NATURE OF SOFTWARE:

Software is:

(1) instructions (computer programs) that when executed provide desired features, function, and performance;

(2) data structures that enable the programs to adequately manipulate information, and

(3) document that describes the operation and use of the programs.

Characteristics of software

- Software is developed or engineered, it is not manufactured in the classical sense.
- Software does not wear out. However it deteriorates due to change.
- Software is custom built rather than assembling existing components. - Although the industry is moving towards component based construction, most software continues to be custom built

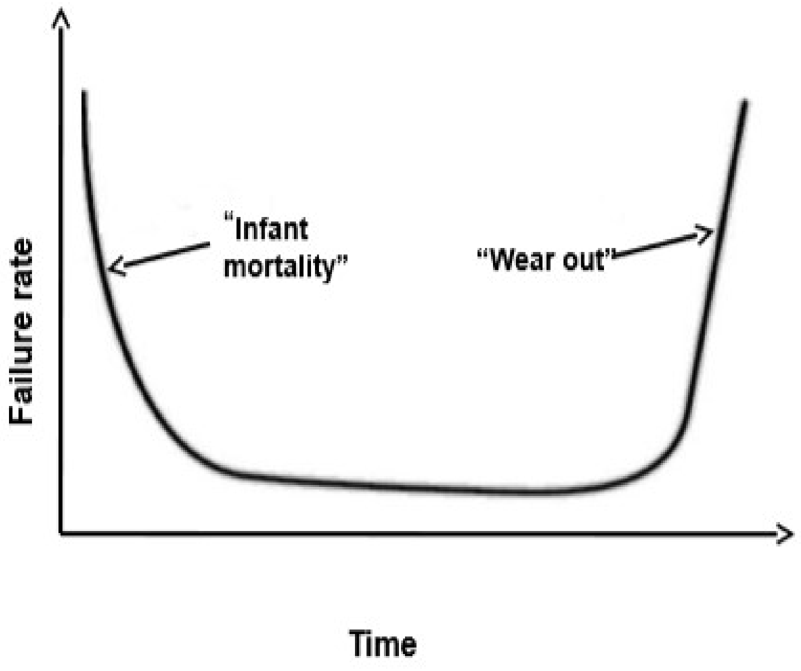


Fig: FAILURE CURVE FOR HARDWARE

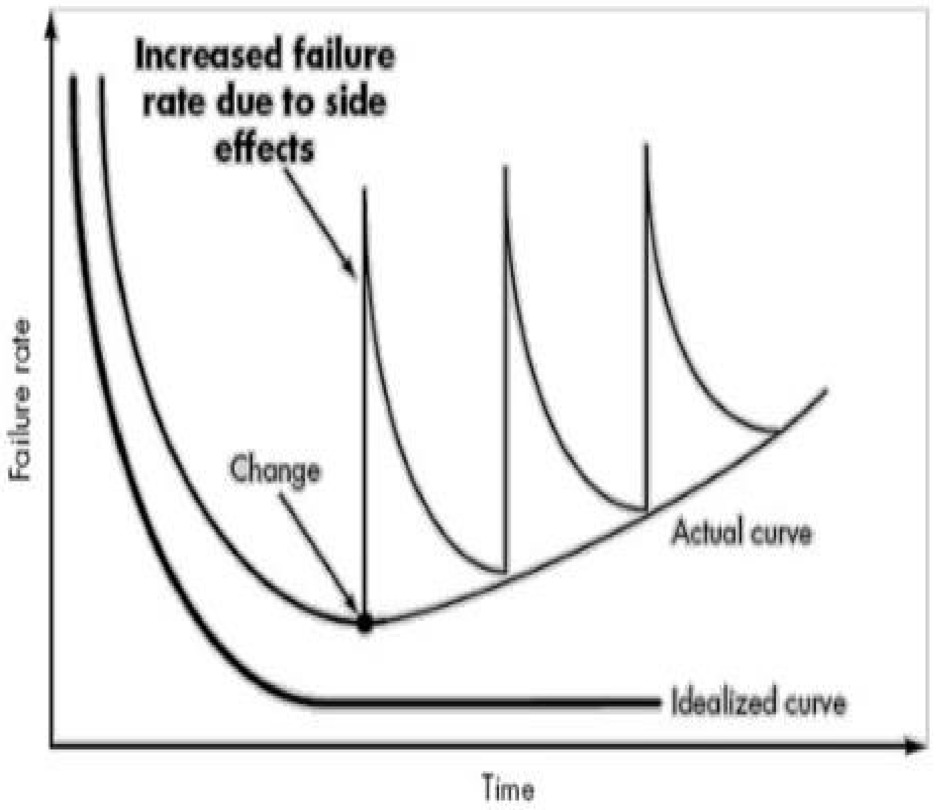


Fig: FAILURE CURVE FOR SOFTWARE

THE CHANGING NATURE OF SOFTWARE

AS A WEEP

Seven Broad Categories of software are challenges for software engineers

- **System software-** System software is a collection of programs written to service other programs. System software: such as compilers, editors, file management utilities.
- **Application software**: stand-alone programs for specific needs. This software are used to controls business needs. Ex: Transaction processing.
- **Artificial intelligence software-** Artificial intelligence (AI) software makes use of nonnumeric algorithms to solve complex problems. Application within this area include robotics, pattern recognition, game playing.

- **Engineering and scientific software**-Engineering and scientific software have been characterized by "number crunching" algorithm.
- **Embedded software** resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)
- **Product-line software** focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)
- **WebApps** (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.

SOFTWARE ENGINEERING

- *[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.*
- *Or*
  - The IEEE definition:
- *Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.*

Software engineering - Layered technology

- Software engineering is a fully layered technology.
- To develop a software, we need to go from one layer to another.
- All these layers are related to each other and each layer demands the fulfillment of the previous layer.

