Formalizing Business Requirements into Functional Specifications

CERTIFICATION OBJECTIVES

4.01 Functional Specifications
4.02 Transforming Requirements into Functional Specifications
4.03 Transforming Functional Specifications into Technical Specifications
✓ Two-Minute Drill
Q&A Self Test
In this chapter, you will extend the business requirements into the next phase—the creation of functional specifications. After the initial requirements have been created, functional specifications are created to determine what functionality the final solutions will have. This creation of the functional specifications allows for the further definition and restating of the requirements. Additionally, this process divides the requirements into groups of responsibility.

Next is the development of the technical specifications document. This document details the specifics about how the solution will be created. The design goals of performance, scalability, availability, deployment, security, and maintainability are addressed in this document. A development strategy is decided upon for handling auditing, error handling, integration, localization, and state management. Also in this document is the selection of how to deploy the application and the different strategies available.

After deployment, the operations strategy examines how the solution will be transitioned to the operations team and how the solution will be supported in the future. Included in this plan is how to maintain the data archiving of the information and upgrade strategies.

**CERTIFICATION OBJECTIVE 4.01**

**Functional Specifications**

The transformation of requirements into functional specifications is the shift from problem definition to solution design. The functional specifications are a repository of documents. This document defines what will be built, how it will be done, and when it will be completed. The functional specifications consist of a summary document that physically describes the functional specifications and lists artifacts that make up the specifications. These collections of documents are considered “living documents,” meaning that they will change throughout the project cycle. The functional specifications document is a record of the decisions and agreements made regarding the functionality of the solution, design goals, and priorities.
Artifacts can include Unified Modeling Language (UML) models such as use case diagrams, usage scenarios, initial requirements, initial features, and various other models. The artifacts from the conceptual, logical, and physical design stages can be in electronic form or stored in formats of various tools. The manifest or summary can exist as an electronic document, such as a Microsoft Word document or Microsoft PowerPoint presentation. The functional specifications are a joint effort by the team.

The functional specifications are a virtual collection of documents.

Goals

There are four goals of the functional specifications. First is to consolidate a common understanding of the business and user requirements. The features of a solution are determined from the business and user requirements. The number of requirements will vary based upon the size of the project. The functional specifications help the customer and the project team agree on a common understanding of the requirements of the solution.

The second goal is to break down the problem and modularize the solution logically. It is important that the team identify the entire problem clearly. This is achieved by breaking the solution into distinct, unambiguous parts. The functional specifications help simplify the solution into logical parts and document them. This segmentation helps identify if design changes need to be made early in the process. Catching these errors now is less risky and less expensive than finding them later.

The third goal is to provide a framework to plan, schedule, and build the solution. The specifications provide information for the team to create tasks and cost estimates and budgets for the project. The program manager can create estimates for resources and time that the project will require. The other purpose is for the testing team to create test cases and test scenarios early in the process. The release management team uses the functional specifications for deployment and to support the development and test environments.

Lastly, the functional specifications serve as a contract between the team and the customer for what will be delivered. This is evidence of what is to be developed and delivered. In some organizations, this is written as a contract between the team and the customer. It is not necessarily a legal document, but can serve as one. This document can be used by a third-party team or groups as an addendum to a project work order.
Chapter 4: Formalizing Business Requirements into Functional Specifications

Risks of Skipping Functional Specifications

Sometimes the team will choose to skip the functional specifications and continue to development. Budget or time constraints may interfere with creating the functional specifications. The risks associated with skipping the functional specifications are:

- The team may develop a solution that does not completely address the customer requirements.
- The team might be unable to clearly define customer expectations and share a common understanding with the customer. Consequently, the team might not know whether they are developing the required solution.
- The team might not have enough detail to validate and verify that the solution meets customer expectations.
- Estimates of budget and schedules cannot be accurately created for the project.

These risk factors are very important and may not be presented as risks of skipping the functional specifications.

Elements of Functional Specifications

The following table describes the possible elements of functional specifications. These elements can be separate documents.

<table>
<thead>
<tr>
<th>Element</th>
<th>Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual design summary</td>
<td>Use cases, usage scenarios, context models such as screen shots</td>
</tr>
<tr>
<td>Logical design summary</td>
<td>Task and task sequence models, logical object and service models, conceptual models of the proposed solution, UI screen flows, logical database model</td>
</tr>
<tr>
<td>Physical design summary</td>
<td>Component packaging, component distribution topology, technology usage guidelines, infrastructure architecture and design, description of UI screens and physical database model</td>
</tr>
<tr>
<td>Standards and processes</td>
<td>Security, packaging, maintainability, supportability, stabilization, deployment</td>
</tr>
</tbody>
</table>
The conceptual design summary provides information on the conceptual design and provides information such as the solution overview and solution architecture. The logical design summary includes information such as users, objects, and attributes. The logical design is the next step in the evolution of the solution. The logical design takes the high-level view generated from the conceptual design and divides the information into different layers. The physical design summary provides a summary of the physical infrastructure design of the solution. The standards and processes serve as a guideline for performing various tasks for the project. In addition, this section includes details about the quality and performance metrics that will be used.

CERTIFICATION OBJECTIVE 4.02

Transforming Requirements into Functional Specifications

The conceptual design summary provides information on the conceptual design and provides information such as the solution overview and solution architecture. The logical design summary includes information such as users, objects, and attributes. The logical design is the next step in the evolution of the solution. The logical design takes the high-level view generated from the conceptual design and divides the information into different layers. The physical design summary provides a summary of the physical infrastructure design of the solution. The standards and processes serve as a guideline for performing various tasks for the project. In addition, this section includes details about the quality and performance metrics that will be used.

CERTIFICATION OBJECTIVE 4.02

Transforming Requirements into Functional Specifications

After gathering detailed information about business and user requirements and business processes, the team can now proceed to the analysis of the information. This is the analysis of the created documents in the envisioning phase where the initial or candidate requirements are created. There are two purposes of the analysis. The first is to review user and business processes and activities. The second purpose is to document and model the context, workflow, task sequence, and environmental relationships of the business.

The transformation is performed in the following steps:

1. Synthesizing information
2. Refining use case diagrams
3. Selecting an appropriate application architecture for the solution
4. Creating a conceptual model of the application
Synthesizing Information

The synthesizing of information is the process of assimilating gathered data and interpreting the results. The team transforms the gathered data into meaningful information, by performing the following tasks:

- Identifying discrete pieces of information about what the user said and did.
- Recording the detailed flow of the tasks that the user performed.
- Identifying tools and pieces of information that were used.
- Identifying exceptions and alternatives that occur while the user performs the task.
- Modeling the relationship between business process, business systems, and users.
- Modeling the current environment in which the user works and any possible changes to that environment that might be required.

The synthesis of information is an important skill for taking the architecture exam. You will have to take the information provided and sift through the information using these steps.

The deliverable for this step includes information models and current usage scenarios. The information models include the relationship between the business process, business system and users, workflow process, and task sequence. Also included are updated user profiles, candidate requirements, and detailed use case scenarios.

Restating Requirements

The candidate requirements at this point are groups put together that are now in need of further organization. This process is the restating of requirements. Restated requirements meet the following criteria: well-defined, concise, testable, and organized in a hierarchy of related requirements. A well-defined requirement is a complete sentence and typically, uses “will,” “may,” “must,” or “should” statements. Each requirement must address one item only. Each requirement should have specific inputs resulting in known outputs. Related items are grouped together to form feature sets. Requirements should be written in the language of the business. The following table is an example of restated requirements.
Requirement IDs are for tracking requirements from creation to implementation.

Categorize Requirements

After restating the requirements, the next step is to categorize them into user, system, operations, and business requirements. User requirements define the non-functional aspect of the user’s interaction with the solution. They help you determine the user interface and performance expectations of the solution in terms of its reliability, availability, and accessibility. A successful solution has completed all requirements and has passed user usability. Also identified in this step is the training required for users to use the solution. An example requirement would be that the user should be able to create an online resume in ten minutes. Another would be that the user should be able to complete a search with results returned within a minute.

Categorizing requirements into groups allows you to focus on finding the requirements you need to focus on for the scenario questions.

System requirements specify the transactions that have been broken down to their simplest state and their sequence in the system. This helps the project team define how the new solution will interact with the existing systems. Critical dependencies are identified and managed by the team.

An example requirement would be for the solution to not require internal users to provide additional credentials other than the logged credentials from the corporate network.

Operations requirements describe what the solution must deliver to maximize operability and improve service delivery with reduced downtime and risks. Key concepts addressed are security, availability, reliability, manageability, scalability, and supportability. An example requirement would be that the site design should include a system for managing the total system throughput and response time within the stated service levels.
Chapter 4: Formalizing Business Requirements into Functional Specifications

Business requirements describe the organization’s needs and expectations for the solution. They define what the solution must deliver in order to capitalize on a business opportunity or to meet business challenges. Requirements are identified by considering the organization as a valid entity with its own set of needs from the solution. These requirements are generally high-level requirements and provide the context in which the solution will operate. An example requirement might be that the solution must be able to interact and communicate with other business processes, applications, and data sources.

Refining Use Case Diagrams

During the envisioning state, the use case diagrams specified high-level diagrams for the organization. The purpose was to present all of the use cases available. The purpose of refining the use case diagrams is to create use cases within the scope of the solution. This is performed by the following four tasks.

First, create subordinate use cases. Each use case is revisited within the scope of the solution. Each task associated with the use case is identified and modeled as subordinate use cases. All of the actors that perform the tasks are identified, and the relationship is identified between the various tasks and actors.

Second, create usage scenarios for each subordinate use case. This takes the original usage scenario and adds detailed information to it. Added to the usage scenario are the following items: a detailed scenario narrative, specifics of the basic course, specifics for an alternative course, and a description the preconditions and postconditions.

Third, validate each use case and usage scenario against the original artifacts and users. This step helps determine whether any steps in the process have not been documented. The features list is then developed based on the requirements. Revisions to the feature list are identified by elaborating on the use cases, and any additions are validated by the customer.

Remember to validate your use cases against the requirements and user responses.

Finally, refine the requirements with the validated use cases and usage scenario information. Because the validation is an iterative process, the requirements need to be checked to make sure that they are valid or need redefining.
Selecting the Appropriate Architecture

The selection of the architecture depends on the understanding of the services that the solution must provide. A service is defined as a unit of application logic that includes methods for implementing an operation, a function, or a transformation. Services can be either simple or complex. For example, services creating, reading, updating, and deleting information are simple. More information concerning services and application architecture will be covered in Chapters 6, 7, and 8.

