Ian Anderson

Chapter one outlines the steps that need to be considered with the design process of creating a program in a professional setting, as well as some difficulties software engineers face, regardless if the program is simple or complex. These difficulties are separated mainly into two categories, program requirements and design constraints. Program requirements are what the program needs to do, while design constraints are the limitations present when designing the program. Program requirements can be further broken down into functional and nonfunctional requirements, or what the requirements require the program to do and the way it needs to be done. Examples of nonfunctional requirements include performance, usability, security, reliability, how well it can be maintained, and how well it can work on a larger scale. Functional requirements on the other hand are usually specific to a particular program, such as the file line sorting program mentioned by the chapter. Examples for that program in particular include what would be the input format for the file that needs to be read, how the lines will be sorted, what characters will be accepted, boundaries, and error conditions.

Design constraints differentiate with both the program itself, and the clients requesting the program. Examples include whether the program should use a command line interface or a graphical user interface, the platforms the program is meant to be used on, the time in which the program must be completed by, and the time and space complexity of the program itself. Apart from the restraints, there are also several design

decisions that must be made by the developers, such as deciding which programming language to use, and which algorithms to implement to improve time and/or space efficiency. Following these decisions, and once the constraints have been adhered to, it is important to test the program throughout the entire coding process and not just at the end. Typically, the best way to do this is through unit testing, where the programmer makes a set of input/output tests to ensure the program is working as intended. It is also important to keep track of how much time is being put into the project, keeping track of how much time both in hours and calendar days worked. During development, developers should set an ideal time constraint for each section of the development process such as planning, testing, and programming. It's a good idea to aim for that time for both financial and scheduling reasons.

Finally, chapter one discusses implementation. This varies from program to program but a couple general things to keep in mind include being consistent, such as keeping naming scheme the same across the entire program, smart naming schemes, testing each function/method one at a time to narrow down errors, and share your code after completion to be reviewed by others to find issues you missed. For the program mentioned in chapter one, implementation includes user interface, JUnit testing, and sorting methods. The chapter goes in depth by showing the code for the discussed program as well as the GUI, and alludes to the costs of GUIs that will be discussed in chapter seven.