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.

3. Record the change request.
4. Submit the change request for approval.

PART VIII

c. Review security implications.

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 "securitycentric 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

Feasibility
Analysis
Design
Implement
Test
Maintain

PART VIII

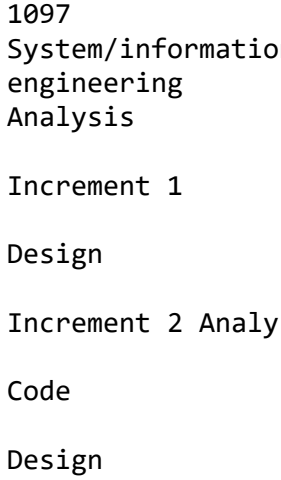
↟CISSP All-in-One Exam Guide

1096
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



Increment 4 Analysis

Figure 24-6

Test

Code

Design

Test

Code

Design

Delivery of
1st increment

Code

Delivery of
2nd increment

Delivery of
3rd increment

Test

Test

Delivery of
4th increment

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.
PART VIII

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
Cumulative cost
Progress

2. Identify and resolve risks

1. Determine objectives

Risk analysis
Risk analysis
Risk analysis
Requirements plan

Review

Prototype 1

Concept of operation

Concept of requirements

Development plan

Verification and validation

Test plan

Verification and validation

Requirements

Draft

Integration

Implementation

4. Plan the next iteration
3. Development and test

Figure 24-7

Spiral methodology for software development

Detailed design

Code

Test
Release

Operational
prototype

Prototype 2

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

PART VIII

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.

Traditional
Analysis

High-level design

Detailed design

Construction

Testing

Implementation

RAD

Testing

Prototype
Cycles

Implementation

BU

FI

E
Figure 24-8

E

RE

IL D

Analysis and
quick design

O N S T R AT

N

D

EM

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

PART VIII

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.

Software development

IT operations

DevOps

PART VIII

Quality assurance

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:

PART VIII

- 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

5

Optimizing

Level

Quantitatively
Managed

4
Level

3

Defined

Level

2

Managed

Level

1

Initial

Level

0

Incomplete

• 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

Security Practices

Business Functions

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

Governance

Design

Implementation

Verification

Operations

Strategy &
Metrics

Threat
Assessment

Secure Build

Architecture
Assessment

Incident
Management

Policy &
Compliance

Security
Requirements

Secure
Deployment

RequirementsDriven Testing

Environment
Management

Education &
Guidance

Security
Architecture

Defect
Management

Figure 24-11 Software Assurance Maturity Model

assessed and matured, which allows the organization to decide what maturity level makes
sense for each practice.
NOTE You can find more information on SAMM at https://owaspsamm.org /model/.

Chapter Review

While there is no expectation that you, as a CISSP, will necessarily be involved in software development, you will almost certainly lead organizations that either produce software or consume it. Therefore, it is important that you understand how secure software
is developed. Knowing this enables you to see what an organization is doing, software
development-wise, and quickly get a sense of the maturity of its processes. If processes
are ad hoc, this chapter should have given you some pointers on how to formalize the
processes. After all, without formal processes and trained programmers in place, you have
almost no hope of producing software that is not immediately vulnerable as soon as it is
put into production. On the other hand, if the organization seems more mature, you can
delve deeper into the specifics of building security into the software, which is the topic
of the next chapter.

Quick Review

- The software development life cycle (SDLC) comprises five phases: requirements gathering, design, development, testing, and operations and maintenance (O&M).
- Computer-aided software engineering (CASE) refers to any type of software that allows for the automated development of software, which can come in the form of program editors, debuggers, code analyzers, version-control mechanisms, and more.
The goals are to increase development speed and productivity and reduce errors.
- Various levels of testing should be carried out during development: unit (testing
individual components), integration (verifying components work together in the production environment), acceptance (ensuring code meets customer requirements), and regression (testing after changes take place).
- Change management is a systematic approach to deliberately regulating the changing nature of projects. Change control, which is a subpart of change

management, deals with controlling specific changes to a system.
• Security should be addressed in each phase of software development. It should not be addressed only at the end of development because of the added cost, time, and effort and the lack of functionality.

1111
• The attack surface is the collection of possible entry points for an attacker. The
reduction of this surface reduces the possible ways that an attacker can exploit a system.
• Threat modeling is a systematic approach used to understand how different threats could be realized and how a successful compromise could take place.
• The waterfall software development methodology follows a sequential approach that requires each phase to complete before the next one can begin.
• The prototyping methodology involves creating a sample of the code for proof-of-concept purposes.
• Incremental software development entails multiple development cycles that are carried out on a piece of software throughout its development stages.
• The spiral methodology is an iterative approach that emphasizes risk analysis per
iteration.
• Rapid Application Development (RAD) combines prototyping and iterative development procedures with the goal of accelerating the software development process.
• Agile methodologies are characterized by iterative and incremental development processes that encourage team-based collaboration, where flexibility and adaptability
are used instead of a strict process structure.
• Some organizations improve internal coordination and reduce friction by integrating
the development and operations (DevOps) teams or the development, operations, and security (DevSecOps) teams when developing software.
• An integrated product team (IPT) is a multidisciplinary development team with representatives from many or all the stakeholder populations.
• Capability Maturity Model Integration (CMMI) is a process improvement approach that provides organizations with the essential elements of effective processes, which will improve their performance.
• The CMMI model uses six maturity levels designated by the numbers 0 through 5. Each level represents the maturity level of the process quality and optimization.
The levels are organized as follows: 0 = Incomplete, 1 = Initial, 2 = Managed, 3 = Defined, 4 = Quantitatively Managed, 5 = Optimizing.
• The OWASP Software Assurance Maturity Model (SAMM) is specifically focused on secure software development and allows organizations to decide their target maturity levels within each of five critical business functions: Governance, Design,
Implementation, Verification, and Operations.

