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

General design

Introduction

- The basic purpose of the overall design
  - Answer "In a nutshell, how should the system be implemented?"
- Overall design is also called outline design or preliminary design
- Target
  - Through this stage of work, the physical elements that make up the system will be divided
    - Programs, files, databases, manual processes and documents, etc.
      - But each physical element is still at the black box level, and the specific content in the black box will be carefully designed in the future
  - Design the structure of the software
    - Determine which modules each program in the system is composed of, and the relationship between the modules

Introduction

- Process task
  - Looking for different solutions to achieve the target system
    - The data flow diagram obtained in the demand analysis stage is the basis for conceiving various possible solutions
  - The analyst selects a number of reasonable options from these alternative options
    - Prepare a system flow chart for each reasonable plan, list all the physical elements that make up the system, conduct a cost/benefit analysis, and develop a schedule to realize the plan
    - The analyst should comprehensively analyze and compare these reasonable solutions, and select the best solution to recommend to the user and the person in charge of the user department.
  - If the user and the person in charge of the user department accept the recommended solution, the analyst should further design the software structure for the best solution
    - Including: after designing the preliminary software structure, it needs to be improved in many ways, so as to obtain a more reasonable structure, carry out the necessary database design, determine the test requirements and formulate the test plan

The general design process

- Two major stages
  - At the system design stage, determine the specific implementation plan of the system
  - Determine the software structure at the structural design stage
- Nine steps
  - plan alternatives
  - Choose a reasonable plan
  - Recommend the best solution
  - Functional decomposition
  - Design software structure
  - Design database
  - Develop a test plan
  - Write a document
  - Examine and review

Plan alternatives

- In the general design stage, the analyst should consider various possible implementation schemes and strive to choose the best scheme from them.
  - At the beginning of the general design phase, there is only the logical model of the system, and analysts have full freedom to analyze and compare different physical implementation schemes.
    - Once the best solution is selected, the performance/price ratio of the system will be greatly improved
- The data flow diagram obtained during the demand analysis stage is an excellent starting point for the general design
  - A common method in the process of conceiving alternative solutions
    - Plan various possible methods of grouping the processing in the data flow diagram, discarding technically impractical grouping methods (for example, the execution time of different processing in the group is incompatible), and the remaining grouping methods represent possible implementation strategies, and could reveal the alternative physical systems

Plan alternatives

- Several reasonable solutions should be selected from a series of alternative solutions obtained in the previous step, usually at least three solutions of low cost, medium cost and high cost are selected.
- When judging which solutions are reasonable, the project scale and goals determined in the problem definition and feasibility study phases should be considered, and sometimes it may be necessary to further seek the opinions of users.
- For every reasonable solution analyst should prepare the following 4 pieces of information:
  - System flow chart
  - List of physical elements that make up the system
  - Cost-benefit analysis
  - The schedule for implementing this system

Recommend the best solution

- The analyst should comprehensively analyze and compare the pros and cons of various reasonable solutions, recommend the best solution, and develop a detailed implementation plan for the recommended solution
- Users and relevant technical experts should carefully review the best system recommended by the analyst
  - If the system does meet the needs of users and is fully achievable under existing conditions, it should be submitted to the person in charge of the user department for further approval
  - After the person in charge of the user department has accepted the plan recommended by the analyst, he will enter the next important stage of the overall design process, i.e., structural design

Functional decomposition

- In order to finally realize the target system, it is necessary to design and provide all the programs and files (or database) that make up the system
  - The design of programs (especially complex large-scale programs) is usually completed in two stages
    - Structural design
      - Structural design is the task of the overall design stage
      - The structure design determines which modules the program consists of and the relationship between these modules
    - Process design
      - Process design is the task of the detailed design stage
      - Process design determines the processing process of each module
- In order to determine the software structure, the complex functions are further decomposed according to the conditions of the realization of the software.
  - The analyst carefully analyzes each process in the data flow diagram in combination with the algorithm description. If the function of a process is too complex, its function must be appropriately decomposed into a series of relatively simple functions
  - Generally speaking, after decomposition, each function should be obvious and understandable to most programmers.
- Functional decomposition leads to further refinement of the data flow diagram. At the same time, an IPO diagram or other appropriate tools should be used to briefly describe the algorithm for each processing after refinement.

Design software structure

- Usually a module in the program completes an appropriate sub-function
- Constraints of software structure
  - Organize the modules into a good hierarchical system, the top-level module calls its lower-level modules to achieve the complete function of the program
  - Each lower-level module then calls a lower-level module to complete a sub-function of the program, and the lowest-level module completes the most specific function
  - The software structure (that is, a hierarchical system composed of modules) can be described by a hierarchical diagram or a structure diagram
- If the data flow diagram has been refined to an appropriate level, the software structure can be directly mapped from the data flow diagram

Design database

- For those application systems that need to use the database, software engineers should further design the database on the basis of the system data requirements determined in the requirements analysis stage

Develop test plan

- Considering testing issues in the early stages of software development can encourage software designers to pay attention to improving the testability of the software during design

Write document

- Formal documents should be used to record the results of the overall design. Related documents usually include:
  - System description, the main content includes
    - The system composition scheme described by the system flow chart, the list of physical elements that make up the system, and the cost/benefit analysis;
    - A general description of the best solution, refined data flow diagram, software structure described by hierarchical diagram or structure diagram, algorithm of each module briefly described by IPO diagram or other tools (for example, PDL language), interface between modules Relationships, and cross-reference relationships among requirements, functions, and modules, etc.
  - User manual
    - According to the results of the overall design stage, modify and correct the preliminary user manual generated in the requirements analysis stage
  - Test Plan
    - Including test strategy, test plan, expected test results, test schedule, etc.
  - Detailed implementation plan
  - Database design results

Examine and review

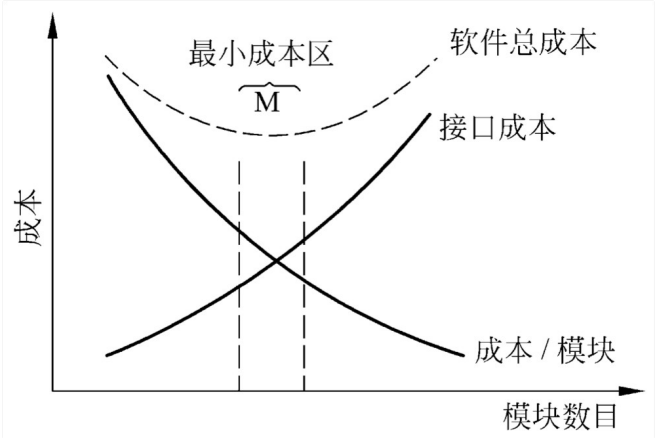
- Finally, a strict technical review should be carried out on the results of the overall design
- After the technical review is passed, the person in charge of the user department will conduct a review from a management perspective

Design principle

- Modularity
  - Module
    - Is a sequence of adjacent program elements (for example, data descriptions, executable statements) defined by boundary elements, and has an overall identifier to represent it
    - According to the definition of the module, procedures, functions, subroutines and macros can all be used as modules
    - Objects in object-oriented methodology are modules, and methods (or services) in objects are also modules
    - Module is the basic building block of the program
  - Modular
    - It is to divide the program into independently named and independently accessible modules, each module completes a sub-function, and these modules are integrated to form a whole, which can complete the specified functions to meet the needs of users
  - The theoretical basis of modularity
    - Let function C(x) define the complexity of problem x, and function E(x) determine the amount of work (time) required to solve problem x. For two problems P1 and P2, if C(P1)>C(P2), obviously E(P1)>E(P2)
    - According to human experience in solving general problems, another interesting rule is C(P1+P2)>C(P1)+C(P2)
    - That is, if a problem is composed of two problems P1 and P2, then its complexity is greater than the sum of the complexity when considering each problem separately.
    - In summary, the following inequality E(P1+P2)>E(P1)+E(P2) is obtained

Design principle

- Modularity (cont.)
  - The theoretical basis of modularity
    - If the software is divided indefinitely, the amount of work required to develop the software is so small that it can be ignored? ?
    - When the number of modules increases, the scale of each module will decrease, and the cost (workload) required to develop a single module will indeed decrease; however, as the number of modules increases, the workload required to design interfaces between modules will also increase.
    - Each program correspondingly has an optimal number of modules M, which minimizes the development cost of the system
  - Although it is not yet possible to accurately determine the value of M, the total cost curve is indeed a useful guide when considering modularity.



