Contents

[1. Description of the project 3](#_Toc436915179)

[1.1 Goal 3](#_Toc436915180)

[1.2 Product 3](#_Toc436915181)

[1.3 Process 3](#_Toc436915182)

[1.3.1 Development / Maintenance 3](#_Toc436915183)

[1.3.2 Defect management 4](#_Toc436915184)

[2. Product History 5](#_Toc436915185)

[2.1 Release 5](#_Toc436915186)

[2.1.1 Release note 5](#_Toc436915187)

[2.1.2 Migration note 5](#_Toc436915188)

[2.1.3 Motivation of changes 5](#_Toc436915189)

[2.2 Adoption 5](#_Toc436915190)

[2.3 Feature Tracking 6](#_Toc436915191)

[2.4 Critical Reviews 6](#_Toc436915192)

[3. Product Structure 7](#_Toc436915193)

[3.1 Architecture 7](#_Toc436915194)

[3.1.1 Layered Architecture 7](#_Toc436915195)

[3.1.2 Key Modules of core framework 8](#_Toc436915196)

[3.2 Design 9](#_Toc436915197)

[3.2.1 High Level Design 9](#_Toc436915198)

[3.2.2 Detailed Design 9](#_Toc436915199)

[3.2.3 Domain Model Design 9](#_Toc436915200)

[3.2.4 DB Design 9](#_Toc436915201)

[3.3 Code-base 10](#_Toc436915202)

[3.4 Documentation 11](#_Toc436915203)

[3.5 Test 12](#_Toc436915204)

[3.5.1 Unit Testing 12](#_Toc436915205)

[3.5.2 Integration Testing 14](#_Toc436915206)

[4 Defect Management 15](#_Toc436915207)

[4.1 Report 15](#_Toc436915208)

[4.1.1 Resources 15](#_Toc436915209)

[4.1.2 Required information to report an issue 15](#_Toc436915210)

[4.2 Repair and Release 15](#_Toc436915211)

[5 Appraisal of the Product and Management 17](#_Toc436915212)

[5.1 Standards Compliance 17](#_Toc436915213)

[5.2 Maintainability 17](#_Toc436915214)

[5.2.1 OO Metrics 17](#_Toc436915215)

[5.2.2 Readability 19](#_Toc436915216)

[5.2.3 Analyzability 20](#_Toc436915217)

[5.3 Reliability 20](#_Toc436915218)

[5.4 Usability 20](#_Toc436915219)

[5.5 Extensibility 21](#_Toc436915220)

[5.5.1 Entity & Service 21](#_Toc436915221)

[5.5.2 Add-on Modules 22](#_Toc436915222)

[5.5.3 Presentation Layer 22](#_Toc436915223)

[5.6 Portability 22](#_Toc436915224)

[5.6.1 Platform 22](#_Toc436915225)

[5.6.2 Web/App container 22](#_Toc436915226)

[5.6.3 Database 23](#_Toc436915227)

[5.7 Performance 23](#_Toc436915228)

[5.8 Scalability 23](#_Toc436915229)

[5.8.1 High transaction volume 23](#_Toc436915230)

[5.8.2 Concurrent users 23](#_Toc436915231)

[5.8.3 Large catalogs 24](#_Toc436915232)

[5.8.4 High conversion rates 24](#_Toc436915233)

[5.9 Conclusion 25](#_Toc436915234)

# Description of the project

## Goal

BroadleafCommerce is an open-source, e-commerce framework written entirely in Java on top of the Spring framework. It is targeted at facilitating the development of enterprise-class, commerce-driven sites by providing a robust data model, services and specialized tooling that take care of most of the "heavy lifting" work. To accomplish this goal, they have developed their platform based on the key feature sets required by world-class online retailers - and are committed to continually expanding their feature offering. They have also taken extra steps to guarantee interoperability with today's enterprise by utilizing standards wherever possible and incorporating best-of-breed, open-source software libraries from the community.

## Product

An enterprise solution feature set ideal for Fortune 500 needs, Broadleaf provides the most sought after functionality for supporting B2C, B2B, and B2B2C eCommerce at the best value in the market. Every solution can be customized to ensure your eCommerce site is tailored to your specific requirements. Robust functionality within a lightweight framework lend to some of the characteristics that cause Broadleaf to stand out from the rest.

Following are the feature list

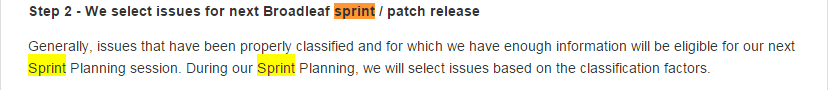
* Shopping Cart
* Offers & Promotions
* Content Management System (CMS)
* Product Management
* Multi-Everything
* Search & Browse
* Order Management
* Customer Management
* Framework Features

Besides community version, Broadleaf has a commercial license which providing additional and professional services.

## Process

### Development / Maintenance

From all the resources information, we are not able to find direct evidence of their SDLC methodology but only a few words mentioned the Sprint.



Hence our assumption is they develop and maintain the software following Agile Scrum.

1. Planning

Each sprint starts with a sprint planning event that aims to define a sprint backlog, identify the work for the sprint, and make an estimated commitment for the sprint goal.

1. Daily scrum

Starts with a Daily Scrum Meeting (aka 15 minute stand-up meeting). Each team-member answers three questions:

* What did I do yesterday that helped the development team meet the sprint goal?
* What will I do today to help the development team meet the sprint goal?
* Do I see any impediment that prevents me or the development team from meeting the sprint goal?

1. Review and retrospective

Each sprint ends with a sprint review and sprint retrospective, that reviews progress to show to stakeholders and identify lessons and improvements for the next sprints.



### Defect management

Broadleaf Defect Management Process is described in section 4.

# Product History

## Release

The first release on BroadleafCommerce GitHub repository is version 0.9.1 on Jul 31, 2010. Till now there are 163 releases including major, minor feature changes and patches.

Currently they still provide support for 7 versions,

* 4.0 (current)
* 3.1
* 3.0
* 2.3
* 2.2
* 2.1
* 2.0

### Release note

However, only releases after version 3.0 come with release note, users still using older version might find it difficult to understand the differences between releases.

### Migration note

Meanwhile, a bright side is Migration Notes are thoroughly provided. In these documentation, the instructions are clearly stated, although there is always risk for attempt of migration, it is helpful to migrate the application based on older version even v1.5 to current v4.0 step by step.

### Motivation of changes

After inspecting few release notes, we are able to identify all four types of maintenance. Corrective changes usually are a part of minor release or patch, on the contrary, Adaptive, Perfective and Preventive are mainly appeared in major release. Possible explanation we can give is that those type of changes require more time and effort to plan and analyze the impact carefully.

* **Corrective**

ThreadLocal changes to SystemTime are not always being cleaned up, which can result in unexpected behavior on a site #1490 (From release note of 4.0.1)

* **Adaptive**

Update to the latest Spring 4.1 version #1116, as Spring 4.1 was released September 4th 2014, and is designed to immediately supersede the 4.0 line, the team decided to migrate Broadleaf from Spring 4.0 to 4.1 (From release note of 4.0)

* **Perfective**

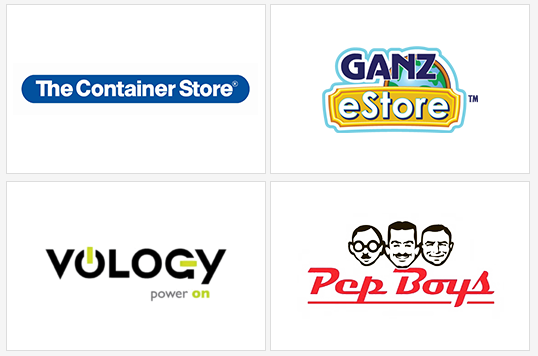
Add save method to Store service/dao #770 (From release note of 4.0)

* **Preventive**

Clean up and refactor the extension managers in AdminAbstractController #1003 (From release note of 4.0)

## Adoption

BroadleafCommerce is proud to power the eCommerce capabilities for some of the most recognized brands around the globe. One thing all of their clients share, is the need to have a commerce that works for their unique commerce needs. No two businesses are exactly alike, so this system is provided to celebrate that fact. Following are some clients running their business with system built on BroadleafCommerce framework.



* The Container Store

Home/Office supplies, Top 500 Internet Retailer Site

* Ganz eStore

Broadleaf Commerce allowed Ganz to launch a re-vamped, elegant, effective eCommerce solution that meets all of their complex business and technical requirements.

* Vology

Network Hardware store, B2B provider selling approximately $140 million of technology solutions each year to small and mid-sized companies, Fortune 2000 businesses, government agencies, as well as educational institutions and districts nationwide.

