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

Maintenance

Introduction

- After the software product is developed and delivered to users, it enters the software operation and maintenance phase. This stage is the last stage of the software life cycle, and its basic task is to ensure that the software can run normally for a long period of time.
- Software maintenance requires a large amount of work. On average, the maintenance cost of large-scale software is as high as about 4 times the development cost. At present, many foreign software development organizations use more than 60% of their manpower to maintain existing software, and as the number of software increases and the service life extends, this percentage continues to rise. In the future, maintenance work may even constrain the hands and feet of software development organizations, so that they have no spare capacity to develop new software.
- The purpose of software engineering is to improve the maintainability of the software, reduce the workload required for software maintenance, and reduce the total cost of the software system.

Definition of software maintenance

- The so-called software maintenance is the process of modifying the software in order to correct errors or meet new needs after the software has been delivered for use.
- The software maintenance can be specifically defined by describing the four activities that may be carried out after the software is delivered and used.
  - Because software testing cannot reveal all the hidden errors in a large software system, there must be the first maintenance activity: during the use of any large program, users will inevitably find program errors and report the problems they encounter To the maintenance staff.
    - The process of diagnosing and correcting errors is called corrective maintenance.
  - All aspects of computer science and technology are advancing rapidly. About every 36 months, a new generation of hardware is announced, new operating systems or modified versions of old systems are often introduced, and external devices and other system components are frequently added or modified; On the other hand, the service life of application software can easily exceed 10 years, which is much longer than the service life of the operating environment when the software was originally developed. Therefore, adaptive maintenance, that is, the activity of modifying software in order to properly cooperate with the changed environment, is a necessary and frequent maintenance activity.

Definition of software maintenance

- The so-called software maintenance is the process of modifying the software in order to correct errors or meet new needs after the software has been delivered for use.
- The software maintenance can be specifically defined by describing the four activities that may be carried out after the software is delivered and used.
  - When a software system runs smoothly, there is often a third maintenance activity: in the process of using the software, users often suggest adding new functions or modifying existing functions, and may also put forward general suggestions for improvement. In order to meet such requirements, comprehensive maintenance is required. This maintenance activity usually accounts for most of the software maintenance work.
  - When the software is modified to improve future maintainability or reliability, or to lay a better foundation for future improvements, the fourth maintenance activity occurs. This maintenance activity is usually called preventive maintenance. Currently, this maintenance activity is relatively small.

Definition of software maintenance

- From the above definition of software maintenance, it is not difficult to see that software maintenance is by no means limited to correcting errors found in use. In fact, more than half of all maintenance activities are complete maintenance. Foreign statistics show that complete maintenance accounts for 50% to 66% of all maintenance activities, corrective maintenance accounts for 17% to 21%, adaptive maintenance accounts for 18% to 25%, and other maintenance activities only account for about 4%.
- It should be noted that the above four types of maintenance activities must be applied to the entire software configuration. Maintaining software documentation and maintaining software executable code are equally important.

Features of software maintenance

- Unstructured maintenance
  - If the only component of the software configuration is the program code, then the maintenance activities start with the arduous evaluation of the program code, and often the evaluation is more difficult due to insufficient internal documentation of the program.
  - For the software structure, the entire data structure, the system interface, performance and/or misunderstandings are often caused by design constraints, and the consequences of changes to the program code are also difficult to estimate:
    - because there is no test documentation, it is impossible to perform regression testing (that is, to ensure that the changes made are not normal before The software function used introduces errors and repeats the tests done in the past).
    - Unstructured maintenance requires a high price (wasted energy and suffers frustration).
    - This maintenance method is the inevitable result of software developed without a well-defined methodology.

Features of software maintenance

- Structured maintenance
  - If there is a complete software configuration, then the maintenance work starts from evaluating the design documents, determining the important structural characteristics, performance characteristics and interface characteristics of the software; estimating the impact of the required changes, and planning the implementation path.
  - Then first modify the design and carefully review the modifications.
  - Next, write the corresponding source code; use the information contained in the test specification for regression testing; finally, deliver the modified software for use again.

