

1.6: Development

Below are just a few examples of different methods of programming found in the workplace. This is not an exhaustive list, but is meant to induce some personal analysis of what approaches might be effective for you. We will look at examples of styles based on the number of programmers—one, two, or many—to demonstrate how programming can be approached as team sizes change.

Staffing Styles

Single Developer

Also called The Lone Cowboy or Lone Wolf. Typically found in small companies, benefits of being a single developer are that you are intimately familiar with the entire code base (at least while you are writing it—we will get to that under [Good Practices](#)), do not have to agree on coding styles, branching, or integrating code, and your functions and classes are built to what you specify.

Downfalls however are also large. A second pair of eyes or fresh mind can help find bugs faster, the workload burden is entirely yours, and any security issues or bugs you are not aware of are likely to be found the hard way, by an end user or malicious visitor.

Paired Programming

Sometimes referred to as Holmes and Watson programming, paired programming is the practice of assigning two programmers to the same task, and having them (quite literally) work side by side. This allows one person to write while the other contemplates code interactions, watches for bugs, and keeps track of tasks. By altering which programmer in the pair is the lead at different intervals both programmers are able to contribute and learn from each other. Proponents of this model will highlight studies that show decreases in bugs, increased performance (it is, after all, harder to sneak in that Facebook post when co-workers are regularly using your screen), increased knowledge across staff members, and less distraction.

This approach may or may not work well depending on the culture present, and paying two staff members to complete one task can be more expensive, possibly offsetting reduced programming time. Poor pairing decisions (e.g., two programmers with little experience) and other factors like addressing sick time and vacations that interrupt teams can also reduce the potential benefits of this approach.

Team Development

Team development allows the work involved in a project to be dispersed among a group of individuals (a necessary step for most large scale projects) and reduces overall development time. A team with well-defined divisions of labor who adhere to an agreed upon set of methods can be a highly effective group.

Detractors are found where agreements on labor or method are ill defined or not adhered to. They can also arise from personal issues or conflicts of personality, and physically dispersed teams may find issues with time zone differences, limitations of communication methods, and increased “lag” time caused by not being face-to-face for immediate communication.

Project Management Methods

Once you move beyond programming alone and into groups, or have multiple parties working on the same project, a management approach will be needed to determine pace, goals, deadlines, and to maintain order and understanding of the project. There are a great deal of approaches to this problem, and we will take a quick look at some of the currently popular solutions.

Agile

While some of the principles of agile development go against the planning process we examined above, it can be effective in instances where fast turnaround is necessary and a highly iterative release process is acceptable. Some of its tenets are frequent communication between parties, self-organized groups motivated for the project, and requirements that are changed as ideas progress through the project. Changing requirements are driven by what is revealed through the iterative releases, and this fluidity is one of the strengths to be found in this approach.

Ultimately, the guiding principle here is to work in the mindset where you get the best people, trust in them, and focus more on the customer’s wishes and the project itself than length contracts, internal processes, and bureaucracy. You are most likely to run into this approach in start-ups that are not burned by an internal bureaucracy and heavily structured atmosphere.

The twelve principles of agile development, according to a [published manifesto22](#) by seventeen software developers are as follows:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Scrum

The Scrum process is a more focused and organized approach than agile development. The Scrum process maintains the defined roles of team members while pursuing the fast-paced development goals of agile. Daily meetings are held at the same time/place, and are typically held standing, to encourage shorter meetings. Core team members lead the 15-minute meeting by briefly reporting on what they did since the last meeting, their plans for the day, and any obstacles they have come across. Stumbling blocks brought up in these reports are addressed by the Scrum Masters, who address obstacles in order to keep the rest of the team on task.

Scrum goals are organized into Sprints, blocks of time usually shorter than 30 days, in which certain tasks of the project should be completed. Sprints are kicked off with planning meetings, and the goal is to have those portions of the project in full working order by the Sprint's completion.

