Software Maintenance

Day 11 Software Maintenance

Charoen Vongchumyen

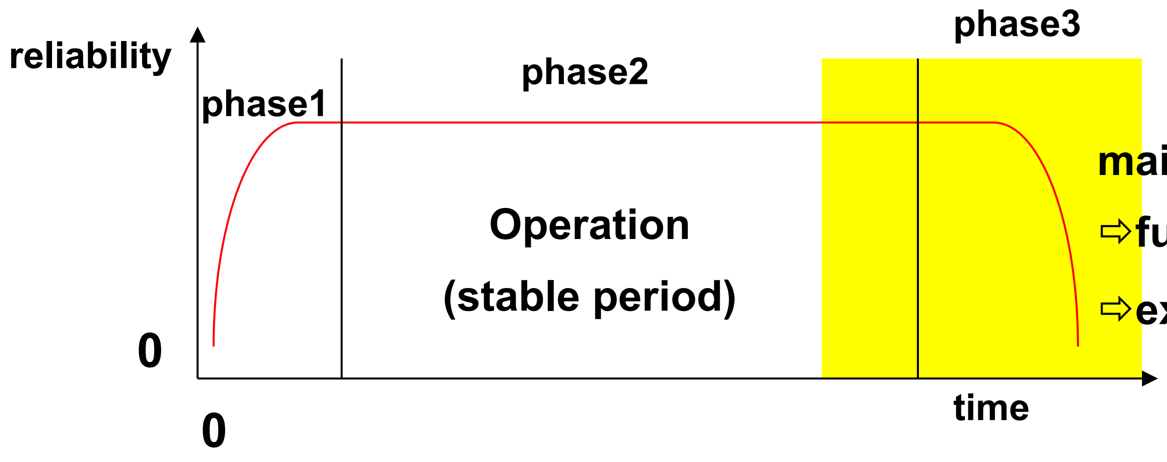
Email : charoen.vo@kmitl.ac.th

04/2021

Thank to Tsuneo Yamaura

Reliability Growth of Software and Hardware

Hardware reliability growth model (Bathtub model)

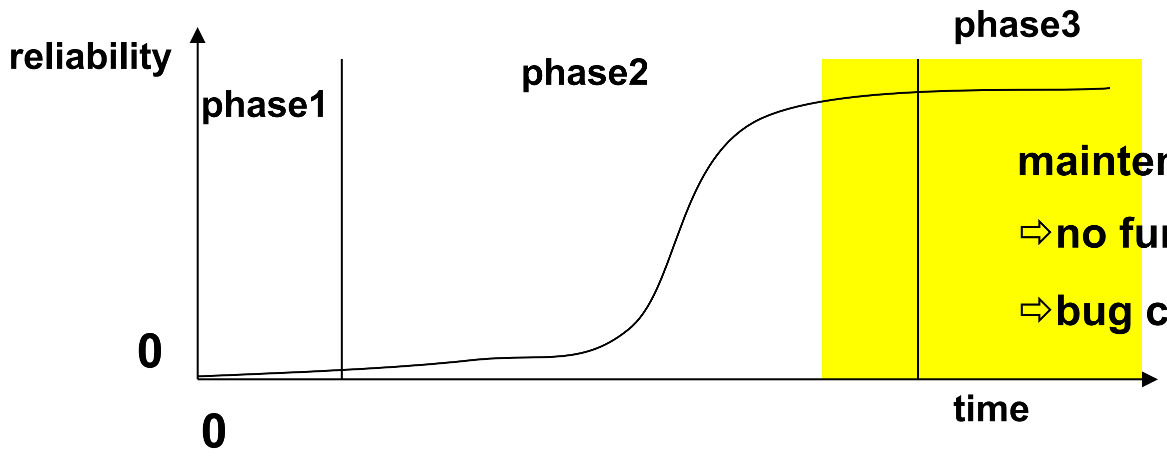


maintenance

⇨functional degradation due to aging

⇨exchange broken parts

Software reliability growth model (Gompertz Curve)



