Towards Safer Medical Device Software Systems: Industry-Wide Learning from Failures and the Use of Safety-Cases to Support Process Compliance

Marion Lepmets
Regulated Software Research Centre
Dundalk Institute of Technology
Dundalk, Ireland
marion.lepmets@gmail.com

Tom McBride
Faculty of Engineering and IT
University of Technology Sydney
Sydney, Australia
tom.mcbride@uts.edu.au

Fergal McCaffery
Regulated Software Research Centre
& Lero
Dundalk Institute of Technology
Dundalk, Ireland
fergal.mccaffery@dkit.ie

***Abstract*—Software safety is checked today in regulatory audits, which verify software development process compliance to regulatory requirements. Ensuring safety is a critical task in complex life-supporting systems and despite many existing ways of assuring it, unanticipated failure will always be possible. Checking process compliance to required standards ensures the quality of the processes by which software is developed but does not necessarily indicate the quality of the resultant software. Since medical device domain is facing an increasing amount of device recalls due to software failures, our goal is to explore the underlying reasons for this and suggest two improvements within this paper. First, we will introduce complicated and complex systems to illustrate why there will always be unforeseeable and unanticipated situations that could cause the failure of the entire system. We will then describe how medical device software systems are reviewed for compliance and safety today, highlighting the shortcomings in the current methods adopted in the medical device domain and suggest the use of systems thinking. We then propose two improvements to medical device software development where process compliance is supported by safety cases and industry-wide learning from experience.**

Keywords—medical device software; software safety; systems thinking; learning from failure; Cynefin framework; safety-critical systems

I. INTRODUCTION

Safety-critical systems such as medical devices embedding software are increasingly complex as they are integrating various sub-systems together. When building such sub-systems, the focus needs to be not only on the quality and safety of the components within a sub-system but also on the interaction of the various sub-systems in order to guarantee a safe and functioning overall system. In addition to that, these systems reside in an environment of various other systems with which it interacts. Systems thinking should be adopted when building such complex systems like medical devices as it encourages understanding of the system, i.e. any set or group of interdependent or temporally interacting parts, by examining the linkages and interactions between the elements that comprise the entirety of the system [1].

One system may have various types of problems that require different solutions. The Cynefin framework [2] provides a way to apply systems thinking as it was developed to distinguish between different types of problems and to recommend practices to solve them.

II. BACKGROUND TO SAFETY-CRITICAL SOFTWARE SYSTEMS

Systems science argues that the only way to fully understand why a problem or element occurs and persists is to understand the parts in relation to the whole. In this Section, we provide a background to complex safety-critical software systems. First, we illustrate the Cynefin framework to elaborate why complex systems have a high probability of unforeseen failure. We then describe the current software compliance practices in the medical device domain.

A. Cynefin Framework

Cynefin was first developed by Dave Snowden in 1999 in the context of knowledge management and organizational strategy as a phenomenological framework, meaning that it is about how people perceive and make sense of situations in order to make decisions. By 2002, it had been developed to include complex adaptive systems theory [2]. In simplest terms, the Cynefin framework exists to help us realize that all situations are not created equal and to help us understand that different situations require different responses to successfully navigate them [3].

Cynefin has two large domains: *Order* and *Unorder*, each containing two smaller domains - *Simple* and *Complicated* in the *Ordered* domain, and *Complex* and *Chaotic* in the *Unordered* domain. In the centre of the framework is the fifth domain called *Disorder* where multiple perspectives fight for prominence, factional leaders argue with one another and cacophony rules. Disorder should be avoided by organizations as it disrupts work. In the domain of order, the most important boundary of sense-making is between what we can use immediately (what is known) and what we need to spend time and energy on finding out (what is knowable). In the domain of *Unorder*, distinctions of 'knowability' are less important

than distinctions of interaction; that is, distinctions between what we can pattern (what is *Complex*) and what we need to stabilize in order for patterns to emerge (what is *Chaotic*). In the *Ordered* domain, the whole is the sum of the parts and the optimization of the system can be achieved by the optimization of the parts. In the domain of *Unorder*, the whole is never the sum of the parts as any action changes the nature of the system. Cynefin's value as a sense-making framework lies in helping system decision-makers understand where their systems lie among these domains, and by extension, what kinds of tools, approaches, processes, or methods are more likely to work successfully in a given system [1].

