Thomson Reuters – Appirio Development Methodology

Draft - Version 1.1

February 16, 2010

Contents

[1 Communications 3](#_Toc253035108)

[2 Definition 3](#_Toc253035109)

[3 Design 3](#_Toc253035110)

[4 Development 3](#_Toc253035111)

[5 Coding Standards 4](#_Toc253035112)

[5.1.1 Coding Standards 4](#_Toc253035113)

[6 Development Roles and Responsibilities 9](#_Toc253035114)

[7 Workflow 9](#_Toc253035115)

[8 Review and Sign-off Process (includes Handoff Standards process) 10](#_Toc253035116)

# Communications

* Weekly status meetings will be held for all projects. Hemant will be invited to these status reviews.
* Project repositories will be maintained on Google sites.
* The Agile methodology is being adhered to. As requirements are identified, a determination as to what Sprint they will fall into will be made. Team Track will be used as the tool to manage requirements. Thomson Reuters will provide initial PBI’s and drive analysis with Appirio.
* Requirements will be gathered in planned sessions, outside of weekly status calls. Thomson Reuters and Appirio will work together to agree on PBI’s that fall into each Sprint.

# Definition

* Hemant will be included in requirements review sessions. Hemant will be copied on all requirements meetings and documentation throughout the projects. Access will be provided to Hemant for all Google project sites so that his team can access project collateral and project plans. Hemant will make a determination as to whether he needs to attend meetings.

# Design

* Dinesh will be included in design review sessions. Dinesh will be copied on all design meetings and documentation throughout the projects. Dinesh will make a determination as to whether he needs to attend meetings.

# Development

* Requirements will be evaluated to determine what development work can be supported by Hemant’s team.
* Appirio will provide the initial analysis on requirements, then work with the Thomson Reuters team to agree on delegation of scope. Once requirements for a Sprint are locked down, Appirio Development will work with Thomson Reuters Development to delegate coding tasks, based upon skillset.
* Thomson Reuters Development team will be accountable for adhering to coding standards defined between Appirio and Thomson Reuters Development.
* Strong gatekeepers will need to be in place to support this blended model. This will increase effort, as we will need to define coding review standards/meetings for the Thomson Reuters team, working with Appirio. Gatekeepers will be a combination of Appirio and Thomson Reuters leads.
* \*\*Appirio will provide development leads onsite for two weeks to build:
  + Strong communication model between the teams
  + Outline coding standards between the teams
  + Determine necessary gatekeepers and reviews

\*\* Appirio still working out the logistics with Development as agreements and specific dates have not been locked down

# Coding Standards

Salesforce.com recommends following Java standards for naming, that is, classes start with a capital letter, methods start with a lowercase verb, and variable names should be meaningful.

It is not legal to define a class and interface with the same name in the same class. It is also not legal for an inner class to have the same name as its outer class. However, methods and variables have their own namespaces within the class so these three types of names do not clash with each other. In particular it is legal for a variable, method, and a class within a class to have the same name.