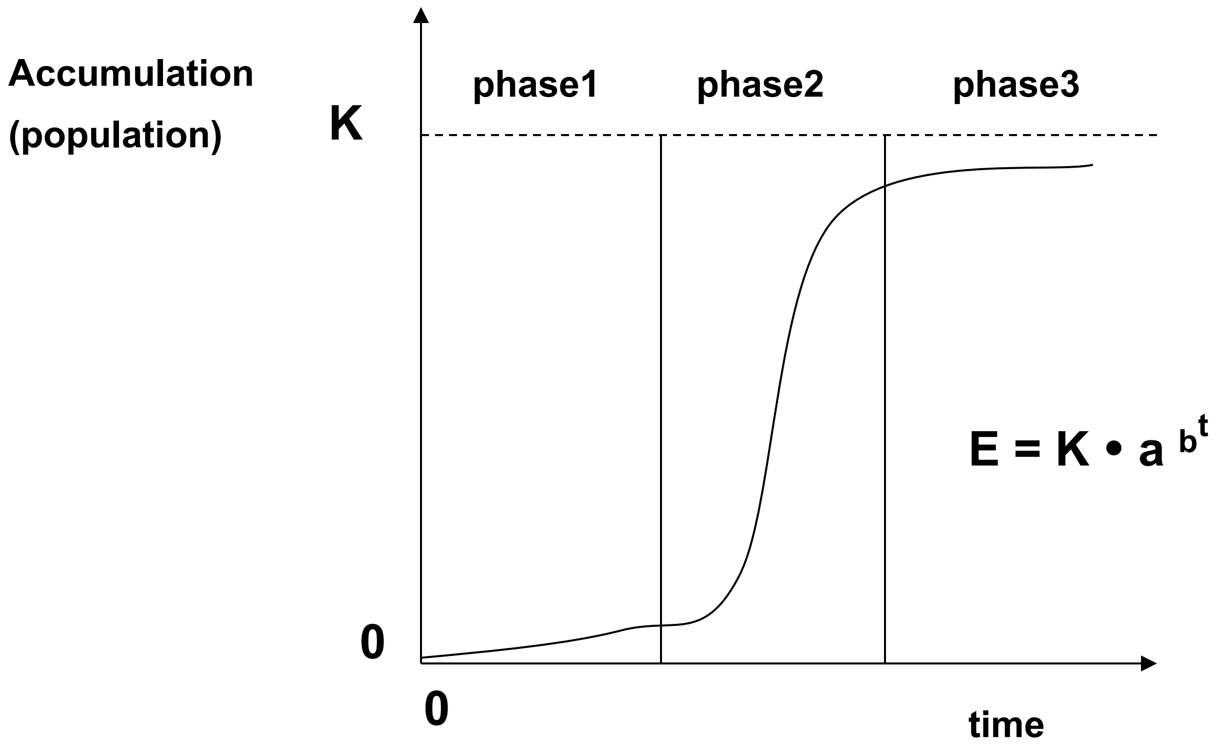
maintenance

⇨no functional degradation due to aging

⇨bug correction, functional enhancement

Gompertz Curve and Quality Growth Model

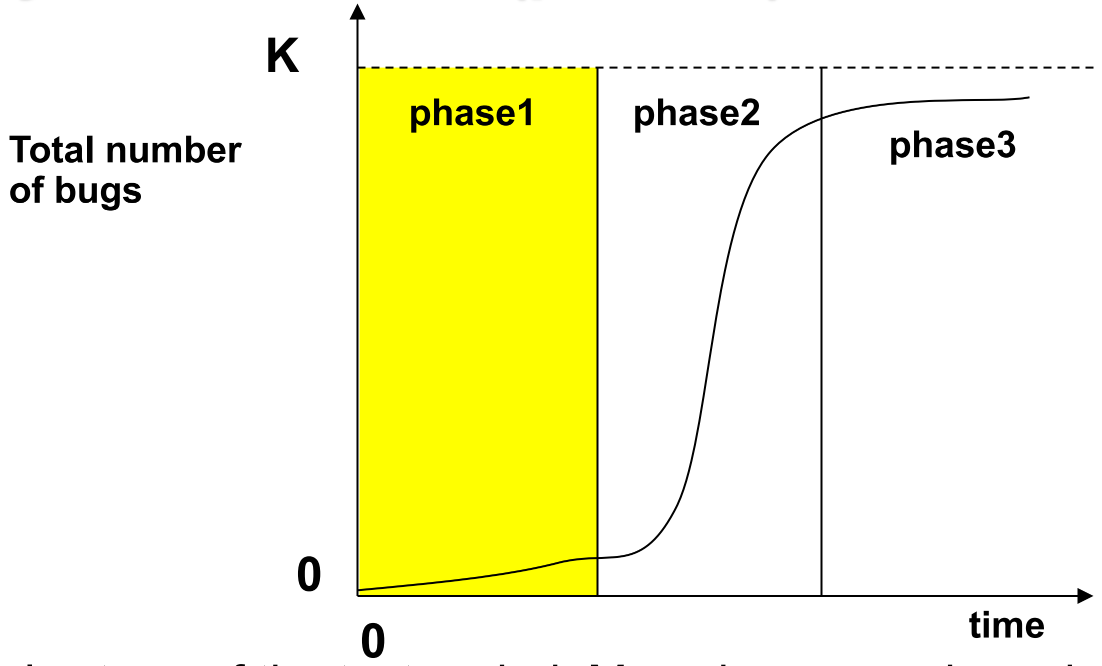
Overview of Gompertz Curve