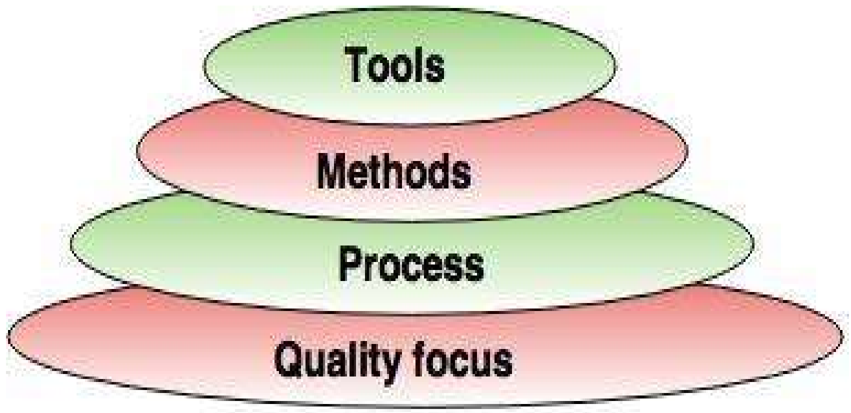


Fig. - Software Engineering Layers

- **The layered technology consists of:**

  1. Quality focus
  The characteristics of good quality software are:
- Correctness of the functions required to be performed by the software.
- Integrity i.e. providing security so that the unauthorized user cannot access information or data.
- Usability i.e. the efforts required to use or operate the software.
- **2. Process**
- It is the base layer or foundation layer for the software engineering. It covers all activities, actions and tasks required to be carried out for software development.
- **3. Methods**
- It provides the technical way to implement the software. It includes collection of tasks starting from communication, requirement analysis, analysis and design modelling, program construction, testing and support.
- **4. Tools**-The software engineering tool is an automated support for the software development. The tools are integrated i.e the information created by one tool can be used by the other tool.
-

THE SOFTWARE PROCESS

- A process is a collection of activities, actions, and tasks that are performed when some work product is to be created. An activity strives to achieve a broad objective with which software engineering is to be applied. An action encompasses a set of tasks that produce a major work .A task focuses on a small, but well-defined objective that produces a tangible outcome.
- A generic process framework for software engineering encompasses five activities:
- **Communication**- Before any technical work can commence, it is critically important to communicate and collaborate with the customer (and other stakeholders[redacted] The intent is to understand stakeholders' objectives for the project and to gather requirements that help define software features and functions.
- **Planning**-Any complicated journey can be simplified if a map exists. The map—called a software project plan—defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.
- **Modeling-**A software engineer creating models to better understand software requirements and the design that will achieve those requirements.
- **Construction**-This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.
- **Deployment-**The software (as a complete entity or as a partially completed increment) is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.
-
-

- Software engineering process framework activities are complemented by a number of umbrella activities. Typical umbrella activities include:
- **Software project tracking and control**—allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.
- **Risk management**—assesses risks that may affect the outcome of the project or the quality of the product.
- **Software quality assurance**—defines and conducts the activities required to ensure software quality.
- **Technical reviews**—assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.
- **Measurement**—defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs; can be used in conjunction with all other framework and umbrella activities.
- **Software configuration management**—manages the effects of change throughout the software process.
- **Reusability management**—defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.
- **Work product preparation and production**—encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

SOFTWARE ENGINEE$RING PRACTICES

- **The Essence of Practice**
- *1. Understand the problem*
- *2. Plan a solution*
- *3. Carry out the plan*
- *4. Examine the result for accuracy*

Software Engineering Principles:

- **The Reason It All Exists**- A software system exists for one reason: to provide value to its users. All decisions should be made with this in mind. Before specifying a system requirement, before noting a piece of system functionality, before determining the hardware platforms or development processes, ask yourself questions such as: “Does this add real value to the system?” If the answer is “no,” don’t do it. All other principles support this one.
- **The Second Principle: KISS (Keep It Simple, Stupid!)** - All design should be as simple as possible, but no simpler. This is not to say that features, even internal features, should be discarded in the name of simplicity.
- **The Third Principle: Maintain the Vision -** A clear vision is essential to the success of a software Project.

The Reason It All Exists – A software system should provide real value to users; if a feature doesn't, don't include it.

KISS (Keep It Simple, Stupid!) – Keep designs as simple as possible without sacrificing necessary functionality.

Maintain the Vision – A clear, well-defined vision is crucial for the success of a software project.

What You Produce, Others Will Consume – Always design and develop software considering that others will use or maintain it.

Be Open to the Future – Software should be adaptable to future changes for long-term value.

Plan Ahead for Reuse – Reusing code and design elements saves time, effort, and enhances efficiency.

Think! – Careful and thoughtful planning leads to better, more effective software solutions.

- **The Fourth Principle: What You Produce, Others Will Consume –**, Always specify, design, and implement knowing someone else will have to understand what you are doing. The audience for any product of software development is potentially large. Specify with an eye to the users.
- **The Fifth Principle: Be Open to the Future -** A system with a long lifetime has more value. these systems must be ready to adapt to these and other changes. **The Sixth Principle: Plan Ahead for Reuse -** Reuse saves time and effort. The reuse of code and designs has been major benefit of using object-oriented technologies.
- **The Seventh principle: Think! -** Placing clear, complete thought before action almost always produces better results. When you think about something, you are more likely to do it right. You also gain knowledge about how to do it right again. If you do think about something and still do it wrong, it becomes a valuable experience

SOFTWARE MYTHS

Software myths—erroneous beliefs about software and the process that is used to build it—can be traced to the earliest days of computing.

Today, most knowledgeable software engineering professionals recognize myths for what they are—misleading attitudes that have caused serious problems for managers and practitioners alike. However, old attitudes and habits are difficult to modify, and remnants of software myths remain.

- 1)Management myths
- 2)Customer myths
- 3)Practitioner myths

Management myths.

Managers with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality.

Myth: We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?

Reality: The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it adaptable? Is it streamlined to improve time-to-delivery while still maintaining a focus on quality? In many cases, the answer to all of these questions is "no."

Customer myths.

A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing/sales department, or an outside company that has requested software under contract

Myth: A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.

Reality: Although a comprehensive and stable statement of requirements is not always possible, an ambiguous "statement of objectives" is a recipe for disaster. Unambiguous requirements (usually derived iteratively) are developed only through effective and continuous communication between customer and developer.

Practitioner's myths.

\ Myths that are still believed by software practitioners have been fostered by over 50 years of programming culture. During the early days, programming was viewed as an art form. Old ways and attitudes die hard.

Myth: Once we write the program and get it to work, our job is done.

