RESEARCH ARTICLE OPEN ACCESS

A Study of Tools for Behavior-Driven Development

Meenakshi Panda, Saishraddha Ray
Gandhi Institute of Excellent Technocrats, Bhubaneswar,India
Shibani Institute of Technical Education, Bhubaneswar, Odisha, India

ABSTRACT
Behavior-Driven Development (BDD) has obtained a lot of attention in recent years from both research and practice points of view. As a new Agile development approach, it is aimed to increase the likelihood of success of a software project by adopting best practices and concepts from Test-Driven Development and Acceptance Test-Driven Developmentandcorrectingtheirdrawbacks.Therearealotoftools that were developed in the last few years to assist software developers in BDD. While this study describes underlying concepts and BDD itself, the main goal of the research is to develop criteria for identifying relevant tools which can be applied in BDD, evaluate and compare them and provide guidelines on which toolkit to choose in order to achieve success in a project. The research approach employed in this study is composed of reviewing relevant literature and analyzing current BDD toolkits for JVM-basedlanguages.

Categories and Subject Descriptors
D.2 [**Software**]: Software Engineering; D.2.9 [**Software Engineering**]: Management—productivity, programming teams, software configuration management
Keywords
Behavior-Driven Development, Test-Driven Development, Au- tomated Acceptance Testing

I. INTRODUCTION

Behavior-Driven Development (BDD) was introduced re- cently as one of the methods in Agile software develop- ment. BDD differs from other approaches in its family by describing a behavior of the system from the perspective of its stakeholders, at all levels of granularity [21]. BDD assures that focusing on such description of the behavior of the system gives better communication and produces a bigger asset for stakeholders when compared to other Agiledevelopmentmethods.Itwasoriginallydevelopedand describedby D. North in his post [28] as a response tothe issuesinTest-DrivenDevelopment(TDD).BDDisbasedon Test-DrivenDevelopmentandAcceptanceTest-DrivenDe- velopment [27]. D. Astels in [19] declared that eventhose peoplewhoapplyTDDalotdonotmakeuseofall benefits from TDD and important aspects of TDD are overlooked and simply ignored. He suggested that a big part of developersarefocusedonwritingverificationtests insteadof thinking in terms of behavior specifications. Taking into accountbehaviorspecificationsallowssoftwareengineersto thinkmoreclearlyabouteachbehavior,relyinglesson testingbyaclassorbyamethod,andhavingbetterexecutable documentation.

Full-scale Software Engineering Seminar 2015/16 RWTH AachenUniver- sity,Germany.

The paper is structured as follows. Section 2 describes the concepts of BDD and other inherited approaches which are needed to understand the requirements for BDD tools. Section 3 gives an overview of the research approach which was used to identify relevant tools for this study. In ad- dition, Section 3 defines diverse dimensions for comparing BDD tools, describes each analyzed toolkit in terms ofthose dimensionsandprovidestheoverallsummaryofcomparison. The last section gives theconclusions.

UNDERLYING CONCEPTS OFBDD

BDD is generally regarded as the evolution of TDD and ATDD. This section will briefly describe relevant aspects of TDD and ATDD in terms of BDD.

Test-DrivenDevelopment

Test-Driven Development is a development practice that involves writing tests before writing the code being tested. One should begin by writing a very small test for code that does not yet exist [21]. TDD is an evolutionary ap- proach that relies on very short development cycles and the agile practices of writing automated tests before writ- ing functional code, refactoring, and continuous integration [24]. Each development cycle consists of three steps: the creation of unit test, implementation, refactoring [23]. The aforementioned approach is named TDD since tests, writ- ten during the first steps of each iteration, drive the design and implementation. As a code base increases in size, more attention is consumed by the refactoring step. The design is constantly evolving and under constant review though it is not predetermined. This is emergent design at a granu- lar level and is one of the most significant by-products of Test-Driven Development[21].

The evaluation [26] by R. Jeffries and G. Melnik claims that the overall quality of a system in terms of the densityof defects improves, although the required effort often in- creases. A study described in [25] suggests that developers are able to produce a better design of a system with well- focused units with a help of TDD.