Gompertz curve is S-shaped formulated curve that present phenomena that gradually grows like population increase.

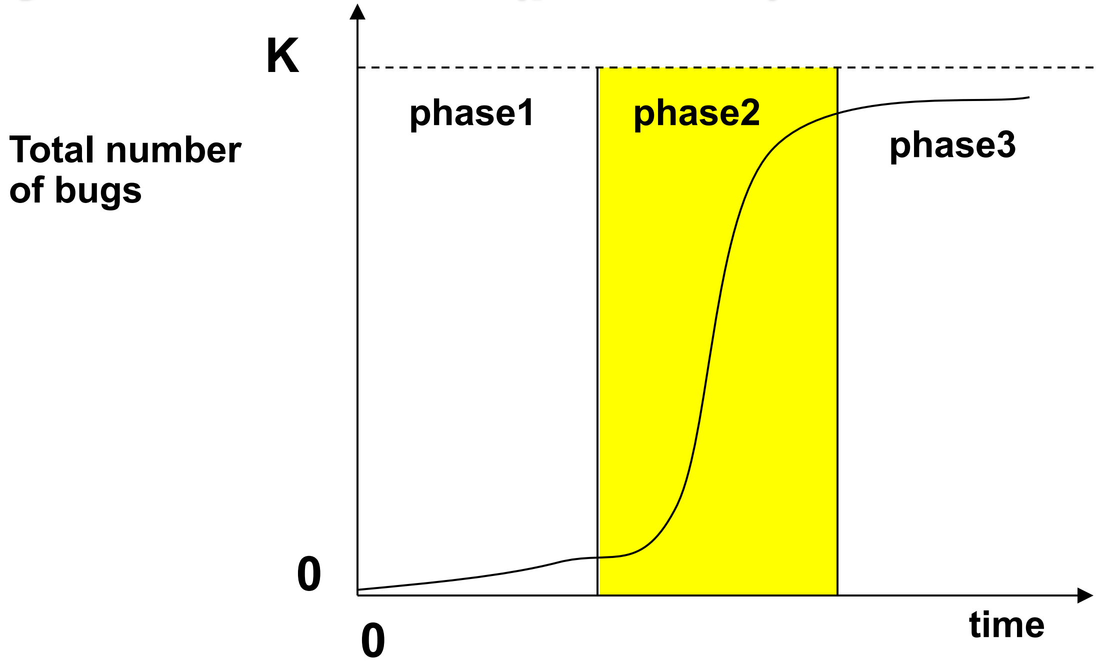
- •phase1 : preparation period
- •phase2 : rapid growing period
- •phase3 : saturation period

Quality Growth Model (phase 1)



