**Javelin Programme**

Development Guidelines

[1. Introduction](#h.k3mmfwtoxgz1)

[1.1 Scope of this document](#h.zhcru4n0pp9t)

[1.2 Purpose of this document](#h.m5nsduvtvq78)

[1.3 Referenced documents](#h.w0503jvku7i2)

[2. General programming guidelines](#h.v40ezvvcte37)

[2.1 Cohesion](#h.o12xtcedzn3q)

[2.2 Coupling](#h.x3ig4e9rahhm)

[2.3 Understandability / Code Clarity](#h.shjbbctag2bq)

[2.4 Avoid Technical Debt](#h.og43wccayadx)

[2.5 Modularization](#h.4tc9fukdohgb)

[2.5.1 Module communications](#h.ud5o5vaa0prh)

[2.5.2 Modules creation](#h.ppa3h0570ikk)

[2.5.3 Modules organization](#h.byfaxhl9lcwp)

[2.6 Code style](#h.wqnqui2yedjs)

[2.6.1 Methods](#h.a5pedehq15uh)

[2.6.1.1 Method names](#h.4ewsbqmohug3)

[2.6.1.2 Method size](#h.lecwdz3wgspb)

[2.6.1.3 Method arguments](#h.iz5gkonxvkko)

[2.6.2 Comments and Documentation](#h.pgznpb7tciip)

[2.6.2.1 Information comments](#h.2o7mveckj15k)

[2.6.2.2 TODO comments](#h.lqft7jlxd9v3)

[2.6.2.3 Public API comments](#h.spgk2ix8zcde)

[2.6.2.4 Misleading and bad comments](#h.7u50w222fi9y)

[2.6.2.5 Complex code comments](#h.iunul7s2ewz4)

[2.6.2.6 Markdown README file](#h.3pxumc7786hy)

[2.6.2.7 Project documentation](#h.jjisbn9dsexp)

[2.6.3 Unfinished Code](#h.tgqcj93cnxjn)

[2.6.4 Cross References in Code to Other Documents](#h.xvocambqqzde)

[2.7 Localization](#h.1dqgdtabq5v0)

[3 Java specific issues](#h.jbetylkg2vo4)

[3.1 Modules in Java](#h.60uhmyrkvvz5)

[3.1.1 Module archetype](#h.43iig2w1gw7l)

[3.1.2 Module repository](#h.97u7llgxqd66)

[3.1.3 Module release](#h.rqwjyynid89l)

[3.2 Testing in Java](#h.juggoffrsuzl)

[3.2.1 BDD tests](#h.1r8fno6eluiq)

[3.2.2 TDD tests](#h.3374orhz9ie6)

[3.2.2.1 TDD life-cycle](#h.ojev9ih9jbco)

[3.2.3 Integration tests](#h.j49sgy54s2qx)

[3.2.4 Unit tests](#h.wnexo5ed8b5c)

[3.3 Design patterns in Java](#h.gi1ecaz0qw5s)

[3.3.1 Creational patterns](#h.d6hlg3l1oga1)

[3.3.1.2 Abstract factory](#h.qej513o7nmgs)

[3.3.1.3 Builder](#h.pv26cagrsqew)

[3.3.1.4 Factory method](#h.13qromcj577z)

[3.3.1.5 Prototype](#h.4bnolqsekawj)

[3.3.1.6 Singleton](#h.f2z1k4k7ekm)

[3.3.2 Structural patterns](#h.qsktlnprkiyu)

[3.3.2.1 Adapter](#h.mmpu6g8i4z3p)

[3.3.2.2 Bridge](#h.trspu2b7rjl4)

[3.3.2.3 Composite](#h.z7h5fc4r2bp2)

[3.3.2.4 Decorator](#h.5q0ljyy4g3j1)

[3.3.2.5 Facade](#h.do7soq75z620)

[3.3.2.6 Flyweight](#h.eo69lvaumdd)

[3.3.2.7 Proxy](#h.wkj6fz22moku)

[3.3.3 Behavioral patterns](#h.n3kh1grz0g8)

[3.3.3.1 Chain of responsibility](#h.alysj4nlszka)

[3.3.3.2 Command](#h.mw4jn6c1nlou)

[3.3.3.3 Interpreter](#h.sr7vlge3pvvi)

[3.3.3.4 Iterator](#h.jcdde8x1qpgh)

[3.3.3.5 Mediator](#h.m0z3owobroq0)

[3.3.3.6 Memento](#h.ble9kw7iz2c3)

[3.3.3.7 Observer (or Publish/Subscribe)](#h.y35jy9kiwo6v)

[3.3.3.8 State](#h.hdg9sgsr34ju)

[3.3.3.9 Strategy](#h.5rrwfp7xzgs8)

[3.3.3.10 Template method](#h.hs2t6fabk9cx)

[3.3.3.11 Visitor](#h.k5o5ioleva7q)

[3.4 Coding style in Java](#h.e8q3p4abgkj5)

[3.4.1 Classes](#h.ynxqj3z4r6zu)

[3.4.2 Interfaces](#h.24c3gghz8pkz)

[3.4.3 Enums and Constants](#h.kejj9nbis8ti)

[3.4.4 Exceptions](#h.k0nd3toracnh)

[3.5 Useful libraries in Java](#h.q3zqminpa6xj)

[3.5.1 Json](#h.cpreomry9h6b)

[3.5.2 Logging](#h.6wpwuhxt10r9)

[3.5.3 Date and Time](#h.tj8my3cuwofq)

[3.5.4 Testing](#h.o5y4i23mf36w)

[3.5.5 Maven](#h.cu3cdsltodi2)

[3.5.5.1 Unit and Functional Testing Plugins](#h.kzh8xed6uuvc)

[3.5.5.2 Code Quality Plugins](#h.3ron7pitpjim)

[3.5.5.3 Core](#h.cgoap0xj40rt)

[3.5.5.4 Commons](#h.4tp7kjnw6hjw)

[3.5.5.5 XML](#h.th7w82s4k9id)

[3.5.5.6 AOP](#h.bpqdk9jgcim8)

[3.5.5.7 Monitoring](#h.vqt64kafjdnm)

[3.5.5.8 Cron and Scheduling](#h.7aavj7rzjs73)

[4 Javascript specific issues](#h.fi3mscrfce0d)

[4.1 Modules in Javascript](#h.b0x8vi4tx9ll)

[4.1.1 Modules repository](#h.qkmz8y54rhpp)

[4.1.2 Module release](#h.a3r3mmh3fizr)

[4.2 Testing in Javascript](#h.y97gnd9d05d1)

[4.2.1 BDD tests](#h.7ybl3t40k3zo)

[4.2.2 Integration tests](#h.iu61ihhyp5b6)

[4.2.3 Unit tests](#h.yxn2nmuf3dtp)

[4.3 Design patterns in Javascript](#h.y97fnsmecvbc)

[4.4 Code style in Javascript](#h.lncedsk4esx3)

[4.5 Documentation in Javascript](#h.e8v0iemcn489)

[4.6 Useful libraries in Javascript](#h.bmc8tkmpwn17)

[4.7 Useful resources for Javascript](#h.962t3qn8v6a)

[5 Google app engine specific issues](#h.x07ivaj136gn)

[5.1 Modules in GAE](#h.u0ea9eo2jktu)

[5.2 Communication between GAE modules](#h.mcm1sdcl9r9n)

[5.3 Testing in GAE](#h.uug1k9bxng9w)

[5.4 Deploy a module to the cloud](#h.8u8l7m1ega8t)

[5.5 Secure an endpoint using Spring security](#h.gvnkc4l1lmak)

[5.6 Useful libraries for GAE](#h.9l7sirjv195w)

[6 Android specific issues](#h.wpxhzb7uoa99)

[6.1 Code style in Android](#h.vq78uxs6q7w2)

[6.2 Android testing](#h.iej3hy1h6lme)

[6.3 Useful libraries in Android](#h.xpdm85upu6f3)

[7 Source control VCS](#h.t1sd867iutdv)

[7.1 GIT Flow](#h.uoaervtjab9z)

[7.2 Tags strategy](#h.pyqqn479wabt)

[7.3 Commit comments](#h.9nvq3jydw0u9)

[7.4 Commit your changes](#h.ebhsyptq5557)

[7.5 Push your changes](#h.dzsoer8865ye)

[7.6 Pull and Merge changes](#h.b5ch45k4o4qs)

[7.7 Ignore files](#h.oouxwdqtlesa)

[8 Continuous integration](#h.cbx57s5dt2rq)

[8.1 Trigger a build](#h.vy99sr2cr56t)

[9 Continuous inspection](#h.ctb5t7oxtqf)

[9.1 Trigger the code inspection](#h.zduusbyaotnw)

[10 Dependency management](#h.ab7jz6v87khe)

[10.1 Overview](#h.ysufmjwkgtig)

[10.2 Versioning scheme](#h.60asdmkfwf3m)

[10.3 Maven](#h.ol5ord8hbmz6)

[10.4 Bower/Grunt](#h.hdpvxva2ufpy)

[11 Release guidelines](#h.whpon14u3nye)

[12 Performance](#h.iwkc7pi4wn0q)

[13 Confluence](#h.5vdfr9h5pvtg)

[14 Security](#h.a4ybjo96faoi)

# 1. Introduction

The G4S ‘Javelin’ Programme (or simply ‘Javelin’) is a cross-business and cross-discipline platform designed, at its most fundamental level, to unify the “Order to Cash” (OTC) business process among G4S’s many different businesses’ pillars and several different lines-of-business throughout the globe.

## 1.1 Scope of this document

This is a technical document, designed for consumption by the Javelin project development and technical teams. It will cover all aspects of software development practices in the context of the Javelin programme, from dependency management, to development, to deployment. It is not intended as a requirements document nor will it cover at any length the objectives of the programme at-large, from a business perspective. This document assumes that the reader has a mid-to-senior level understanding of software development, especially with regard to enterprise Java application design and development; it will not define nor explain fundamental programming concepts. Almost all industry-specific terminology can easily be researched and explained in detail using your favourite search engine.

