Chapter 10: Information Systems Development

Learning Objectives

Upon successful completion of this chapter, you will be able to:

- explain the overall process of developing a new software application;
- explain the differences between software development methodologies;
- understand the different types of programming languages used to develop software;
- understand some of the issues surrounding the development of websites and mobile applications; and
- identify the four primary implementation policies.

Introduction

When someone has an idea for a new function to be performed by a computer, how does that idea become reality? If a company wants to implement a new business process and needs new hardware or software to support it, how do they go about making it happen? In this chapter, we will discuss the different methods of taking those ideas and bringing them to reality, a process known as information systems development.

Programming

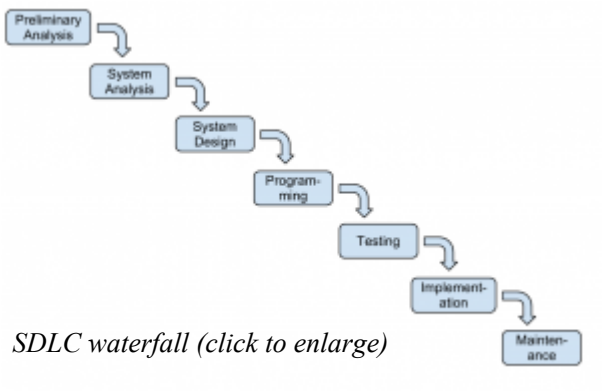
As we learned in chapter 2, software is created via programming. Programming is the process of creating a set of logical instructions for a digital device to follow using a programming language. The process of programming is sometimes called "coding" because the syntax of a programming language is not in a form that everyone can understand – it is in "code."

The process of developing good software is usually not as simple as sitting down and writing some code. True, sometimes a programmer can quickly write a short program to solve a need. But most of the time, the creation of software is a resource-intensive process that involves several different groups of people in an organization. In the following sections, we are going to review several different methodologies for software development.

Systems-Development Life Cycle

The first development methodology we are going to review is the systems-development life cycle (SDLC). This methodology was first developed in the 1960s to manage the large software projects associated with corporate systems running on mainframes. It is a very structured and risk-averse methodology designed to manage large projects that included multiple programmers and systems that would have a large impact on the organization.

Various definitions of the SDLC methodology exist, but most contain the following phases.



SDLC waterfall (click to enlarge)

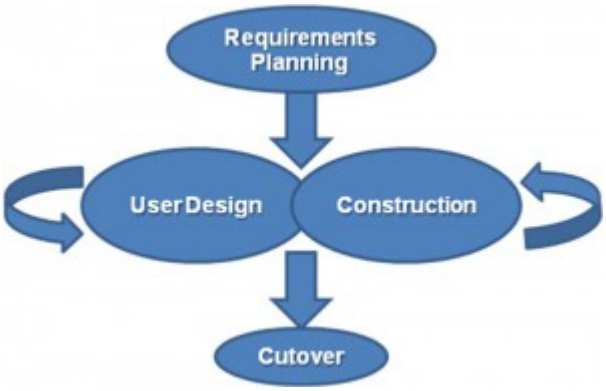
1. Preliminary Analysis. In this phase, a review is done of the request. Is creating a solution possible? What alternatives exist? What is currently being done about it? Is this project a good fit for our organization? A key part of this step is a feasibility analysis, which includes an analysis of the technical feasibility (is it possible to create this?), the economic feasibility (can we afford to do this?), and the legal feasibility (are we allowed to do this?). This step is important in determining if the project should even get started.
2. System Analysis. In this phase, one or more system analysts work with different stakeholder groups to determine the specific requirements for the new system. No programming is done in this step. Instead, procedures are documented, key players are interviewed, and data requirements are developed in order to get an overall picture of exactly what the system is supposed to do. The result of this phase is a system-requirements document.
3. System Design. In this phase, a designer takes the system-requirements document created in the previous phase and develops the specific technical details required for the system. It is in this phase that the business requirements are translated into specific technical requirements. The design for the user interface, database, data inputs and outputs, and reporting are developed here. The result of this phase is a system-design document. This document will have everything a programmer will need to actually create the system.
4. Programming. The code finally gets written in the programming phase. Using the system-design document as a guide, a programmer (or team of programmers) develop the program. The result of this phase is an initial working program that meets the requirements laid out in the system-analysis phase and the design developed in the system-design phase.
5. Testing. In the testing phase, the software program developed in the previous phase is put through a series of structured tests. The first is a unit test, which tests individual parts of the code for errors or bugs. Next is a system test, where the different components of the system are tested to ensure that they work together properly. Finally, the user-acceptance test allows those that will be using the software to test the system to ensure that it meets their standards. Any bugs, errors, or problems found during testing are addressed and then tested again.

