INJECTING SAFETY-CRITICAL CERTIFICATION INTO AGILE SOFTWARE METHODS

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Scott James Minot

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Software Engineering

November 2013

Fargo, North Dakota

North Dakota State University
Graduate School

Title

INJECTING SAFETY-CRITICAL CERTIFICATION INTO AGILE SOFTWARE METHODS

By

Scott James Minot

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Kenneth Magel
Co-Chair

Dr. William Perrizo
Co-Chair

Dr. Sameer Abufardeh

Dr. Robert Gordon

Approved:

11/5/13
Date

Dr. Brian M. Slator
Department Chair

ABSTRACT

Agility offers an adaptable and changeable environment within software development. The benefits that agile methods provide for software development are becoming an even greater possibility in safety-critical software programs. These certified programs go through a rigorous process to ensure the safety of all people involved. As the systems become more complex and there is a need for adaptability, the benefits of agile could save companies considerable time and money without sacrificing the safety factor. In this paper, I will provide ways of incorporating certification for highly critical systems by using a form of agility tailored to fit certification requirements. With time reduction and an increasing ability to change safety-critical software, it will show that it can be a viable option to deploy.

ACKNOWLEDGMENTS

I would like to first thank my supervisor, Dr. Kenneth Magel for his patience, understanding, guidance, constructive comments, and support throughout the development of my work. I would like to thank Dr. Robert Gordon, Dr. William Perizzo, Dr. Tariq King, Dr. Hyunsook Do, and Dr. Sameer Abufardeh for accepting to be on my committee while providing constructive feedback and comments.

I also would like to especially thank the Psychology Department for being understanding and accepting of the time to complete my degree while working full-time for them. I wouldn't have completed this without their flexibility.

DEDICATION

I lovingly dedicate this thesis to my wife, who has supported me every step of the way. To my daughters, Emily and Olivia, for understanding that learning is important throughout many stages of life. To my parents, who have passed on, for the importance of knowledge and hard work. I would like to also thank my Lord Savior, Jesus Christ, for his guidance through this process.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Agile Software Development is a software development methodology that helps with a specific need in a changing and dynamic world. The Agile Manifesto was written for this specific need to provide an alternative to the multitude of documentation obsessed and cumbersome software development processes. The manifesto claims that the following principles are preferred over the more traditional software approach. The first principal states that the inclusion of various individuals and their close relationships are preferred over the use of the many processes and tools that are heavily employed. The next principal states that a usable software deliverable is preferred over its all-encompassing documentation. The next principal states that there is an emphasis where collaboration and the interaction of the customer throughout its development process is more appealing than a negotiated contract without further customer interaction. Finally, the principal states that the response to various changes throughout the development is preferred over following a specific rigid plan. These are all preferred principals over the traditional software approaches. These are the principals and standards that make up Agile Software Development [4]. These standards can be further broken down to show the following objectives. The goal is to make software better.

The objective of agile is to:

- Satisfy the customer through timely and continual delivery of a valuable software product
- Changes to any part of the requirements during its work cycle is encouraged
- At the end of each iteration, a working and usable software is delivered as frequently as possible.

- Groups of people must work together daily throughout the project and build software that is highly relatable and with extensive collaboration
- All the people involved must always use face-to-face conversation and maintain a constant work pace through each iteration.
- People must provide continuous attention to technical excellence, good design, maintaining its simplicity, and increasing the output of quality work.
- Self-organizing teams must learn how they can become more effective and how to continually maintain this behavior. [1]

Software requirements are described as something that the software has to or is required to do. Software requirements sometimes do need to change in real world situations. In the course of development, this is not a big surprise and are expected to happen. There are multitude of reasons why this happens, and there has to be a way to accept this change. The key is to balance these changing requirements with as little impact to the software. The idea of continuous and ongoing development is prevalent in an agile methodology. Flexibility is an important key in ensuring that all of the customer's requirements are met. Some researchers have found that flexibility can be counterproductive and may produce undesirable or uncontrolled changes in the requirements known as scope creep. This can therefore cause an undesirable change in the project schedule's time, an increase in cost of the project, and produce an unnecessary risk. It has been reported that many companies find that safety-certified software costs roughly 10 times as much as non-certified software with its equivalent functionality [10].

Extreme Programming follows these agile principles. It can provide the strengths of agile by making the software lighter and more manageable to work with. It stresses

customer satisfaction. It provides software in usable chunks to fit a specific need that is vital to a software company. Extreme Programming inspires developers to confidently respond to the changing customer requirements, even toward the end of its life cycle. Extreme Programming emphasizes teamwork whereas managers, customers, and developers, and even users are all equal partners in a large collaborative team. This provides a simple and effective environment that enables the teams to become highly productive and release reliable software quickly [15].

Scrum also follows a very similar agile approach as does extreme programming. It includes teams where each particular roles played by a member. Sprints are specific durations of development effort and can range from one week to one month. Each sprint is preceded by a meeting which is planned with its identified tasks and an estimated and specified commitment goal. This is followed by a subsequent meeting, where progress is reviewed. Each day during a sprint, a project team meeting occurs and follows a set of guidelines where all members are prepared with updates. The meeting lasts for 15 minutes in which the location and time are fixed [47].

In the following study, I will be presenting the various certification needs that are located in the common areas of agility used today. The focus will be into molding the certification processes into the agile development philosophy while maintaining the focus of the safety-oriented environment.

1.1. Certification

Safety-critical software is a type of software that if a failure occurs in the software, it will very likely result in the loss or endangerment of human life or the significant harm to the surrounding environment. People involved in the development of safety-critical

software exercise extreme caution against system failure. Most safety-critical systems are closely regulated by government agencies with strict certification requirements. The objective of software certification is to provide a solid foundation in obtaining a reliability measure of the software and/or its components from failure. For many types of safety-critical systems, having guidelines that define its processes and objectives for the creation of quality software has tremendous value for developers of safety-critical systems [11].

1.2. RTCA DO-178B

One highly critical software certification guideline that will be discussed is called the "The Software Considerations in Airborne Systems and Equipment Certification" or simply called RTCA DO-178B [12] and is of high critical importance to aviation. RTCA or Radio Technical Commission for Aeronautics is an association of aeronautical organizations of the United States of America from both government and industry. This is the organization that produces this guideline. The FAA uses the DO-178B as a document that provides guidance in determining if the software will perform reliably in an aerial setting. DO-178B has become a standard in certified software and is the accepted means of certifying all new aviation software [16]. It's main function and purpose is within the development processes. Accordingly, the DO-178B requires the delivery of many documents and records. The quantity of documents needed for a DO-178B certification, and the amount of information it requires, is determined by the level of certification that is requested [13]. DO-178B comes in varying degrees of certification levels starting at Level A, B, C, D, and finally E. Level A is determined to be the most critical. Correspondingly, these DO-178B levels describe the consequences of a potential failure of the software: catastrophic, hazardous-severe, major, minor, or no-effect; accordingly. The DO-178B

certification process is the most demanding at its higher levels. A product certified at a DO-178B Level A would require the most thorough, labor-intensive preparation to meet these strict guidelines. Developers identify the levels of a particular functionality in accordance to how critical they are to the survival of the system. They then separate all of the resources that require less-critical functionality from the ones that are more critical. Failure of a Level-A component is considered to be catastrophic or the continued safe flight of the aircraft. Failure of a Level-B component is considered to be hazardous and severe, significantly reducing its safety limitations but may not risk the loss of life. Failure of a Level-C component is considered to be major, reducing the ability of the crew to do their job but not necessarily decreasing the system safety limitations, and Level-D that provides little or no safety issues [12].

In general, these standards and guidelines serve two purposes. It provides a way for the systems that are procured is free from defects that could lose lives. It also provides guidelines on how these systems are built, so that they perform exactly to the stakeholders' specifications. Therefore these specifications have been tested to provide the most error free system offered.

With the most critical (Level A) software, which is defined as that which could prevent continued safe flight and landing of an aircraft, it must satisfy a level of coverage called the modified condition/decision coverage (MC/DC). It is used in the DO-178B to ensure that Level A software is tested adequately. In satisfying the MC/DC coverage criterion during testing, all testing conditions must be met at least once The following steps must be followed. The first step is that each decision tries every possible outcome. This is followed by each condition in a decision takes on every possible

outcome. Followed by, each entry and exit point is invoked. Lastly, each condition in a decision is shown to independently affect the outcome of the decision [5].

1.3. System Safety Assessment Process

The system safety assessment process with the DO-178B determines and categorizes the failure conditions of the system. These requirements are designed for both hardware and software to prevent or limit the effects of the faults. Its purpose is to provide fault detection and/or fault tolerance. It ensures that these safety-related requirements are a part of the overall system requirements that form the Software Life Cycle. The system requirements in safety-certified software typically include the System Description and Hardware Definition; Certification requirements that include Federal Aviation Regulations and other required certifications; System Requirements allocated to software, including functional, performance, and safety-related requirements; Software Levels and data to support their determination; and finally Safety Strategies and design constraints, including design methods such as partitioning, dissimilarity, redundancy or safety monitoring [12].

If the system is a component of another system with those safety related-requirements and failure conditions for that system, the system safety assessment process determines the impact of the software design. It determines the implementation on system safety using information provided by the software life cycles [12].

1.4. Failure Conditions

The failure condition category of a system is established by determining the severity of failure conditions on the aircraft and its occupants. An error in software may cause a fault that contributes to a failure condition. Thus the level of software integrity necessary for safe operation is directly related to the system failure conditions. The failure

condition category of a system is established by determining the severity of failure conditions on the aircraft and its occupants. An error in software may cause a fault that contributes to a failure condition. Thus the level of software integrity necessary for safe operation is directly related to the system failure conditions.

