

# Implementing Behavior Driven Development in an Open Source ERP

Rogério Atem de Carvalho, Fernando Luiz de Carvalho e Silva, Rodrigo Soares  
Manhães, Gabriel Lima de Oliveira  
Federal Fluminense Institute, NSI, R. Dr. Siqueira 273, Sala F104, Campos, Brazil  
{ratem, fernando.carvalho, rmanhaes, gabriel.oliveira}@iff.edu.br

**Abstract.** A typical problem in Software Engineering is how to guarantee that all system's requirements are correctly implemented through source code. Traditionally, requirement tracing is a manual task comprised of keeping links from requirements to source code, going through different modeling artifacts, including models. However, these techniques cannot guarantee that requirements are always correctly implemented by source code. Aiming at solving this problem, Behavior-Driven Development (BDD) is a specification technique that automatically checks if all functional requirements are treated properly by source code through the connection of the textual description of requirements to automated tests. Given that for Enterprise Information Systems, requirements are usually identified by analyzing business process models, and these processes are implemented through workflows, connecting workflows to automated tests through BDD specifications can provide automated requirements traceability. The aim of this paper is to briefly present this proposal and show how it was implemented for the open source ERP5 system.

**Keywords:** Behavior Driven Development, Automated Tests, ERP.

## 1 Introduction

Requirement tracing is a typical Software Engineering matter that can be described as how to guarantee that all functional requirements are implemented correctly by the source code. Many techniques were developed to address requirement tracing, such as the use of the Requirements Traceability Matrix [1], which keeps track of the source code elements that realize the requirements. These artifacts require that developers determine by hand, for each functional requirement, which classes collaborate to implement it. This technique is time consuming and error prone, given that the traceability work is done manually and through the introduction of an additional artifact - the matrix. The ideal way of keeping track of requirements is directly linking them to source code.

Behavior-Driven Development (BDD) [2] is a specification technique that links functional requirements to source code, through the connection of the textual representation of the requirements to automated tests. BDD extends Test-Driven Development (TDD) [3], which in turn is a technique that consists of writing test cases for every programming task, before these implementations are performed. TDD is intended to achieve the correct solution that exactly matches the business problem

at unit and integration levels [4] – while BDD takes care of the acceptance level, where requirements reside.

For Enterprise Information Systems, identify requirements by analyzing business process models facilitates the participation of the domain experts [5]. Business processes, in turn, are implemented through workflows. Therefore, connecting workflows to automated tests through BDD specifications can provide an end to end traceability of requirements, as well as facilitate communication among developers and domain experts. This paper aims at presenting the application of BDD to the Open Source ERP (FOS-ERP) ERP5 [6], as a way of showing how this technique can drive the development of high quality business systems. The proposal here presented is a simpler variation of the Business Language Driven Development (BLDD) technique [7], which extends BDD by using business process models to drive the testing process. This paper is organized as follows: after this introduction, BDD main characteristics are briefly presented; followed by the description of the application of it to ERP5, and finally conclusive remarks are commented.

## 2 Introducing BDD

BDD starts by defining requirements using specific keywords that tag the type of the sentence, indicating how it is going to be treated in the subsequent development phases. These descriptions are written in an Ubiquitous Language (UL), which is a language structured around the domain model and used by all team members to connect all the activities of the team with the software [8]. In summary, requirements are described sets of Given-When-Then constructs (GWT) [9]:

Given a **Context** (or a system **State**)  
When an **Event** happens (or an user **Action**)  
Then an **Action** is taken (or a system **Reaction**)

From this point onwards, a story runner maps the natural language sentences into the underlying programming language equivalent calls, while keeping the same abstraction level. The next step is to write acceptance tests to the generated steps, pieces of code that excite the source code that implements the system. By wrapping all implementation code with tests, which in turn are automatically tied to the business requirements using the GWT constructs, the need for requirement documentation is fulfilled, having the advantage of making the whole system verifiable automatically at any time. Therefore, the use of BDD allows reducing the risks and effort to implement a given change in an information system; hence the system can be continuously improved without making the cost of change grow exponentially, avoiding Boehm's Cost of Change Curve phenomena [10]. Moreover, the *Then* constructs represent the tests' acceptance criteria, stating an objective method to validate if a given requirement is *done*.