1112

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. The software development life cycle has several phases. Which of the following
lists these phases in the correct order?
A. Requirements gathering, design, development, maintenance, testing, release
B. Requirements gathering, design, development, testing, operations and

maintenance
C. Prototyping, build and fix, increment, test, maintenance
D. Prototyping, testing, requirements gathering, integration, testing
2. John is a manager of the application development department within his company. He needs to make sure his team is carrying out all of the correct testing
types and at the right times of the development stages. Which of the following accurately describe types of software testing that should be carried out?
i. Unit testing Testing individual components in a controlled environment where programmers validate data structure, logic, and boundary conditions
ii. Integration testing Verifying that components work together as outlined in design specifications
iii. Acceptance testing Ensuring that the code meets customer requirements
iv. Regression testing After a change to a system takes place, retesting to ensure functionality, performance, and protection
A. i, ii
B. ii, iii
C. i, ii, iv
D. i, ii, iii, iv
3. Marge has to choose a software development methodology that her team should follow. The application that her team is responsible for developing is a critical
application that can have few to no errors. Which of the following best describes
the type of methodology her team should follow?
A. Cleanroom
B. Joint Application Development (JAD)
C. Rapid Application Development (RAD)
D. Reuse methodology

↟Chapter 24: Software Development

1113
4. Which level of Capability Maturity Model Integration allows organizations to manage all projects across the organization and be proactive?
A. Defined

B. Incomplete
C. Managed
D. Optimizing

5. Mohammed is in charge of a large software development project with rigid requirements and phases that will probably be completed by different contractors.
Which methodology would be best?
A. Waterfall
B. Spiral
C. Prototyping
D. Agile

Use the following scenario to answer Questions 6–9. You're in charge of IT and security at
a midsize organization going through a growth stage. You decided to stand up your own
software development team and are about to start your first project: a knowledge base for
your customers. You think it can eventually grow to become the focal point of interaction
with your customers, offering a multitude of features. You've heard a lot about the Scrum
methodology and decide to try it for this project.
6. How would you go about documenting the requirements for this software system?
A. User stories
B. Use cases
C. System Requirements Specification (SRS)
D. Informally, since it's your first project

7. You are halfway through your first Scrum sprint and get a call from a senior vice
president insisting that you add a new feature immediately. How do you handle this request?
A. Add the feature to the next sprint
B. Change the current sprint to include the feature
C. Reset the project to the requirements gathering phase
D. Delay the new feature until the end of the project

8. Your software development team, being new to the organization, is struggling to work smoothly with other teams within the organization as needed to get the software into production securely. Which approach can help mitigate this internal friction?
A. DevSecOps
B. DevOps
C. Integrated Product Teams (IPT)
D. Joint Analysis Design (JAD) sessions

9. What would be the best approach to selectively mature your software development practices with a view to improving cybersecurity?
A. Software Assurance Maturity Model (SAMM)
B. Capability Maturity Model Integration (CMMI)
C. Kanban
D. Integrated product teams (IPTs)

Answers
1. B. The following outlines the common phases of the software development life cycle:
i. Requirements gathering
ii. Design
iii. Development
iv. Testing
v. Operations and maintenance

2. D. There are different types of tests the software should go through because there are different potential flaws to look for. 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 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
3. A. The listed software development methodologies and their definitions are as follows:
• Joint Application Development (JAD) A methodology that uses a team approach in application development in a workshop-oriented environment.

• Rapid Application Development (RAD) A methodology that combines the use of prototyping and iterative development procedures with the goal of accelerating the software development process.
• 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 critical applications that will be put through a strict certification process.
4. A. The six levels of Capability Maturity Integration Model are
• Incomplete Development process is ad hoc or even chaotic. Tasks are not always completed at all, so projects are regularly cancelled or abandoned.

• 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.
• 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.
• 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.
• 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.
• Optimizing The organization has budgeted and integrated plans for continuous process improvement, which allow it to quickly respond to opportunities and changes.

5. D. The Waterfall methodology is a very rigid approach that could be useful for projects in which all the requirements are fully understood up front or projects for which different organizations will perform the work at each phase. The Spiral, prototyping, and Agile methodologies are well suited for situations in which the requirements are not well understood, and don't lend themselves well to switching contractors midstream.

6. A. Any answer except "informally" would be a reasonable one, but since you are using an Agile methodology (Scrum), user stories is the best answer. The important point is that you document the requirements formally, so you can design a solution that meets all your users' needs.

7. A. The Scrum methodology allows the project to be reset by allowing product features to be added, changed, or removed at clearly defined points that typically happen at the conclusion of each sprint.

8. A. DevSecOps is the integration of development, security, and operations professionals into a software development team. This is a good way to solve the friction between developers and members of the security and operations staff.

9. A. CMMI and SAMM are the only maturity models among the possible answers. SAMM is the best answer because it allows for more granular maturity goals than CMMI does, and it is focused on security.