Acceptance Test-DrivenDevelopment

AcceptanceTest-DrivenDevelopment(ATDD)isonetype of TDD where the development process is driven by accep- tance tests that are used to represent stakeholders' require- ments [29]. M. Wynne and A. Hellesoy in [30] justify the name of acceptance tests as such tests express what the software needs to do in order for the stakeholder to find it acceptable. In the same book they state that in ATDD instead of a business stakeholder providing requirements to the developers without any discussion, the developer and stakeholder work together to write automated tests to sat- isfy thestakeholder.

ATDDassistsdevelopersinthecreationoftestcasesbased on initial requirements of a system. There is a set of tests or acceptance criteria that correspond to one specific requirement. One can say that a requirement is satisfied if all its associated tests or acceptance criteria are satisfied. In ATDD acceptance tests can be automated. ATDD empha- sizes automation of acceptance tests and the specification of customer-readable requirements through concrete exam- ples, which is also referred to as specification by example [18]. Automated acceptance tests encourage all people in- volved into the process to be focused on the aims of the software projects. Automated acceptance tests help your team to focus, ensuring the work you do each iteration is the most valuable thing you could possibly be doing[30].

TDD and ATDD are adopted widely by the industry be- cause they improve software quality and productivity [19] [25].

Behavior-DrivenDevelopment

ThemaingoalofBDDistogetexecutablespecificationsof asystem[28][19].DanNorthstatedthatthemainreasonfor introducing Behavior-Driven Development was the fact that Test-Driven Development was often perceived as a testing technique. He replaced the word ”test” in the name of TDD with ”behavior” in order to emphasize that TDD is about design, nottesting.

BDD has adopted the concept of a ubiquitous language fromDomain-DrivenDesign[21].Asuccessfulsoftwareproject requiresgoodcommunication,whichinturnrelies onashared language. Domain experts think and reason in terms of their domain language. Developers do the same, using concepts from the domain of software development. Analysts and de- velopers translate between these domains, mapping domain concepts to design. However, information can be lost in this translation, which causes different people to have different interpretations of concepts [27]. As Eric Evans describes in his book [22], many software projects suffer from low-quality communication between the domain experts and program- mers on the team. Tests written with a help of tools for BDD are usually defined using a language that business stakehold- ers canunderstand.

One of the key concepts of the BDD is involvement of all stakeholders which is possible via ubiquitous language. Business analysts write down behavioral requirements in the way that will also be understood by developers who later transform these requirements into executable tests.By workingtogethertowritethesetests,teammembersdecide what behavior they need to implement next. They learn how to describe that behavior in a common language that everyone understands[30].

Currently, the understanding of BDD is far from clear and unanimous.Thereisnoonewell-accepteddefinitionofBDD [29].

COMPARATIVESTUDYOFBEHAVIOR-DRIVEN DEVELOPMENTTOOLS

This section is aimed to compare BDD tools as well as to describe a research approach that was used to select certain frameworks from a huge number of tools that are present now. The final comparison can be found in table 1. The full support of a specific feature is marked by ”+”. By ”+/-” or ”+/- -” is marked partial support depending on the extent.

Approach for Identifying RelevantTools

The need to involve all stakeholders in the development process spawned a number of new tools which are aimed to assist all types of stakeholders in applying BDD. Particu- larly, new tools were needed to help non-technical people to read and understand acceptance tests, although the old toolscouldstillbeusedandmanystillcontinuetodo so.

The goal of this research is to create an approach to iden- tify the tools and frameworks which are relevant and can be applied successfully in BDD. BDD is just a technique which can be used without any tools and frameworks. This means that developers can try to utilize not only BDD spe- cific frameworks but also most of the tools for TDD. How- ever, TDD tools tend to be quite free-format and it will take a different amount of time and effort to benefit from those TDD tools in BDDcontext.

Support to some extent of ubiquitous language is themain criterion and BDD characteristic that was used to distin-guishrelevanttoolsforBDDinthisstudy.

Alotoftoolsfromdifferentlanguageswereanalyzeddur- ing the research. Due to the aforementioned selection ap- proach, the following frameworks were considered as those that cannot be used standalone as BDD frameworks: strictly unit-testing tools for all languages (JUnit [9], etc.), tools for mocking(EasyMock[6],Mockito[10],etc.),most UI-testing tools (Selenium [13]), frameworks for testing Web Services and databases. On the other hand, they are often combined with real BDDtools.