Reality: Someone once said that "the sooner you begin 'writing code,' the longer it'll take you to get done." Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Software Engineering

Legacy Systems

Outline

- Introduction
- Legacy System Structures
- Legacy System Design
- Legacy System Assessment

Introduction..

- *Legacy systems*: old computer-based systems still in use by organizations
  - Many of them still business critical
  - Incorporate many changes made over the years
  - Many people have been involved in these changes
  - Replacing legacy systems with new systems is risky, yet keeping them means new changes become more and more expensive

.Introduction.

- Risks of replacing a legacy system:
  - Specification is difficult because existing documentation is typically incomplete
  - Changing business processes (now adjusted to the system) may entail high costs
  - Undocumented, yet important business rules may be embedded in the system; a new system may break these rules
  - The new system may be delivered late, may cost more than expected, and may not function properly

..Introduction

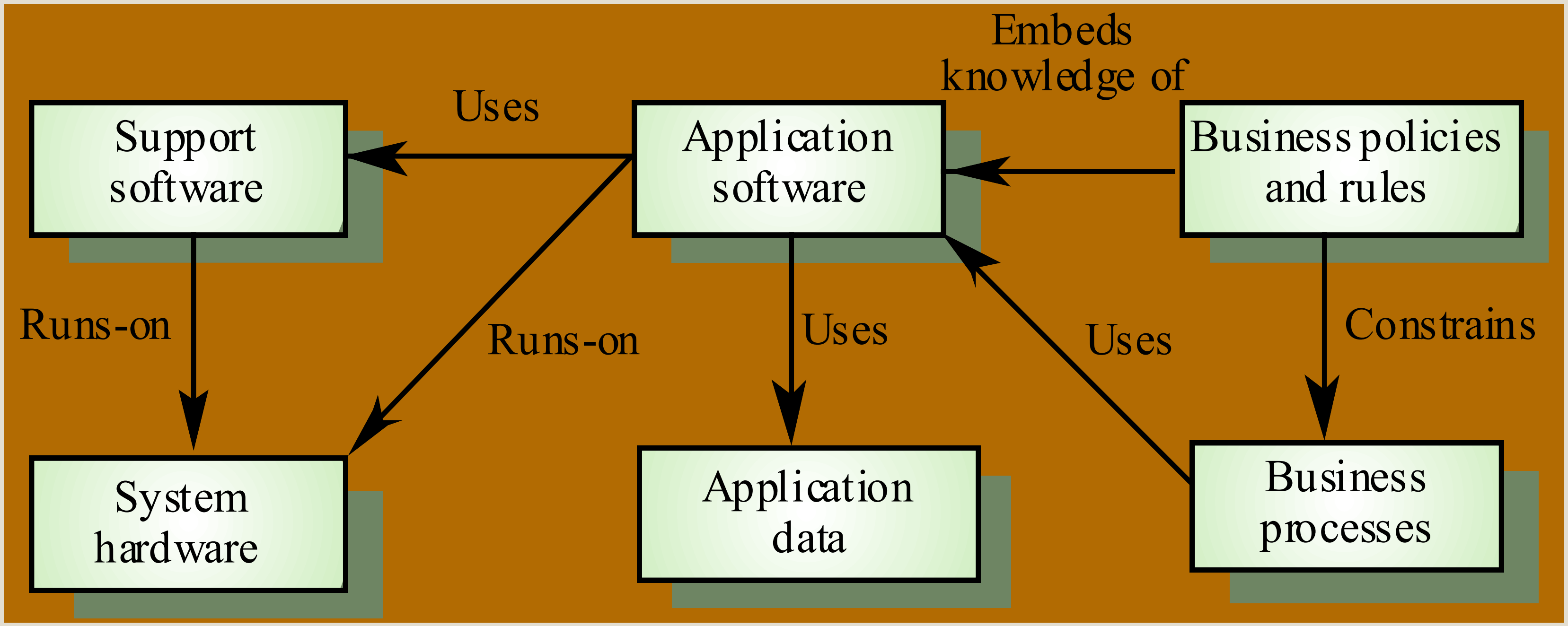
- Factors that make changes to legacy systems expensive:
    - In large systems, different parts were implemented by different teams, without consistent programming style
    - It is difficult to find personnel who knows the obsolete programming languages used in old systems
    - In may cases the only documentation is provided by the source code; even this may be missing
    - It is difficult to understand the system given its *ad hoc* updating over the years
    - Data used by the system is difficult to understand and manipulate; it can also be obsolete and/or redundant

Legacy system structures….

- Legacy systems involve more than software (they are computer-based systems). Typical logical parts of a legacy system are:
  - System hardware
  - Support software
  - Application software (legacy software systems)
  - Application data
  - Business processes
  - Business policies and rules

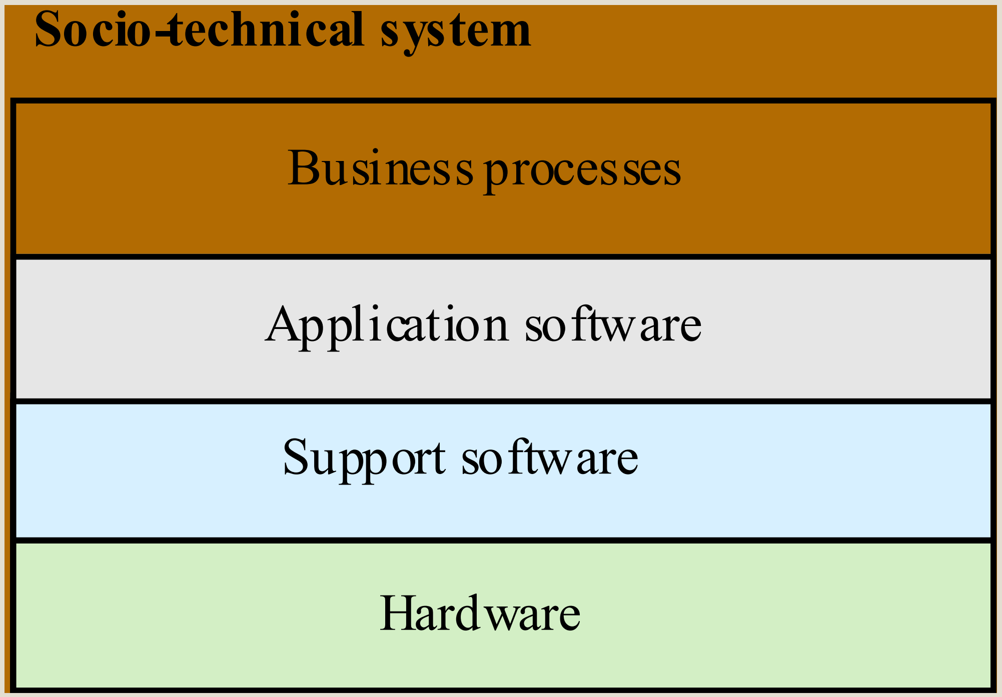
.Legacy system structures…

- Legacy system components [Fig. 26.1 Somm00]