Failure conditions are categorized as catastrophic where failure conditions would prevent continued safe flight and landing. Hazardous/Severe-Major determines that failure conditions would reduce the capability of aircraft or the ability of the crew to manage adverse operating conditions. This would include a large reduction in safety margins or functional capabilities. This would also provide physical distress or higher workload on the crew that may result in an inaccurate or an incomplete performance. It would provide adverse effects on occupants including serious or possible fatal injuries. Major Conditions are failure conditions that cause significant reduction in safety margins or functional capabilities. There would be significant increase in crew workload or in conditions impairing crew efficiency, discomfort to occupants, and possibly include injuries. Minor Conditions are failure conditions which would not significantly reduce aircraft safety. This would involve crew actions that can be accommodated by slight increases to their workload and within acceptable bounds of efficiency. No effect are failure conditions that do not effect the operational capability of the aircraft or increases in workload.

1.5. Software Levels

Software levels are based upon the contribution of software to provide potential failure conditions as determined by the system safety assessment process. The software level implies that the level of effort required to show compliance with software requirements varies with its failure condition. Software Level definitions are as follows.

Level A will produce a condition in software whose behavior would cause or contribute to a failure of system function resulting in catastrophic failure condition of the aircraft; Level B will produce a condition in software whose behavior would cause or contribute to a failure of system function resulting in hazardous/severe-major failure condition of aircraft; Level C will produce a condition in software whose behavior would cause or contribute to a failure of system function resulting in a major failure condition; Level D will produce a condition in software whose behavior would cause or contribute to a failure of system function but with minor failure condition; and Level E will produce a condition in software whose behavior would cause or contribute to a failure of system function with no effect. Once software is confirmed as level E, no further guidelines of this document apply [12].

Airborne systems and equipment mandated by operating regulations and which do not affect airworthiness of the aircraft, for example an aircraft data recorder, the software must be appropriate for its intended function. This would mean that a data recorder would be required and functioning but is not a safety problem if it malfunctioned. This would fall under a legality that would cause monetary damages. If the behavior of a software component contributes to more than one failure condition, than the most severe condition is taken into account for the software level for that software component [12].

If the system safety assessment process determines that the system architecture prevents behavior of the software from contributing to the most severe failure conditions of the system, then the software level is determined by the most severe category of the remaining failure conditions to which the behavior can contribute. The system safety assessment process looks at the architectural design decisions to determine its effect on its software levels or software functionality. The following techniques provide several

architectural strategies that can limit the impact of these errors and to detect these types of errors [12].

1.6. Partitioning

Partitioning is a technique for providing isolation between functionally independent software components to isolate faults and potentially reduce the effort of the software verification process. If the software is protected by partitioning, the software level of each partitioned component may be determined using the most sever condition category rule of thumb. Guidance for designing partitioning protection would be Hardware resources, in controlling coupling, data coupling, and the failure modes of the hardware devices that associated with these protected mechanisms. Software Life Cycle processes should address the partitioning design considerations, including the scope of interactions permitted between the partitioned components [12].

1.7. Multiple Version and Dissimilar Software

Multiple version and dissimilar software is a system design technique in which more than one component of software that provide the same function in a way that may avoid some sources of common errors between components. Dissimilar software versions are used as a means of providing additional protection after the software objectives for the software level have been satisfied [12].

1.8. Safety Monitoring

Safety monitoring is a means of protecting against specific failure conditions by directly monitoring a function for failures which could contribute to the error condition. Through the use of monitoring techniques, the software level of the monitored function may be reduced to the level associated with loss of its related system function. To allow

this reduction there are important attributes of the monitor that should be determined: The software level which the monitor is assigned the most sever failure condition category for the monitored function. System Fault coverage ensures that the monitors design and implementation are such that the faults it detects can be detected under all necessary conditions. Independence of function and monitor ensures that the monitor and protective mechanism is not rendered inoperative by the same failure condition that causes the hazard.

1.9. System User Modifications

System user modifications are also considerations in the software development life cycle. It has been noted that potential effects of user modifications are determined by the system safety assessment process and are used to develop the software requirements, thus the software verification process activities. A change that effects the non-modifiable software, its protections, or the software boundaries is considered a software modification. A modifiable component is a part of the software that is intended to be changed by the user, and a non-modifiable component is one that which is not intended to be changed by the user. Some systems and equipment may include optional functions which may be selected by software programmed options. The option-selectable software functions are used to select a particular configuration within the target computer. The guidance that provides system considerations are user-modifiable software, system requirements should specify the mechanisms which prevent the user modification from affecting system safety while being at the same level as the function it is protecting from errors. If the system requirements do not provide provisions for user-modification, the software should not be modified by the user unless compliance has been implemented. The user should take all responsibility for all aspects of the user-modifiable software. Optional-selectable software,

when programmed opinions are included, whereby the means should be provided that inadvertent selections cannot be made. Commercial-off the shelf (COTS) satisfies objectives of the document. If deficiencies exist in the software life cycle data of COTS software, the data should be increased to satisfy the objectives of the document [12].

2. LITERATURE REVIEW

Several resources have researched the strict certification guidelines that are imposed in safety-related software [26], [27], [28], [33]. There also has been research on the efficiency of producing certified software [29], compatibility and improvement of the certification process [30], safety and hazards of safety-critical software [31], and the varying degrees of safety-criticality [32]. Certification provides and requires a very large need for documentation and traceability that is significantly contrasted to the little or no documentation agile methodologies employs [36], [46], as well as verification tools and code certification [34]. Incorporating an iterative approach for development of safety-critical software and safety-critical systems has been a subject of interest for many years [52].

One resource states that the research into suitability and applicability of Agile Methods for safety-critical software development is still at an early stage. It also states that there is yet to be a successful application of an Agile methodology to a safety-critical project as reported in the literature. It was also argued that an iterative and lightweight approach of an agile methodology can improve on the development of safety critical systems. What it does not do is argue the direct applicability of developing safety-critical systems that require certification. The paper focused on the effect of two specific activities in Agile Methods: lightening the up-front design of safety-critical system development and the iterative development of the software and its safety arguments [8]. Another resource states that all Agile practices can be mapped to a DO-178B software development process and that agility provides a best practices for safety-critical systems. They also propose that

Agile development does not require quick, encompassing change but can be accomplished by incorporating Agile methods into the safety-critical process. [37]

Varying certifications are required for safety-critical systems like aircraft, machinery, etc. to help prevent a failure from happening that could cause loss of life or damage to the environment. The following survey from people in the field also points out how certification can be adapted into the agile environment while keeping the same standards and a failure free software product.

3. SURVEY RESULTS

A survey was conducted as a Google Doc survey. I became familiar with LinkedIn groups that focused on aviation certified software. I asked various people in these groups, particularly people that work with DO-178B/C certification to freely answer a survey based on agile programming and certified aviation software development. The participants came from varying software companies and job descriptions around the world, all commonly working on development of certified aviation software. Out of 55 subjects that reported their job titles, there were 21 software Engineers, 3 System Engineers, 4 Company Owners or Company CEOs', 2 Upper Management, 7 Quality assurance personnel, 5 Designated Engineering Representatives or Certification representatives, 6 Project Managers, 3 Technical Managers, and 4 support personnel. The following answers are presented from each of the following multiple choices.

3.1. Develop Safety-Oriented Certified Software

From a sample of 54 respondents to this question, 47 (85%) responded with "Yes" and 7(15%) with "No". This evidence indicates that most of the respondents are working on projects that require some kind of safety-oriented task. Percentage of development that is dedicated to actual safety-oriented certified software is shown in Table 1.

Table 1. Safety-oriented Certified Development in 1 Year

Actual Development	% of people responded
10%	13%
20%	11%
30%	6%
40%	6%
50%	6%
60%	6%
70%	8%
80%	13%
90%	9%
100%	23%

This shows that more than 70% of the time at least half of the respondents, work on safety projects during a year lifecycle. It also notes that at least one-quarter of the group work less than one-quarter of the time on Safety Software. This information could show that they provide support services and are not required to work on actual software in a year's time. This may also show that some of the stakeholders may not actually work on any of the software, but provide support duties to these projects. These are some indicators that arose out of the answers to this question.

3.2. Safety-Oriented Certified Software Development

From the survey of 55 respondents; 34 (62%) responded with "aircraft software in use during flight"; 1 (2%) responded with "aircraft software not in use during flight"; 6 (11%) responded with "aircraft software used in conjunction with flight activity"; 3 (5%) responded with "software to use for support activities of an aircraft"; 2 (4%) responded with "non aircraft software but certified", 0 responded with "non aircraft software and non-certified", and 9 (16%) responded with "Other". Over half have worked on software that is in use during a flight schedule that would necessarily provide circumstances that provide a

dangerous situation if the software had failed. This is the audience that needs convincing of the possibility of instituting the certification guidelines into an agile environment.

3.3. Use of DO-178B

Of the 55 respondents, 50(91%) responded with “Yes” and 5(9%) with “No”. Most respondents use the DO-178B guidelines for certification requirements. This is almost a complete sample in the actual use of the guidelines for installing software in their daily lives. This is a data sample that provides a perfect basis on looking at specific certification needs.

3.4. Level of Software

25 (45%) responded with “Level A: Software whose behavior, could cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft”; 6 (11%) responded with “Level B: Software whose behavior, could cause or contribute to a failure of system function resulting in hazardous/severe-major failure condition for the aircraft”; 9 (16%) responded with “Level C: Software whose behavior, could cause or contribute to a failure of system function resulting in a major failure condition for the aircraft”; 7 (13%) responded with “Level D: Software whose behavior, could cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft”; 0 responded with “Level E: Software whose behavior, could cause or contribute to a failure of system function with no effect on aircraft operational capability or pilot workload”; 8 (15%) responded with “Other”.

