Software Architecture, Process, and Management, Level 11 Coursework

Stephen McGruer (s0840449)

March 28, 2013

1 Introduction

Since their inception in the mid 1990s agile methods have flourished, moving from a niche methodology to a serious contender to established processes such as the waterfall model and the unified process. Some flavour of agile methods can be found at most small or young companies[1] and is now even used in large organisations [2], including the Department of Defense [3]. Despite this popularity, there remains one area of software development that has yet to see any real usage of agile methodologies: the development of *safety-critical software*.

This lack of adoption has gathered significant attention in academia, with opinions ranging from the simple assertion that agile methods are unsuitable for safety-critical systems [4–6], to claims that agile methods must be heavy tailored for safety-critical system development [7, 8]. This paper takes a view more in line with the latter, but does not believe that the amount of modification necessary to use agile methods in safety-critical development is substantially more than when applying agile processes to any project. That is, the argument put forward by this paper is that *agile methodologies are suitable and beneficial for large scale, safety-critical systems, with a standard amount of modification.*

1.1 What is Agile Development?

The definition of what makes a particular process agile is as debated a subject as how to apply it to safety-critical software - in fact, it is probably debated far more often! In general, agile processes focus on incremental iterative development, assuming that everything (requirements, architecture, environment, etc) can and will change. For the purposes of this paper the definition of agile is focused on the aims laid down in the agile manifesto [9]:

[1] A recent report from Forrester found that 35-46% of developers claimed to be using agile methods, with most choosing to cherry pick from different agile frameworks [1]

Individuals and interactions over processes and tools.
Working software over comprehensive documentation.
Customer collaboration over contract negotiation.
Responding to change over following a plan.

It is important to note that these aims (and the list of agile principles also laid out by [9]) are not strict rules. Users of agile processes are generally welcome to interpret the agile manifesto as they want, as long as they stick to the core idea of incremental iteration. Similarly, while there are many explicit agile methodologies (Scrum, XP, Crystal, etc), each is only an interpretation of the above principles and should not be taken as gospel.

1.2 What is a Safety-Critical System?

Simply put, a safety-critical system[2] is any system where a failure may result in some form of harm. This includes harm to human life (i.e. injury or death), to vital equipment, and to the environment. There are many examples of safety-critical software: nuclear reactors, flight navigational systems, and spacecraft are three obvious choices, but more subtle systems such as telecommunications, defibrillators and even a circuit breaker can all be safety-critical. Due to the serious consequences of system failure, safety-critical systems must be far more reliable than most other systems - a common rule of thumb is that at most one life must be lost per 10^9 hours of operation (over 100,000 years).

Safety-critical software development differs from normal software engineering in a number of ways, primarily focused on increased reliability. Most prominent is the involvement of *safety engineers*, who are distinct from software engineers and whose focus is on making a *safety argument* about the system. The safety argument is an assertion that the system being developed meets some criteria for safe operation, backed by evidence. Evidence may involve system documentation, testing documentation, independent audits of the code, or even formal proofs of the system. The criteria is usually set by a *standards body*, which must accept the safety argument as valid for the system to be *certified*. Certification is the gold standard of safety-critical systems, and is required for almost all safety-critical projects. Well known examples of safety-critical certificates include DO-178B, which is required for all safety-critical systems that operate aboard FAA[3]-approved aircraft,

[2] Also known as 'life-critical' systems.
[3] The U.S. Federal Aviation Administration

and IEC 60880, which is one of many standards for software development related to nuclear reactors.

This emphasis on safety engineering and certification has lead safety-critical system development to depend on heavy-weight processes such as the waterfall and V-model, which are focused on an *up-front design*. Creating the entire design before beginning implementation, and avoiding deviating from the design while implementing allows safety engineers to put together a single safety argument that is made just once, when the system is finished. However, this approach then runs afoul of the same issues that all non-iterative processes face: a lack of flexibility and vastly increased risk during the end-phase of the project (when the final integration/etc occurs). These issues can be tackled by shifting safety-critical system development away from heavy-weight processes and towards agile methodologies.