Creating a Conceptual Model

The final step is the optimization and creation of the conceptual model. This process is the evolving of the solution into its final form. The optimization examines looking at the future state of the solution and examines the current state scenarios. More about this and the conceptual model will be covered in Chapter 5.

CERTIFICATION OBJECTIVE 4.03

Transforming Functional Specifications into Technical Specifications

The technical specifications are a set of reference documents that in the development phase is used to determine the scope of work and define development tasks. These documents may consist of artifacts that detail the class specifications, component models, metrics, and network topologies. Also included are interface definitions, configuration files, dynamic link library (DLL) and assembly names, strong names keys, and deployment elements.

This is a living document that is completed after the conceptual, logical, and physical design phases are complete. It is updated during the development process as components are completed. The sections that would be included in the technical specifications document include items listed in Table 4-1.

During the planning phase, the following design goals should be considered: scalability, availability, reliability, performance, interoperability, and localization. These elements affect the design of the solution because some of the elements will
be required, and others will be based on resources available. Other considerations are
based upon the available technologies, knowledge, and skills of the development and
support staff.

Scalability

Scalability is defined as the capability to increase resources to produce an increase in
the service capacity. This means that a solution can add resources to handle additional
demands without modifying the solution itself. A scalable solution must balance
between software and hardware used to implement an application. The adding of
resources is supposed to create a positive benefit, but it is possible to create negative
results and show no increase in capacity or even cause decreases. For example, an
application that implements load balancing, which is allowing the application to
balance the load between multiple servers, can have a minimal gain if lengthy datasets
are generated in response to the user’s request.

Approach

There are two common approaches to scalability. The first is called scaling up. This
refers to achieving scalability by improving the user’s processing hardware. This translates
into adding more memory, processors or migrating to larger servers. The primary
goal is to increase hardware resources. However, there can be a maximum level achieved where the capacity makes no change. Typically, you can scale up without making application changes to source code.

The second approach is called \textit{scaling out}. This refers to the distributing of the processing load across multiple servers. The collection of computers continues to act as the original configuration. The application should be able to execute without needing information about the server where it is located. This concept is called \textit{location transparency}. Scaling out is a way to increase the fault tolerance of the application.

\textbf{Guidelines}

To design for scalability, use the following guidelines:

- Design processes so that they do not wait. Processes should not wait longer than necessary. A process can be categorized as synchronous or asynchronous. A synchronous process waits for another process to complete before it continues. This is also known as a blocking call.

  A synchronous process can encounter bottlenecks for resources. These bottlenecks can affect the performance and scalability of the application. The solution to this problem is creating asynchronous processes. This is also called a non-blocking call.

  An asynchronous process spawns multiple processes that can finish at different times. Long running operations can be queued for completion later by other processes.

- Design processes so that they do not compete for resources. The biggest causes of problems involve competition for resources. This condition is also known as a race condition. The resource could be memory, processing time, bandwidth, or database connections. There are two ways to handle this problem; one is by first sequencing resource usage to the most available resources and then to the least available resources last. The second option is to acquire resources as late as possible.

- Design processes for commutability. Commutative operations are multiple processes that execute in any order and still obtain the same result. An example of this would be an operation that does not involve transactions. A busy site could create transaction tables that are updated periodically.
by the transaction records, allowing the database to reduce record locks. As shown in the following illustration, an example of a commutative operation would be as follows: when a product is ordered, instead of directly updating the Products table, the amount ordered is entered into the ProductTransactions table. Another process at predetermined time intervals will update the Products table with the appropriate values. This allows the application to process without having to lock the records when an update needs to be performed when a product is ordered. The locking for the transaction will happen less frequently in this scenario.

![Diagram of database tables](image)

- Design components for interchangeability. An interchangeable component is designed to release its resources, move into a pool managed by a resource manager, and to be reusable by a new client. Pooling, for Open Database Connectivity (ODBC), allows a set amount of database connections to be available when needed. This eliminates the time and cost required to create and destroy these objects after every use. These types of pools are controlled by resource managers. Resource managers are the controlling mechanisms for specific types of resources. These types of components are designed to be stateless—in other words, they do not need to save information between calls to the component.

- An example of interchangeable components is a component designed for COM+ component pooling or using Open Database Connectivity (ODBC) connection pooling. With these types of object pooling, you can set the minimum and maximum pool size, and create timeout settings.
Partition resources and activities help prevent bottlenecks in the system. Partitioning activities can also ease the demand on resources such as processors and bandwidth. An example would be using Secure Sockets Layer (SSL) to provide a secure connection. SSL requires a high amount of overhead, and because of this you might decide to only provide SSL support for pages that require high security. This would allow you to create separate servers for handling SSL sessions. The drawback is that the system would be more complex and might require significant overhead for operating. In this situation, partitioning could be created by using many small components instead of using a few large components and by limiting the amount of cross-device communication.

Availability

Availability is a measure of how often the application is available to handle service requests as compared to the planned runtime. For some applications, high availability means that the application needs to work 24 hours a day, 7 days a week. Availability takes into account repair time because an application that is being repaired is not available for use. Advances in hardware and software have increased the quality of high-availability applications.

*Availability does not address business continuation issues such as backups and alternative sites.*

The following table shows the measurements used for calculating availability.

<table>
<thead>
<tr>
<th>Name</th>
<th>Calculation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Time Between Failure (MTBF)</td>
<td>Hours/failure count</td>
<td>Average length of time the application runs before failing</td>
</tr>
<tr>
<td>Mean Time To Recovery (MTTR)</td>
<td>Repair hours/failure count</td>
<td>Average length of time needed to repair and restore service after a failure</td>
</tr>
</tbody>
</table>

The formula for calculating availability is:

\[
\text{Availability} = \left( \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}} \right) \times 100
\]
Chapter 4: Formalizing Business Requirements into Functional Specifications

An example of this would be a 24/7 web site that has two errors a week that each require an hour to fix. The resulting calculation would look like the following, based on a year’s time:

\[
\left(\frac{8736}{104}\right) \div \left(\frac{8736}{104} + 0.5\right) \times 100 = 99.4\%
\]

A common way to describe availability is by using 99.*. You will notice from our example the application is down for 104 hours during the year, but the percentage is still high, so these numbers can be deceiving. To determine the level of availability that is appropriate for your application, you need to answer these questions:

- Who are the customers of the application? What are their expectations from the application?
- How much downtime is acceptable?
- Do internal company processes depend on the service? An example would be to perform data maintenance during the hours of 2:00 A.M. and 4:00 A.M. because the user population is not using the application and the impact is minimized.
- What are the schedule and budget for developing the application?

Designing for availability is intended to prevent and detect errors before they happen. Availability can be ensured by providing multiple routes to application processes and data. Use only tested and proven processes that support the application throughout the solution life cycle. Some techniques used for designing for availability include:

- Reduce planned downtime by using rolling upgrades. Rolling upgrades refers to taking a specific server down in a cluster, updating the component, and bringing the server back online. A cluster consists of multiple computers that are physically networked and logically connected using cluster software. The other server during this time is taking the workload without the user experiencing any downtime. This is usually used for applications that scale out.

- Reduce unplanned downtime with clustering. Clustering is technology for creating high-availability applications. Clustering allows a multiple server web site to withstand failures with no interruption in service. This allows one of the machines in a cluster to be taken down for repairs and the other machines in the cluster will take its responsibilities. This can cause a slight delay for the customers connected. Cluster software can provide this functionality if the
service and application have been designed to be cluster-aware and assigned to a cluster.

- Use network load balancing. Network load balancing (NLB) is used to distribute traffic evenly across available servers, such as in web farms. The farm has multiple web servers and NLB allows balancing the load between the web servers, so if one is busy the next one in the list is used instead. If one of the machines needs to be taken down for maintenance, other machines can be assigned to the farm to prevent an interruption of service.

- Use a redundant array of independent disks (RAID) for data stores. RAID uses multiple hard drives to store data in multiple places. If a disk fails, the application is transferred to a mirrored image and the application continues running. The failed disk can be replaced or recovered without stopping the application, which defines RAID 0. Another type of RAID is RAID 5, which is like RAID 0 except it is striping with parity. The following illustration demonstrates how RAID 5 is stored. For high performance database systems a combination of RAID 0 and RAID1 can be used, known as RAID 10, which combines striping and mirroring.

- Isolate mission-critical applications. This is an application that consumes many resources and is of vital importance to the organization. Each of the resources that the mission-critical application requests can affect the performance and availability of applications sharing the same resources. It is recommended that mission-critical applications use dedicated infrastructures and networks to prevent resource contentions between resource intensive applications.
Chapter 4: Formalizing Business Requirements into Functional Specifications

- Use queuing. Queuing enables your application to communicate with other applications by sending and receiving asynchronous messages. Queuing guarantees message delivery, regardless of connectivity that exists. Queuing allows an application to manage peak workloads that require a lot of hardware. More routes set up for the messages allows for successful and immediate message completion.

Reliability

Reliability of an application refers to the ability of the application to provide accurate results. Reliability and availability are very closely related. While availability measures the capacity to handle all requests, reliability measures how long the application can execute and produce expected results without failing. Unreliable systems are difficult to maintain or improve because the failure points are typically hidden throughout the system.

Reliability of the application is dependent upon the availability of individual components, because all of the systems are related. Application failures can occur for many reasons:

- Erroneous code
- Inadequate testing
- Change management problems
- Operations errors
- Lack of ongoing monitoring and analysis
- Lack of quality software engineering processes
- Interaction with external services or applications
- Changing operation conditions, such as usage level or workload changes
- Unusual events, such as security failures and broadcast storms
- Hardware failures, such as disk, controllers, and so on
- Environmental problems (power, cooling, fire, flood, or natural disasters)

When considering the design for reliability of a solution, you have to examine the expected usage of the solution. Create a profile for the expected usage and design
the solution from the profile. This profile should include how particular services are provided, an evaluation of failure scenarios, and designs for preferred alternatives. The profiles should consider the application's interactions with other applications, as well.

It is difficult to determine reliability problems for systems that have not been developed, but analyzing currently running applications in the organization can give clues to issues. Such analysis reveals the failure frequency and distribution, root causes and possible improvements for existing systems. This information can be used to create a reliable solution.

Creating a high-reliability application depends on the development and testing during the development and stabilization stages of the life cycle. A reliable solution ensures that the data input is correct and expected results happen consistently. The following tasks can help you create a reliable application:

- Putting reliability requirements in the specifications.
- Using a good architectural infrastructure.
- Including management information in the application.
- Using redundancy.
- Using quality development tools.
- Using reliability checks that are provided by the application.
- Implementing error handling.
- Implementing graceful degradation. Graceful degradation refers to the process of adding functionality to your application to enable compatibility with earlier technologies, mainly browsers.