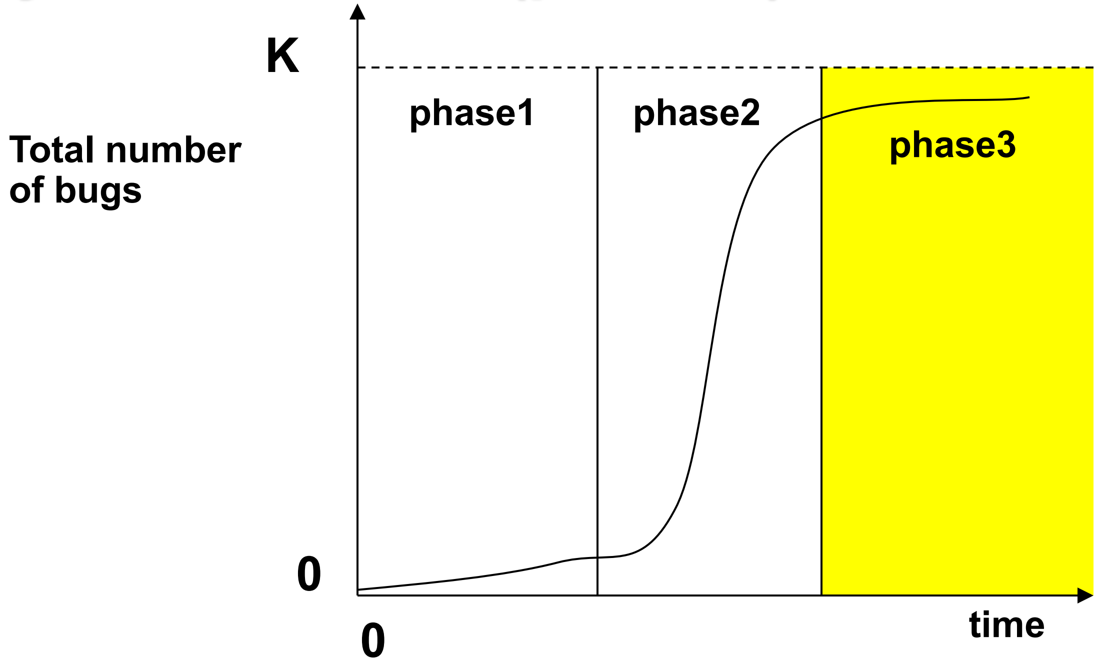
- Early stage of the test period. Many bugs remain undetected
- The 'Main Street' of the basic functions does not go through. Thus errors of basic functions are detected (e.g., missing function). It takes time to fix the bugs.
- A bug must be fixed to execute a test case : it takes time to successfully run a single test case
- A bug overshadows another. A correction of a bug may reveal another behind : it takes time to fix a bug

Quality Growth Model (phase 2)



- ‘Basic functions’ work without problems.
- Many bugs exist on the border and limitation. Since such bugs can be easy to reproduce, it does not take time for fixing.
- When many bugs are fixed there are chances of generating new bugs. It is highly recommended to run regression test periodically.

Quality Growth Model (phase 3)



- Basic functions and error cases work properly
- Bugs related to special cases and combinational cases (e.g., timing errors), and performance bugs will be detected. Such bugs are hard to reproduce, and very difficult to pinpoint what triggers the bugs

Definition of Maintenance in Software

- Software Maintenance in Old Days

  A process to fix bugs in the software released to customers.

- Software Maintenance in Current Days
  - Fixing the remaining errors is not many (17% of maintenance)
  - Majority is ‘to evolve the software to catch the upgrading customer’s system (hardware improvement) and changes of customer’s requirements. ⇨version-up
  - 60 / 60 rule

    ⇨ 60 % of the software development cost goes to maintenance, and 60 % of the maintenance cost goes to functional enhancement.

Changes in Software Development Lifecycle (part 1)

Typical software development lifecycle in old days (30 to 40 years ago)

Requirements specification	Functional specification	Detailed design	Coding	Debugging	Testing

- Maintenance = Bug Fixing

  Thus it was widely thought that a program of high quality did not need bug fixing, and the maintenance was assumed as unnecessary work.

⇩

- When a program is released to a customer, the original project takes on different development : 'bug fixing' of the previous program was handled by the original project on part-time basis.