Design principle

- Modularity (cont.)
  - The advantages of modularity
    - The modular principle can make the software structure clear, not only easy to design, but also easy to read and understand
    - Program errors are usually limited to related modules and the interfaces between them, so modularization makes the software easy to test and debug, which helps to improve the reliability of the software
      - Because changes often involve only a few modules, modularity can improve the modifiability of software
    - Modularization is also helpful to the organization and management of software development projects. A complex large program can be divided into many programmers to write different modules, and technically skilled programmers can be further allocated to write difficult modules.

Design principle

- Abstraction
  - Abstraction is the most powerful thinking tool used by humans in the process of understanding complex phenomena
    - There are always some similarities between certain things, states or processes in the real world (commonality)
    - Collecting and generalizing these similar aspects, and temporarily ignoring the differences between them, is abstract
    - Abstraction is to extract the essential characteristics of things without considering their details for the time being
  - Due to the limitation of human thinking ability, if there are too many factors each time, it is impossible to make precise thinking
    - The only effective way to deal with a complex system is to construct and analyze the system in a hierarchical manner
    - A complex dynamic system can first be constructed and understood with some advanced abstract concepts
    - These high-level concepts can be constructed and understood with some lower-level concepts
    - And so on until the lowest level of specific elements

Design principle

- Abstraction [cont.]
  - When considering a modular solution to any problem, many levels of abstraction can be proposed
    - Use the language of the problem environment at the highest level of abstraction to describe the solution of the problem in a general way
    - Use a more procedural approach at a lower level of abstraction, combining problem-oriented terminology with realization-oriented terminology to describe the solution of the problem
    - Describe the solution of the problem in a directly achievable way at the lowest level of abstraction
  - Every step of the software engineering process is a refinement of the abstract level of the software solution
    - In the feasibility study stage, the software is an integral part of the system
    - During requirements analysis, software solutions are described in a familiar way within the problem environment
    - When transitioning from the overall design to the detailed design, the degree of abstraction also decreases.
    - When the source program is written, it reaches the lowest level of abstraction

Design principle

- Abstraction [cont.]
  - The concept of gradual refinement and modularity is closely related to abstraction
    - With the progress of software development engineering, the modules in each layer of the software structure represent a refinement of the software abstraction level
      - The module at the top of the software structure controls the main functions of the system and affects the overall situation
      - The module at the bottom of the software structure completes a specific processing of data, and allocates control from top to bottom from abstract to concrete, which simplifies the design and implementation of the software, improves the comprehensibility and testability of the software, and Make the software easier to maintain

Design principle

- Gradually refine
  - Gradually refinement can be seen as a technology that prioritizes the various problems that must be solved within a period of time
  - The step-by-step refinement method ensures that every problem will be solved, and every problem will be solved in due course
- Gradually refinement is the basic method used by humans to solve complex problems, and it is also the basis of many software engineering techniques (for example, specification techniques, design and implementation techniques)
  - The gradual refinement can be defined as: "Delay the consideration of the details of the problem as much as possible in order to concentrate on solving the main problem."
- The reason why gradual refinement is so important is that the human cognitive process obeys Miller's law
  - One can only focus on (7±2) knowledge blocks at any time

Design principle

- Gradually refine [cont.]
  - About Miller's Law
    - One can only focus on (7±2) knowledge blocks at any time
      - However, in the process of developing software, the number of knowledge blocks that software engineers need to consider in a period of time is far more than 7
    - For example, a program usually uses more than 7 data, and a user often has more than 7 requirements.
  - The role of the step-by-step refinement method
    - Can help software engineers focus on the most relevant aspects of the current development stage
    - Can help to ignore the details that are necessary for the overall solution, but do not need to be considered at present, and these details will be considered later

    - Miller's law is the basic limitation of human intelligence. We cannot overcome our natural nature. We can only accept this fact, acknowledge our limitations, and do our best to work under this premise.

Design principle

- Localization and information hiding
  - When applying the principle of modularity, a question that naturally arises is: "In order to get the best set of modules, how should the software be decomposed?"
  - According to the principle of information hiding
    - The information (process and data) contained in a module should be inaccessible to modules that do not need this information through design and determination of the module
  - The concept of localization and the concept of information hiding are closely related
    - The so-called localization refers to physically placing some closely related software elements close to each other
    - Using local data elements in a module is an example of localization
    - Obviously, localization helps to realize information hiding
  - "Hidden" means that effective modularity can be achieved by defining a set of independent modules, and these independent modules only exchange information that must be exchanged in order to complete system functions.
    - If the software needs to be modified during testing and later during software maintenance, then using the principle of information hiding as a standard for modular system design will bring great benefits
    - Because most of the data and processes are hidden from other parts of the software (that is, "invisible"), errors introduced due to negligence during modification are rarely likely to spread to other parts of the software

Independent Module

- The concept of module independence is a direct result of the concepts of modularity, abstraction, information hiding and localization
- Develop a module that has independent functions and does not have too much interaction with other modules, and you can achieve module independence
  - That is, I hope to design the software structure in such a way that each module completes a relatively independent specific sub-function, and the relationship with other modules is very simple
- Importance
  - Software with effective modularity (that is, with independent modules) is easier to develop; this is because the functions can be divided and the interface can be simplified. This advantage is especially important when many people work together to develop the same software.
  - Independent modules are easier to test and maintain
    - This is because relatively speaking, the workload required to modify the design and program is relatively small, the error propagation range is small, and modules can be "plugged in" when extended functions are needed. In short, module independence is the key to good design, and design is the key to determining software quality
- Measure
  - The degree of independence of modules can be measured by two qualitative standards: cohesion and coupling
- Coupling measures how closely different modules depend on each other (connection)
- Cohesion measures how closely the various elements within a module are combined with each other

Coupling

- Coupling is a measure of the degree of interconnection between different modules in a software structure
- The strength of the coupling depends on the complexity of the interface between modules, the point of entry or access to a module, and the data passing through the interface
- In software design, a system that is as loosely coupled as possible should be pursued
  - In such a system, any module can be researched, tested or maintained, without the need to know a lot about other modules of the system
  - Due to the simple connection between modules, the possibility of an error occurring in one place being propagated to the entire system is very small
  - Therefore, the degree of coupling between modules strongly affects the understandability, testability, reliability and maintainability of the system
- If each of the two modules can work independently without the presence of the other module, then they are completely independent of each other, which means that there is no connection between the modules and the degree of coupling is minimal
  - However, it is impossible to have no connection between all modules in a software system.

Coupling

- Type
  - Data coupling is low coupling
    - If two modules exchange information with each other through parameters, and the information exchanged is only data, then this coupling is called data coupling
    - At least this coupling must exist in the system, because only when the output data of some modules is used as the input data of other modules, the system can perform valuable functions. Generally speaking, a system can only contain data coupling
  - Control coupling is moderate coupling
    - If there is control information in the transmitted information (although sometimes this control information appears in the form of data), this coupling is called control coupling
    - It increases the complexity of the system, and the control coupling is often superfluous. After the module is properly decomposed, it can usually be replaced by data coupling.

Coupling

- Type
  - Feature coupling
    - If the called module needs to use all the elements in the data structure passed in as a parameter, then it is completely correct to pass the entire data structure as a parameter.
    - However, when the module called by passing the entire data structure as a parameter only needs to use some of the data elements, feature coupling occurs
    - In this case, the called module can use more data than it really needs, which will lead to loss of control over data access, thus providing opportunities for computer crimes
  - Public environment coupling
    - When two or more modules interact through a common data environment, the coupling between them is called common environment coupling
    - The public environment can be global variables, shared communication areas, public coverage areas of memory, files on any storage media, physical devices, etc.

