The Difficulty:
The underlying cause:
If you were to address this, what would you use? Ways that tools would help?
Why people should care about it? Why it is not easy to solve or work around (discuss, why current tools and techniques do not address the problem)?
Speculate why no one solved it to date?

- **Documentation:**

The problem: When software development has to be done in shorter amounts of time, especially in the case of deliverables or deployments, there is less time for documentation and sometimes teams skip documentation. This happens most prominently when it is to fix a bug, ship a feature or meet a deadline or patch that is important to the organization.
Importance: This is important because whenever the code is to be reused, refactored, or even added to, it needs to be understood or documented before moving on and sometimes the best person to document (who wrote the code) might have left the team or would not recall each and every detail.
As a hack, if it was my own code, at a research lab and time was running out, I would recruit someone to document with me. But I propose, using the program to generate documentation, reverse engineering the specifications.
It is not easy because there are both syntax and then human intentions (I am creating this invariant to use at this certain place) and where syntax can hint at some reasoning, intentions might not always be clear from the code itself unless the author is also not with you.
Possible solution: I think some automatic documentation (using some yes or no input from the developer) would be better than no documentation. Using questions and inquiries from the programmer, (and maybe even NLP) a tool should be able to generate documentation. **Currently,** tools like Javadocs allow API documentation, but end-to-end program comments and documentation will be much more useful over long term changes.

- **Testing:**

The problem: When writing code, one can best imagine or predict the errors that the users or the program can end up making and then either put checks, exceptions or build some test cases. However, once you are done coding, you only remember the core basics and then end up testing, thinking, improving only on them. Besides unit tests, most organizations have specialized staff for testing and development. Once the code is written, it is committed and then it is tested and sent back. Similarly, once the code is refactored, then it is tested. Similarly, when requests are received to add features or add properties in an otherwise complete program. So, when editing or refactoring, developers only focus on the code that they are changing. Some of it might point to how or where it is being used or other parts of the program that are being affected. Developers run the code to see if it

does what it is supposed to do, not test for what it is not supposed to do. A separate, after the process, testing comes in.

Currently: I think current systems monitor as the users' type but only check for type checks and other obvious effects. Automated checking of overall program structure changes and effects of code changes might be computationally expensive or maybe distracting to the developers. Current tools also tell the developers if a name is changed is being referred elsewhere but I don't think they inform upon changing a value that now the program has become inconsistent or invalid at certain steps.

Solution: I think firstly code visualizations or code visualizers would be the best way to get a total understanding of all the components of the program. Secondly, automated checking of the program completeness and soundness as the developer types the program, edits or updates it might be useful in mapping that the overall changes, as well as the global effects on the program, are also managed. Showing the total variables and their interactions, as well as visualize the program will make the developer understand simultaneously what is being changed and the overall workflow and what parts each unit or test case covers. Thus, if a part is unused, untested or affected by a change the visualization should also alert, saving time.

- **Design:**

The problem: Specifications are designed by the product managers or senior team members and are mostly designed by talking to customers or going to the field. Developers only try to develop in terms of their understanding of the specifications and sometimes there is a difference in understanding because developers have not been talking to the end-users. Thus, they end up making assumptions or developing for optimal users. Specifications are mostly in the form of documentation and are understood based on the perception of the developers.

Currently: Teams have incorporated user stories and frequent team meetings are used to help remove these confusions and to help let the entire team be on the same page. However, if a developer doesn't communicate about a feature or implements what s/he understands, then it goes till testing to find out the misunderstanding.

Solution: I can't think of any tools that might help this more as this is more of a communication problem. And tools like Jira/Confluence try to minimize the communication gap between developments teams. What we can do is we can also create small releases, and quick iterations to find these gaps of understanding and communication earlier and quickly in the implementation, rather than letting the developers build and expand on inconsistent or incorrect understandings.

We can also create a formal language to map priorities, preferences, and no-change values, etc.

Time Taken: It took me 3.5 hours to finish this. A few problems that I thought already had tools and solutions out there (or were just bad programming practices).