* Pep Boys

Auto parts store with 99,417 Unique SKUs, 723 Stores and 30+ Back-end Integrations

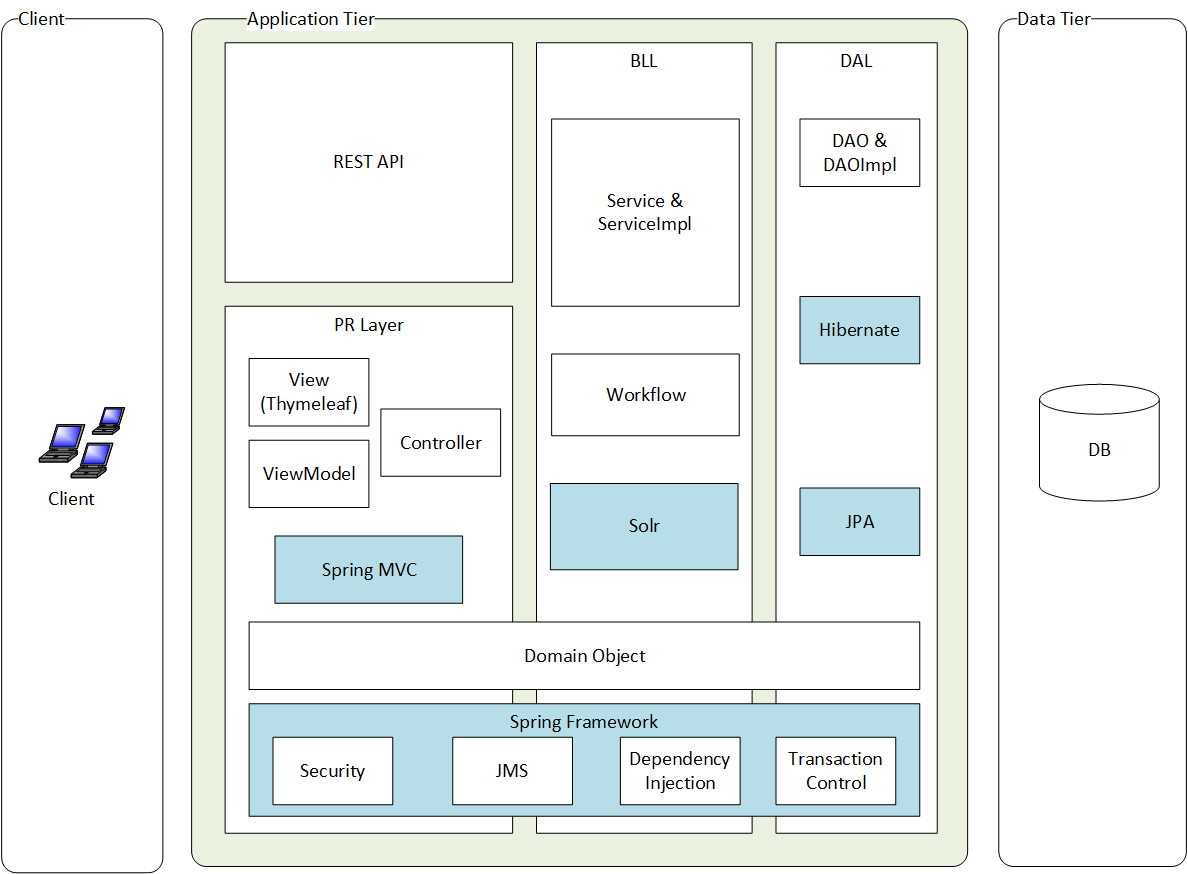
## Feature Tracking

## Critical Reviews

# Product Structure

## Architecture

### Layered Architecture



Broadleaf follows Sun J2EE guideline, its application comprises Presentation Layer, Business Logic Layer and Data Access Layer. Spring Framework provides the component architecture used by Broadleaf Commerce. All three layers has dependency on Broadleaf Domain Models.

#### Presentation Layer

The web view layer of Broadleaf Commerce application is built on Spring MVC framework. The Thymeleaf template engine which is well-integrated with Spring MVC is recommended for processing and generating HTML, XML, JavaScript, CSS and text instead of JSP.

Besides web view, Broadleaf Commerce exposes a set of functional APIs as RESTful services that allow integration with other applications, including mobile applications. These APIs are easy to expose, easy to extend, and provide a simple pattern for adding net new functionality. XML and JSON formats for messages are supported.

#### Business Logic Layer

Besides regular Online Commerce Domain Business Logic implemented as Service and ServiceImpl, there are some other features provided in this layer

* **Configurable Workflow**, as a Key areas in the eCommerce lifecycle. Implementers have full control over the keys steps in pricing and checkout, allowing manipulation of module order, behavior, and custom execution. Composite workflows are also supported to achieve sophisticated, nested behavior.
* **Task Scheduling**, Repetitive tasks can be scheduled through the Quartz job scheduling system.
* **Search**, Flexible domain search capabilities are provided through integration with Solr, a popular Compass and Lucene projects.

#### Data Access Layer

Broadleaf Commerce implements Java Persistence API using Hibernate Object-relational mapping Framework.

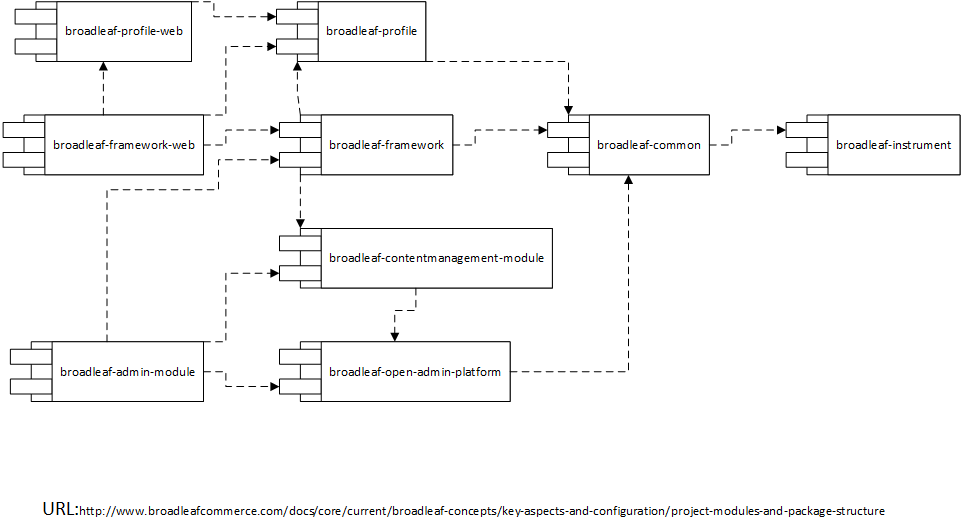
#### Crosscuttings

All above layers have dependency on Broadleaf Commerce Domain Model to transfer data or perform operations.

Broadleaf Commerce takes advantages of numerous features Spring Framework provided, such as

* Spring Security for controlling authentication and authorization at both the code and page level
* Spring JMS for asynchronous processing of application messages
* Dependency Injection and Inversion of Control
* AOP and transaction control

### Key Modules of core framework



There are currently 9 key modules included as part of the Broadleaf Commerce framework. It would be exceptionally rare for a Broadleaf Commerce implementation NOT to use all of these core modules. The above diagram shows their dependency relationship.

#### Framework

This generically named module represents the commerce functionality of Broadleaf Commerce (e.g. Orders, Products, Offers, etc.

* broadleaf-framework, Core Broadleaf framework classes
* broadleaf-framework-web, Spring MVC controllers and related items

#### Profile

This module provides the concept of Customer. It is provided separately from framework in anticipation that some may want to utilize these features without using the Commerce features.

* broadleaf-profile, Customer profile related classes, utility classes, email, configuration merge
* broadleaf-profile-web, Spring MVC controllers and related items supporting the profile module

#### CMS

This module provides content management functionality that supports targeting ad based content to users based on their profile as well as static page management.

* broadleaf-contentmanagement-module, A full-featured content management system that is managed via the administration tool

#### Open Admin

The Broadleaf Commerce admin architecture which allows annotated JPA entities to be administered via a rich UI. The intent of separating this module is to provide some architecture purity while leaving open the possibility that the admin techniques used by Broadleaf Commerce may be provided outside of the Commerce application in the future.