Performance

Performance is defined by metrics such as transaction throughput and resource utilization. Performance is typically defined in terms of response times for users. To determine your performance goals, you should answer the following questions:

- What is the business goal? An example of this would be the solution should handle more orders each week, with an expected increase in revenue. This can then be converted into a performance goal for each functional area.
What is the critical functionality of the solution? Identifying critical features allows you to prioritize the system design. You might decide to degrade the performance of a low priority feature to maintain or increase the performance of a higher priority feature.

What are the features required by different sets of users? Because of varying expectations, the performance requirements of the application can differ. You need to determine the relationship between each functional area and performance goal. For example, in a solution that uses database stores, the organization expects the solution to store valid data quickly. This converts into a performance goal of the solution that the database has fast inserts and updates to the database. Creating profiles helps in partitioning and developing accurate tests for the solution.

For testing, performance goals must be expressed in a way that is measurable in your testing routines.

Performance requirements must be defined before the team proceeds to the developing phase. Good performance requirements must identify project constraints, determine services that the application will perform, and specify the load on the application.

Identifying constraints Constraints in the project include budget, schedule, infrastructure, and the choice of development tools or technologies. You need to design an application so that it meets its performance goals within the limitations of the constraints. For example, a processor intensive application might not be able to be designed if the user’s hardware cannot support the application. Instead of changing some aspects of a project to improve performance, you can modify aspects of the project to improve performance.

Determining features The features of an application correspond to use cases and usage scenarios. For each usage scenario that affects the performance of the application, specify what the user does and how the system responds. This includes how databases and other system services are accessed. This information can help create tests for measuring performance that resemble actual usage of the application as closely as possible.
Specifying the load You can specify the load of the application as the number of clients that will use the application. In addition, examine how the load might vary over time. You can use the load to define the performance metrics of the application.

Selecting a Development Strategy

Before the development phase begins, it is important to verify the development and test environments. The test environment should ideally mirror the production environment. It is important to maintain separation between the production environment and the development and test environments to prevent occurrences in development and testing from affecting live production systems.

The development plan describes the solution development process. It identifies the tasks necessary to create and assemble the components of the solutions. This complements the functional specifications that provide the technical details of what will be developed. The plan also provides consistent guidelines and processes to the teams creating the solution. Some of the key sections of the development plan are:

- **Development objectives** Defines the primary drivers that were used to create the development approach and the key objectives of that approach.
- **Overall delivery strategy** Describes the overall approach to delivering the solution.
- **Tradeoff approach** Defines the approach for making design and implementation of tradeoff decisions.
- **Key design goals** Identifies the key design goals and the priority of each goal.
- **Development and build environment** Describes the development and build environment and how it will be managed.
- **Guidelines and standards** Lists and provides references to all standards and guidelines to be used for the project.
- **Versioning and source control** Describes how versioning and source control will be managed.
Chapter 4: Formalizing Business Requirements into Functional Specifications

- **Build process** Describes the incremental and iterative approach for developing code and for builds of hardware and software components.
- **Components** Provides a high-level description of the set of solution components and how they will be developed.
- **Configuration and development management tools** Identifies all the development tools the team will use during the project.
- **Design patterns** Identifies the design patterns or templates that the team will use for this project and their sources.
- **Development team training** Identifies the training necessary to ensure that the development team will successfully develop the solution.
- **Development team support** Identifies the various types of support the development team will require, the sources of that support, the amount of support of each type that the team will require, and the estimated schedule for support.

The development plan includes strategies for auditing and logging, error handling, integration, globalization, localization, data storage, and state management. These priorities are included in various aspects of the sections of the development plan. The validation of the development plan is to make sure that all of the business rules for the solution are achieved.

**Select Strategies for Auditing and Logging**

Application monitoring is used to ensure that the application is functioning correctly and performing at an optimal level. Automated monitoring enables identification of failure conditions and potential problems. The auditing of an application is typically the responsibility of administrators within the operations team, and the operations team must establish guidelines and procedures for application monitoring. Communicating these procedures to the development team allows both teams to work together to log and monitor information that can assist problem discovery and diagnosis.

Error logging is closely related to monitoring and is a development function. The development team must communicate with the operations team to inform them of the types of error logs generated by the application. Together, both teams must decide on the appropriate logging mechanisms, then develop and monitor applications accordingly.
Understanding the use of error logging and performance monitors as an important method of collecting information.

A monitoring plan defines the processes by which the operational environment will monitor the solution. It describes what will be monitored, how it will be monitored, and how the results of monitoring will be reported and used. Once the details of the monitoring process are completed, they will be incorporated into the functional specifications and then included in the solution design. Some key sections of the monitoring plan are:

- Resource monitoring identifies scarce resources that need monitoring and determining of thresholds.
- Performance monitoring defines the metrics to be gathered for the performance evaluation of components in the solution.
- Trend analysis is the examination of the data to determine how parts of the system are used under various situations.
- Detecting failures describes how the development, operations, and maintenance teams will use the functional specifications and user acceptance criteria to detect failure incidents.
- Event logs describe how the system will capture information and how it will be reviewed.
- What tools are necessary for the teams to use to detect, diagnose, and correct errors and to improve a solution’s performance.

Select Strategies for Error Handling

The reporting of errors is contained in the monitoring plan. The terms error and exception are often used interchangeably. In fact, an error, which is an event that happens during the execution of code, interrupts or disrupts the code’s normal flow and creates an exception object. When an error interrupts the flow, the program tries to find an exception handler—a block of code that tells it how to react—that will help it resume the flow. In other words, an error is the event; an exception is the object that the event creates.

The level of exception handling depends on the level of monitoring required. There are several methods for handling these errors. First, a generic catchall error handling would catch all errors that occur in the solution. This could allow...
a centralized handler to report the error, but it would be unable to respond to the action and gracefully recover. The second option is to code the application where no exceptions are thrown outside of the class that the error resides in. The drawback with this method is that you would have to have all of your functions return some type of status and it would not allow for bubbling of event handling. The last option is a combination of these two methods. You can create a global error handler that is used for reporting purposes and acceptable graceful solution actions, while the handling of errors can be either thrown to the calling client or handled inside of the respective block of code and performing alternative actions instead.

**Select Strategies for Integration**

Integration is the interaction of heterogeneous applications, meaning the application needs to interact with existing applications. This can also be called interoperability. Interoperability reduces operational cost and complexity and takes advantage of existing investments. When designing for interoperability, you gain the following advantages:

- Reduces operational cost and complexity. The customer can continue to work in mixed environments for the near future. A mixed environment is where applications are using different operating systems and platforms.

- The ability for different systems to operate in the same environment together reduces the cost of developing and supporting a heterogeneous infrastructure.

- Enables diverse deployment. A business requirement can state that the application must support external applications that exist on other platforms. An interoperable application enables the organization to continue using the diverse applications that address specific requirements.

- Uses existing investments. Typically, customers have large and diverse ranges of systems installed in their environments and move to a new platform gradually. In a typical environment, a new application must be able to interact with previous applications and might be web aware and need access from a different hosting environment such as an IBM mainframe.

- Extends the functionality of existing applications and protects the investments that the organizations have made.
To integrate heterogeneous applications, you need to consider the following types of interoperability:

- **Network interoperability**  Refers to the ability of multiple vendor systems to communicate with each other without having to use common protocols. Implementing technologies or standards such as HTML, XML, or web services to make use of the Internet can make your applications independent of programming language, platform, and device.

- **Data interoperability**  Refers to the ability of applications to access and use data stored in both structured and unstructured storage systems such as databases, file systems, and e-mail stores. Enterprise applications often require the sharing of data between disparate data sources and multiple applications. Published data exchange standards, such as cascading style sheets, OLE DB, and XML, allow data access to both Windows-based and non-Windows–based data sources.

**Integration between systems is a very important concept. Most commonly integrations are accessing an existing mainframe.**

- **Applications interoperability**  Refers to the infrastructure required to ensure interoperability between applications written for different platforms and languages. These new applications need to work with a wide variety of existing applications. One of the methods of enabling application interoperability is by using Common Language Specification (CLS). CLS is a standard that is currently met by more than 20 different languages, including C#, VB.NET, J++, and COBOL.

- **Management interoperability**  Refers to the tasks of user account management, performance monitoring, and tuning for heterogeneous applications in the organization.

**Select Strategies for Globalization**

Globalization is the process of designing and developing an application that can operate in multiple cultures and locales. Culture and locale are defined by rules
and data that are specific to a given language and geographical area. Globalization involves:

■ Identifying the cultures and locales that must be supported
■ Designing features that support those cultures and locales
■ Writing code that executes properly in all the supported cultures and locales

Globalization enables applications that can accept, display, and output information in different language scripts that are appropriate for various geographical areas. To globalize these functions, you use the concept of cultures and locales. These rules include information about:

■ Character classification
■ Data and time formatting
■ Numeric, currency, weight, and measure conventions
■ Sorting rules

Some of the issues that you need to consider while planning for globalization are:

■ Language issues are the result of differences in how languages around the world differ in display, alphabets, grammar, and syntactical rules.
■ Formatting issues are the primary source of issues when working with applications originally written for another language, culture, and locale. Developers can use the National Language Support (NLS) APIs in Microsoft Windows or the System.Globalization namespace to handle most of this automatically. These factors include addresses, currency types, dates, paper sizes, telephone numbers, time formats, and units of measure.

The following best practices provide some information about globalization practices:

■ Use Unicode as your character-encoding standard to represent text including all application process data, whether text or numerical. Different cultures and locales might use different data encoding techniques.
Unicode is a 16-bit international character-encoding standard that covers values for more than 45,000 characters that are defined out of a possible 65,535. It allows each character in all the required cultures and locales to be represented uniquely.

When implementing a multilingual user interface, you design the user interface to open in the default UI language, and offer the option to change to other languages, users who speak different languages can quickly switch to the preferred interface. This is a common practice for web applications. For .NET Windows applications, the Windows operating system can change its locale and the .NET application will automatically change, provided support for the locale has been added.

Cultural and political issues include disputes related to maps, which can induce governments to prevent distribution in specific regions. To avoid such issues:

- Avoid slang expressions, colloquialisms, and obscure phrasing in all text.
- Avoid maps that include controversial regional or national boundaries.