CHAPTER

Secure Software

This chapter presents the following:
• Programming languages
• Secure coding
• Security controls for software development
• Software security assessments
• Assessing the security of acquired software

A good programmer is someone who always looks both ways
before crossing a one-way street.
—Doug Linder

Quality can be defined as fitness for purpose. In other words, quality refers to how good or bad something is for its intended purpose. A high-quality car is good for transportation. We don't have to worry about it breaking down, failing to protect its occupants in a crash, or being easy for a thief to steal. When we need to go somewhere, we can count on a high-quality car to get us to wherever we need to go. Similarly, we don't have to worry about high-quality software crashing, corrupting our data under unforeseen circumstances, or being easy for someone to subvert. Sadly, many developers still think of functionality first (or only) when thinking about quality. When we look at it holistically, we see that quality is the most important concept in developing secure software. Every successful compromise of a software system relies on the exploitation of one or more vulnerabilities in it. Software vulnerabilities, in turn, are caused by defects in the design or implementation of code. The goal, then, is to develop software that is as free from defects or, in other words, as high quality as we can make it. In this chapter, we will discuss how secure software is quality software. We can't have one without the other. By applying the right processes, controls, and assessments, the outcome will be software that is more reliable and more difficult to exploit or subvert. Of course, these principles apply equally to software we develop in our own organizations and software that is developed for us by others.

Programming Languages and Concepts

All software is written in some type of programming language. Programming languages have gone through several generations over time, each generation building on the next, providing richer functionality and giving programmers more powerful tools as they evolve. The main categories of languages are machine, assembly, high-level, very high-level, and natural languages. Machine language is in a format that the computer's

processor
can understand and work with directly. Every processor family has its own machine
code instruction set, which is represented in a binary format (1 and 0) and is the
most fundamental form of programming language. Since this was pretty much the
only way to program the very first computers in the early 1950s, machine languages
are the first generation of programming languages. Early computers used only basic
binary instructions because compilers and interpreters were nonexistent at the time.
Programmers had to manually calculate and allot memory addresses and sequentially
feed instructions, as there was no concept of abstraction. Not only was programming in
binary extremely time consuming, it was also highly prone to errors. (If you think about
writing out thousands of 1's and 0's to represent what you want a computer to do, this
puts this approach into perspective.) This forced programmers to keep a tight rein on
their program lengths, resulting in programs that were very rudimentary.
An assembly language is considered a low-level programming language and is the
symbolic representation of machine-level instructions. It is "one step above" machine
language. It uses symbols (called mnemonics) to represent complicated binary codes.
Programmers using assembly language could use commands like ADD, PUSH, POP,
etc., instead of the binary codes (1001011010, etc.). Assembly languages use programs
called assemblers, which automatically convert these assembly codes into the necessary
machine-compatible binary language. To their credit, assembly languages drastically
reduced programming and debugging times, introduced the concept of variables, and
freed programmers from manually calculating memory addresses. But like machine code,
programming in an assembly language requires extensive knowledge of a computer's
architecture. It is easier than programming in binary format, but more challenging
compared to the high-level languages most programmers use today.
Programs written in assembly language are also hardware specific, so a program
written for an ARM-based processor would be incompatible with Intel-based systems;
thus, these types of languages are not portable. Once the program is written, it is fed
to an assembler, which translates the assembly language into machine language. The
assembler also replaces variable names in the assembly language program with actual
addresses at which their values will be stored in memory.
NOTE Assembly language allows for direct control of very basic activities

within a computer system, as in pushing data on a memory stack and popping data off a stack. Attackers commonly use assembly language to tightly control how malicious instructions are carried out on victim systems.

The third generation of programming languages started to emerge in the early 1960s.
They are known as high-level languages because of their refined programming structures.

High-level languages use abstract statements. Abstraction naturalizes multiple assembly
language instructions into a single high-level statement, such as IF – THEN – ELSE. This
allows programmers to leave low-level (system architecture) intricacies to the programming
language and focus on their programming objectives. In addition, high-level languages
are easier to work with compared to machine and assembly languages, as their syntax is
similar to human languages. The use of mathematical operators also simplifies arithmetic
and logical operations. This drastically reduces program development time and allows
for more simplified debugging. This means the programs are easier to write and mistakes
(bugs) are easier to identify. High-level languages are processor independent. Code written
in a high-level language can be converted to machine language for different processor
architectures using compilers and interpreters. When code is independent of a specific
processor type, the programs are portable and can be used on many different system types.
Fourth-generation languages (very high-level languages) were designed to further enhance the natural language approach instigated within the third-generation languages.
They focus on highly abstract algorithms that allow straightforward programming implementation in specific environments. The most remarkable aspect of fourthgeneration languages is that the amount of manual coding required to perform a specific
task may be ten times less than for the same task on a third-generation language. This is
an especially important feature because these languages have been developed to be used
by inexpert users and not just professional programmers.
As an analogy, let’s say that you need to pass a calculus exam. You need to be very
focused on memorizing the necessary formulas and applying the formulas to the

correct word problems on the test. Your focus is on how calculus works, not on how the calculator you use as a tool works. If you had to understand how your calculator is moving data from one transistor to the other, how the circuitry works, and how the calculator stores and carries out its processing activities just to use it for your test, this would be overwhelming. The same is true for computer programmers. If they had to worry about how the operating system carries out memory management functions, input/output activities, and how processor-based registers are being used, it would be difficult for them to also focus on real-world problems they are trying to solve with their software. Very high-level languages hide all of this background complexity and take care of it for the programmer.