The high cost of maintenance

- Over the past few decades, the cost of software maintenance has steadily increased. In 1970, the cost of maintaining existing software only accounted for 35% to 40% of the total software budget. In 1980, it rose to 40% to 60%, and in 1990 it rose to 70% to 80%.
- Maintenance costs are only the most obvious cost of software maintenance, and other costs that are not yet obvious may be more concerned in the future. Because the available resources must be used for maintenance tasks, the development opportunity is delayed or even lost. This is an intangible price for software maintenance. Other intangible costs include:
- Users will be dissatisfied when the seemingly reasonable requirements for correction or modification cannot be met in time;
- Due to changes during maintenance, latent errors are introduced into the software, thereby reducing the quality of the software;
- When software engineers must be transferred to maintenance work, it will cause confusion in the development process.
- The last price of software maintenance is a significant drop in productivity, which is often encountered when maintaining old programs.

The high cost of maintenance

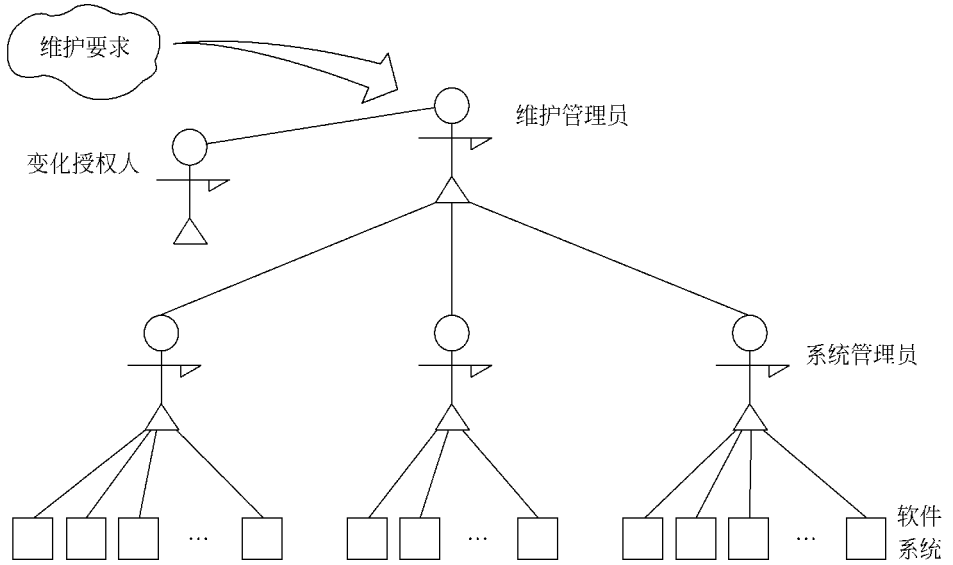
- Most of the problems related to software maintenance can be attributed to the shortcomings of software definition and software development methods. The lack of strict and scientific management and planning in the first two periods of the software life cycle will almost inevitably lead to problems in the final stage. Some issues related to software maintenance are listed below:
  - (1) It is usually very difficult to understand programs written by others, and the degree of difficulty increases rapidly as the software configuration components decrease. If there is only program code without documentation, serious problems will occur.
  - (2) The software that needs to be maintained often does not have qualified documentation, or the documentation is significantly insufficient. Recognizing that software must have documentation is only the first step. Documentation that is easy to understand and fully consistent with the program code is really valuable.
  - (3) When the software is required to be maintained, the developer cannot be expected to give us a detailed description of the software. Because the maintenance phase lasts for a long time, when the software needs to be explained, the person who wrote the program is often no longer nearby.
  - (4) The vast majority of software is designed without considering future modifications. Unless a design methodology that emphasizes the principle of module independence is used, it is difficult and error-prone to modify the software.
  - (5) Software maintenance is not an attractive job. This concept is formed largely because maintenance work often suffers from setbacks.
- The above-mentioned problems exist more or less in the existing software developed without software engineering ideas. A scientific methodology should not be regarded as a panacea, but software engineering at least partially solves every problem related to maintenance.