## 1.2 Purpose of this document

This document aims, primarily, to cover establish a set of standards and patterns across the entire Javelin programme, which will span many Agile Scrum teams. Development process standardization is critical for consistent, high-quality code; this document aims to define those processes involved in the development of the various modules and components that will serve as the basis of the Javelin platform. These standards and patterns form the foundation of the development process and will be enforced either by static analysis tools or by code review. All code submitted to the Javelin code repository will be held against any and all standards expressed within this document; likewise, all development processes are expected to follow the guidelines set forth in this document. These standards apply to all levels and disciplines of development resources assigned to any Javelin programme project or subproject.

# 2. General programming guidelines

## 2.1 Cohesion

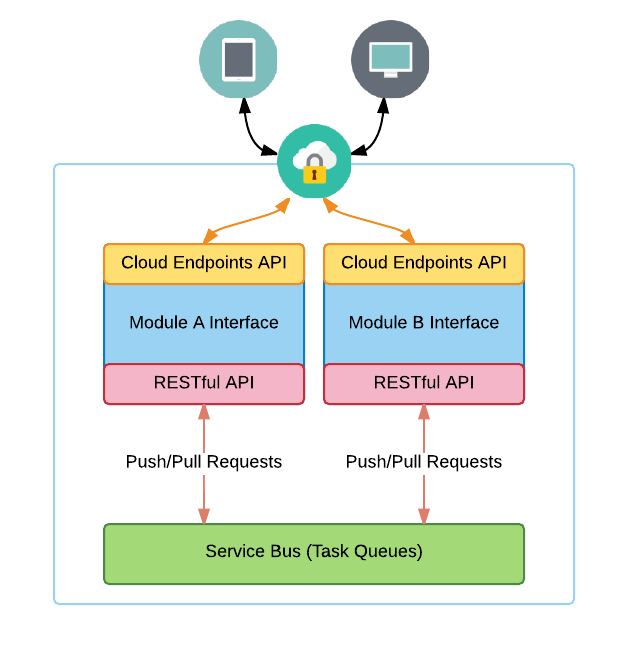
In order to reduce modules and components complexity and increase readability and maintainability we will promote [High Cohesion](https://en.wikipedia.org/wiki/Cohesion_(computer_science))[[1]](#footnote-1) everywhere: that is, the component, classes or modules must focus only on the job that they have to do and nothing else.

[Functional cohesion](http://www.site.uottawa.ca:4321/oose/functionalcohesion.html)[[2]](#footnote-2) is when parts of a module are grouped because they all contribute to a single well-defined task of the module (e.g. tokenizing a string of XML).

## 2.2 Coupling

In order to promote reusability and independency it is preferable to use a [loosely coupled](https://en.wikipedia.org/wiki/Loose_coupling)[[3]](#footnote-3) architecture for both components and modules. Before starting to develop a component or a module, we should try to break it down into definable elements to make them as more reusable and independent as we can.

In order to achieve a loosely coupled architecture in the context of the Javelin Programme, we will employ the use of RESTful interfaces over HTTPS, using Google Cloud Platform [Task Queues](https://cloud.google.com/appengine/docs/java/taskqueue/)[[4]](#footnote-4) to function as a “[service bus](https://www.mulesoft.com/resources/esb/what-esb)[[5]](#footnote-5)”-like mechanism, as the following diagram illustrates:



## 2.3 Understandability / Code Clarity

We have to write code understandable by anybody and not only by ourselves. In order to achieve that, there are some basic rules to follow:

* Use pronounceable, searchable and meaningful names for variables, methods, classes and interfaces
* Try to create methods with 1 or fewer arguments and prefer the Builder patterns when you have to use more arguments
* Try to create small functions with high cohesion and promote encapsulation to hide the complexity
* Avoid ambiguous overloading
* Avoid many levels of inheritance and try to promote programming to an interface;
* Explain yourself in the code using comments
* Keep it simple; try to avoid over-engineering.

## 

## 2.4 Avoid Technical Debt

[Technical debt](https://en.wikipedia.org/wiki/Technical_debt)[[6]](#footnote-6) (or, “tech debt”) is a consequence of poor design, poor coding and lack of testing. The more technical debt, the more difficult will be to maintain and extend the codebase, and it will impact the long-term robustness, stability and performance of the application.

As programmers we can minimize the technical debt following some basic principles:

* Frequent testing (not just Unit Tests -- manually test your own code) and pay attention to details - as programmers we are also testers: even if there are QA people in your team, before delivering anything please test it 10 times
* Good design - try to use design patterns as often as you can
* Peer review - allow other developers to review your code and use pair programming as needed. Initiate code review during the pull request once you have finished your dev.
* Use a continuous inspection tool to inspect your code constantly. Use plugins like Yasca, findbug and CPD.
* Use a continuous integration server - commit your code daily, or several times a day to trigger the continuous integration server to validate your build
* Ensure that a supporting test case is written for every bug found. Stick to the working principle of “Do not repeat the same defect”.

## 2.5 Modularization

The entire application codebase will feature a modular architecture for both back-end and front-end solutions and their componentry.

The benefits of modularization are:

* Scalable development - modular design allows asynchronous project flow
* Single responsibility - each module have only one single reason to change
* Versioning - each module has a version number and when a new version is released it has an increased version number
* Easier modification, maintenance, and deployment - developers can make changes to a module, and deploy it without affecting other modules
* Dependency management - each module will live into a repository manager that makes sure the module version is unique and manages its metadata
* Building robust systems - modularity allows the construction of complex systems composed of smaller parts that can be independently managed and maintained.

### 

### 2.5.1 Module communications

Modules will communicate with one another through their APIs. When we create a module we will create 2 libraries: one for the API, one for the implementation.

The benefit of this architecture are:

* Ability to change the implementation without affecting the APIs
* Ability to change the APIs only for some libraries.

**Example**:

Consider a module person; it is splitted into *person-api-1.0.0* and *person-implementation-1.0.0*. Consider 2 other modules intended to communicate with the person module -- *person-manager* and *person-bus* -- that both need to communicate with the person module.

At some point we need to add a new API to the person module only for the *person-manager* module.

With our architecture we will have only to release a new version of *person-api* and *person-implementation* and tell to the *person-manager* to use the new version without touching the person-bus module at all.

Consequently, Google Cloud Platform natively supports the module pattern with the concept of [App Engine Modules](https://cloud.google.com/appengine/docs/java/modules/)[[7]](#footnote-7), which are independently deployable and have their own versions and instances within the same App Engine Project; while they operate (and may be deployed) independently from one another, however, they share a set of services that are global to all sister modules within the same Cloud Platform Project.

Throughout this document, we will refer to ‘modules’ as both a grouping of work from a conceptual architectural standpoint **and** from the more concrete, App Engine Modules perspective, since they are essentially the same thing.

### 2.5.2 Modules creation

In order to create a new module, we will provide an [archetype](https://maven.apache.org/guides/introduction/introduction-to-archetypes.html)[[8]](#footnote-8) for developers that they can use to create a new module without having to do all the setup and configuration.

### 2.5.3 Modules organization

All modules source code will live in the Javelin programme Git repository; each module will be a separate Git project. The compiled version of each module will live in its respective application repository manager: [Nexus](http://www.sonatype.com/nexus/product-overview)[[9]](#footnote-9) for Java and [Bower](http://bower.io/)[[10]](#footnote-10) for Javascript modules.

In the project we will use aggregators and package manager configurators.

Aggregators are files where we can specify which module we want to build while package manager's configuration file are files where we describe which dependencies we need in our project.

## 2.6 Code style

### 2.6.1 Methods

Methods and functions, if written properly, can describe what the the code is doing -- improving the maintainability and readability of the code.

#### 2.6.1.1 Method names

Use descriptive names for methods and functions. **An enigmatic or short name makes the life of other people more difficult**. A long and descriptive name is better than a long comment. Please choose the name carefully and write them in camel case.

#### 2.6.1.2 Method size

Functions have to be small and have one purpose. Please use narrative functions and promote [encapsulation](https://en.wikipedia.org/wiki/Encapsulation_(computer_programming))[[11]](#footnote-11) to achieve that. Example:

void createAccount(User user) {

validateAccount(user);

saveAccount(user);

notifyAccount(user);

}

#### 2.6.1.3 Method arguments

As mentioned before, **the ideal number of arguments for a function is zero**. One or 2 argument is still acceptable but not 3. Try to use the [Builder pattern](https://en.wikipedia.org/wiki/Builder_pattern)[[12]](#footnote-12) if you have to build object where you need to set a lot of arguments.

2.6.1.4 Method responsibility

A function should either *do something* or *return something* but not both. Please separate the command set from query. A function should also serve one purpose.

#### 

### 2.6.2 Comments and Documentation

Comments and documentation are mandatory on all Javelin programme software projects.

#### 2.6.2.1 Information comments

Each class, script, file must have an header that describe the list of authors, the creation and update dates and and explanation of the intent of it.

#### 2.6.2.2 TODO comments

There should be no TODO comments in the code..

#### 2.6.2.3 Public API comments

Each API must have a very well described comment and examples of how to use it.

#### 2.6.2.4 Misleading and bad comments

”Self-documenting code” practice should be followed.

#### 2.6.2.5 Complex code comments

If there is a complex algorithm implementation then use comments only to describe the intention rather than the implementation

#### 

#### 2.6.2.6 Markdown README file

Each module/project must have a README.md containing all the information to allow the developer to get up to speed with it in terms of development, mantainance, and building.

The README file should use [Markdown syntax](https://en.wikipedia.org/wiki/Markdown)[[13]](#footnote-13).

#### 2.6.2.7 Project documentation

Use confluence to keep the project documentation.

The important documentation to share with the team are:

* Project knowledge base
* Project technical blueprint
* Project development guidelines
* Project tools.

### 2.6.3 Unfinished Code

If you know that the code is non-functional or defective then it should never be committed. If you have to commit for any unavoidable reason, then please create your own branch in git and commit it there.