* broadleaf-admin-module, Contents: Broadleaf Commerce specific administration module that plugs into the open admin platform
* broadleaf-open-admin-platform, Framework for creating extensible administration GUIs for Hibernate managed domains

#### Common

* broadleaf-common, A collection of classes shared by various modules.
* broadleaf-instrument, Allows for runtime instrumentation to override certain Broadleaf annotations

## Design

### High Level Design

As an out of the box product as well as a framework for enterprise eCommerce, high level design document is provided with

* Logical view, help user to identify key components and their responsibilities
* Physical view, help user to understand the possible approach to deploy the software on physical devices and network environment.
* Key architectural decisions such as
  + Choosing the Spring Framework over EJB3
  + Using Solr to achieve catalog browsing and searching

### Detailed Design

From the documentation provided, there is no evidence that they followed a RUP Object-Oriented Analysis and Design Approach and did a detailed level design before writing the code. Our assumption is that they follow the Agile manifesto: **Working software** over comprehensive **documentation.**

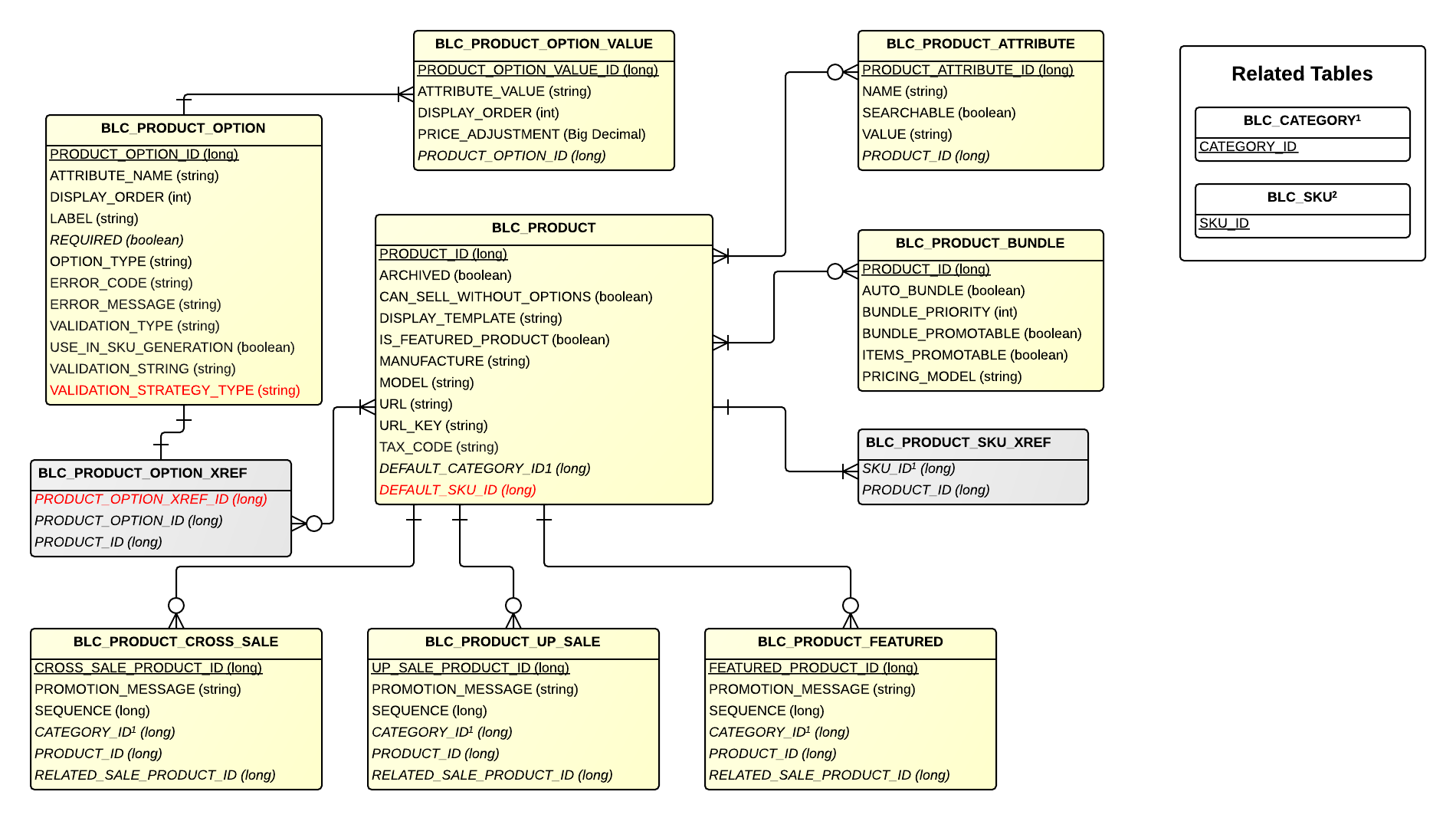
### Domain Model Design

In the source code we discover quite a lot of domain classes found for each feature such as catalog, checkout, inventory and offer etc. Under org.broadleafcommerce.core.catalog.domain package, there are 48 interfaces and classes, including Category, CategoryAttribute, Product, ProductAttribute etc.

However, there is no Domain Model Diagram provided which means that to fully understand it, we have to do reverse engineering to get the class diagram from codes. And with that large amount, it is quite troublesome.

### DB Design

In terms of DB Design, perhaps in order to compensate the lack of Domain Class Design, ER diagrams are satisfactorily comprehensive and in detail. As an example, ER diagram of Catalog Product.



## Code-base

Although we are not able to find the design guideline, detailed design or class diagram, our observation is

* Packages and classes are Well-organized.

For instance, in BroadleafCommerce Framework modules, basically for each feature like catalog, there are dao, domain, service, service.exception etc. which is very helpful for user to expect what class can be found in that package.

* SOLID principles are followed

In the Product entity class, it has an aggregation of objects of ProductAttribute as attribute. Hence every attribute of a product is defined outside the Product class which makes it very flexible to change, add or remove from a product. It illustrate the Single Responsibility Principle and Open-Close Principle.

Furthermore, the ProductAttribute is an interface, its implementation class ProductAttributeImpl has been hiding from Product. It shows the Dependency Inversion Principle.



## Documentation

Following documentations are provided:

|  |  |
| --- | --- |
| **Item** | **Description** |
| Getting Started | Getting environment up and running, show where things live, and walk through a few examples. |
| Broadleaf Concepts | This section is where the majority of information on Broadleaf functionality and concepts can be found. We describe how important operations such as cart modification, pricing, and payment occur as well as additional configuration that Broadleaf supports. |
| Modules | Broadleaf Commerce can be enhanced with add-on modules. |
| Tutorials | This tutorial section will walk you through examples start to finish that demonstrate how to work with various different parts of Broadleaf Commerce. |
| Database Model | Model Changes and current ER Diagram |
| Release Notes | The note of release, including description, bugs, and enhancements. |
| Migration Notes | This section contains migration documents that will help developer transition his current Broadleaf installation to a more recent version. |
| Appendix | Other documents which are associated with the project. |
| Javadocs | Javadocs including Package, Class, Use, Tree, Deprecated, Index, and Help. |

BroadleafCommerce comes with quite a lot of documentations to support users, although it is not so clearly structured.

For example, “Getting Started” and “Tutorials” are similar from the name.

Moreover, same type of information located here and there creates hinder for a new comer to find out what he wants. Under “Broadleaf Concepts” menu item, there are many items like “Merge Configuration”, “Persistence Configuration” and “Database Configuration” within “Key Aspects & Configuration” catalog. Besides that there is an “Additional Configuration” under another menu item.

## Test

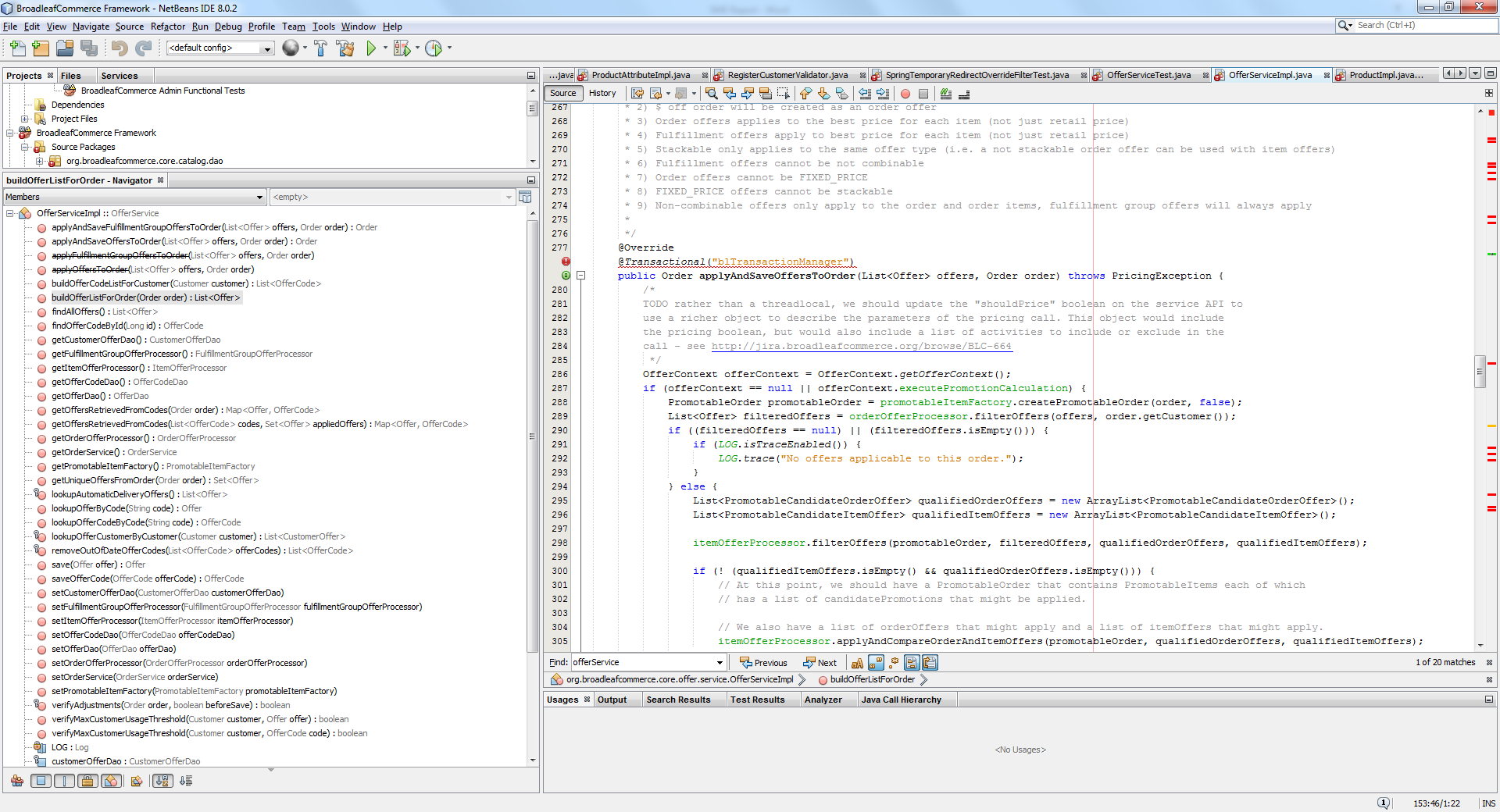
On the wiki of BroadleafCommerce repository, the team express the concerns that test coverage is currently pretty lacking and could definitely be improved. They therefore expect developers who are interested in this framework to help them to achieve the goal