According to [7], BDD provides an automated and cost effective way of keeping requirements traceability, addressing each of the challenges introduced by Kannenberg and Saiedian [11]:

- Cost: it is able to provide full tracing in a way that is even cheaper than Value-based requirement tracing [12]. Requirements are tied to tests, so that if tests return non-expected values or simply are not implemented, the tool will automatically point out the problem.
- Managing change: there is no need to impose “strong discipline in maintaining the accuracy of traceability”, instead, BDD tools make all the necessary checks automatically. Whenever a requirement changes, the tests will not run until the code is also changed accordingly. Moreover, by changing a requirement and immediately running a complete build, errors will pop-up in specific places where the system must be changed, easing effort estimation.
- Different stakeholder viewpoints: standards provide no guidance to traceability [11], through BDD, it is possible for any stakeholder to check system consistency, since it is based on executable documentation. Even end users can push a button and check its higher level error messages.
- Organizational problems: according to [11], “the easiest way to correct organizational problems related to traceability is through the use of policy and training”. BDD provides a proper policy for traceability because it enforces the connection between code and requirements.
- Poor tool support: the correct combination and use of BDD and TDD testing tools results in a development environment which monitors and alerts about inconsistencies introduced in the code.

### 3 BDD in ERP5

ERP5 is an open source ERP that offers an enterprise-wide system based on the open source Zope platform. This platform integrates an object database, a workflow engine, a content management framework, and rapid GUI scripting [6]. ERP5 has different variants for industry, governments, and services, including CRM, MRP, SCM, Sales, Payroll, and others modules.

The motivation to use BDD in an ERP that works on top of a workflow engine is to connect tests directly to the workflows, thus joining the business logic represented by the workflows to the requirements comprehension represented by automated tests, or, in other words, checking the validity of the mapping from the business experts’ definitions to the developers’ definitions.

The first step to use BDD in ERP5 was to map GWT constructs to state-based business process models, given that ERP5’s workflow engine is state-based. This task was facilitated by the fact that GWT constructs represent a state machine, connecting the human concept of cause and effect, to the software concept of input-process-output. According to [13], this convention “is simply a state transition, and that BDD is really just a way to describe a finite state machine. Clearly *Given* is the current state of the system to be explored, *When* describes an event or stimulus to that system, and *Then* describes the resulting state of the system.” Therefore, the set of all scenarios of a given business requirement can be represented by a Finite State Machine.

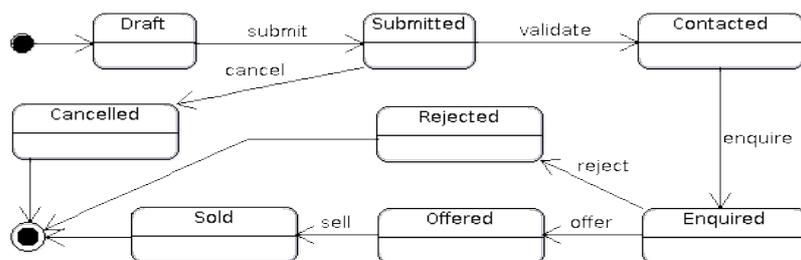
Following this reasoning, and using the mappings from the most common business process patterns to GWT constructs provided by [14], it was developed an ERP5

module which connects the system’s workflows to their automated tests through detailed definitions described using GWT.

The sequence of steps for using the Feature Module for implementing a given business process is as follows:

- a) Represent the business process in the system using a state-based workflow.
- b) Describe the feature’s goal and its business value.
- c) Define each scenario (state change) supplying examples.
- d) Using the values defined in the *Given* and *When* clauses, write the automated tests. Tests success will be checked by comparing their results to the acceptance criteria defined in the *Then* tags.
- e) Write the source code that implements the workflow, and that will be excited by the tests.
- f) Run the tests and check for their correctness.