ThisstudyfocusesonBDDtoolsforJVM-basedprogram- ming languages (Java, Groovy, Scala) with a strong support of ubiquitous language. To determine the relevant BDD frameworks to compare, the Wikipedia list [1] was used as the initial source. The most frequently mentioned tools were selectedwithahelpofasearchbytagsonstackoverflow.com. The last step was to filter the frameworks for JVM-based languages since they can be directly and fairly compared. As a result the following tools were included in theanalysis: Concordion [3], Spock [15], Cucumber [4], JBehave [8] and easyb[5].Inaddition,Serenity(previouslyknown asThucydides)[14]frameworkwasconsideredbutnotincludedinthe comparison. It is less popular with the small community and the main benefit of it is reporting. Selected frameworks satisfy all main BDD requirements and match specificneeds of the study. Therefore, these frameworks were further com- pared.

Dimensions forComparison

DifferentdimensionforcomparingBDD frameworkswere found during the study. BDD is a technique which is per- fectly applicable at various levels. For instance, it can be ap- plied at the code/unit level and at the acceptance/integration level as well. Moreover, these usages are not exclusive and can becombined.

ComparisonBasedonaPrimaryTargetGroupOne dimension for comparison was inspired by J. BandwhodifferentiatesthefollowingflavorsofBDDtoolsbased on their origins and target groups in [20]:

1. Tools with a business readableoutput
2. Tools with a business readableinput

Frameworksfromthefirstcategoryareusuallyfocusedon thedevelopers.Allartifactsinvolvedareownedbythede- velopers and are typically code. This does not make such frameworks useless since responsible and committed devel-opersareoftenthemainstakeholdersinsuccessfulsoftware projects.Otherstakeholdersgetonlyreportswhichtheycan understand[20].Suchkindofframeworksisusuallyseenas a replacement/extension for TDD at a unit-testinglevel.

Tools from the second category (business readable input) try to widen the focus of the BDD process by enabling the bigger involvement of all other stakeholders: customers, business analysts, testers maybe even operations. This in- volvementis possible upfront, meaning before the develop- ershave done their work [20]. Such kind of tools is usually aimed atATDD.

Comparison Based on Support ofCharacteris- tics ofBDD

Another dimension for comparing tools comes from char- acteristics of BDD. The following main characteristics were identified during the study:

3. Ubiquitouslanguage

Thisconceptisanintegralpartof BDD.Therefor e,sup- portofthischaracteristicwasusedasaselectionc riterionfor toolsthatwerecomparedinthestudy.Creatingth eubiqui- touslanguageneedstoinvolveanyone(domaine xpertsand developers) who will use thelanguage.

The important point at this moment is to distinguish the ability of tools for creating a ubiquitous language based on the business domain and ability to use a predefined version of such language which is domain independent. BDD itself alsoincludesapredefinedsimpleubiquitouslangu ageforthe analysis process[29].

4. Automated AcceptanceTesting

All scenarios must be run automatically. This requires automatic import and analysis of acceptance criteria. The code responsible for the execution usually has to read the plaintextspecificationsandprocesstheminacorre sponding way. Such approach lets stakeholders have executable plain text scenarios. In this case, there also should be a standard mechanismofmappingscenariostotestcodewhic hexecutes them. However, scenarios can be simply defined directly in code.

5. Templates for plain text description of user stories and scenarios

Descriptions of features, user stories and scenarios cannot be done in an arbitrary form in BDD. All of them should follow the existing templates and guidelines.

Each user story describes an activity done by a user, clar- ifies a role of the user and which feature of a system allows the user to perform this activity. Moreover, each user story outlines the benefit which the user acquires after perform- ing the activity. Such template contributes to a clear way of representing features the system should support and why they should be supported by the system. In addition, such approach helps to understand what features are more im- portantby comparing the benefits which they provide. De- velopers may use this information to adjust their strategy, priorities, anddeadlines.

A scenario describes how the system that implements a feature should behave when it is in a specific state and an event happens. The outcome of the scenario is an action that changes the state of the system or produces a system output[29].