Close to half of the respondents develop software in the highest degree of potential failure conditions. These are conditions that give a serious look into certification guidelines

that will be used in providing an insight into the ability of using agile in a highly restrictive environment.

3.5. Familiarity with Agile Methodology

The survey results for this question answered with 40 (73%) responded with Yes and 15 (27%) responded with No. Almost ¾ of the people surveyed were familiar with Agile to some degree. This provides an understanding of how agile can effect on how certification works in their development.

3.6. Success with Agility in Developing Safety-oriented Certified Software

The survey results for this question answered with 16 (31%) that responded with Yes and 36 (69%) responded with No. This is an area where I would like to look deeply into. If the Agile methodology is being used successfully, are they developing in a less safety strict environment? It was found that 6 out of 16 respondents that provide yes, actually develop in a Level A environment. This is significant in that 38% that have been successful with agile, developing in the strictest and most safety conscious environment. This will be discussed further.

3.7. DO-178B Change

Out of 50 respondents, 28 (56%) of them responded with “Yes” and 22 (44%) of them responded with “No”. This seems to be an area that needs a further study. Almost half of each group say that certification can be changed. To what degree and by what methodology do they prefer? The DO-178C has been able to accommodate new technology changes and different language groups since DO-178B was written in 1992.

3.8. Involvement in Software Planning

38 (75%) responded with “Yes” and 13(25%) responded “No”. A majority of respondents have provided a software plan in their development objectives. This helps provide a basis on the usability of planning out a best practices

3.9. Level of Safety-Oriented Software would Benefit from Agile

In the following survey, respondents were asked which level of safety-software would best benefit from the agile methodology. 11 (20%) responded with “Level A: Software whose behavior, could cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft”; 5 (9%) responded with “Level B: Software whose behavior, could cause or contribute to a failure of system function resulting in hazardous/severe-major failure condition for the aircraft”; 10 (18%) responded with “Level C: Software whose behavior, could cause or contribute to a failure of system function resulting in a major failure condition for the aircraft”; 9 (16%) responded with “Level D: Software whose behavior, could cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft”; 2 (4%) responded with “Level E: Software whose behavior, could cause or contribute to a failure of system function with no effect on aircraft operational capability or pilot work load”; 18 (33%) responded with “Other”. 1 out of 5 respondents believe that a highly restrictive and potentially risky software methodology could benefit from the use of an agile approach. This provides evidence that there is a need for a different methodology that can fit into a stricter certification environment.

3.10. Teams

26 (56%) responded with “Yes” and 22 (44%) responded with “No”. A greater percentage of respondents use teaming during the certified process. Though agile isn’t the only process to use teaming, it seems that teaming and group interaction may seem likely a serious possibility to pursue for integrating certified software into agile.

4. THE SURVEY OPINIONS

The following survey questions with open-ended answers provide several interesting views on how agile may affect safety-oriented certified software. These questions help formulate some of the further studies into what can become a better practice in the integration of certified software into agile methods.

4.1. Change in Software Requirements

Many of the answers to this question stressed that system level and high level requirements must be completely finished, agreed upon and signed off before anything else. These requirements also need to be testable. The software should be modular, reusable and utilize modeling tools such as UML. This will provide early validation and acceptance of its low level requirements that can help in producing repeatability and reproduction of the *artifact*.

Collaboration between different functional areas of the project and increased customer involvement has been stressed as well. System Engineers should take on the responsibility of engineering the requirements. They work with the system the most. It had also been said that outsourcing should be limited where translation errors tend to make things more costly in the long run. All collaboration should be kept in-house where problems can be resolved easily and worked over collaboratively.

The safety and dependability objectives shall be clear, well thought out and highly documented in order to minimize the risk of hazardous events. These objectives have been proposed as enforceable standards instead of guidelines so that safety and security cannot be sacrificed or changeable, which can make agile sometimes harder to implement.

Other answers have included that there should be better integration of the requirements into the scope or tasks to be performed within the project. It was also proposed to define a small set of requirements, then implement and verify the code in compliance to those requirements within the DO-178B, which would therefore provide significant efficiency improvements and reductions in the re-work, while maintaining safety. It was also proposed to write the lower level software requirements while testing. Then trace these tests back to the high level software requirements. These higher level requirements can meet the goals of agility as well as integrating a much tighter and safer program to meet the goals of the project. Define only the top-level system requirements, interfaces and requirements for each component. Implement a design of the safety critical assurance levels based on an architectural breakdown of critical points.

Problems that some have said is that a truly iterative incremental methodology is seldom used in a safety-oriented environment because of the heavy required use of documentation. It also mentioned that low level requirements and mapping to its design is tedious in this environment and that there seems to be no fusion of the High Level models to the Low level models.

4.2. Benefits and Disadvantages of Making Changes in Software Requirements

Benefits that have come up more than once in the answers were that changes to requirements will reduce development time and thus increase the development cycle and quality output. The source code is therefore verified and less translation errors would create fewer hazards, higher reliability, and less maintenance costs.

There were comments showing a difficult cultural shift for many customers and suppliers to consider new development processes because of the risk of failure in this

safety-oriented environment. They also found with the model-based approach, there needs to be competent people with the correct tools or they can actually cause more errors. It has also been stated that bulky software can produce unused code and a reuse flaw can be a costly problem in light of safety restrictions. There has to be thorough understanding of the scope or required tasks, and that there should be a well-qualified engineering staff to make this successful.

Other respondents feel that the team can focus on a small number of requirements related to its function, and be sure to implement them correctly in a relatively short amount of time. Agile should be modified significantly to incorporate all the reviews, tests and documentation as you develop the system. This should then produce maintainable system documentation. With more iterations, you could catch problems earlier. Velocity tracking has been found to be a more effective means of tracking overall progress. Define the common scope of the project and further define the second level requirements as you develop each component, piece by piece. Then, integrate the safety characteristics into the design. This would give more time on upfront analysis. Use newer technology to increase communication but do not overuse it. It would also help you write better requirements, making sure they are testable and perform these tests at an early stage. Streamline the requirements analysis process and reduce extraneous tests and documentation. Provide better alignment of delivered software with customer requirements. Benefits to this would be fewer hazards, higher reliability, less rework, more sustainable software. This would help you find out what works on the hardware much easier and actually see what is being built. This would therefore provide a good consistent product. You would therefore break down the parts easier while realizing the system better.

Other respondents have felt that certain factors can be difficult in requirements gathering. Regression testing for small changes in the code can be significantly costly to perform. That it is very hard to use agile in a fixed price situation for a project manager. One respondent also said that merging of high level and low level requirements removes some of the objectives needed in DO-178B such as traceability and coherency. One flaw in basic reusable requirements could be a costly one. Traceability can be easily lost, therefore unused code may be injected if streamlined requirements analysis and tests are produced. It has been emphasized that more time is used in the conceptual design phase (requirement analysis) and that this requires more paperwork. It was also stated that prototyping the project from the start may easily lose the overall development focus.

4.3. Benefits of Teaming in a Safety-oriented Certified Software Environment

The opinion that many respondents proposed is to use co-located, cross-functional and well-defined teams actively working toward the same goal. There should be the same knowledge available to all members. The merging of system and software engineers to define and verify requirements should be considered.

Other considerations include setting specific project roles that would build expertise for each person in the project. Develop staging check sheets that would ensure all certification issues, checklists, and forms are completed and verified at each stage. It was also proposed that teaming should be done not only during the design phase but also for the testing phase. Ensure that the stakeholders are aware of all the safety constraints. Small well-connected teams should be assigned for the entire development project. These teams should be kept together once the project is done because it helps in developing good relationships from people that work and know each other. This greatly helps get work done

much easier. Make sure there is highly qualified people in the certification process to eliminate inadvertent requirements-translation errors. Provide solid training and utilize best practices where it is needed. Provide a more efficient documented evidence trail for auditors and reviewers. This can include checklists to remember and coordinate with others.

4.4. Change Certification Process

Repeated answers from respondents have included an agile-based best practices for the certification process. A better development of the low level requirements process has been desired. Seamless integration of certification activities and its artifacts into the software development life cycle has also been desired. Others have stressed that small increments of a software product will be a problem with the workload of documentation. Model-Based Development has been thought of as a way of doing providing this.

Other responses include that some wished guidelines were standards so that each activity is clearly prescribed. The DER needs to be involved more in the development process as well as more involvement from the FAA. It was found that the means of compliance are not well understood by many development members. Making document generation more automated is something that respondents feel should be offered. Understanding more of the aviation business and good practical engineering sense when developing under the DO-178B should be good practice.

Some respondents felt that certification processes are FAA requirements about how software is documented and tested, and that development processes can be any form, this can conflict with the FAA regarding strict guidelines on how software is documented and

tested. Inconsistent expertise and the judgment of different FAA Aircraft personnel have been found that hinders the production of a better product.

4.5. Successful with Agile

Prototyping, and traceability tools, as well as best practices for least critical projects were commented more than once. Best practices were found to work on the least critical projects by testing the system safety throughout its project cycle. It was stated that Design and Coding stages can easily be agile during each applied iteration within the DO-178B Guidelines. Requirements in phases allows more flexibility in building software incrementally. The development of milestones into the project where each milestone is divided into a set of sprints were proposed. Early validation with customers and strong cooperation from the software verification team will help eliminate scope creep and keep the project on schedule. Break down the development cycle as much as possible into smaller pieces that can be accomplished into 1 cycle. It was proposed that tracing should be done at end of the cycle or beginning of next. Presentation of code separated from the underlying code while allowing rapid changes and eliminating the disturbance of the certified code has been proposed.