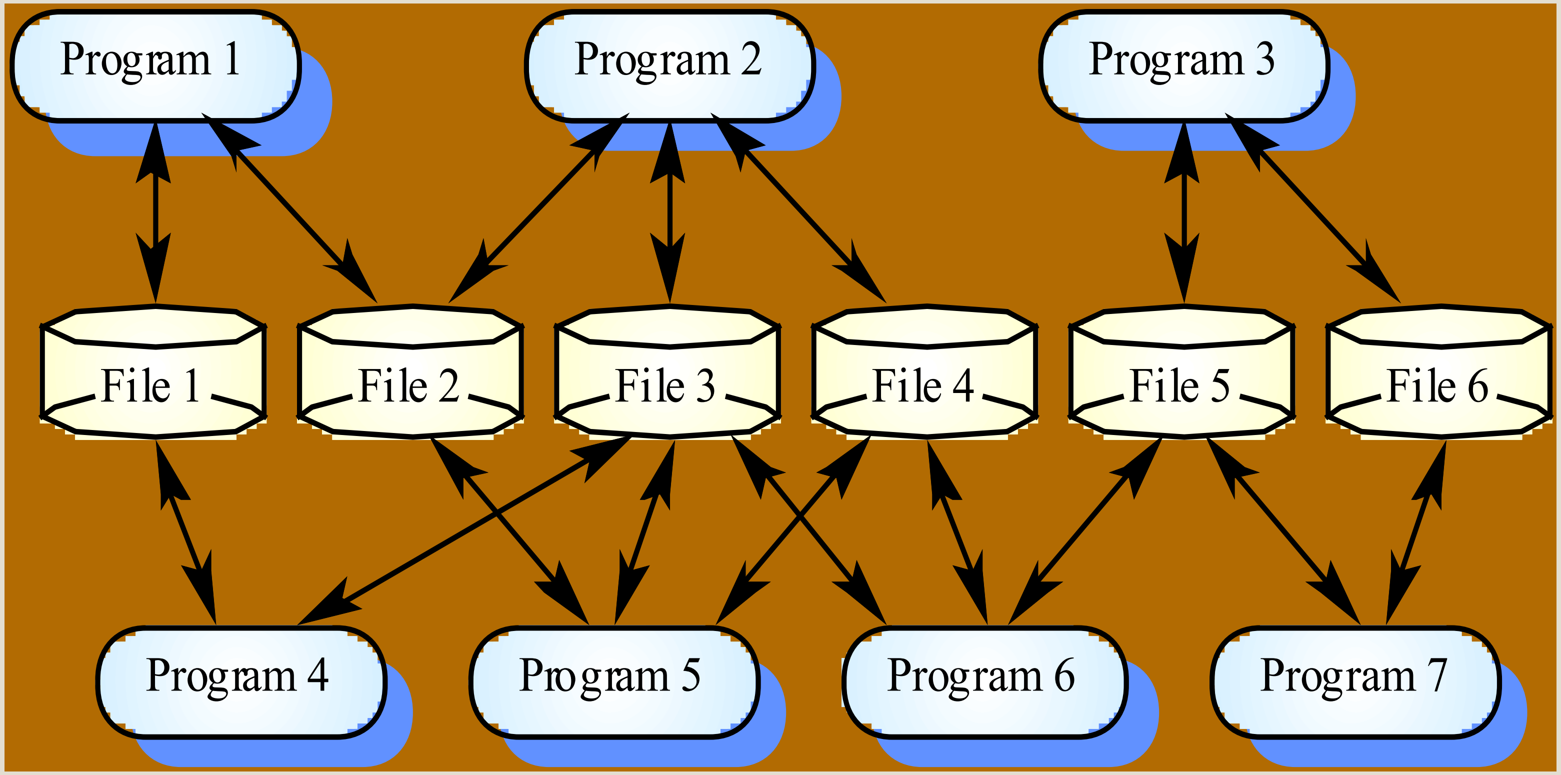
..Legacy system structures..

- Alternative view of legacy systems: a layered model [Fig. 26.2 Somm00]



…Legacy system structures.

- Legacy application software [Fig. 26.3 Somm00] Both programs and data files have been added over the years: components are heterogeneous and strongly coupled

