Development of Health Software using Behaviour Driven Development - BDD

Mohammad Z Anjum[1][a], Silvana Togneri Mac Mahon[1][b] and Fergal McCaffery[2][c]
[1]*Regulated Software Research Centre, Dundalk Institute of Technology, Co. Louth, Ireland*
[2]*School of Computing, Dublin City University, Dublin 9, Ireland*

Keywords: Automated Acceptance Testing, Requirement Engineering, Behaviour Driven Development, BDD, Software Requirements, Medical Software, Health Software.

Abstract: The health software industry is facing an immense challenge of managing quality and preventing software failures. Poorly defined requirements are one of the significant cause of health software failures. Agile practices are being increasingly used by the software industry to develop systems on time and within budget with improved software quality and user acceptance. Behaviour-driven development (BDD) is an agile software engineering practice that can help to improve health software quality vastly. BDD achieves this by prioritising the illustration of software's behaviour using ubiquitous language, followed by automated acceptance testing to assess if the illustrated behaviour was achieved. This paper presents a review of BDD literature, including the characteristics of BDD and examines how BDD can benefit health software quality. The paper reviews health software standards and guidelines, to examine their compatibility with a BDD approach. Finally, the paper details future plans for the development of a framework that provides health software companies with a detailed step by step guideline on how to use BDD to develop safer health software.

1 INTRODUCTION

The software has become an imperative component of medical devices to provide additional functionality (PTC, 2012). Because of the increasing complexity and reliance on software, the health software industry faces an immense challenge of managing quality and reducing defects (Ronquillo, J. G. et al. 2017). The Food and Drug Administration (FDA) in the USA and European Commission for medical devices in Europe ensures patient safety by reviewing health software products and recalling them if the products do not meet the standards set by them (Zuckerman, D. M. et al., 2011).

A study analysing computer-based failures in medical devices reports that 2,303,441 recalls of medical devices out of 12,024,836 were related to software. Software issues accounted for 33.3% of class I recalls, 65.6% of class II recalls, and 75.3% of class III recalls (Alemzadeh, H. et al., 2013). Poorly defined requirements are one of the most significant causes of software failures (Ward, j. et al., 2003). Inadequate time and effort are spent on the requirements-related activities (FDA - medical device recall report, 2013).

To minimise software failures, different software engineering methodologies/practices have been introduced. A software engineering methodology is a framework used to structure, plan, and control the process of developing software. Software engineering methodology also comprises of different levels of software quality assurance (SQA) activities. SQA activities range from requirements engineering to testing and inspections (Tian, J. 2005). Behaviour-driven Development (BDD) is an agile software engineering practice that encourages collaboration between technical and non-technical stakeholders to ensure that all relevant requirements are captured and mutually agreed. (Smart, J. F. et al., 2015).

Although there have been misconceptions about the suitability of the use of agile methods in safety critical domain including health software, recent

[a] https://orcid.org/ 0000-0003-2511-3047
[b] https://orcid.org/ 0000-0003-0179-2436
[c] https://orcid.org/ 0000-0002-0839-8362

Anjum, M., Mahon, S. and McCaffery, F.
Development of Health Software using Behaviour Driven Development - BDD.
DOI: 10.5220/0008984201490157
In *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2020)*, pages 149-157
ISBN: 978-989-758-400-8; ISSN: 2184-4348

research has proved agile methods can be adapted to the unique needs of health software and can be very valuable for the development of high-quality health software (AMMI, 2012). BDD focuses on requirements engineering to generate tests cases by illustrating software behaviour and then producing automated acceptance tests (Egbreghts, A. 2017). This paper considers how different BDD practices are used to improve software quality and user acceptance. The paper also considers how these practices can be used in the health software domain, including for medical device software development.

BDD is a software engineering practice invented by Dan North in the early to mid-2000s to transform Test-Driven Development -TDD into a more efficient software development process . BDD draws on agile and lean practices, in particular, TDD and Domain-Driven Design DDD, (Solis, C. 2011).

1.1 Test-Driven Development - TDD

TDD is a software development practice that uses a 'test first' approach; it involves writing tests before writing the code that is being tested. TDD relies on the repetition of a very short development cycle. Requirements are turned into specific test cases (Smart, J. F. et al., 2015).

