Developing a Software Quality Framework for Low-Code Model Driven Development Platforms Based on Behaviour Driven Development Methodology.

Stephan Braams  
University of Twente  
P.O. Box 217, 7500AE Enschede  
The Netherlands  
s.m.braams@student.utwente.nl

ABSTRACT

From a case study that was conducted for this research, of a company that uses the Low-Code Model Driven Development (LCMDD) Platform Mendix, it appears that no standard approach for testing has emerged, tackling the unique challenges that the LCMDD domain faces. Because traditional testing techniques are not directly applicable in LCMDD, research is needed to investigate a possible quality assurance approach. An innovative methodology that is gaining traction, which might fit the bill, is Behaviour Driven development (BDD). However, limited academic literature is available describing BDD. This paper presents a state-of-the-practice review of the principles that define BDD, using the Glaser variant of Grounded Theory, applied to qualitative data gathered from four though leaders in the field. The testing maturity of the LCMDD field is assessed through a case study and the available testing techniques for three LCMDD platforms. The identified BDD principles are mapped onto the LCMDD process, arriving at a software quality framework, modelled after the Van den Broek et al. Scrum and Testing Framework. The research advances the scientific understanding of LCMDD and BDD, as well helping organisations to adopt a holistic software quality approach based on the BDD principles through the use and adaption of the proposed framework.

Keywords

Behavior driven development, Specification by Example, low-code model driven development platform, Mendix, agile testing, testing maturity, software quality framework

1 Introduction

The Agile Principles from the Agile Manifesto [4] moved the software development industry away from the then prevalent document-driven, heavyweight waterfall style of software development. While best-practices and understanding of the software development process as a product development process instead of a production process[21] are slowly emerging, customer demands and expectations continue to evolve and new ways of developing software are continuing to emerge. One of these fields is Model Driven Development (MDD), which was formalised when the Object Management Group (OMG) defined their vision and standards for Model Driven Architecture (MDA) in 2001 [43]. Where traditionally models were used for description, communication and understanding, in MDD models are the central driving force behind the software development process, and are automatically transformed into code. The transformation process can be performed with varying degrees of human involvement. The tools that are used for transformation and their respective level of human involvement differentiate subfields in MDD [19]. In the 1990’s, proprietary Computer-Aided Software Engineering (CASE) tools were used to partially generate information systems based on models[48]. More modern approaches include OMG’s MDA and Executable UML (xUML) [43]. Another subfield of MDD consists of low-code model-based platforms like the Mendix Platform, the OutSystems Platform and Salesforce App Cloud [9]. Using these platforms, a software developer predominantly interacts with different kinds of models, instead of code[19]. No industry standard name or abbreviation for this field was encountered. Alternatives include Rapid Dev, Model-driven Platform as a Service (PaaS), High productivity PaaS, Rapid Application Delivery (RAD) PaaS and low-code platform. In this paper, this MDD subfield will be referred to as Low-Code Model Driven development (LCMDD)

1.1 Problem statement

Organisations spend up to 50% of their resources on testing, which indicates its importance, but it is commonly experienced as error-prone, unpopular or tedious work[34]. From a wider perspective however, testing is the most used quality assurance technique in the software development industry[30]. The use of the term quality assurance in lieu of the more common term from manufacturing, quality control, hints towards the software development industry’s relationship with quality. Even today, after decades of progress in software development processes and practices, 140 billion euro’s are spent on failing software projects, each year, by governments and the private sector in the EU [10]. Data from the Standish Group, who investigate success and failure statistics of software projects, show that, as of 2015, only 20% of software projects ended successfully[18]. One causal factor in this high failure rate is that even when high functionally qualitative software is delivered, its behaviour (i.e. added-value quality) frequently does not meet the expectations of the customer, which can lead to the failure of the project as a whole [40]. These two sides to software quality are represented clearly in Marick and Crispin’s Agile Testing Matrix [16], where testing activities are classified into four quadrants.
Along the axes of 'Business facing' to 'Technology Facing' and on the side of 'Supporting the Team' are quadrants Q1 (Unit and Component testing; highly automatable) and Q2 (Functional Testing, Examples, Prototypes; moderately automatable). On the side of 'Critique Product' are Q3 (Exploratory Testing, Scenarios, Acceptance Testing; not automatable) and Q4 (Performance, Security & Load testing; supported by tools).

How quality assurance should be approached in the LCMDD field is unclear: the existing body-of-knowledge about testing techniques is not directly applicable. In traditional testing, the cornerstone of the testing process is Unit Testing. A Unit Test verifies a small, isolated piece of code. Unit tests are often not available, or practical, on a LCMDD platform, because virtually no code is directly or conveniently available to test [19]. Instead, the system of models which the developer interacts with should be tested, which indicates the need for the development of a novel testing methodology specific to LCMDD and an evaluation of available model testing techniques. This paper will address the first part of the identified need: the development of a testing methodology for the LCMDD software development process. The wider perspective on software testing suggests that instead of a testing methodology, a software quality approach should be the goal for the LCMDD field. Some of the fields in software testing that are available to base a testing approach on, and are currently developing, are Finite State Machine (FSM) testing [24], Model-Based Testing (MBT) [43], Test-Driven Development (TDD) [3] and Behaviour Driven Development [42]. FSM, MBT and TDD are all based on white box access of the system under test, while the access to LCMDD systems is more akin to black box access. Hence, the software quality approach should be based on Behaviour Driven Development (BDD), since the gap to be bridged is assumed to be the smallest. An indicator to the match between BDD and LCMDD is the fact that the Outsystems Platform supports and integrates BDD as its official quality assurance method [38]. While BDD is gaining traction in practice, academic literature is lagging behind: preliminary searches on Scopus result in only 65 articles that mention BDD. To be able to apply BDD methodology to the LCMDD testing problem, an up-to-date understanding of BDD is needed. How this is achieved is elaborated on in section 2.

