System analysis and design –

Class-bba6th sem.

System Implementation-

System Implementation uses the structure created during **architectural design** and the results of system analysis to construct **system elements** that meet the **stakeholder requirements** and **system requirements** developed in the early **life cycle** phases. These system elements are then integrated to form intermediate **aggregates** and finally the complete **system-of-interest (SoI)**. See System Integration.

Contents

[hide]

- 1Definition and Purpose
- 2Process Approach
  - 2.1Purpose and Principle of the Approach
  - 2.2Activities of the Process
  - 2.3Artifacts and Ontology Elements
  - 2.4Methods, Techniques, and Tools
  - 2.5Checking and Correctness of Implementation
- 3References
  - 3.1Works Cited
  - 3.2Primary References
  - 3.3Additional References

Definition and Purpose

Implementation is the process that actually yields the lowest-level system elements in the system hierarchy (system breakdown structure). System elements are made, bought, or reused. Production involves the hardware fabrication processes of forming, removing, joining, and finishing, the software realization processes of coding and testing, or the operational procedures development processes for operators' roles. If implementation involves a production process, a manufacturing system which uses the established technical and management processes may be required.

The purpose of the implementation process is to design and create (or fabricate) a system element conforming to that element's design properties and/or requirements. The element is constructed employing appropriate technologies and industry practices. This process bridges the system definition processes and the integration process. Figure 1 portrays how the outputs of system definition relate to system implementation, which produces the implemented (system) elements required to produce aggregates and the SoI.

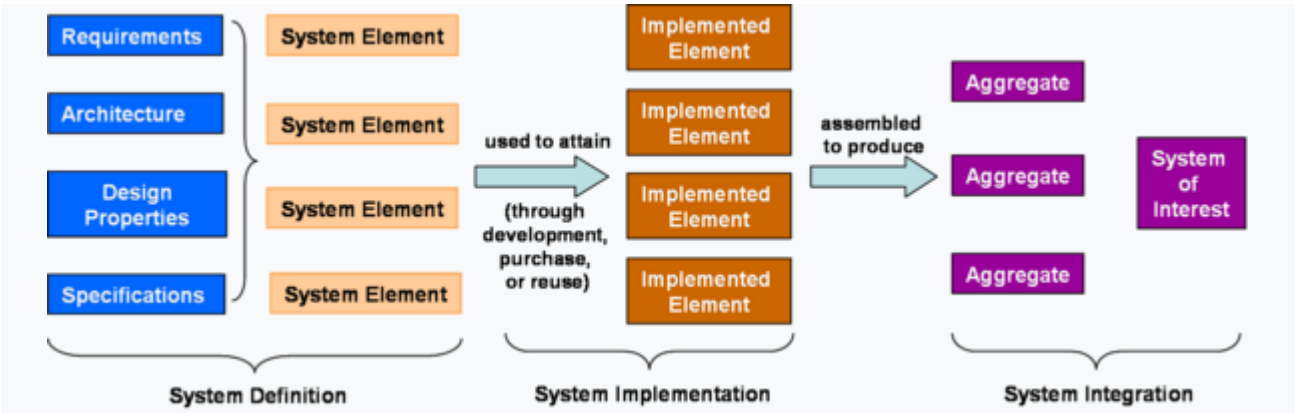


Figure 1. Simplification of How the Outputs of System Definition Relate to System Implementation, which Produces the System Elements Required to Produce Systems and Subsystems. (SEBoK Original)

Process Approach

Purpose and Principle of the Approach

During the implementation process, engineers apply the design properties and/or requirements allocated to a system element to design and produce a detailed description. They then fabricate, code, or build each individual element using specified materials, processes, physical or logical arrangements, standards, technologies, and/or information flows outlined in detailed descriptions (drawings or other design documentation). A system element will be verified against the detailed description of properties and validated against its requirements.

If subsequent verification and validation (V&V) actions or configuration audits reveal discrepancies, recursive interactions occur, which includes predecessor activities or processes, as required, to mitigate those discrepancies and to modify, repair, or correct the system element in question. Figure 2 provides the context for the implementation process from the perspective of the U.S. Defense Acquisition University (DAU).