Comparison Based on Specific Features of Se- lectedTools

The last but not least dimension to compare BDD tools isbased on specific additional features that each tool provides.It is a good idea to combine other useful features with BDDones since such kind of tools can be used standalone to cover more cases without any need to integrate other frameworks.

The following specific features of analyzed frameworks were considered important:

6. Unit-testingfacilities.

There are some TDD techniques that may be helpful in BDD as well. For instance, mocking. It is not a good idea to make use of mocks in acceptance tests on a regular basis. Such tests are supposed to cover the whole system and to test each aspect ofit.

By mocking some parts of the system, you exclude them fromcoverage.However,therearecertaincaseswh enmock- ing is really appropriate: for instance, a module or compo- nent of a system can communicate with a 3rd party system. In this case, the scenario depends on the 3rd party system which is out of the control. Therefore, running such scenar- ios may be difficult and not stable, and the best option here is to mock or simulate that 3rd party system so that your application or product can stillbe tested.

Another useful application of mocking is to follow”test as soon as possible” technique. Developers can mock unimple- mented parts with predefined behavior and test small parts really early in the development cycle. This approach helps to spot all potential bugs during initial implementation. At thispointoftime,itisrequiredlessamountoftimeto inves- tigate and fix the issue than when you have a full complex and comprehensivemodule.

7. Facilities for testing Webapplications.

Web applications are extremely popular nowadays. Most of the new applications are developed for usage in Web. Moreover, there is an emerging strategy for applicationsoft- ware companies is to provide web access to software pre- viously distributed as local applications. Depending on the

Table 1: Comparison of BDD Tools

Support of Features	Cucumber	Concordion	Spock	JBehave	easyb
Business readable input	+	+	-	+	-
Business readable output	+	+	+	+	+
Creation of a ubiquitous language	-	+	-	-	-
Support of a predefined ubiquitous language	+	-	+	+	+/–
Automated acceptance tests	+	+	+	+	+
Plain text description of user stories and scenarios	+	+	+/–	+	-
Unit-testing facilities	-	-	+	+/– –	+/–
Facilities for testing Web applications	+	+	+	+	+

typeofapplication,itmayrequirethedevelopment ofanen- tirely different browser-based interface, or merely adapting an existing application to use different presentation technology [17]. Therefore, it is important for BDD tools to cover Web development and provide correspondingfacilities to make this processeasier.

There are a lot of high-level frameworks that allow the definition of acceptance tests in natural language. But when it comes to the technical implementation of the test cases, developers often have to use the rather low-level WebDriverAPI directly. Thus, it is important to consider to which extent modern BDD tools can be used for developing Web applications and how much effort it might require.
Functionalwebstoriesareapowerfulmechanismt overify the proper behavior of web applications from a user’s stand- point. Combining a framework that supports stories and scenarios with other tools for UI tests yields an easy way to deliver software more quickly andcollaboratively.

Comparison of SelectedTools

The following section describes each of analyzed frame- works independently in terms of developed criteria in the previous section.

Cucumber

Cucumber is definitely a framework with a business read- able input since it supports writing plain text user stories and scenarios which can be later utilized as a basis for cre- ating automated acceptance tests. Analysis of BDD-related questions on stackoverflow.com during this study confirms that Cucumber is one of the most popular and widely used frameworks of thistype.

Cucumber supports various readable report formats. The basic output prints the whole content of the feature which is not always necessary. Luckily, you can easily customize the output to match your needs. Cucumber has a set of built-in formatters. They allow you to visualize the output from your test run in different ways. There are formatters that produce HTML reports, formatters that produce JUnit XML for continuous integration servers like Jenkins, and many more. Moreover, there are a lot of custom formatters whicharedevelopedbyahugecommunityofdevelopers who use thisframework.

Cucumber does not allow you to create your own domain dependent ubiquitous language. However, it supports a pre- defined version of a ubiquitous language called Gherkin. It is plain text with a little extra structure. Gherkin is de- signed to be easy to learn by non-programmers, yet struc- tured enough to allow the concise description of examples toillustratebusinessrulesinmostreal-worlddomains.A Gherkin file is given its structure and meaning using a set of special keywords. There is an equivalent set of these key- words in each of the supported spoken languages [30]. This means that developers can write specifications not only in English but also in more than 60 other spoken languages and allows to widen the targetgroup.
Cucumber supports automated acceptance tests. In ad- dition, it is flexible in defining scenarios and it gives you an opportunity to write scenario outlines, share short setup steps or assertions. You can even call step definitions from other stepdefinitions.
Cucumbereasilyallowstotransformplain-textspecifica- tions into the code out of the box. However, it does have much to offer in terms of unit-testing due to its main aim andorigins.