1.2 Domain-Driven Design (DDD)

DDD is an approach to the development of software in which the focus is on the core domain in this case, health software development. DDD is about making the software a model of a real-world or process. In DDD, developers work closely with a domain expert, i.e., compliance manager or healthcare professional, who explains how the real-world system works in his/ her domain. A ubiquitous language (UL) is used to build a common language, to develop a conceptual description of the system between the developer and the domain expert (Evans, E. 2014).

This paper is structured as follows. Section 2 explains what BDD is. Section 3 discusses the existing literature of behaviour-driven development. Section 4 discusses health software standards and guidelines. Section 5 discusses BDD for health software and outlines plans for future work in this area and Section 6 summary & conclusion.

2 BEHAVIOUR-DRIVEN DEVELOPMENT

The objective of BDD is to create executable and well-defined specifications of the software. The BDD process can be divided into three stages.

Stage 1 - Three or more team members, a business analyst or product owner, a developer and a tester; known as the "Three Amigos." will meet to discuss a feature and write up examples. By getting these three individuals to discuss features at the start ensures clear requirements specification are generated.

This is because:

•The Product owner, e.g., Medical device manufacturer, compliance manager or health expert will have the domain knowledge to judge the relevance of the different scenarios.

•The developer will ensure technical considerations.

•The tester, with a focus on validation, will be able to suggest test cases and point out scenarios that the other team members have overlooked.

This exercise enables the developer to have a deeper understanding of the business requirements.

Stage 2 – The examples are converted into scenarios, which are more structured to allow them to be automated in the form of automated acceptance tests.

Stage 3 - The developers will use the TDD approach, as discussed in section 1.1 to write the code required to make this acceptance test pass (Smart, J. F. et al., 2015).

These three Stages are discussed below in further detail. Section 2.1 discusses Stages 1 and 2, and Section 2.2 discusses Stage 3.

2.1 Requirements Capturing & Specification

BDD offers a specification technique. It supports continuous requirements engineering with the use of stories. These stories help to specify executable requirements in a natural language format using UL (I. Lazăr et al., 2010). Executable requirements act as live documentation, making it easier to receive feedback early and conduct acceptance tests. The UL is used to write stories and scenarios and can guide the developers in understanding what feature/ behaviour are needed to be implemented (Kenneth P. 2011).

2.1.1 Ubiquitous Language (UL)

Requirements are interpreted by the developers and testers to produce software and test scripts. However, different people interpret complex concepts

differently (Evans, E. 2014). Therefore, by using ubiquitous language, BDD helps stakeholders to understand functional specifications. UL allows requirements to be consistent and readable to all, which minimises the possibility of misunderstanding. The UL used by BDD is referred to as Gherkin syntax and is used by the BDD tool Cucumber. However, JBehave, which is also a BDD tool, that has its own syntax, was developed separately and has some differences (Egbreghts, A. 2017).

2.1.2 Behaviour Illustration

According to Liz Keogh, a core member of the BDD community and a contributor to some open-source projects including J-Behave, 'BDD is the art of using examples in conversation to illustrate behaviour'. In BDD, examples and conversation are used to discover and describe the behaviour of the system. Using conversation and examples to specify how you expect a system to behave is a core part of BDD (Keogh L. 2012). Examples are used by BDD to specify scenarios. These examples can be used as a tool to express and discuss business needs and expectations, and they make it much easier to clear misunderstandings (Solis, C. et al., 2011). These BDD requirements tracing & specification techniques can assist health software development companies, to understand complex requirements, including regulatory requirements more effectively. Once requirements are defined, and the acceptance test is written using ubiquitous language, the test is then automated using BDD automation tools.

2.2 Automated Acceptance Testing

In BDD automation for acceptance test is achieved through the tools like Cucumber, JBehave, SpecFlow, frameworks, and test suites automation. BDD tools allow scenarios to be run automatically and use UL using "Given, When, Then" format. (Solis, C. et al., 2011). Time to market has become key even in safety critical domains and demand for implementing continuous integration, and continuous delivery has increased. Automated acceptance testing makes continuous delivery possible. As new releases can be deployed with low risk of introducing regression (G. Lucassen. et al., 2017).

This section discussed requirements capturing & specification techniques used in BDD as well as automated acceptance testing. The next section discusses state of the art, the existing literature of behaviour-driven development and the use of behaviour-driven development, particularly in the safety-critical domain.