The early 1990s saw the conception of the fifth generation of programming languages (natural languages). These languages approach programming from a completely different perspective. Program creation does not happen through defining algorithms and function statements, but rather by defining the constraints for achieving a specified result. The goal is to create software that can solve problems by itself instead of a programmer having to develop code to deal with individual and specific problems. The applications work more like a black box—a problem goes in and a solution comes out. Just as the introduction of assembly language eliminated the need for binary-based programming, the full impact of fifth-generation programming techniques may bring to an end the traditional programming approach. The ultimate target of fifth-generation languages is to eliminate the need for programming expertise and instead use advanced knowledgebased processing and artificial intelligence.

Language Levels

The “higher” the language, the more abstraction that is involved, which means the language hides details of how it performs its tasks from the software developer.

A programming language that provides a high level of abstraction frees the programmer from the need to worry about the intricate details of the computer system itself, as in registers, memory addresses, complex Boolean expressions, thread management, and so forth. The programmer can use simple statements such as “print” and does not need to worry about how the computer will actually get the data

over
to the printer. Instead, the programmer can focus on the core functionality that the
application is supposed to provide and not be bothered with the complex things taking place in the belly of the operating system and motherboard components. As an analogy, you do not need to understand how your engine or brakes work in your car—there is a level of abstraction. You just turn the steering wheel and step on
the pedal when necessary, and you can focus on getting to your destination.
There are so many different programming languages today, it is hard to fit them neatly in the five generations described in this chapter. These generations are the
classical way of describing the differences in software programming approaches and
what you will see on the CISSP exam.

The industry has not been able to fully achieve all the goals set out for these fifthgeneration languages. The human insight of programmers is still necessary to figure out
the problems that need to be solved, and the restrictions of the structure of a current
computer system do not allow software to “think for itself ” yet. We are getting closer to
achieving artificial intelligence within our software, but we still have a long way to go.
The following lists the basic software programming language generations:

- Generation one Machine language
- Generation two Assembly language
- Generation three High-level language
- Generation four Very high-level language
- Generation five Natural language

Assemblers, Compilers, Interpreters
No matter what type or generation of programming language is used, all of the instructions and data have to end up in a binary format for the processor to understand and
work with. Just like our food has to be broken down into specific kinds of molecules
for our body to be able to use it, all code must end up in a format that is consumable
by specific systems. Each programming language type goes through this transformation
through the use of assemblers, compilers, or interpreters.

Assemblers are tools that convert assembly language source code into machine language code. Assembly language consists of mnemonics, which are incomprehensible
to processors and therefore need to be translated into operation instructions.
Compilers are tools that convert high-level language statements into the necessary

machine-level format (.exe, .dll, etc.) for specific processors to understand. The compiler transforms instructions from a source language (high-level) to a target language (machine), sometimes using an external assembler along the way. This transformation allows the code to be executable. A programmer may develop an application in the C language, but when you purchase this application, you do not receive the source code; instead, you receive the executable code that runs on your type of computer. The source code was put through a compiler, which resulted in an executable file that can run on your specific processor type.

Compilers allow developers to create software code that can be developed once in a high-level language and compiled for various platforms. So, you could develop one piece of software, which is then compiled by five different compilers to allow it to be able to run on five different systems.

Figure 25-1 shows the process by which a high-level language is gradually transformed into machine language, which is the only language a processor can understand natively.

In this example, we have a statement that assigns the value 42 to the variable x. Once we feed the program containing this statement to a compiler, we end up with assembly language, which is shown in the middle of the figure. The way to set the value of a variable in assembly language is to literally move that value into wherever the variable is being stored. In this example, we are moving the hexadecimal value for 42 (which is 2a in hexadecimal, or 2ah) into the ax register in the processor. In order for the processor to execute this command, however, we still have to convert it into machine language, which is the job of the assembler. Note that it is way easier for a human coder to write x = 42 than it is to represent the same operation in either assembly or (worse yet) machine language.

If a programming language is considered “interpreted,” then a tool called an interpreter takes care of transforming high-level code to machine-level code. For example, applications that are developed in JavaScript, Python, or Perl can be run directly by an interpreter, without having to be compiled. The goal is to improve portability. The greatest advantage of executing a program in an interpreted environment is that the platform independence and memory management functions are part of an interpreter. The major

disadvantage
Figure 25-1
Converting
a high-level
language
statement
into machine
language code

C source code

x = 42;
Compiler

Assembly language

mov ax, 2ah
Assembler
10100010 10100010 10100010

PART VIII

Machine language

with this approach is that the program cannot run as a stand-alone application, requiring
the interpreter to be installed on the local machine.
NOTE Some languages, such as Java and Python, blur the lines between
interpreted and compiled languages by supporting both approaches. We'll
talk more about how Java does this in the next section.

From a security point of view, it is important to understand vulnerabilities that are
inherent in specific programming languages. For example, programs written in the C language could be vulnerable to buffer overrun and format string errors. The issue is
that some of the C standard software libraries do not check the length of the strings of data
they manipulate by default. Consequently, if a string is obtained from an untrusted source
(i.e., the Internet) and is passed to one of these library routines, parts of memory may be
unintentionally overwritten with untrustworthy data—this vulnerability can potentially
be used to execute arbitrary and malicious software. Some programming languages, such
as Java, perform automatic memory allocation as more space is needed; others, such as C,
require the developer to do this manually, thus leaving opportunities for error. Garbage collection is an automated way for software to carry out part of its memory

management tasks. A garbage collector identifies blocks of memory that were once allocated but are no longer in use and deallocates the blocks and marks them as free. It also gathers scattered blocks of free memory and combines them into larger blocks. It helps provide a more stable environment and does not waste precious memory. If garbage collection does not take place properly, not only can memory be used in an inefficient manner, an attacker could carry out a denial-of-service attack specifically to artificially commit all of a system's memory, rendering the system unable to function.