1.2 Desired Results

The goal of this research is: Developing a software quality framework for the low-code model driven platform development process through the application of a state-of-the-practice understanding of Behaviour Driven Development. In addition to arriving at a state-of-the-practice understanding of BDD, to develop a relevant framework it is important that the LCMDD process is examined carefully. A case study achieves this and adds to the scientific understanding of the characteristics of the LCMDD development process. In addition to an increased description and scientific understanding of both BDD methodology and the LCMDD process, gaps in the current literature and opportunities for future research for these fields are identified. Finally, the framework that is developed provides a foundation for the maturing and formalising of software quality in the LCMDD field, as well as providing a tool for this industry to structure their current software quality approach. The framework can be generalised, which offers the opportunity for a unified approach in applying BDD to achieve software quality in many domains of software development.

1.3 Research Questions

During the development of the framework, the following research questions were considered:

1. What is the state-of-the-practice of BDD methodology, according to thought leaders?
2. How mature is testing in the LCMDD field?
3. How is the LCMDD testing process currently structured?
4. How do BDD principles map to the LCMDD process?

1.4 Structure

In section 2 the available literature is reviewed and the research methodology is described. Sections 3 to 6 follow the structuring of the research and provide the primary methods that are described. Section 7 contains the research strategy used for analysing BDD, which is discussed further in Appendix B. In section 8 the literature is reviewed which provides directions for future research. Section 9 contains the acknowledgements.

2 Methods

In this section, first the available literature will be discussed. Secondly, the research methods and analysis strategy are reviewed. Finally, validity and reliability of the methods will be discussed.

2.1 Available Literature

Literature examining BDD was found to be extremely limited. Only one paper, by Solís and Wang [42], was encountered that addresses BDD and its characteristics directly. Two papers where encountered that empirically investigated the results of BDD [17, 33]. Other papers that were encountered generally only mention BDD in passing or apply it without describing it thoroughly. A systematic literature review can be performed to identify characteristics of BDD and classify the available papers, which is currently being performed by A. Egbreghts at the University of Twente. For the aims of this paper however, investigating how BDD is applied in practice or literature may not be the optimal strategy, as will be explained in section 2.2. No literature was found that directly addressed testing in LCMDD or applying BDD principles to it, but one paper directly describes Mendix [19]. While Marín et al. addressed MBT of MDD processes this is not applicable to LCMDD systems as explained in section 1.1. There is extensive literature covering testing and agile testing. For this paper, Meyers et. al.'s "The art of software testing" [34] is used to define traditional testing and "More Agile Testing" [16] is used for agile testing. Van den Broek et. al. [44] describe the Scrum and Testing Framework, which establishes the relationship between Scrum and testing and whose structure is used for the LCMDD software quality framework proposed in this paper. The literature identification process is further elaborated on in Appendix B.

2.2 Research Methods

The research strategy used for analysing BDD is based on its nature. A preliminary search on BDD reveals that the inventor, Dan North, as well as numerous other writers on the subject, report that BDD is often used wrong in practice [7, 13, 22, 23, 29, 37]. In addition, North identified BDD as a Centred Community rather than a Bounded Community [37]. These terms stem from theology and differentiate two different ways of defining a set or community [20]. A Centred Community is defined by core principles or values, and membership of the set is defined by closeness to the core. This entails that Centred Communities necessarily have fuzzy edges. On the
other hand Bounded Communities define binary membership through boundary markers (figure 1). A useful analogy is the relationship between Agile and Scrum. While the Agile Manifesto[4] defines a clear set of core principles but no clear boundaries, Scrum’s characteristics are defined with high specificity through team roles, rituals, and artefacts. The extent to which an organisation can be said to practice ‘pure’ Scrum can be expressed with a high level of precision through comparison to the prescribed characteristics, while expressing to which extent an organisation is Agile is less trivial. Hence, the classification of BDD as a centered set by North entails that defining a set of external characteristics as is done in Solís and Wang[42] primarily through the available tools, will not result in the accurate identification of BDD principles. Additionally, because of the reported misuse of BDD in practice, empirical research may also be ineffective in identifying the core principles. Extensive qualitative research on the sources of BDD methodology could result in a set of principles that accurately define BDD, similar to the Agile Manifesto. Because of the scope of this research, a ‘working version’ of the BDD Manifesto is arrived at through triangulating core principles and relating them to BDD practices. To efficiently identify the BDD core principles, four thought leaders in the field of BDD have been selected as qualitative sources. Because core principles and thought leaders were evidently not clear from the start of the research, but emerged through triangulation, Grounded Theory[2] was used to approach the qualitative analysis. Since the nature of BDD and the source material makes Strauss’s more explicit data-analysis approach to Grounded theory less suited, Glaser’s more open approach[11] has been used. The focus of the analysis is teleological in nature: rather than describing BDD ontologically, the research aims to identify the intention behind BDD, i.e., “What is BDD for”. This is achieved by identifying which problems BDD aims to solve. The identification and selection of the thought leaders is elaborated in Appendix A.

