rule correctly and apply them whenever necessary.
- We should do this for the rest of our career.

# SE Tools that we are going to use in this class.

# SE Tools

- We discussed that the most important SE tool is "Software Design."
- We have other SE tools, both conceptual and concrete.
- The other most important SE tool is "Software Process."

# Software Process

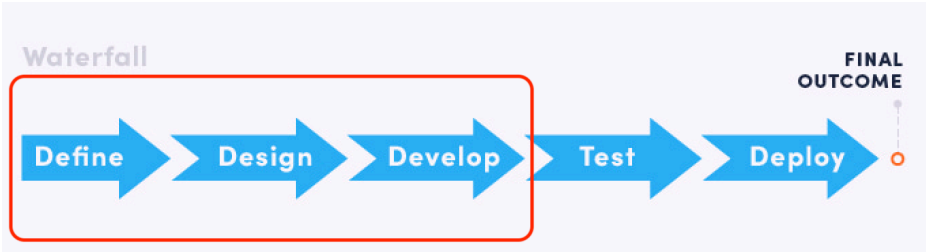
## Definition #Software Process

A software process is a set of related activities that leads to the production of the software.

- From now on, when we talk about "Process" it is about "Software Process."
- Software process is also a model to develop and deliver software products.
- Software process uses the "Divide and conquer" rule and "Only Fools Rush In" rule to manage the complexity from software development.

# SE Process - the application of SE rules

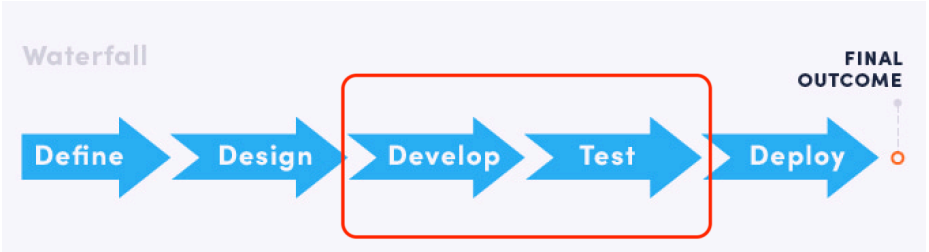
- We can apply the "Divide and Conquer" rule to divide software development activities into multiple stages.
- We also apply the "Only Fools Dive In" rule to define and design what we are going to do before we develop.



## Rule #"Always Make Tests" rule

Code without tests will lead to legacy code.

- We should test whenever we finish development (implementation) following the "Always Make Tests" SE rule.



## Idea #Legacy Code

Legacy code refers to an application system source code type that is no longer supported or maintained.
To most programmers, legacy code is simply code without tests. So, when we do not make tests for code, we make legacy code.

- Legacy code adds complexity that is unmanageable.
- So, we should always add tests whenever we implement any thing to manage complexity.

# Managing Complexity using Process

- We need Software Process to manage the complexity of SE.
- To do this, we divide the software development process into multiple stages (activities).
- We may have different names for the stages. For example, "Define" can be replaced by "Requirements", and "Development" can be replaced by "Coding."
- The number of stages can be varied depending on a situation. In this example we have five stages, but the "Design" stage can be divided into "Architecture" and "Design", and "Development" and "Test" can be merged into just "Development."

# SE Tools - Practicalities

- We discussed two important conceptual SE tools: design and process.
- Let's discuss the concrete/real tools that we are going to use for this course.
- In the next chapters, we are going to (1) install and (2) use the tools making ready for the course project.

SE tools can be grouped into the following categories.

- Documentation Tools
- Requirement Tools
- Design Tools
- Programming Tools
- Version Control System (VCS) Tools
- Testing Tools
- Communication/Team Tools
- Deployment Tools
- Security Tools

# From now on ...

- For the rest of this semester, we learn the ideas and usages of these tools to manage the complexity of SE.
- We install the tools (HW1).
- We understand what are the goals of the tools, and why we need to use the tools.
- We practice to learn how to use the tools.
- We use the tool for building an application as a part of the class project.