Software maintenance process

- The maintenance process is essentially a modified and compressed software definition and development process, and in fact, the work related to software maintenance has already begun long before a maintenance request is made.
  - First, a maintenance organization must be established, then the reporting and evaluation process must be determined, and a standardized sequence of events must be specified for each maintenance requirement.
  - In addition, a record keeping process suitable for maintenance activities should be established, and review standards should be specified.

Maintenance Organization

- Although there is usually no need to establish a formal maintenance organization, even for a small software development group, informal delegation of responsibility is absolutely necessary. Each maintenance requirement is passed to the corresponding system administrator for evaluation by the maintenance administrator. System administrators are technicians who are designated to be familiar with a small part of the product program. After the system administrator evaluates the maintenance task, the change authorizer decides the activities that should be performed
- It is very necessary to clarify the maintenance responsibilities before the maintenance activities start. Doing so can greatly reduce the confusion that may occur during the maintenance process.

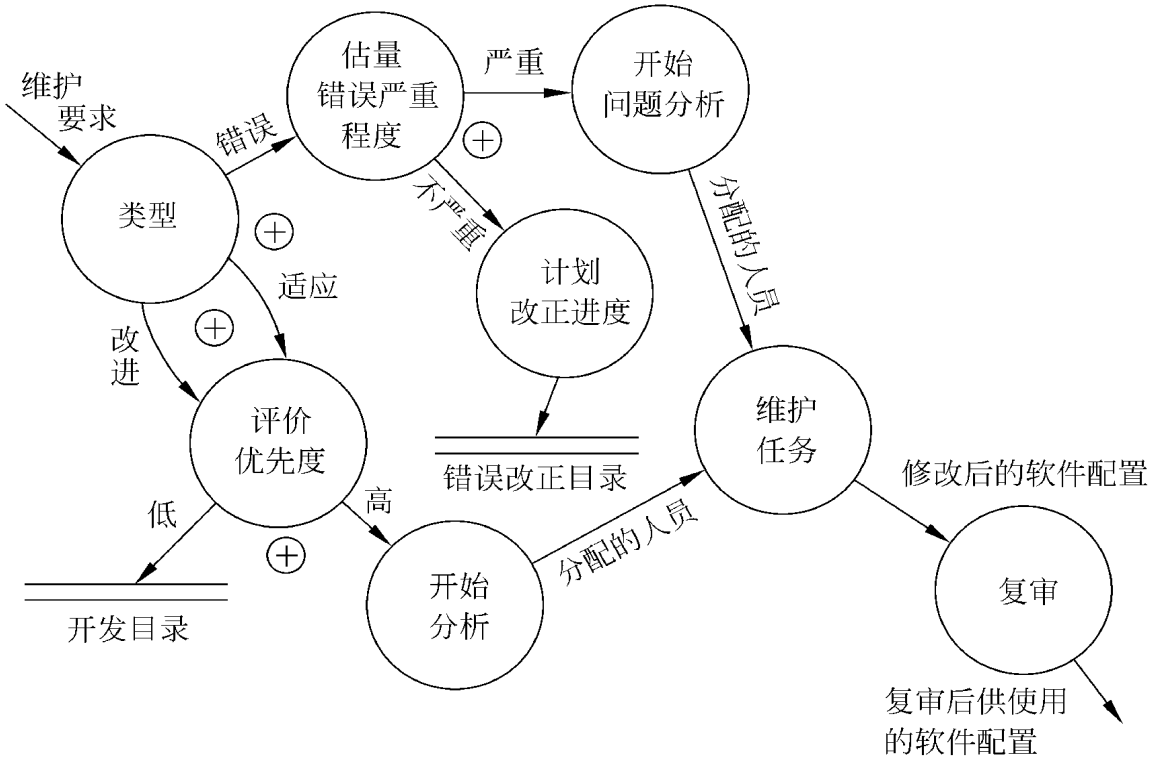


Maintenance report

- All software maintenance requirements should be expressed in a standardized format. Software maintainers usually provide users with a blank maintenance request form—sometimes called a software problem report form, which is filled out by users who request a maintenance activity.
  - If you encounter an error, you must fully describe the environment that caused the error (including input data, all output data, and other relevant information).
- For adaptive or complete maintenance requirements, a short requirements specification should be proposed. As mentioned earlier, the maintenance request form submitted by the user is evaluated by the maintenance administrator and the system administrator.
  - The maintenance requirement form is an externally generated document, which is the basis for planning maintenance activities. The software organization should draw up a software modification report, which gives the following information:
    - (1) The amount of work required to meet the requirements set out in the maintenance requirements table;
    - (2) The nature of maintenance requirements;
    - (3) The priority of this request;
    - (4) after-the-fact data related to the modification.
- Before drawing up a further maintenance plan, submit the software modification report to the change authorizer for review and approval.

Maintained event stream

- You should first determine the type of maintenance required. Users often regard a requirement as correcting software errors (corrective maintenance), while developers may regard the same requirement as adaptive or perfect maintenance. When there are disagreements, they must be resolved through negotiation.



Maintained event stream

- The processing of a corrective maintenance request (the "wrong" path in the figure) starts with the estimation of the severity of the error. If it is a serious error (for example, a critical system cannot operate normally), assign personnel under the guidance of the system administrator and immediately begin the problem analysis process. If the error is not serious, then corrective maintenance and other tasks that require software development resources are planned together.
