Han Zhang, ID#: 0437739

Homework #1, Software Development Difficulties - CSE490

This discusses three examples of difficulties that I encountered during software development.

Difficulty #1 – Testing of multi-users distributing software

Background: My team has developed a game software that involves multi players at different locations. All players log in the server via Internet and play the game in real time. The software is written in JAVA and tested by JUnit. Initially our team felt that this is a simple project. We shall just write a server and a client respectively for Internet communication (via TCP/IP socket), so that game data can be exchanged via the server. Key game logic will be run on each client. Our motivation for this project is to gain initial experience on distributing system and see how different machines can be connected to work together via computer network.

We first test the server with one client, and the software runs well. Communication and control are smooth. We then add one more client, play, and observe. OK. So we keep on expanding clients to test the software, by giving client software to our friends in other cities to play with us.

Problem: By expanding to over 10 players, some users report that the control is not smooth, and the game scene is not continuous / smooth (e.g. the game character may suddenly appear at a new position without proper movement from the old position). There are other problems here and there. We are not sure if these problems come from the different machines running the client code, or the network communication problem between the client and the server, or the server doesn't distribute the data in the correct way. We want to collect data from each client to diagnose what may go wrong to lead to these problems. However, in a distributed environment, we are not sure what are the right approach to test and collect data, especially that we have over 10 players to run at the same time, and our team consists of only 3 people. This leads us to further question. If 10+ players are a problem to us, how can those real-time commercial game system (that support hundreds of thousands of players) be tested in a big scale? JUnit seems to be OK when we test single user program. But we are not sure how it can be used in a distributed environment. And are there other better ways to run the test?

Solution: Without a proper way of testing, we decide to "guess" what the problem may be, and revise the code accordingly to see if the problem disappears. However, we find the efficient is very low this way, again, because of the distributive nature. Most problems happen when client number exceeds 10; and it is hard to coordinate our friends to run the tests with us. Later on, due to time pressure to deliver the project, we move the game logic to the server, and have the server control the game progress and data calculation. This way, the clients will only accept data and display them in the user interface. Many problems disappear; but the game is far less responsive on the client side. User experience is not good.

Difficulty #2 – Hard to estimate what input users may provide

Background: I need to write a web server for a project of CSE333 class. The requirements and specifications are clear because the project requires execution of the HTTP 1.0 standard. So I take the standard, and have my server processes the different type of requests accordingly. The

process is tedious but not difficult to execute. Actually it is quite "mechanical". I test my server from my browser and the server responds OK. I used Eclipse and Junit for development and test.

Problem: When TA graded my server, she commented that my server has two major problems: 1) it can crash when user provides weird URL requests. Obviously, my server didn't consider all the situations that a user can provide to the browser. 2) it misses requests, especially when the server sends 2+ requests in one session; and my server only processes the first request. I took TA's test case and felt surprised that a user can provide such requests to the browser. Scenario 1) is understandable; but I didn't even know that scenario 2) can be accepted by the browser without a problem. To complicate the case more, different browser seems to send some formats slightly differently, and my server fails to respond to some of the different format.

Solution: Once knowing the different inputs that a user can provide to the browser, it is straight forward to add codes to the server to handle these situations. Thinking further, I need to have a default function to process all other requests after the normal requests are processed. But scenario 2) seems to be a harder problem to me. I have no clue at all that 2+ requests can be sent in one session. HTTP 1.0 standard doesn't mention this also. Except this scenario 2), are there other possibilities that even the TA does not know (and so doesn't include in her test bench)? It is simply hard to know what crazy things a client can create to give to the server. Thinking this further, isn't this the reason why there are so many hackings happening out there?

Difficulty #3 – Know the software development principles but don't follow them in execution

Background: From various classes and books, I learn quite some software development principles. E.g. Effective Java teaches us how to write codes that don't depend on each other ("decoupling"). I also learn different design patterns.

Problem: However, in real life, when the team are under time pressure to deliver the project in two weeks, and everyone seems to have other projects / homework to do, all these good practices seem to be forgotten. The team seems to be happy with "workable" code: as long as the code can deliver what the project brief asks for, good practices are not important. There are so many codes written for a project: TA is unlikely to dig into our codes to see that we don't follow good practices of software development. This reminds me reading some reports, saying that software engineers are under tremendous pressure to deliver a lot of new features within a designated time frames; so software quality decreases dramatically. Software is shipped with less features (or not reliable features) at the end.

Solution: We must stick to the good practices to ensure the software quality is good; and make software maintenance easy for other people in the future. However, realistically as a human, software engineer may not be on the same team forever. He may not be in the same company forever. He is evaluated more on the short term (can the software be ready NOW?) rather than the long term (other people needs to read / maintain the code). So what motivation can a software engineer have to stick to the good practices? Furthermore, different software engineer has different training / understanding of good practices. How can the whole team follow a consistent software quality standard? OR, are there a measurement on what a good software quality standard shall be? Why Linux is so much more stable than Windows?