Liam Brew Professor Ens

SSW 555 09.06.2020

Homework 01: Driverless Cars

Problem Identification

There are numerous problems presented by this design challenge. First and foremost is the fact that this system is responsible for human life and safety, and therefore requires sufficient development resources to ensure safe and successful operating conditions. This places it in the most demanding tier of software systems which is those that deal with user safety. This need must also be balanced with many potential business problems and concerns. For example, if the car is “too safe” in that it is overly slow, meticulous, and therefore inefficient at completing journeys, it will suffer in the competitive marketplace as many customers will prefer conventional vehicles. Additionally, overly complex self-driving software will increase the price of this vehicle model. As it can be reasonably assumed that the cost of a self-driving car is greater than that of a conventional car to begin with, any further price increases may dissuade customers and result in a loss of sales. Finally, the car must be developed and deployed quickly enough to remain competitive against competitors, both self-driving and conventional, as it can be assumed that they are updating and improving their product lines as well.

All these problems must be balanced with each other to ensure a successful and profitable product. While safety is foremost, as previously mentioned other aspects must be considered as well. Working with the limited confines of a development team different priorities and acceptable levels must be set to properly allocate what resources are available. Each of the three SDLCs of Waterfall Method, RUP and eXtreme Programming present several unique advantages and disadvantages for working in this project environment.

Waterfall Method

The process of implementing the Waterfall Method involves a sequential approach to developing the software and its environment that will be used in the car. As it is relatively unlikely for requirements to change (the objectives and working environment of the technology are not very flexible to warrant this), effort should be prioritized on the Design, Implementation and Verification stages. The mechanics of the Waterfall Method lend themselves well to these stages, as completing them thoroughly will help reduce the need to make later changes down the road as well as potentially reduce the maintenance burden on the system. Since the project does not have any existing software to make use of, a complete build is necessary. Therefore, work should begin with low-level hardware interfaces and then progress to the operating system. Once that has been established, work on tasks such as the logic behind the car’s driving routines can then begin. To efficiently complete the project, a larger size team would be needed. This is due to the complex nature of the system and the large amount of risk involved.

The implementation of the Waterfall Method holds multiple advantages for this project. Due to this method’s precision and meticulousness, it has a long history in being used in life-or-death critical areas. This track record lends itself extremely well to this project as the cost of failure is extremely high due to potential lawsuits and negative media coverage. It can be reasonably assumed that the requirements of this project will not change much beyond those initially specified, as the objectives of the project can be quite precise and well-defined. This does well to help mitigate the impact of this method’s inflexibility in dealing with requirements changes. Since this project is being built from the ground up and does not make use of any pre-existing software, the Waterfall Method’s thoroughness lends itself well to developing a solid and capable code base from which this project will be based off of. When there is no pre-existing infrastructure, it may be advantageous to sequentially develop software and systems to give later systems a starting point. Also, Waterfalls' extensive documentation will be beneficial here as well in creating and cataloging the initial baseline of new technology.

Despite this, the use of the Waterfall Method does bring disadvantages as well. It is a given that the project will have many unknown factors that may significantly impact development. The rigid structure of the Waterfall Method is not ideal in accommodating large amounts of experimentation towards solving these unknowns. There will come a point in the development process at which a specific solution or path will have to be chosen so that the remainder of the work may be pursued. Due to time constraints limited research and experimentation, this solution may not be the most optimal, or even successful at all. Issues down the road that result from this issue will result in costly delays and modifications to the project due to this method. Related to cost, the Delivery Paradigms for Waterfall Method do not compare favorably with its alternatives. All of these when combined with the low success rate of Waterfall's development paradigm present portfolio of negatives which the development team should strongly consider when choosing their SDLC.

<u>Rational Unified Process (RUP)</u>

The process of utilizing RUP for this project deals with focusing its four stages onto the design challenge at hand. During the inception stage, a general plan for the system is created along with its basic use case model, and a preliminary cost and budget are determined. Next comes the elaboration stage, which should be the priority for this project. Elaboration is the most important here as it seeks to mitigate potential risks by truly defining the use cases and designs of the software system. As this project contains a large amount of expensive risk, it is important to minimize it as much as possible before any actual code writing takes place. The more use cases that are developed and planned out, the quicker code of sufficient quality can be produced. This assists in the later construction stage that encompasses the implementation and testing of software. The more use case scaffolding that was produced during the elaboration stage, the quicker this construction stage will go. Finally, the transition stage features the finalized deployment of the system. This also represents a major stage that should be prioritized, as this is the last chance for any remaining issues to be solved before release. Due to the iterative approach of RUP, it can be reasonably assumed that a smaller project team is required than the one recommended in the Waterfall Method.