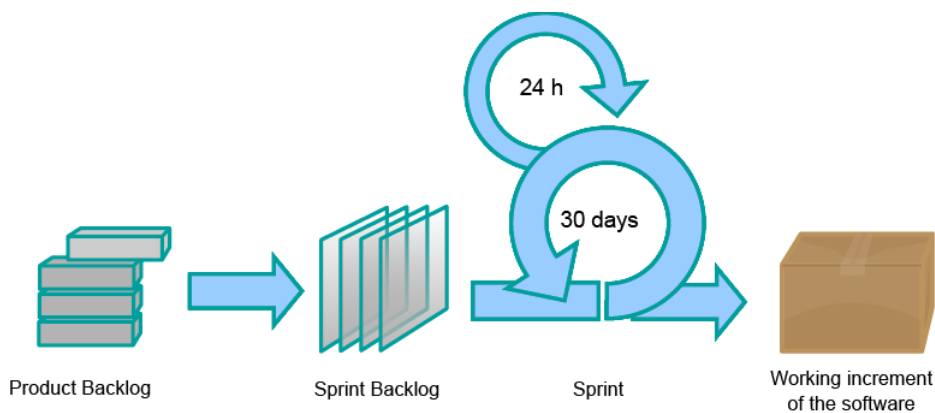


Figure 1.6.1 Scrum. (By Lakeworks

[CC-BY-SA-3.0-2.5-2.0-1.0], via Wikimedia Commons)

Waterfall

The Waterfall approach to project management recognizes the projects are typically cyclical, and builds that recognition into its five stage approach to management:

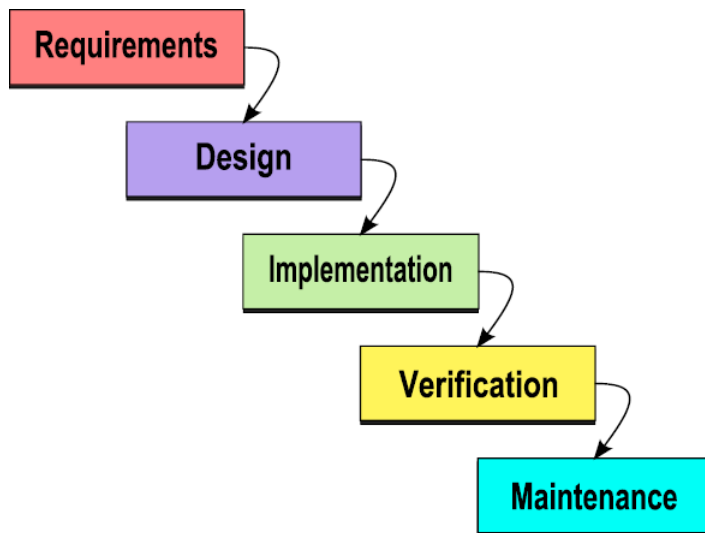


Figure 1.6.2 Waterfall Diagram. (Paulsmith99 at

[en.Wikipedia CC-BY-3.0](#), via [Wikimedia Commons](#))

Requirements

Requirement gathering can be interactions with your users that generate feature requests, initial project definition like its scope, or discovery of other important information during the planning phase of a project.

Design

Development of pseudo-code, storyboards, wireframes, or any other materials necessary to depict what will be implemented to satisfy the requirements. See sections: [Pseudo-Code First!](#), [Storyboarding](#), and [Wireframes](#).

Implementation

The actual programming and development phase to generate the working solution.

Verification

Testing and debugging the solution to ensure it meets the requirements and works as intended.

Maintenance

Ongoing tasks such as database maintenance, modifications to support changing requirements, or new tasks created by the discovery of bugs.

Items discovered in this phase trigger the process to begin again in order to address them. Eliminating a bug, for example, requires determining how to fix the problem, after which the bug resolution will move through the rest of the steps just like the project did initially. New ideas or feature requests will also trigger the requirements stage to begin. As these events will not occur at the same time, the waterfall effect occurs as individual items in the project will be in different phases as the cycle continues.

Structural Patterns

More decisions! Now that we have our team(s), and an approach for managing the project as a whole, we need to determine how to manage the development of the code as well. How the code is organized contributes to how easy or difficult change management becomes, how flexible the system is, and how portable it is. The options of how to approach organizing the actual code are called Architectural Patterns.