Coupling

- Type
  - The complexity of public environment coupling varies with the number of coupled modules, and the complexity increases significantly when the number of coupled modules increases
    - If only two modules have a common environment, then this coupling has two possibilities:
    - One module sends data to the public environment, and the other module fetches data from the public environment. This is a form of data coupling, which is relatively loose coupling.
    - Both modules send data to the public environment and fetch data from it. This coupling is relatively close, between data coupling and control coupling.
    - If the two modules share a lot of data, it may be inconvenient to pass them through parameters. At this time, you can use the common environment to couple
  - The highest degree of coupling is content coupling
    - If one of the following situations occurs, content coupling occurs between the two modules
      - One module accesses the internal data of another module
      - One module goes to the inside of another module without passing through the normal entrance
      - There is a part of the program code overlap between the two modules (only possible in the assembler)
      - A module has multiple entries (this means that a module has several functions)
    - The use of content coupling should be firmly avoided. In fact, many high-level programming languages have been designed to not allow any form of content coupling in the program

Coupling

- Type
  - Design principles for coupling
  - Use data coupling as much as possible, use less control coupling and feature coupling, limit the scope of public environment coupling, and don't use content coupling at all

Cohesion

- Cohesion
  - Marks how closely the various elements in a module are combined with each other. It is a natural extension of the concept of information hiding and localization
  - Simply speaking, ideally cohesive modules do only one thing
- Design should strive to achieve high cohesion
  - Usually moderate cohesion can be used, and the effect is similar to that of high cohesion.
    - However, the effect of low cohesion is very poor, try to avoid using
- Cohesion and coupling
  - Cohesion and coupling are closely related. High cohesion within modules often means loose coupling between modules
  - Cohesion and coupling are both powerful tools for modular design, but practice shows that cohesion is more important, and more attention should be paid to improving the cohesion of modules

Types of low cohesion

- Accidental cohesion
  - If a module completes a set of tasks, even if these tasks are related to each other, the relationship is very loose, which is called accidental cohesion.
    - Sometimes after writing a program, a group of statements are found in two or more places, so these statements are used as a module to save memory, so that occasionally cohesive modules appear
- Logical cohesion
  - If the tasks completed by a module logically belong to the same or similar category, it is called logical cohesion
- Time to gather
  - If the tasks contained in a module must be executed within the same period of time, it is called time-to-gather

Types of moderate cohesion

- Process cohesion
  - If the processing elements within a module are related and must be executed in a specific order, it is called process cohesion
    - When using the program flow chart as a tool to design software, the division of modules is often determined by studying the flow chart, so that the modules are often cohesive in the process.
- Communication cohesion
  - If all elements in the module use the same input data and/or produce the same output data, it is called communication cohesion

Types of high cohesion

- Sequential cohesion
  - If the processing elements in a module are closely related to the same function, and these processing must be executed sequentially (usually the output data of one processing element is used as the input data of the next processing element), it is called sequential cohesion
    - When the modules are divided according to the data flow graph, a sequential cohesive module is usually obtained, and the connection between such modules is often relatively simple
- Functional cohesion
  - If all the processing elements in the module belong to a whole and complete a single function, it is called functional cohesion
  - Functional cohesion is the highest degree of cohesion.

Evaluation of Cohesion

- Function cohesion 10 points
- Sequence cohesion 9 points
- Communication cohesion 7 points
- Process cohesion 5 points
- Temporal cohesion 3 points
- Logical cohesion 1 point
- Accidental Cohesion 0 points

- It is not necessary to precisely determine the level of cohesion. The important thing is to strive to achieve high cohesion during design, and to be able to identify modules with low cohesion, and to be able to modify the design to improve the cohesion of the modules and reduce the degree of coupling between modules, so as to obtain higher module independence

Heuristic rule

- Although heuristic rules are not universally applicable, they can still give software engineers beneficial enlightenment on many occasions, and often help them find ways to improve software design and improve software quality.
  - Improve software structure and increase module independence
  - Module size should be moderate
  - Depth, width, fan-out and fan-in should all be appropriate
  - The scope of the module should be within the control domain
  - Strive to reduce the complexity of the module interface
  - Design a single entry single exit module
  - Module function should be predictable

Improve software structure and increase module independence

- After designing the initial structure of the software, the structure should be reviewed and analyzed, and the modules should be decomposed or merged to reduce coupling and improve cohesion.
  - For example, a sub-function common to multiple modules can be independently formed into a module, which can be called by these modules
  - Sometimes it is possible to decompose or merge modules to reduce the transmission of control information and the reference to the whole process data, and reduce the complexity of the interface

Module size should be moderate

- Experience has shown that the scale of a module should not be too large, it is best to write it in one page
  - Usually no more than 60 lines of statements
  - From the perspective of psychology, it is known that when the number of sentences contained in a module exceeds 30, the comprehensibility of the module decreases rapidly
- Oversized modules are often due to insufficient decomposition, but further decomposition must conform to the problem structure. Generally speaking, module independence should not be reduced after decomposition
- Too small module overhead is greater than effective operation, and too many modules will complicate the system interface
  - Therefore, a module that is too small sometimes does not deserve to exist alone, especially when only one module calls it, it can usually be merged into the upper-level module without having to exist separately

Depth, width, fan-out and fan-in should all be appropriate

- Depth represents the number of layers controlled in the software structure, and it can often roughly indicate the size and complexity of a system
  - There should be a rough correspondence between depth and program length, and the correspondence is changing within a certain range
  - If there are too many layers, you should consider whether there are many management modules that are too simple and whether they can be combined appropriately
- The width is the maximum value of the total number of modules on the same level in the software structure
  - Generally speaking, the larger the width, the more complex the system
  - The biggest influence on the width is the fan-out of the module

The scope of the module should be within the control domain

- The scope of the module
  - The collection of all modules affected by a decision in this module
- Control domain of the module
  - The module itself and the collection of all modules directly or indirectly subordinate to it
- In a well-designed system
  - All modules affected by the judgment should be subordinate to the module that made the judgment
  - It's better to be limited to the module that made the decision and its immediate subordinate modules

Strive to reduce the complexity of the module interface

- The complexity of the module interface is a major cause of software errors
  - The module interface should be carefully designed to make the information transmission simple and consistent with the function of the module
- The interface is complex or inconsistent (that is, it seems that there is no connection between the transmitted data), which is a sign of tight coupling or low cohesion, and the independence of this module should be re-analyzed

Design a single entry single exit module

- Remind software engineers not to make content coupling between modules
- When entering the module from the top and exiting from the bottom, the software is easier to understand and therefore easier to maintain