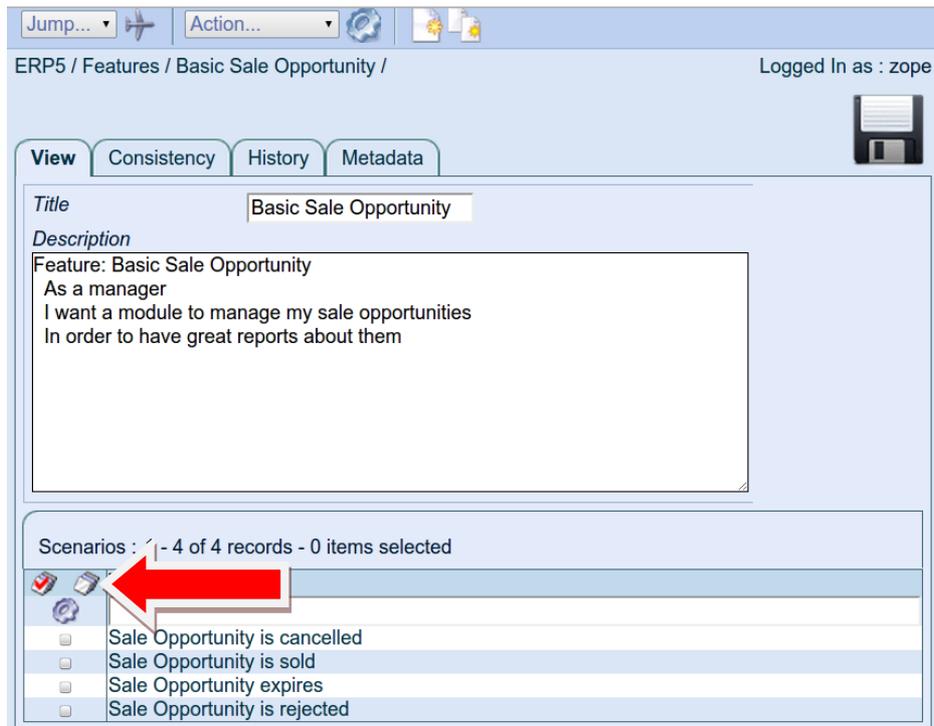
In order to exemplify this process, the ERP5’s Basic Sale Opportunity business process, showed in Fig.1, is used.



**Fig 1.:** ERP5’s Basic Sales Opportunity process.

Fig. 2 shows the Module’s main form used to describe the feature, as well as to list its scenarios. Features are described for the sake of defining their business value. The arrow in the figure indicates the icon to create a new scenario.

Following BDD’s incremental and test-first process, the following sequence of steps is performed for each state change on the workflow: (i) create a scenario defining the example values for the tests and the acceptance criteria, (ii) write automated acceptance tests, (iii) write the equivalent source code, (iv) run the tests until there are successful. In order to support this incremental process, the Feature Module allows the execution of each scenario independently. The result of running the scenario “Sale Opportunity is Submitted” is shown in Fig. 3. The first four lines prepare the environment for the test, creating the workflow object, while the fifth and the sixth lines are responsible, respectively, for firing the transition, and checking if the workflow is in the expected state - *submitted*.