Changes in Software Development Lifecycle (part 2)

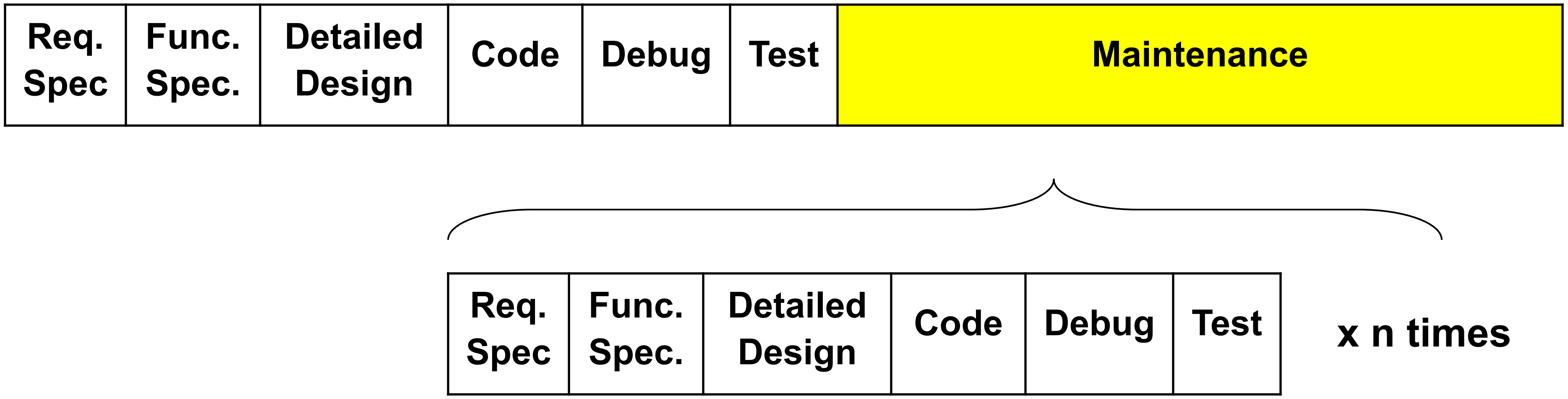
Typical software development lifecycle in 1980s (15 years ago)

Requirements specification	Functional specification	Detailed design	Coding	Debugging	Testing	Maintenance

- Maintenance is viewed as 'Functional enhancement'

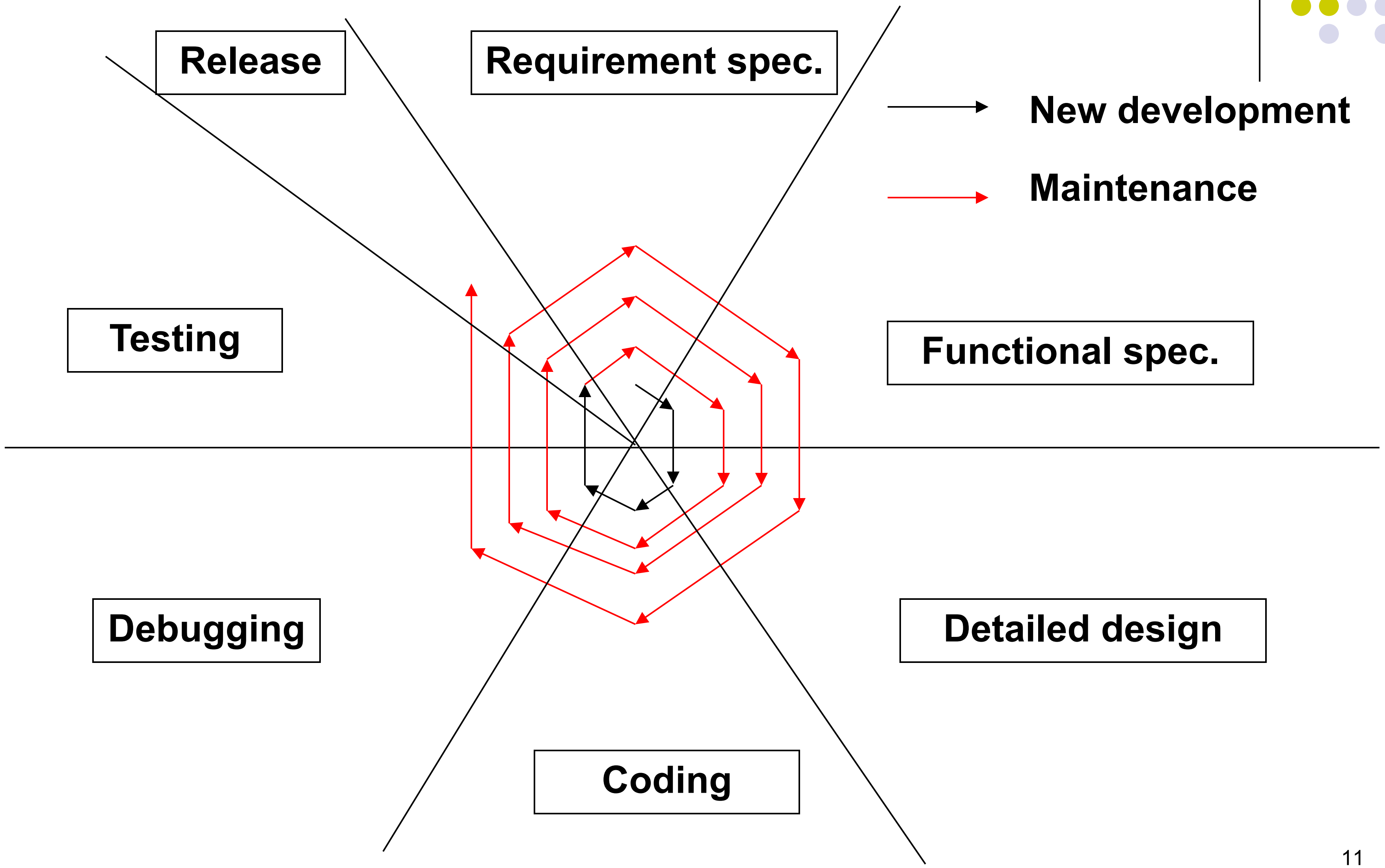
Changes in Software Development Lifecycle (part 3)