Module function should be predictable

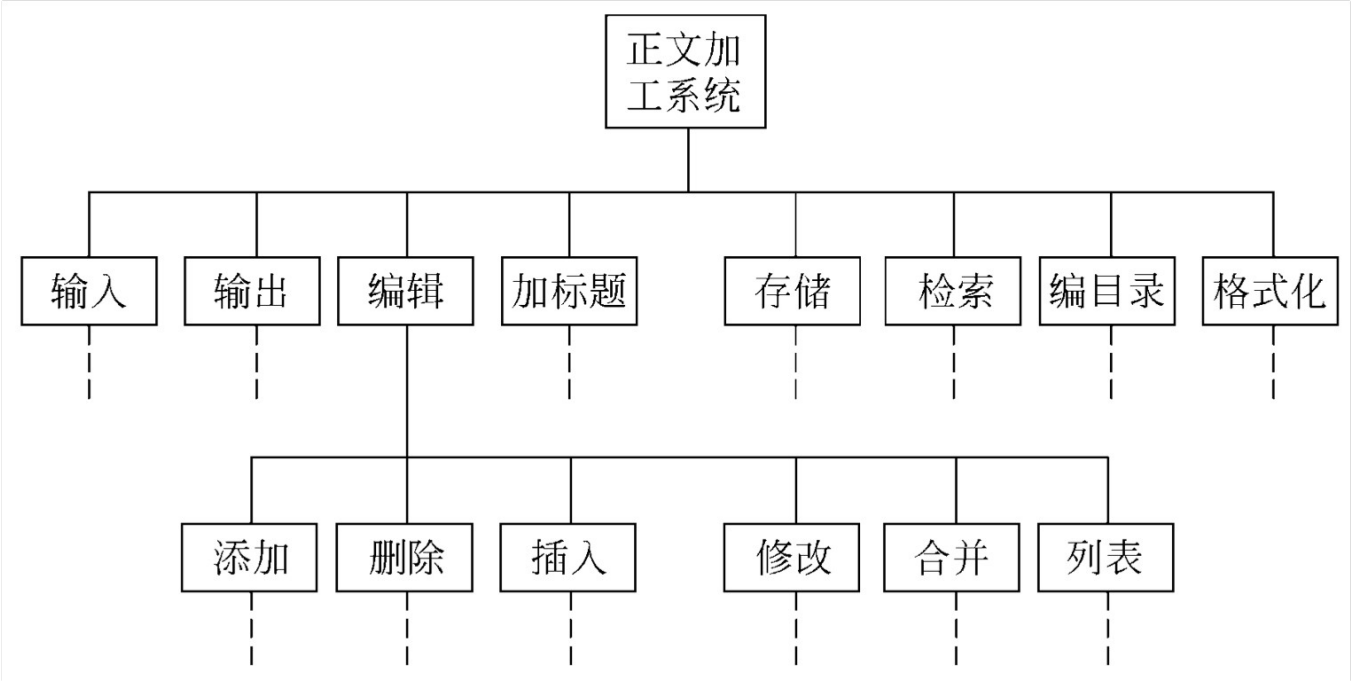
- To prevent the module function from being too limited
  - If a module can be regarded as a black box, that is, as long as the input data is the same, the same output is produced, and the function of the module is predictable
  - The function of a module with internal "memory" may be unpredictable, because its output may depend on the state of the internal memory (such as a certain flag)
    - Since the internal memory is invisible to the upper-level module, such a module is not easy to understand and difficult to test and maintain
  - If a module only completes a single sub-function, it exhibits high cohesion; however, if a module arbitrarily limits the size of the local data structure, excessively restricts the choices that can be made in the control flow or the mode of the external interface, then this The function of the module is too limited, and the scope of use is too narrow

Graphical tool for depicting software structure

- Hierarchy diagram and HIPO diagram
- Structure chart

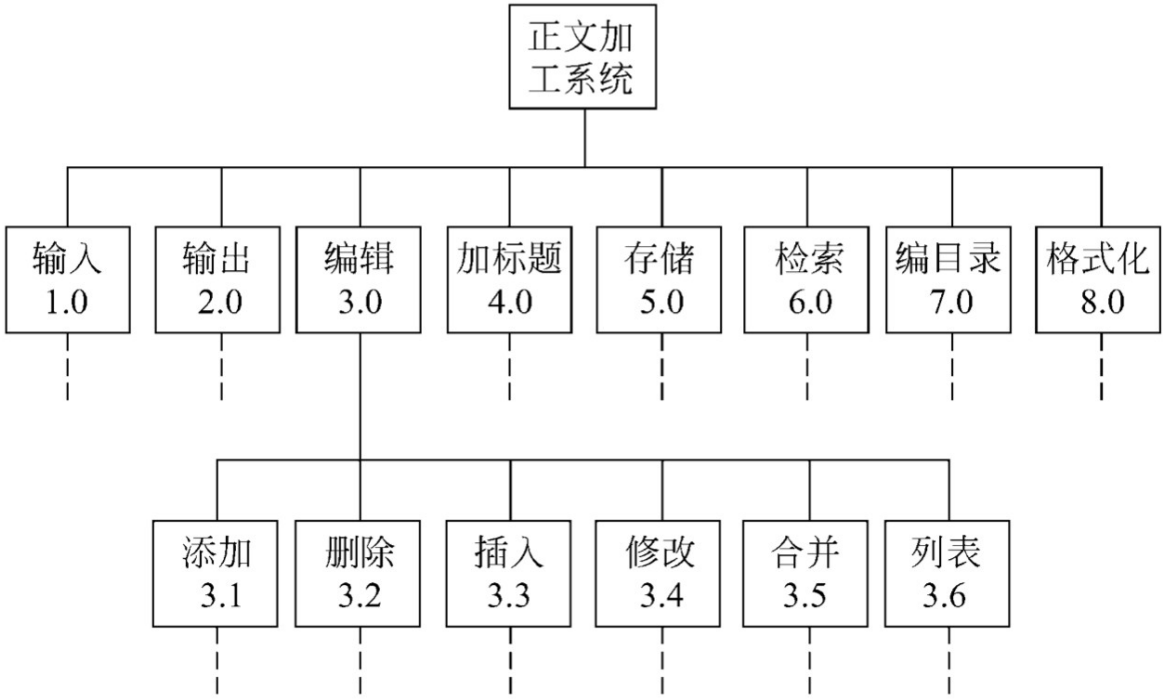
Hierarchy diagram and HIPO diagram

- Hierarchical diagrams are used to depict the hierarchical structure of software
- A rectangular box in the hierarchical diagram represents a module, and the connection between the boxes represents the calling relationship instead of the composition relationship like the hierarchical block diagram.



Hierarchy diagram and HIPO diagram

- Hierarchical charts are very suitable for use in the process of top-down software design
- HIPO diagram is the abbreviation of "hierarchical diagram plus input/processing/output diagram" invented by IBM in the United States
- In order to make the HIPO chart traceable, except for the topmost box in the H chart (hierarchical chart), each box is numbered