2 Why use Agile Methods in Safety-Critical System Development?

This paper claims that agile methods are suitable and beneficial for large, safety-critical system development. Therefore, it must be determined both that agile processes can be used for safety-critical system development (without heavy modification), and that their use improves the development process. Note that as pointed out by Sidky and Arthur [7], organisational buy-in is essential for agile process adoption to be successful. However, as this is true for all organisations, it is not discussed here and organisational buy-in is assumed.

In [10], VanderLeest et al. provide an in-depth discussion of applying agile methodologies to safety-critical systems in the aerospace market. They primarily deal with comparing agile practices with the regulations imposed by the DO-178B standard, and discussing how to adapt the typical approach to DO-178B projects to be iterative. Aerospace engineering is a typical example of a safety-critical field, and so the comments made by VanderLeest et al. can be considered quite general.

Eight 'key' agile practices are determined by the authors to be 'fully compatible' with the DO-178B standard, including: *measuring progress by working software*, *building teams around motivated individuals*, and *simple design*. Another ten practices are identified as 'easily compatible'. The modifications suggested by VanderLeest et al. for each of these appear to be within the normal bounds of agile tailoring, or are unrelated to the agile practice itself. For example, VanderLeest et al. note that face-to-face con-

versation is often limited by the distributed nature of teams in aerospace engineering - however this is a common problem for all distributed teams and has been tackled previously (usually by the use of video conferencing systems). Another example is *welcoming of changing requirements*; VanderLeest et al. note that safety engineers must be made aware of any new requirements so that they can modify the safety argument. This is correct, but is also unrelated to agile processes - as VanderLeest et al. point out, this issue occurs with waterfall processes as well.

VanderLeest et al. do claim that one agile practice is problematic for safety-critical system development, namely *fixed length iterations*. The authors list four difficulties related to this practice: the belief that DO-178B audits present 'lockstep gateways', the use of subcontractors, the size of the projects, and legacy code. However, closer examination shows that the practice is not as problematic as claimed. Two of the difficulties - DO-178B audits and project size - are actually simply viewpoints rather than definite problems. DO-178B audits do not actually need to occur in lockstep (as VanderLeest et al. do note), and agile methods have been frequently applied successfully to larger projects. The use of subcontractors does remain a difficult problem for agile, although methods have been proposed to work around them. Finally, it is not clear why the authors consider legacy code a problem for agile methods; there is certainly nothing obvious about agile that precludes dealing with legacy code.

Ge at al. [8] also discuss the applicability of agile methods to DO-178B (and generally, as they note that DO-178B is representative of many safety-critical standards). The authors report that DO-178B does not define a particular process to be used during development. Instead, DO-178B defines a *framework* that describes the interactions between three separate processes - planning, technical development, and integration. While the traditional waterfall (or V-model) approach applies these three processes in order and only once, that is not enforced by DO-178B. An agile approach of multiple iterations of the three phrases (i.e. having design updates, incremental development, and continuous integration) would be allowed by DO-178B as long as the interactions were honoured. Interestingly, Ge et al. actually dispute that 'pure' agile processes are suitable for safety-critical system development, and suggest a modified approach. Their criticisms of applying agile methods to safety-critical software development are tackled in section 3.

Whilst the above has shown that agile processes are *suitable* for safety-critical system development, there is little point in adopting them unless they are also *beneficial*, that is, if they will improve the experience of the

developers and the quality of the developed system. Much has been written about the benefits of agile processes in general, both for small and large projects [11]. As such, the discussion here focuses on how agile methods can directly aid the development of safety-critical software.

The previously discussed report by VanderLeest et al. [10] also describes some benefits of agile processes for aerospace engineering. They report that test-driven development was useful not only for its primary goal of locating defects, but that it also forced program managers not to 'rush' the requirements definition stage, as they had to know the (current) requirements to define the tests. VanderLeest et at. also highlight the integration problems caused by traditional processes, noting that pushing integration to the end of the development life-cycle causes many of the time and cost overruns that plague the industry, and that the use of continuous integration can be very beneficial for exposing and dealing with the problem earlier. More generally this can be seen as the problem of traditional models pushing the highest risk activities to the end of development, and thus paying a sudden and unexpected price at the point where the project is supposed to be nearly finished. Agile methods generally avoid this, by exposing the risk earlier and allowing it to be accounted for.