6. Implementation. Once the new system is developed and tested, it has to be implemented in the organization. This phase includes training the users, providing documentation, and conversion from any previous system to the new system. Implementation can take many forms, depending on the type of system, the number and type of users, and how urgent it is that the system become operational. These different forms of implementation are covered later in the chapter.
7. Maintenance. This final phase takes place once the implementation phase is complete. In this phase, the system has a structured support process in place: reported bugs are fixed and requests for new features are evaluated and implemented; system updates and backups are performed on a regular basis.

The SDLC methodology is sometimes referred to as the waterfall methodology to represent how each step is a separate part of the process; only when one step is completed can another step begin. After each step, an organization must decide whether to move to the next step or not. This methodology has been criticized for being quite rigid. For example, changes to the requirements are not allowed once the process has begun. No software is available until after the programming phase.

Again, SDLC was developed for large, structured projects. Projects using SDLC can sometimes take months or years to complete. Because of its inflexibility and the availability of new programming techniques and tools, many other software-development methodologies have been developed. Many of these retain some of the underlying concepts of SDLC but are not as rigid.

Rapid Application Development



The RAD methodology (Public Domain)

Rapid application development (RAD) is a software-development (or systems-development) methodology that focuses on quickly building a working model of the software, getting feedback from users, and then using that feedback to update the working model. After several iterations of development, a final version is developed and implemented.

The RAD methodology consists of four phases:
1. Requirements Planning. This phase is similar to the preliminary-analysis, system-analysis, and design phases of the SDLC. In this phase, the overall requirements for the system are defined, a team is identified, and feasibility is determined.
2. User Design. In this phase, representatives of the users work with the system analysts, designers, and programmers to interactively create the design of the system. One technique for working with all of these various stakeholders is the so-called JAD session. JAD is an acronym for joint application development. A JAD session gets all of the stakeholders together to have a

structured discussion about the design of the system. Application developers also sit in on this meeting and observe, trying to understand the essence of the requirements.

3. Construction. In the construction phase, the application developers, working with the users, build the next version of the system.This is an interactive process, and changes can be made as developers are working on the program. This step is executed in parallel with the User Design step in an iterative fashion, until an acceptable version of the product is developed.

4. Cutover. In this step, which is similar to the implementation step of the SDLC, the system goes live. All steps required to move from the previous state to the use of the new system are completed here.

As you can see, the RAD methodology is much more compressed than SDLC. Many of the SDLC steps are combined and the focus is on user participation and iteration. This methodology is much better suited for smaller projects than SDLC and has the added advantage of giving users the ability to provide feedback throughout the process. SDLC requires more documentation and attention to detail and is well suited to large, resource-intensive projects. RAD makes more sense for smaller projects that are less resource-intensive and need to be developed quickly.

Agile Methodologies

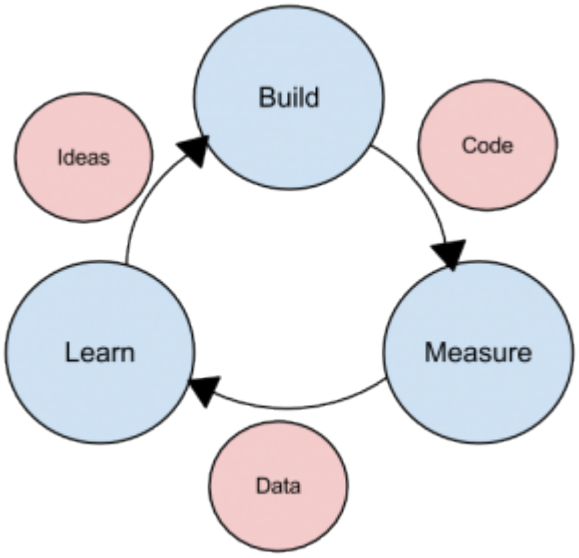
Agile methodologies are a group of methodologies that utilize incremental changes with a focus on quality and attention to detail. Each increment is released in a specified period of time (called a time box), creating a regular release schedule with very specific objectives. While considered a separate methodology from RAD, they share some of the same principles: iterative development, user interaction, ability to change. The agile methodologies are based on the "Agile Manifesto," first released in 2001.