Nothing in technology seems to be getting any simpler, which makes learning this stuff much harder as the years go by. Ten years ago assembly, compiled, and interpreted languages were more clear-cut and their definitions straightforward. For the most part, only scripting languages required interpreters, but as languages have evolved they have become extremely flexible to allow for greater functionality, efficiency, and portability.

Many languages can have their source code compiled or interpreted depending upon the environment and user requirements.

Runtime Environments

What if you wanted to develop software that could run on many different environments without having to recompile it? This is known as portable code and it needs something that can sort of "translate" it to each different environment. That "translator" could be tuned to a particular type of computer but be able to run any of the portable code it understands. This is the role of runtime environments (RTEs), which function as miniature operating systems for the program and provide all the resources portable code needs.

One of the best examples of RTE usage is the Java programming language. Java is platform independent because it creates intermediate code, bytecode, which is not processor-specific. The Java Virtual Machine (JVM) converts the bytecode to the machine

Figure 25-2 The JVM interprets bytecode to machine code for that specific platform.

Java program

Java bytecode
Compiler

ter
pre
r
e
Int

Interpreter

Inte
rpre
ter

code that the processor on that particular system can understand (see Figure 25-2).
Despite its name, the JVM is not a full-fledged VM (as defined in Chapter 7). Instead,
it is a component of the Java RTE, together with a bunch of supporting files like class
libraries.
Let’s quickly walk through these steps:
1. A programmer creates a Java applet and runs it through a compiler.
2. The Java compiler converts the source code into bytecode (not processor-specific).
3. The user downloads the Java applet.
4. The JVM converts the bytecode into machine-level code (processor-specific).
5. The applet runs when called upon.

When an applet is executed, the JVM creates a unique RTE for it called a sandbox.
This sandbox is an enclosed environment in which the applet carries out its activities.
Applets are commonly sent over within a requested web page, which means the applet
executes as soon as it arrives. It can carry out malicious activity on purpose or accidentally
if the developer of the applet did not do his part correctly. So the sandbox strictly
limits the applet’s access to any system resources. The JVM mediates access to system
resources to ensure the applet code behaves and stays within its own sandbox. These
components are illustrated in Figure 25-3.
NOTE The Java language itself provides protection mechanisms, such as garbage collection, memory management, validating address usage, and a component that verifies adherence to predetermined rules.

However, as with many other things in the computing world, the bad guys have

figured out how to escape the confines and restrictions of the sandbox. Programmers
have figured out how to write applets that enable the code to access hard drives and

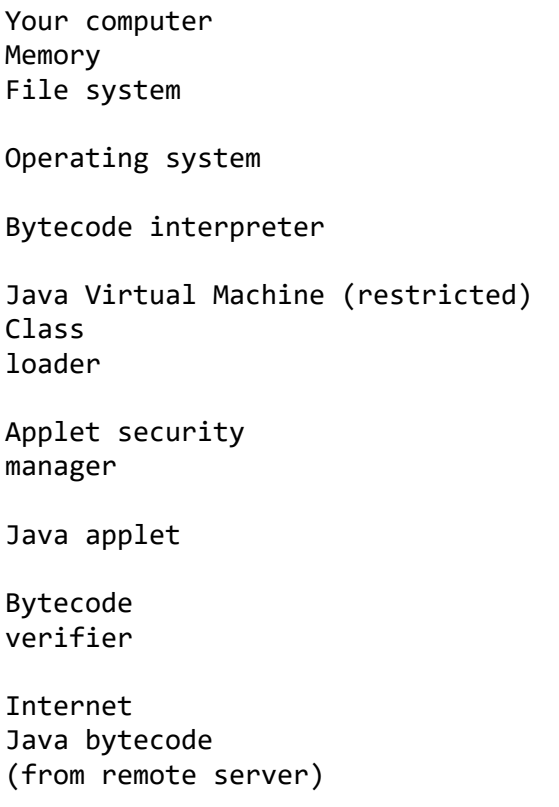
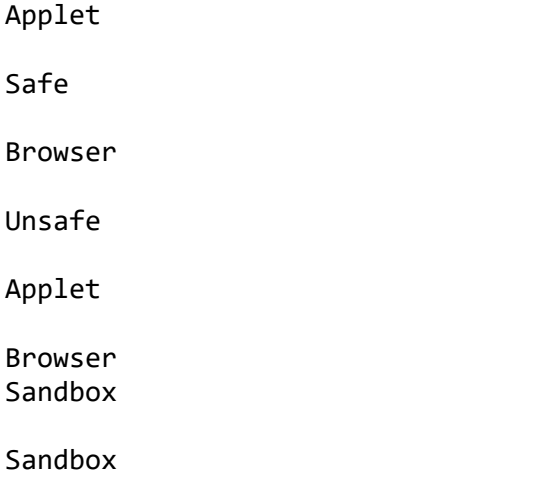


Figure 25-3

Java’s security model

resources that are supposed to be protected by the Java security scheme. This code can
be malicious in nature and cause destruction and mayhem to the user and her system.



