WHITEPAPER

BEHAVIOR-DRIVEN DEVELOPMENT AND BRMS

Deliver the right product to market faster with business rules management and behavior-driven development (BDD)

Justin Holmes

EXECUTIVE SUMMARY

Business experts and application developers in leading organizations need to be able to model, automate, measure, and improve their critical processes and policies. Red Hat® JBoss® BRMS makes this possible with fully integrated business rules management, business process management (BPM), and complex event processing (CEP).

Accommodating change and avoiding feature decay when designing and developing applications is necessary for aligning systems with the requirements brought on by competitive pressures, increased demands for regulatory compliance, and technological advancements. As leading companies strive to keep pace with emergent technologies, the fast and frequent release of software may be achieved using what some agilists call the "three amigos": test-driven development (TDD), behavior-driven development (BDD) and domain-driven design (DDD). These three concepts comprise an approach to BRMS development ensuring predictable and productive processes.

By introducing these application life-cycle management frameworks and illustrating how they may be harnessed with BRMS applications, Red Hat Consulting helps its clients realize business value from agile approaches coupled with proven architectures. Red Hat Consulting helps clients improve speed-to-market, reduce risk, and sustain quality.

This whitepaper offers an introduction to these approaches and explains how enterprise organizations can keep pace with the demands of our rapid global marketplace with a combination of software and professional services from Red Hat.

"Test-driven development (TDD), behavior-driven development (BDD), and domain-driven development (DDD) comprise an approach to BRMS development ensuring predictable and productive processes."

JUSTIN HOLMES
SENIOR CONSULTANT AND SERVICES DELIVERY MANAGER, RED HAT CONSULTING

THE "THREE AMIGOS"

Gojko Adzik is the author of "Specification By Example", a book explaining applications of the behavior-driven development methodology. Adzik introduced "the three amigos" in a presentation at the DDD eXchange Conference in 2010.[1] His thesis demonstrates the basis for and advantages of test-driven development. Test-driven development, when complimented by domain-driven design and behavior-driven development, allows a customer-centric approach to sustainable agile practice.

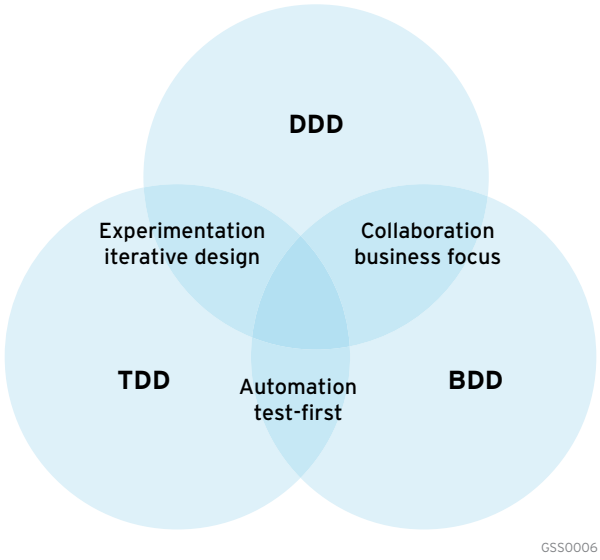

Figure 1: The intersection of test-driven development, domain-driven design and behavior-driven development.

TEST-DRIVEN DEVELOPMENT (TDD)

TDD is a software development process that simplifies codebases, ensures quality and inspires developer confidence. Known to some as "test-first" programming, TDD originated as an aspect of extreme programming (XP), an agile discipline for speeding software development. At its heart, TDD is a process that requires developers to write a test case which declares their program's objective before writing the production code that satisfies that objective. This forces developers to think about how clients will work with their interfaces, leading to cleaner and easier-to-use code. Additionally, the test bed for the project will grow at the same rate as the production code, providing developers with a powerful automated regression suite which catches defects and breaks long before code is shipped. This inspires developers to make more ambitious changes to the codebase as it evolves over the lifetime of a project.