**Select Strategies for Localization**

Localization is the process of adapting a globalized application to a specific locale, using separate resources for each culture that is to be globalized. A resource file contains culture-specific user interface items that are provided to an application as a text file, a .resx file, or a .resource file. An application prepared for localization has two conceptual blocks: the data block and the code block. The data block contains all user-interface string resources. The code block contains the application code that is applicable for all cultures and locales and accesses the correct resources file for the culture currently selected in the operating system. To create a localized version of an application, ensure that the code block is separate from the data block and the application code can read data accurately, regardless of culture and locale. Some issues to consider are:

- **String-related issues**  Strings are the text displayed in the various elements in an application’s user interface, such as menus and dialog boxes. For example, an error statement comprised of multiple concatenated strings in one language could be rearranged in a totally different order for another, making the concatenated result incorrect.
User interface issues:
- The length of a message might differ in different languages.
- Menu and dialog boxes might become larger because of localization.
- Icons and bitmaps must use symbols that are internationally accepted and convey the same meaning.
- The keyboards used in different locales might not have the same characters and keys.
- UI controls should not be hidden or used as parts of strings.

Some best practices to consider are:
- Examine Windows messages that indicate changes in the input language, and use that information to check spellings, select fonts, and so on.
- Detect the culture that your application uses to handle the formatting and change it to correspond to the culture that the user interface application supports.
- Store all user interface elements in resource files so they are separate from the program source code.
- Use the same resource identifiers throughout the life of the project. Changing identifiers makes it difficult to update localized resources from one version to another.
- Avoid text in bitmaps and icons. By having text in the image, the image will have to be recreated for each locale supported, instead of having a neutral image with the text located in resource files.
- Test localized applications on all language variants of the operating system.

Select Strategies for Data Storage
Data storage or data store is typically a database in which data is organized and stored. Data requirements for the solution specify how data will be structured, stored, accessed, and validated in the solution. With the requirements in mind, a technology for storing the data needs to be selected. The physical data model of a database management system (DBMS) defines the internal structure that the DBMS uses
to keep track of data. The various types of physical data models that commonly exist are:

- **Flat-file**  
  A flat-file database access stores all data in a single file as a set of rows and columns. There is no relationship between multiple flat-file databases because each database exists without knowledge of any other database. They can provide fast updates and retrieval because they support an indexing method called the indexed sequence access method (ISAM). Legacy mainframe databases, as an example, implement ISAM storage technology.

- **Hierarchical**  
  Hierarchical databases store a wide range of information in a variety of formats. Examples of this include Microsoft Exchange and the Windows Registry. This type of storage is extensible and flexible. This option is good when the information storage requirements vary greatly.

- **Relational**  
  In a relational model database, data is stored in multiple tables and columns. Relational databases combine the advantages of both flat-file and hierarchical databases by providing good performance and flexibility of storage. The relational model tends to be the most popular because tables can be linked together with unique values. Data integrity is maintained by applying rules and constraints.

**Select Strategies for State Management**

State management is the process by which information over multiple requests is maintained. State management is most commonly used in ASP.NET but does not have to exclusively be used there. ASP.NET provides multiple ways to maintain state between server round trips. Choosing among the options for state management depends on your application. Some criteria to consider include:

- How much information do you need to store?
- Does the client accept persistent or in-memory cookies?
- Do you want to store the information on the client or server?
- Is the information sensitive?
- What sorts of performance criteria do you have for your application?

.NET supports various client-side and server-side options for state management. Client-side options are the ViewState property, hidden fields, cookies, and query
strings. These options are ASP.NET specific. Server-side options include application state, session state, and the database. These options are not available for WinForm applications; a WinForm application can use client-side XML files for state management.

Application state, session state, and using the database are not options for handling state for Windows forms. These are ASP.NET specific options.

Client-side state management conserves the use of server resources. These options tend to have minimal security but faster server performance because demand on server resources is minimal. There are limits on how much information can be stored using the client-side options. Table 4-2 summarizes client-side state management options and suggested uses.

Server-side options for storing page information tend to have higher security than client-side, but they require more server resources. Table 4-3 summarizes server-side state management options and when you should consider using them.

### Selecting a Deployment Strategy

The deployment plan describes the factors necessary for a problem-free deployment and transition to ongoing operations. It includes the processes of preparing, installing, training, stabilizing, and transferring the solution to operations. These processes include details of installation scenarios, monitoring for stability, and verifying the soundness of the new solution. Deployment is the beginning of the realization of business value for a given solution.

<table>
<thead>
<tr>
<th>Method</th>
<th>Use When</th>
</tr>
</thead>
<tbody>
<tr>
<td>View state</td>
<td>You need to store small amounts of information for a page that will post back to itself and provide a basic level of security.</td>
</tr>
<tr>
<td>Hidden fields</td>
<td>You need to store small amounts of information for a page that will post back to itself and security is not an issue.</td>
</tr>
<tr>
<td>Cookies</td>
<td>You need to store small amounts of information on the client and security is not an issue.</td>
</tr>
<tr>
<td>Query string</td>
<td>You are transferring small amounts of information from one page to another and security is not an issue.</td>
</tr>
<tr>
<td>XML file</td>
<td>You need to store application user settings and security settings.</td>
</tr>
</tbody>
</table>
Some key sections of the deployment plan are:

- **Deployment scope** Describes the solution architecture and scale of deployment.
- **Seats** Describes the magnitude of the deployment in terms of sites, number of workstations, countries, and regions.
- **Components** Lists and describes the components to be deployed and any critical dependencies among them.
- **Architecture** Describes the solution’s architecture and how it might affect deployment.
- **Deployment schedule** Identifies the critical dates and anticipated schedule for the deploying phase.
- **Installation** Defines how the overall deployment will occur.
- **Deployment resources** Identifies the workforce that will be needed to complete the deployment and the sources of the personnel.
- **Solution support** Describes how the users will be supported during the deployment.
- **Help desk** Describes the support provided to users and applications by the help desk team.
- **Desktop** Describes any changes in current workstation application support that might be required during deployment.
Servers  Describes any changes in current server support that might be required during deployment.

Telecommunications  Describes any changes in current telecommunication support that might be required during deployment.

Coordination of training  Describes how end-user and support staff training is coordinated with the deployment schedule.

Site installation process  Describes the four phases of site installation: preparation, installation, training, and stabilization.

Select Strategies for Deployment, Such as Coexistence Strategies

The .NET framework provides a number of basic features that make it easier to deploy a variety of applications. These features include:

- **No-impact applications**  This feature provides application isolation and eliminates DLL conflicts. By default, components do not affect other applications.

- **Private components by default**  By default, components are deployed to the application directory and are visible only to the containing application.

- **Controlled code sharing**  Code sharing requires you to explicitly make code available for sharing rather than being the default behavior.

- **Side-by-side versioning**  Multiple versions of a component or application can coexist, you can choose which versions to use, and the Common Language Runtime enforces versioning policy. This type of versioning is only available for when the assemblies are installed into the global assembly cache.

- **XCOPY deployment and replication**  Self-described and self-contained components and applications can be deployed without registry entries or dependencies.

- **On-the-fly updates**  Administrators can use hosts, such as ASP.NET, to update program DLLs, even on remote computers.

- **Integration with the Microsoft Windows Installer**  Advertisement, publishing, repair, and install-on-demand are all available when deploying your application.
Transforming Functional Specifications into Technical Specifications

- **Enterprise deployment**  This feature provides easy software distribution, including using Active Directory.

- **Downloading and caching**  Incremental downloads keep downloads smaller, and components can be isolated for use only by the application for zero-impact deployment.

- **Partially trusted code**  Identity is based on the code rather than the user, policy is set by the administrator, and no certificate dialog boxes appear.

The deployment of a .NET application can be divided into two phases: packaging and distribution. Packaging is the creating of a manifest of required files necessary for distribution. Distribution is the actual process of moving the required files to the appropriate location(s) necessary for the solution.

The .NET framework has the following options for packaging applications. First as a single assembly, or as a collection of private assemblies, which allows the use of .dll or .exe files in their original state. Second, as cabinet (.cab) files, which are a compressed version of the required files. This option makes the distribution or download less time consuming. Lastly, as a Microsoft Windows Installer 2.0 package or in another installer format, which creates .msi files that can be used with Windows Installer.

After preparing the package, the files need to be distributed. There are three primary ways to distribute applications. The first option is the simplest, using XCOPY or FTP. Because Common Language Runtime applications do not require registry entries and are self-describing, the files can just be copied to the appropriate directory, where the application will be running from. The second is a code download, which can be accomplished by using web installers that allow the user to copy files to the client or enable automatically updating applications a location to download updates. Lastly, by using an installer application, you can install, repair, or remove .NET framework assemblies in the global assembly cache and in private directories.

There are three common deployment scenarios: the deployment of an ASP.NET application, a Windows Forms application, and a Windows Forms control by downloading the application to the client. The first, an ASP.NET application, is a package containing the application and .DLLs. These are distributed by XCOPY or FTP. The second, a Windows Forms application, is packaged into a Microsoft Windows Installer package (.msi) and distributed with Windows Installer. The last, a Windows Forms control or other code, is packaged into compressed CAB files (.cab) or compiled libraries (.dll) and the distribution is a file download from the source.
Windows Installer

One of the tools that has been mentioned as a distribution means is the Windows Installer. This is a powerful tool for the setup and distribution of assemblies. The Windows Installer packages can install assemblies to the global assembly cache or to a specific application location. The ability to isolate applications is an important part of the .NET framework. Isolating applications allows for multiple versions of components that need to be distributed.

Windows Installer has the following features that support Common Language Runtime assemblies:

- Installation, repair, or removal of assemblies in the global assembly cache (GAC). The global assembly cache allows for .NET assemblies to be shared between multiple applications.
- Installation, repair, or removal of assemblies in private locations designated for particular applications.
- Rollback of unsuccessful installations, repairs, or removals of assemblies.
- Install-on-demand of strong-named assemblies in the global assembly cache.
- Install-on-demand of assemblies in private locations designated for particular applications.
- Patching of assemblies.
- Advertisement of shortcuts that point to assemblies.

Windows Installer treats an assembly built with the Microsoft .NET framework as a single Windows Installer component. All the files that constitute an assembly must be contained by a single Windows Installer component that is listed in the component table of the Installer. When installing assemblies into the global assembly cache, the Installer does not use the same directory structure and file versioning rules that it uses to install regular Windows Installer components. Instead, assemblies are added and removed from the global assembly cache as a unit—that is, the files that constitute an assembly are always installed or removed together. With private assemblies, these files can be removed and updated individually. Windows Installer uses a two-step transactional process to install products containing assemblies, which enables the installer to roll back unsuccessful installations.
Assembly Versioning

All versioning of assemblies that use the Common Language Runtime is done at the assembly level. The specific version of an assembly and the versions of dependent assemblies are recorded in the assemblies manifest. The default policy is for the runtime to only use versions they were built and tested with. This can be overridden by creating an explicit version policy in the configuration files. These configuration files are the application configuration file, the publisher policy file, and the machine configuration file.

