Jakob Sunde
uw netid: jsunde
cse id: jsunde

One problem that I have run into in development is that, when writing tests, it can be difficult for a programmer to determine what parts of a program need to be mocked, and what aspects need to be incorporated from the program as is. In many cases, almost everything needs to be mocked so that a module can be tested without any dependencies on other aspects of the program. This can become tedious because as the program evolves and the interfaces that are being mocked change, the tests might still pass, when they should be failing since the module being tested relied on the previous interface which is still being mocked in the old way. This wastes programmers time and can lead to bugs being found long after they are introduced making them harder to track down. It might be difficult to automate the creation of mocks because it isn't necessarily easy for a program to infer what reasonable values a mocked class might return. A potential way around this could be some type of formal language that specifications could be written in, and then as the specification for an interface is updated, a mock could be created from that specification, potentially returning default values in the specification.

Whenever working in a codebase that is either largely written by other developers or one that you haven't worked on in a while it can be incredibly difficult to get yourself situated and piece together how a program works and why things are done the way they are. Lots of developers have problems properly documenting their code. I think this is for several reasons; it's less interesting than writing code and the developer just wants to move on or is "in the zone" and doesn't want to stop to document what they are doing, they see it as a waste of time, it can be time consuming to write out documentation with the right formatting, it's easy to procrastinate and just say you'll get to it later. This can end up being very costly for a team, whenever a developer tries to work on something that they didn't write that is poorly documented they end up spending a long time trying to decipher what's going on and what their next step should be. I think one way that is already in use in some IDEs and could be expanded is documentation inference, where it fills out the stub of documentation for you based on a method signature. I think developers might be more inclined to write the documentation the less work it is for them, and also might feel the need to fill it out if this documentation stub was automatically added whenever they declared a method. Another more drastic method of improving the amount of documentation that developers write would be for the IDE to give them a notification whenever they try to check in code with incomplete documentation, just like it does if you have added any TODO items, it warns you, "You have methods with incomplete documentation." This might help developers stay on top of the documentation of their code making it much easier for other members of the team or newcomers to understand their code.

Working on group projects in the past I have run into problems with merge conflicts, because I was working some part of a program while another developer was at the same time. When one of us checked in our changes it was fine, but then when the other one tries to, the version control system says that there is a merge conflict it can't resolve that must be handled manually. This can be a tedious job requiring careful analysis of the changes made by both parties to see what changes need to be kept from both changesets and what changes might be similar enough that only the changes from one changeset needs to be included. This is a problem that is incredibly difficult for a program to handle, it is already very good at handling the changes of multiple users, but when the changes occur in the same area it is hard to decipher which changes need to be included. I think an interesting solution to this problem would be a Google Docs type IDE that allows two developers to work on a file concurrently. It might give them the option when they are about to edit a document and say "Another user is currently editing this file" and then you could jump in and edit the document at the same time, or opt to work separately and let the version control handle it. This might still allow you to run into the same issue, but I don't think the concurrent editing would always be the best option so I think it would be important to allow the developer to still have the option of working independently on the file. However, I think this type of concurrent modification of programs could help reduce the time spent on merge conflicts and would allow for easy Pair Programming over distance.