Many feel that a concern is that the certifiable life cycle data makes the process heavy and there tends to be the lack of design data by auditors. Complex software prevents the refactoring of your design at every step. Writing tests first does not necessarily help you discover what the system is supposed to do.

4.6. Agile Tools and Techniques

The following techniques and tools give an insight into many of the ways that safety-certified software can be integrated into agile. Unit testing was a tool that was highly

used as an answer to the question. Others have stated that Unit testing can be done to gain a better sense that the code is correct and can form a starting point for Verification and Validation. It also was said that Unit testing in a model-based scenario helps find errors early in the development phase and ensure robustness of the low-level requirements. Pair programming is believed by many to increase knowledge output and that its good when you are working out a difficult problem. Peer-reviews were also considered as an alternative to pair-programming. Continuous integration was considered very beneficial and allowed some organizations to respond to customer delivery changes easier. It cut the software build off by a few weeks and provided the product to customer earlier. It also was known to help ensure system risks were minimized. One subject stated that regulations do not allow for this technique and that since each software release must be certified, there is significant overhead to produce this.

Design Patterns were another tool that many have used. Design patterns can reduce intermediate development tasks and complexity. It does not necessarily help solve your problem if you don't understand the problem.

Test driven development was used highly as well. It is a good method in the sense of writing automated or manual test cases in parallel , but not if the requirements are not written. One respondent stated that requirements are the intentional view, and tests are the verification view. The two cannot be transformed from one to the other without loss. There are requirements that can never be fully tested. And in most cases, no set of tests completely specifies the intent of the system. It only allows us to know when we've broken previously written code.

Code factoring was used but some felt that there were diminishing returns at some point and from a practical standpoint of the regulation overhead being so great, that it was a costly process that would rarely be given its ability to work.

4.7. Comments

Respondents mentioned that there were no clear definitions for various decisions and the DER usually makes their own decisions as to what is acceptable. It was proposed that a "Survey of the Pro/Cons" for specific technique concepts of agile development needs to be developed, with a specific definition of each one. Solid documentation is still the best way to communicate the work products required for certification. Once the Systems Engineer can define the requirements accurately, the software programmer can usually implement the functionality. Some have also mentioned that the agile development process eliminates some of the check points and redundancies, making the development faster but at the cost of safety. It has been stressed that there is an ability to selectively pick positive facets of each methodology and mold each together would be something of consideration to the aviation community.

5. RESEARCH APPROACH

Software for airborne systems and their equipment gives the aviation community an acceptable level of confidence. It assures that the software being built, complies with airworthiness requirements set forth by the Federal Aviation Administration. As software in the field of safety becomes more complex and the technology to help alleviate these safety problems become more developed, there needs to be a foundation to help fix these concerns. The following are strategies that will be employed to integrate the safety-certified constraints into an agile approach.

As shown in the DO178B, the certifier has a goal. It is to produce software within its safety guidelines and conform to FAA restrictions. The certifier's objective is to produce the most error free product as possible. His stake in all this is to make sure it conforms to safety measures. The applicant's role or aptly named as "stakeholder" is to produce a safe product that keeps the software's focus. He in turn needs to maintain its livelihood well after the product is installed. Neither the certifier nor the stakeholder wants the software to fail. The stakeholder has monetary goals where he wants to produce a sustainable source of profit. If one failure happens, they both have lost their integrity and their intended goal. Both stakeholder and certifier needs to come to an agreement on how to produce a product that everyone can be pleased with [12].

In this thesis, I will provide is a process that will incorporate this certification process. Safety-secure software is very document heavy so there has to be traceability if a failure arises. This follows a close modified version to extreme programming and is used during the planning of a certified release.

Below [figure 2] shows the certification model (as follows) for the development process of certification that I envision. Each work item will have a corresponding chapter and provides details on how to integrate each of the sections.

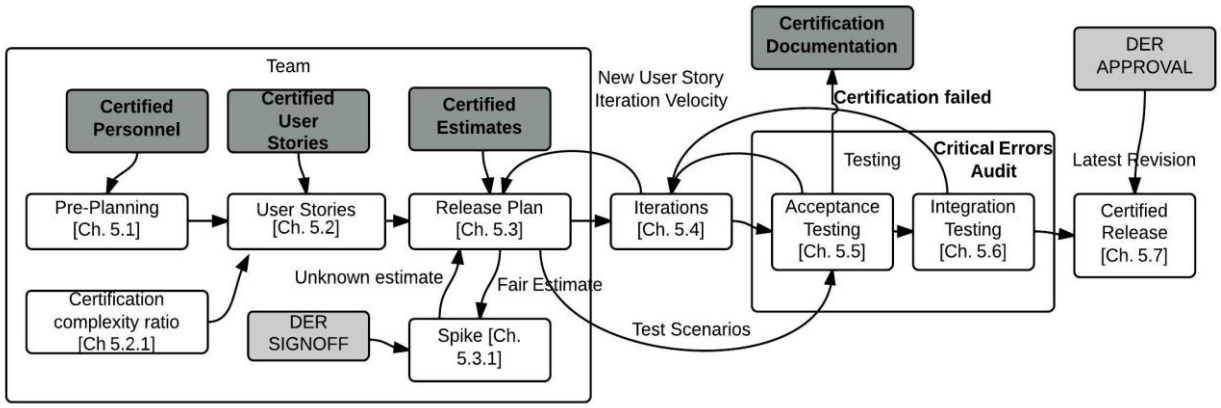

Figure 1. Certification Model

5.1. Pre-Planning and Initial Phase

The first stage would be the pre-planning and initial phase. The following will show steps on how to complete this.

Step 1: The first thing to start is to provide a preliminary idea. In aviation, this is the software that controls or works in conjunction with an aircraft. In this software example, it will be a new autopilot software module designed for a new aircraft that performs roll and pitch that is controlled by the aircraft fly-by-wire system. This idea or inception has to be approved and the expenditures available by the stakeholder.

Step 2: Who do I need to accomplish this? This is to be initiated by the Project manager or equivalent. The first set of people to start this process would be the "Lucky 7" or seven “key” personnel to the project. These seven ideal personnel would provide a

meaningful discussion and begin the “initial process”. Seven personnel would provide a significant voting decision in case of a tie if there needs to be a majority “vote”.

These seven personnel will come from a mixed group of people associated with the software project. These people will be involved in a standard sit down meeting to go over the feasibility of the project. This is called a "round-up" where the team is initiated and they are to be contacted *personally*. It’s easier to use technology to contact people but phone calls and email would not necessarily get as definitive answer as a meeting one on one. This would ensure that they are clearly advised and that they understand their duties. Let them know specifically what the "job" will encompass and that they will agree to it. Get a whole-hearted complete agreement that they will attend and that they provide the needed help in this endeavor. If all people are not 100% on board, this project has failed from the start. They have to agree to attend and provide constructive output.

People that should be involved are as follows: Two developers that will be working throughout the project. These people should be the main developers of the project. It should preferably contain one senior developer and one developer that has experience with the intended software. Include one head DER (designated engineer representative from FAA) that certifies or manages the certification process for the software and one other certifier that will be the main certifier for the software (preferably a DER). These certifiers will be included in all decisions in the software concept.

We need to include the head of the company or division, preferably the head overseer of the project. He will be considered the Stakeholder that will be paying for or determines a need for the software. He has to be the one to provide the development funds, not a representative. He needs to be an integral piece of these meetings. I would include one

highly knowledgeable pilot that has a basic understanding of software development. He would be considered the user and knows more about the use of the software and how he interacts with it. He is the one that will be at risk of failure andwill be in harm's way if a potential failure exists. The pilot may provide better ideas on how to reduce that risk. This gives the developers a "face" to see when they develop.

Last and most important, one head project manager should be included who will be the meeting leader and planner. This will most likely be the leader of the project that oversees the project to completion. This person can be the project manager for the software or a manager that has experience in the software process. This person will be making most of the calls to gain support for the project. All seven of these people are key players in the project and have an equal standing when it comes to getting the software off the ground.

The next step is to provide a convenient location to hold the "round-up" for the preliminary meeting and for subsequent meetings. These following best practices are preferred in following an agile plan. No telecommuting will be permitted except for feedback reasons. All preparation work is to be done together in close and comfortable quarters. Provide a room big enough to hold a group of people that are ready to work. All necessary equipment must be ready to use and in working order. A whiteboard for storyboarding is preferred. One that is easily cleaned and easy to see for all people. The use of electronic whiteboards or mobile applications are acceptable. Electronic whiteboards can be easily wiped and provide excellent ways in producing an electronic copy. Touch tablets will work for this, they are easily accessible and provide multitude of specific, easy applications that help present your thoughts and ideas. Refreshments (enough to make the meeting comfortable and help in providing productive decisions) are preferred. People are

more comfortable when coffee and light food is present. Prefer foods that provide sustainability during long periods of time. Foods with high protein help with brain use. Soft, comfortable chairs provides good posture for productive work.

Step 3: Provide a team atmosphere. If people do not know each other, introduce them. If there are personal differences, put them aside and develop for the advancement of the product. All people need to have a stake in this software to make it work. DERs need to be involved and provide constructive assistance. Understand each other's role, and provide a good work setting for all involved. The User (as in this instance) will be a pilot that will be doing his job with the benefit of the software installed into his plane or others like it. Provide this person with an understanding of what they will do. Provide a clear cut and "simple" plan when it comes to the first of the requirement stages. Everyone is equal when sitting down to provide ideas. Everyone has a place in this development.

Step 4: Develop plans that can be decided on and will work. Bring all your best people together into the Initial Planning Phase and hash out these ideas together until finished. Include all certification personnel that are relevant to the production of this software. Make sure they are included and that they come to this initial planning meeting.