Bowers et al. [12] reported on their use of agile methods for developing a safety-critical system at Motorola. While the authors appear hesitant to come down strongly in favour of agile processes, overall they describe a positive impact of agile methodologies on safety-critical system development. Most notable were the use of small releases, which they explain avoided the same 'late-risk' problems of the waterfall model as noted by VanderLeest et al., and the focus on testing, which they report helped exposed bugs early on and inspired developer confidence.

Finally, Knight [13] discusses the general challenges of safety critical systems (unrelated to agile methodologies), and describes a number of failures that could have been mitigated by using agile processes. The most notable case is a nuclear power reactor system that was found at the end of development to pass only 48% of the defined tests. The remaining 52% of tests did not actually fail, but were found to be impossible to run! Using test driven development or continuous integration here would have exposed the problem much earlier.

3 Arguments Against Agile in Safety-Critical System Development

As discussed in section 1, there are many who are critical of applying agile methods to safety-critical system development. Some believe that agile methods are simply unsuitable to safety-critical system development, while others argue that agile must be heavily modified to work with safety-critical systems. It would be remiss to make the claim that unmodified agile processes are suitable and beneficial for safety-critical systems without mentioning the concerns expressed by critics, and therefore the most common objections are discussed here.

Up-Front Design

A common claim is that safety-critical systems require an *up-front design*, for the purpose of certification, safety arguments, or requirements planning. Critics claim that such up-front designs mean that safety-critical systems must be designed using plan-based methodologies, and that agile is therefore not suitable [4,8]. While it is true that *some* aspects of safety-critical systems require an up-front design, agile methodologies and up-front designs do not necessarily conflict. Agile methodologies encourage developers to *minimize* up-front design and to be open the possibility of change. It does not state that up-front designs are disallowed.

Requirements

Requirements behaviour is an area often held up as a key reason that agile methodologies are not suitable for safety-critical projects [4,7]. The primary reason given is that safety-critical systems have up-front requirements that completely specified and do not change throughout the developmental lifecycle. This contrasts strongly with agile principles. For example, Sidky et al. [7] claim that the agile practice of *evolutionary requirements* (i.e. welcoming changing requirements) is unsuitable to safety-critical development as "all requirements must be known before any actual development occurs" and "it ... conflicts with a characteristic of the system (high complexity)".

While Sidky et al. and other critics are correct in that agile methodologies do generally presume that requirements will change, the belief that safety-critical systems somehow avoid this universal problem is incorrect. There are many examples in the literature discussing the problem of requirements discovery and change *during* safety-critical system development

[14–16]. Safety-critical projects tend to suffer from the same requirements as any other large software project including incomplete requirements, contradictory requirements, and existing requirement change due to external factors. Given the existence of requirements change in safety-critical projects, the benefits of agile methods once again become clear. Indeed even Turk et al. [5], who claim that agile methods are unsuitable for safety-critical software development, recognize that agile practices can help with "exploratory development of critical software in which requirements are not well-defined".

Safety Argument

Safety arguments, which are required for many safety-critical systems, are generally considered to be a barrier to agile method adoption. If a system is developed incrementally and iteratively, a safety argument may have to be provided for each iteration of the system. As safety arguments require a lot of work to produce, and are generally produced by safety engineers who do not directly benefit from agile methods, it is common for safety engineers to oppose the use of agile methods.

There are two counter-points to consider here. Firstly, it is not always true that a safety argument is required for every iteration. Agile methods promise to have a deliverable at the end of each iteration such that the customer could cancel at any point and get *something* of value. However, it is rational to claim that the system is not complete until the final iteration, and therefore should not be expected to support a safety argument until then. Of course, while this argument obeys the rules of agile development, it does not really encompass agile philosophy; safety arguments should be iterative and incremental like the rest of the system. To this end, [8] present the idea of *modular* safety arguments, which break down the whole-system safety argument into per-module arguments that are constructed in parallel with the modules. The modular safety arguments are then merged to form the overall safety argument, and combined with an additional argument that argues about the interactions between modules. This allows safety arguments to be constructed iteratively and incrementally alongside the system.