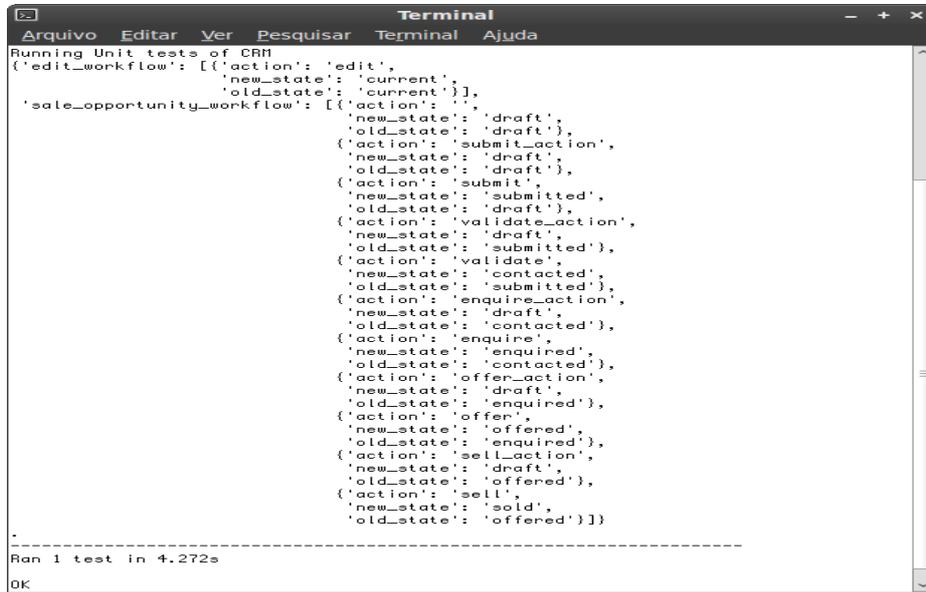


**Fig 2.** Creating the Feature and defining its value.



**Fig. 3** Results of running the scenario “Sale Opportunity is Submitted.” The red arrow marks the firing of the transition and the checking of the workflow state.

After defining all scenarios, it is possible to run the workflow’s different paths. Fig. 4 shows the “happy path” of the sale opportunity, ending in the sold state.



```
Terminal
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
Running Unit tests of CRM
{'edit_workflow': [{'action': 'edit',
                    'new_state': 'current',
                    'old_state': 'current'}],
 'sale_opportunity_workflow': [{'action': '',
                                'new_state': 'draft',
                                'old_state': 'draft'},
                               {'action': 'submit_action',
                                'new_state': 'draft',
                                'old_state': 'draft'},
                               {'action': 'submit',
                                'new_state': 'submitted',
                                'old_state': 'draft'},
                               {'action': 'validate_action',
                                'new_state': 'draft',
                                'old_state': 'submitted'},
                               {'action': 'validate',
                                'new_state': 'contacted',
                                'old_state': 'submitted'},
                               {'action': 'enquire_action',
                                'new_state': 'draft',
                                'old_state': 'contacted'},
                               {'action': 'enquire',
                                'new_state': 'enquired',
                                'old_state': 'contacted'},
                               {'action': 'offer_action',
                                'new_state': 'draft',
                                'old_state': 'enquired'},
                               {'action': 'offer',
                                'new_state': 'offered',
                                'old_state': 'enquired'},
                               {'action': 'sell_action',
                                'new_state': 'draft',
                                'old_state': 'offered'},
                               {'action': 'sell',
                                'new_state': 'sold',
                                'old_state': 'offered'}}]}
-----
Ran 1 test in 4.272s
OK
```

**Fig. 4** Sales opportunity's happy path tests successfully ran.

Fig. 5 shows what happens if the test fails for a given scenario, for demonstration purposes, a wrong expected state called “not enquired” was used as acceptance criterion, while the source code changes the state to “enquired”, raising an assertion error. In that way it is possible to identify which specific part of the workflow is not in accordance to the acceptance criteria, and even a business expert can identify what is the problem by checking the test log. If other errors were present they would be raised one by one.

Although the workflow logic already provides, by nature, the current state (Given), the event and its conditions (When), and the new state (Then), the textual representation of the workflow provided by the GWT constructs is necessary for two reasons: (i) provide example values for the automated texts and (ii) describe internal changes in the information system, such as the creation of objects.

```
Terminal
Arquivo Editar Ver Pesquisar Terminal Ajuda
Running Unit tests of CRM
F{'edit_workflow': [{'action': 'edit',
                    'new_state': 'current',
                    'old_state': 'current'}],
 'sale_opportunity_workflow': [{'action': '',
                               'new_state': 'draft',
                               'old_state': 'draft'},
                              {'action': 'submit_action',
                               'new_state': 'draft',
                               'old_state': 'draft'},
                              {'action': 'submit',
                               'new_state': 'submitted',
                               'old_state': 'draft'},
                              {'action': 'validate_action',
                               'new_state': 'draft',
                               'old_state': 'submitted'},
                              {'action': 'validate',
                               'new_state': 'contacted',
                               'old_state': 'submitted'},
                              {'action': 'enquire_action',
                               'new_state': 'draft',
                               'old_state': 'contacted'},
                              {'action': 'enquire',
                               'new_state': 'enquired',
                               'old_state': 'contacted'}}]}

-----
FAIL: test_SaleOpportunitySold (testCRM.TestCRM)
-----
Traceback (most recent call last):
  File "/home/ciberglo/erp5/erp5.buildout/parts/products-erp5/ERP5Type/tests/backportUnitest.py", line 151, in run
    testMethod()
  File "/home/ciberglo/erp5/erp5.buildout/parts/products-erp5/ERP5/tests/testCRM.py", line 251, in test_SaleOpportunitySold
    self.assertEqual('not_enquired', so.getSimulationState())
AssertionError: 'not_enquired' != 'enquired'

-----
Ran 1 test in 4.440s
FAILED (failures=1)
```