To use the Cynefin framework when trying to categorize a problem space, one must inspect the relationship between cause and effect of the problem space. If the relationship between cause and effect is straightforward and obvious to all, then your problem is in the simple domain. If the relationship between cause and effect is not obvious, but can be analysed in advance, then you have a *complicated* problem. On the other hand, if the cause and effect can only be determined with the benefit of hindsight, then you are in the *complex* domain, while if there is no obvious relationship between cause and effect, you are in the *chaotic* domain.

As software development organisations face *Complex* or *Chaotic* domains they must take on board more new learning, more situational assessment and understanding, looking and combining capabilities to manage emerging patterns and knowledge, applying experiences, looking for diversity of opinions and searching for new wisdom or insights. Here expertise and experience, collaboration and relationships need significant leveraging, as you often diverge / converge while working through the potential answers [4]. The mindset here is different and it is one that is based on detection. Innovation is far more demanding, pushing frontiers, exploring discoveries, dealing in a series of exchanges and recognizing emerging patterns to piece together real 'new to the world' innovations. The Cynefin framework can be used to guide an approach to a set of different situations, but the characteristics also explain enough to help recognize the situation in which one currently resides. Simply put, you may have developed a great solution, but if you apply it in the incorrect context, it may be worthless or worse, harmful [4].

Pelrine [5] suggests that software development activities tend to be weighted more to the complicated and complex domains, with activities related to the coding aspect of software development landing in the *complicated* (or sometimes *simple*) domain, and activities associated with project management landing in the *complex* (sometimes *chaotic*) domain. Tasks dealing with interaction with a computer tended to be in the *ordered* domains, tasks dealing with interaction with other humans tended to be in the *unordered*, i.e., *complex* and *chaotic*, domains. Although this does not suggest that the entire software development activity as a whole is complex, it does suggest that many parts of it are amenable to analysis and treatment using complexity-based tools and techniques.

B. Complex Safety-Critical Systems

Many safety-critical systems fail in unforeseeable ways due to complex interactions of various components within the system as well as interactions with other systems in their ecosystem. When we develop these systems we are unable to predict exactly how they will be used, how they will respond to and what their interactions with their environment will produce. So we predict what we can, control what we can but, ultimately, we must try the system out and see what happens. This is a form of probe-sense-learn suggested by the Cynefin framework as suitable for exploring, understanding and working with complex domains. In software development, such practices correspond to agile methods' rapid feedback loops and iterative development with self-organizing teams and highly skilled individuals. Despite various benefits of agile methods, to date their quality assurance has relied on the knowledge and expertise of the developers supported by various forms of automated quality assurance, e.g. automated testing, configuration management, defect management. They fail to incorporate the quality assurance thinking that is central to safety critical systems development. In designing sub-systems one at a time, the risk management activities are limited to the sub-system while the failure of a system as a whole either being built or at some time in the future is largely not considered. When all possible interactions between system components (including sub-systems) and between the system and its environment as well as the system's particular history are not accounted for in the development of a complex safety-critical system, there is a serious cause for concern in relation to the system safety [6].

