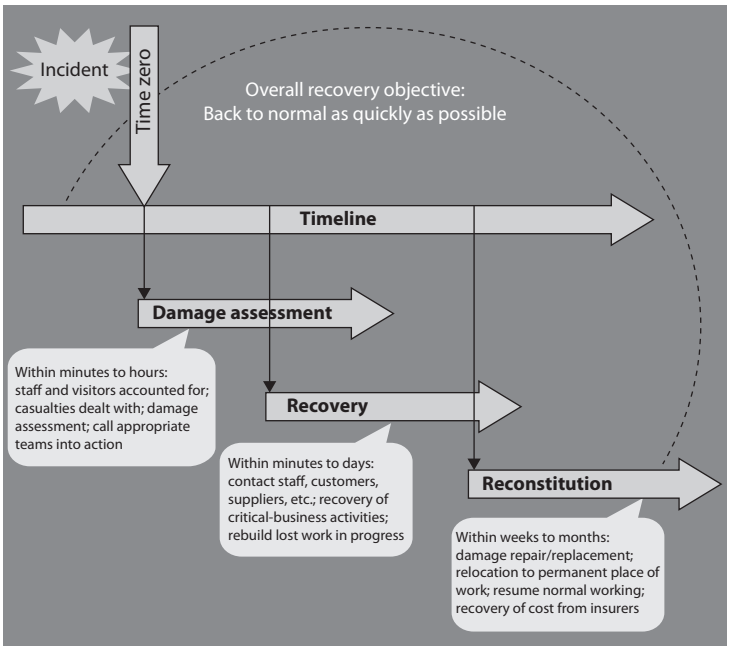
canary in the coal mine. If they survive, then move the more critical components of the organization to the main site.



Training and Awareness

Training your DR team on the execution of a DRP is critical for at least three reasons. First, it allows you to validate that the plan will actually work. If your DR team is doing a walkthrough exercise in response to a fictitious scenario, you'll find out very quickly whether the plan would work or not. If it doesn't work in a training event when the stress level and stakes are low, then there is no chance it would work in a real emergency.

Another reason to train is to ensure that everyone knows what they're supposed to do, when, where, and how. Disasters are stressful, messy affairs and key people may not be thinking clearly. It is important for them to have a familiar routine to fall back on. In a perfect world, you would train often enough for your team to develop "muscle memory" that allows them to automatically do the right things without even thinking.

Lastly, training can help establish that you are exercising due care. This could keep you out of legal trouble in the aftermath of a disaster, particularly if people end up getting hurt. A good plan and evidence of a trained workforce can go a long way to reduce liability if regulators or other investigators come knocking. As always, consult your attorneys to ensure you are meeting all applicable legal and regulatory obligations.

When thinking of training and "muscle memory," you should also consider everyone else in the organization that is not part of the DR team. You want all your staff to have an awareness of the major things they need to do to support DR. This is why many of us conduct fire drills in our facilities: to ensure everyone knows how to get out of the

building and where to assemble if we ever face this particular kind of disaster. There are many types of DR awareness events you can run, but you should at least consider three types of responses that everyone should be aware of: evacuations (e.g., for fires or explosives), shelter-in-place (e.g., for tornadoes or active shooters), and remain-at-home (e.g., for overnight flooding).

Lessons Learned

As mentioned on the first page of this chapter, no battle plan ever survived first contact with the enemy. When you try to execute your DRP in a real disaster, you will find the need to disregard parts of it, make on-the-fly changes to others, and faithfully execute the rest. This is why you should incorporate lessons learned from any actual disasters and actual responses. The DR team should perform a "postmortem" on the response and ensure that necessary changes are made to plans, contracts, personnel, processes, and procedures.

Military organizations collect lessons learned in two steps. The first steps, called a *hotwash*, is a hasty one that happens right after the event is concluded (i.e., restoration is completed). The term comes from the military practice of dousing rifles with very hot water immediately after an engagement to quickly get the worst grit and debris off their weapons. The reason you want to conduct a hotwash right away is that memories will be freshest right after restoring the systems. The idea is not necessarily to figure out how to fix anything, but rather to quickly list as many things that went well or poorly as possible before participants start to forget them.

The second event at which lessons learned are collected in the military is much more deliberate. An after-action review (AAR) happens several days after completion of the DR and allows participants to think things through and start formulating possible ways to do better in the future. The AAR facilitator, ideally armed with the notes from the hotwash, presents each issue that was recorded (good or bad), a brief discussion of it, and then opens the floor for recommendations. Keep in mind that since you're dealing with things that went well or poorly, sometimes the group recommendation will be to "sustain" the issue or, in other words, keep doing things the same way in the future. More frequently, however, there are at least minor tweaks that can improve future performance.

Testing Disaster Recovery Plans

The disaster recovery plan should be tested regularly because environments continually change. Interestingly, many organizations are moving away from the concept of "testing," because a test naturally leads to a pass or fail score, and in the end, that type of score is not very productive. Instead, many organizations are adopting the concept of "exercises," which appear less stressful, better focused, and ultimately more productive to the participants. Each time the DRP is exercised or tested, improvements and efficiencies are generally uncovered, yielding better and better results over time. The responsibility of establishing periodic exercises and the maintenance of the plan should be assigned to a specific person or persons who will have overall ownership responsibilities for the disaster recovery initiatives within the organization.

The maintenance of the DRP should be incorporated into change management procedures. That way, any changes in the environment are reflected in the plan itself.

Tests and disaster recovery exercises should be performed at least once a year. An organization should have no real confidence in a developed plan until it has actually been tested. Exercises prepare personnel for what they may face and provide a controlled environment to learn the tasks expected of them. These exercises also point out issues to the planning team and management that may not have been previously thought about and addressed as part of the planning process. The exercises, in the end, demonstrate whether an organization can actually recover after a disaster.

The exercise should have a predetermined scenario that the organization may indeed be faced with one day. Specific parameters and a scope of the exercise must be worked out before sounding the alarms. The team of testers must agree upon what exactly is getting tested and how to properly determine success or failure. The team must agree upon the timing and duration of the exercise, who will participate in the exercise, who will receive which assignments, and what steps should be taken. Also, the team needs to determine whether hardware, software, personnel, procedures, and communications lines are going to be tested and whether it is all or a subset of these resources that will be included in the event. If the test will include moving some equipment to an alternate site, then transportation, extra equipment, and alternate site readiness must be addressed and assessed.

Most organizations cannot afford to have these exercises interrupt production or productivity, so the exercises may need to take place in sections or at specific times, which will require logistical planning. Written exercise plans should be developed that will test for specific weaknesses in the overall DRP. The first exercises should not include all employees, but rather a small representative sample of the organization. This allows both the planners and the participants to refine the plan. It also allows each part of the organization to learn its roles and responsibilities. Then, larger exercises can take place so overall operations will not be negatively affected.

The people conducting these exercises should expect to encounter problems and mistakes. After all, identifying potential problems and mistakes is why they are conducting the exercises in the first place. An organization would rather have employees make mistakes during an exercise so they can learn from them and perform their tasks more effectively during a real disaster.

NOTE After a disaster, telephone service may not be available. For communications purposes, alternatives should be in place, such as mobile phones or hand-held radios.

A few different types of exercises and tests can be used, each with its own pros and cons. The following sections explain the different types of assessment events.

Checklist Test

In this type of test, copies of the DRP are distributed to the different departments and functional areas for review. This enables each functional manager to review the plan

and indicate if anything has been left out or if some approaches should be modified or deleted. This method ensures that nothing is taken for granted or omitted, as might be the case in a single-department review. Once the departments have reviewed their copies and made suggestions, the planning team then integrates those changes into the master plan.

NOTE The checklist test is also called the desk check test.

Structured Walkthrough Test

In this test, representatives from each department or functional area come together and go over the plan to ensure its accuracy. The group reviews the objectives of the plan; discusses the scope and assumptions of the plan; reviews the organization's reporting structure; and evaluates the testing, maintenance, and training requirements described. This gives the people responsible for making sure a disaster recovery happens effectively and efficiently an opportunity to review what has been decided upon and what is expected of them.

The group walks through different scenarios of the plan from beginning to end to make sure nothing was left out. This also raises the awareness of team members about the recovery procedures.

Tabletop Exercises

Tabletop exercises (TTXs) may or may not happen at a tabletop, but they do not involve a technical control infrastructure. TTXs can happen at an executive level (e.g., C-suite) or at a team level (e.g., SOC), or anywhere in between. The idea is usually to test procedures and ensure they actually do what they're intended to and that everyone knows their role in responding to a disaster. TTXs require relatively few resources apart from deliberate planning by qualified individuals and the undisturbed time and attention of the participants.

After determining the goals of the exercise and vetting them with the senior leadership of the organization, the planning team develops a scenario that touches on the important aspects of the response plan. The idea is normally not to cover every contingency, but to ensure the DR team is able to respond to the likeliest and/or most dangerous scenarios. As they develop the exercise, the planning team considers branches and sequels at every point in the scenario. A *branch* is a point in which the participants may choose one of multiple approaches to respond. If the branches are not carefully managed and controlled, the TTX could wander into uncharted and unproductive directions. Conversely, a *sequel* is a follow-on to a given action in the response. For instance, as part of the response, the strategic communications team may issue statements to the news media. A sequel to that could involve a media outlet challenging the statement, which in turn would require a response by the team. Like branches, sequels must be used carefully to keep the exercise on course. Senior leadership support and good scenario development are critical ingredients to attract and engage the right participants. Like any contest, a TTX is only as good as the folks who show up to play.

EXAM TIP Tabletop exercises are also called read-through exercises.

Simulation Test

This type of test takes a lot more planning and people. In this situation, all employees who participate in operational and support functions, or their representatives, come together to practice executing the disaster recovery plan based on a specific scenario. The scenario is used to test the reaction of each operational and support representative. Again, this is done to ensure specific steps were not left out and that certain threats were not overlooked. It raises the awareness of the people involved.

The exercise includes only those materials that will be available in an actual disaster, to portray a more realistic environment. The simulation test continues up to the point of actual relocation to an offsite facility and actual shipment of replacement equipment.

Parallel Test

In a parallel test, some systems are moved to the alternate site and processing takes place. The results are compared with the regular processing that is done at the original site. This ensures that the specific systems can actually perform adequately at the alternate offsite facility and points out any tweaking or reconfiguring that is necessary.

Full-Interruption Test

This type of test is the most intrusive to regular operations and business productivity. The original site is actually shut down, and processing takes place at the alternate site. The recovery team fulfills its obligations in preparing the systems and environment for the alternate site. All processing is done only on devices at the alternate offsite facility.

This is a full-blown exercise that takes a lot of planning and coordination, but it can reveal many holes in the plan that need to be fixed before an actual disaster hits. Full-interruption tests should be performed only after all other types of tests have been successful. They are the riskiest type and can impact the business in very serious and devastating ways if not managed properly; therefore, senior management approval needs to be obtained prior to performing full-interruption tests.

The type of organization and its goals will dictate what approach to the training exercise is most effective. Each organization may have a different approach and unique aspects. If detailed planning methods and processes are going to be taught, then specific training may be required rather than general training that provides an overview. Higher-quality training will result in an increase in employee interest and commitment.

During and after each type of test, a record of the significant events should be documented and reported to management so it is aware of all outcomes of the test.

Other Types of Training

Other types of training that employees need in addition to disaster recovery training include first aid and cardiac pulmonary resuscitation (CPR), how to properly use a fire extinguisher, evacuation routes and crowd control methods, emergency communications procedures, and how to properly shut down equipment in different types of disasters.

The more technical employees may need training on how to redistribute network resources and how to use different telecommunications lines if the main one goes down. They may need to know about redundant power supplies and be trained and tested on the procedures for moving critical systems from one power supply to the next.

Business Continuity

When a disaster strikes, ensuring that the organization is able to continue its operations requires more than simply restoring data from backups. Also necessary are the detailed procedures that outline the activities to keep the critical systems available and ensure that operations and processing are not interrupted. Business continuity planning defines what should take place during and after an incident. Actions that are required to take place for emergency response, continuity of operations, and dealing with major outages must be documented and readily available to the operations staff. There should be at least two instances of these documents: the original that is kept on-site and a copy that is at an offsite location.

BC plans should not be trusted until they have been tested. Organizations should carry out exercises to ensure that the staff fully understands their responsibilities and how to carry them out. We already covered the various types of exercises that can be used to test plans and staff earlier in this chapter when we discussed DR. Another issue to consider is how to keep these plans up to date. As our dynamic, networked environments change, so must our plans on how to rescue them when necessary.

Although in the security industry "contingency planning" and "business continuity planning (BCP)" are commonly used interchangeably, it is important that you understand the actual difference for the CISSP exam. BCP addresses how to keep the organization in business after a major disruption takes place. It is about the survivability of the organization and making sure that critical functions can still take place even after a disaster. Contingency plans address how to deal with small incidents that do not qualify as disasters, as in power outages, server failures, a down communication link to the Internet, or the corruption of software. Organizations must be ready to deal with both large and small issues that they may encounter.