The characteristics of agile methods include:

- small cross-functional teams that include development-team members and users;
- daily status meetings to discuss the current state of the project;
- short time-frame increments (from days to one or two weeks) for each change to be completed; and
- at the end of each iteration, a working project is completed to demonstrate to the stakeholders.

The goal of the agile methodologies is to provide the flexibility of an iterative approach while ensuring a quality product.

Lean Methodology



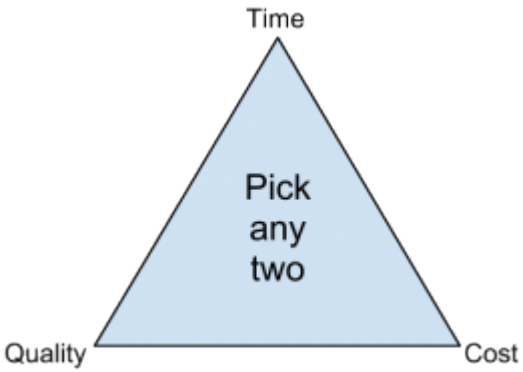
The lean methodology (click to enlarge)

One last methodology we will discuss is a relatively new concept taken from the business bestseller *The Lean Startup,* by Eric Reis. In this methodology, the focus is on taking an initial idea and developing a minimum viable product (MVP). The MVP is a working software application with just enough functionality to demonstrate the idea behind the project. Once the MVP is developed, it is given to potential users for review. Feedback on the MVP is generated in two forms: (1) direct observation and discussion with the users, and (2) usage statistics gathered from the software itself. Using these two forms of feedback, the team determines whether they should continue in the same direction or rethink the core idea behind the project, change the functions, and create a new MVP. This change in strategy is called a pivot. Several iterations of the MVP are developed, with new functions added each time based on the feedback, until a final product is completed.

The biggest difference between the lean methodology and the other methodologies is that the full set of requirements for the system are not known when the project is launched. As each iteration of the project is released, the statistics and feedback gathered are used to determine the requirements. The lean methodology works best in an entrepreneurial environment where a company is interested in determining if their idea for a software application is worth developing.

Sidebar: The Quality Triangle

When developing software, or any sort of product or service, there exists a tension between the developers and the different stakeholder groups, such as management, users, and investors. This tension relates to how quickly the software can be developed (time), how much money will be spent (cost), and how well it will be built (quality). The quality triangle is a simple concept. It states that for any product or service being developed, you can only address two of the following: time, cost, and quality.



The quality triangle

So what does it mean that you can only address two of the three? It means that you cannot complete a low-cost, high-quality project in a small amount of time. However, if you are willing or able to spend a lot of money, then a project can be completed quickly with high-quality results (through hiring more good programmers). If a project's completion date is not a priority, then it can be completed at a lower cost with higher-quality results. Of

course, these are just generalizations, and different projects may not fit this model perfectly. But overall, this model helps us understand the tradeoffs that we must make when we are developing new products and services.

Programming Languages

As I noted earlier, software developers create software using one of several programming languages. A programming language is an artificial language that provides a way for a programmer to create structured code to communicate logic in a format that can be executed by the computer hardware. Over the past few decades, many different types of programming languages have evolved to meet many different needs. One way to characterize programming languages is by their "generation."

Generations of Programming Languages

Early languages were specific to the type of hardware that had to be programmed; each type of computer hardware had a different low-level programming language (in fact, even today there are differences at the lower level, though they are now obscured by higher-level programming languages). In these early languages, very specific instructions had to be entered line by line – a tedious process.