Applet

Applet

Object-Oriented Programming Concepts
Software development used to be done by classic input–processing–output methods. This
development used an information flow model from hierarchical information structures.
Data was input into a program, and the program passed the data from the beginning to

end, performed logical procedures, and returned a result. Object-oriented programming
(OOP) methods perform the same functionality, but with different techniques that work
in a more efficient manner. First, you need to understand the basic concepts of OOP.
OOP works with classes and objects. A real-world object, such as a table, is a member
(or an instance) of a larger class of objects called “furniture.” The furniture class has a set of
attributes associated with it, and when an object is generated, it inherits these attributes.
The attributes may be color, dimensions, weight, style, and cost. These attributes apply
if a chair, table, or loveseat object is generated, also referred to as instantiated. Because
the table is a member of the class furniture, the table inherits all attributes defined for the
class (see Figure 25-4).
The programmer develops the class and all of its characteristics and attributes. The
programmer does not develop each and every object, which is the beauty of this approach.
As an analogy, let’s say you developed an advanced coffee maker with the goal of putting
Starbucks out of business. A customer punches the available buttons on your coffee
maker interface, ordering a large latte, with skim milk, vanilla and raspberry flavoring,
and an extra shot of espresso, where the coffee is served at 250 degrees. Your coffee maker
does all of this through automation and provides the customer with a lovely cup of coffee
exactly to her liking. The next customer wants a mocha Frothy Frappé, with whole milk
and extra foam. So the goal is to make something once (coffee maker, class), allow it to
accept requests through an interface, and create various results (cups of

coffee, objects)
depending upon the requests submitted.
But how does the class create objects based on requests? A piece of software that is
written in OOP will have a request sent to it, usually from another object. The requesting
object wants a new object to carry out some type of functionality. Let’s say that object
A wants object B to carry out subtraction on the numbers sent from A to B. When this
request comes in, an object is built (instantiated) with all of the necessary programming
code. Object B carries out the subtraction task and sends the result back to object A.

Figure 25-4
In object-oriented
inheritance, each
object belongs
to a class and
takes on the
attributes of
that class

Class: Furniture
Color
Dimensions
Weight
Style
Cost

Object: Table
Color = brown
Dimensions = 8×10
Weight = 34Ibs
Style = Western
Cost = 145

It does not matter what programming language the two objects are written in; what matters
is if they know how to communicate with each other. One object can communicate with
another object if it knows the application programming interface (API) communication
requirements. An API is the mechanism that allows objects to talk to each other (as
described in depth in the forthcoming section “Application Programming Interfaces”).
Let’s say you want to talk to Jorge, but can only do so by speaking French and

can only
use three phrases or less, because that is all Jorge understands. As long as you follow these
rules, you can talk to Jorge. If you don’t follow these rules, you can’t talk to Jorge.
TIP

An object is an instance of a class.

What’s so great about OOP? Figure 25-5 shows the difference between OOP and procedural programming, which is a non-OOP technique. Procedural programming is built on the concept of dividing a task into procedures that, when executed, accomplish
the task. This means that large applications can quickly become one big pile of code
(sometimes called spaghetti code). If you want to change something in this pile, you have
to go through all the program’s procedures to figure out what your one change is going
to break. If the program contains hundreds or thousands of lines of code, this is not an
easy or enjoyable task. Now, if you choose to write your program in an object-oriented

Object-oriented design
• Similar object classes
• Common interfaces
• Common usage
• Code reuse—inheritance
• Defers implementation and
algorithm decisions

Figure 25-5

Procedural vs. object-oriented programming

Procedural design
• Algorithm centered—forces early
implementation and algorithm
decisions
• Exposes more details
• Difficult to extend
• Difficult to maintain

language, you don’t have one monolithic application, but an application that is made
up of smaller components (objects). If you need to make changes or updates to some
functionality in your application, you can just change the code within the class that
creates the object carrying out that functionality, and you don’t have to worry

about
everything else the program actually carries out. The following breaks down the benefits
of OOP:

• Modularity The building blocks of software are autonomous objects, cooperating through the exchange of messages.
• Deferred commitment The internal components of an object can be redefined without changing other parts of the system.
• Reusability Classes are reused by other programs, though they may be refined through inheritance.
• Naturalness Object-oriented analysis, design, and modeling map to business needs and solutions.

Most applications have some type of functionality in common. Instead of developing
the same code to carry out the same functionality for ten different applications, using
OOP allows you to create the object only once and reuse it in other applications. This
reduces development time and saves money.

Now that we've covered the concepts of OOP, let's clarify the terminology. A method is
the functionality or procedure an object can carry out. An object may be constructed to
accept data from a user and to reformat the request so a back-end server can understand
and process it. Another object may perform a method that extracts data from a database
and populates a web page with this information. Or an object may carry out a withdrawal
procedure to allow the user of an ATM to extract money from her account.

The objects encapsulate the attribute values, which means this information is packaged
under one name and can be reused as one entity by other objects. Objects need to be able
to communicate with each other, and this happens by using messages that are sent to the
receiving object's API. If object A needs to tell object B that a user's checking account
must be reduced by $40, it sends object B a message. The message is made up of the
destination, the method that needs to be performed, and the corresponding arguments.

Figure 25-6 shows this example.

Messaging can happen in several ways. A given object can have a single connection
(one-to-one) or multiple connections (one-to-many). It is important to map these
communication paths to identify if information can flow in a way that is not intended.
This helps to ensure that sensitive data cannot be passed to objects of a lower
security level.

Object A

Message
(Object B, Withdrawal, 40.00)

Object B

PART VIII

Figure 25-6
Objects
communicate
via messages.

An object can have a shared portion and a private portion. The shared portion is the interface (API) that enables it to interact with other components. Messages enter through the interface to specify the requested operation, or method, to be performed. The private portion of an object is how it actually works and performs the requested operations. Other components need not know how each object works internally—only that it does the job requested of it. This is how data hiding is possible. The details of the processing are hidden from all other program elements outside the object. Objects communicate through welldefined interfaces; therefore, they do not need to know how each other works internally.