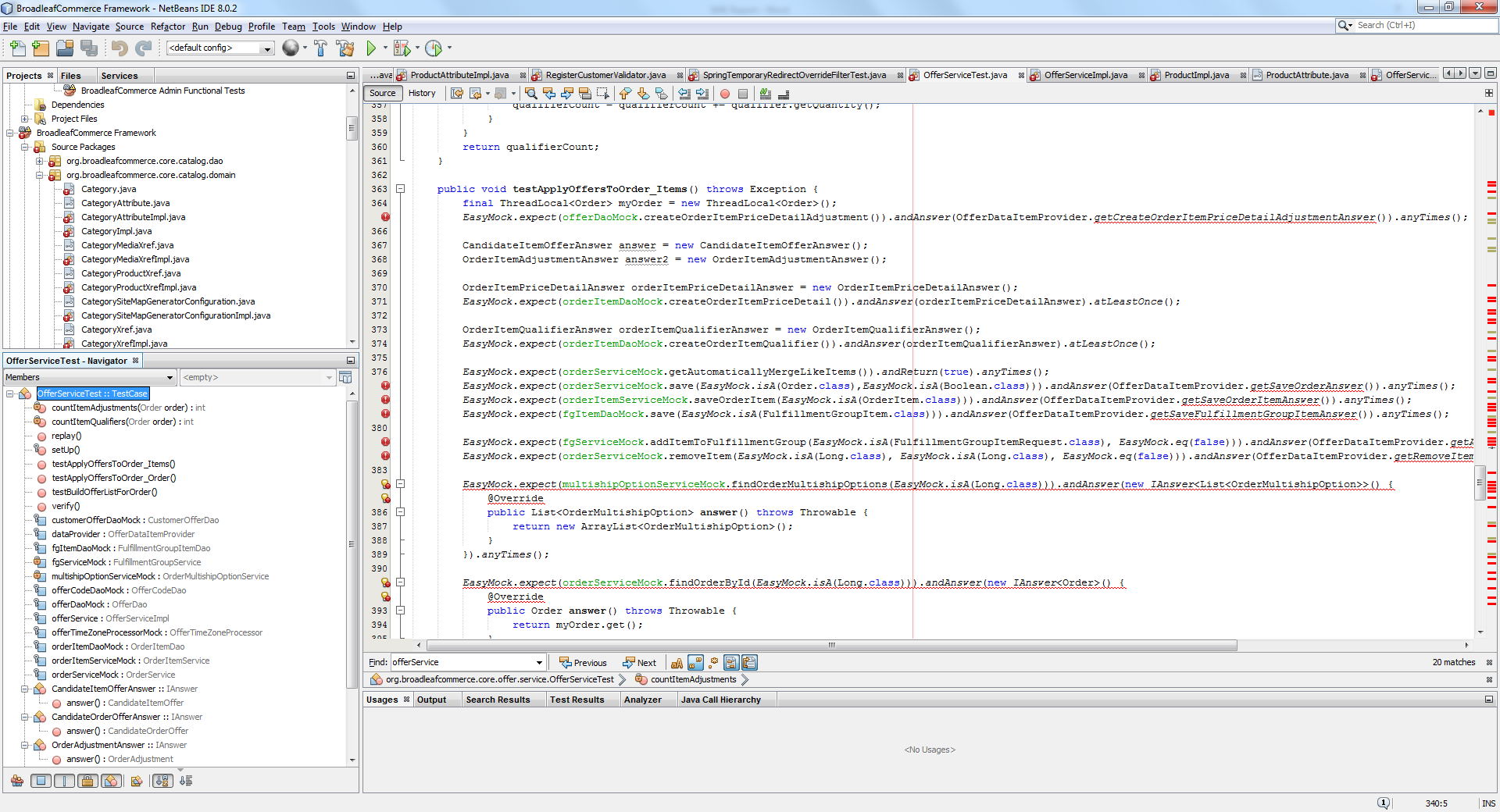
* Get the test coverage to above 70%
* Ensure that there are at least some tests for each of the core modules (for instance, our admin code currently has 0% test coverage)
* Write better tests for the Spring applicationContext merge process
* Hook up Broadleaf to something like Sonar or Cobertura (which has a maven plugin we are using in some of our integration modules). This will help them publicize their test coverage and identify exactly what areas of Broadleaf are missing tests

### Unit Testing

As a J2EE application, BroadleafCommerce uses JUnit and a mocking framework called EasyMock to perform unit test.

According to published source code, the Unit Test strategy is only testing significant method of business logic classes. For instance, the diagram below shows the methods of OfferServiceImpl class. Only two of all methods have been tested.





Due to the widely applied Dependency Injection, some dependencies created, managed and injected into the object by context. To make the class testable, setter method of those dependencies are exposed as Object Seams to enable manipulation of their behaviors by setting mockups into the object.

@Service("blOfferService")

public class OfferServiceImpl implements OfferService {

@Resource(name="blCustomerOfferDao")

protected CustomerOfferDao customerOfferDao;

@Resource(name = "blOrderService")

protected OrderService orderService;

…

@Override

public void setCustomerOfferDao(CustomerOfferDao customerOfferDao) {

this.customerOfferDao = customerOfferDao;

}

@Override

public void setOrderService(OrderService orderService) {

this.orderService = orderService;

}

…

}

### Integration Testing

Spock/Groovy is introduced to perform Integration Tests. And with the 4.1 release of Spring, developers now have the ability to test RESTful services, workflows and other integration test abilities in a much simpler format.

To test a RESTful services, first step is to set up the test spec by with POM and plugin. Then it requires Spring-Test's MockMVC api with its annotations as well as Groovy language. But without the expertise of those technologies, we found that is quite difficult to understand.

# Defect Management

## Report

### Resources

There are two approaches provided for BroadleafCommerce community users to report defects.

* Forum (<http://forum.broadleafcommerce.com/>)

Forums are useful for requesting help from other community members.

* GitHub Issue tracker

GitHub issues are used to track bugs.

In other words, when a user is uncertain whether BroadleafCommerce provides a feature he expects or if he encounters an unintended behavior or a bug, he can go to the forum. Otherwise he should raise an issue on GitHub Issue tracker.

### Required information to report an issue

When user wants to raise an issue, following information is required to help the team reproduce and locate the defect

* Broadleaf version you are using
* Steps to reproduce
* Any stack traces that you receive (if applicable)
* Any additional information that allows us to help you faster
* Any potential fixes you might have already tested

## Repair and Release

There is a well-defined process for defect repair and release in CONTRIBUTING.md file.

* Step1 – Analyze the issue

Team member will first analyze the issue and label the raised issue.



Here are the category of labels

* + Severity-(critical/major/minor)
  + Module-(admin/cms/core/rest/tests)
  + Type-(bug/enhancement/feature).
  + Difficulty-(hard/medium/easy)
  + Affects
  + Target to Release
* Step2 - select issues for next Broadleaf sprint / patch release.

During the sprint planning, team member will select the issues to be fixed and release according to their priorities.

* Step3 - assign a milestone indicating the release that targets for the issue



* Step4 - mark the issue as closed generally once the developer has committed the fix
* Step5 - We release the patch (or new major version) of the software

In some cases, due to some resource or schedule constraints, the issues reported cannot be repaired within a certain time period, which may cause inconvenience to users. Hence an alternative way recommended by BroadleafCommerce team is that since it is an open-source product user can fix the defect then send a pull request in GitHub after passing unit test and regression test. The team will review the validity and merge the changes to be released.