**Fig. 5** Test failed while trying to transit to a wrong state.

#### 4 Conclusions

This paper aimed at briefly presenting the first implementation of the BDD technique into an ERP, in this case, the open source ERP5. Moreover, it showed an ERP5 module specifically developed to integrate the BDD's features with ERP5's workflows. In that way, the goal of joining the advantages of using BDD to keep track of all requirements while using an underlying workflow engine was reached.

A future direction is making the Feature Module able of, by checking a given workflow's configuration, generating the GWT constructs, guaranteeing that all transitions will be implemented. By changing the generation of GWT constructs to a

specific language that reflects a state-based workflow, as proposed by [7], it is possible to use a single a unique language for describing both business process and testing details.

## References

1. Carlos, T. Requirements Traceability Matrix - RTM. <http://www.pmhut.com/requirements-traceability-matrix-rtm>
2. North, D.: Introducing Behavior-Driven Development. Available from <http://dannorth.net/introducing-bdd>
3. Beck, K. Test-Driven Development by Example. Addison-Wesley (2003)
4. Koskela, L. Test-Driven: TDD and Acceptance TDD for Java Developers. Manning, (2007)
5. Heinecke, A., Bruckmann, T., Griebe, T., Gruhn, V.: Generating Test Plans for Acceptance Tests from UML Activity Diagrams. 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems. p. 57-66, (2010)
6. Smets-Solanes, J. and Carvalho, R. A. ERP5: A Next-Generation, Open-Source ERP Architecture. IEEE IT Professional, Vol. 5, n. 4, 38-44, (2003).
7. Carvalho, R. A., Carvalho e Silva, F. L., Manhaes, R. S.: Business Language Driven Development: Joining Business Process Models to Automated Tests. In: V IFIP International Conference on Research and Practical Issues of Enterprise Information Systems, Lecture Notes in Business Information Processing, (2011)
8. Evans, E.: Domain-Driven Design - Tackling Complexity in the Heart of Software, Addison-Wesley, (2004)
9. Behavior Driven Development.org: Behavior Driven Development. Available from <http://behaviour-driven.org>
10. Carvalho, R. A., Johansson, B., Manhaes, R. S.: Agile Software Development for Customizing ERPs. In: Sudhaman Parthasarathy. (Org.). Enterprise Information Systems and Implementing IT Infrastructures: Challenges and Issues. Hershey, USA: Information Science Reference, IGI Global, p. 20-39, (2010)
11. Kannenberg, A., Saiedian, H. Why Software Requirements Traceability Remains a Challenge. CrossTalk, Vol. 22, No. 5, p. 14-21, (2009)
12. Heindl, M., and Stefan, B.: A Case Study on Value-Based Requirements Tracing. Proc. of the 10th European Software Engineering Conference. Lisbon, Portugal, p. 60-69, (2005)
13. Martin, R. C.: The Truth About BDD. Available from <http://blog.objectmentor.com/articles/2008/11/27/the-truth-about-bdd>
14. Carvalho, R. A. Silva, F. L. C. Manhaes, R. S.: Mapping Business Process Modeling constructs to Behavior Driven Development Ubiquitous Language. arXiv: 1006.4892v1 [cs.SE] (2010)