### 

### 2.6.4 Cross References in Code to Other Documents

In many instances a section of code might be tied or refer to some other documents or external sources.

If a module/class contains some cross references to other documents or external sources, there should be a comment that provides the reference to that other document.

In Java (JavaDoc) and Javascript (JSDoc) we achieve cross reference using comments with @see and or @link.

Example:

/\*\*  
 \* @see <a href="http://google.com">http://google.com</a>  
 \*/

/\*\*

\*{@link somepackage.SomeClass#someMethod(paramTypes)}.

\*/

## 2.7 Localization

In a global project, Localization is one of the pillar to remember for the development of each module for both back-end and front-end. The localization items have to be decoupled completely from the code.

# 3 Java specific issues

## 3.1 Modules in Java

When we say an application is **modular**, we generally mean it's composed of a set of highly decoupled, distinct pieces of functionality stored in modules.

### 3.1.1 Module archetype

In order to facilitate the setup of a new module into the development environment, we will use a custom Maven archetype that will generate the entire Maven project with all the dependencies and resources needed.

The developer will have only to use the Maven command $ mvn archetype:generate and select the archetype.

### 3.1.2 Module repository

All our Java libraries releases will live into our Nexus repository.

### 3.1.3 Module release

As mentioned above our Module archetype will contain all the configuration to allow to publish SNAPSHOTS or stable versions to our private repository simply using the Maven release, prepare and deploy commands.

## 3.2 Testing in Java

### 3.2.1 BDD tests

[BDD](https://en.wikipedia.org/wiki/Behavior-driven_development) is an evolution of test-driven development (TDD) and acceptance-test driven design, and is intended to make these practices more accessible and intuitive to newcomers and experts alike. It shifts the vocabulary from being test-based to behaviour-based, and positions itself as a design philosophy.

In Java there a several frameworks that allow us to develop using the BDD approach, and one of the most popular is [Cucumber for Java.](https://cucumber.io/docs/reference/jvm#java)

Cucumber allows us to describe the behavior in plain text using [gherkin](https://github.com/cucumber/cucumber/wiki/Given-When-Then) and write test against the business story.

Our approach to templating a scenario

given (some context)  
when (something happens)  
then (some behavioral validation)

Some of these user scenarios can be written by Business Analysts and in Test First Development approach we can write the test specifications and fail them and then write the business functionality to satisfy the tests and pass in an iterative way.

Example:

Acceptance criteria:

Ability: Parsing User Input  
 Background:  
 Given the game parser  
   
 Scenario: Parse a valid single word command  
 When the command "inventory" is parsed  
 Then the verb is "inventory"  
 And there is no direct object  
 And the parser response is "Success

Cucumber Java Test:

public class ParserSteps {

@Given("^the game parser$")

public void the\_game\_parser() throws Throwable {

// Write code here that turns the phrase above into concrete actions

throw new PendingException();

}

@When("^the command \"(.\*?)\" is parsed$")

public void the\_command\_is\_parsed(String arg1) throws Throwable {

// Write code here that turns the phrase above into concrete actions

throw new PendingException();

}

@Then("^the verb is \"(.\*?)\"$")

public void the\_verb\_is(String arg1) throws Throwable {

// Write code here that turns the phrase above into concrete actions

throw new PendingException();

}

@Then("^there is no direct object$")

public void there\_is\_no\_direct\_object() throws Throwable {

// Write code here that turns the phrase above into concrete actions

throw new PendingException();

}

@Then("^the parser response is \"(.\*?)\"$")

public void the\_parser\_response\_is(String arg1) throws Throwable {

// Write code here that turns the phrase above into concrete actions

throw new PendingException();

}

}

### 3.2.2 TDD tests

#### 3.2.2.1 TDD life-cycle

1. Write the test
2. Run the test (there is no implementation code, test does not pass)
3. Write just enough implementation code to make the test pass
4. Run all tests (tests pass)
5. Refactor
6. Repeat.

Test-driven development is not about testing. Test-driven development is about development (and design), specifically improving the quality and design of code. The resulting unit tests are just an extremely useful by-product.

### 3.2.3 Integration tests

Generally there is not a clear distinction on what is an integration test and what is a unit test.

Our basic rule of thumb is that if

1. a test uses the database
2. a test uses the network
3. a test uses an external system (e.g. a queue or a mail server)
4. a test reads/writes files or performs other I/O

…then it is an integration test and not a unit test.

### 3.2.4 Unit tests

[JUnit](http://www.javaworld.com/article/2073090/testing-debugging/getting-started-with-test-driven-development.html#resources) is the primary test tool for satisfying the first requirement for a Java test framework. JUnit uses reflection to examine the tests and code under tests. This allows JUnit to execute any method of any class and examine the results. We interpret test results using the following nomenclature

* TestCase: Abstract class for implementing a basic unit test
* TestSuite: Composite class for organizing and running groups of tests
* Assertions: For testing expected results (assertNotNull(..), assertEquals(..), assertSame(..), etc.)
* Failure: Indicates a checked test assertion failed (i.e., assertNotNull(..) returned false)
* Error: Indicates an unexpected exception or setup failure that stopped the test.

## 3.3 Design patterns in Java

Before reinventing the wheel we should always try to solve our problem using or adapting an existing pattern. This makes the solution easier to comprehend and faster to develop. Because these patterns have proven track record as they are already widely used and thus reduce the technical risk to the project. Java itself uses lots of common design patterns internally.

### 3.3.1 [Creational patterns](http://en.wikipedia.org/wiki/Creational_pattern)

#### 3.3.1.2 [Abstract factory](http://en.wikipedia.org/wiki/Abstract_factory_pattern)

The intent of abstract factory or kit is to allow the creation of families of related or dependent object. Recognizable by creational methods returning the factory itself which in turn can be used to create another abstract/interface type.

Usage:

Use it when you need a sets of methods to make various objects.

Examples:

* [javax.xml.parsers.DocumentBuilderFactory#newInstance()](http://docs.oracle.com/javase/6/docs/api/javax/xml/parsers/DocumentBuilderFactory.html#newInstance%28%29)
* [javax.xml.transform.TransformerFactory#newInstance()](http://docs.oracle.com/javase/6/docs/api/javax/xml/transform/TransformerFactory.html#newInstance%28%29)
* [javax.xml.xpath.XPathFactory#newInstance()](http://docs.oracle.com/javase/6/docs/api/javax/xml/xpath/XPathFactory.html#newInstance%28%29)

#### 3.3.1.3 [Builder](http://en.wikipedia.org/wiki/Builder_pattern)

The intent of the builder pattern is to move the construction logic for an object outside the class to be instantiated.This is useful when we don't have all the information needed for an object when it is time to construct it.

Recognizable by creational methods returning the instance itself.

Usage:

Use it when you need to build an object and you don’t have all the information or you need to pass many parameters to the constructor.

Examples:

* [java.lang.StringBuilder#append()](http://docs.oracle.com/javase/6/docs/api/java/lang/StringBuilder.html#append%28boolean%29) (unsynchronized)
* [java.lang.StringBuffer#append()](http://docs.oracle.com/javase/6/docs/api/java/lang/StringBuffer.html#append%28boolean%29) (synchronized)
* [java.nio.ByteBuffer#put()](http://docs.oracle.com/javase/6/docs/api/java/nio/ByteBuffer.html#put%28byte%29) (also on [CharBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/CharBuffer.html#put%28char%29), [ShortBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/ShortBuffer.html#put%28short%29), [IntBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/IntBuffer.html#put%28int%29), [LongBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/LongBuffer.html#put%28long%29), [FloatBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/FloatBuffer.html#put%28float%29) and [DoubleBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/DoubleBuffer.html#put%28double%29))
* [javax.swing.GroupLayout.Group#addComponent()](http://docs.oracle.com/javase/6/docs/api/javax/swing/GroupLayout.Group.html#addComponent%28java.awt.Component%29)
* All implementations of [java.lang.Appendable](http://docs.oracle.com/javase/6/docs/api/java/lang/Appendable.html)

#### 3.3.1.4 [Factory method](http://en.wikipedia.org/wiki/Factory_method_pattern)

The intent of the factory method pattern is to let a class define the interface for creating object. In Java a good example is the iterator. Iterators are created often using a factory method.

Recognizable by creational methods returning an implementation of an abstract/interface type.

Usage:

Use it when you have to make and return components of one object various ways.

Examples::

* [java.util.Calendar#getInstance()](http://docs.oracle.com/javase/6/docs/api/java/util/Calendar.html#getInstance%28%29)
* [java.util.ResourceBundle#getBundle()](http://docs.oracle.com/javase/6/docs/api/java/util/ResourceBundle.html#getBundle%28java.lang.String%29)
* [java.text.NumberFormat#getInstance()](http://docs.oracle.com/javase/6/docs/api/java/text/NumberFormat.html#getInstance%28%29)
* [java.nio.charset.Charset#forName()](http://docs.oracle.com/javase/6/docs/api/java/nio/charset/Charset.html#forName%28java.lang.String%29)

#### 3.3.1.5 [Prototype](http://en.wikipedia.org/wiki/Prototype_pattern)

The intent of the prototype pattern is to provide new objects by copying an example rather than creating on new.

Recognizable by creational methods returning a different instance of itself with the same properties.

Usage:

Use it when you have to make new objects by cloning the objects which you set as prototypes.

Examples:

* [java.lang.Object#clone()](http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html#clone%28%29) (the class has to implement [java.lang.Cloneable](http://docs.oracle.com/javase/6/docs/api/java/lang/Cloneable.html))

#### 3.3.1.6 [Singleton](http://en.wikipedia.org/wiki/Singleton_pattern)

The intent of the Singleton pattern is to ensure that the class is instantiated only once.

Recognizable by creational methods returning the same instance (usually of itself) everytime.

Usage:

Use it when you want to have only one instance and reuse that instance of a class.