1. *Adzik, Godjco, DDD eXchange 2010, The Three Amigos, http://www.slideshare.net/skillsmatter/ddd-exchange-2010-gojko-adzic-on-ddd-tdd-bdd*

DOMAIN-DRIVEN DESIGN (DDD)

Domain-driven design (DDD) is an approach to software development that acknowledges the complexity of software development. In order to control and manage the impact of this complexity, Eric Evans, author of "Domain-Driven Design", advocates the development of a domain model, which (in his words) is a model that:

"...discards the dichotomy of an analysis model and design to search out a single model that serves both purposes. Setting aside purely technical issues, each object in the design plays a conceptual role described in the model. This requires development teams to be more demanding of the chosen model since it must fulfill two quite different objectives." [2]

To this end, Evans encourages the use of a "ubiquitous language"—common vocabulary that conveys a shared understanding between subject matter experts and software developers. This language gives life to a domain model by connecting the designs of the whiteboard with the objects in an application. Domain-driven design uses a layered architecture and a set of object design patterns discussed later in this paper. Red Hat consultants apply these techniques in a variety of enterprise development contexts to create effective business rules management system (or BRMS)-centric domain models. The results have been promising and provide empirical evidence of improved development and deployment of applications in complex domain contexts.

BEHAVIOR-DRIVEN DEVELOPMENT (BDD)

BDD is a software development process that combines the principles of TDD with the object-oriented analysis and design of DDD. By focusing on the creation of specific examples of domain model behavior, BDD provides developers and business analysts with a common forum to collaboratively develop "ubiquitous language." Using a variety of open source tools, Red Hat Consulting implements BDD solutions that provide automated testing and reporting using the language of the business. This underpinning of customer collaboration practices is central to agile frameworks and helps align delivered software with the emerging requirements of the customer.

The intersection of these three processes creates an approach that facilitates empirical process control, including inspection, adaption, and transparency. By using tests to illustrate and demonstrate the empirical features of working software, stakeholders can review features with developers after each iteration. Establishing a domain vernacular that allows stakeholders and developers to innovate through business-oriented conversations improves knowledge transfer. Living documentation remains with the code to codify requirements, evolving alongside applications to ensure change is embraced rather than resisted. TDD, DDD, and BDD facilitate the inspection, adaptation, and transparency required by agile approaches. (See Figure 1.)

2. Evans, Eric, Domain Driven Design p. 49

ITERATIVE SOFTWARE DEVELOPMENT AND TESTING USING BDD

Clients striving to keep pace with the pace of change are turning to agile frameworks to organize and sustain ongoing 'design-build-test' teams. These teams use iterative development practices to avoid the delays associated with waterfall and big-bang development approaches. Through TDD, agile teams release high quality software achieved through rigorous unit testing and increased code coverage. These approaches are integrated into an automated build in order to practice continuous integration. BDD takes this a step further by establishing a ubiquitous domain language in the vernacular of the customer.

Through specification by example, requirements are captured in scenarios that document the success criteria found in user stories and other artifacts. These scenarios, in a 'given-when-then' format, may be automated through software frameworks like Cucumber JVM, JBehave, and xUnit. Tests are then stored and managed within source code control environments, such as Git and SVN, and provide living documentation that evolves with the accompanying applications' source code. Tests are executed using open source continuous integration tools like Jenkins. Test results may be reported and distributed to help identify and remedy defects throughout the development life cycle. This is unlike serial testing, which often doesn't occur until late in a waterfall development process.

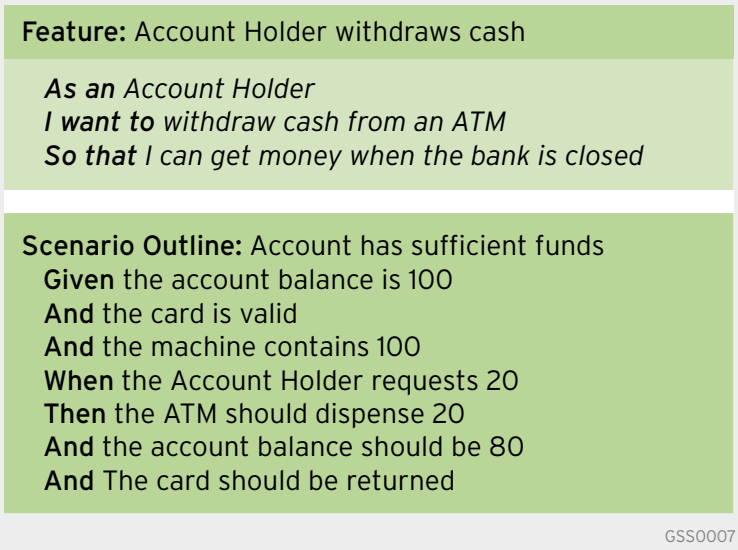

***Figure** 2: An example scenario that may be automated with Cucumber using BDD[3].*

3. http://www.masterthought.net/section/cucumber-reporting

By capturing requirements in a domain language familiar to business stakeholders, working software is demonstrated and reviewed in an iterative delivery process. Revisions to requirements can be captured and maintained with the application source code. This customer-driven technique yields a level of knowledge transfer and customer collaboration that allows for ongoing innovation. Work may be prioritized by business value, risk, complexity, and organizational enablement due to the close contact and ongoing interaction with subject matter experts and stakeholders.