To express the maturity of testing in the LCMDD field, the Testing Maturity Model (TMM) [5], based on the Capability Maturity Model[39], is used. The TMM was developed for assessing the maturity of the test processes in an organisation and providing targets on improving maturity, and hence offers a useful definition of stages that can be used to describe the testing maturity of the LCMDD field. For assessment of the maturity, the currently officially supported testing techniques on three major LCMDD platforms are reviewed. Since no literature was available to assess how the testing process is currently structured at companies that use LCMDD as their core development method, a case study has been performed on a company that uses the LCMDD Mendix Platform to develop their product, itself a Low-Code Model Based Platform as a Service. The case study is described in more detail in Appendix C.

Figure 1. Bounded vs. Centered Set

Core principles defining BDD become clear when North’s formal definition of BDD [29] is broken down:

BDD is a second-generation, outside-in, pull-based, multiple-stakeholder, multiple-scale, high-automation, agile methodology. It describes a cycle of interactions with well-defined outputs, resulting in the delivery of working, tested software-that-matters

Second-generation refers to BDD being derived from and based on the Agile Manifesto and the XP values as well as incorporating TDD, Continuous Integration (CI), Lean principles, DDD and being influenced by Neurolinguistic Programming and Systems Thinking. Outside-in refers to the fundamental role of business value in BDD, which requires the design process to start at high-level abstraction, with the product vision or purpose, and going down abstraction levels, moving through outcomes, feature sets, into individual features, stories, scenarios and finally code. Pull-based refers to the BDD principle Enough is Enough which will be elaborated in Adzic’s subsection, to Deliberate Discovery, which will be elaborated further in this section, and to Real Options, which will be elaborated in Chris Matt’s subsection. Multiple-Stakeholder refers again to the role of business value and of a wide perspective on the influence that core and incidental stakeholders have on a software project. This can be summarised by the BDD principle Deliver Stakeholder Value. Multiple-scope refers to the applicability of the BDD principles on multiple abstraction levels, from code level to system level behaviour and beyond. This is encapsulated in the BDD principle
It’s All Behaviour [29]. High automation refers to the integration of CI and refactoring in BDD which requires comprehensive automated functional tests, but also allows for easy regression, performance and penetration testing and in the end supports Real Options and Deliberate Discovery. Cycle of interactions refers to the iterative nature of Agile and the successive phases of increased specificity (going from outside-in) in BDD. Clearly-defined outputs refers to the concrete way of recording and linking specification of features with test methods through the Gherkin format of describing scenarios (Given, When, Then) and Feature Injection, which will be elaborated in Matt’s subsection. Additionally, the relation between specification and tests allow the test in BDD to become the Living Documentation of the system after implementation. In BDD, examples of behaviour become code tests and ultimately documentation, while Scenarios become acceptance tests and eventually regressions tests. This results in a much higher efficiency and effectiveness of specifications, tests and documentation, since in the traditional process specifications first describe desired behaviour, tests are then manually written that should verify the actual behaviour and after implementation documentation is written which should describe the behaviour of the system. In BDD these three separate artefacts become stages in the life-cycle of a single artefact with complete continuity between the phases. Software-that-matters again refers to the business value and to the definition of successful software that will be discussed in Adzic’s subsection.

The ineffectiveness of estimation and Planning Poker in Scrum noted by North, led to the development of the principles of Deliberate Discovery and the practice of Blink Estimation. The disadvantageous amount of time that individual scoring and estimation of all Stories in advance of a project takes, leads some to call this process Story Card Hell [31]. According to North, the current estimation approach is fractal, where a more fine-grained breakdown of requirements will always result in a higher estimate. North proposes an attitude of Deliberate Discovery, where a careful consideration is made which requirements should be broken down and which ones should not. This relates to the Real Options principle (which will be elaborated in Chris Matt’s subsection) of delaying decomposition of feature sets into features and stories until the last responsible moment. The application of the Theory of Constraints [25] on the insight that the software development process is a product development process instead of a production process [21] led to the conclusion that ignorance is the main constraint for the throughput of the software development process [6]. A factor that contributes to the unattainability of exact estimations on all scales in the development process, from individual Story to full Project length, is the second-order ignorance that is present at the start of any software project[6]. The presence of second-order ignorance throughout a software project supports it as a BDD principle which adheres to the multiple-scope applicability of the principle of It’s All Behaviour. This entails that the principle of Deliberate Discovery should be applied at every level and in every phase of a project. The application of Deliberate Discovery to estimation resulted in North developing Blink Estimation. Blink Estimation cultivates the experience of a diverse group of experts through intuition. Several rounds of discussion between Expert Estimators in a Blink Estimation session results in an order of magnitude estimation of the project length. This estimation is not as precise as traditional Scrum estimation, but in North’s experience must more accurate [36]. North concludes that breaking down requirements and specifica-

3.2 Chris Matts: Feature injection and Real Options

According to Matts, many projects start out as requests for functionality and then teams chase an unknown business value. This issue is addressed by the Feature Injection technique, developed by Matts. Feature Injection can be seen as an implementation of Deliberate Discovery (although it preceded it). It ensures that a team only builds features and projects that deliver business value, essentially implementing the BDD principles of Deliver stakeholder value and Outside-in. The three steps of Feature Injection are [31]:

1. Hunt the value
2. Inject the features
3. Spot the examples

To ensure that software delivers the expected business value, the value needs to be clearly defined and communicated. A pitfall of using User Stories for this, is the earlier addressed Story Card Hell. In lieu of this, Matts proposes that a company maintains a more abstract and explicit value model. To permeate the business value into lower abstraction levels, an alternative User Story structure is proposed by Matts [26]:

In order to <deliver some business benefit> as a <stakeholder>

I want <these features>

As the Deliberate Discovery principle dictates, taking down the abstraction level — i.e. converting business value from the value model into features and subsequently into Stories — should only be performed when necessary, to prevent the negative effects that have been described. Injecting features in this way, the business value scopes the outputs and the output scope the processes and inputs, working Outside-in and ensuring the project is Delivering stakeholder value. A tool to support the injection of features is Adzic’s Impact mapping, which will be discussed in his subsection.

The third step of Feature Injection, Spot the example, is directly related to the BDD story and Scenario definition using the Gherkin steps mentioned above. Injecting Features provide a set of ‘happy path’ scenarios that create the outputs which deliver business value. These scenarios should be expanded with ‘edge cases’ which cover the most likely variants of input. One technique to ensure the principle of Deliver stakeholder value and define the right edge case scenarios is to use of the Three Amigos. This is a concept that BDD incorporated from Acceptance-Test Driven Design (ATDD) [37]. The Three Amigos stand for Business Analysis, Development and Quality Assurance. In a Three Amigos meeting representatives of these three fields work together in breaking down requirements, where they further develop the Ubiquitous Language and resolve misunderstandings and assumptions about the specifications of a feature.

Matts calls the underlying principle behind Deliberate Discovery and Feature Injection ‘Real Options’. Matts has a long history and career as a Business Analyst in the financial domain, and Real Option is based on Financial Option Theory [32]. Financial Option Theory describes the optimal decision process and according to Matts the psychology of uncertainty and decision making (based on NLP and Cognitive Behavioural theory) why this optimal decision process is not followed and irrational decisions
are made. Three characteristics about options that follow from Financial option maths are[32]:

1. Options have value.
2. Options expire.
3. Never commit early unless you know why.

Based on these characteristics, Real Options concludes that avoiding early commitments affords flexibility in later choices[32]. This entails that commitment on any abstraction level should be deferred to the last responsible moment. This principle propagates through the abstraction layers of BDD in accordance with the multiple-scope principle It’s all Behaviour through Deliberate Discovery into Feature Injection and can also be recognised in the TDD principle of writing the absolute minimum necessary amount of code to make a test pass [1]. Developing software in an Agile way means staying flexible in dealing with changing requirements and keep the project Delivering stakeholder value, even if that value changes. Applying Real Options defers unnecessary commitments which keeps a software development process Agile and sustains the capability of producing software-that-matters.

3.3 Liz Keogh: Pixie Driven Development and Cynefin

Liz Keogh notes the long-held assumption that it is possible to get the requirements of a project right before the developers start coding [28]. Requirements often change during a software project – prompting the phrase "shooting at a moving target" – and misunderstandings are an inevitable side-effect of human communication. However, when requirements change during the project, or misunderstandings and assumptions are revealed, this often is a trigger to invest more in analysis upfront, to try to get the requirements the first time right [28]. This tendency is not in accordance with the principle of Deliberate Discovery and seems to trend back to the Waterfall method. Additionally, automated acceptance tests that were intentioned to verify high-level requirements are usually very brittle [28]. This means that they often needed to be updated, which costs a lot of time and makes it harder to make changes in the software system. It should be noted that the cause for every feature or story in a software project is the presence of a problem to be solved. Keogh draws the direct relationship between the need for analysis (and verification through testing) and the complexity of a problem. Instead of investing indiscriminately – either through analysis or acceptance test automation – in each feature and ending up in Story Card Hell, the amount of invested time and resources through discovery in a certain problem should be in proportion to its complexity. Because of this relationship, a useful tool for classifying and understanding the problems is Dave Snowden’s Cynefin Framework [28], according to Keogh. The Cynefin Framework introduces four domains that problems can fall in, with increasing complexity: obvious, complicated, complex and chaotic. A fifth domain is present in the centre, disorder, for when the type of the problem is unclear. Every software project contains problems that haven’t been faced before [45] – either by the team or at all – hence, software projects as a whole are located almost exclusively in the complicated or complex domains [28]. The range of complexity of individual features and stories is wider and spans all the domains. The risk that a problem poses for a project and the amount of Deliberate Discovery that is necessary can be accurately classified using a 5 point based scoring system based on the Cynefin domains, developed by Keogh. This system can be used in addition to Blink Estimation, and is relevant at all abstraction levels, in accordance with the principle It’s all behaviour. Cynefin scoring can guide Deliberate Discovery and avoid Story Card Hell because it makes clear what features should be examined thoroughly and which can be treated with minimum effort (in accordance with the principle Enough is Enough. According to Keogh, a factor that contributes to the creation of software that is not delivering stakeholder value – i.e. not software-that-matters – is the tendency of developers to quickly navigate to the solution space of a problem, before fully understanding the problem space [45]. A technique that she has proposed to help keep the conversation in problem space is Pixie Driven Development [27]. By imagining that instead of code, a magic pixie provides functionality, the prematurely abstraction level collapse can be prevented. Adhering to the It’s all behaviour multiple-scope principle, Pixie Driven Development can be applied at all abstraction levels. For example, at the level of Feature Injection, it can extend the discussion about the actual business value, at a Three Amigos meeting it can uncover implicit misunderstandings or assumptions and at the code level it helps implement useful code patterns like programming by contract or REST [27].