Documentation

One of the key aims behind agile programming is the reduction in the amount of documentation used in software development. Indeed, documentation is mentioned explicitly in the agile manifesto:

> *Working software over comprehensive documentation.*

Critics often use this statement to argue that agile methods result in no documentation being produced, and that documentation is vital for safety-critical system development [7, 17]. At best such claims are mistaken; at worst they are disingenuous. Agile methodologies do *not* encourage a complete lack of documentation. Even extreme programming (XP), which is oft cited as an agile methodology that explicitly bans documentation, does not actually do so. Indeed, Ron Jeffries (co-founder of XP) has refuted such claims [18]:

> Outside your extreme programming project, you will probably need documentation: by all means, write it.

The correct interpretation of the agile manifesto[4] is that often in software development much time is spent creating *unnecessary* documentation that captures only a portion of the team-held knowledge, and which quickly goes out of date. Such documentation is what agile methods disavow. If a piece of documentation is mandated by a standard, or if it is truly being used for communication, agile methodologies have no problem with it.

Traceability

Traceability is an important concept in safety-critical systems, and is often necessary for certification. In a nutshell, tracability is concerned with being able to trace essentially any part of the system (design documents, code, testcases) back to the requirements, and to trace those requirements back to stakeholder rationales. This generally takes the form of a large *requirements traceability matrix* (RTM), which is a lookup table allowing traceability. The RTM generally requires a lot of effort and documentation to create and maintain. As such, critics often use the need for traceability as a reason to discredit the use of agile in safety-critical system development, citing both the huge amount of documentation required (see the documentation section above) and the difficulty in tracking requirements and design when agile methods are being used, since requirements or the design may change.

Traceability is one area where the critics are generally correct, in that agile methods may make it more difficult. However, a vital counter-point to consider is that traceability is incredibly hard already - so hard that even with traditional processes it often fails [19]! In addition, as discussed above requirements and design changes do not suddenly pop into existence due to

[4] In so far as there is a *correct* interpretation. At the very least, it is the interpretation that its creators intended!

agile methods - they exist with traditional processes too, and are often handled worse. Given these facts, agile methods can hardly make traceability much more difficult, and may actually be able to help. For example, Cawley et al. [20] note that agile features like test driven development and good source control practices[5] can aid traceability.

Complexity

The complexity of safety-critical software development is often used as an argument against using agile methods. This is often used to argue that agile practices such as short iterations [12] and refactoring [7, 12] are not suitable for safety-critical systems. Such claims are actually rather worrying. The key requirement for both refactoring and short iterations is understanding the system you are building. If you understand the (current!) design and requirements, it should be possible to decompose it into small enough parts for iterative development. Additionally, it should be kept in mind that iterations in agile development are not required to be as short as two weeks; this is only a suggested length. It is not unreasonable to double (or even triple) this suggested iteration time, and it is hopefully not unreasonable to expect a development team to then be able to produce some form of output in a month and a half!

The claim that refactoring safety-critical projects is not possible is also worrying. If a development team cannot, with the aid of the substantial unit tests and integration tests that they have surely built up due to their agile practices, refactor the system without breaking it, do they really understand it enough for the system to be truly safe?

4 Conclusion

Safety-critical software development is a difficult task. The complexities and pitfalls of standard software development are combined with an increased emphasis on reliability and a requirement to adhere to complicated standards. It is of little wonder then that developers of safety-critical projects continue to cling to aging, heavyweight processes despite the lack of flexibility and increased risk that they cause. This preference for heavyweight processes has both its proponents and critics. The former tend to contend that heavyweight processes are not perfect but that there are no alterna-

[5] Okay, this one isn't actually an agile feature. But it generally goes hand-in-hand with agile methods.

tives, while the latter generally argue in favour of *heavily modified* versions of agile processes.

This paper has demonstrated that agile methods can be both suitable *and* beneficial for large, safety-critical software development, *with little modification necessary.* Benefits include an increased ability to cope with poorly defined requirements, early exposure of unknown risks (through iterative development and continuous integration), and providing confidence in the design via increased testing. Arguments against the use of agile methods in safety-critical systems were also examined and refuted. These include areas where claimed conflict between safety-critical system development and agile methodologies was shown to not actually exist (up-front design, documentation, complexity), where a mistaken belief about safety-critical system development was being used to argue against agile methods (requirements, safety argument), and where the positive effects of introducing agile methodologies actually outweigh the negative effects (traceability).