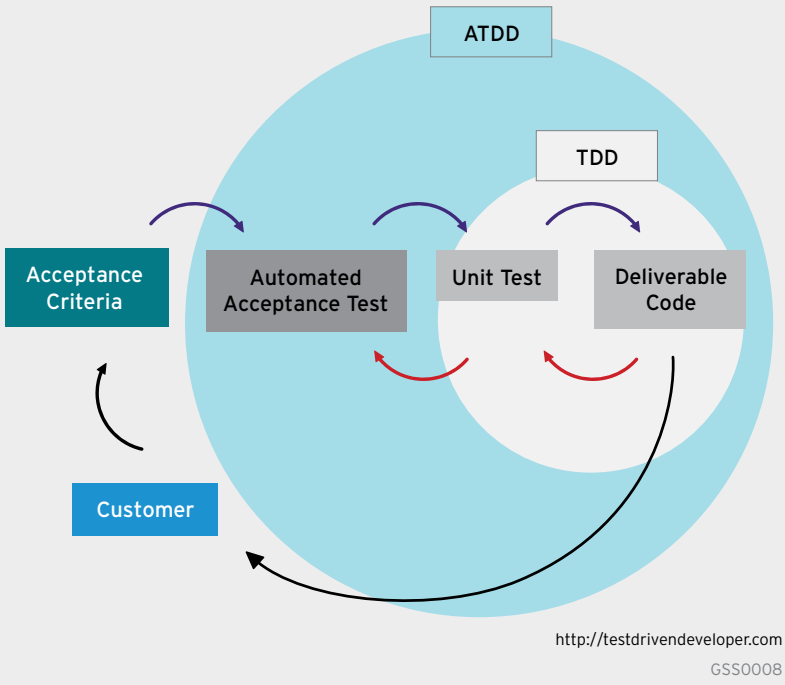

Figure 3: An illustration of an iterative approach with integrated testing

DOMAIN-DRIVEN DESIGN AND THE BRMS KNOWLEDGEBASE

The domain model developed via a DDD process represents a rigorously structured executable model of the business architecture, in which "...each object in the design plays a conceptual role described in the model." [4] In other words, a proper domain model should exhibit the following two traits:

1. Subject matter experts agree that the domain model represents the knowledge needed to automate their business architecture, and
2. Software engineers agree that the domain model alone is responsible for the business behavior exhibited by the overall system.

When viewed through this lens, a proper domain model should be considered a formal knowledge base as described by Ronald Brachman and Hector Levesque, the coauthors of *Knowledge Representation and Reasoning*. The domain model can be said to adhere to what is called the knowledge representation hypothesis by the philosopher Brian Smith.

"Any mechanically embodied intelligent process will be comprised of structural ingredients that a) we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and b) independent of such external semantic attribution, play a formal but causal and essential role in engendering the behaviour that manifests that knowledge." [5]

Knowledge representation and reasoning is considered by many the core subject of artificial intelligence, and it is in this rich field of study that hybrid reasoning systems such as Red Hat JBoss BRMS take root. Connecting the theoretical underpinnings of both the technologies and methodologies used to deliver solutions is a fundamental component of the Red Hat Consulting approach.

In order to provide software developers with building blocks to create domain models, Eric Evans offers a handful of immensely powerful object-oriented design patterns. Developers familiar with open source frameworks such as Hibernate and Spring Data will recognize many of these terms, as they have become common conceptual components of modern software development.

Entities: Objects defined by their continuity and identity, not by the value of their attributes

Value objects: Immutable objects used to describe the attributes of other objects

Aggregates: Entities that manage the life cycle of the objects they own

Factories: Objects that handle the creation of complex entities and aggregates

Repositories: Objects that abstract the access of pre-existing aggregates and are interfaced with the application

Services: Objects that encapsulate complex logic that doesn't naturally belong in a single object

4. *Tirelli, Edson, BRMS Best and Worst Practices And Real World Examples,* ***http://intellifest.org/html/archive_2011/presentations/101_EdsonTirelli__BRMSBestAndWorstPracticesAndRealWorldExamples.pdf***
5. *Brachman, Ronald and Levesque, Hector, Knowledge Representation and Reasoning, pp. 5-6*

