Software Engineering

February 22, 2013

1 Introduction

1.1 What is software engineering

- Software enginnering is not about *programming*
- Software enginnering is the study of the*methodology* of how to produce software $cost-effectively$

1.2 Prerequisites

- Introduction to Computer Science
- Introduction to programming : C/C++/Java
- **Programming experience**

1.3 Why some people think that software engineering is just bullshiting ?

- University education mostly focus on $one-man-band$ task, while software engineering talks about management of a team of people - group work
- The programming task in undergrad courses mostly not more than 1000 lines of code, while software in real world environment can have more than 1,000,000
- The programming task in undegrad courses mostlydoes not require update and maintenance

1.3.1 The nature of Software Engineering

- The nature of management-related courses is really relatively boring
- There is no theory , even some "theory" may contradicts with each other
- Totally real-life practically oriented

Because of these, it is difficult for the student (especially "maths-oriented") to find something in the lecture that can be mapped with their experience, and thus they think that "those management stuff are just bullshiting"

1.4 Why Software Engineering

Software Engineering practices can

- improve the quality of the software product / produce better software
- increase the effectiveness of the software development process
- reduce the risk : redoing the whole project / creating wrong software

1.5 Software Crisis

Without the software engineering practice :

- Y2K Bug
- The grand opening of the Hong Kong Airport was postponed 4 times
- Bank of America Master Net System eventually gave up the software system and 34billion trust accounts transferred.
- Explosion of Ariane 5 prototype in 1996
- Explosion of Boeing's Delta III rocket.
- etc ...

1.6 Software Myths

All the following are WRONG concepts

1.6.1 Management Myths

- 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?
- My people do have state-of-the-art software development tools; after all, we buy them the newest computers.
- If we get behind schedule, we can add more programmers and catch up

Software development is an logical creative process, more people, better equipments does not grant more logical and more creative.

Only few people know how to use newest technologies, training the work to use newest tecnhology is time consuming

Sometime newest technology conflicts with existing technology

Software development has higher chance to fail since worker are not familiar with newest technologies.

1.6.2 Customer Myths

- A general statement of objectives is sufficient to begin writing programs – we can fill in the details later

  Software always need to re-write because not all requirements are listed out

- Project requirements continually change, but change can be easily accommodated because software is flexible

  Rewriting an existing software is an nightmare, especially those "legacy software"

1.6.3 Practitioner's Myths

- Once we write the program and get it to work, our job is done.
- Until I get the program "running" I really have no way of assessing its quality.
- The only deliverable for a successful project is the working program.

  From statistics, 75% of the cost is spended in software mainteances.

2 Software

2.1 What is software

- Source code $\neq$ software , Executable $\neq$ software
- Software = programs + documents + user data + data base
- Software is a product, but it is also the tools to delivers other product

2.2 Type of Software

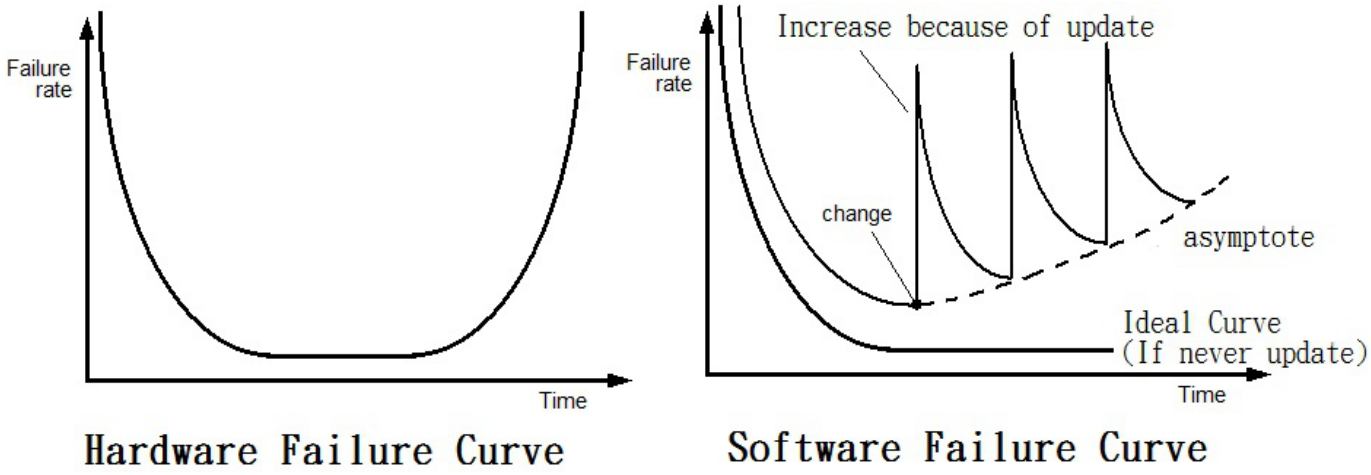
- System Software : Operating System + Software Development System
- Application Software : "apps"
- Different type of software has different requirement : Speed, moibility, security, stability, flexibility, etc.

