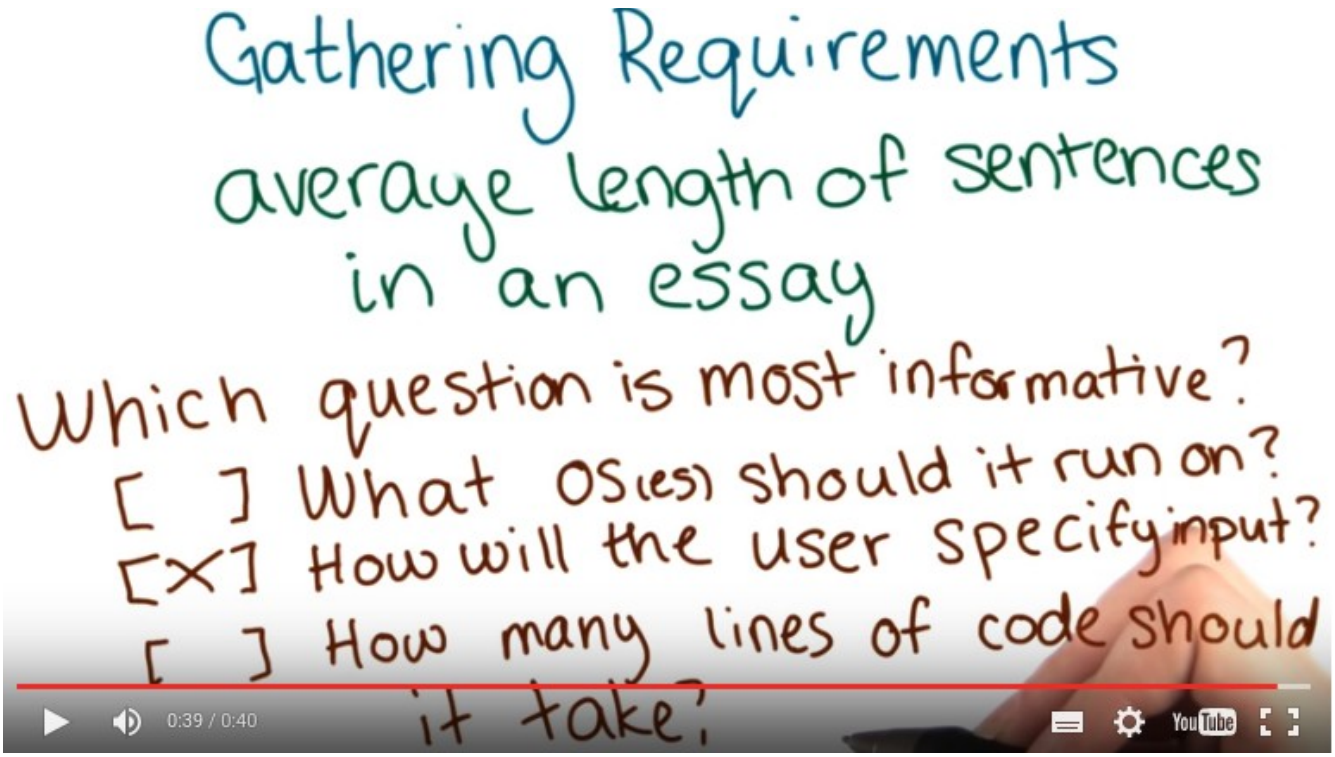
1. Gathering requirements is one of the most difficult tasks a software engineer faces. In industry, you may gather requirements from end users, external clients, or from co-workers in other areas of your own company. Occasionally, you may receive a well documented set of requirements. However, in most cases, you will need to glean the requirements from conversations with the prospective clients, and distill them down into something actionable on your own. Suppose a teacher came to you with a request for a piece of software their students could use to find out the average length of the sentences in their essays. What questions come into mind to help you figure out the full requirements for this project. Write down a list of at least ten questions that might help you determine them. Please take the time to do this before moving on. There's no penalty for looking ahead, but if you skip this exercise you'll cheat yourself out of the benefits of brainstorming and getting your mind around the project before being bombarded by more information.