First-generation languages are called machine code. In machine code, programming is done by directly setting actual ones and zeroes (the bits) in the program using binary code. Here is an example program that adds 1234 and 4321 using machine language:

```
10111001 00000000
11010010 10100001
00000100 00000000
10001001 00000000
00001110 10001011
00000000 00011110
00000000 00011110
00000000 00000010
10111001 00000000
11100001 00000011
00010000 11000011
10001001 10100011
00001110 00000100
00000010 00000000
```

Assembly language is the second-generation language. Assembly language gives english-like phrases to the machine-code instructions, making it easier to program. An assembly-language program must be run through an assembler, which converts it into machine code. Here is an example program that adds 1234 and 4321 using assembly language:

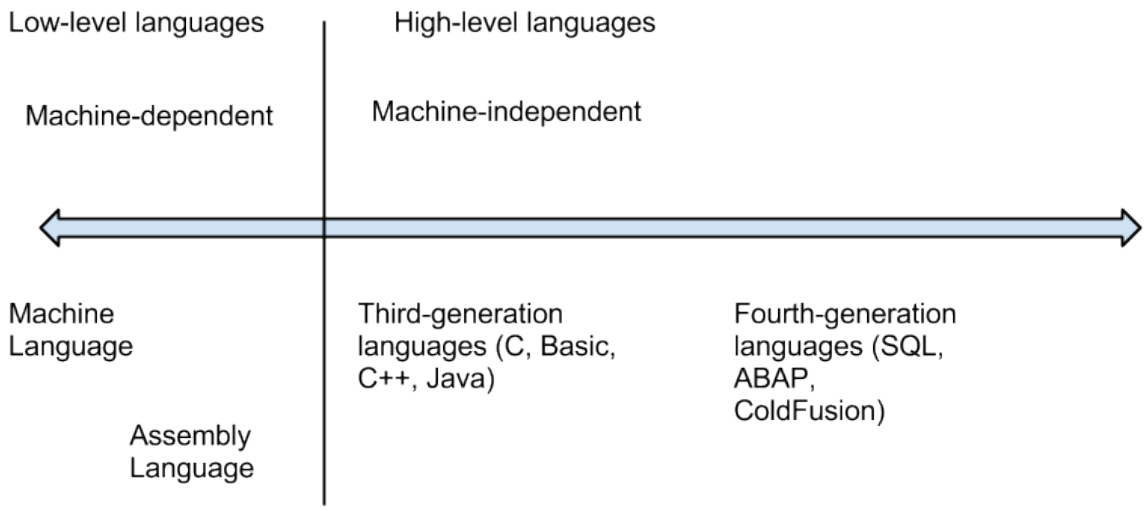
```
MOV CX,1234
MOV DS:[0],CX
MOV CX,4321
MOV AX,DS:[0]
MOV BX,DS:[2]
ADD AX,BX
MOV DS:[4],AX
```

Third-generation languages are not specific to the type of hardware on which they run and are much more like spoken languages. Most third-generation languages must be compiled, a process that converts them into machine code. Well-known third-generation languages include BASIC, C, Pascal, and Java. Here is an example using BASIC:

```
A=1234
B=4321
C=A+B
END
```

Fourth-generation languages are a class of programming tools that enable fast application development using intuitive interfaces and environments. Many times, a fourth-generation language has a very specific purpose, such as database interaction or report-writing. These tools can be used by those with very little formal training in programming and allow for the quick development of applications and/or functionality. Examples of fourth-generation languages include: Clipper, FOCUS, FoxPro, SQL, and SPSS.

Why would anyone want to program in a lower-level language when they require so much more work? The answer is similar to why some prefer to drive stick-shift automobiles instead of automatic transmission: control and efficiency. Lower-level languages, such as assembly language, are much more efficient and execute much more quickly. You have finer control over the hardware as well. Sometimes, a combination of higher- and lower-level languages are mixed together to get the best of both worlds: the programmer will create the overall structure and interface using a higher-level language but will use lower-level languages for the parts of the program that are used many times or require more precision.



The programming language spectrum (click to enlarge)

Compiled vs. Interpreted

Besides classifying a program language based on its generation, it can also be classified by whether it is compiled or interpreted. As we have learned, a computer language is written in a human-readable form. In a compiled language, the program code is translated into a machine-readable form called an executable that can be run on the hardware. Some well-known compiled languages include C, C++, and COBOL.

An interpreted language is one that requires a runtime program to be installed in order to execute. This runtime program then interprets the program code line by line and runs it. Interpreted languages are generally easier to work with but also are slower and require more system resources. Examples of popular interpreted languages include BASIC, PHP, PERL, and Python. The web languages of HTML and Javascript would also be considered interpreted because they require a browser in order to run.