2.3 Properties of software

- Hardware is physical, but software is not physical, it is a logical product
- Hardware is manufactured, but software is developed/engineered
- Hardware product will wear out, but software does not wear out : because software is a logical product, it can have ∞ service life.
- Hardware product is produce using $component - based\ construction$ (you don't directly build a computer, you build a computer by connecting different component, such as CPU, memory, mother board, IC, system fan, heat sink, etc) . but software product is $custom - built$ (you build it directly) , there is still no "$software\ IC$" (research topic)

- Hardware production progress can be observed easily, but it is very difficult to know what progress has been achieved in software development. Thus a "mile stone" is used to judge where the software project has achieved.
- The cost on duplicate a hardware may be very high (consider to duplicate a piano), but duplicate a software is usually very simple (just click the mouse)
- There are raw material for building hardware, but for software, we need to build new programs from scratch (from nothing), so the production cost of software is actually very high.

The following digram is the classical Failure curve (from Reliability Engineering discipiline)



- All product has very high failure rate at the very beginning
- As time pass (with improvement) , the failure rate decrease
- At certain point, the failure keeps constant
- Hardware failure rate eventually increase again because the component ware out (such as rusting, connection ware off)
- But for software, if it never update, then the failure rate will keep the same (forever)
- Once there is changes (update), the software failure rate increase, sine the beta-version usually has many bugs
- Once all the bugs are removed, the beta version become new version, and the failure rate decrease
- Since the overall software system become more complicated (update is adding more feature), thus the overall failure rate is increasing
- * The job of software engineers is to **reduce the height of those peaks**

3 Process Model

- The word *process* in software engineering means " a step-by-step flow of a series of activites to build the software. In other words, it is the *methodology* of how to build a software cost-effectively.
- Different process model fit different task

3.1 The old process before Software Engineering - Code & Debug

- The tradtional method of creating software before the Software Engineering era
- No *methodology*
- Consist of just 2 steps : write the program, then correct the mistakes

Advantage

- No need to bother anything related to "software engineering"

Disadvantage

- No plan , so the logic of the code become very complicated and difficult to follow up after several update
- No analysis, so the software may not meet the customer's requirement.

Application

- Only for very small task, such as programming assignment in the university

3.2 The classical waterfall model

- Step-by-Step : Communication → Planning → Modeling →Construction →Deployment
- Divide the task into different big-segments, different subtask has to be carried out in each step

Advantage

- With plan, the management of the project become easier
- With the plan, reduce the risk of the product not meeting customer's requirement
- Straight forward step by step approach

Disadvantage

- It assume that the requirement can be listed out clearly
- The coding part is in the second last stage, there is a big risk when the customer requirement change
- It is very difficult to go back to previous step
- The product will only be delivered in the **final stage**

3.3 Modified waterfall model

- Adding the feature of $feedback$ in each step
- Making it easier to go back to previous step
- But it makes the porject more complicated

3.4 Incremental Model

- Multiple - Waterfall model
- In incremental model, the real product is designed, implemented, integrated and tested as a series of incremental builds.
- The system after delivery of $n + 1^{th}$ increment is more complicated and mature than after n^{th} increment.

3.5 Prototyping Model

- Similar to incremental model
- But there is no real product, just the prototype is designed, implemented, integrated and tested as a series of incremental builds
- The prototype can be / may be throw away (but not for real product in incremental model)
- The prototype is used as an method to see wether the prototype meet the user's requirement, the feedback of the user can improve the system in the next increment.
- This model is used when the user requirement can not be defined clearly.
- Throw away the prototype meaning throw away resources, thus prototype should model should be used in the step that having high risk.