Of particular interest to BRMS applications is the concept of the domain service. At the core of BRMS is an implementation of sophisticated pattern-matching algorithms which support complex behavior expressed by the confluence of several objects. Take for example the rule found in Figure 4 below. In order to notify a customer that their delivery is 30 minutes away, several objects in the model must be inspected and each one must conform to a certain pattern. However, implementing this behavior in different objects would unnaturally fracture the single concept of alerting a customer of their impending delivery. By moving this concept to a domain service implemented with BRMS, complex relationships between numerous objects can be captured in a single location within the model using a technology optimized for the behavior.

```
rule "Send shipment pick-up alert"
when
There is a shipment order
There is a route assigned to the order
There is a truck GPS reading and the truck is 30 minutes from the pick-up location
then
Send an alert to the customer: Shipment arrival is expected within 30 minutes"
end
```

Figure 4: A sample of rules used within a knowledge engine

INCREASING SPEED-TO-MARKET THROUGH SIMPLIFIED ARCHITECTURE DESIGN AND RULES ENGINES

Simplifying architecture is a way to decrease the time it takes to initially develop an application as well as reduce the time and cost to maintain it. Conventional model view controller (MVC) application architectures have traditionally been employed for online applications that do not require enterprise scalability. The services associated with data access, RESTful interfaces, and multi-tenant environments are critical to larger enterprise applications. This hybrid architecture incorporates service-oriented-architecture (SOA) with MVC architecture. DDD presents an architecture that puts the domain at the center of these common architectures. It takes advantage of BRMS's unique ability to be embedded as Java™ archive (JAR) files in a traditional Java application deployed as web application archives (WAR) or enterprise archives (EAR).

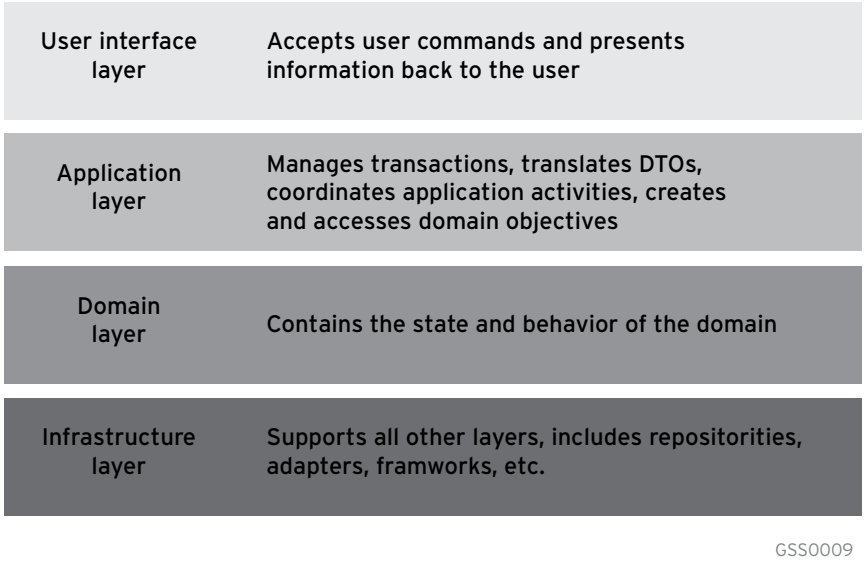

***Figure** 5: A layered architecture based on a domain model approach*

With Red Hat JBoss Application Platform, developers may use multiple software libraries to facilitate this type of development.

USER INTERFACE LAYER

The user interface layer makes up the view component of an MVC solution. By using Java Server Pages (JSPs), an HTML5 front-end with Java Script, and innovative resource libraries such as JQuery, rich Internet applications may be developed quickly.

APPLICATION LAYER

The application layer makes up the controller components of an MVC architecture. The web service API, web application controllers, integration, and batch frameworks might be included at this level. This layer defines the purpose of the application and would deploy in a Red Hat JBoss Application Platform environment as a WAR file.

DOMAIN LAYER

The domain layer is the model layer within a model video controller (MVC) architecture and is the layer where business rules expressing business processes are found. Within the Red Hat JBoss BRMS reference architecture, the components of an encapsulated and loosely coupled knowledge session can be simplified to enable executable logic required by the application design.

INFRASTRUCTURE LAYER

The infrastructure layer provides the technical details of connections to external data sources. Most enterprise-class systems require connections to multiple data sources, so it can be helpful to provide repository interfaces in the domain layer and repository implementations in the infrastructure layer. Object relational model (ORM) technologies such as Hibernate, a community project from Red Hat, are seamlessly integrated with Red Hat JBoss systems.