Make use of "time-boxed" sessions. Work as long as you are useful. Take 20 minute breaks every 90 minutes. Finish certain goals every 90 minutes. Use these breaks to clear your thoughts. If the work is not going anywhere, stop and reconvene the following day with a fresh start after a "time-box is finished. Useful work is everything. Make sure that all needed personnel are present or a knowledgeable person can be used in place of the absent team member. Keep a "backup" person available to convene when one member cannot be present. They must have the same knowledge and understanding of the problem.

They need to know exactly what their colleagues are working on and what has been accomplished. This would mean that replacements for current personnel should be considered an easy replacement. These replacement personnel will be informed and have the knowledge to pick up where the other member left off. Any meeting information should be relayed to the backup person. The teams should be transparent when sitting down to finalize the requirements.

Certification needs can be achieved by these following steps. The certifier is detrimental in this stage. With research of these agile practices, I have found that the area of certification in agile methods can be expanded further and produce better results to maintain safety-oriented standards.

5.2 Utilizing the User Stories

Setup user stories and pinpoint where certification is to be included. The certifier needs to be a major force in the work of designing. Make pin points to where these certification points are needed. Grade these pinpoints by color such as severe risk, high risk, medium risk, low risk (Red, Yellow, Green, or White) or any combination that seems fitting to the organization. Refactor these pinpoints to provide the best outcome and easiest route to get reliable and safe software with all required certifications included. These stories [14] should have a time attached as to what is required to complete one story. The stories are to be broken into tasks. This will force you into finding how to approach that particular story. This will show you how to revise before making a new step.

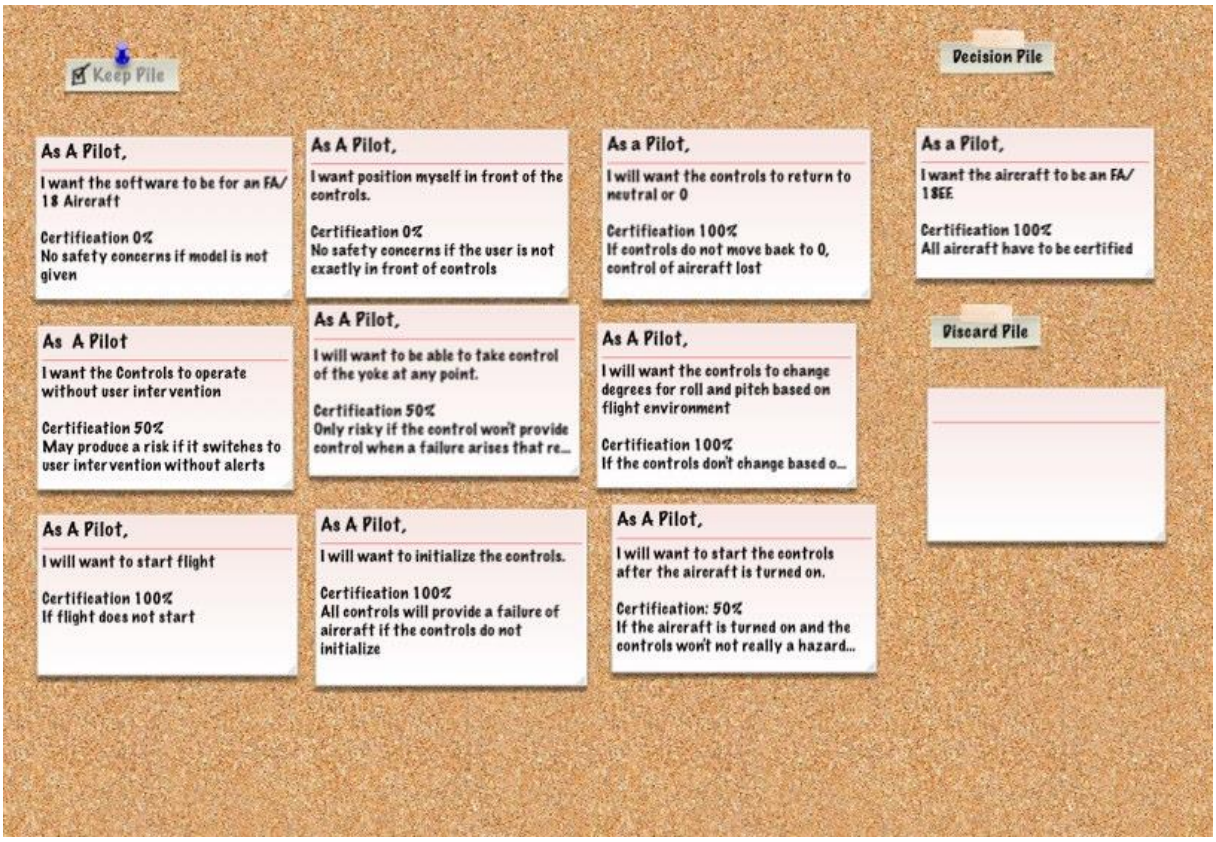

Figure 2. User Storyboard using Corkulous (Ipad)

Aircraft Yoke on fly-by-wire system:

1. Pilot will utilize FA/18 aircraft and position himself at controls.
2. Pilot will start specific controls in use of aircraft.
3. Controls will initialize for FA/18 aircraft.
4. Pilot will start flight.
5. Set degree change for roll and pitch based on flight environment
6. Level out and bring controls back to 0.

Figure 3. Example User Story

How to develop the aircraft yoke using user stories:

- Write user stories in "As a <type of user>, I want <some goal> so that <some reason>" [6]. Check to see if user story can be incorporated with certification.
- Place check marks for certification.

- Put all user stories contributed from team members onto whiteboard.
- Include user roles to these stories on the storyboard.
- Throw out any user story that does not fit into any of the requirements. If one fits but MAY be utilized at some point, place it in a "Decision pile".
- When all Stories are presented, rate the decision pile and put up the ones that work the best.
- Incorporate certification into these stories. Use a whiteboard to "stick" these stories. Place the "pins" where they would fit the best or are required to support the level of safety that would give the best advantage in safety and development time. If color pins can't be used, just rate level of certification severity. Provide a conservative percentage on how much certification would be required for each story.
- Satisfy the percentage and reason for the certification percentage into the story.
- Average each color or severity level and determine the pin (level's) group's certification percentage and get a total percentage of how much certification can be determined for each grouping. Determine cost by each of these groupings, dependent by the cost of certification.
- If a user story has a non-safety certified requirement, see if this part needs to be certified. If safety is a factor, determine the severity of this story, and work around it. Highlight the best avenue, ensuring the utmost safety. Utilize the complexity ratio in Chapter 5.2.1
- Break each story down to the smallest chunk that can be written out.
- Story boards need to be "feasible" or "doable". If they are not, the user story must be eliminated or be formatted into an understandable story.

- Place user stories with their corresponding certification pins in a work pile.
- Can these be refactored to not include certified pins? Can the pins be set as white or determined as no safety factor needed? Can these stories be eliminated if needed? Provide a Time to accomplish each story. Each story point should approximate to 3 weeks.
- Check previous story points based on equivalent stories depending on the current personnel's skill. Make sure it is as accurate as possible.
- If no previous estimates are used, use a spike or what is considered quick exploration of the issue.
- Split a story if it can be split into smaller stories. This should be done if a story is extremely large and difficult to estimate. Make sure the new stories cover the original completely.
- From these stories, we should provide steps that will therefore be put into low level requirements.

5.2.1 Certification Complexity Ratio

This complexity ratio gives a basis to verify if the story project is complex or not. It is not as complex if you have more resources such as people or the lower safety level and will lower based on this. It will provide how many certification personnel to be used in certifying software. This will help in determining how many certified personnel should be available during the story point inclusion to make an agile process easily adaptable. The more severity level of certification the more certifiers should be available to use in certifying the software.

Severity Level Multiplier:

Level A- 4 most severe
Level B- 3 medium severity
Level C- 2 low severity
Level D –1 non severe
story points = p
severity level= l
certified representatives = r
complexity ratio = c

$$\frac{\left(\frac{\mathrm{p}}{100}\right)\mathrm{l}}{\mathrm{r}} = c$$

Certification Complexity ratio

Examples:
2 Story Points being used
Level A software project (most severe)
3 full time certifiers
Complexity ratio: (2/100)(4) /3= 0.03

With only 2 full time certifiers
Complexity ratio (2/100)(4) /2 = 0.04

With Level A to B
Complexity ratio (2/100)(3) /3= 0.02

In the example, subtracting 1 certification representative which provides full duties to the user story, the complexity will increase to 1/100. You will not want to limit the amount of certifiers at that "complexity". If you can reduce the certification level to a Level B, the story point load will lower the complexity to 2/100. As from this exercise there seems that with more points or safety severity, it will produce a much higher ratio that could impact the project.

5.3. Release Plan

Move each certification user's story to its own iteration of 1 to 3 weeks based on its severity level. Use the certification complexity ratio. These are "time-boxed events" or

"spikes". Make sure that the user story is checked off with a certification team at this time. Place each certification story in an iteration with the same level of severity. Attach each of these stories where they fit and connect. Base how many stories to be utilized based on how many programmers are available to do each story point. Each iteration should end with some prototype or product with certification levels set.

Below shows the iteration levels as they should be prioritized by the type of certification level. This model is based on Ambler's prioritized model as shown below [3]. It shows exactly how to add work items based on the amount of personnel needed and how critical the software can be. The more certified personnel should be provided for the higher criticality, thus the iterations should include the priority of the highest critical work item. These can be added or removed from the stack at any time, and stories are re-sorted to provide for these exceptions.

