Ryan Davis

SSW 567

Software's Chronic Crisis

1. Given that the problems of the Denver Airport derive from software-scalability issues, these issues are very common. Creating large software at the time entailed delays, overrun budgets, and logical issues and errors are common issues of software engineering(Gibbs 8).
2. The percentage of "operating failures" as observed in the mid-1990s is "some three quarters of all large operating systems" (Gibbs 3). The average increase in scheduling time "overshoots its schedule by half" (Gibbs 3).
3. The Software Crises include "the difficulties of building big software" (Gibbs 4). This entails the proper construction of working code, the debugging of the code, the testing of the software, and the implementation of the software.
4. Software Engineering is defined as "the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software." (Gibbs 4).
5. Outsourcing affects software standards and interoperability. With many different contributions to a software, it is a challenge for different teams to provide consistent code that will allow the true functionality of the software to be maintained. However, interoperability is beneficial in terms of testing. Different perspectives are beneficial to viewing the functionality of the software, as different teams can provide different inputs to the system. Also, outsourced programmers are often cheaper (Gibbs 96).
6. As provided, "getting software right the first time is hard even for those who care to try" (Gibbs 10). Because the correctness of a program is difficult to identify, thorough testing is implemented to spot bugs and inefficiencies in the software. Even with complete discipline, several issues can arise in different circumstances and use cases.
7. Realtime Systems are unique because issues "are devilishly difficult to spot because… they often occur only when conditions are just so" (Gibbs 11). This implies that there are certain cases where a program can fail, relying on strenuous testing. In Distributed Systems, programs can "run cooperatively on many networked computers," (Gibbs 12) meaning that data can be exchanged throughout a system, and errors are more easily spotted.
8. The Loral team experience was mostly successful, as they "learned to control bugs so well that it can reliably predict how many will be found in each new version of [a] software" (Gibbs 32). Although bugs can still slip through, the group was able to achieve a CMM rating of 5 and prevent future bugs from reoccurring in software.
9. Formal methods allow for error detection in systems, preventing bugs and logical systems through mathematical proof (Gibbs 35-38). The errors in such methods lie in the computational errors that may arise. Furthermore, mathematical computation cannot guarantee the functionality of the deployed software. For example, Odyssey is a company that applied formal methods. Ted Ralston of Odyssey Research Associates provides that formulas are more complex than the actual code (Gibbs 39).
10. Component software is desirable as this structure can be dynamically implemented. In assembling large software, different components "can derive compatible versions by adding additional elements to their interfaces" (Gibbs 88).