Model View Controller

In the MVC approach, we create three distinct concepts in our code. The image below depicts a de-coupled approach to MVC, where the model and view have no direct communication.

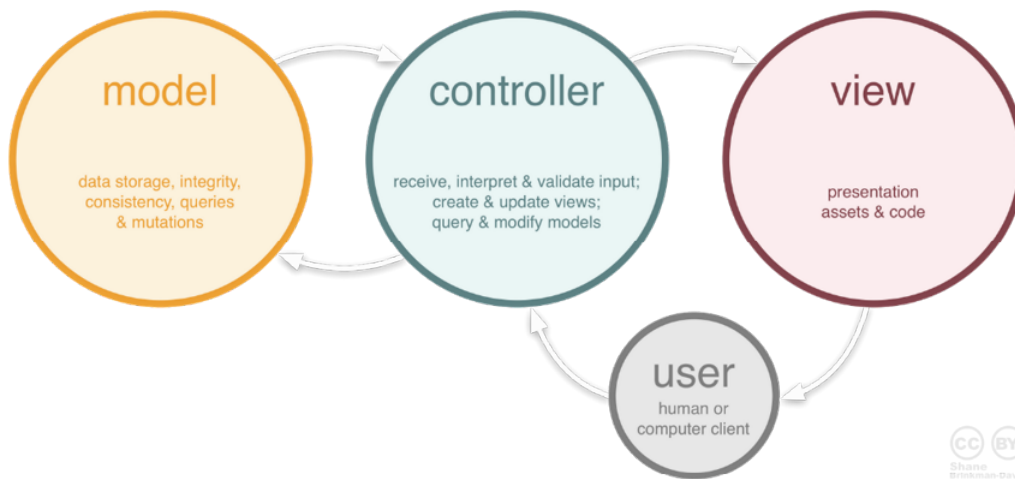


Figure 1.6.3 Model View

Controller. (Shane Brinkman-Davis, December 2012 CC-A 3.0)

Models

Our models contain the logic, functions, and rules that manipulate our data. A model is the only piece of the system that should interact directly with your data source. It should also respond to a request with a consistently formatted reply, and not simply return messages from the data source. This allows you to utilize multiple different sources for your data, or easily change how it is stored, as you would only need to edit your model's interactions and connections instead of adjusting code throughout your entire site.

Views

Views generate the output that is presented to the user. They request, or are given, the data needed to complete the page by the model. In some interpretations of MVC, the controller may also act as a mediator between the model and view—the important fact is that you should not see code in a view section that completes any actions other than formatting and presenting output. The view contains the images, tables, styling, and page formatting that make up the site itself. By keeping these items isolated from the models and controllers, we can easily duplicate our site's appearance somewhere else, or “skin” several sites differently, but have them all use the same data and models for interaction.

Controllers

Controllers recognize changes and events, such as user interaction and results of model responses that drive other actions. The controller will then call the appropriate model(s) to interact with data, and the appropriate view(s) to reflect the changes made. When controllers also manage passing data from the model to the view, the system is considered passive, or de-coupled, as the models and views have no awareness of each other.

Variations

MVC was originally developed as a method for traditional software development. Migrating it to web development is not a simple task, as the concepts become muddled when interacting with multiple languages, and a client-server model of communication. Other approaches to interpreting the MVC method for web development such as model/view/presenter, model/view/adaptor, and presentation/abstraction/control attempt to resolve and clarify implementing this approach online.

Service-Oriented Architecture

SOA is a modular style approach to interaction. It is used in web services and [Cloud Computing](#) (APIs) as a method of providing tools, actions, and information to other systems that allows for larger and/or multiple systems to integrate into. Each item in an SOA approach is oblivious to the actions of other services. It only knows how to request information or actions contained in other services as they are needed.

By building in this approach, the programmers can specify and limit what actions are available and under what conditions (end user's authentication level, nature of data, etc.) all while keeping the actual systems removed from direct interaction with the consumer. This helps protect internal systems by limiting access to raw data and intellectual property while still providing an organized platform from which developers can retrieve what they need.