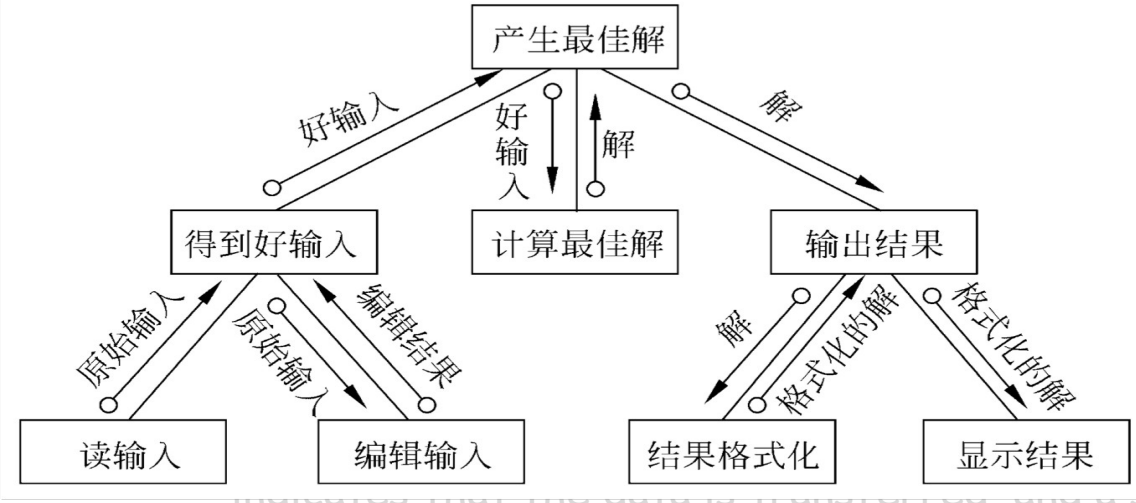


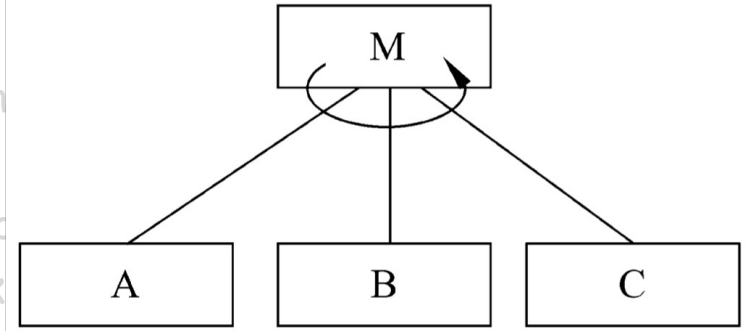
Structure chart

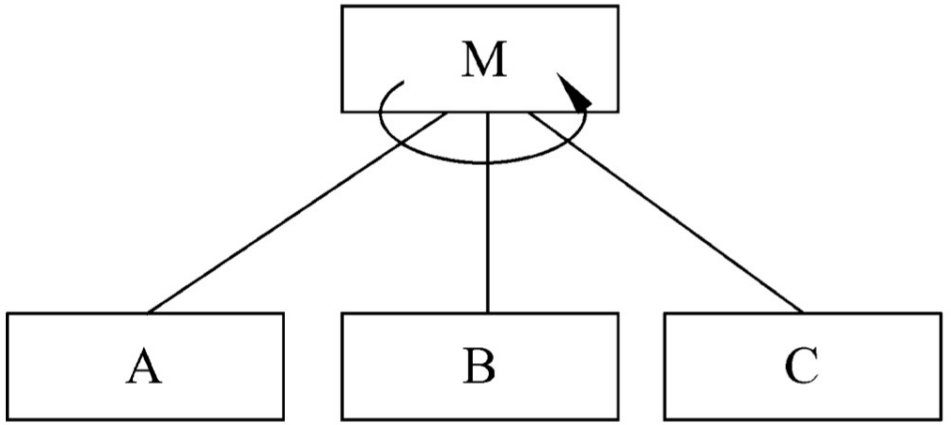
- The structure diagram proposed by Yourdon is a powerful tool for software structure design
- Structure diagram and hierarchical diagram are similar
  - A box in the figure represents a module, and the name or main function of the module is indicated in the box
  - The arrow (or straight line) between the boxes indicates the calling relationship of the module
  - Use annotated arrows to indicate the information passed back and forth during the module call
    - If you want to further indicate whether the information transferred is data or control information, you can use the shape of the tail of the annotation arrow to distinguish: a hollow circle at the tail indicates that the data is transferred, and a solid circle indicates that the transfer is control information.
- Because by convention, the module represented by the upper box in the figure always calls the lower module, even if the arrow is not used, there will be no ambiguity.

Structure chart

- The structure diagram proposed by Yourdon is a powerful tool for software structure design







- Because by convention, the module represented by the upper
  even if the arrow is not used, there will be no ambiguity.

Data flow-oriented design method

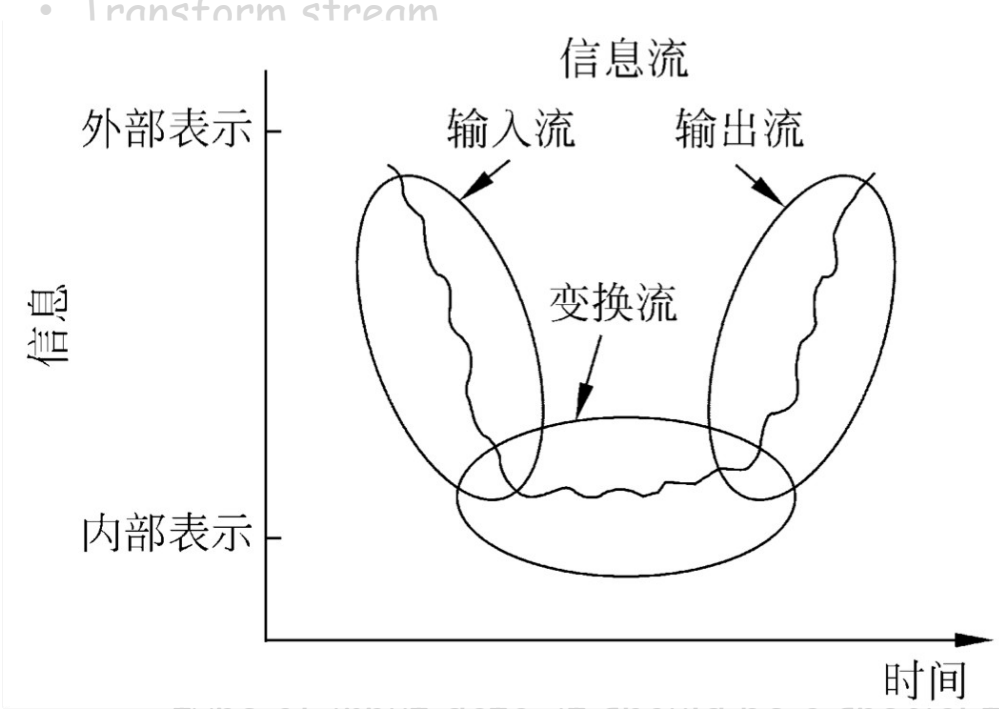
- Target
  - Give a systematic way to design software structure
- In the requirements analysis stage of software engineering, information flow is a key consideration. Data flow diagrams are usually used to describe the processing and flow of information in the system.
- The data flow-oriented design method defines a number of different "mappings", which can be used to transform the data flow graph into a software structure
  - Because any software system can be represented by a data flow graph, the data flow-oriented design method can theoretically design the structure of any software. The so-called structured design method (referred to as SD method), which is a design method based on data flow

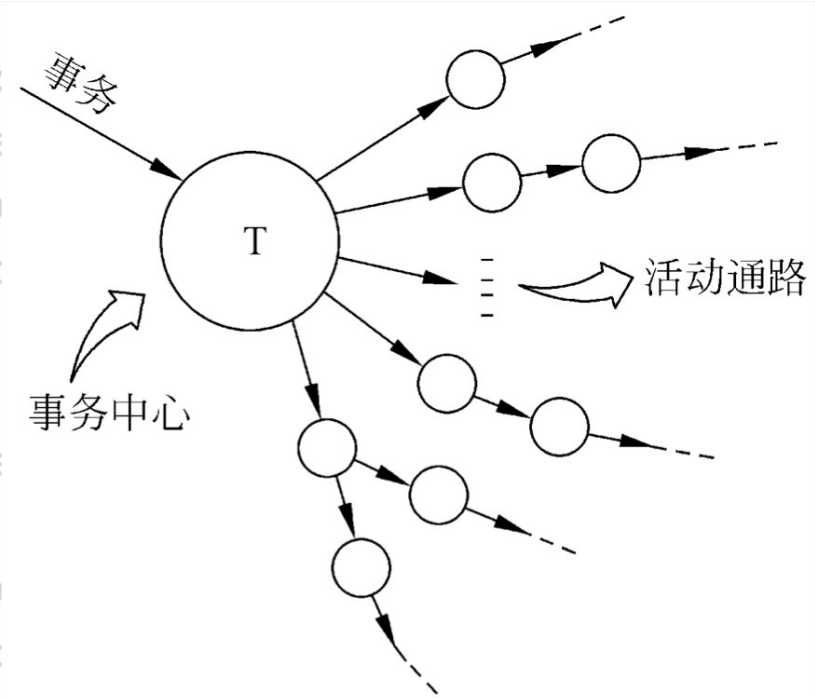
Data flow-oriented design method

- Basic concept
  - Data flow-oriented design method maps information flow into software structure
  - The type of information flow determines the mapping method
- Type of information flow
  - Transform stream
    - Information enters the system along the input path, and at the same time, it is transformed from the external form to the internal form. The information entering the system passes through the transformation center, and after processing, it is transformed into the external form along the output path and leaves the software system. When the data flow graph has these characteristics, this information flow is called a transformation flow
  - Transaction flow
    - The basic system model means transformation flow, therefore, in principle, all information flows can be attributed to transformation flows
    - When the data flow is "transaction-centric", that is, the data reaches a processing T along the input path, and this processing selects one of several action sequences for execution according to the type of input data, it should be a special type of data Flow, called transaction flow