EXAM TIP BCP is broad in scope and deals with survival of the organization. Contingency plans are narrow in scope and deal with specific issues.

As a security professional you will most likely not be in charge of BCP, but you should most certainly be an active participant in developing the BCP. You will also be involved in BC exercises and may even be a lead in those that focus on information systems. To effectively participate in BC planning and exercises, you should be familiar with the BCP life cycle, how to ensure continuous availability of critical information systems, and the particular requirements of the end-user environments. We look at these in the following sections.

BCP Life Cycle

Remember that most organizations aren't static, but change, often rapidly, as do the conditions under which they must operate. Thus, BCP should be considered a life cycle in order to deal with the constant and inevitable change that will affect it. Understanding and

maintaining each step of the BCP life cycle is critical to ensuring that the BC plan remains useful to the organization. The BCP life cycle is outlined in Figure 23-7.

Note that this life cycle has two modes: normal management (shown in the top half of Figure 23-7) and incident management (shown in the bottom half). In the normal mode, the focus of the BC team is on ensuring preparedness. Obviously, we want to start

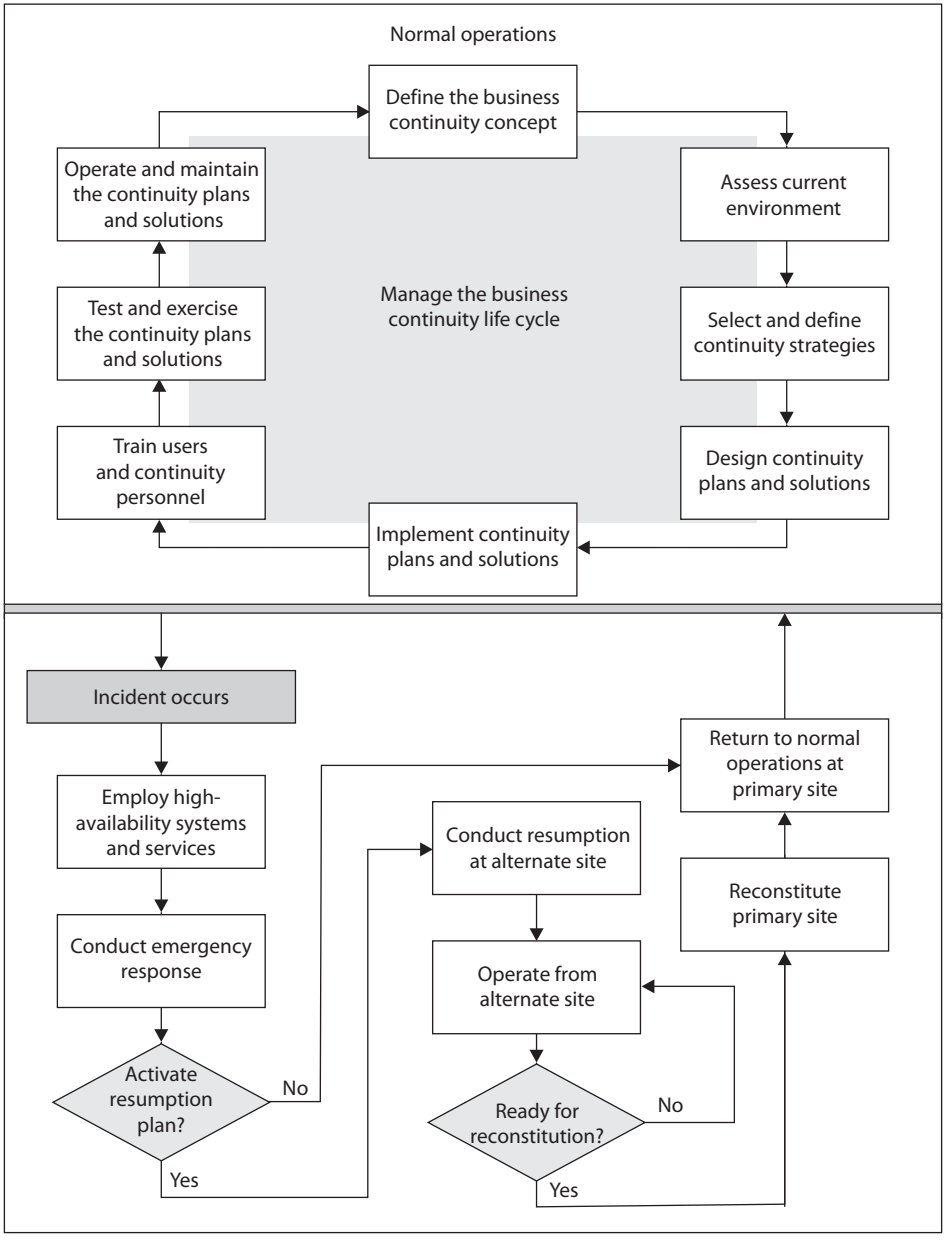


Figure 23-7 BCP life cycle

with a clearly defined concept for what business continuity means for the organization. What are the critical business functions that must continue to operate regardless of what incident happens? What are the minimum levels of performance that are acceptable for these functions?

Once we define the BC concept, we can take a look at the current environment and consider the strategies that would allow continuity of operations under a variety of conditions. It is important to consider that, unlike DR planning, not every type of incident covered in BCP involves loss of IT capabilities. Many organizations suffered tremendously in 2020 because their BCP didn't account for a global pandemic in which many (or even all) staff members would have to work from home for extended periods of time. Information systems are certainly an important part of the continuity strategies, plans, and solutions, but the scope of the BCP is much broader than that of the DRP.

The BC plan is only useful if the organization in general, and the BC team in particular, knows how to execute the plan. This requires periodic training, tests, and exercises to ensure that both the plan and the staff are able to keep the business going no matter what comes their way. As we find gaps and opportunities for improvement, we get to redefine our BCP concept and start another run through the cycle. This continuous improvement is key to being able to switch into incident management mode (at the bottom of Figure 23-7) when needed and execute the BC plan (and, potentially, the DR plan) to keep the business going.

Information Systems Availability

Our main job as CISSPs in the BCP life cycle is to ensure the continuous availability of organizational information systems. To this end, we should ensure the BCP includes backup solutions for the following:

- Network and computer equipment
- Voice and data communications resources
- Human resources
- Transportation of equipment and personnel
- Environment issues (HVAC)
- Data and personnel security issues
- Supplies (paper, forms, cabling, and so on)
- Documentation

The BCP team must understand the organization's current technical environment. This means the planners have to know the intimate details of the network, communications technologies, computers, network equipment, and software requirements that are necessary to get the critical functions up and running. What is surprising to some people is that many organizations do not *totally* understand how their network is configured and how it actually works, because the network may have been established 10 to 15 years ago and has kept growing and changing under different administrators and personnel.

Outsourcing

Part of the planned response to a disaster may be to outsource some of the affected activities to another organization. Organizations do outsource activities—help-desk services, manufacturing, legal advice—all the time, so why not important functions affected by a disaster? Some companies specialize in disaster response and continuity planning and can act as expert consultants.

That is all well and good. However, be aware that your organization is still ultimately responsible for the continuity of a product or service that is outsourced. Clients and customers will expect the organization to ensure continuity of its products and services, either by itself or by having chosen the right outside vendors to provide the products and services. If outside vendors are brought in, the active participation of key in-house managers in their work is still essential. They still need to supervise the work of the outside vendors.

This same concern applies to normal, third-party suppliers of goods and services to the organization. Any BCP should take them into account as well. Note that the process for evaluating an outsourced company for BCP is like that for evaluating the organization itself. The organization must make sure that the outsourced company is financially viable and has its own solid BCP.

The organization can take the following steps to better ensure the continuity of its outsourcing:

- Make the ability of such companies to reliably assure continuity of products and services part of any work proposals.
- Make sure that business continuity planning is included in contracts with such companies, and that their responsibilities and levels of service are clearly spelled out.
- Draw up realistic and reasonable service levels that the outsourced firm will meet during an incident.
- If possible, have the outsourcing companies take part in BCP awareness programs, training, and testing.

The goal is to make the supply of goods and services from outsources as resilient as possible in the wake of a disaster.

New devices are added, new computers are added, new software packages are added, VoIP may have been integrated, and the DMZ may have been split up into three DMZs, with an extranet for the organization's partners. Maybe a company bought and merged with another company and network. Over ten years, a number of technology refreshes most likely have taken place, and the individuals who are maintaining the environment now likely are not the same people who built it ten years ago. Many IT departments experience extensive employee turnover every five years. And most organizational network

schematics are notoriously out of date because everyone is busy with their current tasks (or will come up with new tasks just to get out of having to update the schematic).

So the BCP team has to make sure that if the networked environment is partially or totally destroyed, the recovery team has the knowledge and skill to properly rebuild it.

NOTE Many organizations use VoIP, which means that if the network goes down, network and voice capability are unavailable. The BCP team should address the possible need of redundant voice systems.

The BCP team needs to incorporate into the BCP several things that are commonly overlooked, such as hardware replacements, software products, documentation, environmental needs, and human resources.

Hardware Backups

The BCP needs to identify the equipment required to keep the critical functions up and running. This may include servers, user workstations, routers, switches, tape backup devices, and more. The needed inventory may seem simple enough, but as they say, the devil is in the details. If the recovery team is planning to use images to rebuild newly purchased servers and workstations because the original ones were destroyed, for example, will the images work on the new computers? Using images instead of building systems from scratch can be a time-saving task, unless the team finds out that the replacement equipment is a newer version and thus the images cannot be used. The BCP should plan for the recovery team to use the organization's current images, but also have a manual process of how to build each critical system from scratch with the necessary configurations.

The BCP also needs to be based on accurate estimates of how long it will take for new equipment to arrive. For example, if the organization has identified Dell as its equipment replacement supplier, how long will it take this vendor to send 20 servers and 30 workstations to the offsite facility? After a disaster hits, the organization could be in its offsite facility only to find that its equipment will take three weeks to be delivered. So, the SLA for the identified vendors needs to be investigated to make sure the organization is not further damaged by delays. Once the parameters of the SLA are understood, the BCP team must make a decision between depending upon the vendor and purchasing redundant systems and storing them as backups in case the primary equipment is destroyed.

As described earlier, when potential organizational risks are identified, it is better to take preventive steps to reduce the potential damage. After the calculation of the MTD values, the team will know how long the organization can operate without a specific device. This data should be used to make the decision on whether the organization should depend on the vendor's SLA or make readily available a hot-swappable redundant system. If the organization will lose $50,000 per hour if a particular server goes down, then the team should elect to implement redundant systems and technology.

If an organization is using any legacy computers and hardware and a disaster hits tomorrow, where would it find replacements for this legacy equipment? The BCP

team should identify legacy devices and understand the risk the organization is facing if replacements are unavailable. This finding has caused many organizations to move from legacy systems to commercial off-the-shelf (COTS) products to ensure that timely replacement is possible.

Software Backups

Most organizations' IT departments have their array of software disks and licensing information here or there—or possibly in one centralized location. If the facility were destroyed and the IT department's current environment had to be rebuilt, how would it gain access to these software packages? The BCP team should make sure to have an inventory of the necessary software required for mission-critical functions and have backup copies at an offsite facility. Hardware is usually not worth much to an organization without the software required to run on it. The software that needs to be backed up can be in the form of applications, utilities, databases, and operating systems. The business continuity plan must have provisions to back up and protect these items along with hardware and data.

It is common for organizations to work with software developers to create customized software programs. For example, in the banking world, individual financial institutions need software that enables their bank tellers to interact with accounts, hold account information in databases and mainframes, provide online banking, carry out data replication, and perform a thousand other types of bank-like functionalities. This specialized type of software is developed and available through a handful of software vendors that specialize in this market. When bank A purchases this type of software for all of its branches, the software has to be specially customized for its environment and needs. Once this banking software is installed, the whole organization depends upon it for its minute-by-minute activities.

When bank A receives the specialized and customized banking software from the software vendor, bank A does not receive the source code. Instead, the software vendor provides bank A with a compiled version. Now, what if this software vendor goes out of business because of a disaster or bankruptcy? Then bank A will require a new vendor to maintain and update this banking software; thus, the new vendor will need access to the source code.

The protection mechanism that bank A should implement is called *software escrow*, in which a third party holds the source code, backups of the compiled code, manuals, and other supporting materials. A contract between the software vendor, customer, and third party outlines who can do what, and when, with the source code. This contract usually states that the customer can have access to the source code only if and when the vendor goes out of business, is unable to carry out stated responsibilities, or is in breach of the original contract. If any of these activities takes place, then the customer is protected because it can still gain access to the source code and other materials through the third-party escrow agent.

Many organizations have been crippled by not implementing software escrow. They paid a software vendor to develop specialized software, and when the software vendor went belly up, the organizations did not have access to the code that their systems ran on.

End-User Environment

Because the end users are usually the worker bees of an organization, they must be provided a functioning environment as soon as possible after a disaster hits. This means that the BCP team must understand the current operational and technical functioning environment and examine critical pieces so they can replicate them.