Larger companies might develop an API that provides information from multiple data sources from one place. By creating the common platform, their developers can connect to the data from other applications or websites without having to connect to each

data source. These companies might also allow their partners to access their API in order to provide automate communication between companies, create other add-on tools, or to contribute their own information. Walgreens recently opened a portion of their internal API for just this reason. By allowing third parties access, outside vendors were able to create applications like printing from Instagram right at your local store, or refilling prescriptions from a mobile device.

Unit Testing

Unit testing is an exercise in writing each element of your software to meet very specific requirements. By reducing your project to its smallest testable component, each part can be tested individually. These modules are then connected together to form the larger whole. It is included under methods instead of practices as it is an important approach that needs to be followed by each team member if it is to be used at all. Testing these units before including them in the software should reduce debugging, and more readily allows for testing components before a full implementation has been assembled. This technique can also be combined with other methods because it is more of a coding practice than an approach to project implementation.

Unit testing is frequently used in fast paced approaches like Scrum to help ensure a less error prone product. Keep in mind that this is not a perfect solution. While individual units may test fine, logical problems can still be created as these modules are combined. Continued testing of the results of not only single units but also units working in combination will help in addressing this potential point of failure.

Learn more

Keywords, search terms: Project management, programming architecture, software planning and development

Architectural Blueprints—The “4+1” View Model of Software Architecture
<http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>

Project Management Advisor <http://www.pma.doit.wisc.edu/index.html>

Good Practices

File Organization

As we begin to create more and more files to complete our website, keeping everything in one folder will quickly grow cluttered. To address this, we can create folders just like we do when sorting files in My Documents. Traditionally you can find folders for images, scripts, pages or files, or for different sections of content or tasks, like an admin folder or ecommerce store. How and why these folders are created varies to personal taste or group determinations, and in some cases is done to maintain a particular method of writing code such as model-view-controller.

Pseudo-Code First!

Whiteboards, notepads, and napkins are your friends. Writing out how you plan to tackle a particular problem will help you identify logic problems before you are halfway through coding them, and will help you keep track of what you need to work on as you progress. Creating pseudo-code is the process of writing out in loosely structured sentences what needs to be done. For example, if my task is to look at each element in an array and tell the user if it is true or false, we might draft the following:

```
1. foreach(thing in array){  
2. if(thing / 2 is 0) then show Even  
3. else show Odd  
4. }
```

Imagine that while writing this example out we realize that we want to store the responses for use again later, not just show them to the user. So, let us update our pseudo-code to take that use case into mind:

```
1. foreach(thing in array){  
2. if(thing / 2 is 0) then add to even array  
3. else add to odd array  
4. send arrays back in an array  
5. }
```

Reviewing what we have now, not much looks different, but so far we have not had to rewrite any code either. After some thought, it might occur to us that creating two additional arrays could be more memory intensive and less efficient than editing the one we

already have. So, to simplify things and possibly improve performance, we might try this:

1. `foreach(thing in array){`
2. `if(thing / 2 is 0) then add it to even array, delete from this array`
3. `send arrays back in an array`
4. `}`

Finally, since we are now editing our existing array, we need to make sure we reference it (ensuring our changes are reflected after the `foreach` completes), which also means we only have to pass back our even array:

1. `foreach(reference! Thing in array){`
2. `if(thing / 2 is 0) then add it to even array, delete from this array`
3. `send even array back`
4. `}`

While none of the above examples would work (they are just pseudo-code), we were able to edit a conceptual version of our program four times. Alternatively we would have spent time creating and revising actual lines of code three times, only to keep finding an issue, backing up, and make lots of changes.

Comments

To quote Eagleson's Law, "Any code you have not looked at for six or more months might as well have been written by someone else." This is not to say that your style or approach will change drastically over time, but that your immediate memory of what variables mean, why certain exceptions were made, or what the code, ultimately, was meant to address may not be as apparent as when you last worked on the file. It is natural for us to feel that we will remember these details as they are so obvious when we are creating them. The need for good commenting becomes immediately apparent when reviewing someone else's work, and you are wallowing in frustration trying to figure out what that person was trying to do.