The Java programming language is an interesting exception to this classification, as it is actually a hybrid of the two. A program written in Java is partially compiled to create a program that can be understood by the Java Virtual Machine (JVM). Each type of operating system has its own JVM which must be installed, which is what allows Java programs to run on many different types of operating systems.

Procedural vs. Object-Oriented

A procedural programming language is designed to allow a programmer to define a specific starting point for the program and then execute sequentially. All early programming languages worked this way. As user interfaces became more interactive and graphical, it made sense for programming languages to evolve to allow the user to define the flow of the program. The object-oriented programming language is set up so that the programmer defines "objects" that can take certain actions based on input from the user. In other words, a procedural program focuses on the sequence of activities to be performed; an object-oriented program focuses on the different items being manipulated.

For example, in a human-resources system, an "EMPLOYEE" object would be needed. If the program needed to retrieve or set data regarding an employee, it would first create an employee object in the program and then set or retrieve the values needed. Every object has properties, which are descriptive

fields associated with the object. In the example below, an employee object has the properties “Name”, “Employee number”, “Birthdate” and “Date of hire”. An object also has “methods”, which can take actions related to the object. In the example, there are two methods. The first is “ComputePay()”, which will return the current amount owed the employee. The second is “ListEmployees()”, which will retrieve a list of employees who report to this employee.

Object: EMPLOYEE
Name Employee number Birthdate Date of hire
ComputePay() ListEmployees()

Figure: An example of an object

Sidebar: What is COBOL?

If you have been around business programming very long, you may have heard about the COBOL programming language. COBOL is a procedural, compiled language that at one time was the primary programming language for business applications. Invented in 1959 for use on large mainframe computers, COBOL is an abbreviation of common business-oriented language. With the advent of more efficient programming languages, COBOL is now rarely seen outside of old, legacy applications.

Programming Tools

To write a program, a programmer needs little more than a text editor and a good idea. However, to be productive, he or she must be able to check the syntax of the code, and, in some cases, compile the code. To be more efficient at programming, additional tools, such as an integrated development environment (IDE) or computer-aided software-engineering (CASE) tools, can be used.

Integrated Development Environment

For most programming languages, an IDE can be used. An IDE provides a variety of tools for the programmer, and usually includes:

- an editor for writing the program that will color-code or highlight keywords from the programming language;
- a help system that gives detailed documentation regarding the programming language;
- a compiler/interpreter, which will allow the programmer to run the program;
- a debugging tool, which will provide the programmer details about the execution of the program in order to resolve problems in the code; and

- a check-in/check-out mechanism, which allows for a team of programmers to work together on a project and not write over each other's code changes.

Probably the most popular IDE software package right now is Microsoft's Visual Studio. Visual Studio is the IDE for all of Microsoft's programming languages, including Visual Basic, Visual C++, and Visual C#.

CASE Tools

While an IDE provides several tools to assist the programmer in writing the program, the code still must be written. Computer-aided software-engineering (CASE) tools allow a designer to develop software with little or no programming. Instead, the CASE tool writes the code for the designer. CASE tools come in many varieties, but their goal is to generate quality code based on input created by the designer.

Sidebar: Building a Website

In the early days of the World Wide Web, the creation of a website required knowing how to use hypertext markup language (HTML). Today, most websites are built with a variety of tools, but the final product that is transmitted to a browser is still HTML. HTML, at its simplest, is a text language that allows you to define the different components of a web page. These definitions are handled through the use of HTML tags, which consist of text between brackets. For example, an HTML tag can tell the browser to show a word in italics, to link to another web page, or to insert an image. In the example below, some text is being defined as a heading while other text is being emphasized.

```
<h1>This is a first-level heading</h1>
Here is some text. <em>Here is some emphasized text.</em>
<h2>Here is a second-level heading</h2>
Here is some more text.
```

Simple HTML

This is a first-level heading

Here is some text. *Here is some emphasized text.*

Here is a second-level heading

Here is some more text.

