Project Summary

One of the well-known design techniques is the scenario-based usability engineering (SBUE) technique, where it focuses on creating rich and detailed user-interaction scenarios as a main representation of the software model that focus on the users' goals, their problems, and their context to help in making the right decisions quickly and confidently. Another popular design technique is the software architecture-based (AB) technique, where it focuses on an understanding of the architectural mechanisms used to achieve the software functional, quality, and business requirements at a level of abstraction by providing a series of steps for designing the conceptual software architecture.

Through this project, we will be evaluating both scenario-based usability engineering and software architecture-based designs in terms of how these two designs affect each software quality attribute, using McCall's list of software quality attributes, where we will go through each of those attributes and try to analyze how both designs addressed or helped achieve that specific quality attribute.

The McCall's list of the quality attributes that will be used to perform the comparative analysis is as follows:

- Correctness: the software meets its requirements specification e.g., the accuracy of the distance measurement varies between 5-300 miles
- Reliability: the software performs its intended functions without failure e.g., downtime for a system will not be more than 30 minutes per year
- Efficiency: the amount of hardware or software resources needed to perform a function e.g., a system is not using more than 1 GB of RAM
- Integrity: the software can protect unauthorized users from accessing the software or its data e.g., non-admin users cannot access the air traffic system
- Usability: the software is easy for users to understand and use it functions e.g., air traffic controller can easily understand how to use the navigation system
- Maintainability: it does not take a lot of effort to detect or fix an error during maintenance phase e.g., detecting a software bug is not taking more than an hour
- Flexibility: the software can be modified and improved easily e.g., improving the algorithm to compute speed, times, and distances in an air traffic control system
- Testability: the software can easily be tested to verify that it meets the specified requirements e.g., testing communication between air traffic controllers
- Portability: the software can easily be transferred from one platform to another e.g., moving the air traffic control system from Linux to Windows
- Reusability: the software's code can easily be used in other applications e.g., air traffic detection code to be used in a military application

Comparison

Correctness

I believe that the SBUE technique greatly affects and improves the correctness attribute since it hugely relies on users' interaction with the system and uses their continuous feedback to verify the system specification. For the AB technique, I also think that the correctness attribute is as greatly affected as in SBUE because of how AB technique verifies that the functionality, quality scenarios, and constraints are achieved through the proposed architecture by generating different views and continuously exercising the architecture scenarios.

Although SBUE and AB affect the correctness attribute almost the same, I believe SBUE works the best with smaller projects having more user basis and interaction where you can continuously work with those users to get their feedback to verify the correctness of what you are building like a food delivery app. On the other hand, I believe AB works the best with large-scale projects having multiple different layers where each layer does not have many details in it and does not require that much of user interaction to verify correctness like building an air traffic control system.

Reliability

In contrast, I think both SBUE and AB techniques have no significant effect on reliability since they do not play any role in the algorithms running the software, so it has nothing to do with performing the intended functions without failure. However, the idea of representing systems as scenarios or modules might indirectly improves the reliability attribute by making it easier and faster to detect where a failed function is located and fix it to get the system up running again. As a result, I believe that any project could be a good fit to use both techniques without having to worry about reliability since it is not really affected by those techniques.

Efficiency

Similarly, I think both techniques have no significant effect on efficiency especially in the hardware/software resources aspect, since they focus more on the design aspects of the system instead of the actual logic/algorithm running behind the scenes. However, I think both techniques improve the time efficiency in the sense that it is faster to implement features (and get feedback in SBUE case) because of how well organized and structured these techniques are, that makes the implementation process easier and faster. Accordingly, I think that any project can be a good fit for both techniques in software/hardware efficiency aspect, but projects that have many user scenarios and continuous feedback will benefit from the time efficiency of SBUE. In addition, projects with tight deadlines will also benefit from the time efficiency of both approaches.

Integrity

Although I also think that both techniques have no significant/direct effect on integrity since they do not focus on the actual code implementation, I think the concept of having separate scenarios in SBUE and abstract layers in AB somehow enhances integrity by preventing unauthorized users who already hacked and accessed a specific part of the system from accessing other parts since all parts are independent of each other. Accordingly, I think both techniques fit with any project since they have no direct effect on integrity, unless that project's components are dependent on each other, which in that case neither of the techniques adds any integrity benefit to such project.

Usability

On the contrary, SBUE technique hugely enhances the usability attribute of the system because of how it focuses on users' goals, their problems, their interaction with the system, and their continuous feedback, which makes it easy for users to understand the system and use its functions. That is, the system is almost built by the users themselves to achieve their goals. Furthermore, AB technique also improves the system's usability because of how it focuses on the software functional, quality, and business requirements during the design process which leads to enhancing the user experience. However, I think SBUE has bigger impact on usability than AB because of how deeply users are involved in the process than in AB. Therefore, I think SBUE fits very well with projects that rely heavily on users' interaction and their feedback like a sales simulation software, while AB best fits with projects that rely more on the functionality and process flow than user's interaction/feedback like Amazon Web Services (AWS).