This is not meant to endorse comments that are obvious to the reader from the line, like:

1. `$int = 2 + 2 //we added 2 and 2 together`

Rather, comments are best suited to explaining why agreed upon methods were not used or what difficult to understand code might be doing, like:

1. `// This block of code checks each text file in the folder for the given date and deletes it`
2. `// This is legacy code from when Frank M. worked here. If we change anything, payroll breaks`

Order of Assignment

Many scripting languages distinguish between assignment statements and logical tests based on the number of equal signs used in the statement. For example, giving the variable `$temp` the value of 10 (also called "setting" or "assigning" the variable) can be expressed as `$temp = 10`, while checking the value would look like `if ($temp == 10)`. It is very easy to forget the extra `=` when writing logic statements, since we are so conditioned to using it in singular form.

This creates issues as a logic test written as `if ($temp = 10)` will always be true. First, we are executing what is in the parenthesis—in this case, setting `temp` equal to 10. When this occurs, the system returns a result of "true" to the script—the request has been completed. This is like asking if true is true—it always is! Since this does not cause a problem that stops the program from running, we will not get any errors before running it. The errors will be discovered only when the program does not behave as intended, and depending on the nature of the logic statement we wanted, that might be a rare case, making for some frustrating debugging. These are called logical errors, as the compiler or engine can run the code we gave it, and the error lies between expected execution and what the system is actually programmed to do. We will look at this further in [Chapter 27](#).

To protect ourselves from this, we can invert our logic statements to put our values first. Since we cannot assign a variable to a number, writing `10 = $temp` would be considered invalid, as 10 does not represent a place in system memory. Neither variables nor constants can start with a number, even `$10` is invalid. However, `if (10 == $temp)` is still valid, as the system compares both sides of the equation to each other, and is indifferent to the order.

By placing values first in our logic statements, if we forget to use the correct number of `=`, we will get an error from the engine early on that we can immediately find and fix. Otherwise we are left with a logic error that needs to be traced through our program later when we discover it is not working correctly.

Read Me

Closely connected with the concept of good commenting is the ubiquitous readme file—familiar to many users as the basic text file no one ever reads despite its title’s quiet plea. The readme file’s continued prevalence in such an antiquated file format is done to ensure it is legible on the widest variety of operating systems, and is still considered the best delivery format.

Within your readme file should be notes targeting your end users. Unless you release open source code, they would not be able to read your comments. These notes should tell the user what the application’s requirements are, how to install it, how to get help, and if you like, what notable changes are in the version they are looking at compared to its predecessor(s). Typically, the most common location to put a readme file is in the root folder of your files, or in a sub folder in which the readme notes apply to.

Spacing

Just as we use spacing in documents to convey that a topic change is occurring, we can break up longer string of commands by putting spaces around lines that are grouped together to complete a particular task, signifying that the next set of lines is for another task.

Brackets

Some languages require the programmer to use a combination of parenthesis and brackets to identify what pieces of code belong together. This allows the engine or compiler to delineate between the code that should be tested as a logic statement, code that gets executed if that statement is true, and code that belongs to functions or classes.

As we write our code and reach instances where we need these elements, it is good practice to immediately enter both the starting and ending marker first, then create space between them to enter your code. This will help ensure that you do not forget to close brackets later, or close them in the wrong places when nesting code. Depending on your text editor, it may assist you by automatically adding closing brackets and helping you identify which opening and closing brackets go together.

Indentation

To make your code easier to read, you can use indentation to give the reader an idea of what lines of code belong to different sections. This is typically done inside functions, classes, and control structures. When we nest code, extra indentations are added for each layer within, moving those blocks of text further right to visually distinguish. As we finish the contents of a loop or function, our closing bracket is lined up with our function definition or logic statement to show that the section of code belonging to it is complete.