- The requirements for adaptive maintenance and complete maintenance follow the same event flow path. The priority of each maintenance requirement should be determined, and the required working hours should be arranged as if it were another development task (from all intents and goals, it belongs to development work). If the priority of a maintenance request is very high, maintenance work may begin immediately.
- Regardless of the type of maintenance, the same technical work is required. These tasks include revision of the software design, review, necessary code modification, unit testing and integration testing (including regression testing using the previous testing scheme), acceptance testing and review. Different types of maintenance emphasize different points, but the basic approach is the same. The last event in the maintenance event stream is a review, which again checks the validity of all components of the software configuration and ensures that the requirements in the maintenance requirements table are actually met.
- Of course, there are also maintenance requirements that do not fully meet the above-mentioned event stream. When a vicious software problem occurs, a so-called "fire fighting" maintenance requirement arises. In this case, resources need to be used to solve the problem immediately. If "fire fighting" is a common process for an organization, then its management and technical capabilities must be doubted.

Maintained event stream

- After completing a software maintenance task, it is often beneficial to conduct a situation review. Generally speaking, this review attempts to answer the following questions:
  - Which aspects of design, coding or testing can be done in different ways in the current situation?
  - Which maintenance resources are supposed to be available when they are not?
  - What are the major (and minor) obstacles to this maintenance work?
  - Is there preventive maintenance among the required maintenance types?
  - Situation review has an important impact on the future maintenance work, and the feedback provided is very important for the effective management of the software organization.

Keep maintenance records

- For all stages of the software life cycle, the previous record keeping was inadequate, and software maintenance was not recorded at all. For this reason, it is often impossible to evaluate the effectiveness of maintenance technology, to determine the "goodness" of a product program, and it is difficult to determine the actual cost of maintenance.
- The first problem encountered in maintaining maintenance records is, which data is worth recording? Swanson proposed the following:
  - program identification;
  - number of source sentences;
  - number of machine instructions;
  - program design language used;
  - program Date of installation;
  - Number of program runs since installation;
  - Number of program failures since installation;
  - Level and identification of program changes;
  - Number of source sentences increased due to program changes;
  - Number of source sentences deleted due to program changes ;
  - Number of man-hours consumed for each change;
  - the date of the program change;
  - the name of the software engineer;
  - the identification of the maintenance request form;
  - the maintenance type;
  - the date of the maintenance start and completion;
  - the accumulated man-hours for the maintenance;
  - relative to the completed maintenance
  - The pure benefit of contact.
- The above data should be collected for every maintenance task. You can use these data to form a basis for maintaining the database, and evaluate them as described below.

