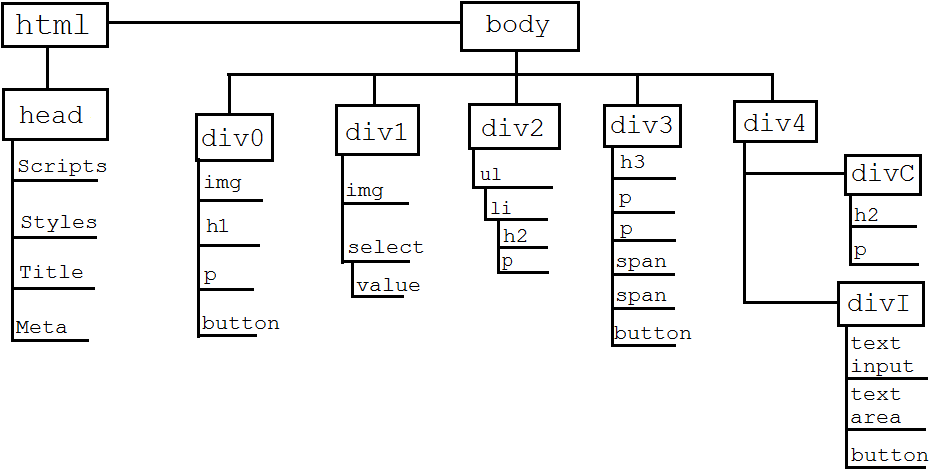
Team Plasma: Mitchell Caisse, Zac Chupka, Trevor Behlman, Dale LaPlante

Module FIN Research Insight Brief

For our final project, we implemented a bug-management system/webpage. This page allows users to create, view, edit, resolve, or otherwise manage bugs as needed. The problem we were solving was the relatively hectic way that bugs are currently handled. This often leads to shoddy software, as bugs are easily overlooked, forgotten about, misunderstood, or handled incorrectly. Because of this, we came up with characteristics that we thought would address the nature of these problems. These features include an organized, central location for all bugs, categorization of bugs to bring attention to those that need it most, and user input/collaboration so that all information is available to everybody at all times. These characteristics, we believe, would help developers (as well as users) of software to quickly submit, review, and address bugs as needed in as efficient and productive a way as possible.

Further planning and design layouts had to be created before implementing the bug management system. Extensive research went into the interaction between the client and the server to prevent slow development. Theoretically, by writing out exactly what views and calls to the server needed to be incorporated into the product, development would progress much faster with fewer issues. Below is the DOM design for the application. This was also created prior to the development phase of the project.

It was also important that different types of users were taken into account. Different personas were created while keeping the scope of the application in mind in order to address all skill levels. This also allowed us to design the application so it is intuitive to users with a lower skill level, but not so simple that experienced users find it unprofessional. As stated previously, personas were created based on the scope of this project. The final product is intended for companies that maintain software and need a bug management system to record their application issues, so it was unnecessary to create a persona for someone such as a child who would not use TerByte.

The personas include, a developer, manager, a product owner, a customer, and a user. Jason is a developer who contributes code to the application.  He uses TerByte to review, discuss, and edit the bugs assigned to him.  He may also view and comment on other bugs if he has any valuable information that might contribute to a fix. Megan is a manager of a group of developers who contribute code to the application.  She often fulfills the duties of a developer in addition to her job as a manager.  She uses TerByte for the same reasons as Jason, with the addition of assigning bugs to the developers she manages where she sees fit. Rich is a Product Owner for the application.  He uses TerByte to keep track of his development teams’ progress on bugs so he can estimate how long it will be until he can discuss user stories with them.  He may also comment on bugs to clarify expected behavior. Customers rarely have access to bug reports, but some enterprise applications come in tiered packages, where higher tier customers have access and lower tier customers do not. Susan is a high tier customer for an application. She can use TerByte through the application that allows high tier customers to report bugs to identify any issues she may come across. Users are often different from customers in that they use software for their career that is purchased by someone representing their company, the customer.  Users often have access to bug reports, especially if the application is open source. Jackie is a user of the product that Susan's company produces. He is not tech savvy and will likely report numerous bugs to Susan's company that aren't actually bugs. All of these personas were considered while designing the layout and functionality of the application.

The layout of our webpage is designed to present the user with any/all needed data as quickly as possible. They don’t have to do much navigation (if any at all) to get to the bug(s) they need or submit a new bug(s). At the very top is the header, which contains the site’s name and logo. More importantly, it contains the button needed to create a new bug. This design mimics the flow of work; putting the start of a bug’s lifecycle in the system at the start of the page. The top left of the screen then contains a table displaying information about all bugs in the system. This is in graph form, allowing users to easily get an idea about the state of the system (meaning all of the bugs at once) immediately upon viewing the page. Below it is the actual listing of all bugs, displaying their titles and priority level. Again, this presents users with the information needed to assess the current state of the system. By seeing which high-priority bugs there are, the number of bugs, etcetera, the user will be able to work on/review the bugs they need to without having to navigate. The two largest sections are to the right of these; one for details on the currently-selected bug (or details for a newly-created bug) and right below it one for comments. These are self-explanatory, however it should be mentioned that the comment section is made as large as the details section so as to emphasize the importance of users’ inputs. This is because in today’s culture user’s comments are devalued. We believe, however, that in a system such as this user input could easily be as valuable as the bug details themselves. This could hold information such as user experiences with/without the bug, speculation on the cause or effects of the bug, etcetera. All of this information is gathered in a readily available section to be reviewed with the bug itself.

At this point in the assignment, we followed the waterfall development model where the design process progress flowed steadily downward. We handled the highest level of the application first, and then moved downward from there. We assessed the requirement specifications of the project and then moved into the design stage. Once we felt that the design was suitable for the application and fulfilled all requirements, we began the development stage. From here, we followed an agile-like model where we split up the work into tasks. Luckily, we all excelled at different aspects of web ware programming. One team member handled the server side code, two members handled some server side tasks and some user interface tasks, and one member worked the interface and styles of the application. We had two six hour sprints that tasks had to be completed in. We met as a team and were able to ask each other for assistance when the interaction of our tasks didn’t cooperate.

Our hybrid waterfall-agile method worked very well. The design and development of the bug management system did not take a considerable amount of time, and we did not come across any major issues when merging the tasks together. We were able to resolve any conflicts when merging our code to the GitHub repository. We all highly recommend using the methods we used during this project as proof is derived from the high quality product TerByte.

These solutions address the characteristics of organization, categorization, and user collaboration that we had set out to solve. With this system, we hope users find their software bugs to be easily manageable and intuitive to address. The value in this is obvious: better/less-buggy software, quicker/easier development and support, and knowledge sharing to improve user-developer relations and awareness.