Cook [7] observed that complex systems contain a large number of latent failures all waiting for the right circumstances to expose them, and a change to a complex system introduces new forms of failure. Cook was describing the characteristics of general complex systems, not complex software systems. He ventured that complex systems are heavily and successfully defended against failure, with multiple layers of defenses and where a catastrophe requires multiple failures rather than a single point failure. After the Therac-25 case in the US, FDA reacted quickly to assure proper requirements were in place for complex software systems embedded in medical devices by checking the medical devices prior to market access for software specification and documentation, software quality assurance practices, compliance with international standards, software design documentation, software testing and coding practices, documented software audit trails and usability among various others [8]. One of the most important aspects that was learnt from that case and is now a common practice in medical device software development is the assumption that developers have to adopt when building a safe system - software can always fail. While it may be true that some complex software systems do have defenses in depth against failure, it would be difficult to claim that all complex software systems have such defenses or that all software developers know how to design and develop fault tolerant systems.

Software systems are modified to correct latent or actual faults, to maintain compatibility with component libraries, or to introduce new functions. Some of the larger, more critical, higher performance e-commerce systems such as Amazon, Facebook, Twitter or Netflix are finding that some well-known design principles really need to be applied rigorously. These are principles such as to maintain high cohesion and low coupling within and between objects or components, or to enforce the SOLID principles of object oriented design. In part these principles help ensure that modifications to the system do not introduce subtle defects. For example, a system may have a component that has a defined interface. That interface is generally taken to be a contract between the component and anything that invokes any of its services. However, over time the component may be modified or enhanced in such a way that the interface now behaves in a subtly different way. Possibly a parameter changed from one type of integer to another type, or a list of enumerated codes was extended. Nothing may happen for quite a long time until something somewhere else in the system changes in such a way that the consequences of the changes to the component now become significant. Given that most software now is constructed from numerous components that, in turn, may be constructed from other components, it is difficult to argue that any software system can be made determinate. Software systems do not suffer from the wear and tear of hardware systems but they do suffer from gradual degradation of their integrity as various components are modified.

C. Medical device software regulation

Safety is the central concern for medical device software development and the development of safe systems is rigorously supported by various regulatory requirements focusing on development process compliance. In other words, a strong emphasis is placed on regulatory oversight and device approval before market release to ensure proper verification and validation of these devices. Due to the increased complexity of software in the devices requiring regulatory review, the time to pre-market approval has increased tremendously [9]. This results in the impediment of innovations in the field as the success of innovations is often dependent on the speed of time-to-market. Furthermore, the high percentage of medical device recalls due to software failures [10] indicates that despite the regulatory efforts to oversee the safety and quality of new devices, many faulty software systems are still being passed through the compliance audits.

Two of the largest global bodies responsible for issuing and managing medical device regulation belong to the central governing functions of the US and EU. In the case of the US, the Food and Drug Administration (FDA) issues the pertinent regulation through a series official channels, including the Code of Federal Regulation (CFR) Title 21, Chapter I, Subchapter H, Part 820 [11]. In the EU, the corresponding regulation is outlined in the general Medical Device Directive (MDD) 93/42/EEC [12], the Active Implantable Medical Device Directive (AIMDD) 90/385/EEC [13], and the *In-vitro* Diagnostic (IVD) Medical Device Directive 98/79/EC [14] - all three of which have been amended by 2007/47/EC [15].

In order to satisfy the regional regulation, there are several international standards published to advise and support medical device companies on their road to compliance. In most countries in the world, the medical device companies need to implement a Quality Management System for which they could use the requirements and guidance provided in ISO 13485 [16].

For a medical device manufacturer to demonstrate that all risks have been identified, analysed, evaluated and mitigated in their development of a safe medical device, a risk management process has to be implemented that would satisfy the requirements outlined in ISO 14971 [17]. In the case of developing software as or embedded in a medical device, the guidance on applying the requirements of risk management process to software development can be followed, i.e. Technical Report IEC 80002-1 [18].

IEC 62304:2006 (IEC 62304 from here on) [19], which can be used in conjunction with ISO 13485, offers a framework for the lifecycle processes necessary for the safe design and maintenance of medical device software. As a basic foundation, IEC 62304 assumes that medical device software is developed and maintained within a QMS such as ISO 13485, but does not require an organisation to be certified in ISO 13485. Therefore, IEC 62304 can be considered to be a software development specific supplement to ISO 13485.