3.4 Gojko Adzic: Impact Mapping and the Software Quality Pyramid

Impact Mapping is a technique that Gojko Adzic developed based on Balic & Ottersen’s Effect Managing IT [12], which helps implement the principle of Deliver stakeholder value in the granularity increasing process of breaking down business value into a distinct feature set. According to Adzic, all software is based on assumptions [15] and the technique he developed helps validate assumptions and make them explicit [12]. Impact mapping starts with defining a clear and measurable business goal, in accordance with the principle Deliver stakeholder value. Subsequently, stakeholders, users and developers collectively extract the project scope from the business goal. This is achieved by consecutively answering the questions: Why are we doing this? (Identifying business value) Who are the people that can create the desired effect? (Identifying core and incidental stakeholders) How can the target group contribute or obstruct the desired effect? (Identifying stakeholder needs) What are the business activities or software capabilities that would support the needs of the stakeholders? (Identifying distinct Features)

Answering each question based on the results of the preceding question results in a 4-layered tree diagram, specifically a mind-map with measurable business value at its core. After the structure of the effect map is present, items on levels 2, 3 and 4 are annotated with 0 to 4 stars to represent a rough estimate of importance and Cynefin scores can be used to specify complexity. The principle of Deliberate Discovery should be observed to maintain focus: new Impact Maps should only be made when necessary (instead of creating maps for all business goals at the start of a project) and rather than expanding all nodes on a level immediately, a new level should only be expanded when necessary. Impact maps provide high organisation level visibility which prevents Story Card Hell and fosters Deliberate Discovery. Constructing an Impact Map with a diverse team helps build the Ubiquitous Language through relating business goals to software capabilities and helps prevent scope creep while supporting scope change and reprioritization.

According to Adzic, similar to the way that project goals are often vague and not universally communicated, software quality is rarely defined precisely, which contributes
to misunderstanding, misinterpretation and confusion about what a project needs to deliver [12]. The interpretation of quality is often delegated to a quality assurance role without much involvement from any business stakeholders. The implicit lack of a proper definition of software quality can also be seen in how software quality is measured. Often, teams measure the number of bugs and the code coverage [14] which leads to the reluctance to delete a test noted by North [35], as it lowers coverage. The flaw with measuring code coverage is the fact that a very thoroughly tested software system – with very high code coverage – but doesn’t Deliver stakeholder value is in the end not seen as possessing high quality. Furthermore, using the number of bugs to measure code quality was proven to be futile in 1990 by Dijkstra, as software testing shows the presence of bugs, not the absence [8]. The definition of quality used by Adzic reveals the errors in logic in how software quality is often approached today. Adzic uses Weinberg’s definition of quality as ‘value to someone (who matters)’. The number of bugs and the code coverage to not prove value in a direct way. The goal of a software project is not to have high code coverage and a low number of bugs. Adzic draws a parallel between certain software metrics and Maslow’s hierarchy of needs: often represented in a pyramid where human needs are ordered from basic survival needs to self-actualization. An important feature of Maslow’s pyramid is that a need that is lower on the pyramid is necessary for achieving the higher levels up to a certain point whereafter a higher availability of the need does not contribute any longer. The levels that Adzic defined for software quality, from lower on the pyramid to higher are:

1. Deployable/Functionally OK
2. Performant/Secure
3. Usable
4. Useful
5. Successful

With the software quality pyramid Adzic introduced a tool that operationalises and quantifies the principle of Enough is Enough and can be applied at multiple abstraction levels. In the spirit of Real Options, only the minimum responsible amount of a lower level on the pyramid should be implemented until the hindrance of achievement in a higher level causes the need for more of a lower level.

In addition of the use of the Software Quality Pyramid, Adzic advocates for the use of the Attribute Component Capability (ACC) Matrix, developed by James Whittaker at Google, which helps visualise risk across the components of a software system.