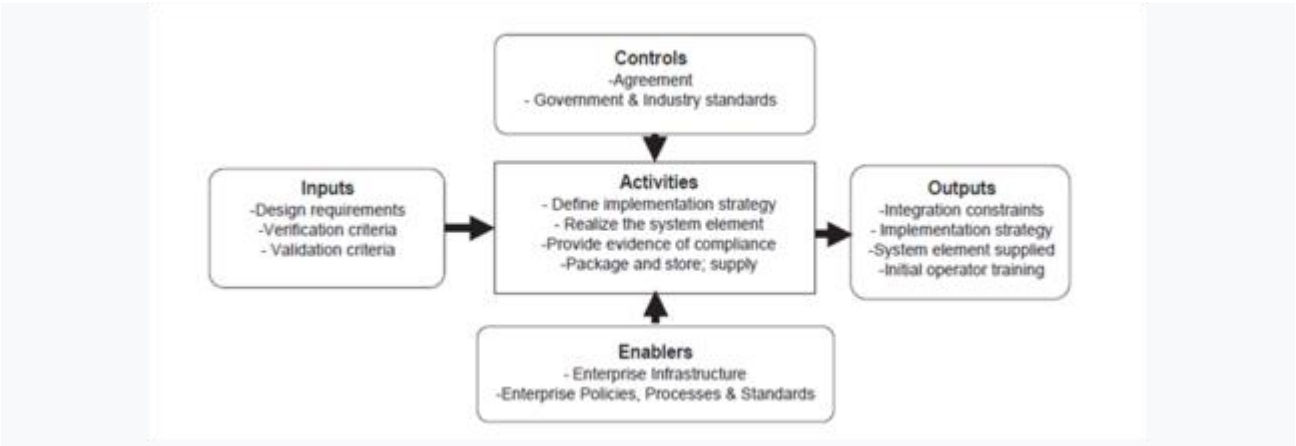


Figure 2. Context Diagram for the Implementation Process (DAU 2010). Released by the Defense Acquisition University (DAU)/U.S. Department of Defense (DoD).

Such figures provide a useful overview of the **systems engineering** (SE) community's perspectives on what is required for implementation and what the general results of implementation may be. These are further supported by the discussion of implementation inputs, outputs, and activities found in the National Aeronautics and Space Association's (NASA's) *Systems Engineering Handbook* (NASA 2007).

It is important to understand that these views are process -oriented. While this is a useful model, examining implementation only in terms of process can be limiting.

Depending on the technologies and systems chosen when a decision is made to produce a system element, the implementation process outcomes may generate constraints to be applied on the architecture of the higher-level system; those constraints are normally identified as derived system requirements and added to the set of system requirements applicable to this higher-level system. The architectural design has tomust take those constraints into account.

If the decision is made to purchase or reuse an existing system element, it has tomust be identified as a constraint or system requirement applicable to the architecture of the higher-level system. Conversely, the implementation process may involve some adaptation or adjustments to the system requirement in order to be integrated into a higher-level system or aggregate.

Implementation also involves packaging, handling, and storage, depending on the concerned technologies and where or when the system requirement needs to be integrated into a higher-level aggregate. Developing the supporting documentation for a system requirement, such as the manuals for operation, maintenance, and/or installation, is also a part of the implementation process; these artifacts are utilized in the system deployment and use phase. The system element requirements and the associated verification and validation criteria are inputs to this process; these inputs come from the **architectural design** process detailed outputs.

Execution of the implementation process is governed by both industrial and government standards and the terms of all applicable agreements. This may include conditions for packaging and storage, as well as preparation for use activities, such as operator training. In addition, packaging, handling, storage, and transportation (PHS&T) considerations will constrain the implementation activities. For more information, refer to the discussion of PHS&T in the System Deployment and Use article. The developing or integrating organization will likely have enterprise-level safety practices and guidelines that must also be considered.

Activities of the Process

The following major activities and tasks are performed during this process:

- **Define the implementation strategy** - Implementation process activities begin with detailed design and include developing an implementation strategy that defines fabrication and coding procedures, tools and equipment to be used, implementation tolerances, and the means and criteria for auditing configuration of resulting elements to the detailed design documentation. In the case of repeated system element implementations (such as for mass manufacturing or replacement elements), the implementation strategy is defined and refined to achieve consistent and repeatable element production; it is retained in the project decision database for future use. The implementation strategy contains the arrangements for packing, storing, and supplying the implemented element.
- **Realize the system element** - Realize or adapt and produce the concerned system element using the implementation strategy items as defined above. Realization or adaptation is conducted with regard to standards that govern applicable safety, security, privacy, and environmental guidelines or legislation and the practices of the relevant implementation technology. This requires the fabrication of hardware elements, development of software elements, definition of training capabilities, drafting of training documentation, and the training of initial operators and maintainers.
- **Provide evidence of compliance** - Record evidence that the system element meets its requirements and the associated verification and validation criteria as well as the legislation policy. This requires the conduction of peer reviews and unit testing, as well as inspection of operation and maintenance manuals. Acquire measured properties that characterize the implemented element (weight, capacities, effectiveness, level of performance, reliability, availability, etc.).
- **Package, store, and supply the implemented element** - This should be defined in the implementation strategy.

