

Lecture No. 1

Introduction to Software Engineering

An Introduction to Software Construction Techniques for Industrial Strength Software

1.1 Introduction

Software engineering is an interesting subject. In order to understand this subject we will need to look at a number of examples and case studies. And we will need to see how we can develop good software and how it could be improved in different scenarios? Before we move on to software engineering we need to understand what software actually is.

➤ What is Software?

When we write a program for computer we named it as software. But software is not just a program; many things other than the program are also included in software.

Some of the constituted items of software are described below.

- **Program:** The program or code itself is definitely included in the software.
- **Data:** The data on which the program operates is also considered as part of the software.
- **Documentation:** Another very important thing that most of us forget is documentation. All the documents related to the software are also considered as part of the software.

So the software is not just the code written in Cobol, Java, Fortran or C++. It also includes the data and all the documentation related to the program.

➤ Why is it important?

Undoubtedly software is playing a vital role in all the field of life these days. We can see many software applications being operated around us in our daily routine.

Some of the major areas in which software has played an important role are identified as under.

- *Business decision-making:* Software systems have played a major role in businesses where you have to analyze your data and on the basis of that analysis you have to make business decisions. This process of data analysis and decision-making has become very accurate and easy by the use of software.
- *Modern scientific investigation and engineering problem solving:* Scientific investigations and engineering problem solving require an intensive amount of

calculations and data analysis. The accuracy of these analyses is also very important in scientific applications. This process has become very easy and accurate by the use of software. For example software systems are becoming more involved in bioinformatics and the process of DNA decoding is only possible by the use of software systems. Similarly many astronomical observations are being recorded and analyzed by the software systems these days.

- *Games*: We see many computer games these days that interests people of all ages. All these games are drive through software systems.
- *Embedded systems*: We see many kinds of gadgets being employed in our daily used things, like small microcontrollers used in our cars, televisions, microwave ovens etc. All these systems are controlled through the software.

Similarly in many other fields like education, office automation, Internet applications etc, software is being used. Due to its central importance and massive use in many fields it is contributing a lot in terms of economic activity started by the software products. Billions and trillions of dollars are being invested in this field throughout the world every year.

➤ Engineering

Before moving on to software engineering lets first discuss something about engineering itself. If you survey some of the dictionaries then you will find the following definition of engineering.

“The process of productive use of scientific knowledge is called engineering.”

1.2 Difference between Computer Science and Software Engineering

The science concerned with putting scientific knowledge to practical use.

Webster’s Dictionary

There are many engineering fields like electrical, mechanical and civil engineering. All these branches of engineering are based on physics. Physics itself is not engineering but the use of physics in making buildings, electronic devices and machines is engineering. When we use physics in constructing buildings then it is called civil engineering. When we use physics in making machines like engines or cars then it is called mechanical engineering. And when we apply the knowledge of physics in developing electronic devices then the process is called electrical engineering. The relation of computer science with software engineering is similar as the relation of physics with the electrical, mechanical or civil engineering or for that matter the relation of any basic science with any engineering field. So in this context we can define software engineering as:

“This is the process of utilizing our knowledge of computer science in effective production of software systems.”

➤ Difference between Software and Other Systems

Now lets talk something about how a software system is different from any other systems. For example, how software is different from a car, a TV or the similar systems or what is the difference between software engineering and other engineering like

mechanical or electrical engineering. Lets look at some of the non-software systems like TV, Car or an Electric Bulb. The car may be malfunctioned due to some problem in engine while driving. Similarly an electric bulb may be fused while glowing and a TV could be dysfunctional while working.

So the major thing that distinguishes a software system from other systems is that;

“Software does not wear out!”

What does that mean?

As we have seen in above example that our non-software systems could be malfunctioned or crash while working. That mean they are affected by the phenomenon of wear and tear. They have a particular life and after that they could have some problem and may not behave and perform as expected. But this is not the case with software. Software systems does not affect by the phenomenon of wear and tear. If a software has any defect then that defect will be there from the very first day and that defect normally called bug. That means if a software is not working then it should not work from the very first day. But this could not be the case that at a particular point in time a software is functioning well and after some time the same software is not performing the same task as required. So software does not have the element of wear and tear. Lets elaborate this point further. We have just talked about software defects which we call bugs. If a part of a car became wear out you just need to get a new one from market and replace the damages one with the new one. And the car will start working properly as it was working previously. Similarly if an electric bulb got fused then you just need to get a new one and put into the socket in place of the fused one and your room will again be illuminated. But the case of software is somewhat different. If a software has a bug then the same process of replacing faulty part with the new one may not work. You cannot remove the bug by just replacing the faulty part of software with the new one. Or it will not be as simple that, you go to the concerned company, get a new CD of that software and it will start working properly. If the software has a bug and that bug was present in the older CD then that will remain in the new one. This is a fundamental difference between software and other systems.