Although ISO 13485 and IEC 62304 are accepted in the majority of countries for QMS and medical device lifecycle process compliance, there are additional requirements outlined by the FDA when the device is to be marketed in the US such as FDA QSR [20] for QMS requirements and FDA Guidance on Premarket Submission [21] for medical device software requirements, respectively. In addition to these, there are also the FDA Guidance on Off-the-Shelf software use in medical devices [22] and FDA Guidance on Software Validation [23] that are widely used in regulatory premarket audits in the US.

All of the above-mentioned international standards and FDA guidance documents provide a process compliance approach to quality and safety of medical device software. Although the guidance they provide is critical, it may not be sufficient to guarantee the safety of the software system that is placed on the market. There could be several reasons why this could be the case, for example a) following a prescribed process description in the system development may not guarantee the improved quality or safety of the end product, b) the insufficient software development experience and knowledge among the auditors who evaluate the quality and safety of the software systems, or c) the standards and the guidance documents may be lagging behind the innovative software development practices implemented to develop the increasingly complex medical device software systems. Additional goal-based safety management practices should be introduced to medical device software development that would support the development of safe medical device systems.

III. RECOMMENDATIONS TO IMPROVE SAFETY OF MEDICAL DEVICE SOFTWARE SYSTEMS

There is a clear need for better safety management in which compliance based approach is supported by additional safety management approaches to allow for a safer system. In this discussion we will provide two recommendations for improving the safety of medical device software systems. First, we will introduce the goal-based approach of safety cases, which are widely used in the development of safety critical systems in various domains in conjunction with the compliance-based approach. Secondly, we will recommend learning from actual failures already made throughout the industry that would help inform the verification and validation processes in the development of complex medical device software systems as well as the safety cases to be constructed around the risks already realized in complex systems.

A. Recommendation 1: Using goal-based approach of safety cases in conjunction with compliance-based approach in complex medical device software systems

A safety case is a risk-based argument that together with corresponding evidence demonstrates that all risks associated with a specific goal in a particular system have been identified, that appropriate risk controls have been put in place, and that there are appropriate processes in place to monitor the effectiveness of the risk controls and the safety performance of the system on an ongoing basis [24]. Manufacturers and operators of safety-critical systems in nuclear power plants, petrochemical facilities, railroads, defense, off shore oil and aviation industries have long been using safety cases to demonstrate that their systems are safe to use in a given context.

Safety cases provide a goal-based approach that aims to overcome the shortcomings of the prescriptive compliance-based approach which is mandatory in medical device industry today. In a compliance based approach to safety management risks may not be properly understood, hindering safety as well as innovation and progress in the industry [25].

Safety cases provide a goal towards which the development and the product must steer, in other words they support the process compliance based safety currently required from medical device manufacturers and medical device software developers. As suggested in EC TR 80002-1 Annex E, safety cases help to structure, document and communicate the demonstration of an adequate level of safety of a medical device ensuring safety being maintained throughout the lifetime of the device [18].

Although safety cases provide an additional risk management approach for developing safety-critical systems, there are shortcomings to them as well. First, there is the confirmation bias or cognitive dissonance suggesting that people are likely to reframe evidence to support their deeply held beliefs [28]. This suggests closed loop thinking in which developers construct a safety case for the foreseeable risks evident through their own perspective of the device and how it will interact with the world rather than a systemic and objective understanding of how their device will actually interact with the world.

Another serious shortcoming that has been pointed out in risk management and safety case arguments is their incomplete or inherently faulty reasoning [29]. Fallacies in system's safety argument could undermine the system's safety claims and contribute to a safety-related failure of the system. Greenwell et al [29] studied the frequency and types of fallacies committed in safety cases of safety-critical systems suggesting taxonomy of fallacies that could be used to detect them in safety case reasoning to prevent system failures.