In most situations, after a disaster, only a skeleton crew is put back to work. The BCP committee has previously identified the most critical functions of the organization during the analysis stage, and the employees who carry out those functions must be put back to work first. So the recovery process for the user environment should be laid out in different stages. The first stage is to get the most critical departments back online, the next stage is to get the second most important back online, and so on.

The BCP team needs to identify user requirements, such as whether users can work on stand-alone PCs or need to be connected in a network to fulfill specific tasks. For example, in a financial institution, users who work on stand-alone PCs might be able to accomplish some small tasks like filling out account forms, word processing, and accounting tasks, but they might need to be connected to a host system to update customer profiles and to interact with the database.

The BCP team also needs to identify how current automated tasks can be carried out manually if that becomes necessary. If the network is going to be down for 12 hours, could the necessary tasks be accomplished through traditional pen-and-paper methods? If the Internet connection is going to be down for five hours, could the necessary communications take place through phone calls? Instead of transmitting data through the internal mail system, could couriers be used to run information back and forth? Today, we are extremely dependent upon technology, but we often take for granted that it will always be there for us to use. It is up to the BCP team to realize that technology may be unavailable for a period of time and to come up with solutions for those situations.

EXAM TIP As a CISSP, your role in business continuity planning is most likely to be that of an active participant, not to lead it. BCP questions in the exam will be written with this in mind.

Chapter Review

There are four key take-aways in this chapter. The first is that you need to be able to identify and implement strategies that will enable your organization to recover from any disaster, supporting your organization's continuity of operations. Leveraging these strategies, you develop a detailed plan that includes the specific processes that the organization (and particularly the IT and security teams) will execute to recover from specific types of disasters. Thirdly, you have to know how to train your DR team to execute the plan flawlessly, even in the chaos of an actual disaster. This includes ensuring that everyone in the organization is aware of their role in the recovery efforts. Finally, the DRP is the cornerstone of the BCP, so you will be called upon to participate in broader business continuity planning and exercises, even if you are not in charge of that effort.

Quick Review

- Disaster recovery (DR) is the set of practices that enables an organization to minimize loss of, and restore, mission-critical technology infrastructure after a catastrophic incident.
- Business continuity (BC) is the set of practices that enables an organization to continue performing its critical functions through and after any disruptive event.
- The recovery time objective (RTO) is the maximum time period within which a mission-critical system must be restored to a designated service level after a disaster to avoid unacceptable consequences associated with a break in business continuity.
- The work recovery time (WRT) is the maximum amount of time available for certifying the functionality and integrity of restored systems and data so they can be put back into production.
- The recovery point objective (RPO) is the acceptable amount of data loss measured in time.
- The four commonly used data backup strategies are direct-attached storage, network-attached storage, cloud storage, and offline media.
- Electronic vaulting makes copies of files as they are modified and periodically transmits them to an offsite backup site.
- Remote journaling moves transaction logs to an offsite facility for database recovery, where only the reapplication of a series of changes to individual records is required to resynchronize the database.
- Offsite backup locations can supply hot, warm, or cold sites.
- A hot site is fully configured with hardware, software, and environmental needs. It can usually be up and running in a matter of hours. It is the most expensive option, but some organizations cannot be out of business longer than a day without very detrimental results.
- A warm site may have some computers, but it does have some peripheral devices, such as disk drives, controllers, and tape drives. This option is less expensive than a hot site, but takes more effort and time to become operational.
- A cold site is just a building with power, raised floors, and utilities. No devices are available. This is the cheapest of the three options, but can take weeks to get up and operational.
- In a reciprocal agreement, one organization agrees to allow another organization to use its facilities in case of a disaster, and vice versa. Reciprocal agreements are very tricky to implement and may be unenforceable. However, they offer a relatively cheap offsite option and are sometimes the only choice.
- A redundant (or mirrored) site is equipped and configured exactly like the primary site and is completely synchronized, ready to become the primary site at a moment's notice.

- High availability (HA) is a combination of technologies and processes that work together to ensure that some specific thing is up and running most of the time.
- Quality of service (QoS) defines minimum acceptable performance characteristics of a particular service, such as response time, CPU utilization, or network bandwidth utilization.
- Fault tolerance is the capability of a technology to continue to operate as expected even if something unexpected takes place (a fault).
- Resilience means that the system continues to function, albeit in a degraded fashion, when a fault is encountered.
- When returning to the original site after a disaster, the least critical organizational units should go back first.
- Disaster recovery plans can be tested through checklist tests, structured walkthroughs, tabletop exercises, simulation tests, parallel tests, or full-interruption tests.
- Business continuity planning addresses how to keep the organization in business after a major disruption takes place, but it is important to note that the scope is much broader than that of disaster recovery.
- The BCP life cycle includes developing the BC concept; assessing the current environment; implementing continuity strategies, plans, and solutions; training the staff; and testing, exercising, and maintaining the plans and solutions.
- An important part of the business continuity plan is to communicate its requirements and procedures to all employees.

Questions

Please remember that these questions are formatted and asked in a certain way for a reason. Keep in mind that the CISSP exam is asking questions at a conceptual level. Questions may not always have the perfect answer, and the candidate is advised against always looking for the perfect answer. Instead, the candidate should look for the best answer in the list.

1. Which best describes a hot-site facility versus a warm- or cold-site facility?

A. A site that has disk drives, controllers, and tape drives

B. A site that has all necessary PCs, servers, and telecommunications

C. A site that has wiring, central air-conditioning, and raised flooring

D. A mobile site that can be brought to the organization's parking lot

2. Which of the following describes a cold site?

A. Fully equipped and operational in a few hours

B. Partially equipped with data processing equipment

C. Expensive and fully configured

D. Provides environmental measures but no equipment

3. Which is the best description of remote journaling?
   - **A.** Backing up bulk data to an offsite facility
   - **B.** Backing up transaction logs to an offsite facility
   - **C.** Capturing and saving transactions to two mirrored servers in-house
   - **D.** Capturing and saving transactions to different media types
4. Which of the following does not describe a reciprocal agreement?
   - **A.** The agreement is enforceable.
   - **B.** It is a cheap solution.
   - **C.** It may be able to be implemented right after a disaster.
   - **D.** It could overwhelm a current data processing site.
5. If a system is fault tolerant, what would you expect it to do?
   - **A.** Continue to operate as expected even if something unexpected takes place
   - **B.** Continue to function in a degraded fashion
   - **C.** Tolerate outages caused by known faults
   - **D.** Raise an alarm, but tolerate an outage caused by any fault
6. Which of the following approaches to testing your disaster recovery plan would be least desirable if you had to maintain high availability of over 99.999 percent?
   - **A.** Checklist test
   - **B.** Parallel test
   - **C.** Full-interruption test
   - **D.** Structured walkthrough test

Use the following scenario to answer Questions 7–10. You are the CISO of a small research and development (R&D) company and realize that you don't have a disaster recovery plan (DRP). The projects your organization handles are extremely sensitive and, despite having a very limited budget, you have to bring the risk of project data being lost as close to zero as you can. Recovery time is not as critical because you bill your work based on monthly deliverables and have some leeway at your disposal. Because of the sensitivity of your work, remote working is frowned upon and you keep your research data on local servers (including Exchange for e-mail, Mattermost for group chat, and Apache for web) at your headquarters (and only) site.

7. Which recovery site strategy would be best for you to consider?
   - **A.** Reciprocal agreement
   - **B.** Hot site
   - **C.** Warm site
   - **D.** Cold site

8. Which of the following recovery site characteristics would be best for your organization?
   - **A.** As close to headquarters as possible within budgetary constraints
   - **B.** 100 miles away from headquarters, on a different power grid
   - **C.** 15 miles away from headquarters on a different power grid
   - **D.** As far away from headquarters as possible
9. Which data backup storage strategy would you want to implement?
   - **A.** Direct-attached storage
   - **B.** Network-attached storage
   - **C.** Offline media
   - **D.** Cloud storage
10. Which of the following would be the best way to communicate with all members of the organization in the event of a disaster that takes out your site?
   - **A.** Internal Mattermost channel
   - **B.** External Slack channel
   - **C.** Exchange e-mail
   - **D.** Call trees

Answers

1. **B.** A hot site is a facility that is fully equipped and properly configured so that it can be up and running within hours to get an organization back into production. Answer B gives the best definition of a fully functional environment.
2. **D.** A cold site only provides environmental measures—wiring, HVAC, raised floors—basically a shell of a building and no more.
3. **B.** Remote journaling is a technology used to transmit data to an offsite facility, but this usually only includes moving the journal or transaction logs to the offsite facility, not the actual files.
4. **A.** A reciprocal agreement is not enforceable, meaning that the organization that agreed to let the damaged organization work out of its facility can decide not to allow this to take place. A reciprocal agreement is a better secondary backup option if the original plan falls through.
5. **A.** Fault tolerance is the capability of a technology to continue to operate as expected even if something unexpected takes place (a fault), with no degradations or outages.
6. **C.** A full-interruption test is the most intrusive to regular operations and business productivity. The original site is actually shut down, and processing takes place at the alternate site. This is almost guaranteed to exceed your allowed downtime unless everything went extremely well.

7. **D.** Because you are working on a tight budget and have the luxury of recovery time, you want to consider the least expensive option. A reciprocal agreement would be ideal except for the sensitivity of your data, which could not be shared with a similar organization (that could, presumably, be a competitor at some point). The next option (cost-wise) is a cold site, which would work in the given scenario.
8. **C.** An ideal recovery site would be on a different power grid to minimize the risk that power will be out on both sites, but close enough for employees to commute. This second point is important because, due to the sensitivity of your work, your organization has a low tolerance for remote work.
9. **C.** Since your data is critical enough that you have to bring the risk of it being lost as close to zero as you can, you would want to use offline media such as tape backups, optical discs, or even external drives that are disconnected after each backup (and potentially removed offsite). This is the slowest and most expensive approach, but is also the most resistant to attacks.
10. **B.** If your site is taken out, you would lose both Exchange and Mattermost since those servers are hosted locally. Call trees only work well for initial notification, leaving an externally hosted Slack channel as the best option. This would require your staff to be aware of this means of communication and have accounts created before the disaster.

PART VIII

Software Development Security

This page intentionally left blank

CHAPTER 24

Software Development

This chapter presents the following:

- Software development life cycle
- Development methodologies
- Operation and maintenance
- Maturity models

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

—John F. Woods

Software is usually developed with a strong focus on functionality, not security. In many cases, security controls are bolted on as an afterthought (if at all). To get the best of both worlds, security and functionality have to be designed and integrated at each phase of the software development life cycle. Security should be interwoven into the core of a software product and provide protection at the necessary layers. This is a better approach than trying to develop a front end or wrapper that may reduce the overall functionality and leave security holes when the software has to be integrated into a production environment.

Before we get too deep into secure software development, however, we have to develop a shared understanding of how code is developed in the first place. In this chapter we will cover the complex world of software development so that we can understand the bad things that can happen when security is not interwoven into products properly (discussed in Chapter 25).

Software Development Life Cycle

The life cycle of software development deals with putting repeatable and predictable processes in place that help ensure functionality, cost, quality, and delivery schedule requirements are met. So instead of winging it and just starting to develop code for a project, how can we make sure we build the best software product possible?

Several *software development life cycle (SDLC)* models have been developed over the years, which we will cover later in this section, but the crux of each model deals with the following phases:

- **Requirements gathering** Determining *why* to create this software, *what* the software will do, and *for whom* the software will be created
- **Design** Encapsulating into a functional design *how* the software will accomplish the requirements
- **Development** Programming software code to meet specifications laid out in the design phase and integrating that code with existing systems and/or libraries
- **Testing** Verifying and validating software to ensure that the software works as planned and that goals are met
- **Operations and maintenance** Deploying the software and then ensuring that it is properly configured, patched, and monitored

EXAM TIP You don't need to memorize the phases of the SDLC. We discuss them here so you understand all the tasks that go into developing software and how to integrate security throughout the whole cycle.

In the following sections we will cover the different phases that make up an SDLC model and some specific items about each phase that are important to understand.

Software Development Roles

The specific roles within a software development team will vary based on the methodology being used, the maturity of the organization, and the size of the project (to name just a few parameters). Typically, however, a team has at least the following roles:

- **Project manager (PM)** This role has overall responsibility for the software development project, particularly with regard to cost, schedule, performance, and risk.
- **Team leads** It is rare for software projects to be tackled by a single team, so we usually divide them up and assign a good developer to lead each part.
- **Architect** Sometimes called a tech lead, this role figures out what technologies to use internally or when interfacing with external systems.
- **Software engineer** The people who actually write the programming code are oftentimes specialists in either frontends (e.g., user interfaces) or various types of backends (e.g., business logic, databases). Engineers that can do all of this are called full-stack developers.
- **Quality assurance (QA)** Whether this is a single person or an entire team, this role implements and runs testing processes that detect software defects as early as possible.

Keep in mind that the discussion that follows covers phases that may happen repeatedly and in limited scope depending on the development methodology being used. Before we get into the phases of the SDLC, let's take a brief look at the glue that holds them together: project management.