That is also the reason why open source software hardly shows declining quality over time as what we have learned from software maintenance and evolution course. With the large amount of users who are keen to contribute to improve the software, defects are identified and repaired effectively.

# Appraisal of the Product and Management

## Standards Compliance

## Maintainability

### OO Metrics

Because Broadleafcommerce is a maven project, so we try to use the IntelliJ to calculate the metrics.

What we can find are:

* Method metrics: Essential cyclomatic complexity, Design complexity, Extended cyclomatic complexity, Cyclomatic complexity.
* Class metrics: Average operation complexity, WMC.

#### Essential cyclomatic complexity

Essential cyclomatric complexity however tells how much complexity is left once we have removed the well-structured complexity.

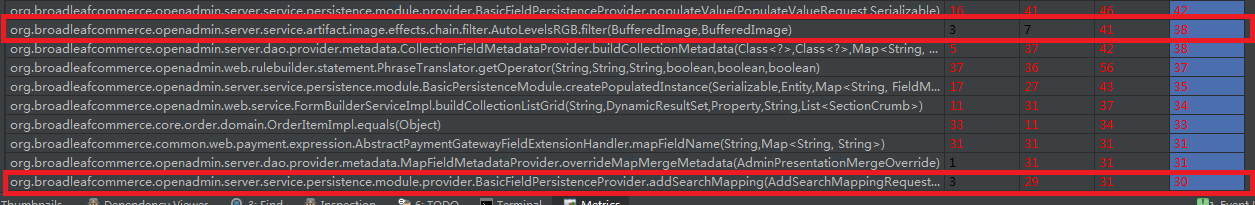
A function with a high Cyclomatic complexity means a function difficult to test. Now if this function has a low essential Cyclomatic complexity it means it's fairly easy to break up this function into other smaller functions which are easier to test individually. When essential complexity is high this refactoring is more difficult as the complexity is more difficult to understand.

So code that has a high essential complexity means that the code is harder to maintain and understand. This code is said to be of lower quality. Code that has a high complexity is harder to test but in general we can do something about this more easily when the essential complexity is low.

In this complexity, the method “org.broadleafcommerce.openadmin.dto.BasicFieldMetadata.equals(Object)” has the high value.

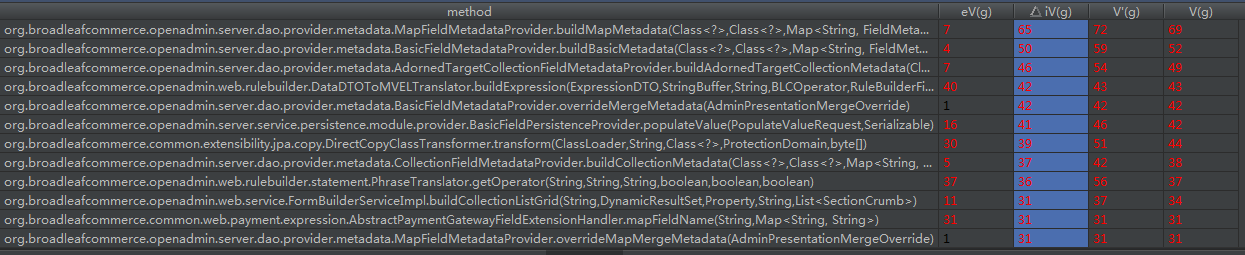
|  |  |
| --- | --- |
|  |  |

Meanwhile there are still some methods that cyclomatic complexity are pretty high, but Essential cyclomatic complexity is low, which means that the code optimization is not finished well:



#### Design complexity

Module Design Complexity (iv(G)) is the complexity of the design-reduced module and reflects the complexity of the module's calling patterns to its immediate subordinate modules. This metric differentiates between modules which will seriously complicate the design of any program they are part of and modules which simply contain complex computational logic. It is the basis upon which program design and integration complexities (S0 and S1) are calculated. The method with high design complexity means the method is hard to maintain because of the complex relationship in the system.



#### Cyclomatic complexity

Cyclomatic Complexity (v(G)) is a measure of the complexity of a module's decision structure. It is the number of linearly independent paths and therefore, the minimum number of paths that should be tested.

According to the calculation of the methods, a table can be concluded:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Cyclomatic complexity | 1-10 | 11-20 | 21-30 | 31-40 | 41-50 | 51-60 | 61 and above |
| Number of Method | 13629 | 180 | 22 | 8 | 6 | 2 | 1 |

According to the calculation result, more than 98.4% methods’ Cyclomatic Complexity is under 10. It means for the module's decision structure, the project team finished it well.

For some particular methods, the Cyclomatic Complexities are pretty big, it is because in these methods, there are too many “IF, ELSE” codes, but for these method, because the Essential cyclomatic complexity is also high, so it’s hard to reduce the complexity.

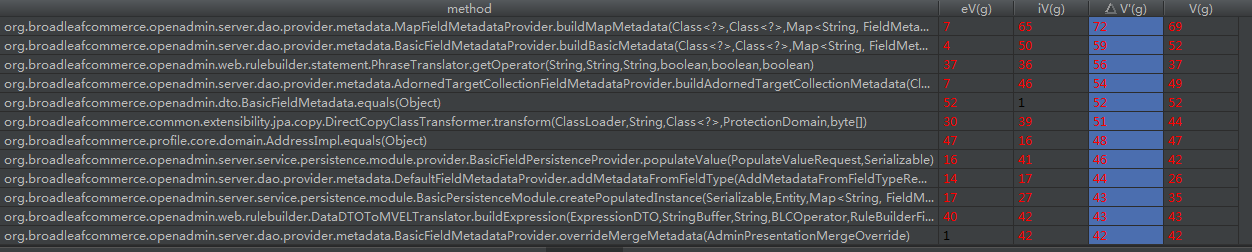
#### Extended cyclomatic complexity

Extended cyclomatic complexity, can also be called CC2, Cyclomatic complexity with Booleans.

CC2 extends cyclomatic complexity by including Boolean operators in the decision count. Whenever a Boolean operator (And, Or, Xor, Eqv, AndAlso, OrElse) is found within a conditional statement, CC2 increases by one. The statements considered are: If, ElseIf, Select, Case, Until, While, When.

The reasoning behind CC2 is that a Boolean operator increases the internal complexity of a decision. CC2 counts the "real" number of decisions, regardless of whether they appear as a single conditional statement or split into several statements. Instead of using Boolean operators to combine decisions into one (x=1 And y=2), you could as well split the decisions into several sub-conditions (If x=1 Then If y=2 Then). CC2 is immune to this kind of restructuring, which might be well justified to make the code more readable. On the other hand, one can decrease CC simply by combining decisions with Boolean operators, which may not make sense.

Including Boolean operators in cyclomatic complexity was originally suggested by Thomas McCabe. In this sense, both CC and CC2 are "original" cyclomatic complexity measures.



#### WMC

The WMC metric is the sum of the complexities of all class methods. It is an indicator of how much effort is required to develop and maintain a particular class. A class with a high WMC indicates that the class is complex (application specific) and therefore harder to reuse and maintain.

According the Statistical Threshold table and the calculation, the result is below (excluding WMC=0):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metric | low | average | High | Very High |
| Number of Class | 659 | 464 | 375 | 197 |

And the average of these data (excluding WMC=0) is 15.54. It means although most of the classes have the low WMC, but the average WMC of the classes is high, which indicates that the project classes are complex and hard to reuse and maintain.

### Readability

Readability is the ease with which a reader can understand a written text. Here, we analysis the readability from four aspects – source code structure, framework, size of project and the complexity of source code.

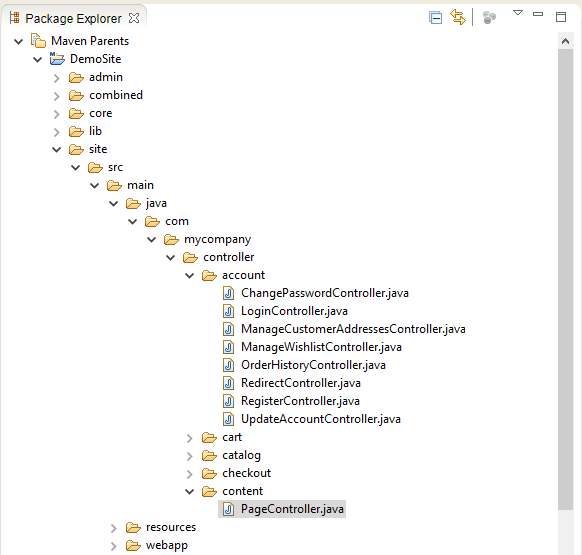


Figure. Part of Source Code Structure Example

The source code structure is very clear and easy to understand. The above figure 1 is the snapshot of source code structure for the site part. The site module implements all the features for public customer including account controller, shopping cart, product browser (catalog) and checkout. For each feature, all necessary actions are designed by different controllers such as the account folder contains login controller, change password controller and so on. Because the project uses Spring framework based on MVC pattern, there is page controller to dispatch pages mapping the url. And the webapp folder contains all the pages will be presented in the front-end. Also, there are some documents that introduce all the modules in the website.