NOTE Data hiding is provided by encapsulation, which protects an object's private data from outside access. No object should be allowed to, or have the need to, access another object's internal data or processes.

These objects can grow to great numbers, so the complexity of understanding, tracking, and analyzing can get a bit overwhelming. Many times, the objects are shown in connection to a reference or pointer in documentation. Figure 25-7 shows how related objects are represented as a specific piece, or reference, in a bank ATM system. This enables analysts and developers to look at a higher level of operation and procedures without having to view each individual object and its code. Thus, this modularity provides for a more easily understood model.

Figure 25-7
Object
relationships
within a program

ATM

Display panel

Button selection

Functions

Printing receipts

Abstraction, as discussed earlier, is the capability to suppress unnecessary details so the important, inherent properties can be examined and reviewed. It enables the separation of conceptual aspects of a system. For example, if a software architect needs to understand how data flows through the program, she would want to understand the big pieces of the program and trace the steps the data takes from first being input into the program all the way until it exits the program as output. It would be difficult to understand this concept if the small details of every piece of the program were presented. Instead, through abstraction, all the details are suppressed so the software architect can understand a crucial part of the product. It is like being able to see a forest without having to look at each and every tree.
Each object should have specifications it adheres to. This discipline provides cleaner programming and reduces programming errors and omissions. The following list is an example of what should be developed for each object:

- Object name
- Attribute descriptions
- Attribute name
- Attribute content
- Attribute data type
- External input to object
- External output from object
- Operation descriptions
- Operation name
- Operation interface description
- Operation processing description
- Performance issues
- Restrictions and limitations
- Instance connections
- Message connections

The developer creates a class that outlines these specifications. When objects

are
instantiated, they inherit these attributes.
Each object can be reused as stated previously, which is the beauty of OOP. This enables
a more efficient use of resources and the programmer’s time. Different applications can
use the same objects, which reduces redundant work, and as an application grows in
functionality, objects can be easily added and integrated into the original structure.
The objects can be catalogued in a library, which provides an economical way for more
than one application to call upon the objects (see Figure 25-8). The library provides an
index and pointers to where the objects actually live within the system or on another system.

Figure 25-8
Applications
locate the
necessary objects
through a library
index.

Library
Application

Application
Application

Components

When applications are developed in a modular approach, like object-oriented methods, components can be reused, complexity is reduced, and parallel development
can be done. These characteristics allow for fewer mistakes, easier modification, resource
efficiency, and more timely coding than the classic programming languages. OOP also provides functional independence, which means each module addresses a specific
subfunction of requirements and has an interface that is easily understood by other parts
of the application.
An object is encapsulated, meaning the data structure (the operation’s functionality)
and the acceptable ways of accessing it are grouped into one entity. Other objects,
subjects, and applications can use this object and its functionality by accessing it through
controlled and standardized interfaces and sending it messages (see Figure 25-9).

Cohesion and Coupling

Cohesion reflects how many different types of tasks a module can carry out. If a module carries out only one task (i.e., subtraction) or tasks that are very similar (i.e., subtract, add, multiply), it is described as having high cohesion, which is a good thing. The higher the cohesion, the easier it is to update or modify the module and not affect other modules that interact with it. This also means the module is easier to reuse and maintain because it is more straightforward when compared to a module with low cohesion. An object with low cohesion carries out multiple different tasks and increases the complexity of the module, which makes it harder to maintain and reuse. So, you want your objects to be focused, manageable, and understandable. Each object should carry out a single function or similar functions. One object should not carry out mathematical operations, graphic rendering, and cryptographic functions—these are separate functionality types, and keeping track of this level of complexity would be confusing. If you are attempting to create complex multifunction objects, you are trying to shove too much into one object. Objects should carry out modular, simplistic functions—that is the whole point of OOP.

↟Chapter 25: Secure Software

Data encapsulation

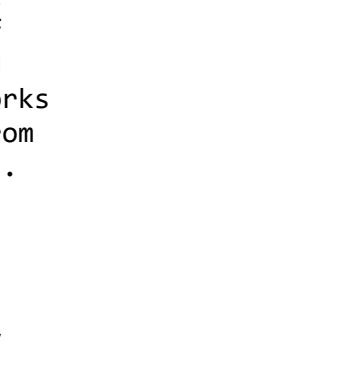


Figure 25-9
The different components of an object and the way it works are hidden from other objects.

Object

PART VIII

Coupling is a measurement that indicates how much interaction one module requires
to carry out its tasks. If a module has low (loose) coupling, this means the module does
not need to communicate with many other modules to carry out its job. High (tight)
coupling means a module depends upon many other modules to carry out its tasks. Low
coupling is more desirable because the module is easier to understand and easier to reuse,
and it can be changed without affecting many modules around it. Low coupling indicates
that the programmer created a well-structured module. As an analogy, a company would
want its employees to be able to carry out their individual jobs with the least amount of
dependencies on other workers. If Joe has to talk with five other people just to get one
task done, too much complexity exists, the task is too time-consuming, and the potential
for errors increases with every interaction.
If modules are tightly coupled, the ripple effect of changing just one module can
drastically affect the other modules. If they are loosely coupled, this level of complexity
decreases.
An example of low coupling would be one module passing a variable value to another
module. As an example of high coupling, Module A would pass a value to Module B,
another value to Module C, and yet another value to Module D. Module A could not
complete its tasks until Modules B, C, and D completed their tasks and returned results
to Module A.