Project Management

Many developers know that good project management keeps the project moving in the right direction, allocates the necessary resources, provides the necessary leadership, and hopes for the best but plans for the worst. Project management processes should be put into place to make sure the software development project executes each life-cycle phase properly. Project management is an important part of product development, and security management is an important part of project management.

The project manager draws up a security plan at the beginning of a development project and integrates it into the functional plan to ensure that security is not overlooked. This plan will probably be broad and should refer to documented references for more detailed information. The references could include computer standards (RFCs, IEEE standards, and best practices), documents developed in previous projects, security policies, accreditation statements, incident-handling plans, and national or international guidelines. This helps ensure that the plan stays on target.

The security plan should have a life cycle of its own. It will need to be added to, subtracted from, and explained in more detail as the project continues. Keeping the security plan up to date for future reference is important, because losing track of actions, activities, and decisions is very easy once a large and complex project gets underway.

The security plan and project management activities could be scrutinized later, particularly if a vulnerability causes losses to a third party, so we should document security-related decisions. Being able to demonstrate that security was fully considered in each phase of the SDLC can prove that the team exercised due care and this, in turn, can mitigate future liabilities. To this end, the documentation must accurately reflect how the product was built and how it is supposed to operate once implemented into an environment.

If a software product is being developed for a specific customer, it is common for a *Statement of Work (SOW)* to be developed, which describes the product and customer requirements. A detailed SOW helps to ensure that all stakeholders understand these requirements and don't make any undocumented assumptions.

Sticking to what is outlined in the SOW is important so that *scope creep* does not take place. If the scope of a project continually extends (creeps) in an uncontrollable manner, the project may never end, not meet its goals, run out of funding, or all of the foregoing. If the customer wants to modify its requirements, it is important that the SOW is updated and funding is properly reviewed.

A *work breakdown structure (WBS)* is a project management tool used to define and group a project's individual work elements in an organized manner. It is a deliberate decomposition of the project into tasks and subtasks that result in clearly defined deliverables. The SDLC should be illustrated in a WBS format, so that each phase is properly addressed.

Requirements Gathering Phase

This is the phase in which everyone involved in the software development project attempts to understand why the project is needed and what the scope of the project entails. Typically, either a specific customer needs a new application or a demand for the product exists in the market. During this phase, the software development team examines the software's requirements and proposed functionality, engages in brainstorming sessions, and reviews obvious restrictions.

A conceptual definition of the project should be initiated and developed to ensure everyone is on the right page and that this is a proper product to develop. This phase could include evaluating products currently on the market and identifying any demands not being met by current vendors. This definition could also be a direct request for a specific product from a current or future customer.

Typically, the following tasks should be accomplished in this phase:

- Requirements gathering (including security ones)
- Security risk assessment
- Privacy risk assessment
- Risk-level acceptance

The security requirements of the product should be defined in the categories of availability, integrity, and confidentiality. What type of security is required for the software product and to what degree? Some of these requirements may come from applicable external regulations. For example, if the application will deal with payment cards, PCI DSS will dictate some requirements, such as encryption for card information.

An initial security risk assessment should be carried out to identify the potential threats and their associated consequences. This process usually involves asking many, many questions to elicit and document the laundry list of vulnerabilities and threats, the probability of these vulnerabilities being exploited, and the outcome if one of these threats actually becomes real and a compromise takes place. The questions vary from product to product—such as its intended purpose, the expected environment it will be implemented in, the personnel involved, and the types of businesses that would purchase and use the product.

The sensitivity level of the data that many software products store and process has only increased in importance over the years. After a *privacy risk assessment*, a *privacy impact rating* can be assigned, which indicates the sensitivity level of the data that will be processed or accessible. Some software vendors incorporate the following privacy impact ratings in their software development assessment processes:

- **P1, High Privacy Risk** The feature, product, or service stores or transfers personally identifiable information (PII), monitors the user with an ongoing transfer of anonymous data, changes settings or file type associations, or installs software.
- **P2, Moderate Privacy Risk** The sole behavior that affects privacy in the feature, product, or service is a one-time, user-initiated, anonymous data transfer (e.g., the user clicks a link and is directed to a website).

- **P3, Low Privacy Risk** No behaviors exist within the feature, product, or service that affect privacy. No anonymous or personal data is transferred, no PII is stored on the machine, no settings are changed on the user's behalf, and no software is installed.

The software vendor can develop its own privacy impact ratings and their associated definitions. As of this writing there are several formal approaches to conducting a privacy risk assessment, but none stands out as "the" standardized approach to defining a methodology for an assessment or these rating types, but as privacy increases in importance, we might see more standardization in these ratings and associated metrics.

The team tasked with documenting the requirements must understand the criteria for risk-level acceptance to make sure that mitigation efforts satisfy these criteria. Which risks are acceptable will depend on the results of the security and privacy risk assessments. The evaluated threats and vulnerabilities are used to estimate the cost/benefit ratios of the different security countermeasures. The level of each security attribute should be focused upon so that a clear direction on security controls can begin to take shape and can be integrated into the design and development phases.

The end state of the requirements gathering phase is typically a document called the Software (or System) Requirements Specification (SRS), which describes what the software will do and how it will perform. These two high-level objectives are also known as functional and nonfunctional requirements. A *functional requirement* describes a feature of the software system, such as reporting product inventories or processing customer orders. A *nonfunctional requirement* describes performance standards, such as the minimum number of simultaneous user sessions or the maximum response time for a query. Nonfunctional requirements also include security requirements, such as what data must be encrypted and what the acceptable cryptosystems are. The SRS, in a way, is a checklist that the software development team will use to develop the software and the customer will use to accept it.

The Unified Modeling Language (UML) is a common language used to graphically describe all aspects of software development. We will revisit it throughout the different phases, but in terms of software requirements, it allows us to capture both functional and nonfunctional requirements with use case diagrams (UCDs). We already saw these in Chapter 18 when we discussed testing of technical controls. If you look back to Figure 18-3, each use case (shown as verb phrases inside ovals) represents a high-level functional requirement. The associations can capture nonfunctional requirements through special labels, or these requirements can be spelled out in an accompanying use case description.

Design Phase

Once the requirements are formally documented, the software development team can begin figuring out how they will go about satisfying them. This is the phase that starts to map theory to reality. The theory encompasses all the requirements that were identified in the previous phase, and the design outlines how the product is actually going to accomplish these requirements.

Some organizations skip the design phase, but this can cause major delays and redevelopment efforts down the road because a broad vision of the product needs to be understood before looking strictly at the details. Instead, software development teams should develop written plans for how they will build software that satisfies each requirement. This plan usually comprises three different but interrelated models:

- **Informational model** Dictates the type of information to be processed and how it will move around the software system
- **Functional model** Outlines the tasks and functions the application needs to carry out and how they are sequenced and synchronized
- **Behavioral model** Explains the states the application will be in during and after specific transitions take place

For example, consider an antimalware software application. Its informational model would dictate how it processes information, such as virus signatures, modified system files, checksums on critical files, and virus activity. Its functional model would dictate how it scans a hard drive, checks e-mail for known virus signatures, monitors critical system files, and updates itself. Its behavioral model would indicate that when the system starts up, the antimalware software application will scan the hard drive and memory segments. The computer coming online would be the event that changes the state of the application. If it finds a virus, the application would change state and deal with the virus appropriately. Each state must be accounted for to ensure that the product does not go into an insecure state and act in an unpredictable way.

The data from the informational, functional, and behavioral models is incorporated into the software design document, which includes the data, architectural, and procedural design, as shown in Figure 24-1.

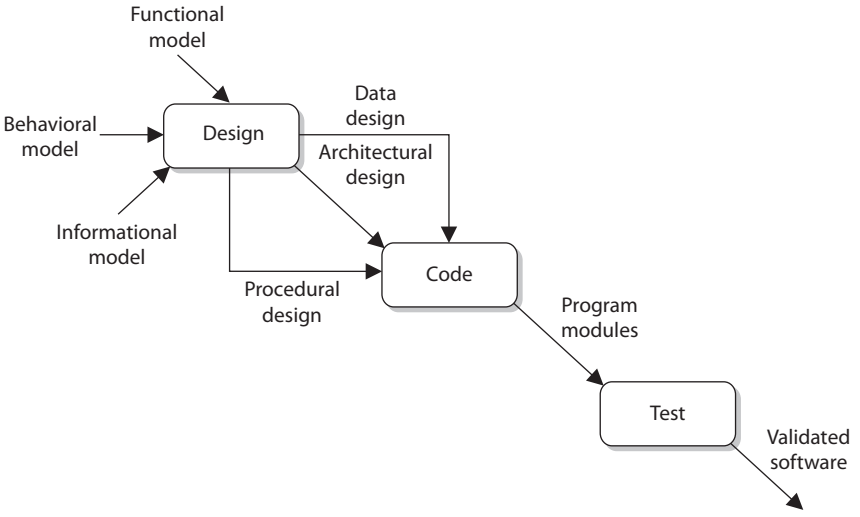


Figure 24-1 Information from three models can go into the design.

From a security point of view, the following items should also be accomplished in the design phase:

- Attack surface analysis
- Threat modeling

An *attack surface* is what is available to be used by an attacker against the product itself. As an analogy, if you were wearing a suit of armor and it covered only half of your body, the other half would be your vulnerable attack surface. Before you went into battle, you would want to reduce this attack surface by covering your body with as much protective armor as possible. The same can be said about software. The software development team should reduce the attack surface as much as possible because the greater the attack surface of software, the more avenues for the attacker; and hence, the greater the likelihood of a successful compromise.

The aim of an *attack surface analysis* is to identify and reduce the amount of code and functionality accessible to untrusted users. The basic strategies of attack surface reduction are to reduce the amount of code running, reduce entry points available to untrusted users, reduce privilege levels as much as possible, and eliminate unnecessary services. Attack surface analysis is generally carried out through specialized tools to enumerate different parts of a product and aggregate their findings into a numeral value. Attack surface analyzers scrutinize files, Registry keys, memory data, session information, processes, and services details. A sample attack surface report is shown in Figure 24-2.

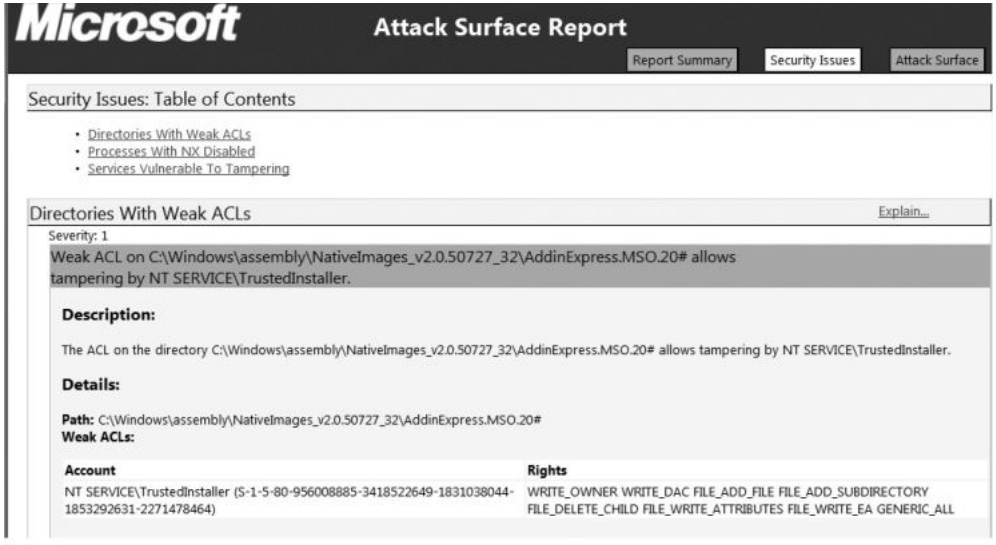


Figure 24-2 Attack surface analysis result

Threat modeling, which we covered in detail in Chapter 9 in the context of risk management, is a systematic approach used to understand how different threats could be realized and how a successful compromise could take place. As a hypothetical example, if you were responsible for ensuring that the government building in which you work is safe from terrorist attacks, you would run through scenarios that terrorists would most likely carry out so that you fully understand how to protect the facility and the people within it. You could think through how someone could bring a bomb into the building, and then you would better understand the screening activities that need to take place at each entry point. A scenario of someone running a car into the building would bring up the idea of implementing bollards around the sensitive portions of the facility. The scenario of terrorists entering sensitive locations in the facility (data center, CEO office) would help illustrate the layers of physical access controls that should be implemented.

These same scenario-based exercises should take place during the design phase of software development. Just as you would think about how potential terrorists could enter and exit a facility, the software development team should think through how potentially malicious activities can happen at different input and output points of the software and the types of compromises that can take place within the guts of the software itself.

It is common for software development teams to develop threat trees, as shown in Figure 24-3. A *threat tree* is a tool that allows the development team to understand all the ways specific threats can be realized; thus, it helps them understand what type of security controls they should implement in the software to mitigate the risks associated with each threat type.