Typical software development lifecycle in current days



- It is widely understood that maintenance phase is for functional and performance enhancement.
- Current days, maintenance is the longest and most cost consuming phase in the software development lifecycle. ⇨approximately 40 % - 80 %

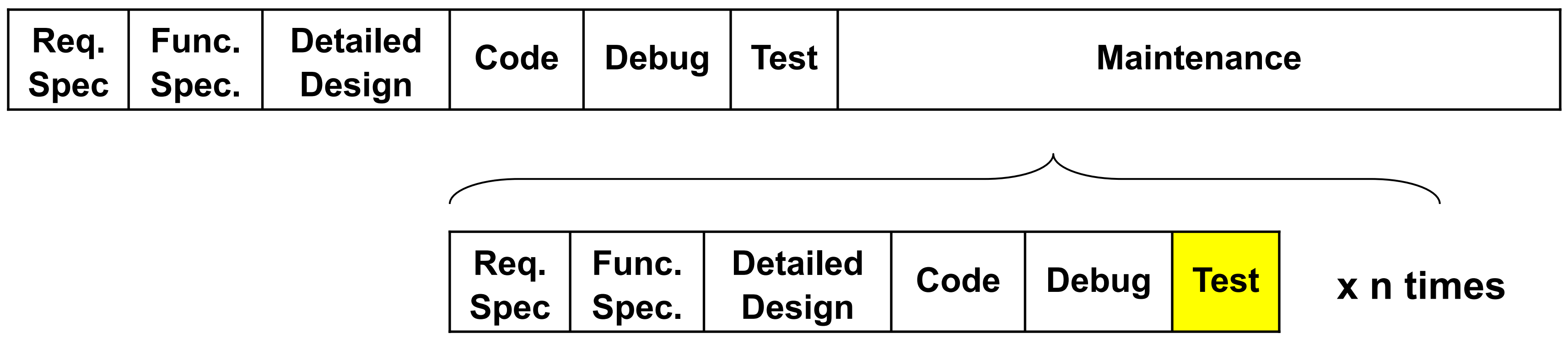
New development method : Spiral Model



Regression test (part 1)

- Special test required to maintenance
- Regression test is to detect ‘functional degradation.’

