Separation of Concerns

Introduction

Separation of concerns is a fundamental philosophy in software development that offers an elegant solution to managing the complexity of large, complex applications. At its core, separation of concerns dictates that every layer of an application should be a self-contained unit, written to perform a single, well-defined task, with no code that ties it to other tasks or concerns. The organization of the code should establish clear boundaries based on purpose, making the code easier to read and maintain. Each component, whether a method in Java or a file in web development, should be laser-focused to enhance the clarity and scalability of the application.

For example, consider a Java application where every class lends organization, and every method has a specific, clear job. In web development, a good start is writing each file to concentrate on a specific job and not mixing your languages in a single file. Separation of concerns can go much further in web development, but "one file, one task, one language" is a powerful starting point. Note that I said a powerful “starting point.” I know this can be a hot-button topic, so I’m approaching it from the point of benefiting junior developers and teams.

Separation of Concerns in Web Development

Separation of concerns in web development begins with an understanding of the results you get from mixing your languages, be it a markup language or a programming language. Fundamentally, HTML, CSS, and JavaScript should be segregated into separate files. Inline and embedded CSS are direct violations of separation of concerns, as is inline JavaScript.

Example of Inline JavaScript

```
<!DOCTYPE html>
<html>
<head>
  <title>Inline JavaScript</title>
</head>
<body>
  <script>
    alert('Hello, world!');
  </script>
</body>
</html>
```

Example of Embedded CSS

```
<!DOCTYPE html>
<html>
<head>
  <title>Embedded CSS</title>
  <style>
    body {
      font-family: sans-serif;
      background-color: black;
    }
    p {
      font-size: 16px;
      color: white;
    }
  </style>
</head>
<body>
  <h1>This is a heading</h1>
  <p>This is a paragraph of text.</p>
</body>
</html>
```

Even the straightforward example presented above can lead to future issues. If embedded CSS is interspersed throughout an HTML page, it makes the page difficult to read and lacks any cohesion. The inline JavaScript presents the additional problem of becoming a potential game of hide and seek as future developers try to figure out exactly where the script is in the code. Lack of separation can also tie your programs together, creating an interdependence where one file depends on another file for functionality. In the following lines of code, a game page contains a button that, when clicked, simulates rolling a 6-sided dice. The HTML code not only creates the button, but uses a JavaScript event handler, “onclick” to run the program.

```
<input type="number" id="d6Num" value="1" min="1">
        <button onclick="rollDice('d6')">Roll</button>
```

In this case, the program for making the dice simulation function is in a JavaScript file, but the "trigger" to roll the dice is the “onclick” in the HTML file. These two files are now inseparable unless both are rewritten. In the future, if a developer isn't paying attention to this and changes the JavaScript file, it could break the functionality of the dice roller. Instead, the two elements could be kept separate so that if someone wants to make a change to the JavaScript file, they will have an easier time finding the event listener that is in the same file. Additionally, if a new dice roller program is found that functions better than the old program, the old could be

swapped out for the new with little effort and no need to rewrite the HTML file. The developer would only need to ensure that the JavaScript event listener was in the new file.

Benefits of Separation of Concerns

Consider a moderately sized business website, one that might have twenty pages where each page has been created as its own file, and each is sprinkled with both styling and JavaScript throughout the pages. Add in PHP files that render HTML through the echo statement. A new developer coming in to take over the webpage would spend days just detangling the website, understanding the DOM flow, and trying to put the pieces in place. If the business has grown, and the new developer was hired to expand the webpage as a result, the results will more than likely mean more time and money spent fixing the website before it can be expanded. On the other hand, if that same webpage was templated, had one folder for HTML templates, a folder for CSS files, a folder for all JavaScript files, all PHP files in one place, and each of those files respected proper separation of concerns, the web developer could sit down and have a handle on the website and be making plans for expansion before lunch.

The internet is full of differing opinions on exactly what separation of concern looks like. Some arguments make legitimate points directed at the logic of separating files by language. One strong argument deals with concerns that span multiple languages within an application, laying out illustrations of how concerns rarely fit in one neat language package. This side argues that the best way to separate a concern that spans multiple languages would be to put the entire concern in one file, regardless of language. One such argument goes further to point out that the human brain likes simplicity and that jumping from file to file just to work on one concern that is addressed by multiple arguments can overcomplicate the creation and maintenance of the application.

What this argument fails to grasp is the level of expertise that is going to be needed to see the boundaries of all concerns and how they will affect your application. Many junior web developers will have no idea what MVC means or what the N-tier approach means, much less have the ability to immediately separate an application into each of these tiers. Web development students should learn one or more of these concepts as they progress, but even a junior developer might not have a firm grasp of the true boundaries of their application. Many students will even struggle with the separation of languages, much less the more complicated boundary, but beginning your separation of concerns with the division of languages presents a concrete position to teaching the concept of separation of concerns. There are many other factors to consider in the debate, but for the purpose of this paper, complexity alone will invalidate many of the arguments against the "one file, one task, one language" approach.

Planning for Separation of Concerns

While this view entails keeping your HTML, CSS, JavaScript, and PHP completely separate, an application with one big JavaScript file that does everything and one big PHP file that does everything will most likely be just as guilty of breaking separation. Writing with a separation of concerns also means writing in small, self-contained modules. This means that separation of concerns is not just about how you write code. It must begin the moment you start to plan your application. When tackling the planning of an application, the first stage is always to figure out what you are trying to accomplish with a program. Once you have your overall goal, you need to begin to organize the steps that the program will run through to accomplish the task you set. Each of these steps should be broken down into coherent chunks, with each one performing a single, well-defined task. Not only will this approach make a division of concerns easier, but it will also make building your application easier.

