ADS 18f Prototype

Drug Interaction Query

(Drug IQ)

Geocent Program Management Plan

Job 4QTFHS150004

4QTFHS150004

Revision - Dr

June 18, 2015



**Distribution Statement: This document is for internal planning purposes only.**

**Document Authorization**

|  |  |
| --- | --- |
| **Review/Approval Title** | **Name** |
| **Originator:**  Technical Architect | Jared Ladner |
| **Approved By**:  Program Manager |  |
| **Approved By**:  Project Lead |  |
| **Approved By**:  Senior Management |  |
| **Approved By**:  Contracts |  |
| **Approved By**:  Quality Assurance |  |

Electronic signatures were provided on <DATE>

**Document Revision History**

Revison history for the PMP is tracked in GitHub at the following link:

<https://github.com/Geocent/18f-prototype/tree/integration/docs/scrum>

Each iteration of the document is stored within the Sprint folders.

**Table of Contents**

1 Project Overview 6

1.1 Project Scope and Objectives (Project Vision) 6

1.1.1 Customer 6

1.1.2 Project Vision 6

1.1.3 Schedule Summary 6

1.1.4 Deliverables 6

2 References 8

2.1 Source Documentation 8

2.2 Project Working Documents 8

3 Organization 9

3.1 External Interfaces: Who do you interface with outside of Geocent for this project? 9

3.2 Internal Structure: Show a model of the Geocent program management structure for this project 9

3.3 Roles and Responsibilities: List the roles, responsibilities and individual within Geocent for your project. 11

4 Managerial Plans 12

4.1 Start-Up Plan 12

4.1.1 Establish Resources 12

4.2 Execution Plan 14

4.2.1 Manage Requirements 14

4.2.2 Manage Communications 15

4.2.3 Manage Quality 16

4.2.4 Manage Measurements (Metrics) 16

4.2.5 Manage Issues 16

4.2.6 Manage Contract Changes 17

4.3 Work Plan 18

4.4 Closeout Plan 18

**Table of Figures**

[Figure 1. Program Organization Chart 9](#_Toc158536968)

# Project Overview

This document establishes the Program Management Plan for the Drug Interaction Query (Drug IQ) project and serves as the execution plan Geocent will use to manage the project.

## Project Scope and Objectives (Project Vision)

### Customer

The customer for this project is GSA 18f. Drug IQ will be designed to support customers (non-medical professionals) and medical professionals.

### Project Vision

The Project Vision is defined at the following link:

<https://github.com/Geocent/18f-prototype/tree/integration>

#### Scope Definition/High-level needs

The scope of the project and project execution will be defined in the ScrumDo Backlog.

#### Measure of success/Acceptance Criteria

The QA acceptance criteria and Definition of Done will be defined and tracked in the ScrumDo Backlog.

#### Assumptions

#### Constraints (from customer)

GSA 18f provided the following constraint:

* Use of OpenFDA dataset

#### Risks

Risks will be identified during Sprint Planning, daily Stand-ups, and Sprint Review and managed to completion by the Scrum Master through the ScrumDo Backlog.

#### High-level Technical Approach

The technical approach for Drug IQ will employ Agile methodologies along with DevOp principles. All technical documentation will be stored in GitHub repository at the following link:

<https://github.com/Geocent/18f-prototype/tree/integration>

### Schedule Summary

Drug IQ will be developed using an Agile schedule based on five Sprints. These are

* Sprint 0 – Capture Sprint
* Sprint 1 – Prototype: Architecture, Development, Test Sprint
* Spring 2 – Prototype Finalization Sprint
* Sprint 3 – Prototype Enhancement Sprint
* Sprint 4 – Stablization/Delivery Sprint

This Agile schedule will be managed in ScrumDo.

#### Period Of Performance

The Period of Performance will be from Wednesday June 17th, 2015 through Wednesday,   
July 1st, 2015.

#### Milestones & Releases

The milestones and releases for Drug IQ will include:

* Initial mock up design
* Initial working prototype on AWS
* Final delivery of Drug IQ

### Deliverables

The following list will comprise the services, products, documentation, and other deliverables associated with the project:

* Working Prototype on Public Site (AWS)
* Technical Approach hosted in GitHub
* Agile practices in ScrumDo
* Complete Agile Delivery Services RFQ Compliance Review Checklist Google Form
* Attachment B: Labor Category Mapping and Discount
* Attachment C: Price Quote Sheet

# References

## Source Documentation

Table 1 lists the documents that will form the foundation for establishing the Drug IQ Program Management Plan. The project source documents are listed in order of precedence in the event there are competing statements within multiple documents.

