CSE 503: Software Engineering

Assignment 1:
Brainstorming about software development difficulties

Younghoon Kim(yhkim01)

1. Type checking

In dynamically-typed languages, it is difficult to guess states of programs since structures of variables can be changed. It is annoying when I insert some lines/function into the codes. For example, when I write some *Javascript* codes, I usually use the inspector of Google Chrome (or console.log() for programming node.js module.) to see what is in the variables that I want to use. I guess it's because 1) some objects' methods and properties are determined in runtime and 2) *Object* (as a type) is too abstract to guess what values it can have.

Typescript, which is compiled to *Javascript,* avoids this problem by enabling to define and check type of objects. But it requires extra codes to define types and sacrifices flexibility of *Object*. One of my attempts to keep using *Javascript* is naming variables with conventions. For example, I name an array of objects as pluralized name of the objects. "People" is an array of "person" object. (I saw this convention from RubyOnRails platfom.)

One of imaginations to relieve this pain could be easier conversion between abstract object to structured objects when programmers want. How about automatically generating codes that types abstract objects widely used in a structured way in *Typescript*? After executing several times of programs, it might be trackable that what objects are stayed with the same properties. Based on that, it might be able to suggest some skeleton codes for typing the objects.

2. Testing

It's hard to decide when I have to start writing test codes. It might be inefficient to write test codes at the very first stage of a project, since codes and development designs are changed a lot, which means that test codes should be modified a lot too. Instead of writing the testing codes, doing instant tests by hands seems more proper in this stage. But, as the project getting bigger and stable, testing codes become more efficient because instant tests are hard to cover the whole scope of codes. Even if a programmer tweaks a part of codes, it might make conflicts outside of that codes unexpectedly by the programmer. When I started writing test codes, I always felt overwhelmed to write a lot of test.

Writing test code has also a difficulty in terms of scope. For example, if I want to test a function that having a number as an input and *true/false* as an output. Is it enough to test the function with only one integer, e.g. 1? Or do I have to test other numbers like two (even), five (prime and

odd)?

I think if another program can generate tests automatically, it will be helpful to resolve this situation. But I don't have any experience of using these kinds of programs and little bit skeptical that will work well because tests I've wrote so far look very different. (It might be my fault since I'm not familiar to write test codes.)

3. Documentation

To reuse codes or import libraries, it requires to understand how they behave. When the codes are well documented and provide runnable code snippets, it is easy to catch how to use it. But most of codes and libraries are not well documented. It might be because documenting is burdensome/hard since 1) it is easy to be outdated as codes are updated and 2) hard to deliver the usage well.

I sometimes understand codes having no documents via their test codes. Well-written test codes are working as code snippets so I copy a block of it and paste on my codes. If codes can generate automatic documents using their test codes, it could be easier to maintain.