Artifacts and Ontology Elements

This process may create several artifacts such as:

- an implemented system
- implementation tools
- implementation procedures
- an implementation plan or strategy
- verification reports
- issue, anomaly, or trouble reports
- change requests (about design)

This process handles the ontology elements shown in Table 1 below.

Table 1. Main Ontology Elements as Handled within System Element Implementation. (SEBoK Original)

Element	Definition Attributes (examples)
Implemented Element	An implemented element is a system element that has been implemented. In the case of hardware it is marked with a part/serial number. Identifier, name, description, type (hardware, software application, software piece, mechanical part, electric art, electronic component, operator role, procedure, protocol, manual, etc.)
Measured Property	A measured property is a characteristic of the implemented element established after its implementation. The measured properties characterize the implemented system element when it is completely realized, verified, and validated. If the implemented element complies with a design property, the measured property should equal the design property. Otherwise one has tomust identify the difference or non-conformance which treatment could conclude to modify the design property and possibly the related requirements, or to modify (correct, repair) the implemented element, or to identify a deviation. Identifier, name, description, type (effectiveness, availability, reliability, maintainability, weight, capacity, etc.), value, unit, etc.

The main relationships between ontology elements are presented in Figure 3.

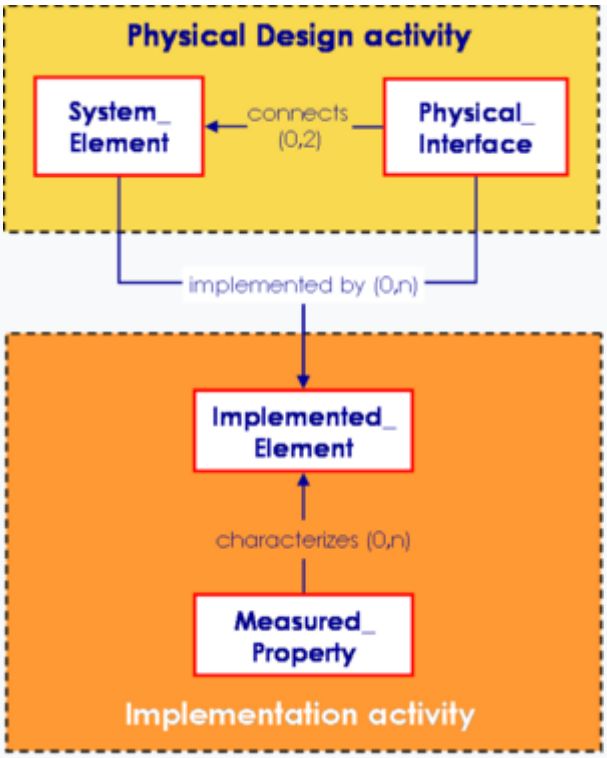


Figure 3. Implementation Elements Relationships with Other Engineering Elements. (SEBoK Original)

Methods, Techniques, and Tools

There are many software tools available in the implementation and integration phases. The most basic method would be the use of N-squared diagrams as discussed in Jeff Grady's book *System Integration* (Grady 1994).

Checking and Correctness of Implementation

Proper implementation checking and correctness should include testing to determine if the implemented element (i.e., piece of software, hardware, or other product) works in its intended use. Testing could include mockups and breadboards, as well as modeling and simulation of a prototype or completed pieces of a system. Once this is completed successfully, the next process would be system integration. **ystem Implementation** uses the structure created during **architectural design** and the results of system analysis to construct **system elements** that meet the **stakeholder requirements** and **system requirements** developed in the early **life cycle** phases. These system elements are then integrated to form intermediate **aggregates** and finally the complete **system-of-interest (SoI)**. See System Integration.

Evaluation of software and hardware

All computers are made up of hardware, software and data. Specific types of hardware and software are used for particular tasks.

Software evaluation

Evaluation is performed to ensure the most suitable piece of software (or hardware) is purchased. Typically evaluation criteria will include:

- **End user needs** - what does the user of the software want to do, what are their present skills and how do they intend to use the software? It is important to be very clear about the problem that is to be tackled by the software. For example, a disabled person who wants to write letters but cannot type might strongly consider software with lots of ready-made letter templates that can then be added to using voice recognition.
- **Functionality** - does the software perform the functions required? Does it have specific facilities? For example, someone buying a spreadsheet application might need to produce graphs and charts.
- **Performance** - how well does the software work? This is normally available as benchmark test reports where independent tests have been carried out using the software.
- **Ease of use** - how easy is the software to use? Is there built-in help? It is important to be happy with the user interface.
- **Compatibility with existing data** - will the new software be able to read any data that is already in use, ie in a different format or file type? If not, is it easy to convert existing files to a readable format?
- **Compatibility with existing hardware** - software is written to run on a specific operating system, eg Windows, OSX (Macs) or Linux. It is sometimes written to run on and take advantage of specific hardware too. The new software needs to be compatible with the existing operating system and hardware.
- **Robustness** - how does the software handle problems? Robust software works well in combination with different hardware and software without crashing.
- **Cost** - costs have to be weighed against the benefits that the software will bring. These may be about making more money or doing something quickly or with fewer staff hours involved. Price doesn't always dictate the best piece of software for the job, ie just because it's more expensive it doesn't necessarily means it's better.
- **Support** - the level of support when using the software can be crucial to making it a success or failure. Is a telephone or web based helpdesk available for the software? Are there any tutorials or training courses available?
- **Customisation** - will the software allow users to change the look and feel so that it does exactly what they need? If so, is this easy to do?