3.5 BDD: What went wrong?

As mentioned in section 2, BDD is often applied unsuccessfully in practice with such frequency that, among other things, it prompted Liz Keogh to giving the talk “10 Years of Doing BDD Wrong” at the Norfolk Developers Conference in 2014 [29], Gojko Adzic to giving the “BDD: Busting the Myths” talk at the Norwegian Developers Conference in 2012 [13] and numerous publications by Dan North [7, 37]. One of the causes for the misuse of BDD, pointed out by North, is approaching BDD as if it is a bounded set (as explained in section 2). As Keogh puts it: “BDD is not ‘do this and become successful’; it is ‘here are some ideas that we found useful in these contexts’.” [29] Another contributing factor, quoted by Adzic, is the attitude towards tools, which Adzic calls ‘insToolation’ [13]. The first thing prospective adopters of BDD often encounter are the tools that are built to support the process, which often gives rise to the expectation that after installation of the tool of choice the described benefits of BDD will automatically follow. As is described in this section, the core of BDD are the underlying principles. A tool that supports the processes that result from applying the principles does not actually cause the user to understand and apply the principles. While novice users often recognise the most outward and recognisable features of BDD, like the Gherkin format, they are equally often unaware of the intention behind the features. For example, the Gherkin steps that are used to define scenarios are part of creating a Ubiquitous Language that creates the possibility for effective communication between the development team and the stakeholders. Conversation can become a tool that helps convert second-order ignorance into first-order ignorance and to eventually resolve it as well as uncovering erroneous assumptions and miscommunication. As mentioned, ignorance is the constraint of the software development process, according to North [6]. Indiscriminate use of the tool that is the main bottleneck of the development process, drastically reduces the output of the process. Having conversations with business stakeholders about every piece of functionality, including the features that are in the Obvious Domain of Cynefin, quickly erodes the willingness of any non-technical stakeholders to collaborate with the development team, and if Deliberate Discovery is disregarded as well, it will lead to Story Card Hell. The absence of prioritisation in software development categorically creates overinvestment in unimportant stories and underinvestment in important ones [14]. As becomes clear from this example, BDD has a distinct domain in which it is applicable. According to Keogh, BDD should primarily be used in the Complex and Complicated Cynefin Domains [28]. In the context of the Agile Testing Matrix, this entails that BDD should be used for activities in quadrants 2 and 3, while application in quadrants 1 and 4 completely defeats its purpose and may be counterproductive. From the challenges and characteristics of the software development process, described by the BDD thought leaders, it follows that software development itself can be classified in the Complex domain. A notable implication of this classification is that in the Complex domain there are no Best-Practices, only emerging practices [28]. This can explain why practices from other complex domains like Finance (option pricing), Manufacturing (Lean principles, Theory of Constraints) and the many domains that Cynefin is applied in, are almost directly translatable and applicable to software development: it seems they aren’t specific to their apparent domains, but they may be specific to the Complex domain. The classification also entails that BDD itself is an emerging practice, and can not be thoughtlessly applied by anyone and produce results, it requires an extensive understanding of its principles and a critical attitude towards how to apply them in a given situation and finding a balance. This is encapsulated by Keogh: “BDD doesn’t have a brain, you’ll have to use your own” [45].

4 Maturity of testing in the LCMDD field

To assess the maturity of the LCMDD field in practice, the official support and integration of testing techniques and methods was examined for three LCMDD platforms (see table 1). Based on these results, LCMDD testing maturity can be classified. The Testing Maturity Model (TMM) [5] defines five levels of testing maturity:

- **Level 1** Initial
- **Level 2** Phase Definition
- **Level 3** Integration
- **Level 4** Management and Measurement
- **Level 5** Defect Prevention and Quality Control
Based on these levels, it is concluded that LCMDD maturity is nearing the end of Level 2 and moving towards level 3. Official support and integration of testing in LCMDD tools only very recently started taking shape, and no industry best practices seem to have developed yet.

5 Quality Assurance in the LCMDD process

In this section, the results from the case study are discussed and the current quality assurance (QA) approach is evaluated using the TMM [5], Agile Testing Matrix [16] and Adzic’s Software Quality Pyramid [14]. The interviewee characteristics and interview setup can be found in Appendix C.

5.1 Case study subject

eMagiz is a, start-up sized company that is based in Enschede, The Netherlands. It originated in 2009 from the logistics consulting firm ‘CAPE groep’ and is still closely related to that company. For example, both companies are housed in the same building. eMagiz’s core product is the integration platform as a service (iPaaS) ‘eMagiz’ which facilitates model driven data integration. The eMagiz application is developed on the Mendix Platform and the company has adopted an Agile development process through the application of Scrum methodology. The team uses a Scrum Sprint length of three weeks, after which a ‘feature freeze’ week follows where no new work should be started. Peer review and acceptance testing with an expert business user are the principle quality assurance activities. The development team at eMagiz has three Product Owners. One PO is the CEO of the company, who handles the budgetary and resource allocation aspects of the development. The CTO is also a PO and his focus is on the functionality of the system. Finally, the Channel Development Manager is a PO with a focus on the business perspective and representing the interest and wishes from the end users of eMagiz.