While our program will run just fine without indentation, it makes it easier to see where you are in your program and where the line you are looking at is intended to be in the logic flow.

```
<html>
<?php
[spacing
function(){
  [ ] echo "Hello World"; // Outputs to screen
}      indentation      commenting
?>
</html>
tags
```

Figure 1.6.4 Code Formatting Examples

Figure 18 Code Formatting Examples

Meaningful Variable Names

When you create variables and functions, try to create names that will have meaning not only to you, but to others who may read your code. While it can be tempting to use a lot of short variable names while writing your code, it will be more difficult later to follow what the variable is supposed to represent. You might decide to use short names like `queryResult` or `query_result` or something longer like `numberOfResumesReceived`. While the latter takes longer to type while coding, the name is very clear on what it represents. As spaces are generally prohibited in variable names, these examples show us a few ways to approach longer names. The method you use is up to you, but should be used consistently throughout your code to reduce confusion. Differences in

how and where you use capitalization or underscores can be used to represent different types of variables like classes or groups of variables.

Short variable names like a simple `x` or generic name like `temp` may have their places in your code, but are best reserved for when they identify a small variable or one which will have a very short shelf life in your code.

Versioning

This is the process of creating multiple versions of your software, instead of continuously overwriting your sole edition of code. By creating different copies of your program as you create new features, you can preserve working copies or even create different versions of your program. This allows you to “roll back” or restore previous versions if unforeseen errors are created in new code, or to allow different features to be tried and discarded over time. Naming conventions for different versions of your code might involve numbers, letters, release stages (i.e. alpha, beta, release candidate, and release) a combination of all of these, or just “development” and “live.”

GitHub

A popular tool for collaborating on projects right now is [GitHub](#).²³ Focusing on open source projects, GitHub is a cloud service with locally installed application options that focuses on branching. Branching facilitates multiple working versions with varying features to co-exist in the same project space, giving the developers the ability to selectively merge new code into their project’s official repository. The platform supports code sharing, generating branches, and includes discussion boards, bugs, and feature request areas.

Development Tools

The following tools can be very useful in accelerating project development by reducing repetitive tasks and providing collections of tools to help you write your code. I would encourage you to refrain from using them until you have at least mastered the material in this text, otherwise, you may complicate your debugging tasks or not fully understand what those tools are doing.

Frameworks

Frameworks are compilations of code that you can use to quickly start a site with a collection of features already in place. In a home building analogy, it would be akin to ordering parts of your house already completed, and having special tools in your toolbox for putting the pre-built pieces together.

A typical framework is a set of files that come with their own instructions, and can be so extensive that they take on a life and syntax beyond the language they are written in. They extend the features normally found in the language they are written in by adding their own classes, functions, and capabilities. The goal is that by giving the framework a few commands, we can create much larger processes like a menu system or complete color scheme. Some frameworks focus on the automation of repetitive tasks like generating forms and pages based on tables in a database, or applying in depth style and structure across a website.

Multiple frameworks can be combined in a single project in order to add a combination of features, for example using one framework for the site layout and another for generating database interactions. Each framework will require some time learning how to use its features, just like learning a programming language. This may be an important factor when deciding when and how many to use in your work.

Smarty²⁴

The Smarty template engine targets the separation of application logic and presentation. While it creates delineations between the code necessary to generate the content and the code to present the content, it is not a full model-view-controller design. A smarty template page supports special tags and commands that are part of the smarty engine. These elements help to generate what the end product will look like, after generating the PHP necessary to build the page. Smarty also uses a template caching approach to facilitate delivering pages faster, only updating cached templates when changes to a smarty file or its dependencies are detected.

Yii Framework²⁵

The Yii framework focuses on reducing SQL statement writing, follows [Model View Controller](#) methods, and helps create themes, web services, and integration with other platforms. It also includes security, authentication, and unit based approaches to development.

Zend Framework²⁶

The Zend framework is focused heavily on modularity, performance, and maintaining an extensible approach to allow continued integration. This framework is popular at the enterprise level, and includes some of the original creators of PHP on staff. Zend itself is a full service PHP company, providing training, development services, and the continued refinement of the PHP language itself.