3 STATE OF THE ART – THE USE OF BDD

This section discusses the existing literature of BDD in safety-critical domains. It is to be noted that despite it being an established practice in the software industry, the academic literature of BDD is still limited (Egbreghts, A. 2017) especially for safety-critical domains.

C. Baillon and S. Bouchez-Mongardé conducted a study and published their work in 2010 on executable requirements in a safety-critical context. They mention different characteristics of BDD as a potential solution to a number of modern software industry problems. The study also proposes that BDD practices can be used to address challenges facing in the safety-critical domain by legacy software. Legacy software are often critical to the companies and over the years have been maintained by a number of programmers. Which means that many changes have been made to the software but the supporting documentation may not be up-to-date. The authors' study proposes using BDD and executable requirements to build a step-by-step understanding of untested legacy software's behaviour (C. Baillon et al., 2010).

Similarly, in 2011, a systematic mapping study of requirements specification and testing techniques mention BDD as a new development paradigm to address requirements traceability problems (Egbreghts, A. 2017).

Diepenbeck and Kühne published a paper in 2015 on behaviour driven development for tests and verification. They proposed BDD for design and verification for safety critical hardware systems. They introduced a new element for defining properties called natural language and supported the assembling of 'property specification language'. They presented an example of a BDD based flow that combines testing and verification using natural language tests and properties as a starting point for the design of the hardware.

In 2017 Hatko, Mersmann, and Puppe published their work where they have described an approach inspired by BDD for specification and analysis of Computer-Interpretable Clinical Guidelines (CIG). These requirements, where stated by medical experts in natural language and are used as design input for the development of CIGs and their analysis using test

cases. The paper demonstrated the applicability of BDD for CIGs. They concluded the approach had shown its applicability regarding usability and expressiveness (Hatko, R., et al., 2014).

The above literature demonstrates the benefit of using BDD, especially its requirement engineering and automated acceptance testing approach. The literature exhibits the potential benefit of using BDD practices in safety critical domain. It also highlights the need for further research in this area of safety critical domains, including medical devices. The literature also shows how particular BDD practices can be used instead of the complete BDD methodology to achieve the desired results.

This section discussed the existing literature of behaviour-driven development in the safety-critical domain. The next section looks at relevant health software standards & guidelines to be considered in order to use BDD to develop health software.

4 HEALTH SOFTWARE REGULATION & BDD

There are a number of different standards and guidelines from government and non-government organisations to ensure standardisation and to regulate the quality of safety-critical software. This also applies to the safety of medical devices software. Many standards are harmonised between USA and EU via "recognised consensus standards" in the USA (U.S. FDA) and European directives in the EU (European Commission). To meet the regulatory requirement, health software developers and medical device manufacturers must understand what the regulators are assessing. Two standards, one guideline and technical information report have been identified as relevant to this research.

These standards, guideline and technical information report, listed below are primarily aimed at health software developers, including medical devices companies to assist them in understanding if their product is designed with safety in mind (Zuckerman, D. M., Brown, P., & Nissen, S. E. 2011).

a) IEC 62304 – Software Life Cycle Processes

The international standard IEC 62304 – medical device software – software life cycle processes (IEC, 2015). This standard harmonised by the EU and the FDA and can be used to ensure compliance for both the EU and the USA market.

b) IEC 82304-1 Health Software Product Processes

The international standard IEC 82304-1 deals with general requirements for safety and security of 'health software products'. IEC 82304-1 inherits quite a lot of its characteristics from IEC 62304 and refers to health software products companies back to IEC 62304 (C. Michaud, 2016). The main reference is in section 5 of IEC 82304-1 titled health software - software lifecycle process.

c) FDA General Principles of Software Validation; Final Guidance for Industry and FDA Staff (GPSV)

GPSV is the guidance on general validation principles that, the FDA considers to be applicable to the validation of medical device. This guidance describes FDA approach to evaluating a software validation system.

d) AAMI - TIR45 - Guidance on the Use of Agile Practices in the Development of Medical Device Software

This Technical Information Report (TIR) provides recommendations for complying with international standards and FDA when using agile practices for the development of medical device software.

The remaining of section 4 discusses these standards and guidelines' requirements compared with the BDD's practices.