Despite these potential pitfalls of using safety cases, they could provide for a more proactive and structured safety management for complex medical device systems in line with the aims of FDA to ensure more comprehensive approaches to prevent safety problems in medical device and healthcare industries [24].

The FDA required that a safety case be included in a 501(k) submission for infusion pumps since 2010 [26]. The FDA is using the infusion pump safety case as a pilot study to assess the results before expanding requirements for their use to all 510K submissions. While medical device manufacturers oppose to the requirement of safety cases on the grounds that they are an impediment in getting a device to market, alternative means of addressing the increasing rate of Class I recalls have not been proposed [27].

B. Recommendation 2: Accommodate industry-wide learning from failures

Current regulatory practices imply that the device manufacturer is responsible for determining the acceptable levels of risk and for ensuring that the device is adequately safe for use in a specific context [6]. There are various problems with such assumption where a) the manufacturer might not be fully aware of the operational context of the device since each user has a different configuration in the environment in which the device will be installed, and b) with software being subcontracted, the requirements and risks are not fully and openly discussed between device/system level and software level resulting not only in potential integration difficulties but risks which could result in faulty devices.

For risk management of complex safety-critical systems, learning from failures must be ingrained in organizations' and in fact the entire industry's culture. In order to learn from adverse incidents, all data have to be taken into account, including the data that cannot immediately be seen. This culture must be supported with context-specific learning strategies required to effectively defect and analyse failures [28]. Organizational learning occurs through shared insight, knowledge, and mental models built on past knowledge and experience - that is, on memory [30]. Such memory can only be built with shared experience that includes the mistakes and errors that have been made and from which other organizations across the industry can learn in the future.

Syed points out that the biggest problem of not admitting to making a mistake or taking everything into account to analyse a potential mistake, is that mistakes that have been made

become impossible to acknowledge and learned from [28]. This, he suggests, is the risk of healthcare industry where the mistakes that have been made are not publicly discussed and analysed because of the prevalent culture of blame-assertion making the possibility for an industry-wide learning from failures very small if not impossible. Yet, at a level of systemic complexity, success can only happen when we admit the mistakes, learn from them and create a climate where it is, in a certain sense, "safe" to fail [28]. Similarly to Syed, Greenwell also stresses the importance of learning from previous observed system failures and incorporating this knowledge when developing the safety-case of the new complex system [29].

FDA today publishes the data about the medical devices that have produced faults or failed while in use with the name of the manufacturer and a short description of the device itself. Unfortunately, there is no public failure repository which would allow medical device software developers learn from the mistakes of others in the development of innovative safety-critical devices of their own. The data of such a repository could be built by the FDA or European Commission as the recalled devices go through detailed analysis. The anonymized failures in the system or software should be described in as detailed manner as possible. These actual failures can then be used by other medical device developers in their pre-acceptance reviews or hazard analysis to produce a safer medical device through industry-wide learning.

We propose that the regulatory authorities of medical devices publish data about failures made in medical device software development across the domain that would allow the industry to learn and improve the safety and quality of the devices placed on the market. We suggest using such data as one of the input sources against which validation, verification and risk management of new medical device software is conducted, as illustrated on Figure 1.

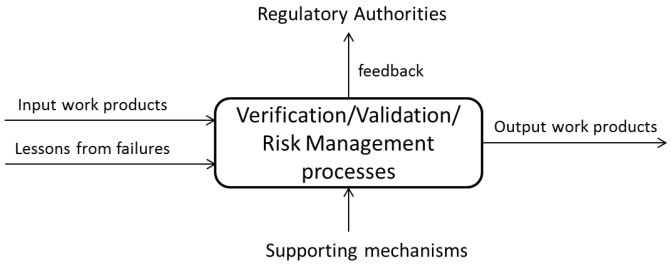

Fig. 1. Using "Lessons from failures" in verification, validation and risk management processes

Furthermore, the lessons learned from each validation, verification and risk management process of new medical device software should provide new insight to findings and possible failures that would be added to the data collected by the regulatory authorities. A similar approach has long been applied in the aerospace industry where industry-wide learning from failures and potential risks are openly discussed and quickly acted upon to prevent further harm in the future [28, 31].