Templates

Similar to the idea behind frameworks, templates are sets of files that dictate the basic structure that provides a layout to your site. Templates typically create a grid format you can select from, like two or three columns, fixed or relative width and height, etc. If you are starting a site fresh and putting it into an empty template, there may be some placeholder content and styling as well. Templates are useful for getting the look and feel of a site up and running fast and there is little concern about the particulars of appearance, or whenever the template meets your needs well. When inserting your content dynamically, multiple templates can be used for one site to change the look and feel quickly based on which one is applied. This might be determined by what type of device your guest is using, or what type of authentication they are using.

Templates can be both freestanding, or can be an extension of a content management system or framework.

Bootstrap27

Bootstrap was created by Twitter in order to help them manage their extremely popular service. As it matured, they made it open source to allow others to utilize the toolset. Their framework provides tools for styling, interaction elements like forms and buttons, and navigation elements including drop downs, alert boxes, and more. Using this framework involves little more than linking to the appropriate JavaScript and CSS files and then referencing the appropriate style classes in your code.

Foundation28

The Foundation system focuses on front end design and follows the principles of responsive web design (see next section). Their approach uses a grid layout to allow flexibility, accelerate prototyping and wire framing, and provides integrated support for multi-platform designs.

Responsive Web Design

As often as possible, this text will focus on coding methods that support responsive web design. This approach replaces the practice of developing several version of your site in order to support different devices. An example of this is when you come across sites with a regular site, a mobile site (for example `m.yoursite.com`) and then provide an app for each of the tablet platforms, or force tablet users into their mobile or desktop experience.

Instead, we will create one set of files that changes its own layout and appearance based on what we know. By using information made available in the initial http request to our site, we can determine the features the user's browser supports, width and height of their screen, and more. We can use this information to instruct our CSS files on what rules to apply when styling the page, how much or little to resize elements on our page, what we want to eliminate or hide on smaller screens, and more.

While this approach is still not a perfect solution, it gives us a much-improved ability to support a wide variety of devices without managing several code bases and developing across multiple proprietary platforms.

Integrated Development Environments

This list is by no means comprehensive. These editors are sufficient to get you started. If you wish to continue in web programming, you may elect to invest in a development platform like Adobe Dreamweaver or another professional product that supports more advanced design, or try any number of other IDEs available that focus on a variety of different languages.

You might consider the following programs to help you write your code (listed in no particular order). Each of these has features particular for web development and should be sufficiently capable to get you through the examples in this text.

Jedit29

A free editor based around Java. Works on multiple platforms (Windows, Mac and Linux) and includes syntax highlighting.

Notepad++30

Notepad++ is a source code text editor with syntax highlighting, multiple document handling using tabs, auto-completion of keywords (customizable), regular expressions in the search and replace function, macro recording and playback, brace and indent highlighting, collapsing and expanding of sections of code, and more.

Bluefish31

Supports many programming and markup languages. An open source development project, multi-platform, and runs on Linux, FreeBSD, MacOS-X, Windows, OpenBSD and Solaris.

TextWrangler32

This editor is related to BBEdit. It does not include as many tools, but retains syntax highlighting and the ability to use FTP within the editor.

HTML-Kit33

This editor is intended for use by web developers, and comes with support for writing HTML, XML and scripts. Among its features are internal preview of your web, integration with HTML Tidy, auto-completion of keywords, etc. Look for the “Previous” version for their free copy.

Application Programming Interfaces

Commonly referred to as APIs, pronounced as the letters of the acronym, application programming interfaces allow us to interact with features and data of a system other than its primary means, whether it is an application or website. Created to address needs of data exchange and integration between systems, APIs provide a controlled method of allowing others to use a system without having direct, unfettered access to the code or database it runs on. Examples of APIs in action are maps on non-google website that are fed from Google Maps with markers, that highlight paths and routes, automate directions, or outline places of interest. All of this is done from within their site or system without you leaving to interact with Google. Another example is the growing popularity of sites for clans or groups of friends in multi-player games that provide results, show game statistics, screen shots, and rankings from a site they create by using the game developer’s API to access their data.

Web-based APIs are, essentially, limited websites. They allow the pages and scripts end users create to communicate with the data source by using a predetermined vocabulary and fixed amount of options. When the user’s message reaches the API, the API completes the requested task such as getting a certain piece of data, or validating credentials, and returns the results, hiding anything the developers do not want revealed, and only provides the features they are comfortable with others using.