5.2 Evaluation of current QA approach

The current approach of eMagiz to assure quality in its software product, according to the conducted interviews, is to significantly rely on unstructured manual testing and confidence in the ability and judgement of the developers. Despite the decentralised nature and the low amount of automation in the testing process, the interviewed developers indicated that it results in an acceptable level of effectiveness. Some factors that may explain this are now discussed. The size of eMagiz, and its main customer CAPE groep, allows for easy and frequent communication, which provides the developers at eMagiz with extensive knowledge of their customer domain. In this environment, it is possible to leave a lot of decisions about the exact implementation of a feature and the necessity for testing up to the developers without much coordination, based on the BDD principles. In environments where developers are less aware of the needs of the end user – i.e. possess less domain knowledge – this approach results in a high risk of costly necessary revisions at the end of the project. From this, it can be concluded that company size is a contextual factor that influences what QA approaches are relevant for an organisation and in the end influences software quality. Another factor that may explain the effectiveness of the current process, which became clear in the interviews, is the use of the LCMDD platform. The use of the Mendix Platform made tests aimed at deployability, performance and security essentially unnecessary. Looking at the Agile Testing Matrix, this implies that the ‘Technology Facing’ quadrants Q1 and Q4 are largely covered by the use of the development platform. The current testing approach mainly takes place in the region of the border of Q2 and Q3. Similarly, using Adzic’s Software Quality Pyramid, it can be concluded that the bottom two levels are practically sufficiently filled by the use of the LCMDD platform, allowing for the primary focus of the testing process to be on the upper two levels: Usability and Usefulness. Therefore, the technology used for software development is also an important contextual factor. Characterising the current approach that was encountered in the case study using the maturity levels defined in the TMM, leads to the conclusion that the testing maturity of eMagiz is at TMM level 3. While testing is integrated into the software development process, the manual and decentralised nature of the current testing approach impedes progression towards levels 4 and 5. While in the interviews an acceptable level of effectiveness of the current approach was reported, there are some disadvantages to be noted. Because of the lack of a set of automated functional tests, regression tests need to be conducted manually, which is inefficient and leaves a significant risk for incomplete coverage of the system. From the case study in follows that some challenges for testing in LCMDD stem from the current difficulty of taking advantage of the automation possibilities for Q2 testing activities. The potential for automation in the context of LCMDD is hindered by the low maturity of the entire field established in section 4, both in the development of available testing techniques and the lack of an established testing methodology.

6 Developing the software quality framework

In section 5 it has been established that the current LCMDD QA process is underdeveloped, particularly the testing techniques and methods that take place in Quadrant Q2 of the Agile Testing Matrix. In section 3, it has been established that the application of BDD methods and concepts are particularly useful for Q2 testing, which makes the fit between BDD methodology and LCMDD clear. To structure the integration of BDD concepts described in section 3 into the LCMDD process, a framework is developed. The framework is an adaptation of the Van den Broek et al.’s Scrum and Testing Framework (STF) [44]. In this section, first the STF will be adapted to reflect the current LCMDD process. Next, the BDD concepts described in section 3 will be integrated.

6.1 Adaptation of the STF

The STF is designed for a software project with a distinct beginning and end. It includes steps for the project initiation and team selection, as well as integrating a ‘Sprint 0’ for training and setup purposes. For the case study subject eMagiz, these components are irrelevant, since they are in continuous development with a single team. The use of the functional anatomy map isn’t elaborated on by Van den Broek et al. and its relationship to the BDD principles is unclear, so it will be disregarded. As established in section 3 the Scrum estimation process will also not
be incorporated. Since the burn-down chart makes use of the individual Story estimations, its role will need to be re-evaluated. Since the aim of the BDD principles is to deliver working, tested software, phase 8 of the STF ‘End Game’ is expected to be very short and only consist of handling exceptional, unexpected testing and user acceptance testing. Because of the nature of the LCMDDD process, as established in section 5, testing activities from Agile Testing Quadrants Q1 and Q4 will not be integrated.

6.2 Integration of BDD concepts

The BDD practices are most relevant in the process of transforming high-abstraction business value into high-granularity BDD stories and scenario’s. The breakdown process starts with selecting a business goal where the Software Quality Pyramid can help guide business decisions about development and investment. Next, Impact Mapping the goal arrives at a feature set. Feature Injection defines individual features and breaking down features into BDD stories with scenario’s in a Three Amigo session provides the artefacts for start-up development. Deliberate Discovery guides when the next step in the process should be initiated and Pixie driven thinking can help keep the participants in solution space on all levels, from Feature Injection to implementation. Parallel to this drill-down process, Blink Estimation and Cynefin scoring can be used, along with the ACC Matrix, to estimate the project length and identify and visualise risk. After the transformation process, implementation consists of the traditional TDD Red/Green/Refactor, which means implementation of the least amount of code until the test passes (‘becomes green’), after which the code is refactored to incorporate design patterns. If DevOps is applied, there is no handoff to a separate maintenance organisation after implementation. The drill down process and the incorporation of BDD practices are represented visually in figure 2 which can be found in Appendix D.

7 Conclusion

In this paper a set of thought leaders that have defined BDD methodology have been identified and their core principles and views described. Because BDD is defined by a set of core principals instead of definitive characteristics, BDD is a Centered Community, rather than a Bounded community. It is concluded that BDD incorporates concepts from multiple domains, including Agile, eXtreme Programming, Neuro-Linguistic Programming, Domain Driven Design, Systems Thinking, Lean Manufacturing, Theory of Constraints and Complexity Theory. From the qualitative analysis that was performed it was concluded that core principles of BDD strive towards a development process that is pull based, outside in, and business value oriented through exploiting collaborative communication using a Ubiquitous Language to resolve second-order ignorance (which is regarded as the main constraint) and deliver software-that-matters. Furthermore, the central role that business value should play, according to BDD, helps scope the project, prioritise work on all abstraction levels and define what quality is. If these principles and the described techniques are made the most of, BDD may be a methodology that helps organisations deliver software-that-matters, under time and budget and bring a change to the abysmally low software project success numbers reported by the Standish Group in their Chaos reports.