Table : Project Source Documents

|  |  |  |
| --- | --- | --- |
| Project Source Documents | | |
| **Work Product Identifier/Title** | **Path / Storage Location** | **Level of CM Control** |
| SOW/RFP/RFQ/Initial Requirements | <https://intranet.geocent.com/secure/businessdevelopment/SiteDirectory/gsaagile/_layouts/15/start.aspx#/> |  |
| Contract | <https://intranet.geocent.com/secure/businessdevelopment/SiteDirectory/gsaagile/_layouts/15/start.aspx#/> |  |
| Proposal | <https://intranet.geocent.com/secure/businessdevelopment/SiteDirectory/gsaagile/_layouts/15/start.aspx#/> |  |

## Project Working Documents

Table 2 lists the documents that will support the regular tracking and management of all aspects of the Drug IQ project. This table includes the name of the work product, the link/path to the document’s location, and the level of Configuration Management (CM) control.

Table : Project Internal Working Documents

|  |  |  |  |
| --- | --- | --- | --- |
| Project Internal Working Documents | | | |
| **Work Product Identifier/Title** | **Path / Storage Location** | **Level of CM Control** |
| Project Management Plan | [PMP](https://intranet.geocent.com/secure/businessdevelopment/SiteDirectory/gsaagile/Approach%20To%20Create%20the%20Prototype/Geocent%20Project%20Management%20Plan%20Template%20-%20Scrum.docx?Web=1) | Project Lead |
| Product Backlog | [Scrumdo.com Backlog](https://www.scrumdo.com/projects/project/18f-ads-prototype/iteration/128546) | Team |
| Backlog Grooming /Sprint Planning Report | <https://github.com/Geocent/18f-prototype/tree/integration/docs/scrum> | Scrum Master |
| Sprint Review | <https://github.com/Geocent/18f-prototype/tree/integration/docs/scrum> | Scrum Master |
| Sprint Launch/Kick Off/Sprint 0 | <https://github.com/Geocent/18f-prototype/tree/integration/docs/scrum> | Scrum Master |
| Sprint Retrospective | Included in Sprint Review | Scrum Master |
| Code Review Forms | <https://github.com/Geocent/18f-prototype/tree/integration/docs/scrum>  Code Review forms tracked by Story number within each Sprint | QA Manager/ Reviewers |
| Sprint Status Reports | [Scrumdo](https://www.scrumdo.com/projects/project/18f-ads-prototype/summary) and export to [GitHub Repo](https://github.com/Geocent/18f-prototype/tree/master/docs/scrum) | Scrum Master |
| Geocent Status Reports | [Scrumdo](https://www.scrumdo.com/projects/project/18f-ads-prototype/summary) and export to [GitHub Repo](https://github.com/Geocent/18f-prototype/tree/master/docs/scrum) | Project Manager |
| Project Review Documents (IPR Presentations) | [Scrumdo](https://www.scrumdo.com/projects/project/18f-ads-prototype/summary) and export to [GitHub Repo](https://github.com/Geocent/18f-prototype/tree/master/docs/scrum) | Project Manager |
| Project Burn-Down Chart | [Scrumdo](https://www.scrumdo.com/projects/project/18f-ads-prototype/summary) and export to [GitHub Repo](https://github.com/Geocent/18f-prototype/tree/master/docs/scrum) | Scrum Master |
| Geocent Project Estimator | [Scrumdo](https://www.scrumdo.com/projects/project/18f-ads-prototype/summary) and export to [GitHub Repo](https://github.com/Geocent/18f-prototype/tree/master/docs/scrum) | Project Manager |

# Organization

## External Interfaces

Table 3 provides the external interfaces that will be used for the Drug IQ project.

Table : External Interfaces

| **External Administrative, Managerial, and Customer Interfaces** | | |
| --- | --- | --- |
| **External Contact Info** | **Organization/Role** | **Geocent Liason** |
| GSA | Receives Deliverables | Contracts@geocent.com |
|  |  |  |

## Internal Structure: Geocent Program Management Structure

Figure 1illustrates the **18f ADS Prototype** program management structure for the effort. This figure also represents the program’s relation to the Geocent organization.

Figure : Program Management Structure

## Roles and Responsibilities

Table 4 details the roles, labor categories, and personnel targeted to support the Drug IQ project.