The result is that end users are free (within the limits of the API) to create their own systems exactly as they want, interfacing with their own systems, or creating all new systems the developers of the API had not thought of or decided not to pursue. APIs can cut down on the development time of your own system as you can use them to support your project, like our example of embedding Google Maps instead of creating or installing a map system. By combining several disparate systems through their APIs, a mashup is created, which is a new feature, application, or service created by combining several others. APIs are often included as part of a software development kit (SDK) that includes the API, programming tools, and documentation to assist developers in creating applications.

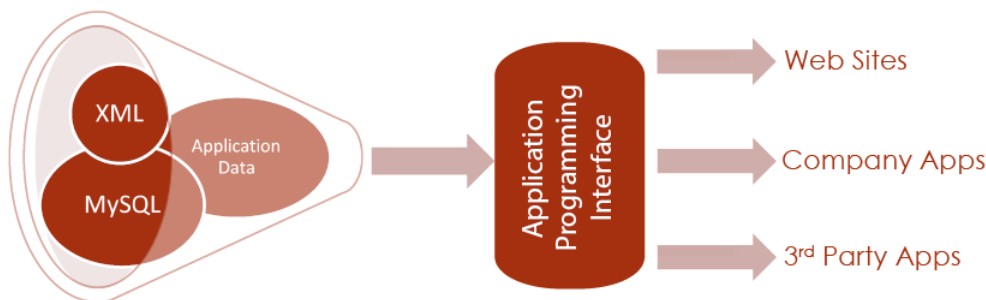


Figure 1.6.5 API Data Flow

Learn more

Keywords, search terms: Development methods, coding practices, team development

Cisco Networking Example: http://docwiki.cisco.com/wiki/Internetworking_Basics

List and description of all top level domains: <http://www.icann.org/en/resources/registries/tlds>

Ongoing comparison of hosting providers: <http://www.findmyhosting.com/>

Software Development Philosophies: http://en.Wikipedia.org/wiki/List_of_software_development_philosophies

1 <http://calculator.s3.amazonaws.com/index.html>

- 2 <http://www.dropbox.com>
- 3 <http://drive.google.com>
- 4 <http://aws.amazon.com>
- 5 <https://www.google.com/search?q=nort...net+censorship>
- 6 <http://thomas.loc.gov/cgi-bin/bdquery/z?d112:h.r.3261>:
- 7 <http://www.opencongress.org/bill/112-s968/show>
- 8 <http://www.govtrack.us/congress/bills/111/s3804/text>
- 9 <http://www.ustr.gov/acta>
- 10 <http://www.riaa.com>
- 11 <http://www.forbes.com/sites/andygree...e-wheel-video/>
- 12 <http://www.raspberrypi.org>
- 13 <http://www.mongodb.org/>
- 14 <http://www.google.com/fonts/>
- 15 <http://asg.web.cmu.edu/rfc/rfc3986.php>
- 16 <http://www.w3.org/TR/REC-xml/>
- 17 <https://www.google.com/webmasters/tool>
- 18 <http://www.bing.com/toolbox/submit-site-url>
- 19 <https://www.google.com/search?q=website+visual+design>
- 20 <http://www.nngroup.com/articles/how-...-on-web-pages/>
- 21 <http://techland.time.com/2013/05/06/...-websites-2013>
- 22 <http://www.agilemanifesto.org/>
- 23 <http://www.github.com>
- 24 <http://www.smarty.net>
- 25 <http://www.yiiframework.com>
- 26 <http://framework.zend.com/>
- 27 <http://twitter.github.io/bootstrap/>
- 28 <http://foundation.zurb.com/>
- 29 <http://jedit.org>
- 30 <http://notepad-plus.sourceforge.net/>
- 31 <http://bluefish.openoffice.nl/>
- 32 <http://www.barebones.com/products/textwrangler>
- 33 <http://www.chami.com/html-kit/>

1.6: Development is shared under a [not declared](#) license and was authored, remixed, and/or curated by LibreTexts.

- **1.6: Development** by [Michael Mendez](#) has no license indicated.