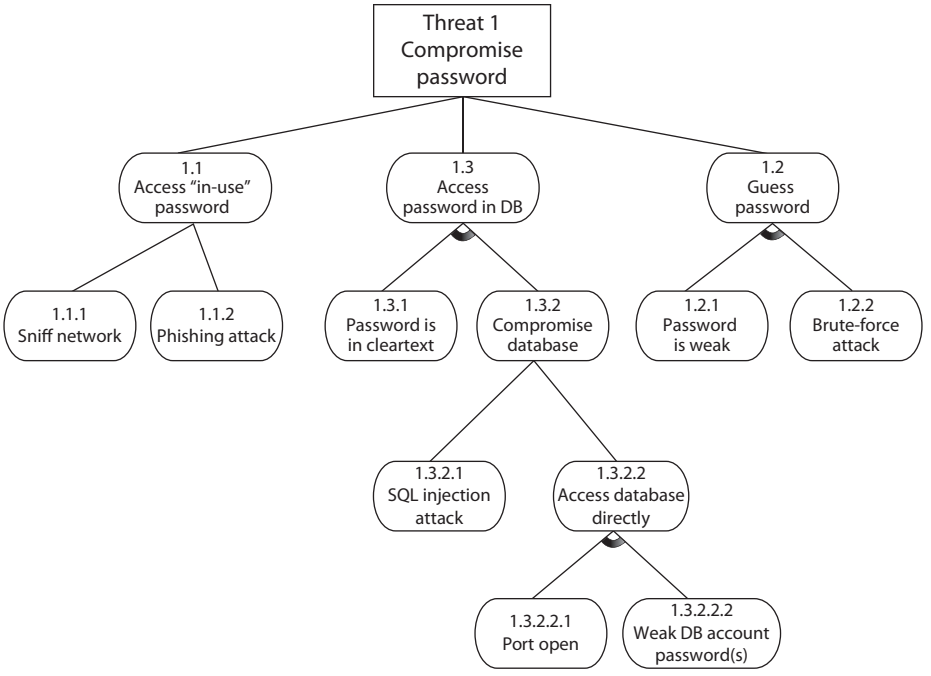


Figure 24-3 Threat tree used in threat modeling

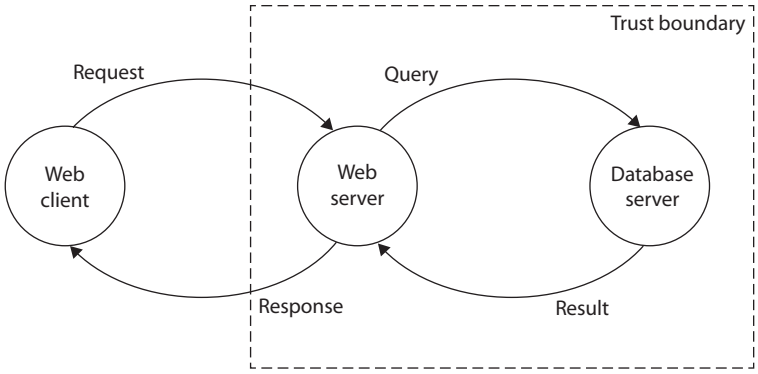


Figure 24-4 A simple flow diagram for threat modeling

There are many automated tools in the industry that software development teams can use to ensure that they address the various threat types during the design stage. One popular open-source solution is the Open Web Application Security Project (OWASP) Threat Dragon. This web-based tool enables the development team to describe threats visually using flow diagrams. Figure 24-4 shows a simple diagram of a three-tier web system showing its trust boundary and the four ways in which the tiers interact. The next step in building the threat model would be to consider how each of these four interactions could be exploited by a threat actor. For example, stolen credentials could allow an adversary to compromise the web server and, from there, issue queries to the database server that could compromise the integrity or availability of records stored there. For each threat identified through this process, the software development team would develop controls to mitigate it.

The decisions made during the design phase are pivotal steps to the development phase. Software design serves as a foundation and greatly affects software quality. If good product design is not put into place in the beginning of the project, the following phases will be much more challenging.

Development Phase

This is the phase where the programmers become deeply involved. The software design that was created in the previous phase is broken down into defined deliverables, and programmers develop code to meet the deliverable requirements.

There are many *computer-aided software engineering (CASE)* tools that programmers can use to generate code, test software, and carry out debugging activities. When these types of activities are carried out through automated tools, development usually takes place more quickly with fewer errors.

CASE refers to any type of software tool that supports automated development of software, which can come in the form of program editors, debuggers, code analyzers, version-control mechanisms, and more. These tools aid in keeping detailed records of requirements, design steps, programming activities, and testing. A CASE tool is designed to support one or more software engineering tasks in the process of developing software. Many vendors can get their products to the market faster because they are "computer aided."

In the next chapter we will delve into the abyss of "secure coding," but let's take a quick peek at it here to illustrate its importance in the development phase. As stated previously, most vulnerabilities that corporations, organizations, and individuals have to worry about reside within the programming code itself. When programmers do not follow strict and secure methods of creating programming code, the effects can be widespread and the results can be devastating. But programming securely is not an easy task. The list of errors that can lead to serious vulnerabilities in software is long.

The MITRE organization's Common Weakness Enumeration (CWE) initiative (https://cwe.mitre.org/top25) describes "a demonstrative list of the most common and impactful issues experienced over the previous two calendar years." Table 24-1 shows the most recent list.

Rank	Name
1	Out-of-bounds Write
2	Improper Neutralization of Input During Web Page Generation ("Cross-site Scripting")
3	Out-of-bounds Read
4	Improper Input Validation
5	Improper Neutralization of Special Elements used in an OS Command ("OS Command Injection")
6	Improper Neutralization of Special Elements used in an SQL Command ("SQL Injection")
7	Use After Free
8	Improper Limitation of a Pathname to a Restricted Directory ("Path Traversal")
9	Cross-Site Request Forgery (CSRF)
10	Unrestricted Upload of File with Dangerous Type
11	Missing Authentication for Critical Function
12	Integer Overflow or Wraparound
13	Deserialization of Untrusted Data
14	Improper Authentication
15	NULL Pointer Dereference
16	Use of Hard-coded Credentials
17	Improper Restriction of Operations within the Bounds of a Memory Buffer
18	Missing Authorization
19	Incorrect Default Permissions
20	Exposure of Sensitive Information to an Unauthorized Actor
21	Insufficiently Protected Credentials
22	Incorrect Permission Assignment for Critical Resource
23	Improper Restriction of XML External Entity Reference
24	Server-Side Request Forgery (SSRF)
25	Improper Neutralization of Special Elements used in a Command ("Command Injection")

Table 24-1 2021 CWE Top 25 Most Dangerous Software Weaknesses List

Many of these software issues are directly related to improper or faulty programming practices. Among other issues to address, the programmers need to check input lengths so buffer overflows cannot take place, inspect code to prevent the presence of covert channels, check for proper data types, make sure checkpoints cannot be bypassed by users, verify syntax, and verify checksums. The software development team should play out different attack scenarios to see how the code could be attacked or modified in an unauthorized fashion. Code reviews and debugging should be carried out by peer developers, and everything should be clearly documented.

A particularly important area of scrutiny is input validation because it can lead to serious vulnerabilities. Essentially, we should treat every single user input as malicious until proven otherwise. For example, if we don't put limits on how many characters users can enter when providing, say, their names on a web form, they could cause a buffer overflow, which is a classic example of a technique used to exploit improper input validation. A *buffer overflow* (which is described in detail in Chapter 18) takes place when too much data is accepted as input to a specific process. The process's memory buffer can be overflowed by shoving arbitrary data into various memory segments and inserting a carefully crafted set of malicious instructions at a specific memory address.

Buffer overflows can also lead to illicit escalation of privileges. *Privilege escalation* is the process of exploiting a process or configuration setting to gain access to resources that would normally not be available to the process or its user. For example, an attacker can compromise a regular user account and escalate its privileges to gain administrator or even system privileges on that computer. This type of attack usually exploits the complex interactions of user processes with device drivers and the underlying operating system. A combination of input validation and configuring the system to run with least privilege can help mitigate the threat of escalation of privileges.

What is important to understand is that secure coding practices need to be integrated into the development phase of the SDLC. Security has to be addressed at each phase of the SDLC, with this phase being one of the most critical.

Testing Phase

Formal and informal testing should begin as soon as possible. *Unit testing* is concerned with ensuring the quality of individual code modules or classes. Mature developers develop the unit tests for their modules before they even start coding, or at least in parallel with the coding. This approach is known as *test-driven development* and tends to result in much higher-quality code with significantly fewer vulnerabilities.

Unit tests are meant to simulate a range of inputs to which the code may be exposed. These inputs range from the mundanely expected, to the accidentally unfortunate, to the intentionally malicious. The idea is to ensure the code always behaves in an expected and secure manner. Once a module and its unit tests are finished, the unit tests are run (usually in an automated framework) on that code. The goal of this type of testing is to isolate each part of the software and show that the individual parts are correct.

Unit testing usually continues throughout the development phase. A totally different group of people should carry out the formal testing. Depending on the methodology and the organization, this could be a QA, testing, audit, or even red team. This is an example

Separation of Duties

Different environmental types (development, testing, and production) should be properly separated, and functionality and operations should not overlap. Developers should not have access to modify code used in production. The code should be tested, submitted to a library, and then sent to the production environment.

of separation of duties. A programmer should not develop, test, and release software. The more eyes that see the code, the greater the chance that flaws will be found before the product is released.

No cookie-cutter recipe exists for security testing because the applications and products can be so diverse in functionality and security objectives. It is important to map security risks to test cases and code. The software development team can take a linear approach by identifying a vulnerability, providing the necessary test scenario, performing the test, and reviewing the code for how it deals with such a vulnerability. At this phase, tests are conducted in an environment that should mirror the production environment to ensure the code does not work only in the labs.

Security attacks and penetration tests usually take place during the testing phase to identify any missed vulnerabilities. Functionality, performance, and penetration resistance are evaluated. All the necessary functionality required of the product should be in a checklist to ensure each function is accounted for.

Security tests should be run to test against the vulnerabilities identified earlier in the project. Buffer overflows should be attempted, interfaces should be hit with unexpected inputs, denial-of-service (DoS) situations should be tested, unusual user activity should take place, and if a system crashes, the product should react by reverting to a secure state. The product should be tested in various environments with different applications, configurations, and hardware platforms. A product may respond fine when installed on a clean Windows 10 installation on a stand-alone PC, but it may throw unexpected errors when installed on a laptop that is remotely connected to a network and has a virtual private network (VPN) client installed.

Verification vs. Validation

Verification determines if the software product accurately represents and meets the specifications. After all, a product can be developed that does not match the original specifications, so this step ensures the specifications are being properly met. It answers the question, "Did we build the product right?"

Validation determines if the software product provides the necessary solution for the intended real-world problem. In large projects, it is easy to lose sight of the overall goal. This exercise ensures that the main goal of the project is met. It answers the question, "Did we build the right product?"

Testing Types

Software testers on the software development team should subject the software to various types of tests to discover the variety of potential flaws. The following are some of the most common testing approaches:

- **Unit testing** Testing individual components in a controlled environment where programmers validate data structure, logic, and boundary conditions
- **Integration testing** Verifying that components work together as outlined in the design specifications
- **Acceptance testing** Ensuring that the code meets customer requirements
- **Regression testing** After a change to a system takes place, retesting to ensure functionality, performance, and protection

A well-rounded security test encompasses both manual tests and automated tests. Automated tests help locate a wide range of flaws generally associated with careless or erroneous code implementations. Some automated testing environments run specific inputs in a scripted and repeatable manner. While these tests are the bread and butter of software testing, we sometimes want to simulate random and unpredictable inputs to supplement the scripted tests.

A manual test is used to analyze aspects of the program that require human intuition and can usually be judged using computing techniques. Testers also try to locate design flaws. These include logical errors, which may enable attackers to manipulate program flow by using shrewdly crafted program sequences to access greater privileges or bypass authentication mechanisms. Manual testing involves code auditing by security-centric programmers who try to modify the logical program structure using rogue inputs and reverse-engineering techniques. Manual tests simulate the live scenarios involved in real-world attacks. Some manual testing also involves the use of social engineering to analyze the human weakness that may lead to system compromise.

At this stage, issues found in testing procedures are relayed to the development team in problem reports. The problems are fixed and programs retested. This is a continual process until everyone is satisfied that the product is ready for production. If there is a specific customer, the customer would run through a range of tests before formally accepting the product; if it is a generic product, beta testing can be carried out by various potential customers and agencies. Then the product is formally released to the market or customer.

NOTE Sometimes developers include lines of code in a product that will allow them to do a few keystrokes and get right into the application. This allows them to bypass any security and access controls so they can quickly access the application's core components. This is referred to as a "back door" or "maintenance hook" and must be removed before the code goes into production.

Operations and Maintenance Phase

Once the software code is developed and properly tested, it is released so that it can be implemented within the intended production environment. The software development team's role is not finished at this point. Newly discovered problems and vulnerabilities are commonly

identified at this phase. For example, if a company developed a customized application for a specific customer, the customer could run into unforeseen issues when rolling out the product within its various networked environments. Interoperability issues might come to the surface, or some configurations may break critical functionality. The developers would need to make the necessary changes to the code, retest the code, and re-release the code.

