CSE 503 Assignment 1: Brainstorming about software development difficulties

Liang He (UW User ID: lianghe | Student ID: 1774517)
Estimated hours for this assignment: 5 hours

Difficulty #1 (implementation):
Situation description: Adding new interfaces or functions to a component in a large program might result in a big change or even reconstruction of the entire program. This happens when the user requirements are not clear and confirmed before the development and more functionalities are needed to implement during the process of the development. For example, an object is created with a set of attributes and behaviors. This object is used in many components in the program. However, one of the object's behaviors has to be overridden because of a new requirement and all those components that use this object are affected. The worst situation is that the whole object might be reconstructed or integrated into other components. In that case, the dataflow and interfaces in the components that use the object data need to re-examine. This is frustrating when the program is large and the components are highly coupled.

Why important and why challenging: I think this important because this happens very often in the software development practices. During the development of a software, programmers are always facing a situation where the functions need to be changed or new features are added. No one has addressed this problem because eventually they can be manually fixed as long as the programmers spend sufficient time debugging and refactoring. However, I think it is challenging when the program is becoming larger and larger and scaled to a larger team for maintenance.

Possible solution: I can image one possible solution to help programmers to deal with this challenge or relieve programmers from the frustrating experience. It could be an auxiliary tool that helps the programmer mark and plan flexible interfaces for those undetermined components when the program is still at an early stage of the development process. When a change is made, the change will automatically propagate to other related components in the program. The programmer also can easily track this change and manually fix some minor issues, since the programmer can see all affected components with this tool.

Difficulty #2 (documentation):
Situation description: When multiple engineers are working on the same program and they need to look at each other's comments in the code, it might be hard for them to understand what the code snippets do if the comments are not written in a sufficiently clear way. For example, a function is commented with the purpose of the function, the input parameters, and the return values. However, it is not clear what components will call this function and when it will be called. Programmers have to build up a map in their mind when they read the comments and refer to other components in the project back and forth.

Why important and why challenging: Documenting the components is extremely important for collaborative development and long-term maintenance. When the entire project gets more

complex or the project is not well designed at the beginning, documentation plays an important role helping programmers to debug and maintain the program. It is challenging because the relationship between components is not very obvious at the first place to the programmers. It usually takes lots of time for programmers to understand what will be affected and what should be done for a particular change. This becomes more challenging when the project is getting more complex.

Possible solution: I propose a visual interface that produces a visual presentation of the abstraction of each component when the programmer is writing the code and building each component. The relationship between components are also shown in this visualization. The visualization could be generated automatically by capturing what file the programmer is editing and what keywords are added in the code. The programmer also has the option to modify some inaccurate information in the visualization manually. The visualization should also be dynamic aligned with the programmer's behavior. For example, when the programmer is editing one component, the corresponding component should be active (in a highlighted mode) and all components that use this currently edited component are list in the interface. This is helpful for programmers to quickly track the change and get aware of the current state of the change.

Difficulty #3 (testing):
Situation description: Debugging and testing the states of multithreaded programs is difficult because those dynamic programs execute unexpectedly in a run time. Programmers usually assess the program and predict the possible results manually, which might cause errors and makes debugging harder if the programmer interprets the program wrongly.

Why important and why challenging: It is common to run multithreaded programs in a project. The increased complexity of multithreaded programs results in a large number of possible states that the program may be in at any given time. Understanding why a particular state is troublesome can be even more difficult because they often fail in an unexpected way.

Possible solution: Since I don't have so much experience of dealing with this problem, I did a research on this and found some principles to help address this problem: (i) designing the multithreaded programs properly up front and avoid introducing bugs to the programs; (ii) using established parallel programming patterns; (iii) adding functions that display the state of the active thread; or (iv) designing the program in a sequential execution.