4.1 Health Software Requirements

Poor software requirements are one of the main reason for the failure of health software devices, as discussed in section 1 of this paper. The both IEC 62304 & IEC 82304 – 1 put great emphasis in health software requirements, particularly in section 5.2 of IEC 62304 and section 4 of IEC 82304-1. As discussed in section 3 of this paper, the BDD's requirements tracing and specification techniques that result in stories are the key to BDD's success. These techniques can be aligned with the requirements elicitation stages defined in section 5.2 of IEC 62304 and section 4 of IEC 82304-1. Below is a list of individual requirements elicitation stages from the two standards and how BDD aligns to the requirements of these stages.

a) Requirements Gathering

Requirements gathering is discussed in detail in section 5.2.2 of IEC 62304 (IEC, 2015) and section 4.2 IEC 82304-1 (IEC, 2017). Both of these standards require different aspects of software requirements for medical devices to be determined and documented including functional and capability requirements, software system inputs and outputs requirements, interfaces between the software system and other systems, security requirements and data definition and database requirements to name a few. J. Ferguson

Smart, in his book 'BDD in action', explains that product owner, along with the team will collectively define requirements in amigos meetings as users' stories for requirements gathering in BDD process (Smart, J. F. et al., 2015). This can include requirements required by these standards.

b) Risk Assessment

Risk Assessment is the section 5.2.3 and 5.2.4 of IEC 62304 and section 4.1 IEC 82304-1. The IEC 62304 asks the manufacturer to implement risk control measures but leaves it up to the manufacturer to decide how. IEC 62304 also requires the manufacturer to re-evaluate the medical device risk analysis after the software requirements are established and updated (IEC, 2015). The TIR45 proposes elaborated stories to define risk requirements and prioritising these stories in the development of backlog (AMMI, 2012).

c) Verification of Requirements

Verification of Requirements is the section 5.2.6 of IEC 62304 and section 4.3 IEC 82304-1. IEC 82304-1 requires for four things at this stage; requirements should not contradict each other, avoid ambiguity; permit the establishment of test criteria, uniquely identified (IEC, 2017). As discussed in section 2.1.1 of this paper by defining the requirements in UL, BDD addresses all four of these requirements (IEC, 2017).

d) Updating Requirements

Updating requirements is the section 5.2.5 of IEC 62304 and section 4.4 IEC 82304-1. Backlog refinement is a BDD activity, which addresses updating requirements (AMMI, 2012). IEC 62304 asks the manufacturer to ensure that existing requirements are re-evaluated and updated as appropriate as a result of the software requirements analysis activity (IEC, 2015). Similarly, the IEC 82304-1 asks the manufacturer to ensure that the health software product use requirements are updated as appropriate, e.g., as a result of health software product use requirements verification or as a result of validation (IEC, 2017).

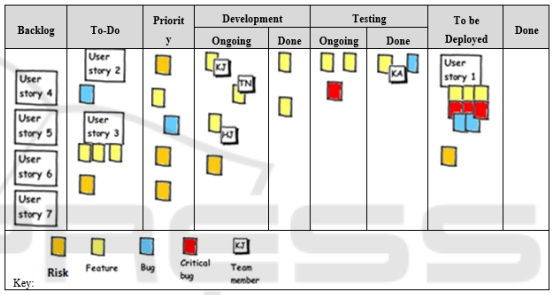
In BDD, the process of backlog refinement allows the team to remove user stories that are not appropriate anymore, as well as defining new user stories if new requirements have surfaced. The next section discusses risk management while using BDD to develop health software.

4.2 Risk Management for Health Software with BDD

Regulations require medical device companies to follow a robust set of human safety risk management activities in their product development. Such activities include risk planning, risk analysis, risk control identification, and risk control verification. The documentation and approval activities should be in place in accordance with the organisation's quality management system. In the case of health software companies the ISO 13485, medical devices - quality management systems is the regulatory standard that can be used as the organisation's quality management system.

A backlog can be considered as a 'to-do list' that details specific outcome based on proposed features, changes to the existing features and the infrastructure, as well as software bugs that need addressing. The backlog allows the team to ensure that they are only using one authoritative source of 'to-do list'.

Table 1: Example backlog board with priority column and risks.



The backlog board in table 1 shows an illustration of proposed health software backlog. Two new columns 'priority' and 'to be deployed' has been added to satisfy the requirements of the standards discussed above. The illustration also uses a different colour to identify different tasks. For example, risks are identified in orange colour on the backlog board, and risk management policy can be used to ensure that the team always prioritise risk on the backlog board.