Almost every software system requires the addition of new features over time. Frequently, these have to do with changing business processes or interoperability with other systems. This highlights the need for the operations and development teams to work particularly closely during the operations and maintenance (O&M) phase. The operations team, which is typically the IT department, is responsible for ensuring the reliable operation of all production systems. The development team is responsible for any changes to the software in development systems up until the time the software goes into production. Together, the operations and development teams address the transition from development to production as well as management of the system's configuration.

Another facet of O&M is driven by the fact that new vulnerabilities are regularly discovered. While the developers may have carried out extensive security testing, it is close to impossible to identify all the security issues at one point and time. Zero-day vulnerabilities may be identified, coding errors may be uncovered, or the integration of the software with another piece of software may uncover security issues that have to be addressed. The development team must develop patches, hotfixes, and new releases to address these items. In all likelihood, this is where you as a CISSP will interact the most with the SDLC.

Change Management

One of the key processes on which to focus for improvement involves how we deal with the inevitable changes. These can cause a lot of havoc if not managed properly and in a deliberate manner. We already discussed change management in general in Chapter 20, but it is particularly important during the lifetime of a software development project.

The need to change software arises for several reasons. During the development phase, a customer may alter requirements and ask that certain functionalities be added, removed, or modified. In production, changes may need to happen because of other changes in the environment, new requirements of a software product or system, or newly released patches or upgrades. These changes should be carefully analyzed, approved, and properly incorporated such that they do not affect any original functionality in an adverse way.

Change management is a systematic approach to deliberately regulating the changing nature of projects, including software development projects. It is a management process that takes into account not just the technical issues but also resources (like people and money), project life cycle, and even organizational climate. Many times, the hardest part of managing change is not the change itself, but the effects it has in the organization. Many of us have been on the receiving end of a late-afternoon phone call in which we're told to change our plans because of a change in a project on which we weren't even working. An important part of change management is controlling change.

Change Control

Change control is the process of controlling the specific changes that take place during the life cycle of a system and documenting the necessary change control activities. Whereas change management is the project manager's responsibility as an overarching

process, change control is what developers do to ensure the software doesn't break when they change it.

Change control involves a bunch of things to consider. The change must be approved, documented, and tested. Some tests may need to be rerun to ensure the change does not affect the product's capabilities. When a programmer makes a change to source code, she should do so on the test version of the code. Under no conditions should a programmer change the code that is already in production. After making changes to the code, the programmer should test the code and then deliver the new code to the librarian. Production code should come only from the librarian and not from a programmer or directly from a test environment.

A process for controlling changes needs to be in place at the beginning of a project so that everyone knows how to deal with changes and knows what is expected of each entity when a change request is made. Some projects have been doomed from the start because proper change control was not put into place and enforced. Many times in development, the customer and vendor agree on the design of the product, the requirements, and the specifications. The customer is then required to sign a contract confirming this is the agreement and that if they want any further modifications, they will have to pay the vendor for that extra work. If this agreement is not put into place, then the customer can continually request changes, which requires the software development team to put in the extra hours to provide these changes, the result of which is that the vendor loses money, the product does not meet its completion deadline, and scope creep occurs.

Other reasons exist to have change control in place. These reasons deal with organizational policies, standard procedures, and expected results. If a software product is in the last phase of development and a change request comes in, the development team should know how to deal with it. Usually, the team leader must tell the project manager how much extra time will be required to complete the project if this change is incorporated and what steps need to be taken to ensure this change does not affect other components within the product. If these processes are not controlled, one part of a development team could implement the change without another part of the team being aware of it. This could break some of the other development team's software pieces. When the pieces of the product are integrated and some pieces turn out to be incompatible, some jobs may be in jeopardy, because management never approved the change in the first place.

Change control processes should be evaluated during system audits. It is possible to overlook a problem that a change has caused in testing, so the procedures for how change control is implemented and enforced should be examined during a system audit.

The following are some necessary steps for a change control process:

1. Make a formal request for a change.
2. Analyze the request:
   a. Develop the implementation strategy.
   b. Calculate the costs of this implementation.
   c. Review security implications.
3. Record the change request.
4. Submit the change request for approval.

5. Develop the change:
   a. Recode segments of the product and add or subtract functionality.
   b. Link these changes in the code to the formal change control request.
   c. Submit software for testing and quality control.
   d. Repeat until quality is adequate.
   e. Make version changes.
6. Report results to management.

The changes to systems may require another round of certification and accreditation. If the changes to a system are significant, then the functionality and level of protection

SDLC and Security

The main phases of a software development life cycle are shown here with some specific security tasks.

Requirements gathering:

- Security risk assessment
- Privacy risk assessment
- Risk-level acceptance
- Informational, functional, and behavioral requirements

Design:

- Attack surface analysis
- Threat modeling

Development:

- Automated CASE tools
- Secure coding

Testing:

- Automated testing
- Manual testing
- Unit, integration, acceptance, and regression testing

Operations and maintenance:

- Vulnerability patching
- Change management and control

may need to be reevaluated (certified), and management would have to approve the overall system, including the new changes (accreditation).

Development Methodologies

Several software development methodologies are in common use around the world. While some include security issues in certain phases, these are not considered "security-centric development methodologies." They are simply classical approaches to building and developing software. Let's dive into some of the methodologies that you should know as a CISSP.

EXAM TIP It is exceptionally rare to see a development methodology used in its pure form in the real world. Instead, organizations typically start with a base methodology and modify it to suit their own unique environment. For purposes of the CISSP exam, however, you should focus on what differentiates each development approach.

Waterfall Methodology

The *Waterfall methodology* uses a linear-sequential life-cycle approach, illustrated in Figure 24-5. Each phase must be completed in its entirety before the next phase can begin. At the end of each phase, a review takes place to make sure the project is on the correct path and should continue.

In this methodology all requirements are gathered in the initial phase and there is no formal way to integrate changes as more information becomes available or requirements change. It is hard to know everything at the beginning of a project, so waiting until the whole project is complete to integrate necessary changes can be ineffective and time consuming. As an analogy, let's say that you are planning to landscape your backyard that is one acre in size. In this scenario, you can go to the gardening store only one time to get

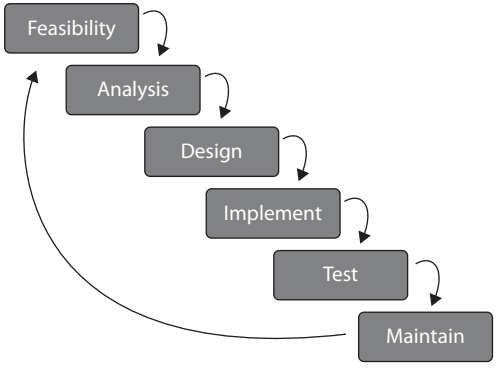


Figure 24-5 Waterfall methodology used for software development

all your supplies. If you identify during the project that you need more topsoil, rocks, or pipe for the sprinkler system, you have to wait and complete the whole yard before you can return to the store for extra or more suitable supplies.

The Waterfall methodology is a very rigid approach that could be useful for smaller projects in which all the requirements are fully understood up front. It may also be a good choice in some large projects for which different organizations will perform the work at each phase. Overall, however, it is not an ideal methodology for most complex projects, which commonly contain many variables that affect the scope as the project continues.

Prototyping

A *prototype* is a sample of software code or a model that can be developed to explore a specific approach to a problem before investing expensive time and resources. A team can identify the usability and design problems while working with a prototype and adjust their approach as necessary. Within the software development industry, three main prototype models have been invented and used. These are the rapid prototype, evolutionary prototype, and operational prototype.

Rapid prototyping is an approach that allows the development team to quickly create a prototype (sample) to test the validity of the current understanding of the project requirements. In a software development project, the team could develop a *rapid prototype* to see if their ideas are feasible and if they should move forward with their current solution. The rapid prototype approach (also called throwaway) is a "quick and dirty" method of creating a piece of code and seeing if everyone is on the right path or if another solution should be developed. The rapid prototype is not developed to be built upon, but to be discarded after serving its purposes.

When *evolutionary prototypes* are developed, they are built with the goal of incremental improvement. Instead of being discarded after being developed, as in the rapid prototype approach, the evolutionary prototype is continually improved upon until it reaches the final product stage. Feedback that is gained through each development phase is used to improve the prototype and get closer to accomplishing the customer's needs.

Operational prototypes are an extension of the evolutionary prototype method. Both models (operational and evolutionary) improve the quality of the prototype as more data is gathered, but the operational prototype is designed to be implemented within a production environment as it is being tweaked. The operational prototype is updated as customer feedback is gathered, and the changes to the software happen within the working site.

In summary, a rapid prototype is developed to give a quick understanding of the suggested solution, an evolutionary prototype is created and improved upon within a lab environment, and an operational prototype is developed and improved upon within a production environment.

Incremental Methodology

If a development team follows the *Incremental methodology*, this allows them to carry out multiple development cycles on a piece of software throughout its development stages. This would be similar to "multi-Waterfall" cycles taking place on one piece of software as

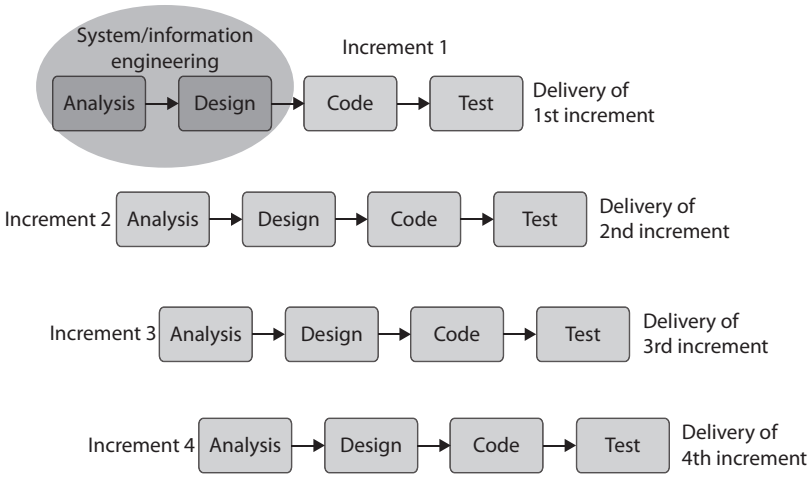


Figure 24-6 Incremental development methodology

it matures through the development stages. A version of the software is created in the first iteration and then it passes through each phase (requirements analysis, design, coding, testing, implementation) of the next iteration process. The software continues through the iteration of phases until a satisfactory product is produced. This methodology is illustrated in Figure 24-6.

When using the Incremental methodology, each incremental phase results in a deliverable that is an operational product. This means that a working version of the software is produced after the first iteration and that version is improved upon in each of the subsequent iterations. Some benefits to this methodology are that a working piece of software is available in early stages of development, the flexibility of the methodology allows for changes to take place, testing uncovers issues more quickly than the Waterfall methodology since testing takes place after each iteration, and each iteration is an easily manageable milestone.

Because each incremental phase delivers an operational product, the customer can respond to each build and help the development team in its improvement processes, and because the initial product is delivered more quickly compared to other methodologies, the initial product delivery costs are lower, the customer gets its functionality earlier, and the risks of critical changes being introduced are lower.

This methodology is best used when issues pertaining to risk, program complexity, funding, and functionality requirements need to be understood early in the product development life cycle. If a vendor needs to get the customer some basic functionality quickly as it works on the development of the product, this can be a good methodology to follow.

Spiral Methodology

The *Spiral methodology* uses an iterative approach to software development and places emphasis on risk analysis. The methodology is made up of four main phases: determine objectives, identify and resolve risks, development and test, and plan the next iteration. The development team starts with the initial requirements and goes through each of these phases, as shown in Figure 24-7. Think about starting a software development project at the center of this graphic. You have your initial understanding and requirements of the project, develop specifications that map to these requirements, identify and resolve risks, build prototype specifications, test your specifications, build a development plan, integrate newly discovered information, use the new information to carry out a new risk analysis, create a prototype, test the prototype, integrate resulting data into the process, and so forth. As you gather more information about the project, you integrate it into the risk analysis process, improve your prototype, test the prototype, and add more granularity to each step until you have a completed product.