Examples used internally:

* [java.lang.Runtime#getRuntime()](http://docs.oracle.com/javase/6/docs/api/java/lang/Runtime.html#getRuntime%28%29)
* [java.awt.Desktop#getDesktop()](http://docs.oracle.com/javase/6/docs/api/java/awt/Desktop.html#getDesktop%28%29)

### 3.3.2 [Structural patterns](http://en.wikipedia.org/wiki/Structural_pattern)

#### 3.3.2.1 [Adapter](http://en.wikipedia.org/wiki/Adapter_pattern)

The intent of the the Adapter pattern is to provide the interface that a client expects while

using the service of a class with a different interface.

Recognizable by creational methods taking an instance of different abstract/interface type and returning an implementation of own/another abstract/interface type which decorates/overrides the given instance.

Usage:

Use it when you have to extend a class, takes in an object, and makes the taken object behave like the extended class.

Examples used internally:

* [java.util.Arrays#asList()](http://docs.oracle.com/javase/6/docs/api/java/util/Arrays.html#asList%28T...%29)
* [java.io.InputStreamReader(InputStream)](http://docs.oracle.com/javase/6/docs/api/java/io/InputStreamReader.html#InputStreamReader%28java.io.InputStream%29) (returns a Reader)
* [java.io.OutputStreamWriter(OutputStream)](http://docs.oracle.com/javase/6/docs/api/java/io/OutputStreamWriter.html#OutputStreamWriter%28java.io.OutputStream%29) (returns a Writer)
* [javax.xml.bind.annotation.adapters.XmlAdapter#marshal()](http://docs.oracle.com/javase/6/docs/api/javax/xml/bind/annotation/adapters/XmlAdapter.html#marshal%28BoundType%29) and [#unmarshal()](http://docs.oracle.com/javase/6/docs/api/javax/xml/bind/annotation/adapters/XmlAdapter.html#unmarshal%28ValueType%29)

#### 3.3.2.2 [Bridge](http://en.wikipedia.org/wiki/Bridge_pattern)

Recognizable by creational methods taking an instance of different abstract/interface type and returning an implementation of own abstract/interface type which delegates/uses the given instance.

Usage:

An abstraction and implementation are in different class hierarchies.

#### 3.3.2.3 [Composite](http://en.wikipedia.org/wiki/Composite_pattern)

Recognizable by behavioral methods taking an instance of same abstract/interface type into a tree structure.

Usage:

Assemble groups of objects with the same signature.

Examples of composite pattern used internally:

* [java.awt.Container#add(Component)](http://docs.oracle.com/javase/6/docs/api/java/awt/Container.html#add%28java.awt.Component%29) (practically all over Swing thus)
* [javax.faces.component.UIComponent#getChildren()](http://docs.oracle.com/javaee/6/api/javax/faces/component/UIComponent.html#getChildren%28%29) (practically all over JSF UI thus)

#### 

#### 3.3.2.4 [Decorator](http://en.wikipedia.org/wiki/Decorator_pattern)

The intent of the decorator pattern is to attach new responsibilities to an object dynamically.

Recognizable by creational methods taking an instance of same abstract/interface type which adds additional behaviour.

Usage:

One class takes in another class, both of which extend the same abstract class, and adds functionality.

Examples used internally:

* All subclasses of [java.io.InputStream](http://docs.oracle.com/javase/6/docs/api/java/io/InputStream.html), [OutputStream](http://docs.oracle.com/javase/6/docs/api/java/io/OutputStream.html), [Reader](http://docs.oracle.com/javase/6/docs/api/java/io/Reader.html) and [Writer](http://docs.oracle.com/javase/6/docs/api/java/io/Writer.html) have a constructor taking an instance of same type.
* [java.util.Collections](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html), the [checkedXXX()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#checkedCollection%28java.util.Collection,%20java.lang.Class%29), [synchronizedXXX()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#synchronizedCollection%28java.util.Collection%29) and [unmodifiableXXX()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#unmodifiableCollection%28java.util.Collection%29) methods.
* [javax.servlet.http.HttpServletRequestWrapper](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequestWrapper.html) and [HttpServletResponseWrapper](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponseWrapper.html)

#### 3.3.2.5 [Facade](http://en.wikipedia.org/wiki/Facade_pattern)

The intent of Faced is to provide an interface that makes a subsystem easy to use.

Recognizable by behavioral methods which internally uses instances of different independent abstract/interface types.

Usage:

One class has a method that performs a complex process calling several other classes.

Examples:

* [javax.faces.context.FacesContext](http://docs.oracle.com/javaee/6/api/javax/faces/context/FacesContext.html), it internally uses among others the abstract/interface types [LifeCycle](http://docs.oracle.com/javaee/6/api/javax/faces/lifecycle/Lifecycle.html), [ViewHandler](http://docs.oracle.com/javaee/6/api/javax/faces/application/ViewHandler.html), [NavigationHandler](http://docs.oracle.com/javaee/6/api/javax/faces/application/NavigationHandler.html) and many more without that the enduser has to worry about it (which are however overridable by injection).
* [javax.faces.context.ExternalContext](http://docs.oracle.com/javaee/6/api/javax/faces/context/ExternalContext.html), which internally uses [ServletContext](http://docs.oracle.com/javaee/6/api/javax/servlet/ServletContext.html), [HttpSession](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpSession.html), [HttpServletRequest](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html), [HttpServletResponse](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html), etc.

#### 3.3.2.6 [Flyweight](http://en.wikipedia.org/wiki/Flyweight_pattern)

Recognizable by creational methods returning a cached instance, a bit the "multiton" idea.

Usage:

The reusable and variable parts of a class are broken into two classes to save resources.

Examples:

* [java.lang.Integer#valueOf(int)](http://docs.oracle.com/javase/6/docs/api/java/lang/Integer.html#valueOf%28int%29) (also on [Boolean](http://docs.oracle.com/javase/6/docs/api/java/lang/Boolean.html#valueOf%28boolean%29), [Byte](http://docs.oracle.com/javase/6/docs/api/java/lang/Byte.html#valueOf%28byte%29), [Character](http://docs.oracle.com/javase/6/docs/api/java/lang/Character.html#valueOf%28char%29), [Short](http://docs.oracle.com/javase/6/docs/api/java/lang/Short.html#valueOf%28short%29) and [Long](http://docs.oracle.com/javase/6/docs/api/java/lang/Long.html#valueOf%28long%29))

#### 3.3.2.7 [Proxy](http://en.wikipedia.org/wiki/Proxy_pattern)

The intent of the proxy pattern is to control access to an object by providing a surrogate or

placeholder for it

Recognizable by creational methods which returns an implementation of given abstract/interface type which in turn delegates/uses a different implementation of given abstract/interface type.

Usage:

One class controls the creation of and access to objects in another class. Use it when you need to control remote objects, cache, or control the access ( security ) to an object.

Examples:

* [java.lang.reflect.Proxy](http://docs.oracle.com/javase/6/docs/api/java/lang/reflect/Proxy.html)
* [java.rmi.\*](http://docs.oracle.com/javase/6/docs/api/java/rmi/package-summary.html), the whole API actually.

The Wikipedia example is IMHO a bit poor, lazy loading has actually completely nothing to do with the proxy pattern at all.

### 3.3.3 [Behavioral patterns](http://en.wikipedia.org/wiki/Behavioral_pattern)

#### 3.3.3.1 [Chain of responsibility](http://en.wikipedia.org/wiki/Chain_of_responsibility_pattern)

Recognizable by behavioral methods which (indirectly) invokes the same method in another implementation of same abstract/interface type in a queue

Examples of chain of responsibility used internally:

* [java.util.logging.Logger#log()](http://docs.oracle.com/javase/6/docs/api/java/util/logging/Logger.html#log%28java.util.logging.Level,%20java.lang.String%29)
* [javax.servlet.Filter#doFilter()](http://docs.oracle.com/javaee/6/api/javax/servlet/Filter.html#doFilter%28javax.servlet.ServletRequest,%20javax.servlet.ServletResponse,%20javax.servlet.FilterChain%29)

Example:

A method called in one class can move up a hierarchy to find an object that can properly execute the method.

#### 3.3.3.2 [Command](http://en.wikipedia.org/wiki/Command_pattern)

The intent of a command pattern is to encapsulate a request in an object.

Recognizable by behavioral methods in an abstract/interface type which invokes a method in an implementation of a different abstract/interface type which has been encapsulated by the command implementation during its creation.

Usage:

An object encapsulates everything needed to execute a method in another object.

Example of Command pattern used internally:

* All implementations of [java.lang.Runnable](http://docs.oracle.com/javase/6/docs/api/java/lang/Runnable.html)
* All implementations of [javax.swing.Action](http://docs.oracle.com/javase/6/docs/api/javax/swing/Action.html)

#### 3.3.3.3 [Interpreter](http://en.wikipedia.org/wiki/Interpreter_pattern)

Recognizable by behavioral methods returning a structurally different instance/type of the given instance/type; note that parsing/formatting is not part of the pattern, determining the pattern and how to apply it is.

Usage:

Define a macro language and syntax, parsing input into objects which perform the correct operations.

Examples:

* [java.util.Pattern](http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html)
* [java.text.Normalizer](http://docs.oracle.com/javase/6/docs/api/java/text/Normalizer.html)
* All subclasses of [java.text.Format](http://docs.oracle.com/javase/6/docs/api/java/text/Format.html)
* All subclasses of [javax.el.ELResolver](http://docs.oracle.com/javaee/6/api/javax/el/ELResolver.html)

#### 3.3.3.4 [Iterator](http://en.wikipedia.org/wiki/Iterator_pattern)

Recognizable by behavioral methods sequentially returning instances of a different type from a queue).

Usage:

One object can traverse the elements of another object.

Example of the Iterator pattern used internally:

* All implementations of [java.util.Iterator](http://docs.oracle.com/javase/6/docs/api/java/util/Iterator.html) (thus among others also [java.util.Scanner](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html)!).
* All implementations of [java.util.Enumeration](http://docs.oracle.com/javase/6/docs/api/java/util/Enumeration.html)

#### 3.3.3.5 [Mediator](http://en.wikipedia.org/wiki/Mediator_pattern)

Recognizable by behavioral methods taking an instance of different abstract/interface type (usually using the command pattern) which delegates/uses the given instance).