Maintainability

Furthermore, I believe both SBUE and AB techniques improves the maintainability of the system. For the SBUE technique, the way it represents the system as multiple separate scenarios separated from each other make their maintainability much easier because a fix or failure will only affect one scenario without affecting others with which the users interact. However, this might be an issue for cases where scenarios rely on each other, which will make it harder to maintain because then fixes will have to be applied to all affected and related scenarios instead of just one. For the AB technique, since it divides the system into multiple abstract layers/modules that are isolated from each other, it makes it easier to maintain the system because you will only have to worry about maintaining the faulted component without affecting other components. Accordingly, I believe both techniques will be best fit for projects where their components, i.e., scenarios or modules, are isolated from each other and require frequent maintenance like the air traffic control system.

Flexibility

Similarly, I think SBUE technique adds a lot of flexibility to the system because of how those user scenarios are separated from each other which allows modifying and improving a specific functionality or scenario without worrying about affecting other functionality or scenarios. In addition, since these stories are always coming from users and their feedback, it makes it easier to understand what needs to be changed or improved. Similarly, I think AB technique's isolated modules hugely support the flexibility attribute by allowing making changes to each module without affecting other modules or dependencies. Besides, the organization of the system as layers makes it easy to understand and know what changes need to be made to what layer, which makes the modification process even faster. As a result, I believe projects that continuously have enhancements and modifications going will be great fit to use both techniques e.g., projects running on agile like Facebook or Twitter.

Testability

On the same token, I believe both techniques improve the testability of the system. For the SBUE techniques, having a system structured as separate components, i.e., user scenarios, helps in determining the testing efforts and helps organize the testing workflow for the system which improves evaluating and assessing the functioning of the system. For AB technique, representing the system as abstract modules increases the testability of the system by isolating components from each other and hiding unnecessary details that each test case needs, which makes the testing process faster, more efficient, and more specific to the functionality needed to be tested. As a result, projects that have separate and isolated user stories or modules where their functionality are not dependent on each other, the testability will be greatly improved using these two approaches. On the other hand, these approaches might not be good fit for projects having functionalities that rely on each other since you will not only have to test one functionality, but you will also have to test its dependencies, which takes longer time and slows down the process.

Portability

On the contrary, I believe both SBUE and AB technique will be considered as drawbacks from the aspect of portability, because of how each technique splits the system into smaller chunks (scenarios for SBUE and views/modules for AB) which makes it harder to move one piece from one platform to another as you will have to handle each individual component separately on each platform instead of just one big chunk of components. Accordingly, I do not think that either of those techniques will be a good fit for projects that have the potential to move from one platform to another, except for very small and simple systems that might consist of one or two features where they only have one or two components (scenarios or layers/modules) which I think is very rare to happen.

Reusability

However, I believe SBUE improves the reusability of the system because of how it splits the system into multiple smaller scenario chunks that allows them to be reusable components in other systems that might possibly have the same scenarios, e.g., what happens if customer cancels an order? This scenario can be applied in multiple systems like Walmart, Amazon, or Best Buy applications. For the AB technique, I believe that its level of abstraction allows the system's components to be easily reused as well either within the same system or in a different system by splitting that system into multiple layers and module, e.g., the module decomposition view used in the air traffic control system might also be used in a marine traffic control system. Accordingly, I believe both SBUE and AB techniques greatly support reusability in the same way while using different kind of components. SBUE uses scenarios as its components while AB uses layers and modules. Accordingly, SBUE fits the best with projects that relies more on user interaction for decision making like merchandise ordering apps, while AB fits better with more abstract systems that consists of multiple layers and processes like garment manufacturing systems.

Conclusion

In summary, through this project, we evaluated both SBUE and AB techniques using McCall's list of software quality attributes and discussed how each of these attributes are affected by each of the two techniques. We concluded that correctness, usability, maintainability, flexibility, testability, and reusability attributes are significantly improved by both techniques, while reliability, efficiency, and integrity might be slightly enhanced but not as same as the previously mentioned attributes. In contrast, portability is the only attribute that was considered as a drawback for using both techniques.

Additionally, we also discussed what kind of projects would be appropriate for applying both techniques. Projects that rely heavily on users' interaction and their continuous feedback, especially small-scale projects that have multiple user scenarios that only relates to one system, should consider using the SBUE techniques. However, projects that rely more on the functionality and process flow than user interaction and feedback, especially large-scale projects that consists of/affects multiple layers or even multiple systems, should consider using the AB technique. That is, I believe projects that involve more software than hardware like mobile or web applications for business/sale services would be appropriate for applying the SBUE technique e.g., DoorDash, Etsy, etc., while projects that involve microservices, internal systems, manufacturing process, or services that use multiple hardware components/layers (like sensors, controllers, etc.) would be appropriate for applying the AB technique e.g., Tesla manufacturing process, Microsoft Azure, etc.