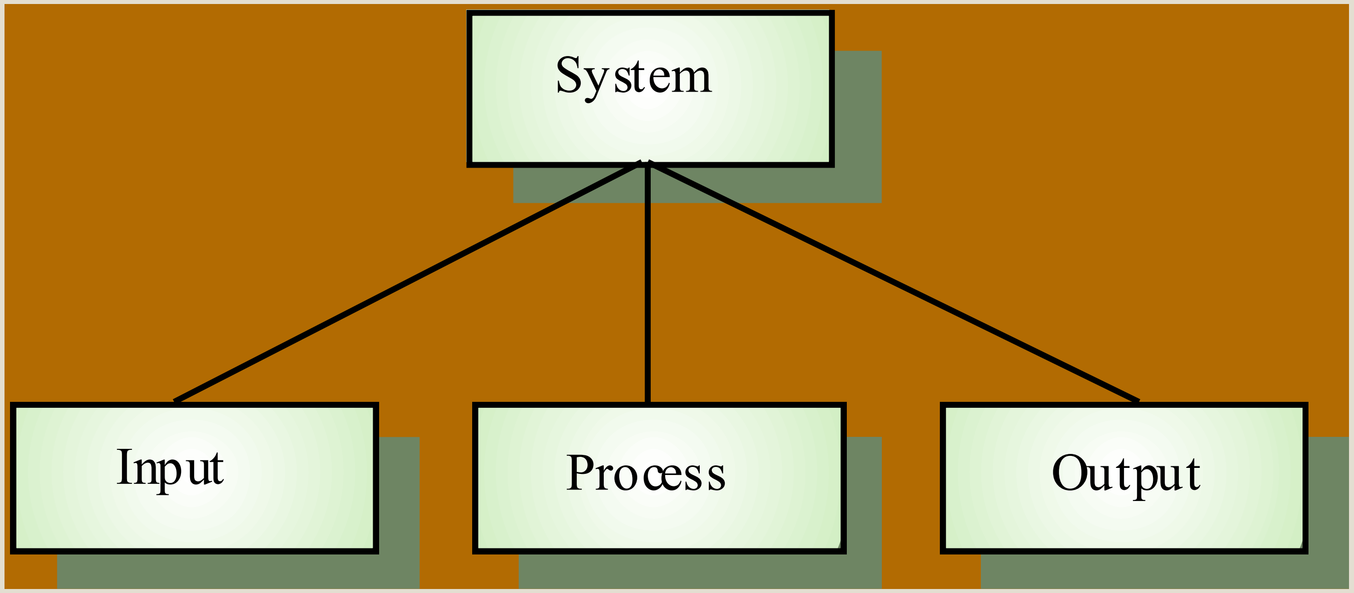


Legacy system design.....

- Legacy systems design is typically function oriented. Two main classes:
    - *Batch processing systems*: both input and output is provided in “batches,” e.g., a payroll system
    - *Transaction processing systems*: input & output related to a database transaction, e.g., a flight reservation system
- Both batch processing and transaction processing systems usually follow an *IPO model*:
    - Input: inputs are collected from one or more sources
    - Processing: some computations are performed on inputs
    - Output: results are provided either in batches or as single-transaction outputs
    - All IPO components may further be organized according the IPO model

.Legacy system design….

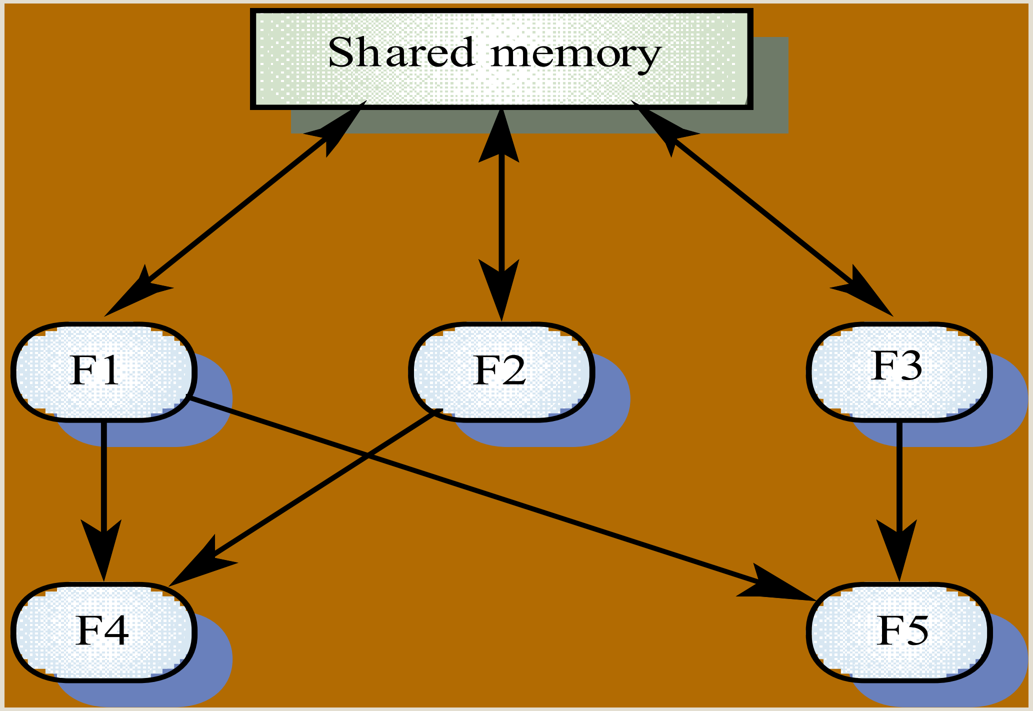
- The IPO Model [Fig. 26.7 Somm00].



..Legacy system design...

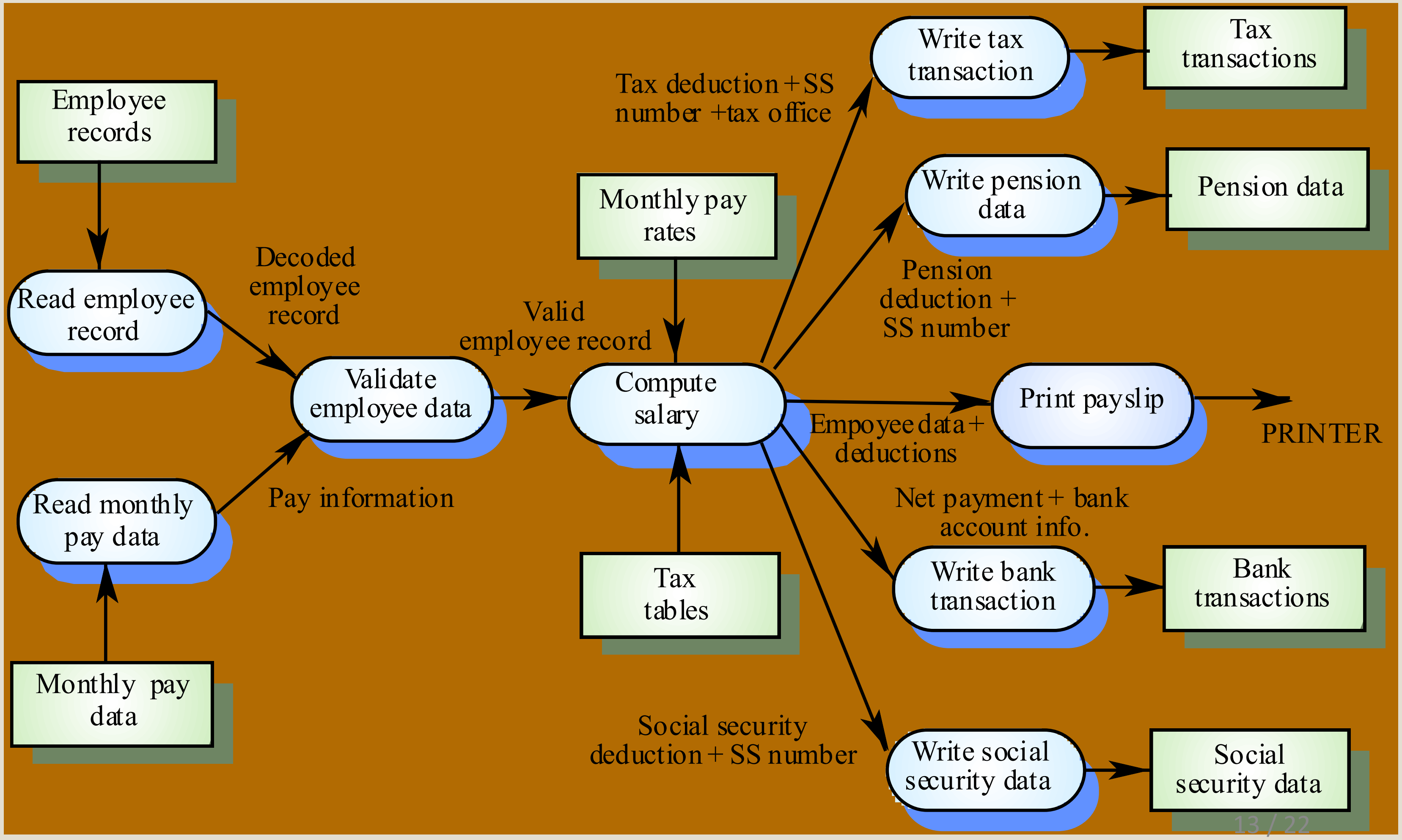
- A function-oriented view on design [Fig. 26.6, Somm00].