Software Engineering Complexity of Software

Here the subject is again the same that how software systems are different from other systems. Have you ever noticed that how many different models of a car do a car manufacturing company release in a year? And how many major changes are made in new models and what is the frequency of these changes. If you think a little bit on it then you will realize that once the system is finalized then the changes in new models are of very minor nature. A drastic change is very unlikely in these kinds of systems. So the frequency of changes in these systems is very low and of minor nature. Like body shape could be changed a little, a new gadget could be added and the like but it is very unlikely that a fundamental change in engine is made. On the other hand if you observe the activities of a software manufacturing company, you will realize that these companies make changes of fundamental nature in their software systems. They constantly change their systems whether in the form of enhancements, in the form of interface change or

they are making a new system altogether. In other words they are making changes in their systems in many different dimensions. But in non-software systems these kind of changes are not that much frequent. One of the major reasons of increased bugs in software systems is this high frequency of change. You can well imagine that if a car manufacturing company manufacture cars in the similar way then how long these cars will remain useful, how much effort they have to put to design these cars, how much time they will require to mature the design, and how much time they would be needing to start production of such cars. If they try to cut-short that time, meaning that if they try to release cars after every six-months or a year without proper testing and that release has a fundamental change then that kind of cars will also have lots of bugs and will not be road-worthy.

Therefore one of the major reasons of complexity in software is due to its basic nature that the software passes through a constant process of evolution. *The name of the game is change and evolution all the times in all the dimensions.* This change has the direct impact on software in the form of defects. Therefore software engineers also have to deal with the challenge of managing this process of change and evolution.

1.3 Software Crisis

What is Software Crisis?

Computer systems were very new and primitive in early fifties and the use of software was also very limited at that time. It was limited to some scientific applications or used to process the data of census. In 1960s a great amount of rapid improvement was made in hardware. New hardware and new computer systems were made available. These computer systems were far more powerful than the computers of early fifties. It is all relative, the computers of 1960s are primitive as compare to the computers we have these days but were far more powerful than the computers of early fifties. More powerful hardware resulted into the development of more powerful and complex software. Those very complex software was very difficult to write. So the tools and techniques that were used for less complex software became inapplicable for the more complex software. Let's try to understand this with the help of an example.

Let's imagine a person who used to live in a village and who have constructed a hut for him to live. Definitely he should have face some problems in the beginning but was managed to build a hut for him. Now if you ask him to construct another hut, he may be able to construct one more easily and in a better way. This new hut may be better than the first one and he may construct it in a relatively less time. But if you ask him to construct concrete and iron houses then he may not be able to handle it. Since he made a hut and he know how to make a place to live so you may expect from him to build concrete and iron buildings. If this is the case then you should all agree that the building constructed by that person will not have a stable structure or he may not even be able to build one.

In early 60s software had suffered from the similar kind of problem to which we call *Software Crisis*. Techniques that were used to develop small software were not applicable for large software systems. This thing resulted in the following consequences.

- In most of the cases that software which was tried to be build using those old tools and techniques were not complete.
- Most of the times it was delivered too late.
- Most of the projects were over-budgeted.
- And in most of the case systems build using these techniques were not reliable – meaning that they were not be able to do what they were expected to do.

As a result of these problems a conference were held in 1960 in which the term software crisis was introduced. And the major issue discussed was that the development of software is in crisis and we have not been able to handle its complexities. And the term of Software Engineering was also coined in the same conference. People have said that, we should use engineering principles in developing software in the same way as we use these principles in developing cars, buildings, electronic devices etc. Software engineering is the result of software crisis when people realized that it is not possible to construct complex software using the techniques applicable in 1960s. An important result of this thing was that people had realized that just coding is not enough.