Technical Infrastructure and Operational Practices and Infrastructure

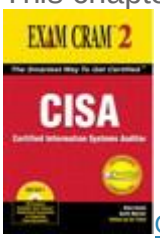
- By Allen Keele and Keith Mortier
- Apr 4, 2005

Contents

1. IT Organizational Structure
2. Evaluating Hardware Acquisition, Installation, and Maintenance
3. Evaluating Systems Software Development, Acquisition, Implementation, and Maintenance
4. Evaluating Network Infrastructure Acquisition, Installation, and Maintenance
5. The TCP/IP Protocol Suite
6. Routers
7. Internet, Intranet, and Extranet
8. Evaluating IS Operational Practices
9. Evaluating the Use of System Performance and Monitoring Processes, Tools, and Techniques
10. Exam Prep Questions

This chapter is from the book

This chapter is from the book



CISA Exam Cram: Certified Information Systems Auditor

Learn More Buy

Evaluating Hardware Acquisition, Installation, and Maintenance

A significant part of the information architecture is the computing hardware. These systems include the following:

- **Processing components**—The central processing unit (CPU). The CPU contains the electrical/electronic components that control or direct all operations in the computer system. A majority of devices within the information architecture are CPUs (supercomputers, mainframes, minicomputer, microcomputer, laptops, and PDAs).

- **Input/output components**—The I/O components are used to pass instructions or information to the computer and to generate output from the computer. These types of devices include the keyboard, the mouse (input), and monitors/terminal displays.

Computers logically fall into categories and differ depending on the processing power and size for the organization. The following are the basic categories for computers:

- **Supercomputers**—These types of computers have a large capacity of processing speed and power. They are generally used for complex mathematical calculations. Supercomputers generally perform a small number of very specific functions that require extensive processing power (decryption, modeling, and so on). Supercomputers differ from mainframes in that mainframes can use diverse concurrent programs.

- **Mainframes**—Mainframes are large general-purpose computers that support large user populations simultaneously. They have a large range of capabilities that are controlled by the operating system. A mainframe environment, as opposed to a client/server environment, is generally more controlled with regard

to access and authorization to programs; the entire processing function takes place centrally on the mainframe. Mainframes are multiuser, multithreading, and multiprocessing environments that can support batch and online programs.

- **Minicomputer**—Minicomputers are essentially smaller mainframes. They provide similar capabilities but support a smaller user population (less processing power).

- **Microcomputer (personal computers)**—Microcomputers are primarily used in the client/server environment. Examples include file/print servers, email servers, web servers, and servers that house database- management systems. Individual workstations also fall into the microcomputer category and are used for word processing, spreadsheet applications, and individual communications (email). Microcomputers are generally inexpensive because they do not have the processing power of larger minicomputers or mainframes.

- **Notebook/laptop computers**—Notebook and laptop computers are portable and allow users to take the computing power, applications, and, in some cases, data with them wherever they travel. Notebooks and laptops today have as much computing power as desktop workstations and provide battery power when traditional power is not available. Because of the mobile nature of notebook and laptop computers, they are susceptible to theft. Theft of a laptop computer is certainly the loss of a physical asset, but it also can include the loss of data or unauthorized access to the organization's information resources.

- **Personal digital assistants (PDAs)**—PDAs are handheld devices and generally have significantly less processing power, memory, and applications than notebook computers. These devices are battery powered and very portable (most can fit into a jacket pocket). Although the traditional use of a PDA is for individual organization, including the maintenance of tasks, contacts lists, calendars, and expense managers, PDAs are continually adding functionality. As of this writing, a significant number of PDAs provide wireless network access and have either commercial off-the-shelf software or custom software that enables users to access corporate information (sales and inventory, email, and so on). Most PDAs use pen (stylus)–based input instead of the traditional keyboard, effected by using either an onscreen keyboard or handwriting recognition. PDAs are synchronized with laptop/desktop computers through serial interfaces through the use of a cradle or wireless networking (802.11 or Bluetooth). The synchronization can be user initiated or automated, based on the needs of the user.