The runtime performs the following steps to resolve an assembly-binding request. This is where the application locates and loads dependent assemblies.

1. Checks the original assembly reference to determine the version of the assembly to be bound.
2. Checks for all applicable configuration files to apply version policy.
3. Determines the correct assembly from the original assembly reference and any redirection specified in the configuration files, and determines the version that should be bound to the calling assembly.
4. Checks the global assembly cache, codebases specified in configuration files, and then checks the application’s directory and subdirectories using the probing rules.

*Versioning is done only on assemblies with strong names.*

Each assembly has a version number. As such, two assemblies that differ by version number are considered by the runtime to be completely different assemblies. This version number is physically represented as a four-part number with the following format:

<major version>.<minor version>.<build number>.<revision>

For example, version 1.2.200.0 indicates 1 as the major version, 2 as the minor version, 200 as the build number, and 0 as the revision number.

The version number is stored in the assembly manifest along with other identity information, including the assembly name and public key, as well as information on relationships and identities of other assemblies connected with the application. When
an assembly is built, the dependency information for each assembly that is referenced is stored in the assembly manifest. The runtime uses these version numbers, in conjunction with configuration information set by an administrator, an application, or a publisher, to load the proper version of a referenced assembly. The runtime distinguishes between regular and strong-named assemblies for the purposes of versioning. Version checking only occurs with strong-named assemblies. A strong-named assembly adds a public key and a digital signature to the assembly. A strong-named assembly is guaranteed to be globally unique.

**EXERCISE 4-1**

Creating Strong-Named Assemblies and Versioning Assemblies

In this exercise, you will examine how versioning works with .NET. The .NET framework allows applications to support multiple versions of dependent assemblies. This allows the application to determine which version of the assembly it needs to execute.

1. Start Visual Studio .NET.
3. Change the name of the project to WinVersioningApp and change the location to c:\.
4. Click More and make sure that Create Directory For Solution is checked.
5. In New Solution Name, type **Versioning** and click OK. A new project and solution is created.
7. In the Name field, type **VersionLibrary** and click OK. The solution now has two projects, VersionLibrary and WinVersioningApp.
8. Modify the AssemblyVersion line of AssemblyInfo.vb in the VersionLibrary project to resemble the following example. The AssemblyInfo.vb file contains descriptive information about the assembly being built. It can contain information about who built it and why and strong-name assembly keys. The change that will be made will prevent the version number from automatically incrementing.
Transforming Functional Specifications into Technical Specifications

From
<Assembly: AssemblyVersion("1.0.*")>
To
<Assembly: AssemblyVersion("1.0.0.0")>

9. Add the following code to the Class1.vb. The code should look like the following illustration.

```
Public Class Class1
    Public Function Version() As String
        Return "1.0.0.0"
    End Function
End Class
```

Public Function Version() as String
    Return "1.0.0.0"
End Function

10. To create a strong-name key for the VersionLibrary project, select Start | All Programs | Microsoft Visual Studio .NET | Visual Studio .NET tools | Visual Studio .NET Command Prompt. A command prompt window will display. This window has set up the path environment for the DOS environment for .NET.

11. Type cd\ and press ENTER.

12. Type SN -k StrongName.snk and press ENTER. This creates a key file for creating strong-named assemblies.
13. Copy StrongName.snk to your project directory `c:\Versioning\VersionLibrary`.

14. Add the following line at the end of the AssemblyInfo.vb file in VersionLibrary. This syntax indicates where the StrongName.snk file is located. After the project is built, the assembly is two levels down from the project, the signing of the assembly is performed, and the correct location of the signing key must be found.

   ```vbnet
   <Assembly: AssemblyKeyFile("..\..\..\StrongName.snk")>
   ```

15. Right-click VersionLibrary project and select Build. The ClassLibrary project should build with no problems.


17. Add a button to Form1.vb on the WinVersioningApp project and double-click on the button in the designer.
18. In the event handler, type the following:

```vbnet
Private Sub Button1_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles Button1.Click
    Dim ox As New VersionLibrary.Class1()
    MsgBox(ox.Version)
End Sub
```

19. Build and run WinVersioningApp. Click the button on the screen (shown as Button1 in the following illustration) and a message box should appear with 1.0.0.0 as the displayed text.

![WinVersioningApp screenshot](image)

20. In the .NET command prompt window, locate the path to your VersionLibrary .dll assembly and add the file to the global assembly cache. Type the following command:

```
GACUTIL /i VersionLibrary.dll
```

21. Version 1.0.0.0 is now in the global assembly cache. Change the version number in the Assemblyinfo.vb file on VersionLibrary to 2.0.0.0.

22. Change the Version function in Class1 to return “2.0.0.0” and build VersionLibrary and not the solution, by selecting the VersionLibrary project and selecting Build | Build VersionLibrary.
Chapter 4: Formalizing Business Requirements into Functional Specifications

23. Close Visual Studio.NET and run WinVersioningApp again. It still displays 1.0.0.0 when the button is clicked on the form.

24. In the .NET command prompt window, execute the following command where the VersionLibrary.dll is located.

   GACUTIL /i VersionLibrary.dll

25. Type the following command in the command window. This command lists all components installed into the global assembly cache that start with the name VersionLibrary.

   GACUTIL /1 VersionLibrary

26. A list displays the two VersionLibrary.dll that you created, each with different version numbers.

27. Select Start | All Programs | Administrative Tools | Microsoft .NET Framework Configuration. This .NET administrative tool allows for the viewing and configuring of .NET assemblies and code access security policies.

28. Select Application | Add An Application To Configure. A dialog box appears showing all .NET applications that have run on your machine.


30. Select Configured Assemblies | Action Menu Item | Add.

31. Select Choose Assembly. In the window that opens, select the first item (“Choose an assembly from the list of assemblies this application uses”), then click Choose Assembly. Highlight VersioningLibrary and click Select. Click Finish.
32. The VersionLibrary Properties window is displayed. Click the Binding Policy tab. Under Requested Version, enter 1.0.0.0. Under New Version, enter 2.0.0.0. Click OK to close the window.
33. Run WinVersioningApp. This time, the result from the button click will be 2.0.0.0 instead of 1.0.0.0.

In this exercise, you created a solution with a Windows application and a class. You added assemblies into the global assembly cache and then rebinded the assembly's dependent files to other versions.

**Side-by-Side Execution**

Side-by-side execution is the ability to run multiple versions of the same assembly simultaneously. Components that can run side by side do not have to maintain backward compatibility. This allows applications to use newer components when necessary. For example, a component called Employee supports side-by-side executing. Between version 1.0.0.1 and 1.0.0.2 some incompatibilities were introduced. Multiple releases of the application have followed, and some releases support version 1.0.0.1 and others support 1.0.0.2. Each release can use the appropriate version needed while still having multiple versions of the Employee component installed. For side-by-side execution to occur, the necessary assemblies must be installed in the global assembly cache (GAC).

Support for side-by-side storage and execution of different versions of the same assembly is an integral part of versioning and is built into the infrastructure of the runtime. Because the assembly's version number is part of its identity, the runtime
can store multiple versions of the same assembly in the global assembly cache and load those assemblies at runtime. Although the runtime provides the ability to create side-by-side applications, it is not an automatic process.

There are two types of side-by-side execution:

- **Running on the same computer** In this type of side-by-side, multiple versions of the same application run on the same computer at the same time without interfering with each other.

- **Running in the same process** An application running in the same process means that single applications with multiple dependencies can run multiple versions of the same component. An example of this would be web applications.

**Select Strategies for Licensing**

License specifications provide details for the developing and deploying phases on the number of licenses necessary for compliance. It helps to determine the purchasing requirements early in the project for both hardware and software that will be required for the solution. Purchasing specifications developed early in the process ensure that there is sufficient time for the approval process and that vendors have sufficient time to deliver hardware so the schedule will not be affected. Depending on the type of solution and number of users who will use the solution, you need to specify the number of licenses of any software that might be used.

Another area of concern is licensing of your solution. If you are developing components, your components can be licensed to ensure that only valid users can develop and use them. There are other models for licensing, such as a one-time license, subscription models for a specific time period or number of uses, or licensing based on the number of users or processors.

**Select Strategies for Data Migration**

Data migration is the moving of data from an existing solution to a new solution. When data from an existing solution is identified as part of the new solution, data migration becomes a critical element. A well-tested migration is necessary to prevent the introduction of risks that were never accounted for during the planning. More information about risk planning is in Chapter 1.

A document called the migration plan describes the migration from the existing systems to the new solution. Migration is typically more important in infrastructure
deployments, but can be just important in application development projects.

A migration plan includes the following sections:

- **Migration strategies** Describe how the migration process will be guided. These strategies are generally geared towards multiple software releases and preparation for each. The strategy can be organized in sections to allow for multiple software releases of solution components.

- **Tools** Identify what will be used to support the migration strategy.

- **Migration guidelines** Describe the constraints, validations, and order for the data migration.

- **Migration process** Describes the preparatory activities in addition to the migration stages necessary to complete the migration process.

- **Test environment** Describes an environment that mirrors the production environment.

- **Rollback plan** Describes how a customer can roll back to the prior configuration if problems occur during migration.

**Exam Watch**

A *common way to migrate data is to use Microsoft SQL Data Transformation Services (DTS) packages.*

### Selecting a Security Strategy

Designing security features and policies is one of the most important aspects of an application. To be able to design a secure application, you should be familiar with the principles of security. You should consider these principles when creating security strategies. For more information on security practices, refer to Security Best Practices in MSDN. The following are some security principles:

- Whenever possible, you should rely on tested and proven security systems rather than creating your own custom solution.

- You should never trust external inputs. Validate all data that is entered by users or submitted by other systems.

- If your application receives unencrypted sensitive data from an external system, you should assume that the information is compromised.

- Service accounts should have the least permissions required. A service account refers to a user account that services on the computer use to access resources.
This account can be a local account or a network domain account for access to network resources.

- Risk will increase with the number of components and the amount of data you have made available through the application, so you should minimize the amount of public method and properties available to users.

- Do not enable services, account permissions, and technologies that you do not explicitly need. When you deploy the application on client or server computers, the default configuration should be secure. For example, if your web application needs to access network resources, the account used for IIS needs to have permission to access the minimum amount necessary.

- Encrypting data implies having keys and a proven encryption algorithm. Secure data storage will prevent access under all circumstances.