The iterative approach provided by the Spiral methodology allows new requirements to be addressed as they are uncovered. Each prototype allows for testing to take place

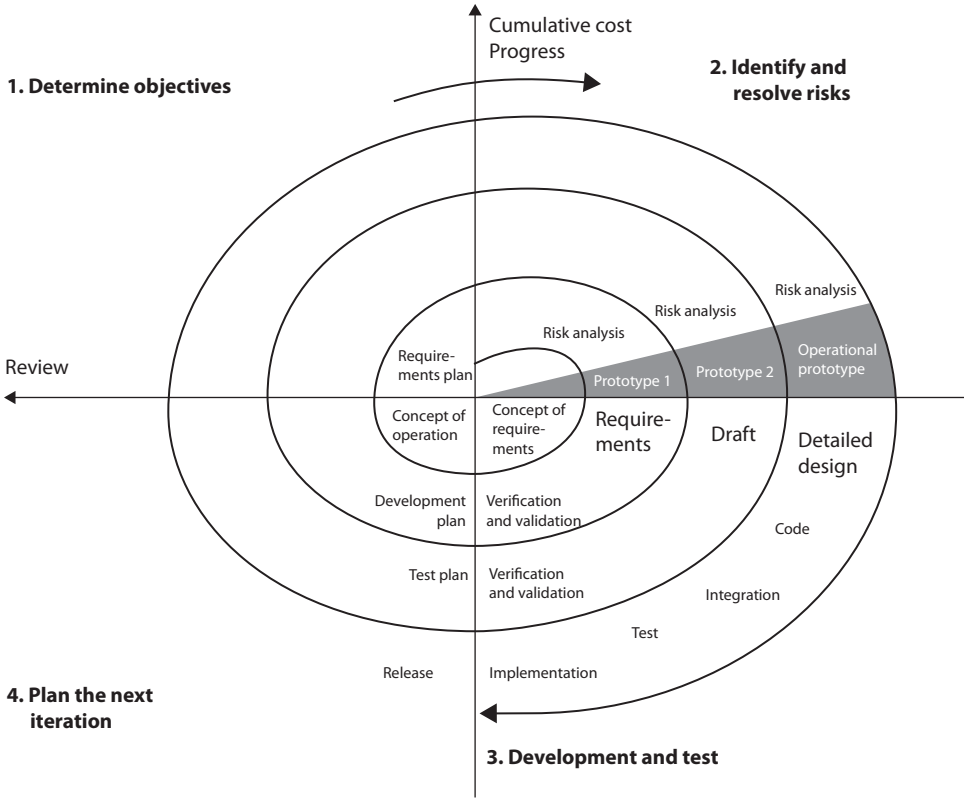


Figure 24-7 Spiral methodology for software development

early in the development project, and feedback based upon these tests is integrated into the following iteration of steps. The risk analysis ensures that all issues are actively reviewed and analyzed so that things do not "slip through the cracks" and the project stays on track.

In the Spiral methodology the last phase allows the customer to evaluate the product in its current state and provide feedback, which is an input value for the next spiral of activity. This is a good methodology for complex projects that have fluid requirements.

NOTE Within this methodology the angular aspect represents progress and the radius of the spirals represents cost.

Rapid Application Development

The *Rapid Application Development (RAD)* methodology relies more on the use of rapid prototyping than on extensive upfront planning. In this methodology, the planning of how to improve the software is interleaved with the processes of developing the software, which allows for software to be developed quickly. The delivery of a workable piece of software can take place in less than half the time compared to the Waterfall methodology. The RAD methodology combines the use of prototyping and iterative development procedures with the goal of accelerating the software development process. The development process begins with creating data models and business process models to help define what the end-result software needs to accomplish. Through the use of prototyping, these data and process models are refined. These models provide input to allow for the improvement of the prototype, and the testing and evaluation of the prototype allow for the improvement of the data and process models. The goal of these steps is to combine business requirements and technical design statements, which provide the direction in the software development project.

Figure 24-8 illustrates the basic differences between traditional software development approaches and RAD. As an analogy, let's say that the development team needs you to tell them what it is you want so that they can build it for you. You tell them that the thing you want has four wheels and an engine. They bring you a two-seat convertible and ask, "Is this what you want?" You say, "No, it must be able to seat four adults." So they leave the prototype with you and go back to work. They build a four-seat convertible and deliver it to you, and you tell them they are getting closer but it still doesn't fit your requirements. They get more information from you, deliver another prototype, get more feedback, and on and on. That back and forth is what is taking place in the circle portion of Figure 24-8.

The main reason that RAD was developed was that by the time software was completely developed following other methodologies, the requirements changed and the developers had to "go back to the drawing board." If a customer needs you to develop a software product and it takes you a year to do so, by the end of that year the customer's needs for the software have probably advanced and changed. The RAD methodology allows for the customer to be involved during the development phases so that the end result maps to their needs in a more realistic manner.

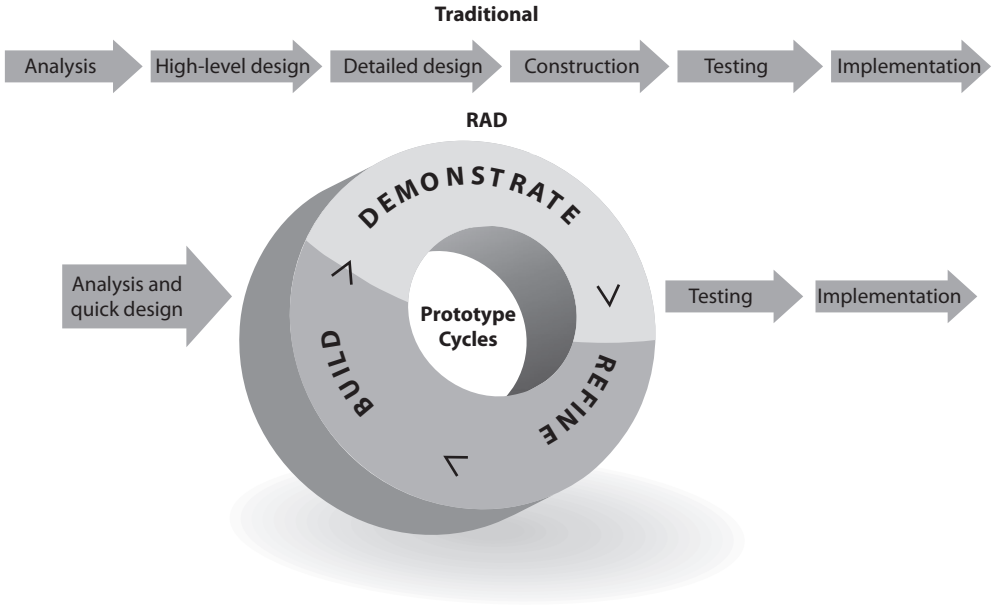


Figure 24-8 Rapid Application Development methodology

Agile Methodologies

The industry seems to be full of software development methodologies, each trying to improve upon the deficiencies of the ones before it. Before the Agile approach to development was created, teams were following rigid process-oriented methodologies. These approaches focused more on following procedures and steps instead of potentially carrying out tasks in a more efficient manner. As an analogy, if you have ever worked within or interacted with a large government agency, you may have come across silly processes that took too long and involved too many steps. If you are a government employee and need to purchase a new chair, you might have to fill out four sets of documents that need to be approved by three other departments. You probably have to identify three different chair vendors, who have to submit a quote, which goes through the contracting office. It might take you a few months to get your new chair. The focus is to follow a protocol and rules instead of efficiency.

Many of the classical software development approaches, as in Waterfall, provide rigid processes to follow that do not allow for much flexibility and adaptability. Commonly, the software development projects that follow these approaches end up failing by not meeting schedule time release, running over budget, and/or not meeting the needs of the customer. Sometimes you need the freedom to modify steps to best meet the situation's needs.

Agile methodology is an umbrella term for several development methodologies. The overarching methodology focuses not on rigid, linear, stepwise processes, but instead on incremental and iterative development methods that promote cross-functional teamwork

and continuous feedback mechanisms. This methodology is considered "lightweight" compared to the traditional methodologies that are "heavyweight," which just means this methodology is not confined to a tunnel-visioned and overly structured approach. It is nimble and flexible enough to adapt to each project's needs. The industry found out that even an exhaustive library of defined processes cannot handle every situation that could arise during a development project. So instead of investing time and resources into deep upfront design analysis, the Agile methodology focuses on small increments of functional code that are created based on business need.

The various methodologies under the Agile umbrella focus on individual interaction instead of processes and tools. They emphasize developing the right software product over comprehensive and laborious documentation. They promote customer collaboration instead of contract negotiation, and emphasize abilities to respond to change instead of strictly following a plan.

A notable element of many Agile methodologies is their focus on user stories. A *user story* is a sentence that describes what a user wants to do and why. For instance, a user story could be "As a customer, I want to search for products so that I can buy some." Notice the structure of the story is "As a <user role>, I want to <accomplish some goal> so that <reason for accomplishing the goal>." For example, "As a network analyst, I want to record pcap (packet capture) files so that I can analyze downloaded malware." This method of documenting user requirements is very familiar to the customers and enables their close collaboration with the development team. Furthermore, by keeping this user focus, validation of the features is simpler because the "right system" is described up front by the users in their own words.

EXAM TIP The Agile methodologies do not use prototypes to represent the full product, but break the product down into individual features that are continuously being delivered.

Another important characteristic of the Agile methodologies is that the development team can take pieces and parts of all of the available SDLC methodologies and combine them in a manner that best meets the specific project needs. These various combinations have resulted in many methodologies that fall under the Agile umbrella.

Scrum

Scrum is one of the most widely adopted Agile methodologies in use today. It lends itself to projects of any size and complexity and is very lean and customer focused. Scrum is a methodology that acknowledges the fact that customer needs cannot be completely understood and will change over time. It focuses on team collaboration, customer involvement, and continuous delivery.

The term *scrum* originates from the sport of rugby. Whenever something interrupts play (e.g., a penalty or the ball goes out of bounds) and the game needs to be restarted, all players come together in a tight formation. The ball is then thrown into the middle and the players struggle with each other until one team or the other gains possession of the ball, allowing the game to continue. Extending this analogy, the Scrum methodology

allows the project to be reset by allowing product features to be added, changed, or removed at clearly defined points. Since the customer is intimately involved in the development process, there should be no surprises, cost overruns, or schedule delays. This allows a product to be iteratively developed and changed even as it is being built.

The change points happen at the conclusion of each *sprint*, a fixed-duration development interval that is usually (but not always) two weeks in length and promises delivery of a very specific set of features. These features are chosen by the team, but with a lot of input from the customer. There is a process for adding features at any time by inserting them in the feature backlog. However, these features can be considered for actual work only at the beginning of a new sprint. This shields the development team from changes during a sprint, but allows for changes in between sprints.

Extreme Programming

If you take away the regularity of Scrum's sprints and backlogs and add a lot of code reviewing, you get our next Agile methodology. Extreme Programming (XP) is a development methodology that takes code reviews (discussed in Chapter 18) to the extreme (hence the name) by having them take place continuously. These continuous reviews are accomplished using an approach called *pair programming*, in which one programmer dictates the code to her partner, who then types it. While this may seem inefficient, it allows two pairs of eyes to constantly examine the code as it is being typed. It turns out that this approach significantly reduces the incidence of errors and improves the overall quality of the code.

Another characteristic of XP is its reliance on test-driven development, in which the unit tests are written before the code. The programmer first writes a new unit test case, which of course fails because there is no code to satisfy it. The next step is to add just enough code to get the test to pass. Once this is done, the next test is written, which fails, and so on. The consequence is that only the minimal amount of code needed to pass the tests is developed. This extremely minimal approach reduces the incidence of errors because it weeds out complexity.

Kanban

Kanban is a production scheduling system developed by Toyota to more efficiently support just-in-time delivery. Over time, Kanban was adopted by IT and software systems developers. In this context, the *Kanban* development methodology is one that stresses visual tracking of all tasks so that the team knows what to prioritize at what point in time in order to deliver the right features right on time. Kanban projects used to be very noticeable because entire walls in conference rooms would be covered in sticky notes representing the various tasks that the team was tracking. Nowadays, many Kanban teams opt for virtual walls on online systems.

The Kanban wall is usually divided vertically by production phase. Typical columns are labeled Planned, In Progress, and Done. Each sticky note can represent a user story as it moves through the development process, but more importantly, the sticky note can also be some other work that needs to be accomplished. For instance, suppose that one of the user stories is the search feature described earlier in this section. While it is being developed, the team realizes that the searches are very slow. This could result in a task being

added to change the underlying data or network architecture or to upgrade hardware. This sticky note then gets added to the Planned column and starts being prioritized and tracked together with the rest of the remaining tasks. This process highlights how Kanban allows the project team to react to changing or unknown requirements, which is a common feature among all Agile methodologies.

DevOps

Traditionally, the software development team and the IT team are two separate (and sometimes antagonistic) groups within an organization. Many problems stem from poor collaboration between these two teams during the development process. It is not rare to have the IT team berating the developers because a feature push causes the IT team to have to stay late or work on a weekend or simply drop everything they were doing in order to "fix" something that the developers "broke." This friction makes a lot of sense when you consider that each team is incentivized by different outcomes. Developers want to push out finished code, usually under strict schedules. The IT staff, on the other hand, wants to keep the IT infrastructure operating effectively. Many project managers who have managed software development efforts will attest to having received complaints from developers that the IT team was being unreasonable and uncooperative, while the IT team was simultaneously complaining about buggy code being tossed over the fence at them at the worst possible times and causing problems on the rest of the network.

A good way to solve this friction is to have both developers and members of the operations staff (hence the term DevOps) on the software development team. *DevOps* is the practice of incorporating development, IT, and quality assurance (QA) staff into software development projects to align their incentives and enable frequent, efficient, and reliable releases of software products. This relationship is illustrated in Figure 24-9.