Because of using Spring framework, we have to cost more time to understand all the concept about this framework such as MVC pattern, annotation and so on if we don't know that before. Without that, we would be hard to locate a specific class or understand how it works.

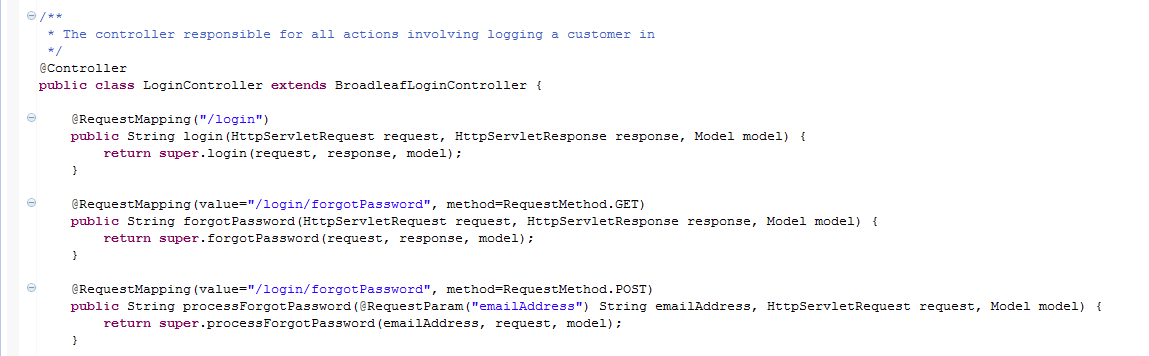


Figure. Annotation Example in login

And for the implementation, the project is big and most controllers are inheritance. Some algorithms are too complicated because these are too many if-else statements stacking in one method, as the above shows in the 5.2.1.1’s example. That's hard to read.

So according to the source code structure we can easy to read in module level, which is the benefit to the readability. But the high coupling with framework may hinder the reading if we don't know that before. And the size of the project and the complexity of source code must be a block to the readability due to the high usage of inheritance and the high level of complexity we calculated.

### Analyzability

The analyzability depends on the ability of tracking error or status. So the logging engineer is important to it. Normally we can get the logs from server log, database log and customized log API like Spring logback. Server log will record the exception java code console when system runs. According to their provided Javadoc which contains all the definition of all classes, we get to know most classes implement exception handler. So the server log will be benefit to analyzability.

Except that, Broadleaf also integrates with Quality of Service (QoS) monitoring for vendor services. QoS does not interfere with a call to a vendor service (such as USPS), but does provide a way for that service to report on its status. Based on how to configure QoS, the maintainer can be notified of vendor service status changes in any number of ways. By default, Broadleaf will merely log vendor service status changes to the console. If this behavior is not sufficient, Broadleaf also provides support for email notifications for QoS status changes. So QoS also improve the analyzability.

## Reliability

## Usability

Usability is the ease of use and learnability. The usability would be discussed in developer perspective in term of using Broadleaf to develop an eCommerce website.

To use Broadleaf, we have to install it first. The installation guide is provided and explains very clearly step by step. But the installation is not smooth. The problem we met is caused by the third party plugin – Maven which helps build the development environment. When we installed the Maven and started to build the environment following the guide, we got nothing error about Maven but only about the environment. After the hard time to search and browse the forum, we got nothing. Finally, we fixed it by installing Maven again following another guideline. It seems the installation guide is far to the last update and some settings of the third party tools and plugins have changed.

Broadleaf use Thymeleaf as a templating engine and default UI rendering engine and use Spring framework following the MVC pattern. And most controller is defined and can be used. So to simply develop an eCommerce website we can only finish the configuration works and choose a UI template to use. It is similar to CMS like wordpress. We also can customize all the things if necessary. Broadleaf has provided with documents like Javadoc to support that. That's also benefit to learnability.

So far as we know the usability is good but these exists some problems in installation and documentation. Some documents are provided and some other important documents is lacking such as design documents that will help us easy to understand and develop the project.

## Extensibility

The extensibility is the ability to extend new functionality in which the system’s internal structure or data flow are minimally or not affected. As the provided documents say, Broadleaf has considered extensible design.

“Extensibility is at the core of our design, and almost every aspect of Broadleaf can be overridden, added to or otherwise modified to enhance or change the default behavior. This includes all of our services, data access objects and entities.” – from Broadleaf website

### Entity & Service

There are two types of extensions – extending entities and extending services by inheritance. The following is an implementation example for a custom entity.

@Entity

@Table(name = "MY\_ORDER")

public class MyOrderImpl extends OrderImpl {

private static final long serialVersionUID = 1L;

@Column(name = "MY\_ORDER\_ID")

private String myOrderId;

... getters / setters / equals / hashcode ...

}

Here, we've created a new custom entity that extends OrderImpl from Broadleaf Commerce. We've added our custom order id field (myOrderId), and using JPA annotation, have tied this entity and it's fields to a table and columns in the database.

Now that we've created our entity extension, we need to notify Broadleaf Commerce of its existence and how to instantiate it. To do this, we need to add a bean configuration to application context:

<bean id="org.broadleafcommerce.core.order.domain.Order" class="com.mycompany.order.domain.MyOrderImpl" scope="prototype"/>

Finally, we need to add the new entity class to the persistence unit configuration in your persistence.xml. This could end up looking similar to the following:

<persistence-unit name="blPU" transaction-type="RESOURCE\_LOCAL">

...

<class>com.mycompany.order.domain.MyOrderImpl</class>

...

</persistence-unit>

This is the least amount of configuration required for Broadleaf Commerce to utilize entity. Once configured, Broadleaf Commerce will create an instance of MyOrderImpl whenever a new Order instance is required. The services also can be extended as well.

### Add-on Modules

Broadleaf can be enhanced with add-on modules. Add on modules represent functionality that can be incrementally added to the Broadleaf Commerce framework. Add-on modules may be free, open source or commercial. Examples of free, open source modules include the Inventory and SEO modules. Some other modules are payment module, tax module, shipping module and so on.

### Presentation Layer

In Broadleaf, the administration application is based on our new Open Admin platform, which provides a clear path for customization using standard object oriented techniques. We also can enjoy the same level of extensibility in the admin platform that we already enjoy in the core framework (extending entities and services).

The presentation layer is enhanced from Broadleaf 2.0. First, Broadleaf provides classes that perform all of the necessary logic by @controller annotation. It is up to the implementor to extend the Broadleaf\*Controller class, annotate their own class with @Controller and provide methods that will match certain URLs. It is benefit to customizing URL, sharing controller logic and adding custom logics. We are really easy to customize logics to controller. The following is the adding custom logics example from Broadleaf.

**@RequestMapping(value = "/some/path")**

**public String doSomething(HttpServletRequest request, HttpServletResponse response, Model model) {**

**... do some cool stuff ...**

***// Call the super controller (or not if you want to completely override the functionality)***

**String returnPath = super.doSomething(request, response, model);**

**... do more stuff ...**

***// Return the template specified by Broadleaf (or not if you want to return your own template)***

**return returnPath;**

## Portability

### Platform

Broadleaf can be hosted regardless of what operation system as long as there is Java Virtual Machine running on it. It can be deployed to cloud such as Amazon EC2, Rackspace Cloud, CloudBees, CloudFoundry, Hiroku, and Amazon Elastic Beanstalk etc.

### Web/App container

Broadleaf Commerce is generally application server agnostic. Broadleaf requires a servlet container such as Apache Tomcat. Broadleaf has also been tested with Jetty, JBoss Application Server, Oracle WebLogic Application Server, and IBM WebSphere Application Server. You may also run certain non-web components of Broadleaf outside of a Servlet container (e.g. a batch process).

### Database

* RDBMS

Broadleaf Commerce is database agnostic in terms of RDBMS. You can use any relational database that Hibernate supports. We have done the majority of our testing with MySQL 5.1, Oracle 9i and 10g, and HSQLDB.

* NoSQL database

Not currently, as Hibernate advances to allow JPA mappings for NoSQL databases, it is in progress. This may be a drawback because user cannot take advantages from the high performance and other benefits NoSQL brings presently.

## Performance

Traditional RDBMS may become a bottleneck in an enterprise system. Broadleaf Commerce depends heavily on JPA and Hibernate with RDBMS. This ORM technology has its own benefits and drawbacks. It significantly reduces the effort to implement persistence mechanism. On the other hand, for an incredibly complex product it is, the performance will become a serious problem.

The way Hibernate basically promises to deliver performance is through caching and changing the amount of data that is fetched each time. This works for most of the cases where you use Hibernate. We haven’t found Hibernate has some query optimization mechanisms for different particular database such as Mysql, Oracle or PostgreSQL. Only generic ones like using parameter binding to allow query statement to be cached.