Follow the STRIDE principles. Each letter in the STRIDE acronym specifies a different category of security threat: spoofing identify, tempering, repudiation, information disclosure, denial of service, and elevation of privilege. STRIDE is used to define a security thread model to predict and evaluate potential threats to the system. Each of these threats are identified in Chapter 3. Most security threats are actually a combination of types. The threat model identifies all of the possible ways a user can violate the solution and use cases are created to identify these. Testing is created later to determine if the security threats are minimized or eliminated.

<table>
<thead>
<tr>
<th>SCENARIO &amp; SOLUTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A malicious user views and changes payroll data that travels between a web browser and the web server. List and categorize threats…</td>
</tr>
<tr>
<td>This type of action would fall into the categories of tampering and information disclosure. The user is viewing and editing restricted data.</td>
</tr>
<tr>
<td>Rank the risk of each threat…</td>
</tr>
<tr>
<td>Tampering is ranked higher than information disclosure. Both are important, but editing the restricted data is more critical.</td>
</tr>
<tr>
<td>Apply the threat model…</td>
</tr>
<tr>
<td>In applying the threat model, the site needs to encrypt data between the web browser and web server. This can be accomplished by creating public and private keys and encrypting the data. This will eliminate the risks of tampering and information disclosure.</td>
</tr>
</tbody>
</table>
Selecting Strategies to Ensure Data Privacy

Signing code with a strong name defines the unique identity of code and guarantees that code has not been compromised. Code signing is the process of providing a set of code with credentials that authenticate the publisher of the code. The credentials of the code can be verified prior to installing and running the code. The purpose is to ensure that users know the origin of the code and to help prevent malicious users from impersonating the identity of a publisher. It also verifies that the code has not been changed by unauthorized sources since it was published.

When encrypting data, there are two types of keys available, private and public. Private key cryptography uses a single non-public key to cipher and decipher data. This is also called symmetric cryptography. The algorithms used encrypt blocks of data at a time. Based on the type of encryption used determines the number of bytes in the block.

Public key cryptography uses a private key and a public shared key that anyone can use. The public key and the private key are keyed pairs. Data ciphered with the public key can only be deciphered with the private key, and data signed with the public key can only be verified with the public key. This is known as asymmetric cryptography—one key is used to cipher and the other is used to decipher. Public key cryptography follows a different model as far as ciphering the data. It uses a fixed buffer size instead of a variable length like private key cryptography. For example, public keys could be used to send small amounts of data between two public systems.

Cipher refers to the process of disguising data before it is sent or stored. Data that has undergone encryption is called ciphertext. Data that has not been encrypted is referred to as plaintext. Decipher is the process of decrypting ciphertext into readable plaintext. The processes of encrypting and decrypting data rely on the techniques of hashing and signing data.

Hashing is the process of matching data of any length to a fixed-length byte sequence. The fixed-length byte sequence is called a hash. A hash is obtained by applying a mathematical function, called a hashing algorithm.

Signed data is a standards-based data type. Signed data consists of any type of content plus encrypted hashes of the content for zero or more signers. The hashes are used to confirm the identity of a data signer and to confirm that the message has not been modified since it was signed.
Code Access Security
Code access security allows code to be trusted to varying degrees, depending on the code’s origin, code’s evidence, or strong name signature, and on other aspects of the code’s identity. For example, code that is downloaded from your organization’s intranet and published by your organization might be trusted to a greater degree than code downloaded from the Internet and published by an unknown entity. The .NET framework allows you to include features in your application that request a specific level of security privilege from the system.

Select Strategies to Ensure Secure Access
Role-based security relates mostly to the spoofing identity security threat by preventing unauthorized users from performing operations that they are not authorized to perform. Role-based security allows code to verify the identity and role membership of the user.

The .NET framework supports role-based security by using the principal object. This object contains the security context of the current thread. The associated identity contains at the minimum the type of security protocol and the name of the user. Security is based on the security context of the Principal object. The IPrincipal object has two classes that implement this interface, GenericPrincipal and WindowsPrincipal. Each are both used to determine the user’s identity and any roles that they belong to.

Authentication
Authentication is the process of discovering and verifying the identity of a user by examining the user’s credentials and then validating those credentials against some authority. Examples of commonly used authentication mechanisms include the operating system, Passport, and application-defined mechanisms like NTLM and Kerberos authentication.

The WindowsPrincipal and WindowsIdentity objects contain information about the Windows accounts. If you are using Windows NT or Windows 2000 domain for authentication, these are the objects you want to use. The following code illustrates how to get access to the WindowsPrincipal:

[VB.NET]
AppDomain.CurrentDomain.SetPrincipalPolicy( _
PrincipalPolicy.WindowsPrincipal)
Dim MyPrincipal as WindowsPrincipal = ___
Chapter 4: Formalizing Business Requirements into Functional Specifications

The following table lists the supporting values that can be used for setting the principal policy.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoPrincipal</td>
<td>No principal or identity objects should be created.</td>
</tr>
<tr>
<td>UnauthenticatedPrincipal</td>
<td>Principal and identity objects for the unauthenticated entity.</td>
</tr>
<tr>
<td>WindowsPrincipal</td>
<td>Principal and identity objects that reflect the operating system token associated with the current execution thread should be created, and the associated operating system groups should be mapped into roles.</td>
</tr>
</tbody>
</table>

**The WindowsPrincipal is the most common principal used. It is used for Windows Authentication, otherwise use the GenericPrincipal.**

The GenericPrincipal and GenericIdentity are used for authorization independent of Windows NT and Windows 2000 domain. A classic example of this would be an application that requests user name and password for authorization, and the information is checked against a database or XML file. The following code shows how to use the GenericPrincipal is very similar to the WindowsPrincipal object.

[C#]
```
GenericIdentity MyIdentity = new GenericIdentity("MyUser");
String[] MyStringArray= ("Manager","User");
GenericPrincipal MyPrincipal = new GenericPrincipal(MyIdentity, MyStringArray);
Thread.CurrentPrincipal = MyPrincipal;
```

[VB.NET]
```
Dim MyIdentity as New GenericIdentity("MyUser")
```
Dim MyStringArray as String() = {"Manager","User"}
Dim MyPrincipal = new GenericPrincipal(MyIdentity,MyStringArray)
Thread.CurrentPrincipal = MyPrincipal

After you obtain the principal, it can be examined for validation.

**Authorization**  
Authorization is the process of determining whether a user is allowed to perform a requested action. Authorization occurs after authentication and uses information about a user’s identity and roles to determine what resources that user can access. .NET’s role-based security can be used to implement authorization.

To validate where the user has permissions to perform specific actions in role-based security, is to use the PrincipalPermission object. This object determines what user and role has permission to perform the task. For example, the following code checks for the user Bob and that he is in the Manager role to perform the task. This is considered using an imperative security check.

[C#]
PrincipalPermission princPerm = new PrincipalPermission("Bob","Manager");

[VB.NET]
Dim princPerm As New PrincipalPermission("Bob","Manager")

Another way this can be done is using a declarative security check, which uses attributes for it definition. The syntax for this method is very similar to the previous code:

[C#]
[PrincipalPermissionAttribute(SecurityAction.Demand, Name="Bob", Role = "Teller")]

[VB.NET]
<PrincipalPermissionAttribute(SecurityAction.Demand, Name := "Bob", Role := "Teller")>

If the user accessing the method did not satisfy the role or user conditions an exception would be thrown. The type of exception that is thrown is called the SecurityException.

**examWatch**  
There are two methods of verifying the user using role-based security, declarative (using attributes) or imperative (using code).
Authorization Strategy  To design authentication and authorization strategies for your application, you need to perform the following steps:

- Identify the resources that the application provides to the clients. An example of a resource would be web server resources such as web pages and web services.
- Select an authorization strategy. There are two authorization strategies: role-based and resource-based. Role-based has users assigned to roles and the user is verified against the authorized roles to determine if the requested operation can be performed. In a resource-based strategy, individual resources are secured by using Windows access control lists (ACLs). The application impersonates the caller prior to accessing resources, which allows the operation system to perform standard access checks. Impersonation is the act of performing an action using a different security context than the calling user’s.
- Choose the identity or identities that should be used to access resources across the layers of your application. There are four types of identities that can be used:
  - Original caller’s identity  Assumes an impersonation or delegation model in which the original caller identity can be obtained and then flowed through each layer of your system.
  - Process identity  This is the default case. Local resources access and downstream calls are made using the current process identity.
  - Service account  Uses a fixed service account. For example, for database access this might be a fixed user name and password presented by the component connecting to the database. When a fixed Windows identity is required, use an Enterprise Services server application. This allows a centralized location for a component. The machine that is running the service can determine the user name and password independently.

**Impersonation allows a component to pass the calling user’s security context to determine if the user can use network resources.**

**Service accounts can secure parts of an application that must access network resources.**
Custom identity When you do not have Windows accounts to work with, you can construct your own identities that can contain details that relate to your own specific security context.

To support per-user authorization, auditing, and per-user data retrieval, you might need to flow the original caller's identity through various application tiers and across multiple computer boundaries.

Selecting an authentication approach is based upon the nature of your application's user base, the types of browsers they are using, and whether they have Windows accounts, and your application's impersonation/delegation and auditing requirements.

Decide how to flow the identity and security context at the application level or at the operating level. To flow identity at the application level, use method and stored procedure parameters, for example, adding the user name into the database queries. Operating system identity flow supports Windows auditing, SQL Server auditing, and per-user authorization based on Windows identities.

Selecting an Operations Strategy

The operations strategy deals with the daily activities to run the system. Security administration is responsible for maintaining a safe computing environment by developing, implementing, and managing security controls. System administration is responsible for the day-to-day task of keeping enterprise systems running and for assessing the impact of planned releases. Network administration is responsible for the design and maintenance of the physical components that make up the organization's networks, such as servers, routers, switches, and firewalls. Service monitoring and control observes the health of specific IT services and acts when necessary to maintain compliance. Storage management deals with on-site and off-site data storage for the purpose of data restoration and historical archiving and ensures the physical security of backups and archives.

These objectives are geared around predefined service-level agreements (SLA) negotiated by the project team and support team. A service-level agreement is defined as an agreement between the service provider and the customer that defines the responsibilities of all participating parties and that binds the service provider to provide a particular service of a specific agreed-upon quality and quantity. It constrains the demands that customers can place on the service of those limits that are defined by the agreement.
Select Strategies for Data Archiving and Data Purging

Data archiving is determining what needs to be backed up, when, and how often. The key requirements to be met for the backup process are defined by the application and the organization. Requirements to consider are described in the following categories:

- **Size requirements**  The amount of required data storage needs to be determined for each data type. Understanding whether terabytes or megabytes of data are involved has a major influence on the specific strategy.

