Deliverable 4.1

It is impossible to determine the true cost of many projects until they are finished, and software is exception. In fact, the abstract nature of IT-related investments makes it much more difficult to place firm estimates on the ambition and scope of the code to be written, which according to "Why Software Fails" often leads to rather vague and arbitrary specifications that cause an colossal mismatch between the final product and the customer's demands. Contractors often offer impossible time tables in order to win a bid, and the increased pressure on developers to deliver inadvertently leads to failure.

Although it is difficult to pinpoint the specific cause for all errors, the article attempted to narrow down the scope of most: gross mismanagement, largely attributed to systematic (methodological) failure. These descriptions cover many of the common factors outlined in the article, such as unrealistic or unarticulated project goals, inaccurate estimates of needed resources, badly defined system requirements, poor reporting of the project's status, unmanaged risks, poor communication among customers, developers, and users, use of immature technology, inability to handle the project's complexity, sloppy development practices, poor project management, stakeholder politics and commercial pressures.

Although the article goes into detail on what usually goes wrong, it gives very little remedies. In the case of Sydney Water Corp., like many other projects described in the article, inadequate planning and specifications doomed the project rather than any technological limitation. Hence the best solution to future mismanagement is that of Praxis, a British company which had made it its mission to complete through and rigorous specification outlines prior to engaging in any project. The article suggests this is the best way to prevent future issues in the project, and save cost in the long run. Without careful and extensive analysis, the authors are skeptical problems with bad planning leading to bad software will be resolved.

Similar problems plagued the Healthcare.gov site, in addition to technology problems not mentioned as much in the first article. The Heathcare.gov site was contracted out to a few, select contractors promised timely delivery but not necessarily quality assurance. The nature of this being a government contract has forced these software houses to abide by arcane federal law, such as the forced discontinuity between back and front end developers - which in turn led to severe API mismatch. Moreover, the nature of the federal contract in a changing political environment dictated the specs changed at least seven times - an intense undertaking for a company already having to interface with many many legacy systems (where modifications are not trivial).

The lecture described so called Agile development, which consists of a back and forth between a client and a contractor to determine the exact deliverable. However, federal contracts in effect specifically prohibit Agile development, opting for instead highly predetermined and firm specifications on the contractor side (which can only be changed unilaterally). Prohibiting this in effect causes a "Waterfall approach" for development and testing, where expectations win over realities and testing is saved for the last minute.

The issue with the last minute testing of the website was an example where technology and policy were both lacking. The site was unable to handle the load on day 1, and because the product was rolled out to 100% of potential users the developer could not subject the product to real world testing. The technological shortcoming was its inability to handle a serious load, which some articles cited as being tested the day before for up to 5000, less than the 50000 expected and much less than the 300000 actual users on day 1.

I do not believe, and neither does the article or the lecture, that errors in large scale software projects can be attributed to a fundamental flaw in the theoretical workings of a system. Most can be attributed to bad management, and some can be attributed to lack of proper testing (which can be a result of bad developers, bad management, or both). None of this can be classified as "human error" - both sources stress **no code should be viewed by only one pair of eyes**. Regardless of the size, debugging and load testing should be done extensively and by multiple people. A mistake like the mismanagement of the projects described in the sources is on the scale of many, many people, and is hence no more human error than congress is - it is rather systematic failure.

In a world increasingly modulated by machines, quality assurance becomes ever the more critical. Gross mismanagement of many IT projects leads to unrealistic expectations leads to faulty products. Developers must employ a careful, hierarchical approach, and deadlines must be placed realistically as to not compromise testing.