International standards and FDA guidance documents cannot be revised so frequently as to be able to keep up with the innovative software being developed. That is why an actual failures database could contribute to the safer complex medical device software systems informing developers of known failures in software validation, verification and risk management processes as well as in building safety cases that target known risks in industry. Failure repository could help construct safety cases of complex medical device systems by directing attention to failures that have happened and may happen again. This would lead to a more realistic and efficient risk management when compared to what is expected from medical device software developers today - all errors that may cause failures be eliminated, which is impossible to reach in reality.

IV. Conclusion

There is a clear need for better safety management in which a process compliance approach is supported by safety cases and systems thinking for safer complex medical device software systems. Complex systems don't have a clear relationship between cause and effect making it impossible to address all errors that can lead to failures. Complex systems benefit from hindsight of previous failures to enhance their risk management. In other words, safety of medical device software systems may further be improved when the industry could learn from the previous failures across the medical device industry. This requires not only examples but information about actual failures in devices that have been recalled to ensure that both the developers improve the safety of their devices and that the regulatory auditors better ensure that faulty devices do not get to the market. With having such known database, the safety cases as well as process compliance could target these specific areas providing evidence on how safety has been ensured in the new devices.

We advocate that medical device researchers and practitioners collaborate in providing data to an industry wide knowledge base where actual failures could inform the safety case development for new complex medical device systems.

Acknowledgment

This research is supported in part by the Science Foundation Ireland Research Centres Programme, through Lero - the Irish Software Research Centre (http://www.lero.ie) grant 10/CE/I1855 & 13/RC/20194.

References

[1] H. W. Dettmer, "Systems Thinking and the Cynefin Framework - A Strategic Approach to Managing Complex Systems," 2011.

[2] C. F. Kurtz and D. J. Snowden, "The new dynamics of strategy: Sense-making in a complex and complicated world," *IBM Systems Journal,* vol. 42, pp. 462-483, 2003.