Cucumber doesn't know how to talk to databases, web apps, or any external system. People install other libraries and use them in their step definitions and support code to connecttothoseexternalsystems[30].Forinstance ,youcan integrate Selenium or Capybara [2]. The latter framework poses special interest in combination with Cucumber since both of them are written in Ruby. This language fits BDD since it is natural to read. There is no specifically suited framework for UItesting.

Serenity has also a separate module for integrationwith Cucumber. It is an easy way to get incredible reports that are automatically generated for the BDDtests.

Concordion

Concordion is also a tool with a business readable output. DespitethefactthatConcordionrequiresbasicsof HTML,it is still a framework from the second category since it allows to write specifications in a highly customway.

Concordionalsoprovidesreadableoutputfromtes tswhich can beunderstood and used by all stakeholders. If all tests areexecutedthenframeworkproducesacomple tesetofcol- oredoutputHTMLfiles,whichdevelopersorthe irmanagers can publish on a web-server. There is also a possibility to use custom CSS or JavaScript, or include images or other resources,intheConcordionoutputbymeansofs impleex- tensions.Moreover,therearesomeexistingexte nsions.For instance,oneofthemaddsscreenshotstoConcor dionout- puttodiagnoseproblemsorimprovethedocume ntation.

Rather than forcing product owners to use a specially structured language for specification by example, Concor- dion lets you write the specifications in a normal language using paragraphs, tables, and proper punctuation. This makes them much more natural to read and write and helps everyone understand and agree about what a feature is sup- posed to do [3]. However, Concordian requires basic knowl-edge of HTML which can be a significant drawback. This framework also does not support predefined ubiquitous lan- guages such as Gherkin.

Concordion allows to write automated acceptance tests. It alsoprovidesabiglevelofflexibilityindoingitasC ucumber. Moreover, Concordion allows to have and edit plain text descriptions of stories andscenarios.

Concordion does not offer a lot in terms of unit-testing. It as well as Cucumber does not have any specific framework for UI testing that suits particularly well only for it. However, Concordion can be used to test Web applications since it is commonly used with Selenium.

Spock

Spock is a good example of tools with a strictly business readable output. It is not only as powerful as strictly unit- testing frameworks in terms of applicability at code/unit level, but it also supports writing specifications. Spock can not only fully replace JUnit but also provide the extended set of features with mocking and stubbingmechanisms.

Spock does not support the creation of a ubiquitous lan- guage. Moreover, it out of the box supports the concept of a ubiquitous language with some significant restrictions. For instance, developers have to mix the story descriptions and code. There is an extension called Pease that creates Spock tests from Gherkin specifications. With Pease, you are able to separate your requirements and your test code and still access the full power of the Spock framework [11].

Spock allows you to write automated acceptance tests. Spockcanbeusedasareplacementorextensionfor standard unit-testing frameworks, such as JUnit. Moreover, Spock has the widest range of features in terms of unit-testing. It is a complete testing framework with mocking, stubbing, and other helpfultechniques.

Spock provides simple integration and takes advantage of Geb framework. Geb is a browser automation framework writteninGroovybasedonSeleniumWebDriver.I tisaimed tomakeallcodeformodelingbehaviorofauseron UIpages concise and clear. Spock has also a great support for testing RESTfulAPIs.

JBehave

JBehaveissimilartoConcordionandC ucumbersinceit is a tool with business readable input. It lets execute text- baseduserstorieswithahelpofGherkinoritsown syntax.

JBehaveprovidesdifferentoutputformats.Fori nstance,it canprintatext- basedconsoleoutput,produceatext-based output file, an HTML file or an XMLfile.

JBehave does not provide an ability to define a ubiquitous language, but it supports the aforementioned Gherkin. In addition, you can make use of its own syntax to describe scenarios.