  The main problem is the sharing of data (system state information), which can lead to unpredicted changes in functions' behavior. Also, it is unlikely that a single person understands all parts of a large system.



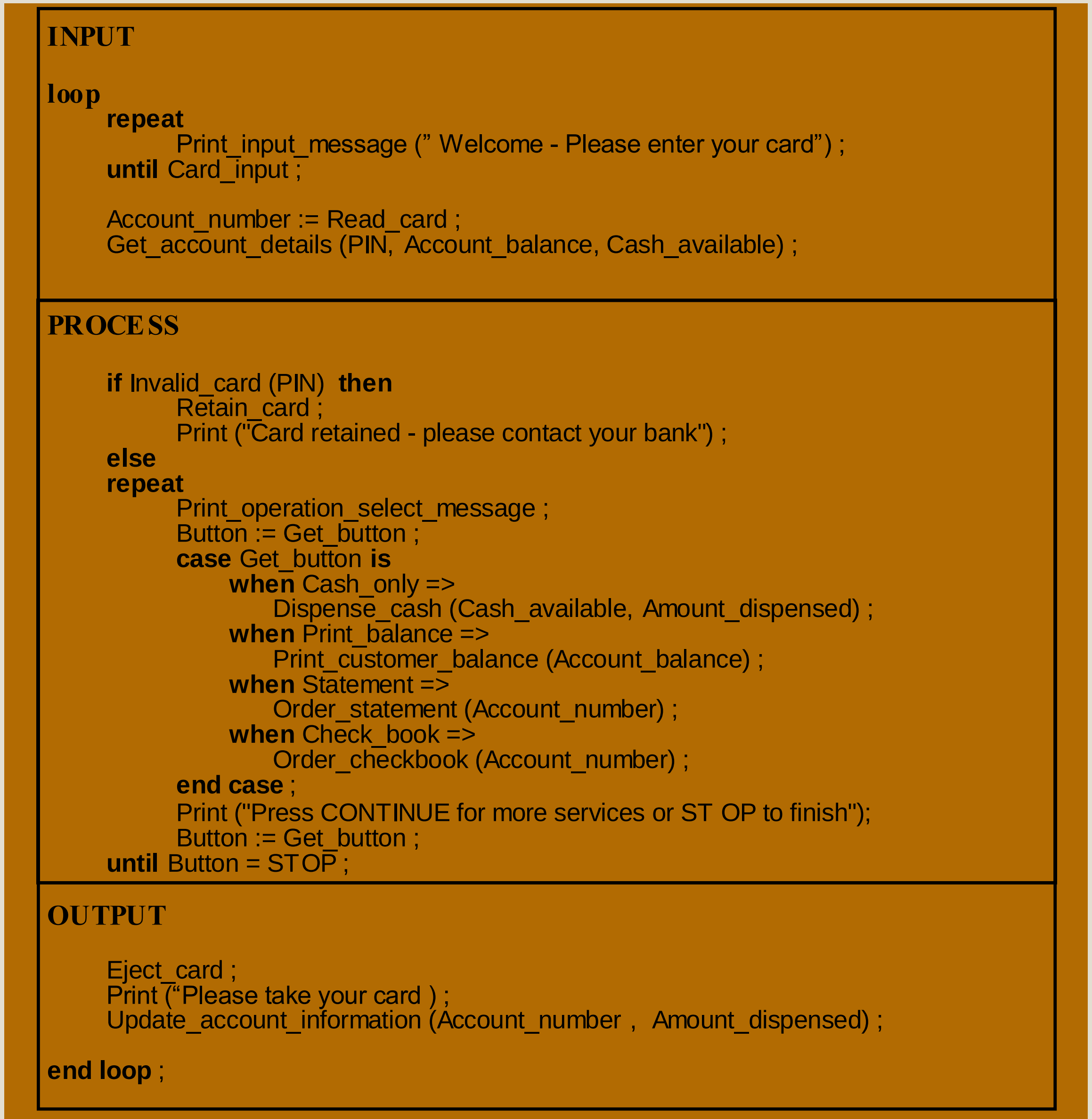
…Legacy system design..

- Data flow diagrams are often used to design function-oriented systems. Example: a payroll system [Fig. 26.8 Somm00]



….Legacy system design.