Data flow-oriented design method

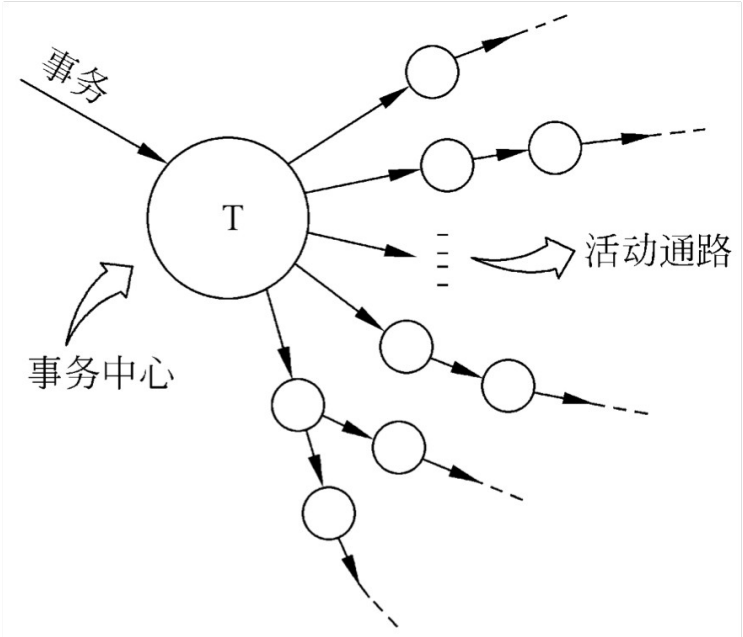
- Basic concept
  - Data flow-oriented design method maps information flow into software structure
  - The type of information flow determines the mapping method
- Type of information flow
  - Transform stream

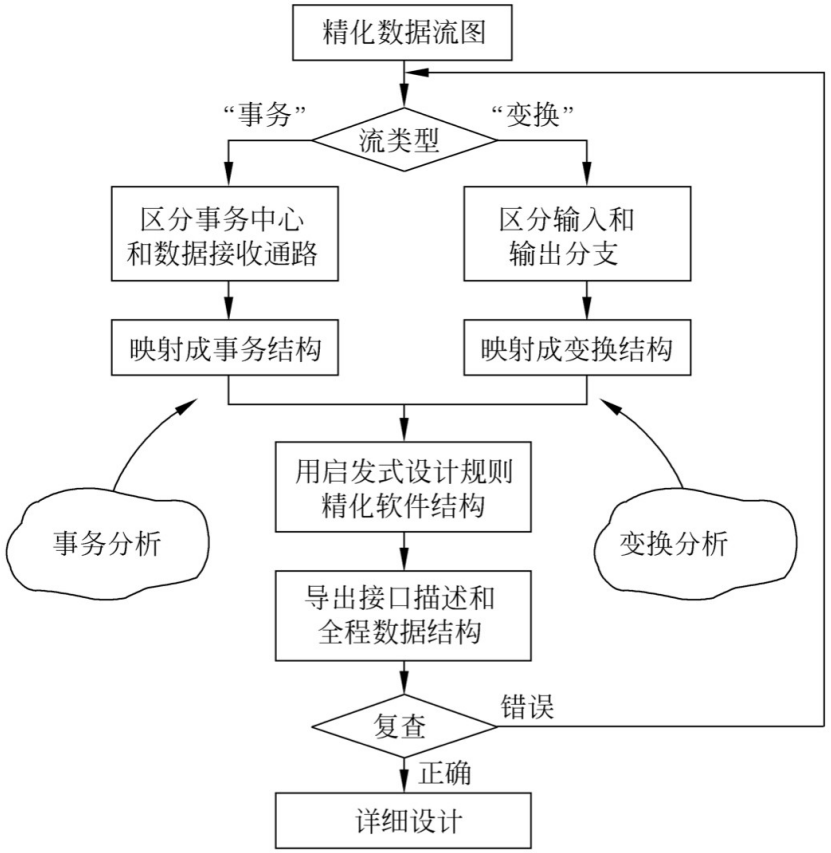




Data flow-oriented design method

- Processing T is called the transaction center, it completes the following tasks
  - Receive input data (input data is also called transaction)
  - Analyze each transaction to determine its type
  - Select an activity path according to the type of transaction





Data flow-oriented design method

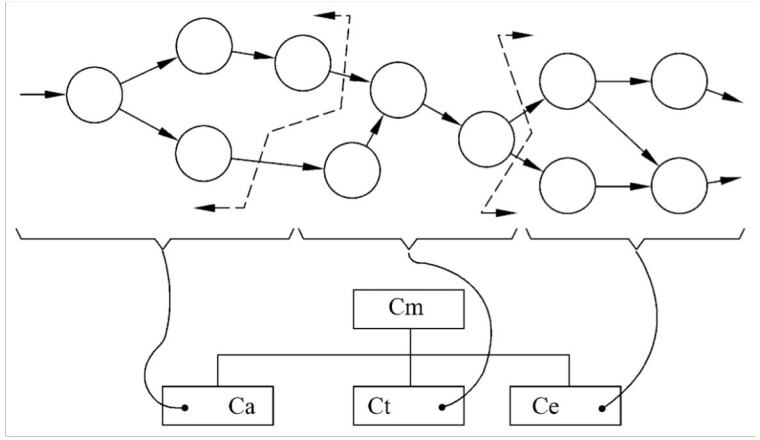
- Transformation analysis
  - Transformation analysis is a general term for a series of design steps. After these steps, the data flow diagram with the characteristics of the transformation flow is mapped into the software structure according to a predetermined pattern.
- Instance
  - Design automotive digital dashboard to complete the following functions:
    - (1) Realize the sensor and microprocessor interface through analog-to-digital conversion;
    - (2) Display data on the LED panel;
    - (3) Indicate miles per hour (mph), mileage, miles per gallon (mpg), etc.;
    - (4) Indicate acceleration or deceleration;
    - (5) Speeding warning: If the vehicle speed exceeds 55 mph, the speeding warning bell will sound.

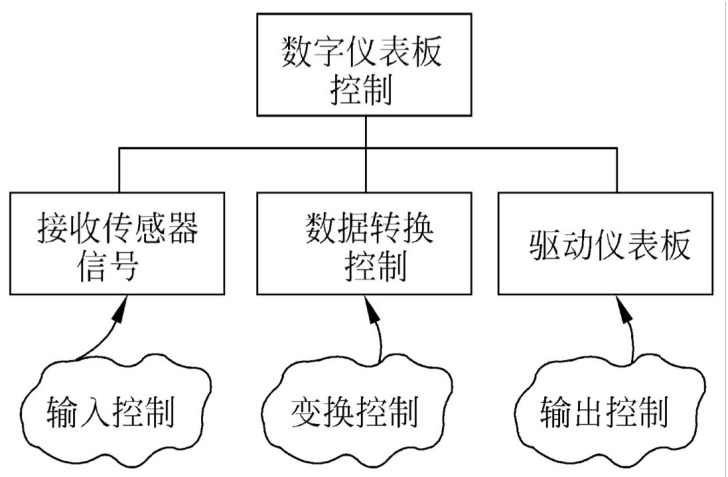
Data flow-oriented design method

- Design steps
  - Step 1: Review the basic system model
    - The purpose of the review is to ensure that the input data and output data of the system are in line with reality.
  - Step 2: Review and refine the data flow diagram.
    - The data flow diagram obtained during the requirements analysis stage should be carefully reviewed and refined if necessary. It is not only necessary to ensure that the data flow diagram gives the correct logical model of the target system, but also that each process in the data flow diagram represents a relatively independent sub-function of a moderate scale.
  - Step 3: Determine whether the data flow graph has transformation characteristics or transaction characteristics.
    - All information flows in a system can be considered as transformation flows. However, when encountering information flows with obvious transaction characteristics, it is recommended to use transaction analysis methods for design.
    - In this step, the designer should determine the global characteristics of the data flow based on the dominant attributes in the data flow diagram. In addition, local areas with different characteristics from the global characteristics should be isolated, and the software structure derived from the global characteristics can be refined according to the characteristics of these sub-data streams in the future.