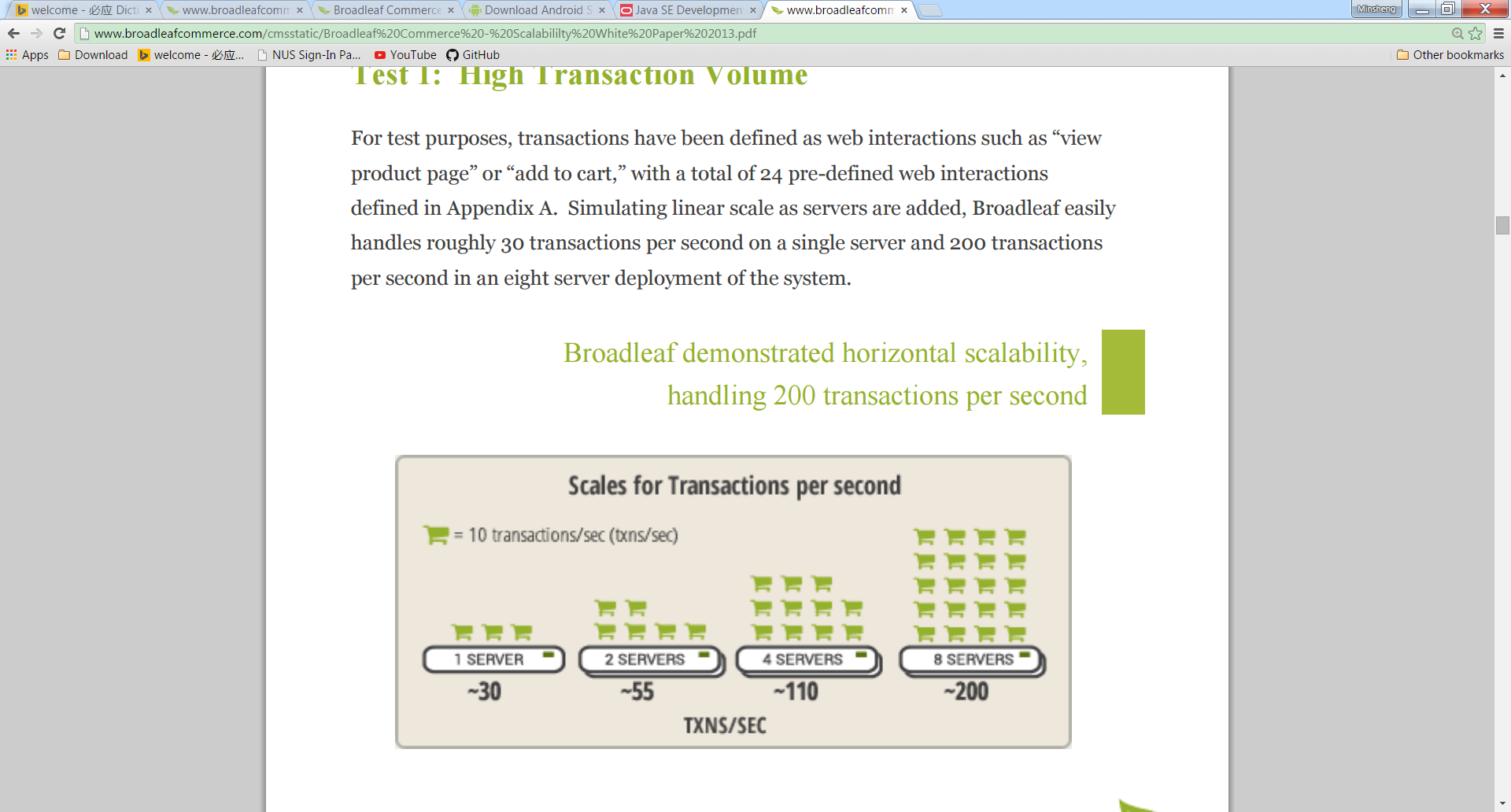
To improve performance, further configuration and target Database tuning technique is also required, like SQL clauses tuning, reasonable indexes, delicate table structures, data partitions etc.

## Scalability

Broadleaf Commerce (Broadleaf) provides companies with a platform for building high performance commerce solutions. There is an Enterprise Edition Scalability whitepaper provided which gives four types of scenarios that requires the system to scale out and the test result to exhibits consistent peak performance.

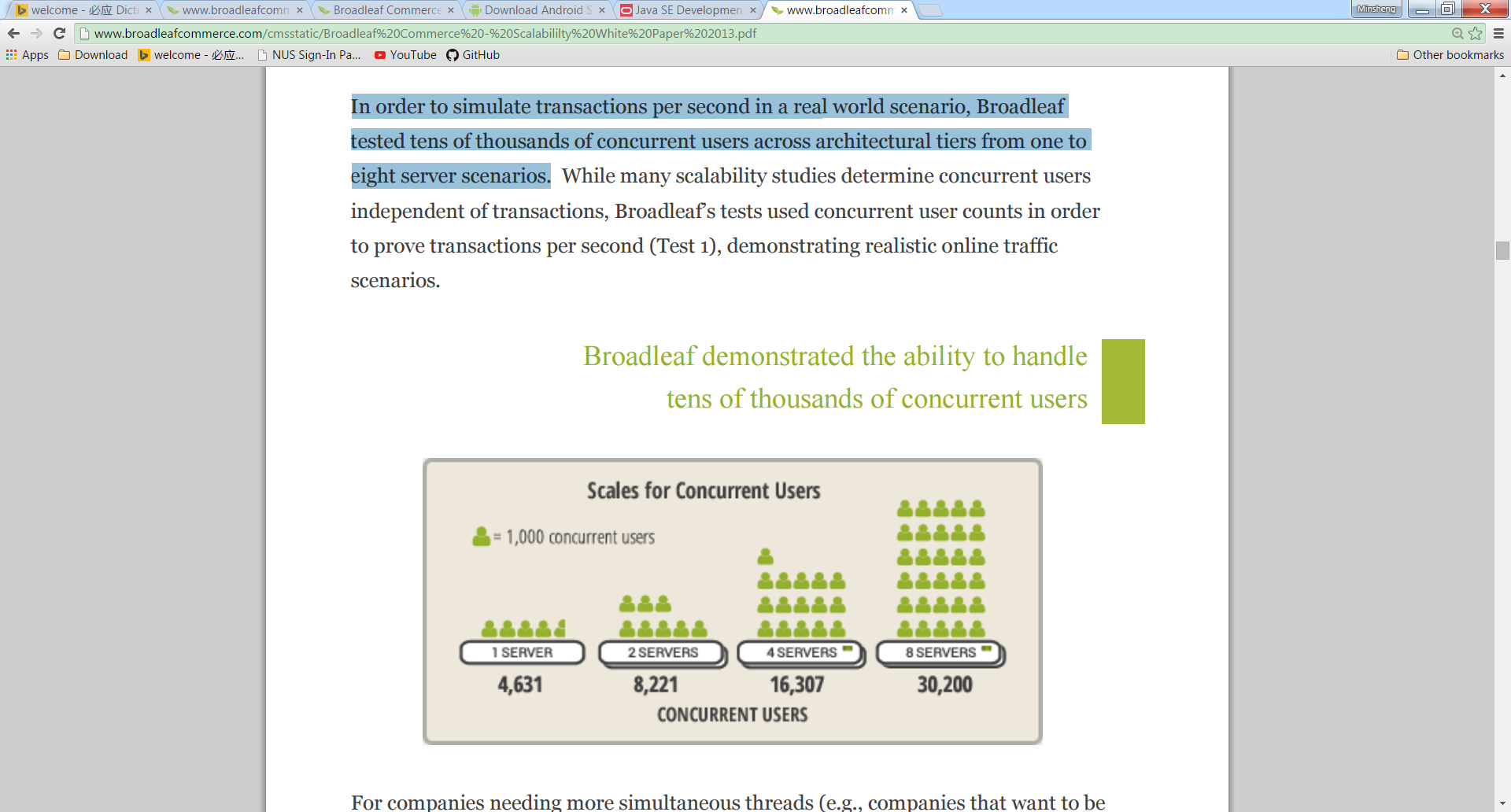
### High transaction volume

Simulating linear scale as servers are added, Broadleaf easily handles roughly 30 transactions per second on a single server and 200 transactions per second in an eight server deployment of the system.