Evaluation of maintenance activities

- Without valid data, maintenance activities cannot be evaluated. If you have started to save maintenance records, you can make some quantitative measurements of maintenance work. Maintenance work can be measured at least from the following 7 aspects:
  - (1) The average number of failures per program run;
  - (2) The total number of man-hours used for each type of maintenance activity;
  - (3) Average number of program changes made per program, language, and maintenance type;
  - (4) The average number of man-hours spent on adding or deleting a source sentence during the maintenance process;
  - (5) Maintain the average number of human hours spent in each language;
  - (6) The average turnaround time of a maintenance request form;
  - (7) Percentage of different maintenance types.
- Based on the results of quantitative measurement of maintenance work, decisions about development technology, language selection, maintenance workload planning, resource allocation, and many other aspects can be made, and such data can be used to analyze and evaluate maintenance tasks.

Software maintainability

- The maintainability of the software can be qualitatively defined as: how easy it is for the maintainer to understand, correct, change or improve the software. In the previous chapters, it has been emphasized many times that improving maintainability is the key goal that governs all steps of software engineering methodology.

Factors that determine software maintainability

- Maintenance is the modification made after the software is delivered to use. Before modification, the object to be modified must be understood, and necessary tests should be carried out after modification to ensure that the modification is correct. If it is corrective maintenance, it must also be debugged in advance to determine the specific location of the error. Therefore, the main factors that determine the maintainability of the software are as follows:
  - 1. Comprehensibility
    - Software comprehensibility is expressed as the degree of difficulty for outside readers to understand the structure, function, interface and internal processing of the software. Modularity (good module structure, high cohesion, loose coupling), detailed design documentation, structured design, internal program documentation and a good high-level programming language, etc., all have important contributions to improving the intelligibility of the software.
  - 2. Testability
    - The ease of diagnosis and testing depends on how easy the software is to understand. Good documentation is essential for diagnosis and testing. In addition, software structure, available testing tools and debugging tools, and previously designed testing procedures are also very important. The maintainer should be able to get the test plan used in the development phase for regression testing. At the design stage, every effort should be made to design the software to be easy to test and easy to diagnose.
    - For program modules, program complexity can be used to measure its testability. The greater the loop complexity of the module, the more executable paths, therefore, the more difficult it is to fully test it.

Factors that determine software maintainability

- Maintenance is the modification made after the software is delivered to use. Before modification, the object to be modified must be understood, and necessary tests should be carried out after modification to ensure that the modification is correct. If it is corrective maintenance, it must also be debugged in advance to determine the specific location of the error. Therefore, the main factors that determine the maintainability of the software are as follows:
  - 3. Modification
    - The degree to which the software is easy to modify is directly related to the design principles and heuristic rules discussed in Chapter 5 of this book. Coupling, cohesion, information hiding, localization, the relationship between control domain and scope, etc., all affect the modifiability of software.
  - 4. Portability
    - Software portability refers to the difficulty of transferring programs from one computing environment (hardware configuration and operating system) to another computing environment. The program codes related to hardware, operating system and other external devices are concentrated in specific program modules, and the programs that must be modified due to environmental changes can be limited to a few program modules, thereby reducing the difficulty of modification.