Table : Drug IQ Roles, Labor Categories, and Personnel

| **Drug IQ Project Roles** | | | |
| --- | --- | --- | --- |
| **Role** | **Responsibilities, Required Knowledge/Skills** | **Labor Category** | **Person** |
| Product Owner | * See Attachment A Labor Category Descriptions.pdf | Product Manager | Keith Alphonso |
| Project Lead | * See Attachment A Labor Category Descriptions.pdf | Technical Archtiect | Jared Ladner |
| Scrum Master | * See Attachment A Labor Category Descriptions.pdf | Agile Coach | Roberta Hazelbaker |
| UX Lead | * See Attachment A Labor Category Descriptions.pdf | Visual Designer | Vance Lowe |
| Developers | * See Attachment A Labor Category Descriptions.pdf | Frontend Web Developers | Randy Nolan Aaron Whitney  Josh Penton |
| DevOPS Lead | * See Attachment A Labor Category Descriptions.pdf | DevOPS Eng | Tyler Sanders |
| QA Lead | * See Attachment A Labor Category Descriptions.pdf | Delivery Manager | Brian Priest |

# Managerial Plans

## Start-Up Plan

The start-up plan will constitute Sprint 0. It will begin with a Kick-off meeting to outline the details of the project requirements set by GSA 18f, the project vision, project goals, Sprint iterations, Epics and Features and Definition of Done will be set. Sprint 0 is considered the Capture Sprint, which includes reviewing materials and establishing the environment, infrastructure, Continuous Integration (CI), Configuration Management (CM), Testing plans, and Scrum schedules. The following actions were addressed and implemented:

* Establish Communications
* Set up GitHub repo
* Set up Skype channel
* Set up ScrumDo
* Set up intranet document site
* Initiate project: Sprint 0
* Concept
  + - Review OpenFDA data and define concept for application
    - Conduct brainstorming meeting
    - Develop Vision Document
* Development
  + - Study OpenFDA APIs and architecture requirements
    - Develop initial architecture design
    - Develop Architecture Document
* Infrastructure / DevOps
  + - Build out standard Geocent CI environmnent
    - Develop CI description document
* Develop PMP / Rules of the Road
  + - Identify Branching Strategy
    - Establish Definition of Done
      * Test coverage requirements
      * Check-in processes
      * Code review requirements
    - Establish Deployment process / Configuration Management (CM)
    - Organize Scrum Plan
      * Daily Standups
      * Sprint duration / schedule
      * Backlog grooming schedule

### Establish Resources

#### Staffing

The staffing plan for Drug IQ is detailed in Table 4 and Attachment A Labor Category Descriptions.pdf.

#### Training

Table 5 details the training that has been identified as necessary for execution of the Drug IQ project.

Table : Training Course Description

|  |  |  |
| --- | --- | --- |
| **Training** | **Description** | **Formal/Informal** |
| AngularJS | Training will be provided to all developer resources by the Frontend Designer | Informal |
| Docker | Training will be provided to all developer resources by the DevOps Eng | Informal |

#### Facilities

The primary work facility for this effort will be Geocent facilities in New Orleans and Baton Rouge, LA; Stennis Space Center, MS; and Charleston, SC.

#### Subcontractors

Subcontractors will not be used on this project.

#### Tools

The tools and technologies that will support the development of Drug IQ are listed in the README.md file located at the <https://github.com/Geocent/18f-prototype> .

#### Hardware Resources

Table 6 details the hardware resources that will be needed to support the Drug IQ project.

Table : Hardware Resources

| **Product** | **Specifications** | **Qty** |
| --- | --- | --- |
| Dev Laptops | Mac Book Pro | 8 |
| Other Laptop | Dell | 1 |
| AWS | 3 Mid Servers: CI, Dev, and Prod  ‘ads.geocent.com’  ads-dev.geocent.com  ads-ci.geocent.com | 3 |
| ScrumDo | Geocent licensed | 7 |
| GitHub | Open License | 7 |

## Execution Plan

The execution plan to support Drug IQ will follow Scrum methodology.

### Manage Code

All code will be managed in GitHub Repository ([GitHub Repo](https://github.com/Geocent/18f-prototype)).