A similar approach was discussed by G. K Hanssen and T. Stalhane in SafeScrum – Agile Development of Safety-Critical Software.(Hanssen. G. K. et al., 2018).

The nature of health software is safety critical and does not always allow the software to release as frequently as non-safety critical software (AMMI, 2012). The backlog board's 'to be deployed' column allows teams to use the column as a place holder until it is considered safe to release the iteration. Similarly, what is classed, as 'done' for stories of health software has to be more defined and detailed. For this reason, the DoD for health software requirements must be defined as part of the requirement elicitation process. The following section of this paper discusses this further.

4.3 Definition of Done (DoD) for Health Software Requirements

A doneness policy & procedure can be created to help define what will be classed as 'done' for a health software story and the individuals that will need to sign off on doneness. These doneness policies & procedures are known as DoD, a set of criteria which must be met before a software item is considered to be complete. The DoD is agreed upon by the development team and the product owner. The DoD can be used at various points in a development project; for example, a DoD can be created to ensure a software system is complete, or to ensure a software requirement is complete. Part of the doneness policy & procedure will be to verify and validate the story to assess the doneness. This is done using a number of different verification and validation techniques.

This section discussed the health software regulations with respect to how BDD can be used and adapted to develop safer health software. The following section proposes a framework for health software development using BDD characteristics discussed in this paper.

5 BDD FOR HEALTH SOFTWARE

This section of the paper will focus on outlining an approach to the development of a behaviour-driven health software framework (hBehave), as the solution for the problems detailed in this paper. This section will also discuss, how this framework will be validated. As discussed in section 1 agile practices including BDD despite being very successful in non-health industry are rarely used in medical software development , because of health software companies uncertainty around losing their certification by changing their practices (AMMI, 2012) (Clarke, P. et al., 2014).

The framework will provide health software companies with detailed step by step guideline on how to use BDD to develop health software. The framework will also detail how to adapt BDD to satisfy different health software regulations. The aim of the hBehave framework will be to act as a handbook for health software companies to help migrate their development process to BDD. It will be broken down into different processes derived from IEC 62304 and agile software development life cycle as detailed in AMMI TIR45 (AMMI, 2012). The remaining section of this paper discusses different aspects of the hBehave framework including the framework processes, processes table structure and framework validation.

5.1. Behaviour-Driven Health Software Framework – hBehave

The framework processes detailed below derive from IEC 62304, figure 1, software development processes and activities.

5.1.1 Framework Processes

a) Development Planning

Development Planning is the first activity according to software development processes and activities defined in figure 1 of IEC 62304 – Medical device software – software life cycle processes (IEC, 2015). Development planning is also the first activity of each layer of software development in BDD / agile development as detailed in AMMI TIR45 in figure 4 (AMMI, 2012). The development planning is performed at each layer of the BDD development, including project, release, increment, and story layer (AMMI, 2012). The main activities and output of this process of the framework will be the development of a backlog board, unique for each project and project policies and procedures including the definition of done.

b) Requirement Elicitation

Requirement elicitation is the 2nd activity according to software development processes and activities defined in figure 1 of IEC 62304 – Medical device software – software life cycle processes titled as requirement analysis (IEC, 2015). Requirement elicitation is also the 2nd activity of project and story layer of software development in BDD / agile development as detailed in AMMI TIR45 in figure 4 (AMMI, 2012). The main activities and output of this process of the framework are requirement discovery, requirement definition and formation of requirements.

c) Software Architecture & Design

Software Design covers 3rd and the 4th activities in software development processes and activities defined in figure 1 of IEC 62304 – Medical device software – software life cycle processes titled as software architectural design and software detailed design (IEC, 2015). Software design is also the 3rd activity of project layer of software development in BDD / agile development as detailed in AMMI TIR45 in figure 4; labelled as infrastructure spikes. Spikes are stories that will result in the team learning, prototyping, and ultimately developing execution strategy. Software emergent, the 3rd activity of the

story layer and as software detailed design, the 4th activity of the story layer (AMMI, 2012).

d) Unit Implementation & Verification

Unit implementation & verification is the 5th activity in software development processes and activities defined in figure 1 of IEC 62304 – Medical device software – software life cycle processes (IEC, 2015). This activity is also the 5th activity of story layer of agile development life cycle as detailed in AMMI TIR45 in figure 4 (AMMI, 2012). This process of the framework will follow the test-driven development – TDD's approach and main activities and output of this process are unit implementation (TDD), unit testing (TDD) and refactoring (TDD).