JBehave can be used to implement automated acceptance tests. It also lets transform plain-text specifications into the code out of the box. JBehave has limited unit-testing facilities. For instance, this tool bundles a mocking framework known as a Mini- mock. JBehave has an extension called JBehaveWeb which providessupportforweb-relatedaccessorfunctionality.JBe- have integration with Selenium and WebDriverAPIs aims to facilitate common tasks. Amongst these, one of the most common is the management of the lifecycle, e.g. starting and stopping the browser[8].

JBehave works well with Serenity since there is a sepa- rate module in Serenity for combining with JBehave. Seren- ity uses simple conventions to make it easier to get started writing and implementing Serenity stories and reports on both JBehave and Serenity steps, which can be seamlessly combined in the same class, or placed in separate classes, depending on your preferences.

Easyb

Easyb is one more example of tools with an only business readable output. It is similar to Spock in this respect.

Easyb does not allow to create a ubiquitous language. This framework provides the worst support of theconcept ofubiquitouslanguagesincethecodeandspecificationare mixedtogetherandtherewasnopluginorextensiontosup-port,forinstance,Gherkinoranypredefinedlanguageatthe momentofstudy.However,thecodewithgiven/when/then sectionshelpsallstakeholderstogetinsightaboutthetested scenario easilyenough.

Easyb provides functionality forautomatedacceptancetests, but there is no way to support plaintextdescriptions.Easyb has fewer features at the unit-testinglevelthanSpock, but more than other analyzed frameworks.Italsocan be used together with Selenium [13], Selenide[12]andTellurium[16].Moreover,easyb canbecombinedwithFEST [7]frameworktoenabletestingofSwing-basedapplications.

TelluriumisbuiltonUImoduleconcept,whichmakesitpos- sible to write reusable and easy to maintain tests against the dynamicRIAbasedwebapplications.Selenideissimpleand powerful in use wrapper-library over Selenium intended to shortthelinesofcodetomakethewholetestsmorereadable and understandable. There is a special plug-in for working withdatabases.

Summary ofComparison

AllanalyzedtoolsaresuitableforBDDbuttheyareaimed atdifferentlevels.Spockandeasybarefocusedontheunit-testinglevel,whileJBehave,Concordion,andCucumberare more suitable for acceptance/integrationtesting.

Only Concordion supports to some extent creation of a specific ubiquitous language for a project. JBehave, Cucum- ber support predefined ubiquitous languages, while Spock and easybhave some significant restrictions in this regard. For instance, developers mix the story description and corresponding code using these tools. Even despite the fact that you can use a plain text to define all method names, story and code are very tightly coupled and reside in one file.

All analyzed frameworks support automated acceptance tests. However, Concordion, JBehave, and Cucumber have more ways to define the scenarios. These tools also provide a clear separation between the code and scenarios allowing to define user stories and scenarios in plain text. Hence, these tools are more flexible and powerful for this particular task. Spock has the aforementioned Pease extension which provides the ability to define scenarios in Gherkin, but there is no such solution for easyb.

Both Spock and easyb have much more to offer than Cu- cumber, Concordion, and JBehave from the unit-testing point of view. However, there are a lot of standalone specific tools such as Mockito, EasyMock which can be integrated into all analyzed frameworks to add needed functionality.

Other toolkits that can be easily combined with analyzed frameworks were mentioned per each framework. Those tools were selected by review of the literature, tutorials, and documentation.

II. CONCLUSIONS

BDD inherits main concepts from TDD and automated acceptance testing augmenting them with other ones such as ubiquitous language. This combination is aimed to make use of all benefits provided by each inherited approach and address their drawbacks. BDD can be adapted and applied at various levels of development. It puts the strong focus on behaviorinsteadofstructureateachlevel.BDDch angesthe way all stakeholders think about testing. Its main goal to verify what a tested object does and not what the internal structure of the object is. This difference makes a huge impact on the overall development process since behavior is much more significant than the internalstructure.