Earlier in this section, we discussed some of the attributes of computing systems, including multiprocessing, multitasking, and multithreading. These attributes are defined as follows:

- **Multitasking**—Multitasking allows computing systems to run two or more applications concurrently. This process enables the systems to allocate a certain amount of processing power to each application. In this

instance, the tasks of each application are completed so quickly that it appears to multiple users that there are no disruptions in the process.

- **Multiprocessing**—Multiprocessing links more than one processor (CPU) sharing the same memory, to execute programs simultaneously. In today's environment, many servers (mail, web, and so on) contain multiple processors, allowing the operating system to speed the time for instruction execution. The operating system can break up a series of instructions and distribute them among the available processors, effecting quicker instruction execution and response.

- **Multithreading**—Multithreading enables operating systems to run several processes in rapid sequence within a single program or to execute (run) different parts, or threads, of a program simultaneously. When a process is run on a computer, that process creates a number of additional tasks and subtasks. All the threads (tasks and subtasks) can run at one time and combine as a rope (entire process). Multithreading can be defined as multitasking within a single program.

Risks and Controls Relating to Hardware Platforms

In aligning the IT strategy with the organizational strategy, IT provides solutions that meet the objectives of the organization. These solutions must be identified, developed, or acquired. As an IS auditor, you will assess this process by reviewing control issues regarding the acquisition, implementation, and maintenance of hardware. Governance of the IT organization and corresponding policies will reduce the risk associated with acquisition, implementation, and maintenance. Configuration management accounts for all IT components, including software. A comprehensive configuration-management program reviews, approves, tracks, and documents all changes to the information architecture. Configuration of the communications network is often the most critical and time-intensive part of network management as a whole. Software development project management involves scheduling, resource management, and progress tracking. Problem management records and monitors incidents and documents them through resolution. The documentation created during the problem-management process can identify inefficient hardware and software, and can be used as a basis for identifying acquisition opportunities that serve the business objectives. Risk management is the process of assessing risk, taking steps to reduce risk to an acceptable level (mitigation) and maintaining that acceptable level of risk. Risk identification and management works across all areas of the organizational and IT processes.

CAUTION

A configuration-management audit should always verify software licensing for authorized use.

The COBIT framework provides hardware policy areas for IT functions. These policy areas can be used as a basis for control objectives to ensure that the acquisition process is clearly defined and meets the needs of the organization. The COBIT areas address the following questions:

- **Acquisition**—How is hardware acquired from outside vendors?
- **Standards**—What are the hardware compatibility standards?
- **Performance**—How should computing capabilities be tested?
- **Configuration**—Where should client/servers, personal computers, and others be used.
- **Service providers**—Should third-party service providers be used?

One of the key challenges facing IT organizations today is the speed of new technology releases in the marketplace and detailed baseline documentation for their organizations. IT organizations need a process for documenting existing hardware and then maintaining that documentation. This documentation supports the acquisition process and ensures that new technologies that meet the business objectives can be thoroughly tested to ensure that they are compatible with the existing information architecture.

Contained within the COBIT framework regarding hardware and software acquisition, the auditor will consider the control objectives defined in Table 3.1.

Table 3.1 Acquisition Control Objectives

Identify Automated Solutions	**Control Objective**
1.1 Definition of Information Requirements	The organization's system development life cycle methodology should require that the business requirements for the existing system and the proposed new or modified system (software, data, and infrastructure) are clearly defined before a development, implementation, or modification project is approved. The system development life cycle methodology should specify the solution's functional and operational requirements, including perfor-mance, safety, reliability, compatibility, security, and legislation.
1.2 Formulation of Alternative Courses of Action	The organization's system development life cycle should stipulate that alternative courses of action should be analyzed to satisfy the business requirements established for a proposed new or modified system.
1.3 Formulation of Acquisition Strategy	Information systems acquisition, development, and maintenance should be considered in the context of the organization's IT long- and short-range plans. The organization's system development life cycle methodology should provide for a software acquisition strategy plan

	defining whether the software will be acquired off-the-shelf; developed internally, through contract, or by enhancing the existing software; or developed through a combination of these.
1.4 Third-Party Service Requirements	The organization's system development life cycle methodology should require an evaluation of the requirements and specifications for an RFP (request for proposal) when dealing with a third-party ser-vice vendor.
1.5 Technological Feasibility Study	The organization's system development life cycle methodology should require an examination of the technological feasibility of each alternative for satisfying the business requirements established for the development of a proposed new or modified information system project.
1.6 Economic Feasibility Study	In each proposed information systems development, implementation, and modification project, the organization's system development life cycle methodology should require an analysis of the costs and benefits associated with each alternative being considered for satisfying the established business requirements.
1.7 Information Architecture	Management should ensure that attention is paid to the enterprise data model while solutions are being identified and analyzed for feasibility.
1.8 Risk Analysis Report	In each proposed information system development, implementation, or modification project, the organization's system development life cycle methodology should require an analysis and documentation of the security threats, potential vulnerabilities and impacts, and the feasible security and internal control safeguards for reducing or eliminating the identified risk. This should be realized in line with the overall risk-assessment framework.
1.9 Cost-Effective Security Controls	Management should ensure that the costs and benefits of security are carefully examined in monetary and nonmonetary terms, to guarantee that the costs of controls do not exceed benefits. The decision requires formal management sign-off. All security requirements should be identified at the requirements phase of a project and should be justified, agreed to, and documented as part of the overall business case for an information system. Security requirements for business continuity management should be defined to ensure that the proposed solution supports the planned activation, fallback, and resumption processes.