This layered approach simplifies the architecture of the applications and builds on the simplicity and elegance of the domain model, facilitated through the DDD mapping mentioned previously. Instead of complex systems with too many moving parts, BRMS and Drools engines may be deployed with only Plain Old Java Objects (POJOs) and the common elements of a J2EE application.

The hybrid architecture of MVS (which is common to the spring framework) and SOA (which is a key JEE architecture) ensures scalability and allows enterprise-level performance using supported open source software from Red Hat. When messaging or database access is needed, JBoss environments and the mature open source libraries available within them provide clients diverse alternatives.

BEHAVIOR-DRIVEN DEVELOPMENT IN AN ENTERPRISE AGILE ENVIRONMENT

The simplified architectures in this paper allow the crafting of a knowledge engine within the domain layer to speed development and implement object-oriented approaches. Since agile development teams typically have only 5 to 9 members, there are usually multiple teams operating in parallel. The delivery of each team's component at the end of a potentially shippable increment (PSI) may by synchronized through lightweight agile frameworks such as the scaled agile framework (SAFe).

If rules engine development is used as a Kan Ban swim lane, a team devoted to its design, development, and testing might be organized within the overall program. This team may operate autonomously and independently through the use of specification by example and BDD tools, such as Cucumber-JVM or JBehave.

Instead of having to wait for components from other teams to exercise the knowledge sessions, dependencies may be reduced and blockers or impediments may be avoided through the BDD approach. Through the repository interface, test implementations of a repository can be written to provide data to a knowledge session using the 'given' statements that set up the context of a scenario. 'When' statements can be used to specify a rule flow group or agenda group to exercise based on a certain event. This connects the BDD scenarios to the feature BRMS implements and helps manage large knowledge bases. 'Then' statements can be implemented using Drools queries to request information from the knowledge session and conventional xUnit Assert calls to verify the state of said information. An entire rules application can use the automated scenario to function without the user interface or access to external data sources.

When this approach is employed, productivity increases through the elimination of dependencies. When the rules engines are integrated with modules and components produced by other teams and, after integration, the anticipated results are not achieved, then the BDD tests may be used to isolate integration errors and quickly remedy defects at the time of integration.

Additionally, in agile environments, the BDD approach aligns stakeholders with the rules engine far in advance of integration with other components. Errors in specification, improper representation of requirements, elaboration of outcomes, and comprehensive review of use cases may all occur early in the application life cycle. Innovation often occurs through these review processes because as initial functionality and use cases are demonstrated, customer requirements may be further refined. Since the domain model is mapped through DDD, changes in the rules engine are easily incorporated into the model so that affected components are also updated.

SUMMATION

Agile organizations must develop applications that yield high returns at a reduced cost. To accomplish this goal, these organizations must find ways to support more productive teams. Organizations recognize that quality software has to be testable and supportable in enterprise-level production environments. Success requires accurate, understandable requirements and an integrated approach to quality assurance (QA).

BDD is not just a way to develop software correctly but a way to develop the correct software. Red Hat Consulting uses BDD and Red Hat JBoss BRMS to improve our clients' success rates. Our technologies and methodologies provide a means of communication and collaboration to help our clients take advantage of the complementary skills found within cross-functional design-build-test teams. The result is high-quality applications that quickly deliver business value at reduced cost.

ABOUT THE AUTHOR

Justin Holmes is a senior consultant and services delivery manager with Red Hat Consulting. He was a Jefferson Scholar at the University of Virginia where he studied the connections between computing and the humanities, earning a Bachelors Degree in Computer Science. Mr. Holmes specializes in developing BRMS solutions and has delivered projects for the health insurance, financial services, and retail industries. His passion for connecting subject matter experts to software developers with business rules and automated testing practices has been instrumental in developing Red Hat's agile practice.

ABOUT RED HAT

Red Hat is the world's leading provider of open source solutions, using a community-powered approach to provide reliable and high-performing cloud, virtualization, storage, Linux, and middleware technologies. Red Hat also offers award-winning support, training, and consulting services. Red Hat is an S&P company with more than 70 offices spanning the globe, empowering its customers' businesses.

NORTH AMERICA
1 888 REDHAT1

EUROPE, MIDDLE EAST AND AFRICA
00800 7334 2835
europe@redhat.com

ASIA PACIFIC
+65 6490 4200
apac@redhat.com

LATIN AMERICA
+54 11 4329 7300
info-latam@redhat.com

facebook.com/redhatinc
@redhatnews
linkedin.com/company/red-hat

redhat.com
#11153337_v3_0713