Usage:

An object distributes communication between two or more objects.

Example:

* [java.util.Timer](http://docs.oracle.com/javase/6/docs/api/java/util/Timer.html) (all scheduleXXX() methods)
* [java.util.concurrent.Executor#execute()](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/Executor.html#execute%28java.lang.Runnable%29)
* [java.util.concurrent.ExecutorService](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ExecutorService.html) (the invokeXXX() and submit() methods)
* [java.util.concurrent.ScheduledExecutorService](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ScheduledExecutorService.html) (all scheduleXXX() methods)
* [java.lang.reflect.Method#invoke()](http://docs.oracle.com/javase/6/docs/api/java/lang/reflect/Method.html#invoke%28java.lang.Object,%20java.lang.Object...%29)

#### 3.3.3.6 [Memento](http://en.wikipedia.org/wiki/Memento_pattern)

The intent of the memento pattern is to provide storage and restoration of object state.

An example is the undo/redo

Recognizable by behavioral methods which internally changes the state of the whole instance).

Usage:

One object stores another object state.

Example:

* [java.util.Date](http://docs.oracle.com/javase/6/docs/api/java/util/Date.html) (the setter methods do that, Date is internally represented by a long value)
* All implementations of [java.io.Serializable](http://docs.oracle.com/javase/6/docs/api/java/io/Serializable.html)
* All implementations of [javax.faces.component.StateHolder](http://docs.oracle.com/javaee/6/api/javax/faces/component/StateHolder.html)

#### 

#### 3.3.3.7 [Observer (or Publish/Subscribe)](http://en.wikipedia.org/wiki/Observer_pattern)

The intent of observer is to define one to many dependency between objects so that when

one object change stat, all its dependent are notified and can react to the change

Recognizable by behavioral methods which invokes a method on an instance of another abstract/interface type, depending on own state.

Usage:

An object notifies other object(s) if it changes.

Examples:

* [java.util.Observer](http://docs.oracle.com/javase/6/docs/api/java/util/Observer.html)/[java.util.Observable](http://docs.oracle.com/javase/6/docs/api/java/util/Observable.html)
* All implementations of [java.util.EventListener](http://docs.oracle.com/javase/6/docs/api/java/util/EventListener.html)
* [javax.servlet.http.HttpSessionBindingListener](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpSessionBindingListener.html)
* [javax.servlet.http.HttpSessionAttributeListener](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpSessionAttributeListener.html)

#### 3.3.3.8 [State](http://en.wikipedia.org/wiki/State_pattern)

Recognizable by behavioral methods which changes its behaviour depending on the instance's state which can be controlled externally.

Usage:

An object appears to change its` class when the class it passes calls through to switches itself for a related class.

Example:

* [javax.faces.lifecycle.LifeCycle#execute()](http://docs.oracle.com/javaee/6/api/javax/faces/lifecycle/Lifecycle.html#execute%28javax.faces.context.FacesContext%29) (controlled by [FacesServlet](http://docs.oracle.com/javaee/6/api/javax/faces/webapp/FacesServlet.html), the behaviour is dependent on current phase (state) of JSF lifecycle)

#### 3.3.3.9 [Strategy](http://en.wikipedia.org/wiki/Strategy_pattern)

The intend of the strategy pattern is to encapsulate alternative approaches or strategies in

separate classes.

Recognizable by behavioral methods in an abstract/interface type which invokes a method in an implementation of a different abstract/interface type which has been passed-in as method argument into the strategy implementation.

Usage:

An object controls which of a family of methods is called. Each method is in its` own class that extends a common base class.

Example of the Strategy pattern used internally:

* [java.util.Comparator#compare()](http://docs.oracle.com/javase/6/docs/api/java/util/Comparator.html#compare%28T,%20T%29), executed by among others Collections#sort().
* [javax.servlet.http.HttpServlet](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServlet.html), the service() and all doXXX() methods take HttpServletRequest and HttpServletResponse and the implementor has to process them.
* [javax.servlet.Filter#doFilter()](http://docs.oracle.com/javaee/6/api/javax/servlet/Filter.html#doFilter%28javax.servlet.ServletRequest,%20javax.servlet.ServletResponse,%20javax.servlet.FilterChain%29)

#### 3.3.3.10 [Template method](http://en.wikipedia.org/wiki/Template_method_pattern)

Recognizable by behavioral methods which already have a "default" behaviour defined by an abstract type.

Usage:

An abstract class defines various methods, and has one non-overridden method which calls the various methods.

Examples:

* All non-abstract methods of [java.io.InputStream](http://docs.oracle.com/javase/6/docs/api/java/io/InputStream.html), [java.io.OutputStream](http://docs.oracle.com/javase/6/docs/api/java/io/OutputStream.html), [java.io.Reader](http://docs.oracle.com/javase/6/docs/api/java/io/Reader.html) and [java.io.Writer](http://docs.oracle.com/javase/6/docs/api/java/io/Writer.html).
* All non-abstract methods of [java.util.AbstractList](http://docs.oracle.com/javase/6/docs/api/java/util/AbstractList.html), [java.util.AbstractSet](http://docs.oracle.com/javase/6/docs/api/java/util/AbstractSet.html) and [java.util.AbstractMap](http://docs.oracle.com/javase/6/docs/api/java/util/AbstractMap.html).
* [javax.servlet.http.HttpServlet](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServlet.html), all the doXXX() methods by default sends a HTTP 405 "Method Not Allowed" error to the response. You're free to implement none or any of them.

#### 3.3.3.11 [Visitor](http://en.wikipedia.org/wiki/Visitor_pattern)

Recognizable by two different abstract/interface types which has methods defined which takes each the other abstract/interface type; the one actually calls the method of the other and the other executes the desired strategy on it.

Usage:

One or more related classes have the same method, which calls a method specific for themselves in another class.

Examples:

* [javax.lang.model.element.AnnotationValue](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/element/AnnotationValue.html) and [AnnotationValueVisitor](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/element/AnnotationValueVisitor.html)
* [javax.lang.model.element.Element](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/element/Element.html) and [ElementVisitor](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/element/ElementVisitor.html)
* [javax.lang.model.type.TypeMirror](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/type/TypeMirror.html) and [TypeVisitor](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/type/TypeVisitor.html)

## 3.4 Coding style in Java

We will follow the Google's coding standards for source code in the Java Programming Language. A Java source file is described as being *in Google Style* if and only if it adheres to the rules herein.

Please follow the link below to see the **complete** definition of Google's coding standards:

<https://google-styleguide.googlecode.com/svn/trunk/javaguide.html>

### 3.4.1 Classes

Classes should be small, have single responsibility and high cohesion. They should have only one reason to change. We should also use encapsulation and meaningful names to make it more readable and easy to understand.

Example:

public class Version {

public int getMajorVersionNumber()

public int getMinorVersionNumber()

public int getBuildNumber()

}

Another important rule is to minimize the mutability and favor composition over inheritance.

### 3.4.2 Interfaces

We should program to interface and not via implementation.

Prefer interfaces to abstract classes; abstract classes should primarily be used for objects that are closely related, whereas interfaces are better at providing common functionality for unrelated classes.

Example:

List myList = new ArrayList(); // programming to the List interface

instead of

ArrayList myList = new ArrayList(); // this is bad

### 3.4.3 Enums and Constants

Use Enums instead of int constants.

Store the value of the enum into its instance field and don’t derive it.

Example:

public Enum SportType{

FOOTBALL(1),

TENNIS(2),

BASKETBALL(3)

….

### 3.4.4 Exceptions

Exceptions are to be used only for exceptions conditions and not for ordinary control flow.

A well designed API shouldn’t force its client to use exceptions to control its flow.

Prefer user defined exceptions to generic ones.

Use checked exceptions for conditions that are recoverable and runtime exceptions for programming errors where a precondition is violated.

Avoid unnecessary use of checked exceptions and favor reusing of pre existing exceptions.

Don’t ignore exceptions with empty catch block and remember to add the finally clause.

## 

## 3.5 Useful libraries in Java

### 3.5.1 Json

[Gson](http://code.google.com/p/google-gson/) - Google Gson library.

[Jackson](http://jackson.codehaus.org/) - Jackson is a high-performance JSON processor (parser, generator).

### 

### 3.5.2 Logging

[Apache Log4j](http://logging.apache.org/log4j/1.2/) - Apache Log4j 1.2.

### 3.5.3 Date and Time

[Joda-Time](http://www.joda.org/joda-time/) - Date and time library to replace JDK date handling.

### 3.5.4 Testing

[JUnit](http://junit.org/) - JUnit is a regression testing framework written by Erich Gamma and Kent Beck. It is used by the developer who implements unit tests in Java.

[Mockito](http://www.mockito.org/) - Mock objects library for java.

[EasyMock](https://www.google.pl/search?q=EasyMock) - EasyMock provides an easy way to create Mock Objects for interfaces and classes generating them on the fly.

[Hamcrest Core](https://www.google.pl/search?q=Hamcrest%20Core) - This is the core API of hamcrest matcher framework to be used by third-party framework providers. This includes the a foundation set of matcher implementations for common operations.

[Hamcrest All](https://www.google.pl/search?q=Hamcrest%20All) - A self-contained hamcrest jar containing all of the sub-modules in a single artifact.

[Jbehave](http://jbehave.org/) - JBehave is a framework for [Behaviour-Driven Development](http://en.wikipedia.org/wiki/Behavior_driven_development) (BDD) written in Java.

[Cucumber](https://cucumber.io/) - Cucumber is a framework for [Behaviour-Driven Development](http://en.wikipedia.org/wiki/Behavior_driven_development) (BDD) written initially in Ruby and then ported to Java.

### 3.5.5 Maven

#### 3.5.5.1 Unit and Functional Testing Plugins

[Surefire](http://maven.apache.org/plugins/maven-surefire-plugin/) - Unit tests of a particular application during test stage of a build life cycle can be executed with this plugin. Reports are generated in plain text files (\*.txt) format and XML files (\*.xml).