1.10 Audit Trails Design	The organization's system development life cycle methodology should state that adequate mechanisms for audit trails must be available or developed for the solution identified and selected. The mechanisms should provide the capability to protect sensitive data (for example, user IDs) against discovery and misuse.
1.11 Ergonomics	Management should ensure that the IT function adheres to a standard procedure for identifying all potential system software programs, to satisfy its operational requirements.
1.13 Procurement Control	Management should develop and implement a central procurement approach describing a common set of procedures and standards to be followed in the procurement of information technology–related hardware, software, and services. Products should be reviewed and tested before their use and the financial settlement.
1.14 Software Product Acquisition	Software product acquisition should follow the organization's procurement policies.
1.15 Third-Party Software Maintenance	Management should require that before licensed software is acquired from third-party providers, the providers have appropriate procedures to validate, protect, and maintain the software product's integrity rights. Consideration should be given to the support of the product in any maintenance agreement related to the delivered product.
1.16 Contract Application Programming	The organization's system development life cycle methodology should require that the procurement of contract programming services be justified with a written request for services from a designated member of the IT function. The contract should stipulate that the software, documentation, and other deliverables are subject to testing and review before acceptance. In addition, it should require that the end products of completed contract programming services be tested and reviewed according to the related standards by the IT function's quality assurance group and other concerned parties (such as users and project managers) before payment for the work and approval of the end product. Testing to be included in contract specifications should consist of system testing, integration testing, hardware and component testing, procedure testing, load and stress testing, tuning and performance testing, regression testing, user acceptance testing, and, finally, pilot testing of the total system, to avoid any unexpected system failure.

1.17 Acceptance of Facilities	Management should ensure that an acceptance plan for facilities to be provided is agreed upon with the supplier in the contract. This plan should define the acceptance procedures and criteria. In addition, acceptance tests should be performed to guarantee that the accommodation and environment meet the requirements specified in the contract.
1.18 Acceptance of Technology	Management should ensure that an acceptance plan for specific technology to be provided is agreed upon with the supplier in the contract. This plan should define the acceptance procedures and criteria. In addition, acceptance tests provided for in the plan should include inspection, functionality tests, and workload trials.

The selection of computer hardware requires the organization to define specifications for outside vendors. These specifications should be used in evaluating vendor-proposed solutions. This specification is sometimes called an *invitation to tender* (ITT) or a *request for proposal* (RFP).

Per ISACA, the portion of the ITT pertaining to hardware should include the following:

- Information-processing requirements
    - Major existing application systems and future application systems
    - Workload and performance requirements
    - Processing approaches (online/batch, client/server, real-time databases, continuous operation)
- Hardware requirements
    - CPU speed
    - Peripheral devices (sequential devices, such as tape drives; direct-access devices, such as magnetic disk drives, printers, CD-ROM drives, and WORM drives)
    - Data-preparation/input devices that accept and convert data for machine processing
    - Direct-entry devices (terminal, point-of-sale terminals, or automated teller machines)
    - Networking capability (Ethernet connections, modems, and ISDN connections)
- System software applications
    - Operation systems software (current version and any required upgrades)

    - Compilers
    - Program library software
    - Database-management software and programs
    - Communication software
    - Access-control software
- Support requirements
    - System maintenance (for preventative, detective [fault reporting], or corrective purposes)
    - Training (user and technical staff)
    - Backups (daily and disaster)
- Adaptability requirements
    - Hardware/software upgrade capabilities
    - Compatibility with existing hardware/software platforms
    - Changeover to other equipment capabilities
- Constraints
    - Existing hardware capacity
    - Deliver dates
- Conversion requirements
    - Test time for the hardware/software
    - System-conversion facilities
    - Cost/pricing schedule

The acquisition of hardware might be driven by requirements for a new software acquisition, the expansion of existing capabilities, or the scheduled replacement of obsolete hardware. With all these events, senior management must ensure that the acquisition is mapped directly to the strategic goals of the organization. The IT

steering committee should guide information systems strategy—and, therefore, that its acquisitions—align with the organization's goals.