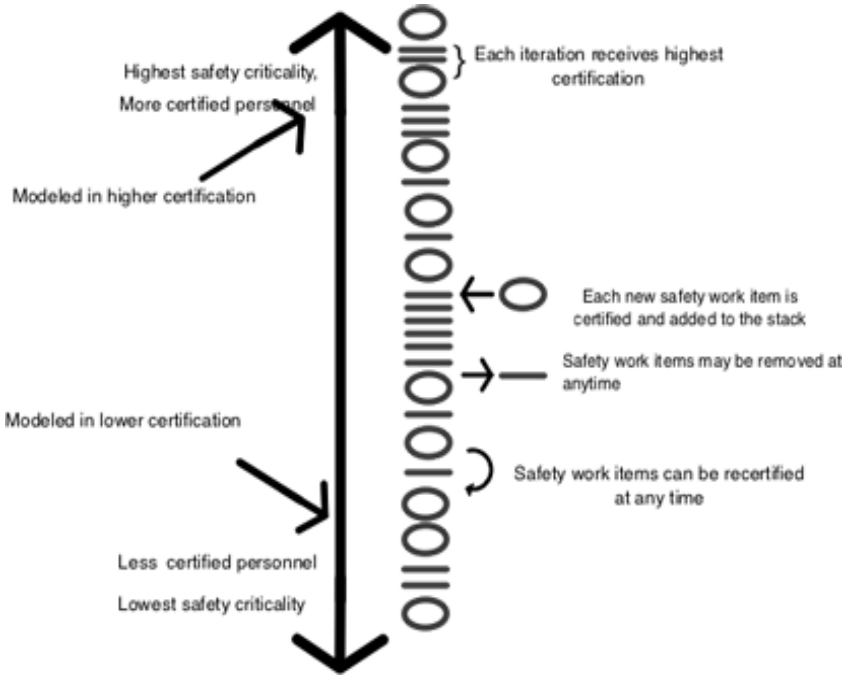

Figure 4. Managing Work as a Certified Stack.

5.4 Iterations

The Design and Coding stages are produced through 3 week iterations of stories. How many iterations is dependent on the story load. These iterations must deliver running, tested, integrated software. That means the certification has to be integrated and modified to fit into the software process. The software also needs to adhere to strict testing before it gets applied to the shelf or added as a usable iteration of the software. Life Cycle data or requirements can be produced from the acceptance tests. Certification has to be adapted while being dependent on the safety reliability and by what software safety ratio it produces. Place a risk factor based on (Severity- 4 for high severity, 3 for medium severity, 2 for low, 1 for non-severe). Include certified personnel at specified work times when certified user stories will be iterated.

Iterate all Stories that have a safety work level that are equal or lower than the highest safety severity. Each certified iteration checks the story of highest magnitude. Its narrows down each certified iteration down to one iteration cycle. It may take longer to satisfy conditions for certification than non-certified stories. Remain within the 3 week cycle period to keep things running smoothly. If more certified hands are to be used, then account for them. Do not wait to finish other stories. Proceed with the other stories that need work. Have them accepted before the certification is finished. Finish an iteration while the certification iteration is waiting. Keep focus on the level of safety used in the certification stories. Higher levels get more emphasis than the other stories. If stories can be broken down into smaller non-certified pieces, do it. Keep the severity at the lowest denominator.

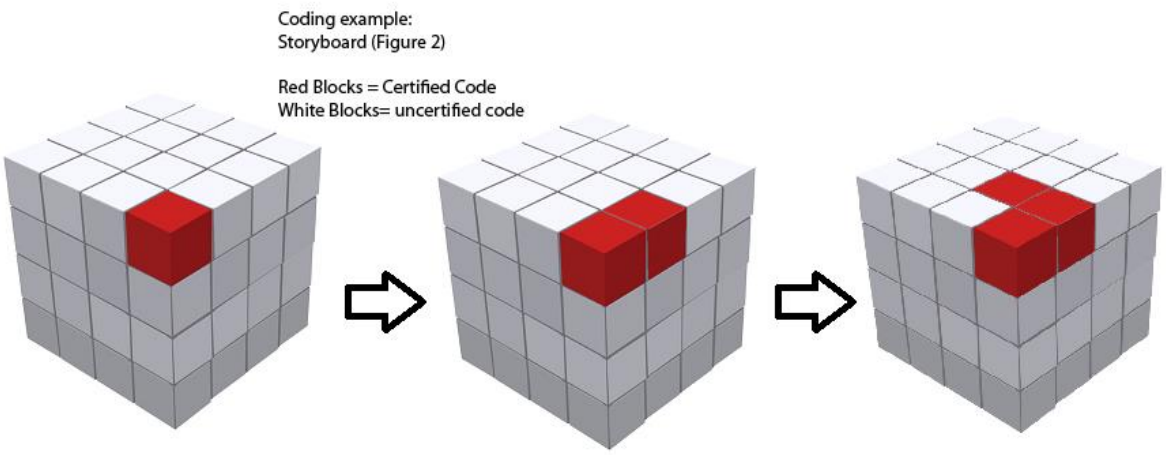

Figure 5. Certified Code Blocking

5.5. Acceptance Tests

Acceptance tests should be run concurrently using each story. Test around boundaries. Try several numbers (above, below, on or close to boundary number) while testing all conditions. Story card using title test with a description of the test. Separate test cards to utilize for certification testing. Utilize tests that are critical for all certification stories. Develop stories with constraints. Run acceptance tests at regular intervals that are expected to be implemented at the end of the iteration. Make a special graph of certification acceptance testing. Verify that all tests are run. Make aware to all team members which tests failed. Dependent on the risk evaluation of the test, will then determine if certification will continue. Put a special section on the cork board for constraints. QA is a needed part of these tests. Include Certification personnel into the QA team when certification of user stories are needed.

5.6. Integration Testing

Integrate all user story iterations into the complete release with the inclusion of certification. Dedicate two embedded testers (one developer and one certification personnel

(DER)) to oversee the testing of the integrated software and then hand it over to the stakeholder as completed.

5.7. Initial Release and Post-Mortem

Once the release is given to the customer, gather constructive feedback from all people involved in the project. Rate these feedback responses. Keep them for the next release plan. With this feedback, provide a Do's and Don'ts after the first release. This will be used in pre-planning before the next process. Add certification concerns in the feedback from certification personnel. Put this altogether to provide a baseline for future work.

5.8. Moving Certification into Agile Practices

Certification can be moved into agile with gradual steps. The consensus is that the avionics certification culture is hard to change. As Chenu [22] explained, "This is why we have a practice of XP tailored to our industry. The values and principles of XP remain perfectly compatible with safety issues. However some practices need to be adapted."

The following areas of development can be gradually moved into agile development. Requirements are a dilemma in certification because of the size and amount. Chenu [22] notes, "Requirements and traceability are an issue, so we've added the continuous traceability practice. As we work, we record the traceability links. Full traceability is continuously and automatically consolidated, quantified, checked and finally published." This helps in the safety analysis process. One respondent of the survey replied, "Adopt a very thin requirements hierarchy, concentrating on testable low-level requirements and eliminating intermediate layers…that capturing requirements in specification tools like display design tools, bus design tools, etc…reviewing and verifying them within these tools". One other respondent said that "repeatability is important since

once the design is approved, reproducing the exact artifact is important." Chenu [22] also notes, "Documentation must be considered as part of the product. Therefore, the documents must be incrementally written to be potentially shippable after each iteration."
Teaming has already been found to help in several instances during safety-oriented development and some respondents say they are used in the conventional development of certification software. Experienced Developers and the gradual lack of motivation to work on large, long-lasting projects, seems to have its share of problems [22]. Extreme Programming helps because it enforces training while you work with multidisciplinary teams and working in pairs that seem to increase the likelihood of experience gain [22]. Teamwork, team spirit and self-organizing teams help to ensure the human element [22]. Teaming can be incorporated anytime into the certification process with less restrictive development processes first to help plan a more seamless inclusion and that if failure exists will not pose an immediate danger.

Nowhere is there evidence that shows in the DO-178B, that there is a "preferred" method of development within the guidelines of the DO-178B. This was proposed by several responses in the survey. With cost being shown that it can be sometimes 10 times the cost of developing under DO178B than a regular software project, it could help provide a cost savings that would make this beneficial. It has been discussed, that the required documentation for certification can hinder the use of agile. It has been mentioned that checklists and development tools can help the requirements gathering process. It can make it much easier to implement other areas of agile such as iterations and testing. K. Nils mentioned, "DO-178B guidelines do not allow individual software components to be

certified. Only the complete system, which is typically integrated from many independently developed components, can be safety certified" [10].

Test Driven Development is another section of development that needs focus and to slowly integrate it into the certification process. Write code only after a failed unit test for that code. The tests always come first and it fails either by code not being written or does not contain the logic to successfully work [25]. Vuori [23] says "that tests will remain in place during implementation and they will remain as a safety net for refactoring. The development of a feature will not be considered "ready" until all the tests pass."

6. EVALUATION AND DISCUSSION

Certified software has the ability to utilize several aspects of Agile where it can be accepting of changes to the software and the use of newer techniques in getting the software delivered quickly. It incorporates the agile ideal where certifiers are ingrained into the full software process. They become more of a team than thought previously. They have a more hand in how things are done and have a better look at the software throughout the process.

The problems that can arise from utilizing the certifiers on a more transparent basis is that the resources may not be budgeted for that year. It was found through one study that certification under the DO-178B standard adds at least 25 - 40 percent to total project costs, and it can climb as high as 75 - 150 percent [50]. This alone would increase exponentially if we added more certifiers to a project or take certifiers away from other work. The increased resources would most likely double when utilizing a certifier full-time to the project. This can cost up to 3 times the cost of an uncertified project. On the other hand, these resources can be recouped from a shorter production timeframe in producing a working deliverable and the changes that can be instituted at any time during the software cycle. Because of this, the project will be shorter and it will provide the ability for the certifier to work on the next project and dedicate that time to it. The amount of projects completed by the certifier will be much higher in a year's time. This would increase the certifier's productivity. The benefits producing safety-critical certified software may come from a timely software position in a market saturated with software duplication. Software changes that come from new advances in the field can be added at various times to the

software release being developed. In several ways, these agile changes can provide benefits that can outweigh the negative aspects.