- **Data location**  The physical and logical location of the data to be backed up must be determined. In addition to simply identifying the physical location of the data, it must be ensured that backup devices can access the data to be backed up and restored. This will help in determining the types of devices required for performing backups, the media required, and the necessary time window.

- **Backup timetable**  How often the data needs to be backed up per data type needs to be determined. For example, application data can be backed up daily, system data every week, and critical database transactions twice a day.

While defining the backup requirements for different data types, planning should also occur on how the storage media should be secured and maintained. Based on requirements some information may need to be stored off-site. The strategies that you employ can be different for each application and server. For example, SQL Server 2000 employs the use of transaction logs that can be backed up to prevent the use of full backups. There are three types of backups that you can perform:

- A full backup copies all selected files and marks each file as having been backed up, by clearing the archive attribute. For a full backup, only the most recent copy of the backup tape is required to restore all files. A full backup is usually performed the first time a backup set is created. It is a common practice to perform a full backup at regular intervals, such as every week or month. Intervals should be based on the relative dynamic nature of the data generated by the solution.

- A differential backup copies only those files that are new or have been changed since the last full or incremental backup. This kind of backup does not mark files as having been backed up. If a combination of full and differential backups is used, files or tapes from both the last full and differential backups are needed to restore files and folders.
An incremental backup backs up only those files that are new or have changed since the last full or incremental backup. If a combination of full and incremental backups is used, the last normal backup set, as well as all incremental backup sets, is needed to restore data.

**Incremental backups do not have the same behavior for SQL Server 2000.**

Data purging is the elimination of unnecessary information that is not being used with the application. Over a period, solutions can generate a large amount of data. It has to be determined when it can be deleted because it is not useful anymore. The data can be removed or moved into an archive location for safekeeping. If the backup strategy is correct, the data can be removed without worry because it can be recovered if needed. There can be legal implications that require that the data be archived for long periods of time. For example, in the medical field, archived data and medical records must be preserved for various periods of time, such as seven years or longer based on the age of the client.

**Select Strategies for Upgrades**

It’s common for errors and modifications of solutions to happen, forcing upgrades for the application after it has been rolled out to the production environment.

Typically, applications are upgraded for the following reasons:

- To apply bug fixes to remedy problems with your application
- To provide enhancements for existing features
- To provide new features or capabilities

One of the major advantages of the .NET architecture is that it provides comprehensive support for modifying or upgrading applications after they have been deployed. There are several primary methods for upgrading. First is to reinstall the application. Second is to use Windows Installer to create an update package to modify the necessary components. Windows Installer can perform the Add/Remove Program, Delete File, and Add New File functions. The deleting and adding of files to an already installed solution increases the maintenance and total cost of ownership for the solution. The third option is to create a side-by-side assembly functionality. The last option would be to have the application be able to be self-updating and download and install patches as necessary.
Create a Support Plan

Your support plan should address who will provide support, what level of support they need to provide, and how users can report problems. Determine who will support the users: will it be the project team, the help desk, or external resources? If the help desk provides the support, how will you train them? What will be the role of the project team?

Determine what service levels you can support. For example, must critical problems be resolved within a specified number of hours? During what hours must support be available to users?

Document the change management and problem management processes for the solution. Your process should address these issues:

- How are change requests submitted, approved, tested, and implemented?
- Where do users post their problems?
- Can they report problems to an existing system or do you need a new mechanism, such as a web site, where users can log their problems and questions?
- How will you review, prioritize, and fix problems?
- What escalation process will you use to notify the appropriate personnel?

Creating a Test Plan

The test plan describes the strategy used to organize, and manage the project’s testing activities. It identifies testing objectives, methodologies and tools, expected results, responsibilities, and resource requirements. A test plan ensures that the testing process will be conducted in a thorough and organized manner and will enable the team to determine the stability of the solution. The test plan breaks the testing process into different elements, including unit testing, database testing, infrastructure testing, security testing, integration testing, user acceptance and usability testing, performance testing, and regression testing. These concepts are defined in Chapter 9 in more detail. All of these tests center around unit test. Each of them are at different levels and stages of the development process. Key sections of a test plan are:

- Test approach and assumption  Describes at a high level the approach, activities, and techniques to be followed in testing the solution.
Transforming Functional Specifications into Technical Specifications

- **Major test responsibilities**  Identifies the teams and individuals who will manage and implement the testing process.
- **Features and functionality to test**  Identifies at a high level all features and functionality that will be tested.
- **Expected results of tests**  Describes the results that should be demonstrated by the tests.

*The expected results for a test can be specific results and answers or could be the visual cue of responses from the system.*

- **Deliverables**  Describes the materials that must be made available or created to conduct the tests and that will be developed from the test to describe test results.
- **Testing procedures and walkthrough**  Describes the steps the testing team will perform to ensure quality tests.
- **Tracking and reporting status**  Defines the information that test team members will communicate during the testing process.

*Reliable tracking of error and testing results help determine the quality of the product.*

- **Bug reporting tools and methods**  Describes the overall bug reporting strategy and methodology.
- **Schedules**  Identifies the major test cycles, tasks, milestones, and deliverables.

The pilot plan describes how the team will move the candidate release version of the solution to a staging area and test it. The goal of the pilot is to simulate that equipment, software, and components that the solution will use when it is active. The plan also identifies how issues discovered during the pilot will be solved. The pilot helps the project team prove the feasibility of the solution approaches, experiment with different solutions, and obtain user feedback and acceptance on proposed solutions.

*The pilot is a very important part of testing. You can determine what processes work and do not work before going to production.*
Creating a Training Plan

A training plan describes the appropriate level and type of training for all users. Training can be delivered before, during, or after installation. The training plan should follow these guidelines:

- Deliver appropriate training for each user
- Vary training based on users’ background and experience
- Include a variety of media
- Train local support staff

Consider alternatives to the traditional courseware approach to training, such as coaching sessions, and a mentoring program for advanced and support users.

CERTIFICATION SUMMARY

In this chapter, we examined the functional specifications document. The functional specifications document represents what the product will be. The goals of the functional specifications are to consolidate a common understanding of the business and user requirements. This document also provides a framework for planning, scheduling, and creating the solution. This could be considered a contract between the team and the customer for what will be delivered. The functional specifications consist of the conceptual design summary, logical design summary, physical design summary, standards, and processes used by the team. The conceptual design is the process of gathering, analyzing, and prioritizing business and user perspectives on the problem and the solution, and then creating a preliminary version of the solution. The conceptual design consists of three steps: research, analysis, and optimization.

The technical specifications are a set of reference documents that usually include the artifacts of physical design, such as class specifications, component models, metrics, and network component topologies. To create this document, design considerations needed to be incorporated into the document. These design goals are scalability, availability, performance, integration, and localization.

The scalability design can be implemented two ways: scaling up or scaling out. Scaling up is the ability to add hardware components to existing server hardware, and scaling out is adding new hardware to the existing environment. Next, we discussed strategies for handling availability, which is the measure of how often
an application is available for use. These are normally explained as percentages of  
nines. The process of designing for reliability involves reviewing the application’s  
expected usage pattern, specifying the required reliability profile, and engineering  
the software architecture with the intention of meeting the profile. Performance is  
defined by key application metrics, such as transaction throughput and resource  
utilization. Integration is the ability to integrate heterogeneous applications. Localization  
is the process of adapting a globalized application to a particular culture and locale.  

The development plan describes the solution development process used for the  
project. This plan includes a monitoring plan, how the data will be stored, and state  
management techniques to be employed by the team. Application monitoring is  
used to ensure that the application is functioning correctly and performing at the  
optimal level. The monitoring plan describes what will be monitored, how the  
application will be monitored, and how the results of monitoring will be reported  
and used.

The test plan describes the strategy and approach used to plan, organize, and  
manage the project’s testing activities, and identifies testing objectives, methodologies  
and tools, expected results, responsibilities, and resource requirements.  

The deployment plan describes the factors necessary for a smooth deployment  
and transition to ongoing operations and includes the processes of preparing, installing,  
training, stabilizing, and transferring the solution to operations.
Chapter 4: Formalizing Business Requirements into Functional Specifications

TWO-MINUTE DRILL

Functional Specifications
- The functional specifications represent what the product will be.
- Functional specifications are a virtual repository of project and design-related artifacts.

Transforming Requirements into Functional Specifications
- Synthesizing of information is the process of assimilating gathered data and interpreting the results.
- Restating requirements involves categorizing and prioritizing the requirements.

Transforming Functional Specifications into Technical Specifications
- Scaling involves the designing of processes so that they don’t have to wait. It also looks at the allocation of resources for processes and designing for commutability and interchangeability.
- Availability is the examination of the expectations of the customer of how much downtime is expected and how internal resources are available for the solution to achieve high availability.
- The monitoring plan determines what, how, and when results will be monitored and reported.
SELF TEST

Functional Specifications

1. The technical team has diagnosed that the web application gets very slow when more than 1,000 people are concurrently using the application. Choose the correct methods to scale the application without adding additional servers.
   A. Add more processors
   B. Add more servers
   C. Upgrade to a more powerful computer
   D. None of the options available

2. While consulting with a client to design a scalable web application, what should be considered?
   A. Design synchronous processes
   B. Design processes that do not compete for resources
   C. Design components for interchangeability
   D. Partition resources and activities

3. The solution that is being designed needs to run 24 hours a day and 7 days a week. Which of the following considerations are correct?
   A. Reduce planned downtime
   B. Reduce unplanned downtime without clustering
   C. Use network load balancing
   D. Use queuing

Transforming Requirements into Functional Specifications

4. The current application's availability is about 50 percent. The application needs to be redesigned to reduce the planned downtime. Which are the correct techniques?
   A. Promote untested code to production
   B. Take all servers off-line for updates
   C. Use clustered servers
   D. Use network load balancing
58 Chapter 4: Formalizing Business Requirements into Functional Specifications

5. While identifying requirements for the new application, you want to show the customer the importance of reliability considerations. Which of these items are correct?
   A. Deploy untested code
   B. Prevent operational errors
   C. Prevent lost revenue and customers
   D. Isolate mission-critical applications

6. The client is having problems with code that consistently has errors and is designed incorrectly. What techniques can be suggested to improve the reliability of the application?
   A. Lack of requirements documents
   B. Using quality development tools
   C. Lack of quality software engineering processes
   D. Implementing error handling