  Functional degradation : when one function is changed, completely different function does not work
- You have to periodically run the regression test in the maintenance phase



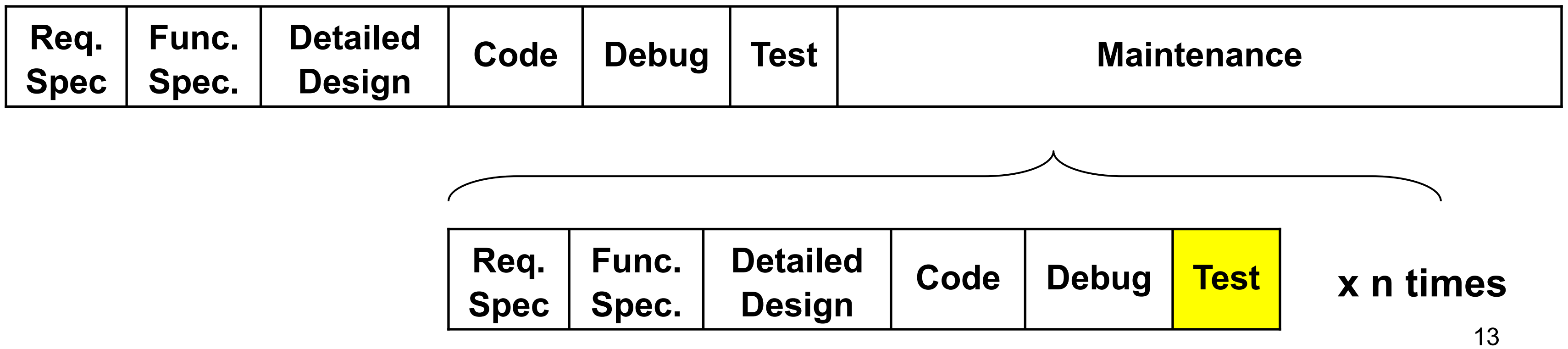
Regression test (part 2)

- The regression test is to prove that all the functions correctly work.
- How to design the regression test

  (1) Sample 5 % - 10 % of the test cases designed at the test phase, and run this regression test every time the program is changed (even in the case that a single character was changed).

  ⇨ Make the regression test a test suit for frequent execution

  (2) Every time functional enhancement and bug fixing were made, test cases for the enhancement and bug fixing must be added to the regression test cases (the regression test grows bigger).



Lifecycle of Maintenance

- Distribution of Time and Cost in New Development

• Requirement Specification	: 20 %
• Design	: 20 %
• Coding	: 20 %
• Testing and Debugging	: 40 %

- Distribution of Time and Cost in Maintenance

• Defining and understanding the changes	: 15 %
• Reviewing documents	: 5 %
• Understanding Source Code	: 25 %
• Testing and Debugging	: 30 %
• Updating Documents	: 5 %

Why Maintenance Looks Easy? (but actually super-difficult?)

(1) The original system is up and running.

⇨ An original developer tends to think that feasibility study is not necessary.

(2) When an original developer handles maintenance, he knows the details of the system structures and processing.

⇨ ‘I do not have to research the software’

⇨ Most likely the original team will be dismissed.

(3) The size of maintenance is rather small.

⇨ Tends to handle with power rather than systematic approach and sophisticated processing.

⇩

These caused terrible confusion in “Y2K”

A Case Study : Global Confusion of Y2K (part 1)

Background of Y2K

(1) In 1960s, 1970s, “using less data space” was much more important than “clean designing.” Thus, programmers used 2 digits for the year data rather than 4 digits

⇨ “Software development methodology” was premature, and programming was an art.

⇨ There was no concept of “maintenance.”

(2) Nobody could imagine that COBOL programs survived more than 30, 40 years (Is this like oil never runs out?).

⇨ Y2K became a global problem.

⇨ Rather than the Y2K itself, the IT industries were surprised to know that many COBOL programs were still up and running.

A Case Study : Global Confusion of Y2K (part 2)

Reasons of Confusions and Problems of Y2K

(1) The “Release date” was quite clear : 0 a.m. Jan 1st ,2000

⇨ Many projects made delayed starts

(2) The problems were very clear.

⇨ Everybody understood the problems but could not handle properly.

⇨ Was hard to pinpoint the source code to be modified.