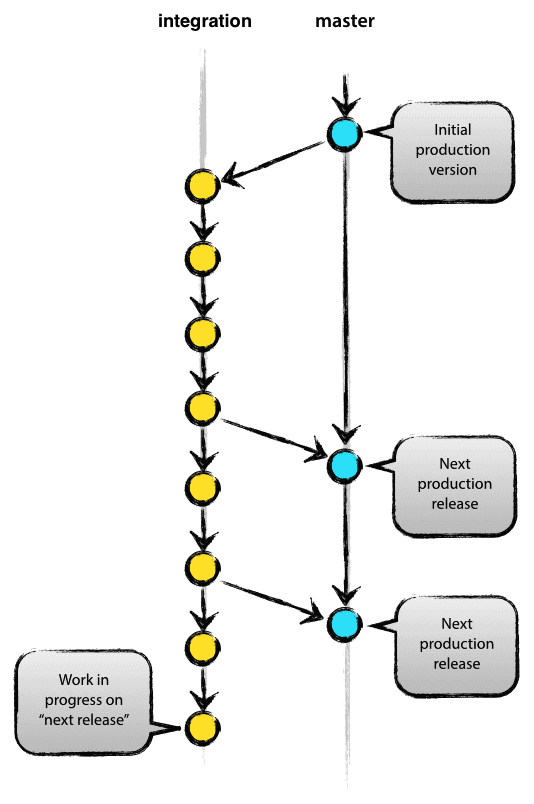
#### Branching Strategy

Geocent will utilize the branching strategy documented by Vincent Driessen (<http://nvie.com/posts/a-successful-git-branching-model/>). The branching strategy will consist of

* Main Branch
* Supporting Branch
* Feature Branch
* Release Branch
* Hotfix Branch

This section documents Mr. Driessen’s principles that Geocent has employed with much success. These branching strategies will be implemented to support Drug IQ.

**Main Branch**

The central repository will contain two main branches:

* master
* integration

Both branches will have infinite lifetime conditions (Figure 2).

The master branch at origin should be familiar to every Git user. Parallel to the master branch, another branch exists called integration.

We consider origin/master to be the main branch where the source code of HEAD always reflects a production-ready state.

We consider origin/integration to be the main branch where the source code of HEAD always reflects a state with the latest delivered development changes for the next release. Some would call this the “integration branch.” This is where the code for automatic nightly builds are stored.

When the source code in the integration branch reaches a stable point and is ready to be released, all of the changes should be merged back into master and tagged with a release number.

Figure : Master/Integration Branch

Therefore, each time when changes are merged back into master, it is considered a new production release. At this point we would use a Git hook script to automatically build and roll-out Drug IQ to production servers everytime there is a commit on master.

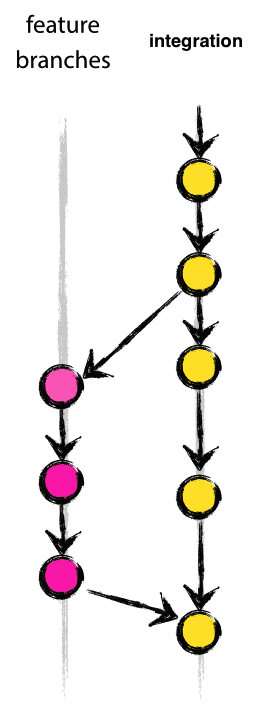
**Supporting Branch**

Next to the main branches master and integration, our development model will use a variety of supporting branches. Supporting branches will be employed to aid with parallel development between team members, for ease in tracking of features, to prepare for production releases, and to assist quickly with fixing live production problems. Unlike the main branches, these branches have a limited life time, because once they are not longer needed they will be removed.

The different types of branches we plan to use are:

* Feature Branches
* Release Branches
* Hotfix Branches

Each of these branches have a specific purpose in terms of being the originating branch and the merge targets. We plan to adher to the strict rules governing these branches.

**Feature Branches**

Feature branches (Figure 3) can branch off from:

integration

but, must eventually be merged back into:

integration

**In support of Drug IQ, we will use** feature **as the naming convention for Feature branches.**

Feature branches are also referred to as topic branches, These will be used to develop new features for the current or future release. When starting development of a feature, the target release for the feature may not be known. The essence of a feature branch is that it exists as long as the feature is in development. Eventually it will be merge bck inot the development or discarded, if deemed unnecessary to the final product. Feature branches typically exist in developer repositories only. Collaboration can occur on a feature; however it is never found in the origin.

**Creating a Feature Branch**

When starting work on a new feature, branch off from the integration branch using:

$ git checkout -b myfeature integration

Switched to a new branch "myfeature"

Figure : Feature Branch

This will begin the new feature branch off the main integration branch.