Simple HTML output

While HTML is used to define the *components* of a web page, cascading style sheets (CSS) are used to define the *styles* of the components on a page. The use of CSS allows the style of a website to be set and stay consistent throughout. For example, if the designer wanted all first-level headings (h1) to be blue and centered, he or she could set the "h1″ style to match. The following example shows how this might look.

```
<style>
h1
{
color:blue;
text-align:center;
}
</style>
<h1>This is a first-level heading</h1>
Here is some text. <em>Here is some emphasized text.</em>
<h2>This is a second-level heading</h2>
Here is some more text.
```

HTML with CSS

HTML with CSS output

The combination of HTML and CSS can be used to create a wide variety of formats and designs and has been widely adopted by the web-design community. The standards for HTML are set by a governing body called the World Wide Web Consortium. The current version of HTML is HTML 5, which includes new standards for video, audio, and drawing.

When developers create a website, they do not write it out manually in a text editor. Instead, they use web design tools that generate the HTML and CSS for them. Tools such as Adobe Dreamweaver allow the designer to create a web page that includes images and interactive elements without writing a single line of code. However, professional web designers still need to learn HTML and CSS in order to have full control over the web pages they are developing.

Build vs. Buy

When an organization decides that a new software program needs to be developed, they must determine if it makes more sense to build it themselves or to purchase it from an outside company. This is the "build vs. buy" decision.

There are many advantages to purchasing software from an outside company. First, it is generally less expensive to purchase a software package than to build it. Second, when a software package is purchased, it is available much more quickly than if the package is built in-house. Software applications can take months or years to build; a purchased package can be up and running within a month. A purchased package has already been tested and many of the bugs have already been worked out. It is the role of a systems integrator to make various purchased systems and the existing systems at the organization work together.

There are also disadvantages to purchasing software. First, the same software you are using can be used by your competitors. If a company is trying to differentiate itself based on a business process that is in

that purchased software, it will have a hard time doing so if its competitors use the same software. Another disadvantage to purchasing software is the process of customization. If you purchase a software package from a vendor and then customize it, you will have to manage those customizations every time the vendor provides an upgrade. This can become an administrative headache, to say the least!

Even if an organization determines to buy software, it still makes sense to go through many of the same analyses that they would do if they were going to build it themselves. This is an important decision that could have a long-term strategic impact on the organization.

Web Services

As we saw in chapter 3, the move to cloud computing has allowed software to be looked at as a service. One option companies have these days is to license functions provided by other companies instead of writing the code themselves. These are called web services, and they can greatly simplify the addition of functionality to a website.

For example, suppose a company wishes to provide a map showing the location of someone who has called their support line. By utilizing Google Maps API web services, they can build a Google Map right into their application. Or a shoe company could make it easier for its retailers to sell shoes online by providing a shoe-size web service that the retailers could embed right into their website.

Web services can blur the lines between "build vs. buy." Companies can choose to build a software application themselves but then purchase functionality from vendors to supplement their system.

End-User Computing

In many organizations, application development is not limited to the programmers and analysts in the information-technology department. Especially in larger organizations, other departments develop their own department-specific applications. The people who build these are not necessarily trained in programming or application development, but they tend to be adept with computers. A person, for example, who is skilled in a particular software package, such as a spreadsheet or database package, may be called upon to build smaller applications for use by his or her own department. This phenomenon is referred to as end-user development, or end-user computing.

End-user computing can have many advantages for an organization. First, it brings the development of applications closer to those who will use them. Because IT departments are sometimes quite backlogged, it also provides a means to have software created more quickly. Many organizations encourage end-user computing to reduce the strain on the IT department.

End-user computing does have its disadvantages as well. If departments within an organization are developing their own applications, the organization may end up with several applications that perform similar functions, which is inefficient, since it is a duplication of effort. Sometimes, these different versions of the same application end up providing different results, bringing confusion when departments interact. These applications are often developed by someone with little or no formal training in programming. In these cases, the software developed can have problems that then have to be resolved by the IT department.

End-user computing can be beneficial to an organization, but it should be managed. The IT department should set guidelines and provide tools for the departments who want to create their own solutions. Communication between departments will go a long way towards successful use of end-user computing.

Sidebar: Building a Mobile App

In many ways, building an application for a mobile device is exactly the same as building an application for a traditional computer. Understanding the requirements for the application, designing the interface, working with users – all of these steps still need to be carried out.

So what's different about building an application for a mobile device? In some ways, mobile applications are more limited. An application running on a mobile device must be designed to be functional on a smaller screen. Mobile applications should be designed to use fingers as the primary pointing device. Mobile devices generally have less available memory, storage space, and processing power.