In addition, the senior managers of the IT steering committee should receive regular status updates on acquisition projects in progress, the cost of projects, and issues that impact the critical path of those projects. The IT steering committee is responsible for reviewing issues such as new and ongoing projects, major equipment acquisitions, and the review and approval of budget; however, the committee does not usually get involved in the day-to-day operations of the IS department.

The IT organization should have established policies for all phases of the system development life cycle (SDLC) that controls the acquisition, implementation, maintenance, and disposition of information systems. The SDLC should include computer hardware, network devices, communications systems, operating systems, application software, and data. These systems support mission-critical business functions and should maximize the organization’s return on investment. The combination of a solid governance framework and defined acquisition process creates a control infrastructure that reduces risk and ensures that IT infrastructure supports the business functions.

As an IS auditor, you will look for evidence of a structured approach to hardware acquisition, implementation, and maintenance. These include written acquisition policies and outline the process for feasibility studies, requirements gathering, and the approval process of the IT steering committee. After hardware is procured, the IT organization must have a defined project-management and change-control process to implement the hardware. All hardware acquired must fall under existing maintenance contracts and procedures, or contracts must be acquired and procedures updated to reflect the new hardware. The hardware should be tested according to written test plans before going into production, and the hardware should be assigned to the appropriate functional areas (such as systems administration) to ensure that production responsibility is clearly defined. The acquired hardware, whether a replacement or new to the IT infrastructure, should be secured (physical, logical) and added to the business continuity plan.

Change Control and Configuration Management Principles for Hardware

The change-control and configuration-management processes detail the formal documented procedures for introducing technology changes into the environment. More specifically, *change control* ensures that changes are documented, approved, and implemented with minimum disruption to the production environment and maximum benefits to the organization.

During the normal operation of the IT infrastructure, there will be changes to hardware and software because of normal maintenance, upgrades, security patches, and changes in network configurations. All changes within the infrastructure need to be documented and must follow change control procedures. In the planning stages the party responsible for the changes (such as end users, line managers or the network administrator) should develop a

change-control request. The request should include all systems affected by the change, the length of resources required to implement the change (time and money), and a detailed plan. The plan should include what specific steps will be taken for the change and should include test plans and back-out procedures, in case the change adversely affects the infrastructure. This request should go before the change-control board that votes on the change and normally provides a maintenance window in which the change is to be implemented. When the change is complete and tested, all documentation and procedures that are affected by the change should be updated. The change-control board should maintain a copy of the change request and its review of the implementation of the change.

What is Software Testing? Introduction, Definition, Basics & Types –

What is Software Testing?

What is Software Testing?

SOFTWARE TESTING is defined as an activity to check whether the actual results match the expected results and to ensure that the software system is Defect free. It involves execution of a software component or system component to evaluate one or more properties of interest.Software testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements. It can be either done manually or using automated tools. Some prefer saying Software testing as a White Box and Black Box Testing.

Why is Software Testing Important?

Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss, and history is full of such examples.

- In April 2015, Bloomberg terminal in London crashed due to software glitch affected more than 300,000 traders on financial markets. It forced the government to postpone a 3bn pound debt sale.
- Nissan cars have to recall over 1 million cars from the market due to software failure in the airbag sensory detectors. There has been reported two accident due to this software failure.
- Starbucks was forced to close about 60 percent of stores in the U.S and Canada due to software failure in its POS system. At one point store served coffee for free as they unable to process the transaction.
- Some of the Amazon's third party retailers saw their product price is reduced to 1p due to a software glitch. They were left with heavy losses.
- Vulnerability in Window 10. This bug enables users to escape from security sandboxes through a flaw in the win32k system.
- In 2015 fighter plane F-35 fell victim to a software bug, making it unable to detect targets correctly.
- China Airlines Airbus A300 crashed due to a software bug on April 26, 1994, killing 264 innocent live

- In 1985, Canada's Therac-25 radiation therapy machine malfunctioned due to software bug and delivered lethal radiation doses to patients, leaving 3 people dead and critically injuring 3 others.
- In April of 1999, a software bug caused the failure of a $1.2 billion military satellite launch, the costliest accident in history
- In may of 1996, a software bug caused the bank accounts of 823 customers of a major U.S. bank to be credited with 920 million US dollars.

Types of Software Testing

Typically Testing is classified into three categories.

- Functional Testing
- Non-Functional Testing or Performance Testing
- Maintenance (Regression and Maintenance)

Testing Category	Types of Testing
Functional Testing	• Unit Testing • Integration Testing • Smoke • UAT (User Acceptance Testing) • Localization • Globalization • Interoperability • So on
Non-Functional Testing	• Performance • Endurance • Load • Volume

	• Scalability • Usability • So on
Maintenance	• Regression • Maintenance

This is not the complete list as there are more than 150 types of testing types and still adding. Also, note that not all testing types are applicable to all projects but depend on the nature & scope of the project.