- A transaction processing system: design of an ATM [Fig. 26.9 Somm00]



```
INPUT

loop
        repeat
                Print_input_message (" Welcome - Please enter your card") ;
        until Card_input ;

        Account_number := Read_card ;
        Get_account_details (PIN, Account_balance, Cash_available) ;

PROCESS

        if Invalid_card (PIN)  then
                Retain_card ;
                Print ("Card retained - please contact your bank") ;
        else
        repeat
                Print_operation_select_message ;
                Button := Get_button ;
                case Get_button is
                        when Cash_only =>
                                Dispense_cash (Cash_available, Amount_dispensed) ;
                        when Print_balance =>
                                Print_customer_balance (Account_balance) ;
                        when Statement =>
                                Order_statement (Account_number) ;
                        when Check_book =>
                                Order_checkbook (Account_number) ;
                end case ;
                Print ("Press CONTINUE for more services or ST OP to finish");
                Button := Get_button ;
        until Button = STOP ;

OUTPUT

        Eject_card ;
        Print ("Please take your card ) ;
        Update_account_information (Account_number ,  Amount_dispensed) ;

end loop ;
```

…..Legacy system design

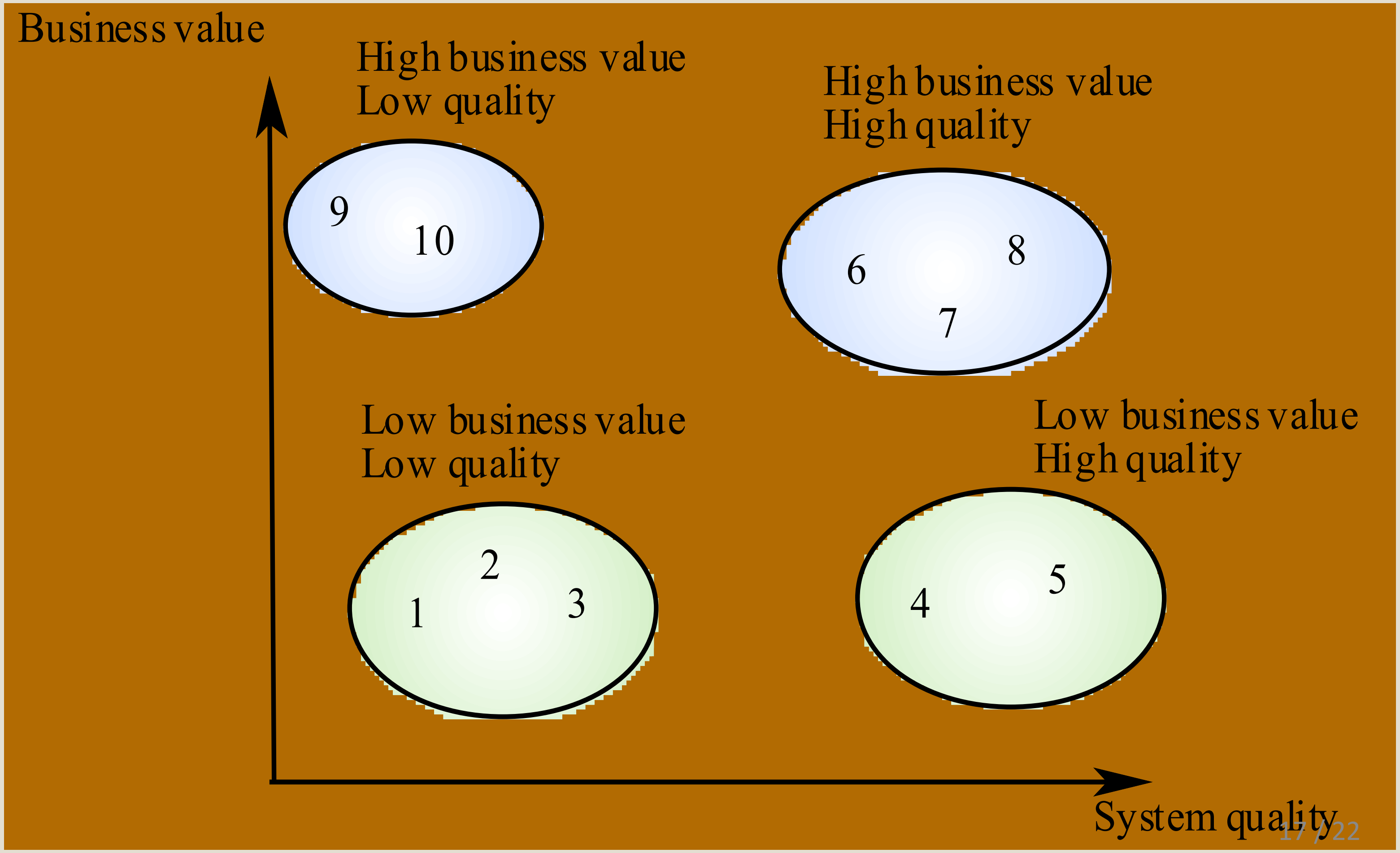
- Function oriented design is not restricted to legacy systems. Can be applied to new systems where:
  - Data processing relies on processing transactions and updating a data store
  - The company has invested heavily in structured methods, including staff training, development practices, and CASE tools
- An interesting challenge for a company: to work with both approaches, function-oriented and object-oriented

Legacy system assessment…..

- Strategic approaches for dealing with legacy systems:
  - Scrap the system completely
    - When business practices have changed and no longer depend significantly on the system (they may be supported by new COTS)
  - Continue to maintain the system
    - The system works well, is fairly stable, and users do not request many changes
  - Transform the system to improve maintainability
    - When system quality was affected negatively by changes, yet changes are still required
  - Replace the system with a new one
    - When obsolete hardware precludes further operation or the new system can be built at reasonable cost

.Legacy system assessment….