Data flow-oriented design method

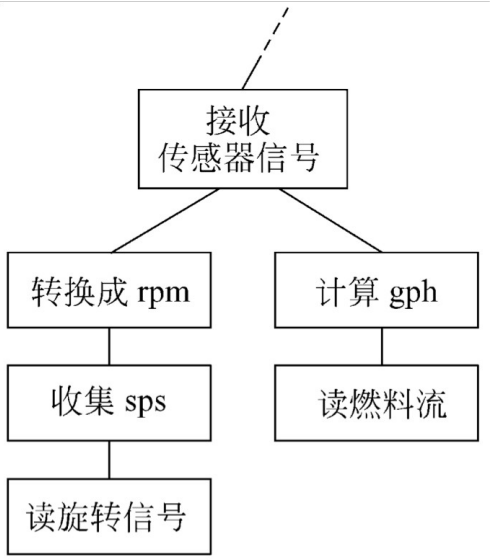
- Design steps
  - Step 4: Determine the boundary between the input stream and the output stream, so as to isolate the transformation center.
    - The boundary of the input stream and the output stream is related to their interpretation, that is, different designers may choose slightly different points in the stream as the location of the boundary. Of course, you should be careful when determining the boundary, but moving the boundary along the data flow path by a processing frame distance usually has only a small impact on the final software structure
  - Step 5: base completes "first-level decomposition"
    - The software structure represents the top-down distribution of control. The so-called decomposition is the process of distribution control.
    - In the case of the transformation flow, the data flow graph is mapped into a special software structure, which controls the information processing process such as input, transformation, and output.

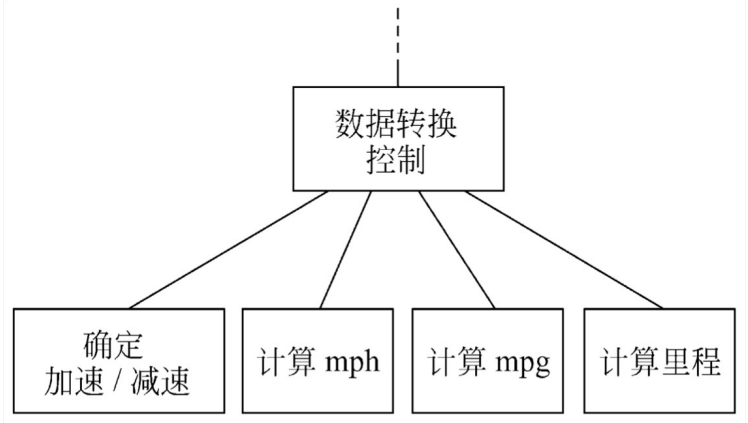


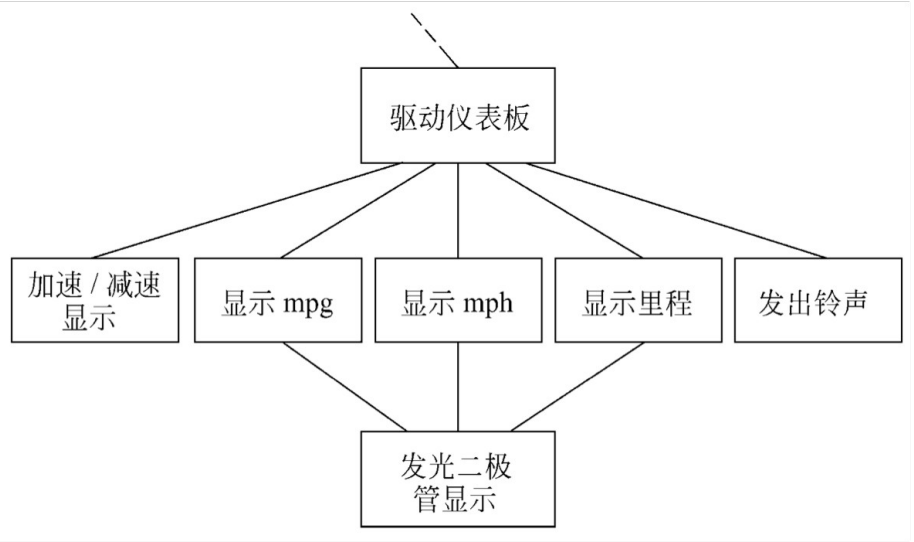


Data flow-oriented design method

- Design steps
  - Step 6: Complete the "Second-level Decomposition"
    - The so-called second-level decomposition is to map each process in the data flow graph to an appropriate module in the software structure
    - The way to complete the second level of decomposition is,
      - Starting from the boundary of the transformation center and moving outward along the input path, each process in the input path is mapped into a low-level module under the control of Ca in the software structure;
      - Then move outward along the output path, and map each process in the output path to a low-level module directly or indirectly controlled by the module Ce;
      - Finally, each process in the transformation center is mapped into a module controlled by Ct.





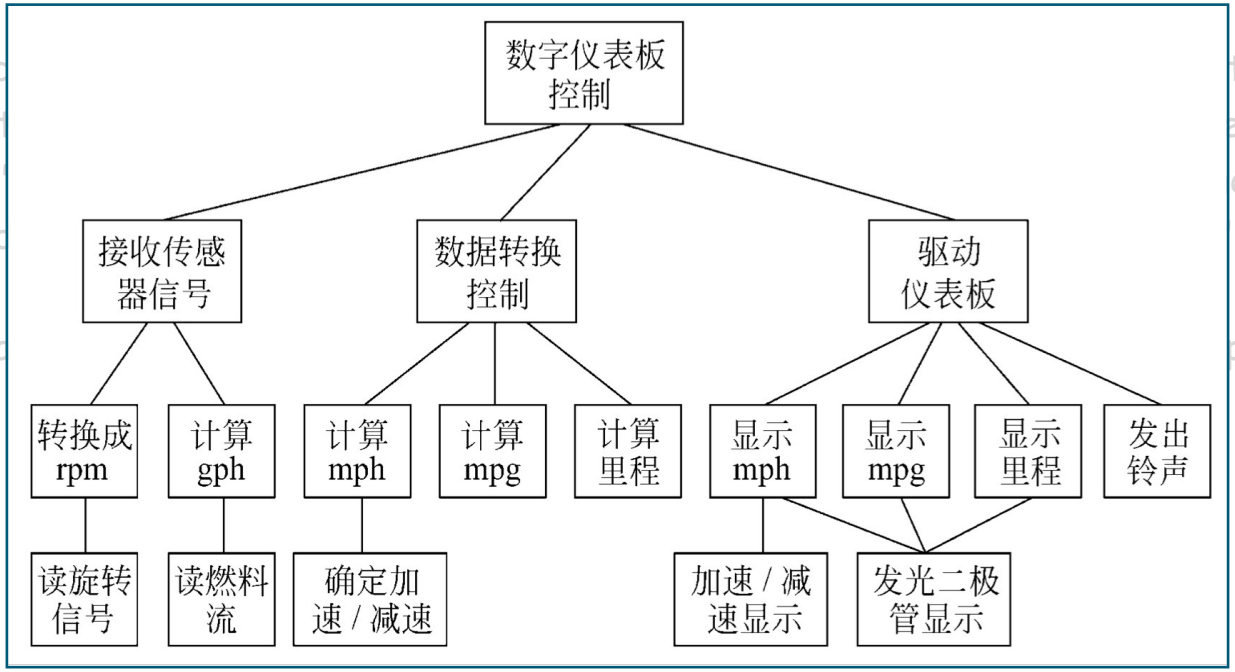


Data flow-oriented design method

- Design steps
  - Step 7: Use design metrics and heuristic rules to further refine the software structure obtained from the first segmentation
    - The software structure obtained from the first division can always be refined according to the principle of module independence, and the software structure obtained from the previous design steps can be modified.
      - The modules "convert to rpm" and "collect sps" in the input structure can be combined
      - The module "determine acceleration/deceleration" can be placed under the module "calculate mph" to reduce coupling
      - The module "acceleration/deceleration display" can be placed below the module "display mph" accordingly