Mobile applications also have many advantages over applications built for traditional computers. Mobile applications have access to the functionality of the mobile device, which usually includes features such as geolocation data, messaging, the camera, and even a gyroscope.

One of the most important questions regarding development for mobile devices is this: Do we want to develop an app at all? A mobile app is an expensive proposition, and it will only run on one type of mobile device at a time. For example, if you create an iPhone app, users with Android phones are out of luck. Each app takes several thousand dollars to create, so this may not be the best use of your funds.

Many organizations are moving away from developing a specific app for a mobile device and are instead making their websites more functional on mobile devices. Using a web-design framework called responsive design, a website can be made highly functional no matter what type of device is browsing it. With a responsive website, images resize themselves based on the size of the device's screen, and text flows and sizes itself properly for optimal viewing. You can find out more about responsive design here.

Implementation Methodologies

Once a new system is developed (or purchased), the organization must determine the best method for implementing it. Convincing a group of people to learn and use a new system can be a very difficult process. Using new software, and the business processes it gives rise to, can have far-reaching effects within the organization.

There are several different methodologies an organization can adopt to implement a new system. Four of the most popular are listed below.

- Direct cutover. In the direct-cutover implementation methodology, the organization selects a particular date that the old system is not going to be used anymore. On that date, the users begin using the new system and the old system is unavailable. The advantages to using this methodology are that it is very fast and the least expensive. However, this method is the riskiest as well. If the new system has an operational problem or if the users are not properly prepared, it could prove disastrous for the organization.
- Pilot implementation. In this methodology, a subset of the organization (called a pilot group) starts using the new system before the rest of the organization. This has a smaller impact on the company and allows the support team to focus on a smaller group of individuals.
- Parallel operation. With parallel operation, the old and new systems are used simultaneously for a limited period of time. This method is the least risky because the old system is still being used

while the new system is essentially being tested. However, this is by far the most expensive methodology since work is duplicated and support is needed for both systems in full.

- Phased implementation. In phased implementation, different functions of the new application are used as functions from the old system are turned off. This approach allows an organization to slowly move from one system to another.

Which of these implementation methodologies to use depends on the complexity and importance of the old and new systems.

Change Management

As new systems are brought online and old systems are phased out, it becomes important to manage the way change is implemented in the organization. Change should never be introduced in a vacuum. The organization should be sure to communicate proposed changes before they happen and plan to minimize the impact of the change that will occur after implementation. Change management is a critical component of IT oversight.

Maintenance

Once a new system has been introduced, it enters the maintenance phase. In this phase, the system is in production and is being used by the organization. While the system is no longer actively being developed, changes need to be made when bugs are found or new features are requested. During the maintenance phase, IT management must ensure that the system continues to stay aligned with business priorities and continues to run well.

Summary

Software development is about so much more than programming. Developing new software applications requires several steps, from the formal SDLC process to more informal processes such as agile programming or lean methodologies. Programming languages have evolved from very low-level machine-specific languages to higher-level languages that allow a programmer to write software for a wide variety of machines. Most programmers work with software development tools that provide them with integrated components to make the software development process more efficient. For some organizations, building their own software applications does not make the most sense; instead, they choose to purchase software built by a third party to save development costs and speed implementation. In end-user computing, software development happens outside the information technology department. When implementing new software applications, there are several different types of implementation methodologies that must be considered.

Study Questions

1. What are the steps in the SDLC methodology?
2. What is RAD software development?

3. What makes the lean methodology unique?
4. What are three differences between second-generation and third-generation languages?
5. Why would an organization consider building its own software application if it is cheaper to buy one?
6. What is responsive design?
7. What is the relationship between HTML and CSS in website design?
8. What is the difference between the pilot implementation methodology and the parallel implementation methodology?
9. What is change management?
10. What are the four different implementation methodologies?

Exercises

1. Which software-development methodology would be best if an organization needed to develop a software tool for a small group of users in the marketing department? Why? Which implementation methodology should they use? Why?
2. Doing your own research, find three programming languages and categorize them in these areas: generation, compiled vs. interpreted, procedural vs. object-oriented.
3. Some argue that HTML is not a programming language. Doing your own research, find three arguments for why it is not a programming language and three arguments for why it is.
4. Read more about responsive design using the link given in the text. Provide the links to three websites that use responsive design and explain how they demonstrate responsive-design behavior.