The main intends of the study were to provide all under- lying concepts of BDD, develop the research approach for identifying relevant tools for applying BDD and to compare the selected tools for JVM-based languages from different perspectives. One of the most important features of BDD is involvement of all stakeholders in the development process. Therefore, the special attention was paid to the concept of the ubiquitous language. Support to some extent of a pre- defined ubiquitous language or creation of a new domain specific one was chosen as the criterion to select relevant tools for comparison. The study defines three dimensions for comparing BDD frameworks: based on a target group, on the support of characteristics of BDD and based on spe- cific features of selectedtools.

The results of the performed comparison indicate that there is a strong support of main BDD concepts by analyzed toolswhichmakesBDDpossiblewithJVM-basedlanguages. However, the study also shows that tools with better sup- port of unit-testing facilities usually require some tuning to pose an interest for all stakeholders. All analyzed tools have a nice integration with a vast varietyof other tools. This is crucial since it enables applying BDD for different kinds of applications. For instance, there is a set of frameworks for each analyzed tool that makes possible BDD for Web applications.

REFERENCES

[1] Behavior-driven development. https://en.wikipedia.org/wiki/Behavior-driven_development. Retrieved November 20,2015.

[2] Capybara. https://rubygems.org/gems/capybara. Retrieved December 2,2015.

[3] Concordion. http://concordion.org/.Retrieved November 23,2015.

[4] Cucumber. https://cucumber.io/. Retrieved December 9,2015.

[5] Easyb. http://easyb.org/. Retrieved December 9, 2015.

[6] Easymock. http://easymock.org/.Retrieved December 9,2015.

[7] Fest. https://code.google.com/p/fest/. Retrieved December 2,2015.

[8] Jbehave. http://jbehave.org/. Retrieved December 2,2015.

[9] Junit.http://junit.org/.RetrievedDecember9, 2015.

[10] Mockito. http://mockito.org/. Retrieved December 9,2015.

[11] Pease. http://pease.github.io/.Retrieved December 1,2015.

[12] Selenide. http://selenide.org/. Retrieved December 2,2015.

[13] Selenium. http://www.seleniumhq.org/. Retrieved December 2,2015.

[14] Serenity bdd. http://www.thucydides.info/.Retrieved December 9,2015.

[15] Spock. https://code.google.com/p/spock/. Retrieved December 2,2015.

[16] Tellurium. https://code.google.com/p/aost/. Retrieved December 2,2015.

[17] Web application. https://en.wikipedia.org/wiki/Web_applicationt. Retrieved December 2,2015.

[18] G. Adzic. Specification by Example: How Successful Teams Deliver the Right Software. Manning Publications,2011.

[19] D. Astels. A new look at test-driven development. Technical report,2005.

[20] J. Bandi. Classifying bdd tools.http://blog.jonasbandi.net/2010/03/classifying-bdd-tools-unit-test-driven.html, 2010.

[21] D. Chelimsky, D. Astel, B. Helmkamp, D.North,Z. Dennis, and A. Hellesoy.The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends. Pragmatic Bookshelf, 2010.

[22] E. Evans and M. Fowler. Domain-Driven Design. Addison-Wesley Publishing Company,2004.

[23] M. Fowler, K. Beck, J. Brant, W. Opdyke,andD. Roberts. Refactoring: Improving the Design of Existing Code. Addison-Wesley Publishing Company, 1999.

[24] D. Janzen and D. H. Saiedian. Test-driven development: concepts, taxonomy, and future directions. Computer, 38(9):43–50, September2005.

[25] D. Janzen and D. H. Saiedian. Does test-driven development really improve software designquality? Software, IEEE, 25(2):77–84, March-April2008.

[26] R. Jeffries and G. Melnik. Guest editors introduction: Tdd - the art of fearless programming. Software, IEEE, 24(3):24–30, May-June2007.

[27] J. H. Lopes. Evaluation of behavior-driven development. Master's thesis, Faculty EEMCS, Delft University of Technology,2012.

[28] D. North. Introducing bdd.http://dannorth.net/introducing-bdd/, 2006. Retrieved November 1,2015.

[29] C. Solis and X. Wang. A study of the characteristics of behaviour driven development. In Proceedings of the 7th EUROMICRO Conference on Software Engineering and Advanced Applications, pages 383–387,2011.

[30] M. Wynne and A. Hellesoy. The Cucumber Book: Behaviour-Driven Development for Testers and Developers (Pragmatic Programmers). The Pragmatic Bookshelf,2012.