Factors that determine software maintainability

- Maintenance is the modification made after the software is delivered to use. Before modification, the object to be modified must be understood, and necessary tests should be carried out after modification to ensure that the modification is correct. If it is corrective maintenance, it must also be debugged in advance to determine the specific location of the error. Therefore, the main factors that determine the maintainability of the software are as follows:
  - 5. Reusability
    - The so-called reuse (reuse) refers to the same thing repeatedly used in different environments without modification or slight changes. The extensive use of reusable software components to develop software can improve the maintainability of the software from the following two aspects:
      - (1) Generally, reusable software components undergo rigorous testing during development, and their reliability is relatively high, and some errors will be found and cleared during each reuse process. Over time, such components will become essential There is no error on it. Therefore, the more reusable components used in the software, the higher the reliability of the software, and the less the need for corrective maintenance.
      - (2) It is easy to modify the reusable software components so that they can be re-applied in the new environment. Therefore, the more reusable components used in the software, the easier it is to maintain adaptability and integrity.

Documentation

- Documentation is the decisive factor affecting the maintainability of the software. Since large-scale software systems that have been used for a long time will inevitably undergo multiple modifications during use, documentation is more important than program code.
- The documentation of the software system can be divided into two categories: user documentation and system documentation. User documentation mainly describes system functions and usage methods, and does not care how these functions are implemented; system documentation describes various aspects of system design, implementation, and testing.
- In general, the software documentation should meet the following requirements:
  - (1) How to use this system must be described, even the simplest system cannot be used without this description;
  - (2) Must describe how to install and manage this system;
  - (3) The system requirements and design must be described;
  - (4) The implementation and testing of the system must be described in order to make the system maintainable.
  - The user documentation and system documentation are discussed below.

Documentation

- 1. User documentation
  - User documentation is the first step for users to understand the system. It should enable users to get an accurate initial impression of the system. The structure of the document should enable users to easily read the relevant content as needed.
  - User documentation should include at least the following 5 aspects:
    - (1) Function description, explaining what the system can do;
    - (2) Installation documents, explaining how to install the system and how to adapt the system to a specific hardware configuration;
    - (3) The user manual, which briefly explains how to start using the system (should explain how to use commonly used system functions through rich examples, and also explain how to recover and restart in case of user operation errors);
    - (4) The reference manual, which describes in detail all the system facilities that users can use and how to use them. It should also explain the meaning of various error messages that may be generated by the system (the most important requirement for the reference manual is complete, so it is usually formalized. Description technology);
    - (5) The operator's guide (if a system operator is required), explaining how the operator should deal with various situations in use.
  - The above content can be used as independent documents, or as different sub-volumes of a document. The specific method should be determined by the scale of the system.

Documentation

- 2. System documentation
  - The so-called system documentation refers to a series of documents related to system implementation from problem definition, requirement description to acceptance test plan. Documents describing system design, implementation, and testing are extremely important for understanding and maintaining procedures. Similar to the user documentation, the structure of the system documentation should also guide the reader from an overview of the system to a more formal and specific understanding of each aspect and feature of the system. The previous chapters of this book have introduced the documents that should be generated at each stage in more detail, so I won't repeat them here.

Maintainability review

- Maintainability is a basic feature that all software should have. It must be ensured that the software has the maintainable factors during the development phase. At each stage of the software engineering process, we should consider and strive to improve the maintainability of the software. In the technical review and management review before the end of each stage, the maintainability should be reviewed.
- In the review process of the requirements analysis stage, the parts to be improved and the parts that may be modified should be noted and indicated; the portability of the software should be discussed, and the system interface that may affect the maintenance of the software should be considered.
- During the formal and informal design review period, the software structure and process should be evaluated based on the goals of easy modification, modularity and functional independence; the design should be prepared for the parts that may be modified in the future.
- Code review should emphasize the two factors that affect maintainability, coding style and internal documentation.
- Reusable software components should be used as much as possible in the design and coding process. If new components need to be developed, attention should be paid to improving the reusability of components.

Maintainability review

- Each test step can imply that preventive maintenance may be required in the program before the software is officially delivered for use. The most formal maintainability review is performed at the end of the test. This review is called a configuration review. The purpose of configuration review is to ensure that all components of the software configuration are complete, consistent and understandable, and have been cataloged and archived for the convenience of modification and management.
- After completing each maintenance work, the software maintenance itself should be carefully reviewed.
- Maintenance should focus on the entire software configuration, and should not only modify the source code. When the modification of the source code is not reflected in the design document or user manual, serious consequences will occur.