RUP provides several advantages for the development of this project. RUP's design philosophy of tackling the larger risks first will help reduce the overall risk and danger presented by the project. Its emphasis on reusable, component-based software as well as flexibility in design and requirements changes makes it a good candidate for experimentation and the discovery of any unknown factors. This also lends itself well to any customer modification requests that take place during the development process. Its basis on the best practices of software engineering ensures that the produced code is of high quality in terms of both conventional standards as well as continuous verification and validation testing. The supporting disciplines of RUP, most notably project and configuration management, facilitates a controlled environment that is continuously in sync with user needs.

A potential source of issues for the RUP method is the fact that no preexisting software exists that this project can make use of. This requires the generation of a complete code base off which the system can be built. Because of this vastness of work that is required, an iterative approach may not be best in this situation. The use of RUP may require too much time dedicated to requirements mapping and component breakdown, resulting in the resources available for code development being reduced. With so many subcomponents both within the software system as well as outside of it (various cameras, sensors, motors etc.), it may be difficult to attain the same level of thorough safety and quality evaluation that the Waterfall Method provides. Additionally, due to the relatively static nature of the requirements of this project, it may be impossible to make the most out of RUP's improved flexibility.

eXtreme Programming

The eXtreme Programming process begins with the Planning Game stage of design. Here, business analysts and technical engineers decide upon their respective aspects of the project. This stage may and should be considered a priority due to the following decisions that take place during it: priority, effort estimation, and most importantly technical consequences. The latter will have a large impact on the safety of the finalized system which is a key aspect of the design problem. This relates directly to the Simple Design and Testing practices, two other processes which are prioritized. The Simple Design aspect ensures that the design of the system is as sleek and minimalistic as possible with no superfluous features. This helps reduce the likelihood of failure in the final product as there are digital moving parts. The Testing practice ensures that the system is sufficiently tested enough to ensure favorable safety ratings, with the Continuous Integration practice proving added benefit and security to this measure. The final prioritized practice of eXtreme Programming is Coding Standards. This provides an additional layer of security in that all the team's code is looked at by members. While these are the prioritized practices of eXtreme Programming, the remainder contribute to the benefit of the project as well.

The benefits of eXtreme Programming on this project involve its ability to respond to any requirements changes that may appear. It offers a large degree of flexibility in dealing with changes as they occur, embedding them into the project without drastically impacting in-progress development work. Additionally, eXtreme Programming handles presented by new technology in fixed-time projects well. This is applicable here. As stated, this project has no prebuilt software base to draw from, meaning that new technologies will inevitably have to be used. Additionally, it is reasonable to assume that there is a fixed time allocated for development that fits within the project's greater release date schedule.

Despite its benefits, eXtreme Programming may not be the most ideal SDLC for this project. As previously mentioned at length, the requirements of this project are unlikely to change drastically, negating much of the benefit that eXtreme Programming offers. Additionally, it is unlikely that only automated tests will be made use of. While automated unit and functional testing will play a role, this type of project requires end testing of the finished product (incorporated hardware and software designs). This is to ensure the safety of the delivered vehicle in its end state. Furthermore, this SDLC's preference of a small and co-located team may not be feasible in this project's environment. A safe car represents a significant engineering challenge to tackle and therefore many engineers. The self-driving software team will have to interface significantly with other teams, such as the automobile design team as well as the team responsible for implementing the various sensors and motors that the software will be based on. It may not be feasible or even possible to have all these entities at a single co-inhabited location.

Conclusion

For this project I would recommend the Waterfall Method. While the other SDLCs certainly have their benefits and may result in an overall more efficient project process, the Waterfall Method has a proven track record for dealing with these types of critical software projects. As safety is one of the most important aspects of this design problem, it would be best to stick with a methodology that is known to work.