⇨ Could not precisely estimate the period and person-power required to handle Y2K

⇨ Did not have enough (old) programmers who know COBOL

Issues of Maintenance (part 1)

(1) It is hard to understand the program structures and logics

- This is the biggest problem in maintenance.
- Why understanding is so difficult?
  - When moving from the requirement phase to design phase, the functions you must define rapidly expand (because, you have to define detailed functions).
  - There are many ways to design the requirement specification.
  - Designing is a complex and repetitive processing.
  - If the complexity of the target increases 25 %, the complexity of the solution is doubled (100 % more).
- Understanding the original source code is like reverse engineering.

Issues of Maintenance (part 1)

(2) Very frequently, “developer ≠ maintenance programmer”

(3) A maintenance programmer takes care of many LOC (10 to 100 more than the newly developed case).

- ⇨ Takes a lot of time to pinpoint the source code to be changed
- ⇨ If it is hard to pinpoint the source code to be modified, they will add new source code instead of changing the source code.
- ⇨ Verification of modification is not easy

⇩

- Hard keep the consistency of the software.
- The structure of the software may not be consistent.
- The software size increases.
- Many “untouched modules.”

Issues of Maintenance (part 3)

(4) A new bug may sneak into the software when a bug is fixed, or a function is enhanced.

- ⇨ functional degradation
- ⇨ Or, insufficient modification
- ⇨ Need to run regression test

(5) There is no design documents. If any, it is not kept up – to – the – minute.

- ⇨ In the worst case, you can rely on only the source code.
- ⇨ This is the major reason why the engineers hate to take care of maintenance.

(6) Misunderstanding to maintenance

- ⇨ Maintenance is not a creative job.
- ⇨ Maintenance is a sort of garbage cleaning?

Issues of Maintenance (part 4)

What is the Major Issue in Software Maintenance?

Data From U.S. Navy 1988

1 st place (8.7 / 10 points) : Engineers left (to different companies)

2 nd place (7.5 / 10 points) : Hard to understand the program.

Not enough documents.

3 rd place (6.9 / 10 points) : Pinpointing the source code to be changed

Four Types of Current Maintenance

(1) Bug fixing (17 % of maintenance cost)

(2) Functional and performance enhancement (60 %)

(3) Porting (18 %)

(4) Preventive maintenance (5 %)

Four Types of Current Maintenance : Bug Fixing

- 17 % of the maintenance cost
- Type of Bugs to Be Fixed
  - Bugs reported by customers

    ⇨ Actual system does not work as the customers required.
  - Back logs

    ⇨ Bugs that were detected but not fixed to keep the schedule
- Fixing bugs at maintenance phase is quite expensive !

  ⇨ Many works
  - Changing / fixing requirement specification, functional specification, design document, test specification, manuals.
  - Re – installing, and testing at user site (User's system must be stopped)

  ⇨ The original development project has been dismissed

Four Types of Current Maintenance : Functional and Performance Enhancement

- 60 % of the maintenance cost
- Type of Functional Enhancement
  - Customers say, “This is what we asked, but not what we need”
  - Customer’s business expands / changes, hardware changes.

    ⇨ functional and performance enhancement
  - Back logs

    ⇨ Implementation of functions that were deleted in the first version due to keep the schedule.

    ⇨ Average amount of back logs (in 1988)

• PC programs	: 19 months
• Micro Computer	: 26 months
• Main Frame Programs	: 46 months

Four Types of Current Maintenance : Porting

- 18 % of the maintenance cost
- Type of Porting
  - Different OS's
  - Different hardware
  - Different interface of new software packages

Four Types of Current Maintenance :
Preventive Maintenance

- 5 % of the maintenance cost
- Maintenance to reduce future problems. Is called “re – factoring.”
  - ⇨ Keeping the software flexible to easily handle customers. future requests and changes of operational environment.
  - ⇨ Needs time and money : “Rich man’s Maintenance”
- Example of Preventive Maintenance
  - Handling hardware change (e.g., supporting new device drivers)
  - Handling future OS’s and middleware (e.g., interface wrapping)
  - Preparing future functional enhancement (e.g., Framework: increase the module and functional independency, and localizes the changes)