3.6 Spiral model

- A circular variation of waterfall model
- Risk management is involed in each development cycle, thus the risk is highly reduced.
- Since risk management is involed, thus the project progress become slower

3.7 Rational Unified Process RUP

- It sums up all the advantage of different type of process model
- The development process is divided into different phases.

3.7.1 Advantage

- It has all the advantage of the process model mentioned above

3.7.2 Disadvantage

- Time consuming
- The project become a huge project , it is difficult to manage

4 Agile Software Development

- The business environment keeps changing, so *speed* is an essential factor in software development.
- The software development team need to *catch up* with the *changes* (customer's requirement & business environment)

Because of that, **The Manifesto for Agile Software Development** appear

- Some people think that the software process model listed above is *clumsy* , *tedious* and *institutional*
- They give up those traditional approach in software development, and seek for $faster$, *simplier* software deveopment methodologies
- Thus, another group of new software development process models started to pop up.

4.1 Properties of Agile Software Process Model

- Keep it Simple, Stupid !
- Heavily drawing customer onto the team
- Rapid, incremental delivery of software (even a very very small change)

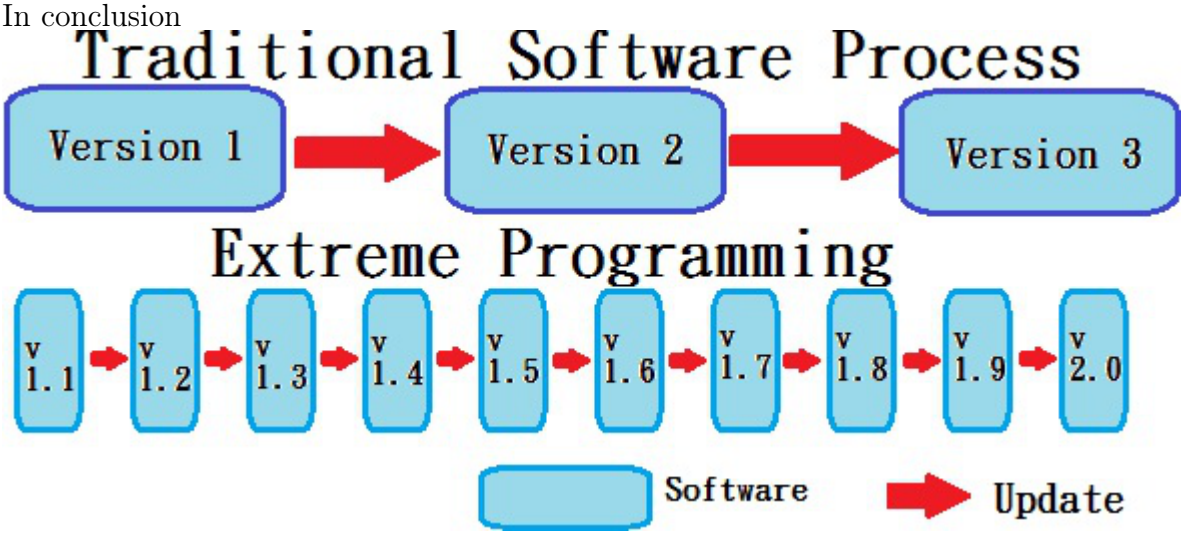
There are many agile process model, one of the most widely used agile process is the *Extreme Programming* (XP)

4.2 Extreme Programming XP

- XP draw the customer onto the team by allowing the customer to create *user story*
- Then development team assesses each story & assigns a *cost* : the time needed to complete that required functionality (*project velocity*)
- Customer decide which *user story* work first, a *commitment* is made on delivery date
- Encourage delivery of software with **samll** increment (**the samller the better**) , the ideas is that , deliver the updated software as frequent as possible, as fast as possible, only make the smallest updates, do not make redunant update , because the user requrement may change tomorrow, so only finish the taks that the user require.
- Another advantage of "small" increment is that it reduce the cost in testing and re-writing the program.
- Because the software requirements are now represented as many small *user stories* , so finish those *user story* with highest *project velocity*
- Keep it Simple, Stupid !
- Encourages "refactoring"—an iterative refinement of internal program design
- XP is very useful for the task that the **requirement will always change** such as the end-user does not have any concept about the system. Since "*uncertainty of the system* " is the "only" things that you "*certainly*" know !

- After the unity testing, the "acceptance tests" are defined by the end-users.
- When that increment pass all the test, that user story is said to be finished.

In conclusion



−END−