e) Software Integration & Integration Testing

Software integration & integration testing is the 6th activity in software development processes and activities defined in figure 1 of IEC 62304 – Medical device software – software life cycle processes (IEC, 2015). In BDD / agile development life cycle, the Software integration & integration testing is performed in release layer, increment layer as well as story layer as detailed in AMMI TIR45 (AMMI, 2012).

f) System & Regression Testing

System testing is the 7th activity in software development processes and activities defined in figure 1 of IEC 62304 – Medical device software – software life cycle processes (IEC, 2015). System & regression testing is performed in release layer, increment layer and the system testing is performed story layer in BDD / agile development life cycle, as detailed in AMMI TIR45 in figure 4 (AMMI, 2012).

g) Software Release

Software release is the 8th activity in software development processes and activities defined in figure 1 of IEC 62304 – Medical device software – software life cycle processes (IEC, 2015). Software release is performed in release layer of BDD / agile development life cycle, as detailed in AMMI TIR45 in figure 4 (AMMI, 2012). The main activities and output of this process of the framework are automation of acceptance testing and generation of living documentation.

This section detailed the processes for the behaviour driven health software framework, as part of the proposed future work. To continue the development of the framework, our future work will focus on further development of different aspects of the hBehave framework. The future work will also focus on validation of Behaviour-driven health software framework. The research will consider the following points in terms of the validation the efficiency of proposed framework, the reliability of proposed framework and ease of adaptability of proposed framework.

Data will be requested from health software companies to understand their existing software development processes, in particular around requirement engineering processes and acceptance testing processes. After analysing this data, the proposed hBehave framework will be revised to reflect health software companies' needs derived from the data collected. The framework will then be presented to health software companies for implementation, and the results will be observed. This method of validation will be used to assess the quality of hBehave in terms of efficiency, reliability and adaptability.

6 SUMMARY & CONCLUSION

In this paper, the software engineering practice BDD was discussed. As well as how BDD practices can help to address requirements related to health software failure as identified in section 1. This paper explained software quality problems in health software. The benefits of behaviour-driven development's practices were outlined with the empirical examples from the literature. BDD's practices can be used to address challenges such as poorly defined requirements.

Section 4 of the paper identified two standards IEC 62304 and IEC 82304-1, a guideline by FDA, the GPSV and technical information report by AMMI, the TIR45 as relevant to this research. Followed by, the identified standards and guidelines were discussed in detail along with BDD to established BDD's compatibility as a health software engineering practice. The IEC 82304-1 is seen as a breakthrough, as it works along IEC 62304 to provides the flexibility needed by health software developers to use agile approaches to develop software (C. Michaud, 2016).

Section 5 of this paper details future work, including behaviour-driven health software framework – hBehave, as the potential solution for the problems detailed in this paper, as well as how this proposed solution will be validated. The proposed framework aims to provide health software companies with detailed step by step guideline on how to use BDD to develop safer health software.

In conclusion, our research findings show that BDD has the potential as a health software development technique. Furthermore, BDD's practices, in particular requirements tracing & specification and automated acceptance testing, has

the potential to address the research problem detailed in section 1 of this paper. High-level mapping has been done between Behaviour-driven health software frameworks processes and IEC 62304 & IEC 82304, which shows promising compatibility. However, significant work has to be done to develop Behaviour-driven health software framework and where required adapt BDD to fulfil the regulatory requirements to build confidence among health software companies.

ACKNOWLEDGEMENTS

This work was supported with the financial support of the Science Foundation Ire-land grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

REFERENCES

PTC (2012). Software Development for Medical Devices, Software Development for Medical Devices, PTC.

Ronquillo, J. G., & Zuckerman, D. M. (2017). Software-Related Recalls of Health Information Technology and Other Medical Devices: Implications for FDA Regulation of Digital Health. *The Milbank Quarterly*, 95(3), 535–553. doi: 10.1111/1468-0009.12278

Zuckerman, D. M., Brown, P., & Nissen, S. E. (2011). Medical device recalls and the FDA approval process. *Archives of Internal Medicine,* 171(11), 1006–1011. doi: 10.1001/archinternmed.2011.30

Egbreghts, A. (2017). A Literature Review of Behavior Driven Development using Grounded Theory.