References

[1] P. Krill, "Agile software development is now mainstream." `http://www.infoworld.com/d/developer-world/agile-software-development-now-mainstream-190`, Jan. 2010.

[2] M. Lindvall, D. Muthig, A. Dagnino, C. Wallin, M. Stupperich, D. Kiefer, J. May, and T. Kahkonen, "Agile software development in large organizations," *Computer*, vol. 37, no. 12, pp. 26–34, 2004.

[3] S. Bellomo and C. C. Woody, "Dod information assurance and agile: Challenges and recommendations gathered through interviews with agile program managers and dod accreditation reviewers," 2012.

[4] B. Boehm, "Get ready for agile methods, with care," *Computer*, vol. 35, no. 1, pp. 64–69, 2002.

[5] D. Turk, R. France, B. Rumpe, *et al.*, "Limitations of agile software processes," in *Third International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2002)*, 2002.

[6] M. Lindvall, V. Basili, B. Boehm, P. Costa, K. Dangle, F. Shull, R. Tesoriero, L. Williams, and M. Zelkowitz, "Empirical findings in agile methods," *Extreme Programming and Agile MethodsXP/Agile Universe 2002*, pp. 81–92, 2002.

[7] A. Sidky and J. Arthur, "Determining the applicability of agile practices to mission and life-critical systems," in *Software Engineering Workshop, 2007. SEW 2007. 31st IEEE*, pp. 3–12, IEEE, 2007.

[8] X. Ge, R. F. Paige, and J. A. McDermid, "An iterative approach for development of safety-critical software and safety arguments," in *Agile Conference (AGILE), 2010*, pp. 35–43, IEEE, 2010.

[9] J. Highsmith and M. Fowler, "The agile manifesto," *Software Development Magazine*, vol. 9, no. 8, pp. 29–30, 2001.

[10] S. VanderLeest and A. Buter, "Escape the waterfall: Agile for aerospace," in *Digital Avionics Systems Conference, 2009. DASC'09. IEEE/AIAA 28th*, pp. 6–D, IEEE, 2009.

[11] L. Vijayasarathy and D. Turk, "Agile software development: A survey of early adopters," *Journal of Information Technology Management*, vol. 19, no. 2, pp. 1–8, 2008.

[12] J. Bowers, J. May, E. Melander, M. Baarman, and A. Ayoob, "Tailoring xp for large system mission critical software development," *Extreme Programming and Agile MethodsXP/Agile Universe 2002*, pp. 269–301, 2002.

[13] J. C. Knight, "Safety critical systems: challenges and directions," in *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, pp. 547–550, IEEE, 2002.

[14] R. R. Lutz and I. C. Mikulski, "Requirements discovery during the testing of safety-critical software," in *Proceedings of the 25th International Conference on Software Engineering*, pp. 578–583, IEEE Computer Society, 2003.

[15] R. R. Lutz and I. Carmen Mikulski, "Operational anomalies as a cause of safety-critical requirements evolution," *Journal of Systems and Software*, vol. 65, no. 2, pp. 155–161, 2003.

[16] L. K. Rierson, "Changing safety-critical software," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 16, no. 6, pp. 25–30, 2001.

[17] J. Drobka, D. Noftz, and R. Raghu, "Piloting xp on four mission-critical projects," *Software, IEEE*, vol. 21, no. 6, pp. 70–75, 2004.

[18] S. Ambler, *Agile modeling: effective practices for extreme programming and the unified process*. Wiley, 2002.

[19] J. Cleland-Huang, "Just enough requirements traceability," in *Computer Software and Applications Conference, 2006. COMPSAC'06. 30th Annual International*, vol. 1, pp. 41–42, IEEE, 2006.

[20] O. Cawley, X. Wang, and I. Richardson, "Lean/agile software development methodologies in regulated environments–state of the art," *Lean Enterprise Software and Systems*, pp. 31–36, 2010.