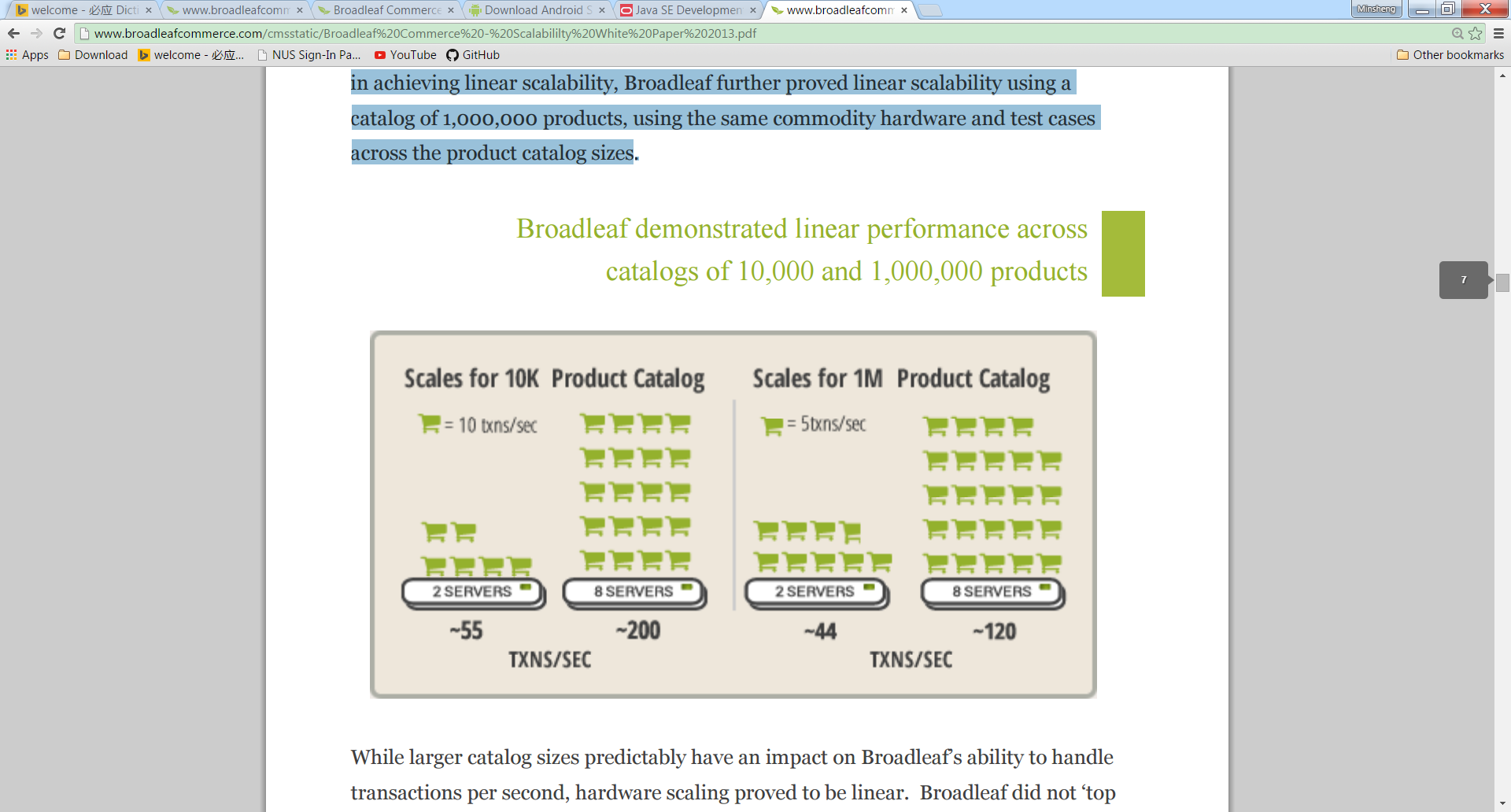
### Concurrent users

In order to simulate transactions per second in a real world scenario, Broadleaf tested tens of thousands of concurrent users across architectural tiers from one to eight server scenarios.



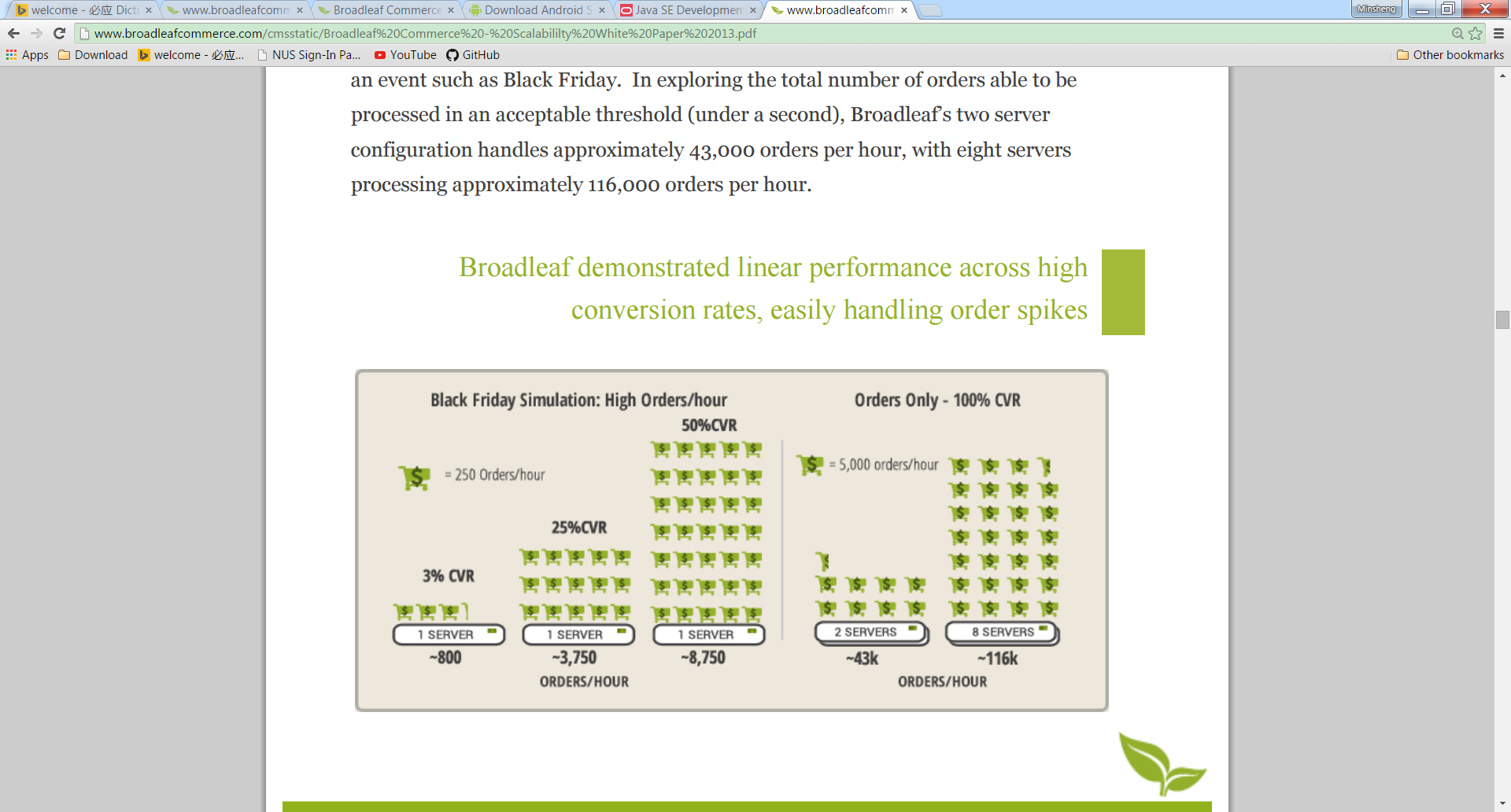
### Large catalogs

For corporations requiring larger catalog sets, Broadleaf tested an online catalog with 1,000,000 products. While Test 1 and 2 covered a catalog of 10,000 products in achieving linear scalability, Broadleaf further proved linear scalability using a catalog of 1,000,000 products, using the same commodity hardware and test cases across the product catalog sizes.



### High conversion rates

For corporations with increased conversion rates above industry averages, Broadleaf tested up to 100% conversion in proving, once again, linear scale.



## Conclusion

As an out of the box product as well as a framework for enterprise eCommerce business, Broadleaf satisfactorily fulfills the requirements as claimed, particularly in terms of the functional features, extensibility to customization and scalability to handle high volume of traffic.

However, there are some downsides as described above regarding to software quality, especially maintainability.

1. **Documentation**, the lack of detailed design activity and document, user may have to comprehend by re-engineering from the code to gain required understanding.
2. **Source code Readability**, some parts of the source code are extremely complicate with a lot of conditional branches. Moreover, the widely use of annotation and Spring AOP makes the code even harder to understand. It increases the cost and difficulty to maintain the system.
3. **Coverage of testing**, the Broadleaf team also realizes that the current coverage of testing is significantly insufficient that possibly will cause serious problem when the product running in production.
4. **Portability**, as stated presently the portability issue is on persistence layer. To make use of high performance NoSQL database, user has to implement their own persistence logic to replace the code on Hibernate.

In summary, Broadleaf can be used as a solid foundation to build a unique and powerful enterprise level eCommerce application. Indeed it has few problems must be taken into consideration, but meanwhile a dedicated project team and a community of developers with willing heart are solving problems and improving its quality.