From a conducted case study and the available testing techniques that were encountered, it follows that testing maturity in the Low-Code Model Driven Development (LCMDD) field is of TMM levels 2 to 3. Additionally, from the fact that LCMDD platforms inherently cover testing activities from Agile Testing Quadrants Q1 and Q4 it is concluded that BDD is a good fit for approaching software quality in LCMDD since it focusses on Q2 and Q3 testing. The order and application of BDD principles and practices into the LCMDDD process has been structured through the development of a framework. Through the analysis of BDD and quality assurance in LCMDDD, the research contributes to literature by expanding the limited amount of available academic work that directly addresses them. The presented state-of-the-practice review of the core principles of BDD and the developed software quality framework can contribute to practice by allowing organisations in and outside of the LCMDDD domain to implement a holistic quality assurance approach.

8 Discussion

The qualitative research in the paper on the core principles of BDD is necessarily of a limited scope. Future research could result in a collectively exhaustive description of BDD which could be encapsulated in the form of a BDD manifesto. Although two studies exist [33, 17], further empirical study of the benefits or disadvantages of BDD is needed, where the reported frequent misapplication of BDD needs to be taken into account. A historical examination of the origins and development of BDD will help separate the closely related methodologies of BDD, ATDD, Story-Test Driven Development (STDD) and Specification by Example (SBE). Further research into model testing techniques and empirical study of model testing processes will also be valuable to evaluate the assumptions that are made in this paper about the testing process in LCMDDD and the possibilities for future development. One avenue that may be examined is that of Domain Specific Language Testing, as proposed in Ten Buren [43]. The concepts and theories that are presented as core principles of BDD in this paper have a certain degree of validity due to the extensive experience of the thought leaders that propose them, but none of them have been scientifically rigorously evaluated. While some contextual factors are identified in this paper, no validated and complete conceptual model of factors that influence software quality was encountered which describes their exact operation, mutual influence and relation. Additionally, the framework that is developed in this paper is necessarily based on the assumptions addressed earlier and hence should be empirically validated.

9 Acknowledgments

First, I would like to thank Lucas Meertens for being my supervisor. Thanks for your time and challenging me to push the research further, offering valuable resources and your flexibility in shaping the research. Second, I want to thank eMagiz for allowing me to work at their premises once a week, the hospitality and the great atmosphere, a special thanks goes out to Alexander Willemsen and Bas Elainga who cleared time on their busy schedules for an interview and provided me with valuable insights in the current development and testing process. Finally I would like to thank Abigail Eghbreghts and Merijn Kleinreesink for sharing their knowledge and resources on BDD and MDD.

10 References

A three-step process was used to identify relevant literature for the literature review that was used to answer research questions 1 and 2. After exploratory searching, relevant articles were selected based on searches in Scopus and Google Scholar. Subsequently, backward and forward citing were used on the selected articles to identify related work. The keywords that were used for the search phrases relevant to each research question are recorded here to ensure reproducibility.

<table>
<thead>
<tr>
<th>Question</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BDD, Behavior/Behaviour Driven Development, Specification by Example, Dan North, Chris Matts, Liz Keogh, Gojko Adzic</td>
</tr>
<tr>
<td>2</td>
<td>Low code development platform, rapid app development, Mendix, Outsystems, Salesforce, testing</td>
</tr>
</tbody>
</table>

C.2 Interviewee Characteristics

From the seven-man team, three members were selected to be interviewed, based on their role, relevance and experience. The Scrum Master has 2.5 years experience at eMagiz in addition to two years work experience as a front-end developer and tester. Additionally the Scrum Master works part-time at the University of Twente and was also the supervisor of this research. The CTO has 10 years of experience at eMagiz, being involved from the very beginning. The third interviewed team member has 3.5 years of experience at eMagiz, and has worked for a multitude of other companies before working at eMagiz.

C.1 Interview Setup

The interviews were conducted face-to-face on the premises of eMagiz and had a total duration of 50-70 minutes. The interviews were semi-structured and subsequently identified the personal experience of the interviewee, the software development process at eMagiz and the tools and methods used as well as the problems the use of them aimed to solve, testing and testing goals at eMagiz, risks and their respective importance for eMagiz, and the available (future) possibilities for testing for eMagiz. To assess the familiarity with BDD concepts, the 16 most relevant concepts were printed on business cards and interviewees were asked to identify concepts they recognised. The main features of the answers of interviewees have been transcribed afterwards.

APPENDIX

A Selection of Thought Leaders

<table>
<thead>
<tr>
<th>Name</th>
<th>Involvement</th>
<th>Prolificacy</th>
<th>Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan North</td>
<td>Initial developer</td>
<td>Low</td>
<td>Y</td>
</tr>
<tr>
<td>Chris Matts</td>
<td>Colleague, initial collaborator, frequent contributor</td>
<td>High</td>
<td>Y</td>
</tr>
<tr>
<td>Liz Keogh</td>
<td>Colleague, initial collaborator, frequent contributor</td>
<td>High</td>
<td>Y</td>
</tr>
<tr>
<td>Gojko Adzic</td>
<td>Frequent collaborator and collaborator</td>
<td>High</td>
<td>Y</td>
</tr>
<tr>
<td>Dave Astels</td>
<td>Initial promotor</td>
<td>Low</td>
<td>N</td>
</tr>
</tbody>
</table>

For gathering data from the thought leaders several sources were used. The thought leaders have personal weblogs where they published their views (dannorth.net, theitriskmanager.wordpress.com, lizkeogh.com and gojko.net respectively), as well as some of their articles being published on InfoQ.com. Talks that the thought leaders had given at IT-conferences were retrieved using Youtube.com and Vimeo.com.

Figure 2. The proposed LCMDD Software Quality framework