[Selenium](http://mojo.codehaus.org/selenium-maven-plugin/) - As the name suggests, this plugin supports the use of Selenium with Maven. Automated web-application testing can be done by invoking the Selenium Remote control server through the Selenium Maven Plugin.

[Surefire Report Plugin](http://maven.apache.org/plugins/maven-surefire-report-plugin/) - Using this plugin XML files can be parsed and rendered to DOXIA which enables the creation of a web interface version of the test results. The results are generated in HTML format.

#### 3.5.5.2 Code Quality Plugins

[Findbugs](http://mojo.codehaus.org/findbugs-maven-plugin/) -Findbugs is a plugin for Java programs to tackle bugs. The bug patterns concept is used. Basically the bug pattern is a code idiom. Most often it is an error. Java bytecode is inspected for bug patterns using static analysis.

[Corbertura](http://mojo.codehaus.org/cobertura-maven-plugin/) - As the name suggests, Corbertura features can be brought into the Maven 2 environment by this plugin. Unit testing efforts can be determined by this plugin which helps to understand the part of Java program lacking test coverage.

[Dependency](http://maven.apache.org/plugins/maven-dependency-plugin/) - for module dependency analysis.

[Maven SonarQube plugin](http://mvnrepository.com/artifact/org.codehaus.sonar/sonar-maven3-plugin/5.1) - mvn sonar:sonar to run your analysis, no need to worry about anything else - SonarQube handles that for you.

[Maven Versions Plugin](http://mojo.codehaus.org/versions-maven-plugin/) - scans a project's dependencies and produces a report of those dependencies which have newer versions available. Updates the parent section of the child modules of a project so the version matches the version of the current project.

[Maven Help Plugin](http://maven.apache.org/plugins/maven-help-plugin/) which is mandatory to debug a POM.

#### 3.5.5.3 Core

[Guava: Google Core Libraries for Java](https://www.google.pl/search?q=Guava:%20Google%20Core%20Libraries%20for%20Java) - Guava is a suite of core and expanded libraries that include utility classes, google's collections, io classes, and much much more. Guava has only one code dependency - javax.annotation, per the JSR-305 spec.

#### 3.5.5.4 Commons

[Commons IO](http://commons.apache.org/io/) - The Commons IO library contains utility classes, stream implementations, file filters, file comparators, endian transformation classes, and much more.

[Commons Lang](http://commons.apache.org/lang/) - Commons Lang, a package of Java utility classes for the classes that are in java.lang's hierarchy, or are considered to be so standard as to justify existence in java.lang.

[Apache Commons Codec](http://commons.apache.org/proper/commons-codec/) - The Apache Commons Codec package contains simple encoder and decoders for various formats such as Base64 and Hexadecimal. In addition to these widely used encoders and decoders, the codec package also maintains a collection of phonetic encoding utilities.

[Apache Commons Lang](http://commons.apache.org/proper/commons-lang/) - Apache Commons Lang, a package of Java utility classes for the classes that are in java.lang's hierarchy, or are considered to be so standard as to justify existence in java.lang.

[Apache Commons Configuration](http://commons.apache.org/configuration/) - Tools to assist in the reading of configuration/preferences files in various formats.

[Apache Commons BeanUtils](http://commons.apache.org/proper/commons-beanutils/) - Apache Commons BeanUtils provides an easy-to-use but flexible wrapper around reflection and introspection.

[Protocol Buffer Java API](http://code.google.com/p/protobuf) - Protocol Buffers are a way of encoding structured data in an efficient yet extensible format.

[Apache Commons FileUpload](http://commons.apache.org/proper/commons-fileupload/) - The Apache Commons FileUpload component provides a simple yet flexible means of adding support for multipart file upload functionality to servlets and web applications.

#### 3.5.5.5 XML

[Apache Commons Digester For XML](http://commons.apache.org/proper/commons-digester/) - Digester is a XML to Java mapping utility that has been heavily used in XML communication using java. Reading XML config files and data can be really easy the digester.

[JSoup](http://jsoup.org/) - jSoup is a library that is used for parsing HTML file in Java. It can be used for web application unit testing and web scraping. Its “jQuery-like” and “regex” selector syntax is very easy to use for data extraction. jSoup is quite fast in retrieving the raw page content.

[JDOM](http://www.jdom.org) - JDOM is a popular DOM parser library that is widely used in many java applications for parsing XML.

[DOM4J](http://dom4j.sourceforge.net/) - Its a simple open source library for manipulation of XML, XSLT and XPath.

[XERCES](http://xerces.apache.org/xerces-j/) - Yet another popular XML parser.

[XStream](http://xstream.codehaus.org) - XStream is a simple library to serialize objects to XML and JSON.

#### 3.5.5.6 AOP

[Spring-AOP](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html) - proxy-based AOP. It needs the spring framework.

[AspectJ](https://eclipse.org/aspectj/) - a seamless aspect-oriented extension to the Java programming language.

#### 3.5.5.7 Monitoring

[Metrics](https://dropwizard.github.io/metrics/3.1.0/) - Metrics provides a powerful toolkit of ways to measure the behavior of critical components in your production environment. Metrics has modules for common libraries like Jetty, Logback, Log4j, Apache HttpClient, Ehcache, JDBI, Jersey.

[JAMon](http://jamonapi.sourceforge.net/) - JAMon allows developers to track their applications performance and behavior using predefined modules without code changes.

#### 3.5.5.8 Cron and Scheduling

[Quartz](http://www.quartz-scheduler.org/) - Quartz is a richly featured, open source job scheduling library that can be integrated within virtually any Java application -

Spring built in scheduler - If all you need to is to execute methods on a bean every X seconds, or on a cron schedule, and we use Spring, then we can use the Spring @Scheduled annotation, but for more complex scheduling tasks Quartz offers more.

# 4 Javascript specific issues

## 4.1 Modules in Javascript

While writing large JavaScript applications, one of the simplest things one can do is divide the code base into several files. The problem comes when we have to load/unload all modules and their dependencies on demand.

[ADD Webpack Here] [RequireJS](http://requirejs.org/) is a JavaScript library that helps in lazily loading JavaScript dependencies. Modules are just JavaScript files with some RequireJS syntactic code in them. RequireJS implements [Asynynchronous Modules](http://wiki.commonjs.org/wiki/Modules/AsynchronousDefinition) specified by [CommonJS](http://requirejs.org/docs/commonjs.html). RequireJS offers simple APIs to create and refer to modules.

The dependency injection system built into AngularJS deals with the objects needed in a component; while dependency management in RequireJS deals with the modules or, JavaScript files.

When RequireJS attempts to load a module, it checks for all dependent modules and loads them first.

The following code block defines an Angular module:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  1 | define(['app/config',  'app/ideasDataSvc',  'app/ideasHomeController',  'app/ideaDetailsController'],    function(config, ideasDataSvc, ideasHomeController, ideaDetailsController){  var app = angular.module('ideasApp', ['ngRoute','ngResource','ngGrid']);  app.config(config);  app.factory('ideasDataSvc',ideasDataSvc);  app.controller('ideasHomeController', ideasHomeController);  app.controller('ideaDetailsController',ideaDetailsController);  }); |

On a component side we need to write our Javascript code in a modular way using the Javascript Module Pattern.

Each library has to have single responsibility and public APIs.

### 4.1.1 Modules repository

[ADD Npm Webpack here] In order to store and share the project Javascript libraries across the project development teams we will have a private [Bower](http://bower.io/) registry that is able to store private Javascript libraries and also to proxy the public bower registry containing all the public Javascript libraries.

### 4.1.2 Module release

In order to achieve automatic push tagged Bower package releases to endpoint we will combine Bower with [Grunt](http://gruntjs.com).

The Grunt Javascript Task Runner script will be able to create different build for different environment depending on the profile specified into the grunt script.

The grunt script will build the library with all its dependencies, minify it, publish it to the private repository, create a git tag and update the version on the dependencies list file.

## 4.2 Testing in Javascript

### 4.2.1 BDD tests

For behavior-driven development in Javascript we will use existing popular frameworks such as [Jasmine](http://jasmine.github.io/). Jasmine is a behavior-driven development framework for testing JavaScript code.

Ex.

describe("A suite", function() {  
 it("contains spec with an expectation", function() {  
 expect(true).toBe(true);  
 });  
});

### 4.2.2 Integration tests

For integration and end to end tests we will use existing Javascript framework such as Protractor.

Protractor is an end-to-end test framework for AngularJS applications, and uses Selenium Web Driver to drive tests. It runs tests against your application running in a real browser, interacting with it as a user would.

describe('angularjs homepage todo list', function() {  
 it('should add a todo', function() {  
 browser.get('https://angularjs.org');  
  
 element(by.model('todoList.todoText')).sendKeys('write first protractor test');  
 element(by.css('[value="add"]')).click();  
  
 var todoList = element.all(by.repeater('todo in todoList.todos'));  
 expect(todoList.count()).toEqual(3);  
 expect(todoList.get(2).getText()).toEqual('write first protractor test');  
  
 // You wrote your first test, cross it off the list  
 todoList.get(2).element(by.css('input')).click();  
 var completedAmount = element.all(by.css('.done-true'));  
 expect(completedAmount.count()).toEqual(2);  
 });  
});

### 4.2.3 Unit tests

For unit tests we will use existing Javascript test-runner tools such as Karma. Karma is a JavaScript test-runner built with Node.js, and meant for unit testing.

We will use karma and jasmine for our unit tests

// tests start here  
 it('should have variable text = "Hello World!"', function(){  
 expect(scope.text).toBe('Hello World!');  
 });

## 4.3 Design patterns in Javascript

As mentioned in the Java Design Pattern section, before solving the problem with our solution, we should think if we can apply and reuse any of the existing bullet proof design patterns.

Here some good links for the Javascript patterns:

<http://www.dofactory.com/javascript/design-patterns>

<http://shichuan.github.io/javascript-patterns/>

## 4.4 Code style in Javascript

*Code* conventions can help in reducing the brittleness of programs.

We will follow the popular code convention of Airbnb.