[3] R. O'Connor and M. Lepmets, "Exploring the Use of the Cynefin Framework to Inform Software Development Approach Decisions," presented at the Proceedings of the International Conference on Software and Systems Process 2015 (ICSSP 2015), Tallinn, Estonia, 2015.
[4] P. Hobcraft. (2014, March). *The Use of the Cynefin Model for Innovation*.
[5] J. Pelrine, "On Understanding Software Agility - A Social Complexity Point of View," *E:CO,* vol. 13, pp. 26-37, 2011.
[6] M. A. Sujan, F. Koornneef, and U. Voges, "Goal-Based Safety Cases for Medical Devices: Opportunities and Challenges," presented at the SAFECOMP 2007, Nurmberg, Germany, 2007.
[7] R. I. Cook, "How complex systems fail," *Cognitive Technologies Laboratory, University of Chicago. Chicago IL,* 1998.
[8] N. Leveson, "Medical Devices: The Therac-25," in *Safeware: System Safety and Computers*, ed: Addison-Wesley, 1995, pp. 1-49.
[9] T. Kampfrath and S. W. Cotten, "The new collaborative path in medical device development: The medical device innovation consortium," *Clinical Biochemistry,* vol. 46, pp. 1320-1322, 2013.
[10] FDA. (2012, 12.04). *FDA News on Software Failures Responsible for 24% of all Medical Device Recalls*. Available: http://www.fdanews.com/newsletter/article?articleId=147391&issueId=15890
[11] FDA. (15.05). *Chapter I - Food and drug administration, department of health and human services subchapter H - Medical devices, Part 820 - Quality system regulation*. Available: http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/CFRSearch.cfm?CFRPart=820
[12] European Commission, "Directive 93/42/EEC of the European Parliament and of the Council concerning medical devices," ed. European Commission, Brussels, Belgium, 1993, p. 43.
[13] European Commission, "Council directive 90/385/EEC on active implantable medical devices (AIMDD)," ed. Brussels, Belgium, 1990, p. 35.
[14] European Commission, "Directive 98/79/EC of the european parliament and of the council of 27 october 1998 on in vitro diagnostic medical devices," ed. Brussels, Belgium, 1998, p. 43.
[15] European Commission, "Directive 2007/47/EC of the European Parliament and of the Council concerning medical devices," ed. Brussels, Belgium: EC, 2007, p. 35.
[16] ISO, "ISO 13485: Medical Devices - Quality Management Systems - Requirements for Regulatory Purposes," ed. Geneva, Switzerland: ISO, 2003, p. 57.
[17] ISO, "ISO 14971 - Medical Devices - Application of Risk Management to Medical Devices ", ed. Geneva, Switzerland: ISO, 2009, p. 82.
[18] IEC, "IEC TR 80002-1 - Medical Device Software - Part 1: Guidance on the Application of ISO 14971 to Medical Device Software," ed. Geneva, Switzerland: IEC, 2009, p. 58.
[19] IEC, "IEC 62304: Medical Device Software - Software Life-Cycle Processes," ed. Geneva, Switzerland: IEC, 2006, p. 151.
[20] FDA, "Quality System Information for Certain Premarket Application Reviews - Guidance for Industry and FDA Staff," ed. FDA, USA, 2003, p. 19.
[21] FDA, "FDA Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices," ed. USA: FDA, 2005, p. 20.
[22] FDA, "FDA's Guidance for industry, FDA reviewers and compliance on - Off-The-Shelf Software Use in Medical Devices," ed. USA: FDA, 1999, p. 26.
[23] FDA, "FDA's General Principles of Software Validation; Final Guidance for Industry and FDA Staff," ed. USA: FDA, 2002, p. 43.
[24] M. A. Sujan, I. Habli, T. B. Kelly, S. Pozzi, and C. W. Johnson, "Should healthcare providers do safety cases? Lessons from cross-industry review of safety case practices," *Safety Science,* vol. 84, pp. 181-189, 2016.
[25] R. Hawkins, I. Habli, T. Kelly, and J. McDermid, "Assurance cases and prescriptive software safety certification: A comparative study," *Safety Science* vol. 59, pp. 55-71, 2013.
[26] Gessnet. (2014, 7th April). *Innovative Solutions for Medical Device Risk Management & Assurance Cases - ISO 14971, Cybersecurity, Interoperability, and Regulatory Compliance*. Available: http://www.gessnet.com/#!blank/c1sfr
[27] S. Eagles and F. Wu, "Reducing risks and recalls: safety assurance cases for medical devices," *Biomedical Instrumentation & Technology, Association For The Advancement Of Medical Instrumentation,* vol. 48, pp. 24-32, 2014.
[28] M. Syed, *Black Box Thinking - The Surprising Truth About Success*: John Murray Publishers, 2015.
[29] W. S. Greenwell, J. C. Knight, C. M. Holloway, and J. J. Pease, "A Taxonomy of Fallacies in System Safety Arguments," presented at the 24th International System Safety Conference, Albuquerque, NM, USA, 2006.
[30] R. Stata, "Organizational Learning – The Key to Management Innovation," *Sloan Management Review,* vol. 30, p. 64, 1989.
[31] D. R. Wallace and D. R. Kuhn, "Failure Modes in Medical Device Software: an Analysis of 15 Years of Recall Data," *International Journal of Reliability, Quality, and Safety Engineering,* vol. 8, 2001.