- Next

YOU MIGHT LIKE:

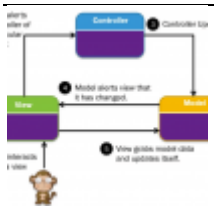
SOFTWARE TESTING

What is Spike Testing? Learn With Example

What is Spike Testing? SPIKE TESTING is defined as a type of performance testing in which...

Read more

SDLC



MVC Tutorial for Beginners: What is, Architecture & Example

What is MVC Framework? The Model-View-Controller (MVC) framework is an architectural pattern that...

Read more

SOFTWARE TESTING

15 BEST Test Data Generation Tools in 2020

Test data generation is the process of making sample test data used in executing test cases. There are...

Read more

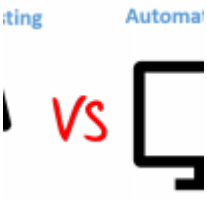
AGILE TESTING

Scrum Vs. Kanban: Know the Difference

What is Scrum? Scrum is an agile process that helps to deliver the business value in the shortest time....

Read more

SOFTWARE TESTING



Automation Testing Vs. Manual Testing: What's the Difference?

What is Manual Testing? Manual testing is testing of the software where tests are executed...

Read more

SDLC

10 Best Programming Language to Learn in 2020

With time old programming languages become obsolete while new programming languages are launched,...

Read more

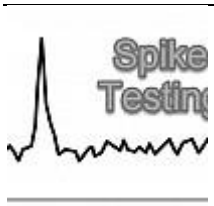
Testing Tutorials

-

- Next

YOU MIGHT LIKE:

SOFTWARE TESTING

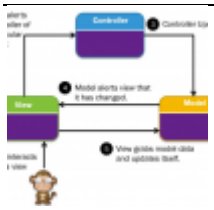


What is Spike Testing? Learn With Example

What is Spike Testing? SPIKE TESTING is defined as a type of performance testing in which...

Read more

SDLC



MVC Tutorial for Beginners: What is, Architecture & Example

What is MVC Framework? The Model-View-Controller (MVC) framework is an architectural pattern that...

Read more

SOFTWARE TESTING

15 BEST Test Data Generation Tools in 2020

Test data generation is the process of making sample test data used in executing test cases. There are...

Read more

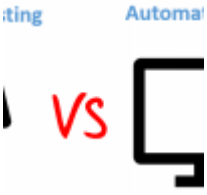
AGILE TESTING

Scrum Vs. Kanban: Know the Difference

What is Scrum? Scrum is an agile process that helps to deliver the business value in the shortest time....

Read more

SOFTWARE TESTING



Automation Testing Vs. Manual Testing: What's the Difference?

What is Manual Testing? Manual testing is testing of the software where tests are executed...

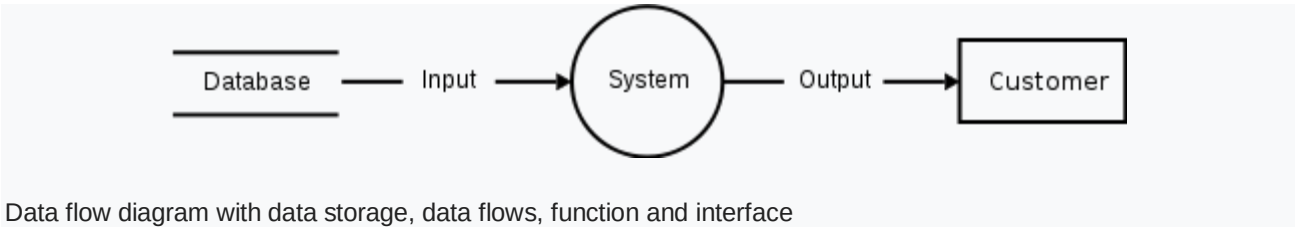
Read more

SDLC

10 Best Programming Language to Learn in 2020

With time old programming languages become obsolete while new programming languages are launched,...

Data-flow diagram -



Data flow diagram with data storage, data flows, function and interface

A **data-flow diagram** (DFD) is a way of representing a flow of a data of a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. Specific operations based on the data can be represented by a flowchart.[1]

Data Dictionary

Definition - What does *Data Dictionary* mean?

A data dictionary is a file or a set of files that contains a database's metadata. The data dictionary contains records about other objects in the database, such as data ownership, data relationships to other objects, and other data.

The data dictionary is a crucial component of any relational database. Ironically, because of its importance, it is invisible to most database users. Typically, only database administrators interact with the data dictionary.