EXAM TIP Objects should be self-contained and perform a single logical function, which is high cohesion. Objects should not drastically affect each other, which is low coupling.

The level of complexity involved with coupling and cohesion can directly impact the security level of a program. The more complex something is, the harder it is to
secure. Developing “tight code” not only allows for efficiencies and effectiveness but
also reduces the software’s attack surface. Decreasing complexity where possible reduces
the number of potential holes a bad guy can sneak through. As an analogy, if you were

responsible for protecting a facility, your job would be easier if the facility had a small number of doors, windows, and people coming in and out of it. The smaller number of variables and moving pieces would help you keep track of things and secure them.

Application Programming Interfaces

When we discussed some of the attributes of object-oriented development, we spent a bit of time on the concept of abstraction. Essentially, abstraction is all about defining what a class or object does and ignoring how it accomplishes it internally. An application programming interface (API) specifies the manner in which a software component interacts with other software components. We already saw in Chapter 9 how this can come in handy in the context of Trusted Platform Modules (TPMs) and how they constrain communications with untrusted modules. APIs create checkpoints where security controls can be easily implemented. Furthermore, they encourage software reuse and also make the software more maintainable by localizing the changes that need to be made while eliminating (or at least reducing) cascading effects of fixes or changes.

Besides the advantages of reduced effort and improved maintainability, APIs often are required to employ the underlying operating system's functionality. Apple macOS and iOS, Google Android, and Microsoft Windows all require developers to use standard APIs for access to operating system functionality such as opening and closing files and network connections, among many others. All these major vendors restrict the way in which their APIs are used, most notably by ensuring that any parameter that is provided to them is first checked to ensure it is not malformed, invalid, or malicious, which is something we should all do when we are dealing with APIs.

Parameter validation refers to confirming that the parameter values being received by an application are within defined limits before they are processed by the system. In a client/server architecture, validation controls may be placed on the client side prior to submitting requests to the server. Even when these controls are employed, the server should perform parallel validation of inputs prior to processing them because a client has fewer controls than a server and may have been compromised or bypassed.

Software Libraries

APIs are perhaps most familiar to us in the context of software libraries. A software library

is a collection of components that do specific tasks that are useful to many other components. For example, there are software libraries for various encryption algorithms,

managing network connections, and displaying graphics. Libraries allow software developers to work on whatever makes their program unique, while leveraging known-good
code for the tasks that similar programs routinely do. The programmer simply needs
to understand the API for the libraries she intends to use. This reduces the amount of
new code that the programmer needs to develop, which in turn makes the code easier to
secure and maintain.
Using software libraries has potential risks, and these risks must be mitigated as part
of secure software development practices. The main risk is that, because the libraries are
reused across multiple projects in multiple organizations, any defect in these libraries
propagates through every program that uses them. In fact, according to Veracode's 2020
report "State of Software Security: Open Source Edition," seven in ten applications use
at least one open-source library with a security flaw, which makes those applications
vulnerable. Keep in mind that these are open-source libraries, which (as we will discuss
later in this chapter) are subject to examination by any number of security researchers
looking for bugs. If you use proprietary libraries (including your own), it may be much
harder to find these vulnerabilities before the threat actors do.

Secure Software Development

So far in this chapter (and the previous one), we've discussed software development in
general terms, pointing out potential security pitfalls along the way. We now turn our
attention to how we can bake security into our software from the ground up. To do so,
however, we have to come from the top down, meaning we need an organizational policy
document that clearly identifies the strategic goals, responsibilities, and authorities for
mitigating risks associated with building or acquiring software. If the executive leadership doesn't push this, it just won't happen, and the policy document puts everyone on
notice that secure coding is an organizational priority.
Secure coding is a set of practices that reduces (to acceptable levels) the risk of

vulnerabilities in our software. No software will ever be 100 percent secure, but we can
sure make it hard for threat actors to find and exploit any remaining vulnerabilities if we
apply secure coding guidelines and standards to our projects.

Source Code Vulnerabilities

PART VIII

A source code vulnerability is a defect in code that provides a threat actor an opportunity
to compromise the security of a software system. All code has defects (or bugs), but a vulnerability is a particularly dangerous one. Source code vulnerabilities are typically caused
by two types of flaws: design and implementation. A design flaw is one that, even if the
programmer did everything perfectly right, would still cause the vulnerability. An implementation flaw is one that stems from a programmer who incorrectly implemented a part
of a good design. For example, suppose you are building an e-commerce application that
collects payment card information from your customers and stores it for their future purchases. If you design the system to store the card numbers unencrypted, that would be a
design flaw. If, on the other hand, you design the system to encrypt the data as soon as it

is captured but a programmer incorrectly calls the encryption function, resulting in the
card number being stored in plaintext, that would be an implementation vulnerability.
Source code vulnerabilities are particularly problematic when they exist in externally
facing systems such as web applications. These accounted for 39 percent of the external
attacks carried out, according to Forrester’s report “The State of Application Security,
2021.” Web applications deserve particular attention because of their exposure. The Open Web Application Security Project (OWASP) is an organization that deals specifically with web security issues. OWASP offers numerous tools, articles, and resources
that developers can utilize to create secure software, and it also has individual member
meetings (chapters) throughout the world. OWASP provides development guidelines, testing procedures, and code review steps, but is probably best known for its OWASP
Top 10 list of web application security risks. The following is the most recent Top 10 list
as of this writing, from 2017 (the 2021 version should be published by the time you’re