Several respondents had felt that agile should be merged into the certification process of safety-oriented development instead of the traditional waterfall methods. There has been a lot of developers from the survey that worked in some capacity in all risk levels and utilized many of the agile methods in the certification arena of safety-oriented development. It was also stressed that the DO-178B was not designed to mitigate one developmental methodology over the other. It stressed guidelines on how to certify these highly risky software programs. A lot of the feelings from the survey for agile development, came from the cost of time delays in a competitive workplace. They also felt that software program doesn't have to freeze at every step along the way to completion before accomplishing anything else. Many others, felt that communication was a key part of accomplishing a software project either by groups or by individual interaction. Many felt that the communication in the waterfall methods of certification was sometimes lacking.

Other aspects of Agile that didn't seem to fair well with respondents were the lack of documentation that certification surely needs for verifying its accurateness. I found in the survey that it is not the lack of documentation that is the key issue that agile seems to eschew, but the efficient use of it. Traditional standards tend to keep documentation around just because it was written somewhere that it was required. The main focus of agile is to streamline the use of documentation but not completely ignore it. I found that many respondents agree with the use of tools or specific guidelines in the retention of these artifacts. Once the artifacts are no longer used, they are discarded for new updated

versions. Documentation tools or automation can complete this removal of documents and artefacts that may pile up during the developmental process.

Therefore, I have provided a look into new ways of making certified software more agile by adding certifiers at key times during the process. I have added some respondents advice in how agile can work in this environment. They have given much insight on what works with agile and what will need to be modified for it to work.

7. SUMMARY OR CONCLUSIONS

The goals that were achieved during this study was to determine if certification can be inducted into agile development. As provided from my insight into the problem, certification for software will likely need more certification experts on a full-time basis and need a hands on approach. This involvement will surely provide more support overhead but will ultimately produce shorter time in producing a software product. One source has said, "such practices applied on a large scale DO178B level-A project have enabled us to measure encouraging results. The cycle-time has been reduced from one year to 20 days. Nine incremental versions have been delivered on schedule to the customer in a 3-year time-frame". Other sources have seen similar reductions in time from the conception to each incremental version.

I see the merger to be a success in several areas in the development process. The main issue that can be derived from fully integrating the agile process is in the use of documentation. Documentation is very heavy in safety-oriented software and is very much needed for its "checks and balances" when it comes to verifying the failure along the development process that could result in detrimental catastrophe. Where I see the change is in how documentation is used and collected. This is an area that I feel would help benefit the safety development process in finding new ways of making documentation fit more into an agile philosophy. I believe there can be more work to be done in that area that can help progress the time to develop safety-oriented software or cutting down the redundancy in certification.

DEFINITIONS

Aircraft fly-by-wire system: A fly-by-wire (FBW) system replaces the manual flight control of an aircraft with an electronic interface. The movements of the flight controls are converted into electronic signals transmitted by wires. Flight control computers determine how actuators should move at each control surface to provide the expected response. Commands from the computers also work without the pilot's knowledge in stabilizing the aircraft and in performing other tasks [7].

Applicability: Tests designed in support of a team that while delivering software from a business perspective ensures that the right product is built.

Artifact: One of many kinds of tangible products produced during the development of software such as use cases, class diagrams, UML models, requirements and design documents

DER (Designated Engineering Representative): an individual, appointed in accordance with 14 CFR § 183.29 or job specification of Designated engineering representatives [20], who holds an engineering degree or equivalent, possesses technical knowledge and experience, and meets the qualification requirements of order 8100.8D : Designee Management Handbook [19]. DER may be appointed to act as a Company DER and/or Consultant DER. Company DERs can act as DER for their employer and may only approve, or recommend approval, of technical data to the FAA for the company. Consultant DERs are individuals appointed to act as an independent (self-employed) DER to approve or recommend approval of technical data to the FAA [12].

Feasibility: An analysis and evaluation of a proposed project to determine if it is technically feasible, within estimated cost, and profitable. Feasibility studies are almost

always conducted where large software projects are at stake. Also called a feasibility analysis.

High Level Requirements (HLR): High-level requirements are produced directly through analysis of system requirements and system architecture. Usually, these high-level requirements are further developed during the design process, thus producing one or more successive, lower levels of requirements. [12].

Low Level Requirements (LLR): Low-level requirements are software requirements from which source code can be directly implemented without further information [12].

Traceability: Traceability is about understanding how high-level requirements -- objectives, goals, aims, aspirations, expectations, needs -- are transformed into low-level requirements. It is therefore primarily concerned with the relationships between layers of information. It is also used to follow the life of a requirement, in both forwards and backwards directions starting at its origins, through the development and specification process, to its deployment and use [9]

Test Driven Development: Test Driven Development (TDD) is a type of practice, where test are developed for a feature or other task before there is a design or implementation [23]

Validation: Involves checking that the program as implemented meets the expectations of the customer [2].

Velocity Tracking: A measurement of the amount of selected story cards as long as the estimated developer weeks on the cards add up to the number of available developer weeks for an iteration [17].

Verification: Involves checking that the program conforms to its specification without execution, This effort may be either informal (using a checklist) or formal (by using a proof) [2].

REFERENCES

[1] Agile Alliance, *The Twelve Principles of Agile Software* [Online]. Available: http://www.agilealliance.org/the-alliance/the-agile-manifesto/the-twelve-principles-of-agile-software/

[2] S. Ambler. (2007) *Effective Practices for Modeling and Documentation* [Online] http://www.agilemodeling.com/essays/agileDocumentationBestPractices.htm

[3] R. Binder "Testing Object-oriented Systems: Models, Patterns, and tools" Addison-Wesley (2000).

[4] K. Beck and et al. "Manifesto for agile software development" 2001.

[5] J.J. Chilenski and S.P. Miller. (1994), "Applicability of modified condition decision coverage to software testing," *Software Engineering Journal*, pp.193-200, September 1994

[6] M. Cohn. (2008, April 25). *Advantages of the "As a user, I want" user story template.* [Online] http://www.mountaingoatsoftware.com/blog/advantages-of-the-as-a-user-i-want-user-story-template.

[7] D. Crane. "Dictionary of Aeronautical Terms, third edition". Aviation Supplies & Academics, 1997. ISBN 1-56027-287-2

[8] Xiaocheng Ge et al., "An Iterative Approach for Development of Safety-Critical Software and Safety Arguments," in 2010 Agile Conference, 2010, pp. 35-43.

[9] O.C.Z Gotel and A.C.W. Finkelson. *An analysis of the requirements traceability problem*, in Proceedings of ICRE94, 1st International Conference on Requirements Engineering, Colorado Springs, Co, IEEE CS Press, 1994

[10] K. Nilsen. (June 2004).*Certification Requirements for Safety-Critical Software | RTC Magazine* [Online]. Available: http://www.rtcmagazine.com/articles/view/100010. [Accessed: 08-Feb-2012].

[11] G. Rose. (2003). *Safety-Critical software* [Online]. Available: http://www.lynuxworks.com/products/whitepapers/safety-critical.php3 [Accessed: 08-Feb-2012]

[12] RTCA. *RTCA: Software Considerations in Airbone Systems and Equipment Certification.* Radio Technical Commission for Aeronautics (RTCA), Washington, D.C. Standard Document no. DO-178B, December 1992

[13] Unknown, "What is DO-178B?" http://www.lynuxworks.com/solutions/milaero/do-178b.php3 (Accessed: May 15th 2013)

[14] K. Waters and J. Woods. (2012, August 12). *How Detailed Should Tasks Be in a User Story?* [Online] Available: http://www.allaboutagile.com/how-detailed-should-tasks-be-in-a-user-story/

[15] D. Wells. (2009). *Extreme Programming: A gentle introduction* [Online]. Available: http://www.extremeprogramming.org/. [Accessed: 09-Feb-2012].

[16] A. Wils, et al. "Agility in the Avionics Software World"XP2006, Oulu, Finland, Springer

[17] M. Fowler. (January 2001) *Planning and Running an XP Iteration* [Online]. Available: http://www.martinfowler.com/articles/planningXpIteration.html

[18] D. McCoy. (December 2011). *Startup software company aims to trim time from FAA certification process* [Online]. Wichita Business Journal

Available: http://www.bizjournals.com/wichita/print-edition/2011/12/09/startup-software-company-aims-to-trim.html

[19] Federal Aviation Administration. (October 28 2011). *Order 8100.8D: Designee Management Handbook.* Washington, D.C.: U.S. Department of Transportation.