- Assessing legacy systems example [Fig. 26.10 Somm00]



Legacy system categories

- Low quality, low business value
  - These systems should be scrapped
- Low-quality, high-business value
  - These make an important business contribution but are expensive to maintain. Should be re-engineered or replaced if a suitable system is available
- High-quality, low-business value
  - Replace with COTS, scrap completely or maintain
- High-quality, high business value
  - Continue in operation using normal system maintenance

Business value assessment

- Assessment should take different viewpoints into account
  - System end-users
  - Business customers
  - Line managers
  - IT managers
  - Senior managers
- Interview different stakeholders and collate results

..Legacy system assessment…

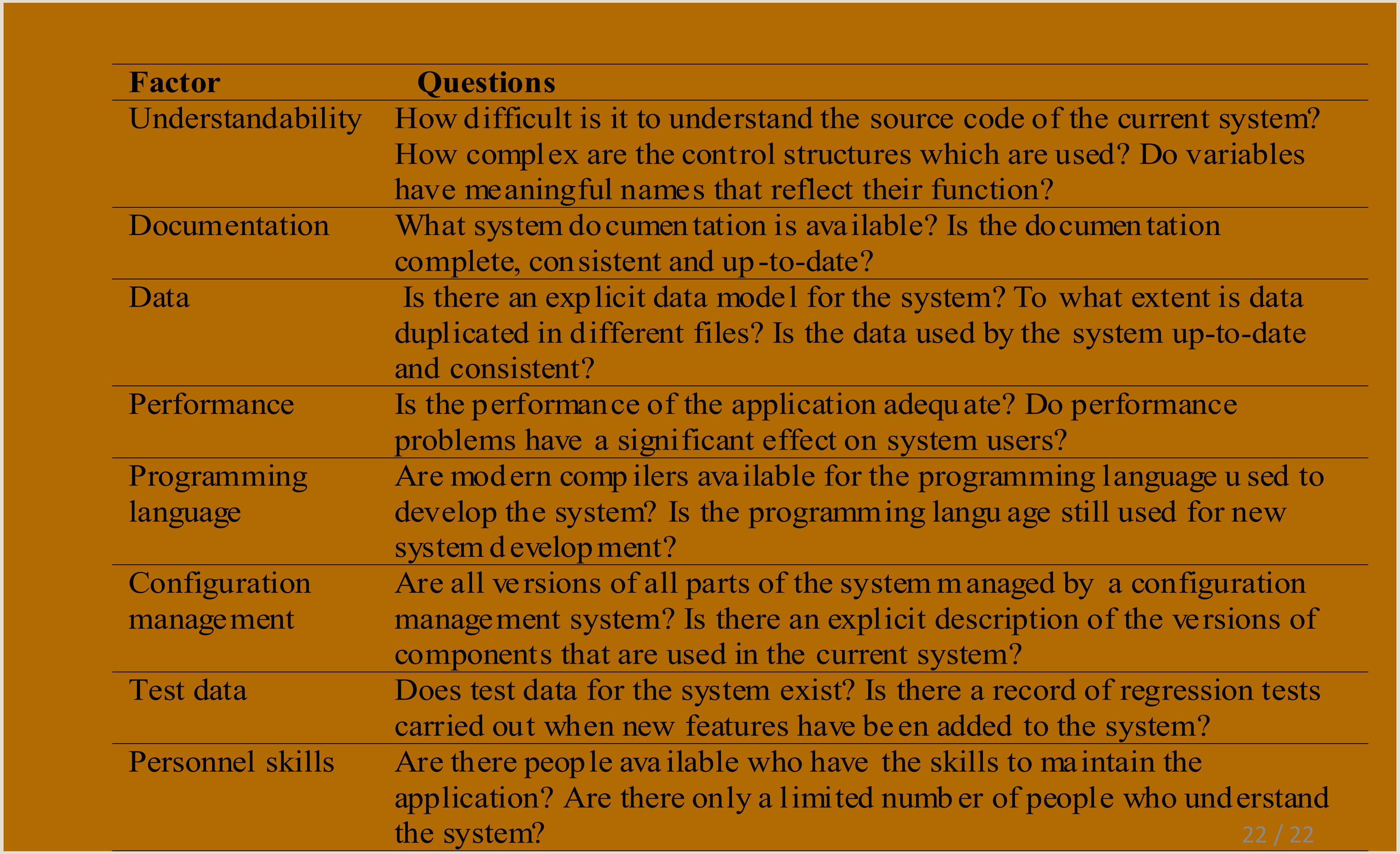
- Assessment of legacy systems includes:
  - Business value assessment (subjective). Viewpoints:
    - End-users: look at system’s functionality and performance
    - Customers: look at the quality of services provided
    - Business managers: assess the usefulness of the system in terms of business support
    - IT managers: are concerned with the availability of technical support for the system
    - Senior managers: interested in system’s contribution to the business goals
  - System quality assessment (next)

…Legacy system assessment..

- System quality assessment. Look at all components of the system. Hence:
  - Business process assessment. Possible questions:
    - Are defined process models and procedures in place?
    - Are processes applied consistently across the company?
    - What adaptations have been made?
    - Are relationships with other business processes necessary?
    - Are processes suitably supported by application software?
  - Environment assessment: support software & hardware platform (maintenance costs, faults, etc. – slide 21)
  - Application software assessment. Factors considered as in slide 22 and quantitative data such as:
    - Number of system change requests
    - Number of different user interfaces
    - Volume of data used by the system

…..Legacy system assessment

- Factors in application software assessment [Fig. 26.12 Somm00]



Factor	**Questions**
Understandability	How difficult is it to understand the source code of the current system? How complex are the control structures which are used? Do variables have meaningful names that reflect their function?
Documentation	What system documentation is available? Is the documentation complete, consistent and up-to-date?
Data	Is there an explicit data model for the system? To what extent is data duplicated in different files? Is the data used by the system up-to-date and consistent?
Performance	Is the performance of the application adequate? Do performance problems have a significant effect on system users?
Programming language	Are modern compilers available for the programming language used to develop the system? Is the programming language still used for new system development?
Configuration management	Are all versions of all parts of the system managed by a configuration management system? Is there an explicit description of the versions of components that are used in the current system?
Test data	Does test data for the system exist? Is there a record of regression tests carried out when new features have been added to the system?
Personnel skills	Are there people available who have the skills to maintain the application? Are there only a limited number of people who understand the system?

Key points

- A legacy system is an old system that still provides essential business services
- Legacy systems are not just application software but also include business processes, support software and hardware
- Most legacy systems are made up of several different programs and shared data
- A function-oriented approach has been used in the design of most legacy systems

Key points

- The structure of legacy business systems normally follows an input-process-output model
- The business value of a system and its quality should be used to choose an evolution strategy
- The business value reflects the system’s effectiveness in supporting business goals
- System quality depends on business processes, the system’s environment and the application software