More Complex Software Applications

This conception is also very common these days. People think that if one knows how to code then that's sufficient. But just writing code is not the whole story. People have realized this fact way back in 1960s that only coding is not sufficient to develop software systems, we also need to apply engineering principles.

Software Engineering as defined by IEEE:

Let's look at some of the definitions of software engineering.

Software Engineering as defined by IEEE (International institute of Electric and Electronic Engineering). IEEE is an authentic institution regarding the computer related issues.

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

Before explaining this definition lets first look at another definition of Software Engineering given by Ian Sommerville.

“All aspects of software production’ Software engineering is not just concerned with the technical processes of software development but also with activities such as software project management and with the development of tools, methods and theories to support software production”.

These definitions make it clear that Software Engineering is not just about writing code.

1.5 Software Engineering

1.5 Software Engineering

Software Engineering is the set of processes and tools to develop software. *Software Engineering is the combination of all the tools, techniques, and processes that used in software production.* Therefore Software Engineering encompasses all those things that are used in software production like:

- Programming Language
- Programming Language Design
- Software Design Techniques
- Tools
- Testing
- Maintenance
- Development etc.

So all those thing that are related to software are also related to software engineering.

Some of you might have thought that how programming language design could be related to software engineering. If you look more closely at the software engineering definitions described above then you will definitely see that software engineering is related to all those things that are helpful in software development. So is the case with programming language design. Programming language design is one of the major successes in last fifty years. The design of Ada language was considered as the considerable effort in software engineering.

These days object-oriented programming is widely being used. If programming languages will not support object-orientation then it will be very difficult to implement object-oriented design using object-oriented principles. All these efforts made the basis of software engineering.

Well-Engineered Software

Let's talk something about what is well-engineered software. Well-engineered software is one that has the following characteristics.

- It is reliable
- It has good user-interface
- It has acceptable performance
- It is of good quality
- It is cost-effective

Every company can build software with unlimited resources but well-engineered software is one that conforms to all characteristics listed above.

Software has very close relationship with economics. Whenever we talk about engineering systems we always first analyze whether this is economically feasible or not. Therefore you have to engineer all the activities of software development while keeping its economical feasibility intact.

The major challenges for a software engineer is that he has to build software within limited time and budget in a cost-effective way and with good quality

Therefore well-engineered software has the following characteristics.

- Provides the required functionality
- Maintainable
- Reliable
- Efficient
- User-friendly
- Cost-effective

But most of the times software engineers ends up in conflict among all these goals. It is also a big challenge for a software engineer to resolve all these conflicts.

The Balancing Act!

Software Engineering is actually the balancing act. You have to balance many things like cost, user friendliness, Efficiency, Reliability etc. You have to analyze which one is the more important feature for your software is it reliability, efficiency, user friendliness or something else. There is always a trade-off among all these requirements of software. It may be the case that if you try to make it more user-friendly then the efficiency may suffer. And if you try to make it more cost-effective then reliability may suffer. Therefore there is always a trade-off between these characteristics of software.

These requirements may be conflicting. For example, there may be tension among the following:

- Cost vs. Efficiency
- Cost vs. Reliability
- Efficiency vs. User-interface

A Software engineer is required to analyze these conflicting entities and tries to strike a balance.

Challenge is to balance these requirements.

Software Engineers always confront with the challenge to make a good balance of all these things depending on the requirements of the particular software system at hand. He should analyze how much weight should all these things get such that it will have acceptable quality, acceptable performance and will have acceptable user-interface.

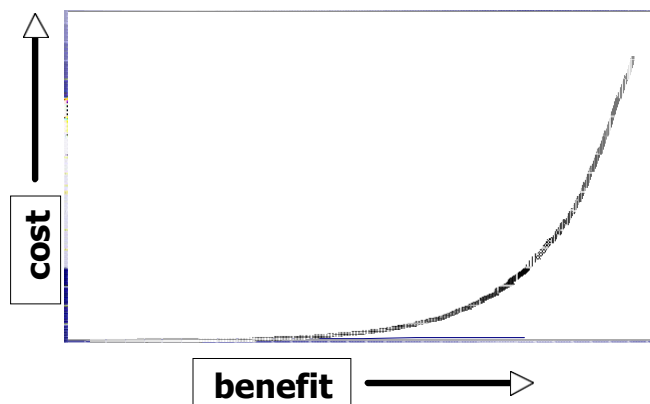
In some software the efficiency is more important and desirable. For example if we talk about a cruise missile or a nuclear reactor controller that are droved by the software systems then performance and reliability is far more important than the cost-effectiveness and user-friendliness. In these cases if your software does not react within a certain amount of time then it may result in the disaster like Chernobyl accident.