Small, well-defined modules will also make it easier to maintain another important aspect of coding separation of concerns: cohesion. Cohesion is a measure of how well elements inside a module belong together. If your modules are well-defined, then bringing them together, for example, so that all methods in a class have common functionality, becomes easier than grouping large, complex methods. A program that is written with small modules that are grouped logically by functionality is said to have “high cohesion.”

High cohesion is directly related to another aspect of separation of concerns referred to as "loose coupling." We have already discussed how desirable it is to separate your HTML, CSS, and JavaScript, and this is one example of loose coupling, as your modules should depend on each other as little as possible. Coupling is defined as "The degree to which software components depend on each other." By maintaining high cohesion, we also make it easier to achieve a high degree of "loose coupling" where our small, unified parts work independently of each other to achieve the purpose of the website. Loose coupling also allows us to update and exchange various parts of the website over time as it grows and as the components themselves are updated. Computer science does not stand still, and the rapid changes in the tools that we use to design and program our webpages can be met with ease if individual parts do not depend on the existence of other specific parts, such as the example of the JavaScript dice roll simulator.

Advantages of Separation of Concerns

Effective management of the separation of concerns offers numerous advantages. As we've previously emphasized the importance of each division having a singular purpose, it's worth noting that such practices also facilitate code reuse. Consider a website with multiple pages that require a JavaScript file to validate user-entered email addresses. There's no need to duplicate this code multiple times; instead, the same file can be accessed and utilized by each instance where it's required. However, if this functionality is entwined within an HTML file, problems may arise when external JavaScript code attempts to interact with the inline JavaScript,

potentially leading to unintended outcomes. By keeping our code small and separated, we also facilitate reuse, making life easier in the future.

Looking to the future, most organizations aim for growth, meaning an expansion to their website. Separating concerns goes hand in hand with scalability. Scalability of a website is not only the process of adding more webpages, but also the capability of the webpage to handle expanded needs of an organization. A website coded with proper separation will be able to scale to the needs of the organization without the need for rewriting the entire website, or even large portions of it. In an ideal situation, nothing will be rewritten, but instead new modules will be placed beside existing modules in a "snap together" fashion and the website will roll forward with new functions and some small additions to the navigation. This would only be accomplished with small modules that have high cohesion and loose coupling.

Another advantage to having a high separation of concerns is how it can help a team of developers work on one application. If individual components of the application can be worked on by a different team, and then those components can be brought together seamlessly as each one is completed, there will be no need for one team to wait on the work of another and the necessity of meetings between the teams will be minimized, freeing time to work on the code.

Other advantages have been hinted at in previous paragraphs, but it must be recognized that separation of concerns inherently means that your code will be easier to read, easier to debug, and easier to maintain. Small modules are easier to comb through than long, extended modules full of multiple functions that span many concerns. The shortened modules will be less complex and, thus easier to understand. These facts alone make the code easier to read, and if your modules have one task, there should be less need to comment out sections or write additional sections just to test the module in isolation. It becomes more efficient to isolate the problems and efficiently address issues.

The Role of Templating in Separation of Concerns

When addressing efficiency in web development, there will come a time in any web developers learning path that they realize that writing multiple pages of HTML is not efficient when you must put the same header and same footer code in every page. CSS makes the effort slightly better in condensing styling for all those pages into one place, but how can efficiency be accounted for when the same code is copy/pasted for every page? Templating is the answer.

Templating is taking your HTML code, breaking it up into its natural parts, and using a templating engine to use that one part across your pages as needed. Just as discussed previously, small modules do one well-defined task. With HTML, the task is design efficiency. Instead of the same header copy/pasted to the top of twenty webpages, instead you have one header template that the template engine enables you to reuse at the top of every page for unity and cohesiveness to your pages. Reuse of code, ease of maintaining, all the advantages of separation of concerns applies to HTML as much as it does Java, JavaScript, PHP, or any

programming language. Using a templating engine means that HTML code is written once and can be changed once when updated. It is a major step forward in efficiency and a step forward in maintaining separation of concerns. When you can use template variables inside of template blocks, as with template engines such as Jinja, then it becomes even more efficient and a boon to the developer's work.

Returning to the issue of multiple teams working on an application, one major advantage to using templates in the design of a website will be seen when the design team can move forward with their job without waiting on or depending on the completion of features being worked on by the development teams. Design remains isolated within its own module, even in cases where it must interact with JavaScript or PHP. Without templating, the teams would have limited independence. Each employee would be handcuffed to the work of another, with meetings stealing valuable time each step of the way as developers and designers would need to know exactly where the other was and what was being done.

Conclusion

The principle of separation of concerns stands as a foundational pillar of software development, offering not only a systematic approach to managing complexity but also a multitude of practical advantages. This philosophy advocates for the isolation of distinct responsibilities within an application, emphasizing the creation of self-contained, well-defined modules that contribute to code clarity, maintainability, and scalability.

By adhering to the separation of concerns, developers can promote code reuse, streamline collaboration among teams, and efficiently accommodate future expansion needs. The discipline of maintaining small, coherent modules with high cohesion and loose coupling fosters code that is not only easier to read and debug but also more adaptable to the ever-evolving landscape of technology.

Whether it be the division of languages in web development or the compartmentalization of functionality in application design, separation of concerns emerges as a fundamental practice that empowers developers to create robust, maintainable, and scalable software solutions. As the software development landscape continues to evolve, embracing this principle remains essential for achieving code efficiency, simplicity, and longevity.