Data flow-oriented design method

- Design steps
  - Step 7: Use design metrics and heuristic rules to further refine the software structure obtained from the first segmentation
    - The software struc… to the principle of module independence, and t… e modified.
      - The modules "… ed
      - The module "c… calculate mph" to reduce coupling
      - The module "c… play mph" accordingly



Data flow-oriented design method

- The purpose of the above 7 design steps is to develop an overall representation of the software
  - After the software structure is determined, it can be reviewed as a whole, so that the software structure can be evaluated and refined
  - Modifications during this period require little additional work, but they can have a profound impact on the quality of the software, especially the maintainability of the software.

Transaction/Activity analysis

- When the data stream has obvious transaction characteristics, that is, when there is an obvious "launch center" (transaction/activity center), you can use the transaction analysis method to design the software structure

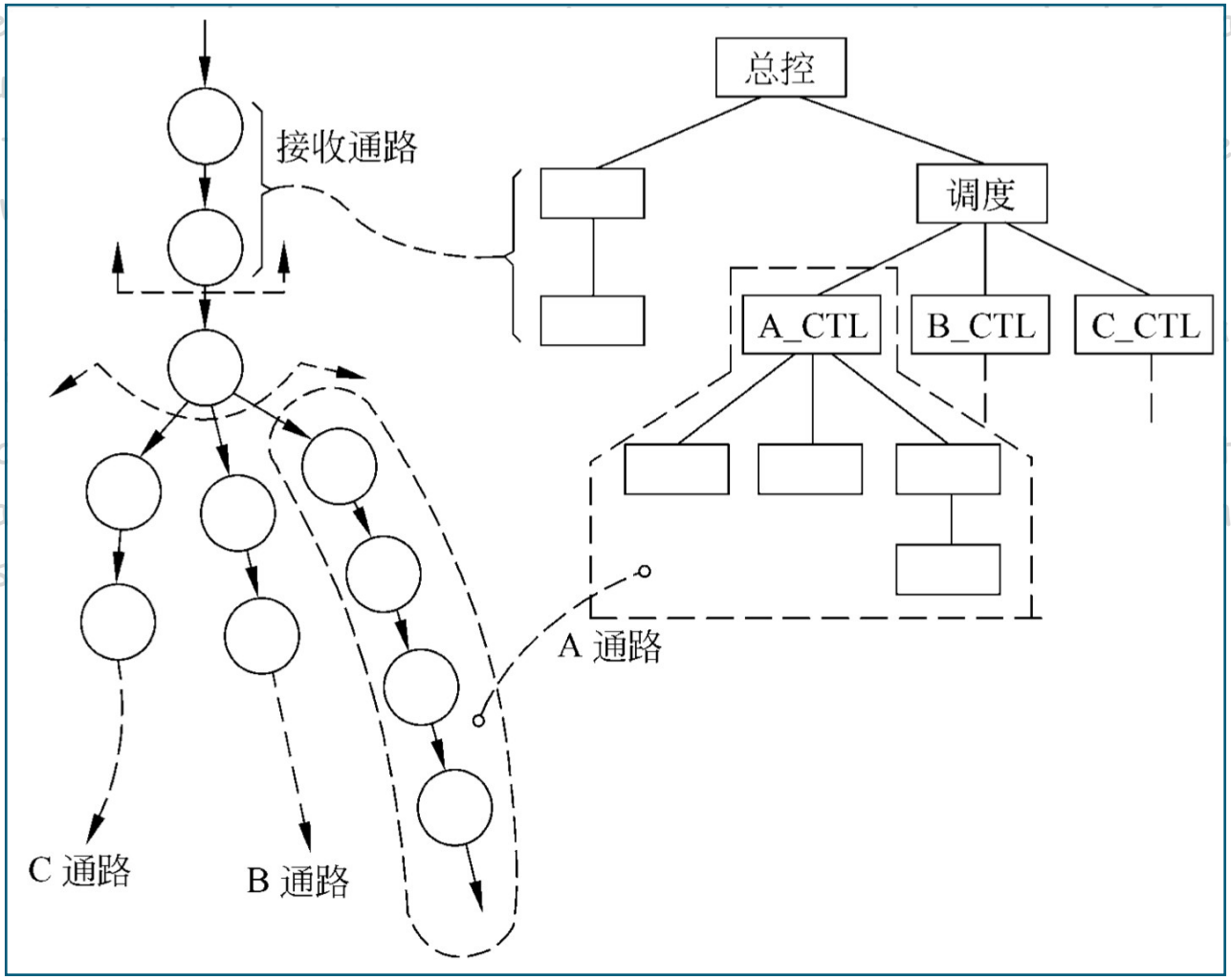
- The design steps of transaction analysis and transformation analysis are mostly the same or similar
    - The main difference lies in the mapping method from the data flow diagram to the software structure.

Transaction/Activity analysis

- The software structure mapped by the transaction flow includes a receiving branch and a sending branch
  - The method of receiving the branch structure is very similar to the method of transforming and analyzing the input structure.
    - That is, starting from the boundary of the transaction center, map the processing along the receiving flow path into modules
  - The structure of the sending branch contains a scheduling module, which controls all activity modules in the lower layer; then each activity flow path in the data flow graph is mapped to a structure corresponding to its flow characteristics
- For a large system, transformation analysis and transaction analysis are often applied to different parts of the same data flow diagram, and the resulting substructures form "components", which can be used to construct a complete software structure

Transaction/Activity analysis

- The software structure mapped by the transaction flow includes a receiving branch and a sending branch
  - The method of re… …ansforming and analyzing the inpu…
    - That is, start… …ssing along the receiving flo…
  - The structure of … …all activity modules in the lower layer; t… … structure corresponding to …
- For a large system, tra… …o different parts of the same data flow dia… … can be used to construct a complete s…



Transaction/Activity analysis

- Generally speaking, if the data flow does not have significant transaction characteristics, it is best to use transformation analysis

- Conversely, if there is an obvious transaction center, transaction analysis techniques should be used

- During use, if you follow the mapping rules of transformation analysis or transaction analysis mechanically, you may get some unnecessary control modules.
  - If they are really not very useful, then they can and should be combined
  - If the function of a control module is too complicated, it should be decomposed into two or more control modules, or an intermediate control module should be added

Design Optimization

- "The value of a "best design" that can't work is questionable
  - Should be committed to the development of software that can meet all functional and performance requirements, and is worthy of acceptance in accordance with design principles and heuristic design rules
- The software structure should be refined as much as possible in the early stages of the design
  - Can derive different software structures, then evaluate and compare them, and strive to get the "best" result
  - This possibility of optimization is one of the real advantages of separating software structure design from process design

Design Optimization

- It is reasonable to use the following method to optimize the software that plays a decisive role in time
  - Develop and refine the software structure without considering the time factor;
  - In the detailed design stage, select the most time-consuming modules and carefully design their processing procedures (algorithms) to improve efficiency;
  - Use high-level programming language to write programs;
  - Isolate those modules that take up a lot of processor resources in the software;
  - When necessary, redesign or rewrite the code of the above-mentioned resource-intensive modules with a machine-dependent language in order to improve efficiency.
  - The above optimization method adheres to a motto: "Make it work first, and then make it fast."