[20] U.S. Government Printing Office. (January 4, 2004) *14 CFR - Code of Federal Regulations - Title 14: Aeronautics and Space : 14 CFR § 183.29* Available: http://www.gpo.gov/fdsys/pkg/CFR-2011-title14-vol3/pdf/CFR-2011-title14-vol3-sec183-29.pdf

[21] A. Sidky and J. Arthur. "Determining the Applicability of Agile Practices to Mission and Life-Critical Systems," Software Engineering Workshop, 2007. SEW 2007. 31st IEEE , vol., no., pp.3,12, March 6 2007-Feb. 8 2007 doi: 10.1109/SEW.2007.61

[22] E. Chenu. (2009) "Agility and Lean for Avionics". Available At: http://manu40k.free.fr/AgilityAndLeanForAvionics1.pdf

[23] M. Vuori. (June 2011). "Agile Development of Safety-Critical Software". *Laitosraportti - TUT Publication series*. pp 49-50, June 2011.
Available at: http://dspace.cc.tut.fi/dpub/handle/123456789/20554

[24] A. Peterson. (April 2007). "Design in Test-Driven Development".
Available at: http://www.adampetersen.se/articles/designintdd.htm

[25] V. Subramaniam and A. Hunt. "Practices of an Agile Developer"
Raleigh, North Carolina and Dallas Texas: The Pragmatic Bookshelf (December 2006)

[26] A. Wassyng et al. "Software Certification Consortium: Certification Methods for Safety-Critical Software"

[27] P. Steele. “Certrification-Based Development of Critical Systems” ICSE 2012, Zurich, Switzerland. Doctoral Symposium

[28] S. Nelson. “Certification Processes for Safety-Critical and Mission-Critical Aerospace Software”. NASA/CR–2003-212806

[29] R. Wolfig, “Parameters for Efficient Software Certification” Vienna University of Technology, Real-Time Systems Group.

[30] E. Kesseler. “Integrating air transport elicits the need to harmonise software certification while maintaining safety and achieving security” Aerospace Science and Technology, (8) 2004 page 347 – 358. (c) Elsevier SAS 2004.

[31] J. Jacky. “Safety-Critical Computing: Hazards, Practices, Standards, and Regulation”

[32] P. Amey. “Smart Certification Of Mixed Criticality Systems”. Praxis High Integrity Systems, 20 Manvers St., Bath BA1 1PX, UK

[33] A. Kornecki and and J. Zalewski. “Software Certification for Safety-Critical Systems: A Status Report”

[34] E. Denney and S. Trac “A Software Safety Certification Tool for Automatically Generated Guidance, Navigation and Control Code”

[35] B. Gallina et al. "Towards a Safety-Oriented Process Line for Enabling Reuse in Safety Critical Systems Development and Certification," *Software Engineering Workshop (SEW), 2012 35th Annual IEEE* , vol., no., pp.148,157, 12-13 Oct. 2012

[36] R. Paige et al. 2011. “High-integrity agile processes for the development of safety critical software.” *Int. J. Crit. Comput.-Based Syst.* 2, 2 (July 2011), 181-216.

[37] S. H. VanderLeest and A. Buter "Escape the waterfall: Agile for aerospace," *Digital Avionics Systems Conference, 2009. DASC '09. IEEE/AIAA 28th* , vol., no., pp.6.D.3-1,6.D.3-16, 23-29 Oct. 2009

[38] A. Soderberg and B. Vedder. "Composable Safety-Critical Systems Based on Pre-certified Software Components," *Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on* , vol., no., pp.343,348, 27-30 Nov. 2012

[39] F. Zeng et al. "Software Safety Certification Framework Based on Safety Case," *Computer Science & Service System (CSSS), 2012 International Conference on* , vol., no., pp.566,569, 11-13 Aug. 2012

[40] Preschern, C.; Dietrich, K., "Structuring Modular Safety Software Certification by Using Common Criteria Concepts," Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on , vol., no., pp.47,50, 5-8 Sept. 2012

[41] J. Hill and S. Tilley. "Creating Safety Requirements Traceability for Assuring and Recertifying Legacy Safety-Critical Systems," *Requirements Engineering Conference (RE), 2010 18th IEEE International*, vol., no., pp.297, 302, Sept. 27 2010-Oct. 1 2010

[42] Y. Jeppa. "Flight Control Software: Mistakes Made and Lessons Learned," *Software, IEEE*, vol.30, no.3, pp.67, 72, May-June 2013

[43] C. Webster et al. "Delivering software into NASA's Mission Control Center using agile development techniques," *Aerospace Conference, 2012 IEEE* , vol., no., pp.1,7, 3-10 March 2012

[44] S. Wolff. "Scrum goes formal: Agile methods for safety-critical systems," 2012 Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA), vol., no., pp.23,29, 2-2 June 2012

[45] R.A. Chisholm and C. Royal Military College of (2007). “Agile Software Development Methods and DO-178B Certification”, Royal Military College of Canada (Canada).

[46] M. Peraldi-Frati and A. Albinet “Requirement traceability in safety-critical systems” CARS '2010, April 27, Valencia, Spain

[47] K. Schwaber and J. Sutherland. 2013. “The Scrum Guide, The Definitive Guide to Scrum: The Rules of the Game” https://www.scrum.org/Scrum-Guides

[48] B. Douglass. 2013. “Agile analysis practices for safety-critical software development” IBM Developer Works. http://www.ibm.com/developerworks/rational/library/agile-analysis-practices-safety-critical-development/index.html

[49] B. Douglass and L. Ekas. Oct. 2012. “Adopting agile methods for safety-critical systems development” IBM Developer Works. http://public.dhe.ibm.com/common/ssi/ecm/en/raw14313usen/RAW14313USEN.PDF

[50] V. Hildeman. “DO-178B Costs versus Benefits.” Atego HighRely. http://highrely.com/whitepapers.php

[51] E. Chenu.“Agile & Lean software development for avionic software” Thales Avionics. http://www.erts2012.org/Site/0P2RUC89/7A-4.pdf

[52] T. Stalhane et al. “The application of Safe Scrum to IEC 61508 certifiable software”. ESRL 2012, Helsinki, Finland.

APPENDIX. SURVEY QUESTIONS

Safety-Oriented Certified Software Development requires many guidelines to ensure the safety of the public and property in case of failure. Agile development is a collaborative and adaptive form of software development. Can we take agile development and integrate it into a safety-oriented certified development without sacrificing any of the safety and failure rates? PLEASE ANSWER ALL QUESTIONS HONESTLY AND PROFESSIONALLY

What company do you work for?

How many people are approximately employed in the company you work for?

What is your company's line of work?

What is your present position?

Do you develop any safety-oriented certified software?

- Yes
- No

What safety-oriented certified software do you develop?

- aircraft software in use during flight
- aircraft software not in use during flight
- aircraft software used in conjunction with flight activity
- software to use for support activities of an aircraft
- non aircraft software but certified
- non aircraft software and non-certifed
- Other:

How much of your development is dedicated to actual safety-oriented certified development in a single year? Percentage %

Do you use the DO-178B for any of your certification guidelines?

- Yes

- No

What Level would you classify your software if potential failure conditions would present itself? The Failure Condition Categorization defines the failure conditions in the DO178B (Page 7)

- Level A: Software whose behavior, could cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft.
- Level B: Software whose behavior, could cause or contribute to a failure of system function resulting in hazardous/severe-major failure condition for the aircraft.
- Level C: Software whose behavior, could cause or contribute to a failure of system function resulting in a major failure condition for the aircraft.
- Level D: Software whose behavior, could cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft.
- Level E: Software whose behavior, could cause or contribute to a failure of system function with no effect on aircraft operational capability or pilot workload.
- Other:

Of the software life cycle process, which part of development are you active in (Requirements, Design, Coding, Integration, or other (explain)) ?

If you could change any of the certification process, what would it be? What Development method would you use? EXPLAIN

Agile Development and Safety-Oriented Certified Development

Agile Software Development is a software development methodology that tackles the dynamic and ever changing environment. The Agile Manifesto was written as a specific need for an alternative to documentation driven, heavy-weighted software development processes. The inner workings of the manifesto prefers certain principals but are not exclusive. The manifesto claims that individuals and interactions are preferred over processes and tools. Working software is preferred over comprehensive documentation. Customer collaboration is preferred over contract negotiation. Responding to change is preferred over following a specific plan.

Certifications are required for certain safety-critical systems like aircraft, machinery, etc. in that if a failure happens it could cause loss of life or damage to the environment. The iterative cycles, little documentation, more personal interaction, and the development of little pieces of workable software may seem to be a difficult task to accomplish in which certification is required. My objective is to see how much certification can be utilized using an agile approach to development.

Are you familiar with Agile Development?

- Yes
- No

Have you been successful in using Agility in any of your safety-oriented certified software activities?

- Yes
- No

If you have been successful with Agility, what steps were taken and how well did it work? EXPLAIN

Do you feel the DO-178B guidelines can be changed without sacrifice to safety and security?

- Yes
- No

Requirements analysis can be a long and difficult process where delicate people skills are needed to build the software to the specifications of the stakeholder. What would you change in the requirements of a certified software development without sacrificing security and safety?

With the changes in Software requirements as you previously stated, what would you believe to be the benefit? What would be the disadvantage?

Are you involved in any of the Software Planning using the DO-178B guidelines?

- Yes
- No

If you are involved in software planning, which part?

What level of safety-certified software WOULD YOU FEEL would benefit the greatest in an Agile approach? See definitions of Levels in previous question

- Level A: Software whose behavior, could cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft.
- Level B: Software whose behavior, could cause or contribute to a failure of system function resulting in hazardous/severe-major failure condition for the aircraft.
- Level C: Software whose behavior, could cause or contribute to a failure of system function resulting in a major failure condition for the aircraft.
- Level D: Software whose behavior, could cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft.
- Level E: Software whose behavior, could cause or contribute to a failure of system function with no effect on aircraft operational capability or pilot workload.
- Other:

Team Composition is usually cross-functional and self-forming without any corporate hierarchy or roles in Agile. Do you presently use teams in any of the facets of the certified process?

- Yes
- No

With teaming, do you feel it would be beneficial in a safety-oriented certified software environment? Why?

Do you use any of these Agile tools and techniques such as continuous integration, pair-programming, test driven development, design patterns, code refactoring, unit testing, and other techniques during certified development? Which ones and what benefits does it provide over the conventional way?

Any comments or challenges you would like to add in conclusion to this survey?