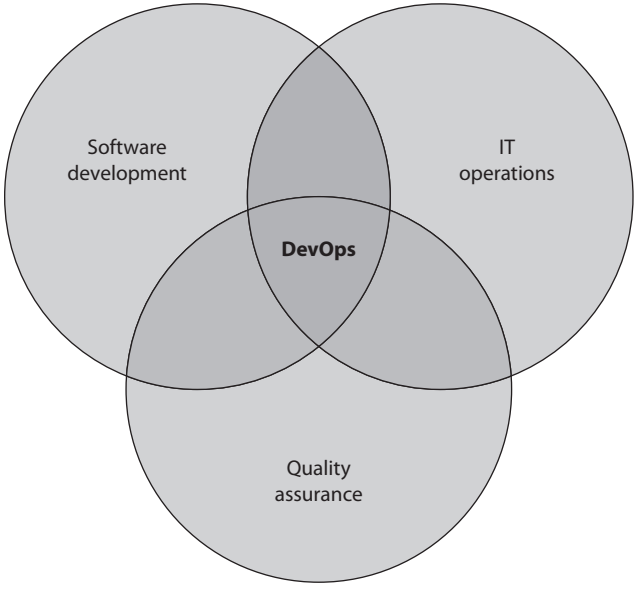


Figure 24-9 DevOps exists at the intersection of software development, IT, and QA.

Ultimately, DevOps is about changing the culture of an organization. It has a huge positive impact on security, because in addition to QA, the IT teammates will be involved at every step of the process. Multifunctional integration allows the team to identify potential defects, vulnerabilities, and friction points early enough to resolve them proactively. This is one of the biggest selling points for DevOps. According to multiple surveys, there are a few other, perhaps more powerful benefits: DevOps increases trust within an organization and increases job satisfaction among developers, IT staff, and QA personnel. Unsurprisingly, it also improves the morale of project managers.

DevSecOps

It is not all that common for the security team to be involved in software development efforts, but it makes a lot of sense for them to be. Their job is to find vulnerabilities before the threat actors can and then do something about it. As a result, most security professionals develop an "adversarial mindset" that allows them to think like attackers in order to better defend against them. Imagine being a software developer and having someone next to you telling you all the ways they could subvert your code to do bad things. It'd be kind of like having a spell-checker but for vulnerabilities instead of spelling!

DevSecOps is the integration of development, security, and operations professionals into a software development team. It's just like DevOps but with security added in. One of the main advantages of DevSecOps is that it bakes security right into the development process, rather than bolting it on at the end of it. Rather than implementing controls to mitigate vulnerabilities, the vulnerabilities are prevented from being implemented in the first place.

Other Methodologies

There seems to be no shortage of SDLC and software development methodologies in the industry. The following is a quick summary of a few others that can also be used:

- **Exploratory methodology** A methodology that is used in instances where clearly defined project objectives have not been presented. Instead of focusing on explicit tasks, the exploratory methodology relies on covering a set of specifications likely to affect the final product's functionality. Testing is an important part of exploratory development, as it ascertains that the current phase of the project is compliant with likely implementation scenarios.
- **Joint Application Development (JAD)** A methodology that uses a team approach in application development in a workshop-oriented environment. This methodology is distinguished by its inclusion of members other than coders in the team. It is common to find executive sponsors, subject matter experts, and end users spending hours or days in collaborative development workshops.

Integrated Product Team

An *integrated product team (IPT)* is a multidisciplinary development team with representatives from many or all the stakeholder populations. The idea makes a lot of sense when you think about it. Why should programmers learn or guess the manner in which the accounting folks handle accounts payable? Why should testers and quality control personnel wait until a product is finished before examining it? Why should the marketing team wait until the project (or at least the prototype) is finished before determining how best to sell it? A comprehensive IPT includes business executives and end users and everyone in between.

The Joint Application Development methodology, in which users join developers during extensive workshops, works well with the IPT approach. IPTs extend this concept by ensuring that the right stakeholders are represented in every phase of the development as formal team members. In addition, whereas JAD is focused on involving the user community, an IPT is typically more inward facing and focuses on bringing in the business stakeholders.

An IPT is not a development methodology. Instead, it is a management technique. When project managers decide to use IPTs, they still have to select a methodology. These days, IPTs are often associated with Agile methodologies.

- **Reuse methodology** A methodology that approaches software development by using progressively developed code. Reusable programs are evolved by gradually modifying preexisting prototypes to customer specifications. Since the reuse methodology does not require programs to be built from scratch, it drastically reduces both development cost and time.
- **Cleanroom** An approach that attempts to prevent errors or mistakes by following structured and formal methods of developing and testing. This approach is used for high-quality and mission-critical applications that will be put through a strict certification process.

We covered only the most commonly used methodologies in this section, but there are many more that exist. New methodologies have evolved as technology and research have advanced and various weaknesses of older approaches have been addressed. Most of the methodologies exist to meet a specific software development need, and choosing the wrong approach for a certain project could be devastating to its overall success.

EXAM TIP While all the methodologies we covered are used in many organizations around the world, you should focus on Agile, Waterfall, DevOps, and DevSecOps for the CISSP exam.

Review of Development Methodologies

A quick review of the various methodologies we have covered up to this point is provided here:

- **Waterfall** Very rigid, sequential approach that requires each phase to complete before the next one can begin. Difficult to integrate changes. Inflexible methodology.
- **Prototyping** Creating a sample or model of the code for proof-of-concept purposes.
- **Incremental** Multiple development cycles are carried out on a piece of software throughout its development stages. Each phase provides a usable version of software.
- **Spiral** Iterative approach that emphasizes risk analysis per iteration. Allows for customer feedback to be integrated through a flexible evolutionary approach.
- **Rapid Application Development** Combines prototyping and iterative development procedures with the goal of accelerating the software development process.
- **Agile** Iterative and incremental development processes that encourage team-based collaboration. Flexibility and adaptability are used instead of a strict process structure.
- **DevOps** The software development and IT operations teams work together at all stages of the project to ensure a smooth transition from development to production environments.
- **DevSecOps** Just like DevOps, but also integrates the security team into every stage of the project.

Maturity Models

Regardless of which software development methodology an organization adopts, it is helpful to have a framework for determining how well-defined and effective its development activities are. Maturity models identify the important components of software development processes and then organize them in an evolutionary scale that proceeds from ad hoc to mature. Each maturity level comprises a set of goals that, when they are met, stabilize one or more of those components. As an organization moves up this maturity scale, the effectiveness, repeatability, and predictability of its software development processes increase, leading to higher-quality code. Higher-quality code, in turn, means fewer vulnerabilities, which is why we care so deeply about this topic as cybersecurity leaders. Let's take a look at the two most popular models: the Capability Maturity Model Integration (CMMI) and the Software Assurance Maturity Model (SAMM).

Capability Maturity Model Integration

Capability Maturity Model Integration (CMMI) is a comprehensive set of models for developing software. It addresses the different phases of a software development life cycle, including concept definition, requirements analysis, design, development, integration, installation, operations, and maintenance, and what should happen in each phase. It can be used to evaluate security engineering practices and identify ways to improve them. It can also be used by customers in the evaluation process of a software vendor. Ideally, software vendors would use the model to help improve their processes, and customers would use the model to assess the vendors' practices.

EXAM TIP For exam purposes, the terms CMM and CMMI are equivalent.

CMMI describes procedures, principles, and practices that underlie software development process maturity. This model was developed to help software vendors improve their development processes by providing an evolutionary path from an ad hoc "fly by the seat of your pants" approach to a more disciplined and repeatable method that improves software quality, reduces the life cycle of development, provides better project management capabilities, allows for milestones to be created and met in a timely manner, and takes a more proactive approach than the less effective reactive approach. It provides best practices to allow an organization to develop a standardized approach to software development that can be used across many different groups. The goal is to continue to review and improve upon the processes to optimize output, increase capabilities, and provide higher-quality software at a lower cost through the implementation of continuous improvement steps.

If the company Stuff-R-Us wants a software development company, Software-R-Us, to develop an application for it, it can choose to buy into the sales hype about how wonderful Software-R-Us is, or it can ask Software-R-Us whether it has been evaluated against CMMI. Third-party companies evaluate software development companies to certify their product development processes. Many software companies have this evaluation done so they can use this as a selling point to attract new customers and provide confidence for their current customers.

The five maturity levels of CMMI are shown in Figure 24-10 and described here:

- **Level 0: Incomplete** Development process is ad hoc or even chaotic. Tasks are not always completed at all, so projects are regularly cancelled or abandoned.
- **Level 1: Initial** The organization does not use effective management procedures and plans. There is no assurance of consistency, and quality is unpredictable. Success is usually the result of individual heroics.
- **Level 2: Managed** A formal management structure, change control, and quality assurance are in place for individual projects. The organization can properly repeat processes throughout each project.

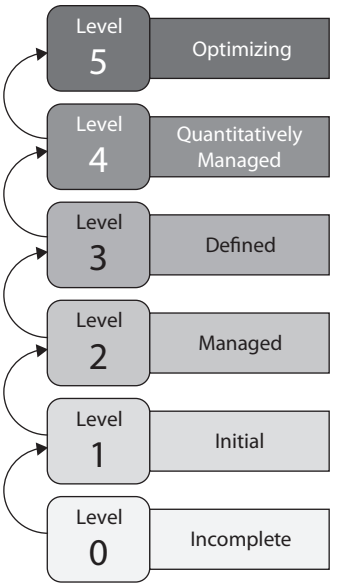


Figure 24-10 CMMI staged maturity levels

- **Level 3: Defined** Formal procedures are in place that outline and define processes carried out in all projects across the organization. This allows the organization to be proactive rather than reactive.
- **Level 4: Quantitatively Managed** The organization has formal processes in place to collect and analyze quantitative data, and metrics are defined and fed into the process-improvement program.
- **Level 5: Optimizing** The organization has budgeted and integrated plans for continuous process improvement, which allow it to quickly respond to opportunities and changes.

Each level builds upon the previous one. For example, a company that accomplishes a Level 5 CMMI rating must meet all the requirements outlined in Levels 1–4 along with the requirements of Level 5.

If a software development vendor is using the Prototyping methodology that was discussed earlier in this chapter, the vendor would most likely only achieve a CMMI Level 1, particularly if its practices are ad hoc, not consistent, and the level of the quality that its software products contain is questionable. If this company practiced a strict Agile SDLC methodology consistently and carried out development, testing, and documentation precisely, it would have a higher chance of obtaining a higher CMMI level.

Capability maturity models (CMMs) are used for many different purposes, software development processes being one of them. They are general models that allow for

maturity-level identification and maturity improvement steps. We showed how a CMM can be used for organizational security program improvement processes in Chapter 4.

The software industry ended up with several different CMMs, which led to confusion. CMMI was developed to bring many of these different maturity models together and allow them to be used in one framework. CMMI was developed by industry experts, government entities, and the Software Engineering Institute at Carnegie Mellon University. So CMMI has replaced CMM in the software engineering world, but you may still see CMM referred to within the industry and on the CISSP exam. Their ultimate goals are the same, which is process improvement.

NOTE CMMI is continually being updated and improved upon. You can view the latest documents on it at https://cmmiinstitute.com/learning/appraisals/levels.

Software Assurance Maturity Model

The OWASP *Software Assurance Maturity Model (SAMM)* is specifically focused on secure software development and allows organizations of any size to decide their target maturity levels within each of the five critical business functions: Governance, Design, Implementation, Verification, and Operations, as shown in Figure 24-11. One of the premises on which SAMM is built is that any organization that is involved in software development must perform these five functions.

Each business function, in turn, is divided into three security practices, which are sets of security-related activities that provide assurance for the function. For example, if you want to ensure that your Design business function is done right, you need to perform activities related to threat assessment, identification of security requirements, and securely architecting the software. Each of these 15 practices can be independently

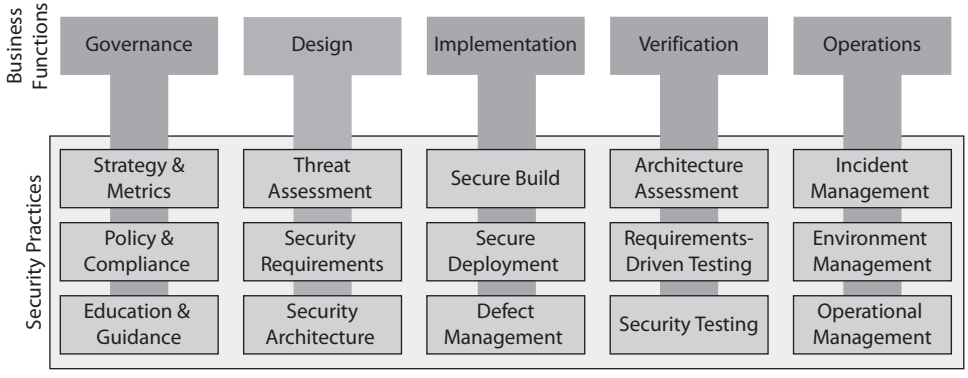


Figure 24-11 Software Assurance Maturity Model