2. Now that you've written some of your own questions, consider the following three. Which is the most likely to be useful for determining the detailed requirements? Maybe, what OSes should it run on? Or maybe, how will the user specify input? Or finally, how many lines of code should it take?

3. The last question is almost never a reasonable one. For one thing, the client should not need to know or care about how many lines of code make up the program's source code. In forming requirements, you should avoid implementation specific questions that do not directly interface with the user. The first question is very relevant in some situations. For example, a graphic sentence with video game or performance is key. However, you should not write any operating system specific code unless absolutely needed, and should strive to make your code platform independent whenever possible. The second question, however, is very relevant. Now that you've thought a bit about what you might ask of the client requesting this program, let's watch Alvin, one of Udasea's engineers, asking his own questions. He'll be speaking with Lauren, the client.

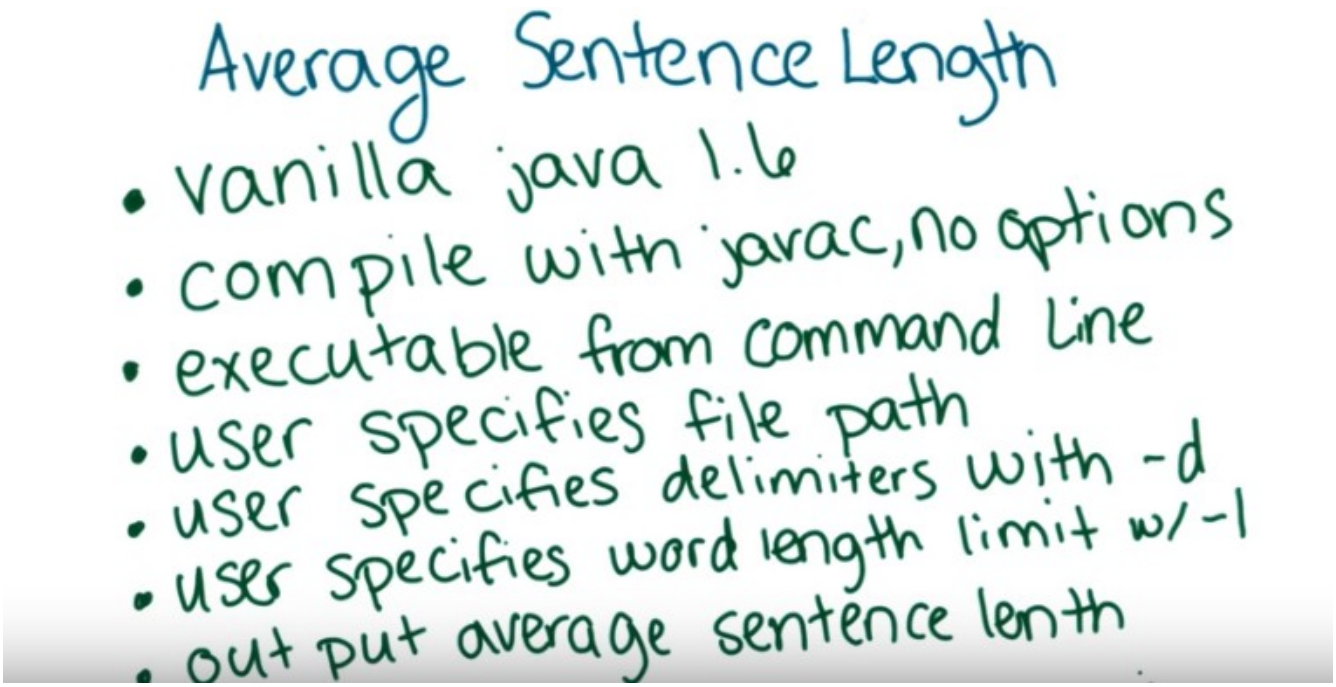


4. Hi, I'm Lauren. >> Hi, I'm Alvin. >> I'm an instructor at a university nearby and I've been noticing that when my students write their essays, they have very long, very wordy sentences and I would like to develop some kind of tool that they can use to keep track of this and maybe perfect their writing style. Do you think that's something you could do? >> I think so. Let's start by helping me get acquainted with the students in the class. So how many students do we have in this class typically? >> Usually about 45 per unit, but I can have up to six units a semester. >> 45 students, and six sections per semester. That's a farily reasonable size. So, do you know anything about what the students are using as far as computers go? >> I don't know what kind of computers they're using. And they could be, I don't know, anywhere from having no tech experience to being pretty proficient. >> Do you know anything about how familiar the students are with computers in general? >> I'm sure we have some people on the low end that have never done any type of programming, and then some people who are pretty self-sufficient. >> Okay, and I guess my last question related to the students is, what is the students actually submitting to you. >> They've been sending just raw text files via email to me. >> So from the sounds of things we have a fairly broad, I guess base of students to work with, both in technical proficiency as well as their operating system environments potentially. So I think what we'll probably do to start off with is make a command line, Java based tool. That the students can run and we'll give them a fair amount of you know, documentation on how to use the tool. And I expect that there will be a lot of little error conditions that may happen that we want to produce a reasonably friendly message were anything to go wrong. >> Yeah. That'd be great. >> So, a little bit more, I guess about the actual essay itself, its submission, what constitutes a word? >> I really only care about the longer words, so, is there a way that we can only count words that are maybe above three letters? >> I think that's something we can do. And I think that because you seem a little bit unsure we might be able to have that be a little bit more flexible than we otherwise