**Incorporating a Finished Feature on Integration**

Finished features will be merged back into the integration branch so it can be added to the upcoming release using:

$ git checkout integration

Switched to branch 'integration'

$ git merge --no-ff myfeature

Updating ea1b82a..05e9557

(Summary of changes)

$ git branch -d myfeature

Deleted branch myfeature (was 05e9557).

$ git push origin integration

The --no-ff flag ensures that the merge will always create a new commit object, even if the merge could be performed with a fast-forward. This will avoid losing information about the historical existence of a feature branch and groups together all commits that together added the feature. This is illustrated in Figure 4.

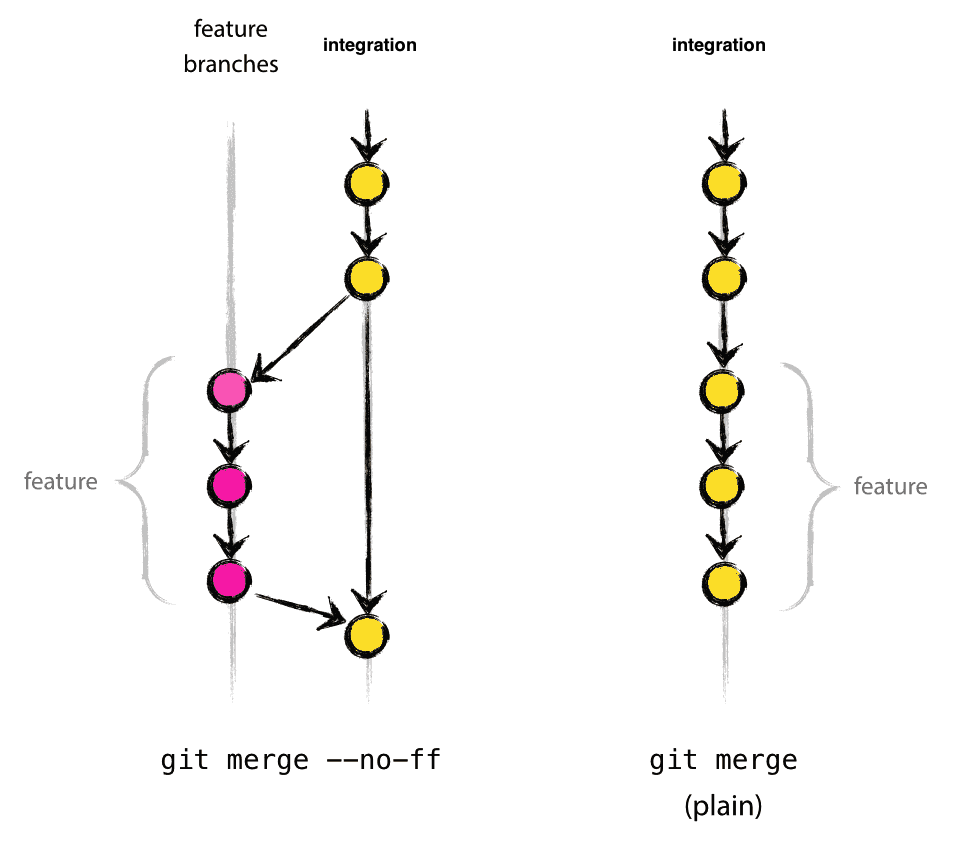


Figure : Feature Branch Incorporated Back to Integration

In the latter case, it is difficult to see from the Git history which of the commit objects together have implemented a feature. The log message would need to be reviewed to understand what occurred. Hence, Geocent, will use the --no-ff flag to easily document feature branch merging back into the integration branch.

**Release Branches**

The release branch will branch off from:

integration

but must be merge back into:

integration and master

The branch naming convention for release branches is release-\*.

Release branches will support preparation of a new production release. They will allow for last-minute updates, minor bug fixes, and prepares meta-data for the release in terms of version number, build dates, and so forth. By doing all of this work on a release branch, the integration branch will be cleared to receive features for the next release.

The key moment to branch off a new release branch from integration will be when integration reflects the desired state of the new release. This means that all features that are targeted for the release-to-be-built are merged in to integration. Future targeted features will not be merged until after the current release branch is branched off.

Versioning releases will occur at the start of a release branch and will be assigned a version number. Once the integration branch reflects changes for the “next release”, the version nmber value will be set. That decision will be made on the start of the release branch and will be carried out by the project’s rules on version number bumping.