Medical Devices, European Commission (2019). Retrieved from https://ec.europa.eu/growth/sectors/medical-devices_en

Keogh L. *It's about the examples you can't find, not the ones you can.* Liz Keogh, lunivore. https://lizkeogh.com/2012/02/20/its-about-the-examples-you-cant-find-not-the-ones-you-can/. Published February 20, 2012. Accessed December 17, 2019.

Alemzadeh, Homa, et al. Analysis of Safety-Critical Computer Failures in Medical Devices. *IEEE Security & Privacy, vol. 11*, no. 4, 2013, pp. 14–26., doi:10.1109/msp.2013.49.

FDA - *Medical Device Recall Report (2013).* Retrieved from http://www.fda.gov/downloads/AboutFDA/CentersOffices/OfficeofMedicalProductsandTobacco/CDRH/CDRHTransparency/UCM388442.pdf

Ward, j, Shefelbine, S., & Clarkson, P. J. (2003). Requirements capture for medical device design. *In requirements capture for medical device design.*

Martin, J., Murphy E.A., Crowe, J.A. and Norris B. (2006). Capturing User Requirements in Medical Device Development: *The Role of Ergonomics, Physiological Measurement* 27(8) pp. R49-R62.

Smart, J. F., & North, D. (2015). Bdd in action: behavior-driven development for the whole software lifecycle. *Shelter Island, NY: Manning.*

G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, S. Brinkkemper and D. Zowghi. (2017) "Behavior-Driven Requirements Traceability via Automated Acceptance Tests," *IEEE 25th International Requirements Engineering Conference Workshops (REW)*, Lisbon, 2017, pp. 431-434.

Wallace, D. R., & Kuhn, D. R. (1999). Lessons from 342 Medical Device Failures. *In Lessons from 342 Medical Device Failures.*

Ward, J. R., & Clarkson, P. J. (2004). An analysis of medical device-related errors: prevalence and possible solutions. *Journal of Medical Engineering & Technology,* 28(1), 2–21. doi: 10.1080/03091903100012374 7

Tian, J. (2005). Software Reliability Engineering. Software Quality Engineering, 371–387. doi: 10.1002/0471722324.ch22

(AMMI, 2012), "Technical Information Report AAMI TIR45 : Guidance on the use of AGILE practices in the development of medical device software.

Evans, E. (2014). Domain-driven design reference: definitions and pattern summaries. *Indianapolis: Dog ear publishing.*

Solis, C., & Wang, X. (2011). A Study of the Characteristics of Behaviour Driven Development. *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications.* doi: 10.1109/seaa.2011.76

Hatko, R., Mersmann, S., & Puppe, P. (2014). Behaviour-driven development for Computer-Interpretable Clinical Guidelines. In Behaviour-driven development for Computer-Interpretable Clinical Guidelines (Vol. 1289).

I. Lazăr, S. Motogna, and B. Pârv. (2010) "Behaviour-driven development of founda-tional UML components," Electron. Notes Theor. *Comput. Sci., vol. 264*, no. 1, pp. 91–105, 2010.

Kenneth P. (2011), Lean-Agile Acceptance Test-Driven Development: Better Soft-ware Through Collaboration.

C. Baillon and S. Bouchez-Mongardé. (2010). Executable requirements in a safety-critical context with Ada," *Ada User J.*, vol. 31, no. 2, pp. 131–135.

U.S. FDA. *Recognized Consensus Standards*. Retrieved from https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfStandards/search.cfm

European Commission. *Harmonised Standards - European Commission.* Retrieved from https://ec.europa.eu/growth/single-market/european-standards/harmonised-standards_en

IEC, (2015). "Medical device software — Software life-cycle processes," *Bs En 62304-2006 +a12015, vol. 3*, no. November 2008, 2015.

IEC, (2017). BS EN 82304-1:2017 - Health Software -Part 1: General requirements for product safety BS," 2017.

FDA, (2002) - General Principles of Software Validation; Final Guidance for Industry and FDA Staff

C. Michaud, (2016) "*IEC 82304-1 - Consequences on agile software development pro-cesses - Software in Medical Devices, by MD101 Consulting*." [Online]. Retrieved from: https://blog.cm-dm.com/post/2016/04/08/IEC-82304-1-Consequences-on-agile-software-development-processes.

Hanssen. G Kjetil, Stålhane. T, Myklebust, T , (2018) SafeScrum® – Agile Development of Safety-Critical Software