would. >> Great. >> What does a sentence mean to you? Is it kind of flexible? >> I would say anything that ends in a period or even a question mark. Maybe even an exclamation mark. or, something even, maybe even with a comma or semi-colon. I really only care about the sentences that aren't gramatically correct and are too long. >> I think we can probably make that a little bit more flexible too so that way, you can kind of say we're going to, or you want to include them. >> Mm-hm. >> So maybe, sounds like you are little bit on the fence about whether or not say, a comma should be considered a sentence. >> Mm. >> Entirely on its own or not. So we can probably make that a little bit configurable as well. And so, just to confirm the actual end result to the student is the average number of words per sentence? >> Yeah, yeah that'd be fine. >> Okay, overall to start off with, looks like we have some sort of customization of the word length that we want to look for. >> Mm-hm. Yeah. >> We have some kind of variability in what we want to have as acceptable sentence structure. So, periods, question marks, semicolons, things like that. And, the end result to the student is if we're successful, they'll get the average number of words per sentence. Otherwise we tell them something a little bit helpful to kind of put them on the right track to use the tool correctly. >> Yeah that's the error codes right? >> Hopefully not error codes but something a little bit nicer. >> [LAUGH] Okay. >> So I think I have enough to get started and produce something that's you know, a reasonable I guess, rough draft. Of something that you can use to help your class out. >> Great. Thank you. >> Thank you.

Instructor note: Please note that for your team's implementation of Lauren's requested program, **the comma is not to be used as a default sentence delimiter**(all the other delimiters that Lauren mentions should however be considered). Also, Lauren forgot to mention that **":" should also be a delimiter**.

5. You've heard Alvin come up with several conclusions for how to set up the program. And we're

going to ask that you follow his instincts. We'll spell that out here in a little more technical details so that everyone is working from the same basic starting point. The program must be written in Java and must not make you any nonstandard Java libraries. You will be tested on a machine with the vanilla installation of Java 1.6. Your program must compile on the command line using the Java C command without any additional options. All code required to execute the program that is not part of the standard JDK, must be included as source code with your program. Your program should be an application. That is, it should have a main method and should be executable from the command line using the Java command. The user should be able to provide a file path to the file they wish to be analyzed as a command line argument. User should be able to specify which delimiters count as sentence separators, using the flag -d, defaulting to Lauren's initial thoughts on what should be used as delimiters. The user should be able to specify a lower limit for word length, using the flag -l, defaulting to Lauren's guess at what value might be good. Finally, the program's output should be the average sentence length. Rounded down to the nearest integer.



Instructor Notes:
Even if the video says otherwise, it is fine to use more recent versions of Java for this project (i.e., 1.7 and 1.8).