Therefore software development is a process of balancing among different characteristics of software described in the previous section. And it is an art to come up with such a good balance and that art can be learned from experience.

Law of diminishing returns

In order to understand this concept let's take a look at an example. Most of you have noticed that if you dissolve sugar in a glass of water then the sweetness of water will increase gradually. But at a certain level of saturation no more sugar will dissolve into water. Therefore at that point of saturation the sweetness of water will not increase even if you add more sugar into it.

The law of diminishing act describes the same phenomenon. Similar is the case with software engineering. Whenever you perform any task like improving the efficiency of the system, try to improve its quality or user friendliness then all these things involve an element of cost. If the quality of your system is not acceptable then with the investment of little money it could be improved to a higher degree. But after reaching at a certain level of quality the return on investment on the system's quality will become reduced. Meaning that the return on investment on quality of software will be less than the effort or money we invest. Therefore, in most of the cases, after reaching at a reasonable level of quality we do not try to improve the quality of software any further. This phenomenon is shown in the figure below.



Software Background

Caprice Jones a renowned practitioner and researcher in the field of Software Engineering, had made immense research in software team productivity, software quality, software cost factors and other fields related to software engineering. He made a company named Software Productivity Research in which they analyzed many projects and published the results in the form of books. Let's look at the summary of these results.

He divided software related activities into about twenty-five different categories. They have analyzed around 10000 software projects to come up with related activities.

such a categorization. But here to cut down the discussion we will only describe nine of them that are listed below.

- Project Management
- Requirement Engineering
- Design
- Coding
- Testing
- Software Quality Assurance
- Software Configuration Management
- Software Integration and
- Rest of the activities

One thing to note here is that you cannot say that anyone of these activities is dominant among others in terms of effort putted into it. Here the point that we want to emphasize is that, though coding is very important but it is not more than 13-14% of the whole effort of software development.

Fred Brook is a renowned software engineer; he wrote a great book related to software engineering named “A Mythical Man Month”. He combined all his articles in this book. Here we will discuss one of his articles named “No Silver Bullet” which he included in the book.

An excerpt from “No Silver Bullet” – Fred Brooks

Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these we seek bullets of silver that can magically lay them to rest. The familiar software project has something of this character (at least as seen by the non-technical manager), usually innocent and straight forward, but capable of becoming a monster of missed schedules, blown budgets, and flawed projects. So we hear desperate cries for a silver bullet, something to make software costs drop as rapidly as computer hardware costs do. Skepticism is not pessimism, however. Although we see no startling breakthroughs, and indeed, such to be inconsistent with the nature of the software, many encouraging innovations are under way. A disciplined, consistent effort to develop, propagate and exploit them should indeed yield an order of magnitude improvement. There is no royal road, but there is a road. The first step towards the management of disease was replacement of demon theories and humors theories by the germ theory. The very first step, the beginning of hope, in itself dashed all hopes of magical solutions. It told workers that progress would be made stepwise, at great effort, and that a persistent, unrelenting care would have to be paid to a discipline of cleanliness. So it is with software engineering today.

So, according to Fred Brook, in the eye of an unsophisticated manager software is like a giant. Sometimes it reveals as an unscheduled delay and sometimes it shows up in the form of cost overrun. To kill this giant the managers look for magical solutions. But

unfortunately magic is not a reality. We do not have any magic to defeat this giant. There is only one solution and that is to follow a disciplined approach to build software. We can defeat the giant named software by using disciplined and engineered approach towards software development.

Therefore, *Software Engineering is nothing but a disciplined and systematic approach to software development.*

1.6 Summary

Today we have discussed the following things related to software engineering.

- What is software engineering?
- Why is it important?
- What is software crisis?
- How software engineering derived from software crisis.
- What is the importance of engineering principles in developing software?
- What is balancing act and how apply in software engineering?
- What is law of diminishing returns?
- And what are the major activities involved in the development of software.