### Coding Standards

|  |
| --- |
| **Why have coding standards?**   * Hardly any software is maintained for its whole life by the original author. * Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly. * If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create. * Standards make it easier to share work, maintain code, and increase the overall quality of a project.   **General Coding Standards:**   * We write code primarily for people, not machines.  It's easy to write code a machine can understand - the professional standard is to write code a person can understand. * Writing code is not the time to display how clever you are. Always err on the side of clarity. * Make sure your code will be readable and understandable 12 months from now, by a person who has never seen it. * Even when "prototyping," do not take shortcuts. Prototypes often morph into production code. Take the time with every single line of code you write to make it maintainable and readable. * Always try to use the latest version of the API available GA. This ensures optimum performance and supports latest methods. * Proper commenting should be used but do not over comment. Comment at the top of each class / trigger / page, and before complicated methods.   + The class comment should contain our company name, brief class purpose, and a simple change history, showing author and gist of the change.  Example format:     - //     - // (c) 2009 Appirio, Inc.     - //     - // Generate a new lead from a sales-related case     - //     - // 8/5/2009     Jane Doe       Original     - // 8/9/2009     John Smith     Added logic to handle contacts with no first name     - //   + Method comments should include argument data types and what the method does.   + Code blocks should have at least a one liner describing a function that they provide.   + Code that implements a business rule should be called out with a comment.   + Hopefully, one day Apex code will have Java like documentation and your comments will be included in Apex generated docs. So good commenting is key for making usable docs.) * The application does what the business wants when things go wrong. e.g. if the apex update limit is hit what should the program do. This has to be accepted by the business.   **Technology Specific Standards:**  **Apex:**   * No DML inside of getter methods - If the page refreshes all of the getter methods get re-executed. * All variable declaration is to be done outside of loops - APEX garbage collection does not occur until after all of the APEX code is executed. * Use Constants wherever possible in the code for error messages, etc so that they can be internationalized if necessary. * Naming convention for all Apex Classes, Apex Controllers, and Apex Pages should include the name of the Application. * Naming convention for all Test code should include the Application and the function and the word Test. * There should not be any DML statements inside of a for loop. * The Apex code must have 92% or above code coverage using test methods. * All the execution logic should be in Apex classes and the trigger should invoke the classes as required. * Apex Code must provide proper exception handling. * Test methods should verify actual correct execution via liberal use of System.assert().  Avoid rigging test methods to always return success solely to meet Salesforce.com code coverage requirements. * Although Apex is case-insensitive, always follow capitalization standards according to Java. * Errors should, when possible, be attached to the field that caused them.   + Correct:     - Trigger.new[i].someField.addError("You cannot set someField to a value less than 0.");   + Only do this if the error isn't specific to a particular field:     - Trigger.new[i].addError("Something bad happened."); * When testing for governor limits, use the Limit class - do not hard-code governor limits. * Avoid assuming a single SObject wil be returned from a SOQL query, since being wrong will cause a runtime exception.   + Correct:     - List<Account> accountList = [select id from account where someField = '123'];   + Wrong:     - Account acct = [select id from account where someField = '123']; * Another way to handle it is using a for loop to immediately iterate through your result. if there are no results, it won't error and won't run the code in the loop. you could even throw in  a "break;" in the for loop if you were worried about too many responses.   + for(Contact c :[Select Id, Name from Contact where Email = :currentUser.Email and AccountId = :helpDeskAccount.Id LIMIT 1]){     //do stuff  }   **Apex Triggers:**   * No complex logic inside of triggers. * Naming convention for all Apex Triggers should include the name of the Object and Trigger. * There should only be one trigger for each object. * The triggers must support bulk inserts and updates on the object the trigger is coded. * Proper care and testing must be taken to ensure that any updates made by Trigger have expected behavior on the triggered workflow rules, validation Rules, assignment rules etc. * Keep in mind the order of execution when coding trigger. This is important as this may impact some other behavior already at the object level against which you are coding.   + Triggers and Order of Execution When a record is saved with an insert, update, or upsert statement, the following events occur in order:   + The original record is loaded from the database (or initialized for an insert statement)   + The new record field values are loaded from the request and overwrite the old values   + All before triggers execute   + System validation occurs, such as verifying that all required fields have a non-null value, and running any user-defined validation rules   + The record is saved to the database, but not yet committed   + The record is reloaded as required for Apex and/or workflow rules   + All after triggers execute   + Assignment rules execute   + Auto-response rules execute   + Workflow rules execute   + Escalation rules execute   + All DML operations are committed to the database   + Post-commit logic executes, such as sending email   **Visualforce:**   * Picklists should be defined inside of the Controller instead of hardcoded inside of the page.   **Javascript:**  **S-Controls (Deprecated):**   * Top-level scripts should include the latest version of the AJAX Toolkit API. * Top-level links should include a reference to the Salesforce.com CSS. * Avoid use of deprecated APIs, e.g. DynaBean instead of sforce.SObject. * Use Salesforce.com-style buttons instead of default button styles:   + <input type="button" class="btn"> * Make sure to use queryMore() or QueryResultIterator() syntax when retrieving records, if there is any chance of retrieving more than 2,000. |

# Development Roles and Responsibilities



# Workflow



# Review and Sign-off Process (includes Handoff Standards process)

# Specification Review for APEX Coding

<https://spreadsheets.google.com/viewform?formkey=dGxLczJ3SkdYXzgxZ2RDeFFSV3FTSVE6MA>

Read through the specification first, then review.  
Go through the project request first and see if it makes sense overall. Requests can be very confusing when they are read out of context. Also make sure you read all references made in the request. Once you understand it from a high level you can then dive into the details.  
  
Could you implement the request as is?  
Is it possible to understand the request without requiring much additional information? Is it easy to understand how this request fits into the overall application being developed or can you understand how it would be used?  
  
Terminology  
Is all the terminology well defined? An implementer should understand all the terms by reading the request and supporting material. Additional information should not be required.

\* Required

Top of Form

Are the requirements clear and concise. It is easy to understand what the requestor wishes to receive. \*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |  |
| Unsatisfatory |  |  |  |  | Exceeded expectations |

Was the language easy to understand and to the point. \* If the English, the specification writer was unclear or not concise, please detail what areas of the specification could have been simplified.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |  |
| Unsatisfatory |  |  |  |  | Exceeded expectations |

Functional requirements are clear, concise and to the point \*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |  |
| Unsatisfatory |  |  |  |  | Exceeded expectations |

Are all inputs and outputs well defined? \* Are protocols and formats defined.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |  |
| Unsatisfatory |  |  |  |  | Exceeded expectations |

A concise and clear example of how the project is going to be used is included. \* Example should clarify how the request will be utilized.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |  |
| Unsatisfatory |  |  |  |  | Exceeded expectations |

## VisualForce

Optional if VF is required

Is a CSS defined and the graphical requirements detailed? \*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |  |
| Unsatisfatory |  |  |  |  | Exceeded expectations |

Are the appropriate stylesheets attached \*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |  |
| Unsatisfatory |  |  |  |  | Exceeded expectations |

Did the submitter attach wireframes? If not were they able to accurately describe the layout of the page? \*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |  |
| Unsatisfatory |  |  |  |  | Exceeded expectations |





Bottom of Form