**Creating a Release Branch**

Release branches will be created from the integration branch. For example, version 1.1.5 is the current production release. The state of integration is ready for the “next release” and we have decided that this will become version 1.2 (rather than 1.1.6 or 2.0). Branch off will occur and the release branch will have a name reflecting the new version number. This will be accomplished using:

$ git checkout -b release-1.2 integration

Switched to a new branch "release-1.2"

$ ./bump-version.sh 1.2

Files modified successfully, version bumped to 1.2.

$ git commit -a -m "Bumped version number to 1.2"

[release-1.2 74d9424] Bumped version number to 1.2

1 files changed, 1 insertions(+), 1 deletions(-)

After creating a new branch and switching to it, the version number will be bumped. We will use the bump-version.sh, a fictional shell script, that will change files in the working copy to reflect the new version. After this, the bumped version number will be committed.

This new branch will exist there for a while, until the release is rolled out definitely. During that time, bug fixes will be applied in this branch (rather than on the integration branch). Adding large new features here will be strictly prohibited. Any new features will need to be merged into integration, and wait for the next big release.

**Finishing a Release Branch**

When the state of the release branch is ready to become a real release, some actions will need to be carried out. First, the release branch will be merged into master. Next, that commit on master will be tagged for easy future reference to this historical version. Finally, the changes made on the release branch will need to be merged back into integration, so that future releases also contain these bug fixes.

The first two steps in Git include:

$ git checkout master

Switched to branch 'master'

$ git merge --no-ff release-1.2

Merge made by recursive.

(Summary of changes)

$ git tag -a 1.2

The release is now done, and tagged for future reference.

To keep the changes made in the release branch, the changes will need to be merged back into integration using:

$ git checkout integration

Switched to branch 'integration'

$ git merge --no-ff release-1.2

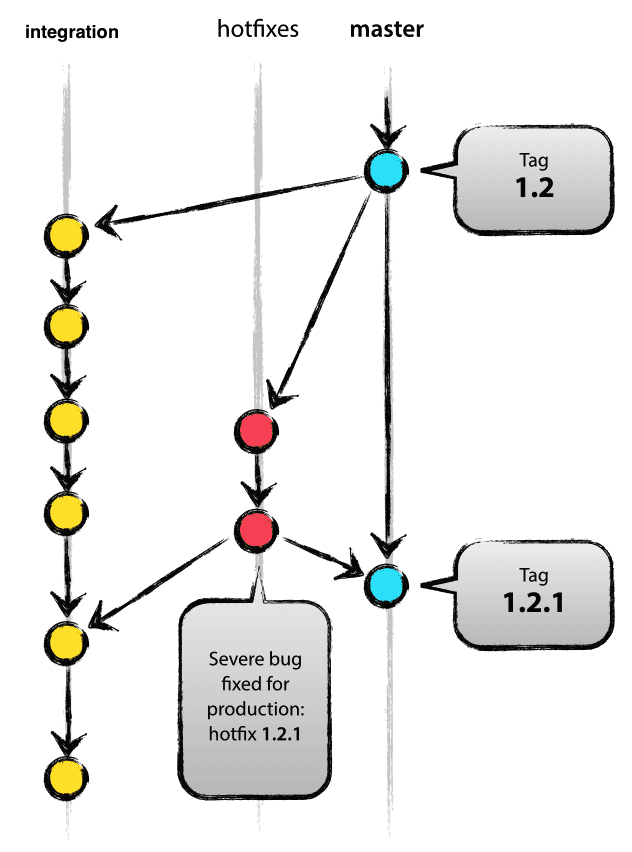
Merge made by recursive.

(Summary of changes)

At this point, the release branch will be done and will be removed using:

$ git branch -d release-1.2

Deleted branch release-1.2 (was ff452fe)

**Hotfix Branches**

Hotfix branches will branch off from:

master

but will need to be merged back into:

integration and master

The naming convention for hotfix branches is hotfix-\*.

Hotfix branches, as shown in Figure 5, are very much like release branches in that they are also meant to prepare for a new production release, albeit unplanned. They arise from the necessity to act immediately upon an undesired state of a live production version. When a critical bug in a production version must be resolved immediately, a hotfix branch will be branched off from the corresponding tag on the master branch that marks the production version. The importance of a hotfix branch is that that a team member can still continue working new features on the integration branch; while another team member is preparing a quick production fix.