Here the link of the Airbnb Javascript code style:

<https://github.com/airbnb/javascript/tree/master/es5>

### 

## 4.5 Documentation in Javascript

We will write APIs and code documentation and we will use a Javascript documentation generator such as [JSDoc](http://usejsdoc.org/) or [YUIDoc](http://yui.github.io/yuidoc/).

## 

## 4.6 Useful libraries in Javascript

[echarts](http://ecomfe.github.io/echarts/index-en.html) - open source tool for data visualization.

[epoch](https://fastly.github.io/epoch/) - real time charting library.

[d3](http://d3js.org/) - popular data visualization tool.

[Polyfill libraries](https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills) - cross browser polyfill libraries.

[lodash](https://lodash.com/) - utility library delivering consistency, modularity, performance, & extras.

[numeral](http://numeraljs.com/) - library for formatting and manipulating numbers.

[date](https://github.com/datejs/Datejs) - library for parsing, formatting and processing dates.

[pretty date](https://www.npmjs.com/package/pretty-date) - format old JavaScript dates in a “pretty” way. ex : 2 hours ago , 3 minutes ago.

[xregexp](http://xregexp.com/) - library that provides augmented and extensible regular expressions.

[jQuery UI](http://ui.jquery.com/themeroller/) - The jQuery UI allows us to design custom user interfaces for Web applications using the [jQuery](http://jquery.com/) library.

[Firebug](http://getfirebug.com/) - Firebug is an extremely popular and well-regarded front-end debugging tool. It has all the features you’d expect from a JavaScript debugging tool, such as the ability to set breakpoints in your code so that you can step through your script.

[AJAX Libraries API](http://code.google.com/apis/ajaxlibs/) - Google’s AJAX Libraries API allows you to serve popular JavaScript libraries using its CDN, which reduces the server load on your website.

[JS Minifier](http://fmarcia.info/jsmin/test.html) - JS Minifier is a Web-based tool for shrinking your JavaScript code to make it as lightweight as possible.

[Bootstrap](http://getbootstrap.com/) - Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.

[Less](http://lesscss.org/) - CSS pre-processor.

## 4.7 Useful resources for Javascript

* The best place to find Angular modules is [ngmodules.org](http://ngmodules.org/)
* Javascrpsource - <http://www.javascriptsource.com/>
* SuperHeroJS - <http://superherojs.com/>.

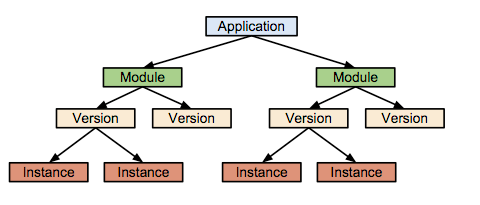
# 5 Google app engine specific issues

## 

## 5.1 Modules in GAE

At the highest level, an App Engine application is made up of one or more *modules*.

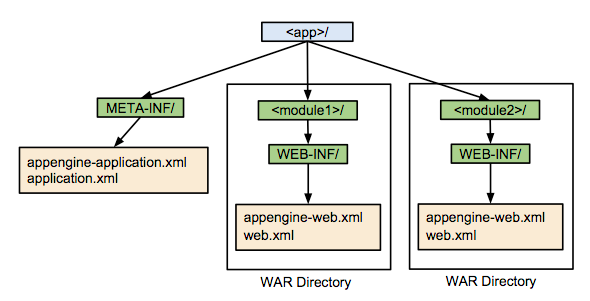
When we deploy a module, we always deploy a specific version of the module. For this reason, whenever we speak of a module, it usually means a version of a module.



Stateful services (such as Memcache, Datastore, and Task Queues) are shared by all the modules in an application.

An App Engine application that uses modules is organized as an unpacked Java Enterprise Archive (EAR) directory structure. The top-level EAR directory contains a single META-INF subdirectory, and a separate directory for each module in the app. These module directories are organized as unpacked Java Web Application Archives (WAR). Each WAR directory usually has the same name as the module it defines, but this is not required.

We will extend the default GAE module archetype with our customized version of it.



Below the link to the GAE module documentation:

<https://cloud.google.com/appengine/docs/java/modules/>

## 5.2 Communication between GAE modules

The project modules will communicate internally using the message bus (publisher/subscriber pattern) and private RESTful APIs over HTTPS.

They will communicate with the rest of the world using a middleware layer via [endpoints](https://cloud.google.com/appengine/docs/java/endpoints/).

## 5.3 Testing in GAE

Here the link where you can see how to write unit tests for GAE applications:

<https://cloud.google.com/appengine/docs/java/endpoints/>

We can then use Maven for our GAE integration tests.

Ex.

<plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-failsafe-plugin</artifactId>  
 <version>${maven.failsafe.version}</version>  
 <executions>  
 <execution>  
 <goals>  
 <goal>integration-test</goal>  
 <goal>verify</goal>  
 </goals>  
 </execution>  
 </executions>  
</plugin>

<plugin>  
 <groupId>com.google.appengine</groupId>  
 <artifactId>appengine-maven-plugin</artifactId>  
 <version>${appengine.target.version}</version>  
 <executions>  
 <execution>  
 <id>start-gae</id>  
 <phase>pre-integration-test</phase>  
 <goals>  
 <goal>devserver\_start</goal>  
 </goals>  
 </execution>  
 <execution>  
 <id>stop-gae</id>  
 <phase>post-integration-test</phase>  
 <goals>  
 <goal>devserver\_stop</goal>  
 </goals>  
 </execution>  
 </executions>  
</plugin>

## 5.4 Deploy a module to the cloud

We will use the IntelliJ GAE plugin in order to debug, run locally and deploy the module to the cloud.

<https://www.jetbrains.com/idea/features/google_app_engine.html>

## 5.5 Secure an endpoint using Spring security

For the internal REST APIs we won’t need to use Spring security because the security will be managed by the Google App Engine PaaS itself (see <security-constraint> with a <role-name>admin</role-name> in the [modules documentation](https://cloud.google.com/appengine/docs/java/modules/#Java_Communication_between_modules)).

In order to protect the public endpoints we will use standard Spring Security using annotations.

## 5.6 Useful libraries for GAE

[Objectify](https://github.com/objectify/objectify) - a library that simplify the Cloud Datastore Low-Level API.

# 6 Android specific issues

## 6.1 Code style in Android

Please see the [Code style](#h.wqnqui2yedjs) in the Java section of this document.

For the UI please have a look at the link below:

<https://developer.android.com/guide/topics/ui/index.html>

## 6.2 Android testing

As part of the definition of done, the story/feature must include Unit tests and UI tests. To align with our web UI tool Selenium, UI tests will be written in <http://selendroid.io/> for the Android application.

The workflow for testing will be:

* test on local emulator
* test with an actual G4S device

for each story/feature.

The CI environment will use SauceLabs hosted service to test against other general devices to ensure compliance with Android devices and detect any potential issues hidden by the use of the G4S hardware.

For Android best practices in testing see: <http://developer.android.com/training/testing.html>

We will use the Java guidelines with the following exceptions:

<https://source.android.com/source/code-style.html>

## 6.3 Useful libraries in Android

[Butterknife](http://jakewharton.github.io/butterknife/) - Annotate fields with @Bind and a view ID for Butter Knife to find and automatically cast the corresponding view in your layout.

[Lambok](https://projectlombok.org/setup/android.html) / [hrisey](https://github.com/mg6maciej/hrisey) - boilerplate code suppressor tool for Android platform.

[Otto](http://square.github.io/otto/) - an event bus designed to decouple different parts of your application while still allowing them to communicate efficiently.

[Picasso](http://square.github.io/picasso/) - A powerful image downloading and caching library for Android.

[okhttp](http://square.github.io/okhttp/) - An HTTP & SPDY client for Android and Java applications.

[Retrofit](http://square.github.io/retrofit/) - A type-safe REST client for Android and Java.

[Parceler](http://parceler.org) - Android Parcelables made easy through code generation.

[Crouton](https://github.com/keyboardsurfer/Crouton) - Context sensitive notifications for Android.

[Icepick](https://github.com/frankiesardo/icepick) - Android Instance State made easy.

[Leakcanary](https://github.com/square/leakcanary) - A memory leak detection library for Android and Java.

# 7 Source control VCS

For this project we are going to use the popular GIT repository.

The main difference between GIT and SVN is that GIT is decentralized.

With git you have the source control repository on your laptop and when you are ready you can push the changes to the server repository.

Here a good tutorial to become a GIT Guru:

<https://www.atlassian.com/git/tutorials>

## 7.1 GIT Flow

In the past 5 years one of the most popular git flow has been [git-flow](http://nvie.com/posts/a-successful-git-branching-model/).

The problem of git-flow is that it is a step backward from Continuous Delivery(CD) which is all about ensuring that every new incremental bit of work that passes various testing, validation, verification, and other processes is immediately deployable. By focusing so heavily on building well-defined releases only after work has been delivered to the proper branches, it misses the point of CD altogether.

This is why github doesn’t use it.

We will use our own git flow based on the github-flow principles:

|  |  |
| --- | --- |
| git_workflow_github_flow.jpg | * Anything in the master branch is deployable * To work on something new, create a Jira Story/Bug id named branch off of master (ie: PALL-123) * Commit to that branch locally and regularly push your work to the same named branch on the server * When you need feedback or help, or you think the branch is ready for merging, open a pull request * After someone else has reviewed and signed off on the feature, you can merge it into master * Once it is merged and pushed to ‘master’ on the origin, you can and should deploy immediately |

## 7.2 Tags strategy

When we release a new version of any module we will create the TAG and store in into the GIT repository. Tagging will be automated and it will be triggered at the end phase of the module release process.

## 7.3 Commit comments

Each commit comments must start with the user story or bug ID and be descriptive. We should be looking to create a hook in git so that whenever you are committing code it will ask for a story id

Ex. git commit -m ‘[PALL-123] added create account api implementation’

## 7.4 Commit your changes

Commit your changes to your local repository as often as possible. Typically 3 times a day!