Maintainability review

- Whenever changes are made to the data, software structure, module process or any other related software features, the corresponding technical documents must be modified immediately. Design documents that do not accurately reflect the current state of the software may be worse than no documents at all. In the future maintenance work, it is likely that the software cannot be understood correctly because the documentation does not completely conform to the actual situation, thus introducing too many errors in the maintenance.
- Users usually use and evaluate software based on user documents that describe the characteristics and usage of the software. If the modification to the executable part of the software is not reflected in the user documentation in a timely manner, it will inevitably make the user dissatisfied because of frustration.
- If the software configuration is strictly reviewed before the software is re-delivered for use, document problems can be greatly reduced. In fact, some maintenance requirements may not require modification of the software design or source code, but simply indicate that the user documentation is unclear or inaccurate, so only necessary maintenance of the documentation is required.

Preventive maintenance

- Almost all software development organizations with a long history have some "old" programs that were developed more than a decade ago. At present, some old programs are still serving users, but the software engineering methodology was not used to guide the development of these programs. Therefore, the architecture and data structure of these programs are very poor, and the documentation is incomplete or even completely undocumented. There is no complete record of the changes that have been made.
- How to meet the user's maintenance requirements for the above-mentioned old programs? In order to modify this type of program to adapt to the new or changed needs of users, there are several options:
  - (1) Make repeated attempts to modify the program, and "fight tenaciously" with the invisible design and source code to achieve the required modification;
  - (2) Master the internal working details of the program as much as possible through careful analysis of the program in order to modify it more effectively;
  - (3) On the basis of in-depth understanding of the original design, use software engineering methods to redesign, recode and test those parts of the software that need to be changed;
  - (4) Under the guidance of software engineering methodology, the program is completely redesigned, recoded and tested. For this purpose, CASE tools (reverse engineering and reengineering tools) can be used to help understand the original design.

Preventive maintenance

- The first approach is very blind, usually people use the latter three approaches. The fourth approach is called software reengineering. Such maintenance activities are the preventive maintenance mentioned in section 8.1 of this chapter, while the third approach is essentially partial reengineering.
- The preventive maintenance method was proposed by Miller, who defined this method as: "Applying today's methodology to yesterday's system to support tomorrow's needs."
- At first glance, it seems like a waste to redevelop a large program when a working version of the program already exists. In fact, it is not. The following facts can explain the problem:
  - (1) The cost of maintaining a line of source code may be 14-40 times the cost of initially developing the line of source code;
  - (2) Modern design concepts are used when redesigning the software architecture (program and data structure), which may be of great help to future maintenance;
  - (3) Since the existing program version can be used as a software prototype, the development productivity can be much higher than the average level;
  - (4) The user has more experience in using the software, therefore, it is easy to make fresh changes to the requirements and scope of changes;
  - (5) Use reverse engineering and reengineering tools to automate part of the work;
  - (6) A complete software configuration can be established in the process of completing preventive maintenance.

Coding style

- 5. Efficiency
- Efficiency mainly refers to two aspects: processor time and memory capacity.
- (3) Input and output efficiency
  - If the user's mental work is economical in order to provide input information to the computer or to understand the information output by the computer, then the efficiency of communication between humans and computers is high. Therefore, simplicity and clarity are also the key to improving the efficiency of human-machine communication.
  - The communication efficiency between hardware is a very complicated issue, but from the point of view of writing programs, there are some simple principles that can improve the efficiency of input and output. e.g:
    - All input and output should be buffered to reduce the overhead for communication;
    - The simplest access method should be selected for secondary storage (such as disk);
    - The input and output of the secondary storage should be carried out in units of information groups;
    - If the "super efficient" input and output are difficult to understand, this method should not be used.
    - These simple principles apply to both the design and coding phases of software engineering.