Figure : Hotfix Branching

**Creating the Hotfix Branch**

Hotfix branches will be created from the master branch. For example, if version 1.2 is the current production release running live and causing troubles due to a severe bug and changes on integration are unstable; then branching off a hotfix branch to fixing the production problem can occur. Creating the hotfix branch will use:

$ git checkout -b hotfix-1.2.1 master

Switched to a new branch "hotfix-1.2.1"

$ ./bump-version.sh 1.2.1

Files modified successfully, version bumped to 1.2.1.

$ git commit -a -m "Bumped version number to 1.2.1"

[hotfix-1.2.1 41e61bb] Bumped version number to 1.2.1

1 files changed, 1 insertions(+), 1 deletions(-)

After branching off, the version number will need to be bumped in accordance with versioning rules. The bug will be fixed and the commit will occur using:

$ git commit -m "Fixed severe production problem"

[hotfix-1.2.1 abbe5d6] Fixed severe production problem

5 files changed, 32 insertions(+), 17 deletions(-)

**Finishing a Hotfix Branch**

When finished, the bugfix will need to be merged back into master and integration, in order to safeguard that the bugfix will be included in the next release. This is similar to how release branches will be finished.

First, update master and tag the release using:

$ git checkout master

Switched to branch 'master'

$ git merge --no-ff hotfix-1.2.1

Merge made by recursive.

(Summary of changes)

$ git tag -a 1.2.1

Next, include the bugfix in integration using:

$ git checkout integration

Switched to branch 'integration '

$ git merge --no-ff hotfix-1.2.1

Merge made by recursive.

(Summary of changes)

There is one exception to the rule. When the release branch hotfix changes are done, they will need to be merged into that release branch, instead of integration. Back-merging the bugfix into the release branch will eventually result in the bugfix being merged into integration, when the release branch is finished. If work in integration immediately requires this bugfix and cannot wait for the release branch to be finished, we will safely merge the bugfix into integration which will be ready.

This temporary branch will need to be removed using:

$ git branch -d hotfix-1.2.1

Deleted branch hotfix-1.2.1 (was abbe5d6).

This branching strategy will be used to support the Drug IQ project. This will be documented in [GitHub Repo](https://github.com/Geocent/18f-prototype).

### Manage Requirements

Table 7 details the practices and tools that will be used for managing requirements.

Table : Tools and Methods to Manage Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirements Management Method/Tool** | **Procedure** | **Evidence** | **Description** |
| Initial Requirements | Proposal Process | Proposal | Initial requirements and understanding are provided in the proposal and accepted with award. |
| Requirements Understanding | Backlog Grooming | ScrumDo Stories | Requirements will be analyzed, decomposed, clarified, prioritized, and documented. |
| Requirement Commitment | Sprint Planning | ScrumDo Assignment | During Sprint planning the customer commits to a specific set of user stories. |
| Requirements Change Management | Backlog Grooming | ScrumDo | Requirements will be analyzed, decomposed, clarified, prioritized, and documented. |
| Requirements Tracability | Backlog Grooming | Requirement for GitHub check-ins to reference ScrumDo Stories |  |
| Requirements implementation consistency | Sprint Review | Review tasks in ScrumDo | User accepts Stories implemented in Sprint |

### Manage Communications

The following communications mechanisms will be used to ensure effective communication between stakeholder management, Geocent senior management, and the project team.

#### Reporting Mechanisms

| **Status and Reporting Plan** | | |
| --- | --- | --- |
| **Report Name** | **Performed By Whom** | **Frequency** |
| Geocent Status Report | Project Manager | N/A |
| Sprint Status Report | Scrum Master | end of each sprint |
| Required Contract Deliverable Reports | Project Manager | end of project |
|  |  |  |

#### Required Meetings

| **Meeting & Interface Plan** | | |
| --- | --- | --- |
| **Meeting Name & Purpose** | **Attendees** | **Frequency** |
| Sprint planning | Team | 2 days |
| Sprint Review | Team | 2 days |
| Backlog Grooming | Customer  Leads | As needed |
| In Progress Review | Project Leads  Geocent Leadership | N/A |
| Daily Scrum Standup | Team | Daily |

### Manage Quality

Geocent will implement Quality Assurance in order to assure it meets both internal standards and contractual obligations. QA activities will take place according to Delivery Manager Quality Plan.