7. After gathering requirements for the new solution, you are asked to identify performance requirements and techniques for the new solution. Which of these apply?
   A. Apply forms authentication
   B. The # of orders that can be created in an hour
   C. Implement performance counters in the application
   D. Use existing technology investments

8. You are writing the technical specifications document for your solution. The customer needs to understand the need for interoperability with the existing mainframe, which stores its current order data. Which of these concepts are correct?
   A. Ignore existing technologies
   B. Remove the mainframe hardware
   C. Enable optimal deployments
   D. Use existing investments

9. The new application being designed needs to support multiple languages. What must be identified in the technical specifications document on how to prepare the application for globalization?
   A. Update user documentation
   B. Design features that support those cultures and locales
   C. Write code that executes properly in all of the supported cultures and locales
   D. Performance monitoring
Transforming Functional Specifications into Technical Specifications

10. After creating the functional specifications, the customer is asking what the next step is in the process. Identify the purposes of the development plan.
   A. Determine development objectives
   B. Determine vision of project
   C. Document guidelines and standards
   D. Identify design patterns to be used

11. You are creating the technical specifications document. Identify for the support team the purpose of the test plan.
   A. Implement testing
   B. Identify methodologies
   C. Identify tools
   D. Identify versioning and source control standards

12. You are writing the deployment plan for the new application. What are the correct items that are needed to be identified in the deployment plan?
   A. Determine licensing
   B. Architecture
   C. Resource threshold monitoring
   D. Installation instructions

13. You are writing the technical specifications document. Identify the correct items about the technical specifications.
   A. Define scope of work
   B. Define configuration settings
   C. Define user training
   D. Define assembly names

14. You are writing the functional specifications document. The user is trying to understand the purpose of this document. Identify the correct items about the functional specifications document.
   A. Consolidate a common understanding of the business and user
   B. Develop an incomplete product
   C. Break down the problem and modularize the solution logically
   D. Serve as a contract between the team and the customer for what will be delivered
60 Chapter 4: Formalizing Business Requirements into Functional Specifications

15. While writing the functional specifications document, identify the correct risks associated with not creating functional specifications.
   A. Developing a solution that does not completely address the customer requirements
   B. Unable to clearly define customer expectations
   C. Lower risk because it is unnecessary
   D. Unable to validate solution to customer requirements

16. You are writing the functional specifications document. The first step in conceptual design is the analysis phase. Identify the correct items about this phase.
   A. Reviewing the user and business research
   B. Documenting and modeling the context, workflow, task sequence, and environmental relationships
   C. Defining candidate requirements
   D. Validating and testing the improved business processes

17. You are in the process of restating the business requirements gathered in the initial phase of the project. Identify the common characteristics of the restated requirements.
   A. Requirements must be incomplete sentences not using will, may, must, or should.
   B. Requirements should be clear and concise.
   C. Requirements created in a testable manner.
   D. Organized in a hierarchy of related requirements

18. You are in the process of restating the business requirements gathered in the initial phase of the project. During the restatement of requirements, they are grouped into categories. Identify the main categories of requirements.
   A. User
   B. System
   C. Operations
   D. Support

19. The current application is not very secure. The new application will be used by a broader range of users and systems within the organization. You are writing the technical specifications document and need to identify security strategies for the application. Identify the correct security strategies.
   A. Trust external input
   B. Apply the principle of most privilege
C. Rely on tested and proven security systems
D. Default to secure mode

20. You are writing the functional specifications for the new application. The existing application had many security problems. Identify the correct security principles associated with STRIDE.

A. Spoofing identity
B. Repudiation
C. Information tampering
D. Degradation of privileges

LAB QUESTION

Create a C#.NET console application to copy web server log files from remote locations. The application should preprocess the log files and load them into a warehouse table. The information is then generated to rollup information from the warehouse table and populate the report structures. The source servers and file locations will be stored multiple key nodes in the config file that define: A) the web server name, B) the file location of the logs and c) the web site URL that the logs describe traffic for. All temporary files must be destroyed upon completion of processing and all system notifications, success or failure, should be written to the event log. The application config file should define a Panic E-mail key which should be used when a catastrophic failure occurs.

Refine the following requirements:

■ The application will be configured by using a configuration file that will determine the characteristics needed for the application to execute.
■ The logger application will notify the Panic E-mail Address when catastrophic errors occur. All other notifications will be sent to the event log of the running machine.
■ The parsing of the web logs will be handled on a file-by-file basis. This allows for the possibility that certain web servers are not configured correctly and for future expansion. The only files that can be parsed are files that are at least one day old.
■ Storing of resulting processed web logs.
SELF TEST ANSWERS

Functional Specifications

1. ✓ A and B. These options are for scaling up. Scaling up either upgrades components or replaces machines with more powerful machines.
   ✗ C and D. Adding more servers is incorrect because this is a scaling out option.

2. ✓ B, C, and D. Designing processes that do not compete for resources allows for a minimum set of hardware and increases performance. Partitioning resources and activities allows applications to minimize the shared resources needed.
   ✗ A. Synchronous processes are an incorrect process type for scalability; it should be asynchronous.

3. ✓ A, C, and D. The purpose of availability is to keep an application running continuously. This can be performed by reducing planned downtime, using network load balancing, and queuing.
   ✗ B. Clustering is a needed option for availability. Clustering allows applications to shut down a machine in the cluster and still maintain the services needed.

Transforming Requirements into Functional Specifications

4. ✓ C and D. When planning downtime, you want to minimize the time services are not available. Using rolling updates allows for the release of updates in phases. Clustering and load balancing allow machines to be taken down without loss of services.
   ✗ A and B. Taking all the servers off-line makes services completely unavailable and can negatively affect business. Putting untested code into production will eventually lead to downtime for fixing bugs.

5. ✓ B and C. Reliability is concerned with the correctness of the application. Applications that contain errors in code and in operational logic can be very costly to fix and maintain.
   ✗ A and D. Isolating mission-critical applications is an availability design goal. Deploying untested code is one of the major reasons for establishing reliable code.

6. ✓ B and D. Implementing reliability goals for an application requires error handling, quality testing, and reuse of reliable components.
   ✗ A and C. Lack of quality software engineering is a cause of reliability failure.
7. **✓** B and C. Performance requirements are concerned with identifying critical features that need to have metrics assigned to how they perform.  
**✗** A and D. The use of existing technology investments is not a design goal of performance. Applying forms authentication is a security requirement.

8. **✓** C and D. Interoperability is the leveraging of existing hardware and software that can exist in different environments and require various protocols for communication.  
**✗** A and B. Ignoring existing technologies is very costly and time consuming. Removing the mainframe hardware is not an option in this situation; the goal is to preserve the hardware.

9. **✓** B and C. Globalization is the beginning of the process of localization. To create a world-ready application requires the preparation of resource files and identifying features that are culturally correct.  
**✗** A and D. Performance monitoring is clearly an incorrect globalization consideration. Updating user documentation is a factor, but is not correct for this stage of the project.

### Transforming Functional Specifications into Technical Specifications

10. **✓** A, C, and D. The development plan is the road map for the design, standards, and features of the application.  
**✗** B. The vision must be clearly defined before creating the development plan and finalizing the business requirements.

11. **✓** B and C. The test plan identifies what will be tested, how it will be tested, and what types of results are needed.  
**✗** A and D. Identifying versioning and source control standards is a development plan element. Testing happens during the development stage.

12. **✓** A, B, and D. Deployment plans are used to determine what is needed to deliver the product to production, how this will be accomplished, and who is responsible for the deployment and support.  
**✗** C. Resource threshold monitoring is an important technique for performance testing and requirements.

13. **✓** A, B, and D. The technical specifications document identifies what work will be performed and how the environments will be set up.  
**✗** C. The technical specifications are not concerned with user training because this is handled by the deployment plan.
14. A, C, and D. The functional specifications document is generally a contract between the project team and the customer to determine if all features are defined correctly, and is presented in business terms. B. Developing an incomplete product is always a danger associated with not gathering proper requirements.

15. A, B, and D. By not creating the functional specifications, you run the risk of developing a solution that meets the requirements but is unusable or creating a solution then finding out too late that the business processes were incorrect. C. Defining candidate requirements is a step performed during the creation of the functional specifications.

16. A, B, and C. The analysis of the conceptual design is the reexamination of the original requirements and creating a more detailed modeling diagram and requirements. D. Validation and testing the improved business process is a task performed later in the optimization of the conceptual design.

17. B, C, and D. Restating business requirements is the breaking down of requirements into simpler forms and groups. These restated business requirements will be the basis for the test plan. A. The requirements should be complete sentences and use will, may, must, or should.

18. A, B, and C. Requirements are grouped by their functionality, whether they are for the user, for the operation of the application, or for the system. D. Support is a subcategory that can fit under the User category.

19. C and D. Security strategies are designed around the goals of protecting the data and the application. This involves applying the minimal amount of permissions and proven security methods and models. A and B. Trusting external input is always a problem. If the external system has information that can change data incorrectly, it would be very difficult to determine if it was authenticated to do this unless security was involved. The principle of most privilege is incorrect; the correct concept is applying security of least privilege. Give the minimal amount of permissions necessary.

20. A and B. Spoofing identify and repudiation are correct STRIDE principles. C and D. Information tampering and degradation of privileges are incorrect STRIDE principles.
LAB ANSWER

The following is an example of how the requirements should be refined:

- **Req 1.0** The application will be configured by using a configuration file that will determine the characteristics needed for the application to execute.

**User Services**

- 1.1 User Services. Run application from command line and derive parameters from configuration file.

**Business Services**

- 1.2 Implement configuration file.
- 1.2.1 Define key for remote web server locations. Required information includes web server name, location of log files, and web site URL.
- 1.2.2 Define e-mail key for catastrophic failures.

**Data Services**

- 1.3 Define key for data access.
- 1.4 Define functionality for multiple keys in database to hold web server settings.

The logger application will notify the Panic E-mail Address when catastrophic errors occur. All other notifications will be sent to the event log of the running machine.

**User Services**

- 2.0 Verify event log entries.

**Business Services**

- 2.1 Create event log entries on success and failure.
- 2.2 Create e-mails when catastrophic events occur in the application.

The parsing of the web logs will be handled on a file-by-file basis. This allows for the possibility that certain web servers are not configured correctly and for future expansion. The only files that can be parsed are files that are at least one day old.

**Business Services**

- 3.0 Parse web logs based on configuration key information. Each web server instance will have a different name for the respective web log information directory. This information must be contained in the configuration file.
Chapter 4: Formalizing Business Requirements into Functional Specifications

3.1 Reverse lookups must be performed against the IP address to determine location.
Storing of resulting processed web logs.

Business Services

4.0 Store web logs based on URL information.