## 7.5 Push your changes

Push your changes to the server repository ( branch or master ) several times a day. If you push your changes in a branch where the CI server is listening to, you will trigger the CI build.

## 7.6 Pull and Merge changes

Keep your code up to date with changes from other people.

## 7.7 Ignore files

Before commiting be sure that you have a .gitignore file in the root of the project.

The gitignore ensure that you don’t commit files not wanted files into the git repository.

Here a list of the most popular git ignores (You can combine them in one).

<https://github.com/github/gitignore>

# 8 Continuous integration

## 8.1 Trigger a build

In order to trigger a build there are two implementations:

1. Poll the repository to see if changes occurred. This might "skip" a commit if two commits come in within the same polling interval.
2. Configure the repository to have a post-commit hook which notifies the CI (Jenkins) that a build needs to start.

We adopt the second options that means that every time the developer push his changes to the main remote repository the CI will trigger the build.

# 9 Continuous inspection

## 9.1 Trigger the code inspection

The [SonarSource Plugins](http://docs.sonarqube.org/display/PLUG/SonarSource+Plugins?src=breadcrumbs-parent) allows us to trigger SonarQube analysis from the CI server (Jenkins) using either a:

* Build step to trigger the analysis with the SonarQube Runner
* Post-build action to trigger the analysis with Maven

In order to achieve that we need a CI Jenkins server up and running with the Sonar Jenkins Plugin installed and our [Sonar server configured in the general settings](http://docs.sonarqube.org/display/PLUG/Configuring+SonarQube+Jenkins+Plugin#ConfiguringSonarQubeJenkinsPlugin-AddingSonarQubeServer). In maven we shall just [add a post build action](http://docs.sonarqube.org/display/PLUG/Triggering+SonarQube+on+Jenkins+Job#TriggeringSonarQubeonJenkinsJob-TriggeringaProjectAnalysiswithMaven) to our existing job.

# 10 Dependency management

## 10.1 Overview

Dependency management is made up of two pieces. Firstly, it needs to know about the things that the project needs to build or run, in order to find them. We call these incoming files the *dependencies* of the project. Secondly, it needs to build and upload the things that the project produces. We call these outgoing files the *publications* of the project.

As mentioned in the section above we will use Maven to manage the Java dependencies and Grunt + Bower for the Javascript ones.

## 10.2 Versioning scheme

Our versioning scheme uses the following standards:

* MajorVersion
* MinorVersion
* IncrementalVersion
* BuildNumber
* Qualifier

For example:

* MajorVersion: 1.2.1
* MinorVersion: 2.0
* IncrementalVersion: 1.2-SNAPSHOT
* BuildNumber: 1.4.2-12
* Qualifier: 1.2-beta-2

## 10.3 Maven

The basic syntax most commonly used for defining dependencies in Maven is adding a tuple of the group-id, artifact-id and requested version to the dependencies section of the build script. The build tool then tries to resolve these dependencies, by searching for them in its local and remote-defined repositories.

<dependency>  
 <groupId>com.google.code.gson</groupId>  
 <artifactId>gson</artifactId>  
 <version>2.3.1</version>  
</dependency>

Maven enables you to specify a range of versions that are acceptable to use as dependencies. shows a range of version specifications:

|  |  |
| --- | --- |
| **Range** | **Meaning** |
| (,1.0] | x <= 1.0 |
| 1.0 | It generally means 1.0 or a later version, if 1.0 is not available. Various Maven plugins may interpret this differently, so it is safer to use one of the other, more specific options. |
| [1.0] | Exactly 1.0 |
| [1.2,1.3] | 1.2 <= x <= 1.3 |
| [1.0,2.0) | 1.0 <= x < 2.0 |
| [1.5,) | x >= 1.5 |
| (,1.0],[1.2,) | x <= 1.0 or x >= 1.2.  Multiple sets are separated by a comma. |
| (,1.1),(1.1,) | This excludes 1.1 if it is known not to work in combination with the library. |

## 10.4 Bower/Grunt

The basic syntax for used for Bower/Grunt dependencies is adding them to a JSON file as the following example:

{

"name": “app-name,

"version": “0.0.1”,

"dependencies": {

"angular": "^1.3.0",

"angular-bootstrap": "0.12.0",

"json3": "^3.3.0",

"es5-shim": "^4.0.0",

"bootstrap": "^3.2.0",

"font-awesome": "^4.2.0",

"angular-animate": "^1.3.0",

"angular-cookies": "^1.3.0",

"angular-resource": "^1.3.0",

"angular-route": "^1.3.0",

"angular-sanitize": "^1.3.0",

"angular-touch": "^1.3.0",

"angular-bowser": "0.0.1",

"angular-ui-router": "^0.2.13",

"ngInfiniteScroll": "1.2.0",

"angular-local-storage": "^0.1.5"

},

"devDependencies": {

"angular-mocks": "~1.3.0",

"angular-scenario": "~1.3.0"

},

"appPath": "app"

}

# 11 Release guidelines

We already covered the release process for the modules into the Javascript/Java modules section and dependency management.

Each project will follow the same versioning rules.

We will work in a continuous delivery environment that means that we need to be able to release at any time on any environment.

Our goal is to deliver an increment of the product with testable features at the end of each Sprint.

We will automate our deployment so that we can deploy new features when they are ready.

The deployment will have different stages:

* commit/push stage - the developer push his code into the version of control;
* continuous integration - all the tests will be run and the code will be built by the continuous integration server;
* continuous inspection - the code will be inspected by the continuous inspection server;
* shared sandbox environment - the code will be sent to a shared sandbox environment ( testing ) where the application is deployed and run;
* specialist testing environments - we may need to perform additional testing for specialist requirement like performance testing, penetration testing and accessibility testing;
* production environment - when all the mentioned above phases have passed we can deploy it to the production environment in the same way as we did for the other environments. This means that we are not releasing the code for the first time, but only performing a tasks that has been validated at each stage in the previous phases.

# 12 Performance

When we develop we need to take care of the performance all the time.

This is valid for Mobile, BackEnd and Frontend applications.

Below some useful links for optimization:

Google App Engine

<https://cloud.google.com/appengine/articles/spring_optimization>

Javascript

<https://developers.google.com/speed/articles/optimizing-javascript>

<http://www.alexkras.com/11-tips-to-improve-angularjs-performance/>

<https://www.airpair.com/angularjs/posts/angularjs-performance-large-applications>

Java

<http://www.oracle.com/technetwork/java/5-136747.html>

<http://blog.jooq.org/2015/02/05/top-10-easy-performance-optimisations-in-java/>

# 13 Confluence

Confluence is team collaboration software. Written in Java and mainly used in corporate environments.

Confluence will be used in conjunction with JIRA, our requirements analysis tool which will hold user stories as well as testing stories. We will use Confluence to hold the documentation and project artifacts for team collaboration.

Confluence is a wiki type system, composed of pages and links, teams can quickly share content very easily and get comments and feedback.

Confluence will be used to capture knowledge for the duration of the project.

# 14 Security

By default, all communications will take place over [HTTPS](https://en.wikipedia.org/wiki/HTTPS)[[14]](#footnote-14), as per Google standard.

All RPCs, either external or internal, will take place over HTTPS. [App Engine supports HTTPS natively](https://cloud.google.com/appengine/docs/java/config/webxml#Secure_URLs).

1. "Cohesion (computer science) - Wikipedia, the free ..." 2011. 1 Jul. 2015 <<https://en.wikipedia.org/wiki/Cohesion_(computer_science)>> [↑](#footnote-ref-1)
2. "functional cohesion concept from the Object Oriented ..." 2002. 1 Jul. 2015 <<http://www.site.uottawa.ca:4321/oose/functionalcohesion.html>> [↑](#footnote-ref-2)
3. "Loose coupling - Wikipedia, the free encyclopedia." 2011. 1 Jul. 2015 <<https://en.wikipedia.org/wiki/Loose_coupling>> [↑](#footnote-ref-3)
4. "Task Queue Java API Overview - Google Cloud Platform." 2014. 7 Jul. 2015 <<https://cloud.google.com/appengine/docs/java/taskqueue/>> [↑](#footnote-ref-4)
5. "What is an ESB? | MuleSoft." 2015. 7 Jul. 2015 <<https://www.mulesoft.com/resources/esb/what-esb>> [↑](#footnote-ref-5)
6. "Technical debt - Wikipedia, the free encyclopedia." 2011. 1 Jul. 2015 <<https://en.wikipedia.org/wiki/Technical_debt>> [↑](#footnote-ref-6)
7. "App Engine Modules in Java - Java — Google Cloud Platform." 2014. 7 Jul. 2015 <<https://cloud.google.com/appengine/docs/java/modules/>> [↑](#footnote-ref-7)
8. "Maven – Introduction to Archetypes." 2010. 1 Jul. 2015 <<https://maven.apache.org/guides/introduction/introduction-to-archetypes.html>> [↑](#footnote-ref-8)
9. "Nexus Repository Managers - Product Overview - Sonatype ..." 2015. 1 Jul. 2015 <<http://www.sonatype.com/nexus/product-overview>> [↑](#footnote-ref-9)
10. "Bower." 2013. 1 Jul. 2015 <<http://bower.io/>> [↑](#footnote-ref-10)
11. "Encapsulation (computer programming) - Wikipedia, the free ..." 2015. 2 Jul. 2015 <<https://en.wikipedia.org/wiki/Encapsulation_(computer_programming)>> [↑](#footnote-ref-11)
12. "Builder pattern - Wikipedia, the free encyclopedia." 2011. 2 Jul. 2015 <<https://en.wikipedia.org/wiki/Builder_pattern>> [↑](#footnote-ref-12)
13. "Markdown - Wikipedia, the free encyclopedia." 2011. 2 Jul. 2015 <<https://en.wikipedia.org/wiki/Markdown>> [↑](#footnote-ref-13)
14. "HTTPS - Wikipedia, the free encyclopedia." 2011. 8 Jul. 2015 <<https://en.wikipedia.org/wiki/HTTPS>> [↑](#footnote-ref-14)