#### Developer Testing

Develoeprs create and execute tests locally includes:

* Unit Testing: Uses karma by executing ‘gulp test’
* Integration Testing: Uses Protractor by executing ‘gulp protractor’

#### Continous Integration Testing

On branch ‘integration’ and ‘master’ the CI will execute:

* Unit Testing: Uses karma by executing ‘gulp test’: <https://ads-ci.geocent.com/job/build-dev/>
* Integration Testing: Uses Protractor by executing ‘gulp protractor’: <https://ads-ci.geocent.com/job/functional-dev/>

#### Usability Testing

Usability testing will initial be performed on UI Wireframe Mockups, then later conducted to be on the working prototype after the first working iteration is deployed to a stable environment: <http://ads.geocent.com>

UX issues and requests tracked as defined in Section 4.2.6.

#### User Acceptance Testing

User Acceptance Testing is conducted the Product Owner during Sprint Review individual feature and a full UAT is conducted on each realease.

#### Code Review

Code reviews will be conducted as a Task in the development stories. Peers will conduct code reviews looking for:

* Feature Implementation completion
* Unit test coverage
* E2E test review
* Validated on CI: <http://ads-dev.geocent.com>

### Manage Measurements (Metrics)

| **Program Measurements and Metrics Plan** | | | | |
| --- | --- | --- | --- | --- |
| **Measure** | **Frequency Collected** | **Collected by Whom** | **Analyzed by Whom** | **Used by Whom** |
| Schedule Adherence | daily | PM | Senior Management | Geocent |
| Budget Adherence | daily | PM | Senior Management | Geocent |
| Funding | daily | PM | Senior Management | Geocent |
|  |  |  |  |  |

The project measures will be collected, analyzed, reported, and used according to the plan above.

### Manage Issues

The project will use [Scrumdo.com](https://www.scrumdo.com/projects/project/18f-ads-prototype/summary) to track all bugs, issues and change requests that are not UX related. Issues can be resolved by one of the following:

1. Fix the issue
2. Modify the requirement
3. Obtain a waiver on the issue

Resolving issues may involve negotiation between one or more parties. If the parties are unable to agree who is responsible for resolving the issue, the issue must be escalated to higher levels of project leadership.

User Experience issues will be recorded and maintained in [GitHub Repo](https://github.com/Geocent/18f-prototype).

1. Issues found during UX meetings will be recorded in [GitHub Issue Tracker](https://github.com/Geocent/18f-prototype/issues)
   1. Issue will include: name of user, related UX session, and change request
2. If approved by the Product Manager, Story/Stories will be created in [Scrumdo.com](https://www.scrumdo.com/projects/project/18f-ads-prototype/summary)
3. Status will be updated in GitHub to reflect if it will be worked, rejected, working, completed

**Monitor Project Risks**

Project risks shall be documented in the project status report and updated periodically as defined by the PMP (same period as status report). These risks will be captured during regular project meetings as defined by the PMP.

**Manage Corrective Action to Closure**

Issues are identified and analyzed during periodic meetings as a regular agenda item (as defined by the PMP).

If an issue is deemed to require a corrective action, documented as part of the meeting report (as defined by the PMP) and reported in regular status reports until resolved.

### Manage Contract Changes

Geocent will address schedule and requirements changes on a case-by-case basis, providing impact assessment to customer and revising this PMP accordingly.

Customer requested changes to the contract, or changes due to program internal re-planning activities will not be performed without agreement by the Project Engineer, Program Manager, and Contracts.

The incorporation of these authorized changes will be made in a timely manner and strictly controlled. Traceability to the original baseline budget will be maintained in order to provide a basis against which program growth can be measured.

## Work Plan

Work plan is documented in tasks and stories in ScrumDo

## Closeout Plan

| **Program Closeout Plan** | | |
| --- | --- | --- |
| **Activities** | **Performed By Whom** | **Comments** |
| Identify and determine disposition of outstanding work | PM |  |
| Prepare next release, operations, maintenance or transition plan | NA |  |
| Close all logs and accounting records | PM/Contracts |  |
| Conduct post-project review and document lessons-learned | NA |  |
| Complete project team post-assignment evaluations and release team members to next assignment | Senior Management |  |
| Complete customer satisfaction review | NA |  |
| Analyze project metrics, product quality, and issue final project report | NA |  |
| Update estimating models and risk factor tables | NA |  |
| Archive documents, records, reports and notify appropriate personnel | CM |  |
| Prepare “Lessons Learned” Survey | NA |  |
| Prepare past performance writeup & presentation slide | PM |  |