PHP Dev

Deployment Guidelines

Author: Shafeeg Karadsheh

Draft Version: 3.0

Last Updated: 20120815

Table of Contents

1 Infrastructure Overview

1.1 Environments

During the lifecycle of any development project, we encounter countless hurdles from analysis, estimation, prioritization, unit and acceptance testing, quality assurance, and deployment. It is extremely important that all releases traverse various development environments to ensure the code being pushed is "production-ready".

There are 5 environments we will review:

LOCALHOST

The initial sandbox environment for code creation resides on the engineer workstation. Tested is also completely locally in preparation for deployment to the DEV environment.

DEVELOPMENT (DEV)

During most development projects, we are dealt complex functional requirements that require distribution of development tasks (frontend, backend, etc.). After progressing through the initial localhost environment, the dev environment will typically be the first point for code integration. All code initally created will be put to the test against new environment variables and against other collaborative team code. QA work begins and runs a general *"smoke"* test to verify all components have been given a quick run-through so that we can resolve any issues prior to deployment on the next environment. This environment is hosted on a locally-available engineering server, typically with administrator-level privileges.

INTEGRATION (INT)

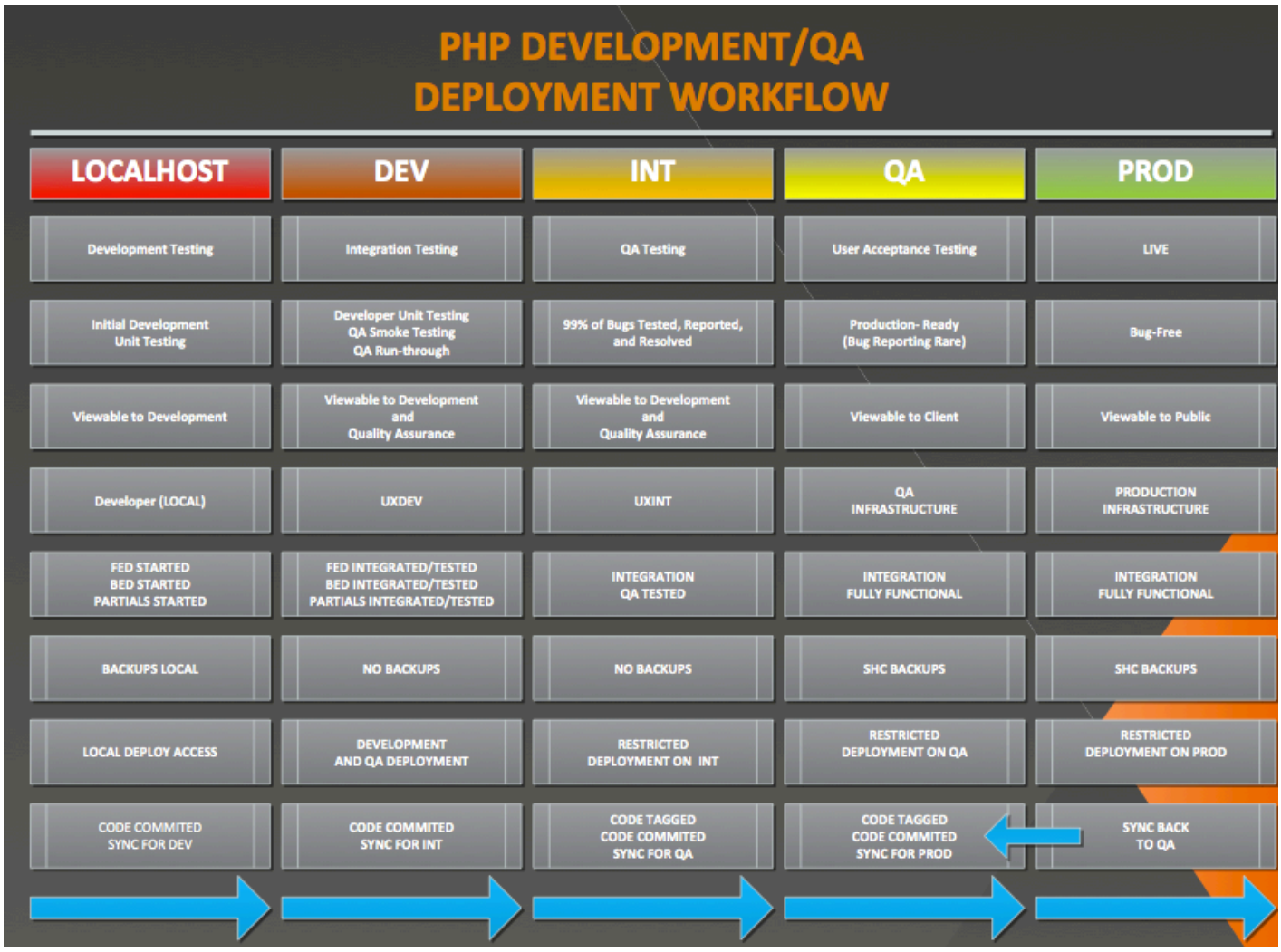
The INT environment contains all functionally-merged code ready for official QA testing. This environment will experience the most interaction, as it is will be the last internally-facing environment prior to external (non-engineer) visibility. This consists of a full suite of QA tests, bug-resolution, as well as any next-iteration development for an existing live site. Like the DEV environment, this environment is also hosted on a locally-available server. Only a subset of individuals, or *"gatekeepers"*, will be given deployment access from this environment to the next to diligently determine production-ready code.

QA

Official public facing environment where clients, product, and project managers are given internal exposure and visibility into the project. At this juncture of the development cycle, 99% of bugs should be resolved. Any bug reports should be rare and/or unique in nature. Upon further development and future iterations, we would ideally only test environment changes. The QA environment should be considered a mirror image of production so that we can throroughly test any environmental modifications prior to live public release. Similar to the INT environment, only "*gatekeepers"* will be given deployment access from this environment to the next to diligently determine production-ready code.

PROD

Production will be the live environment served to the public. There should not be any software bugs present on this environment. At this point, all development should be solidified after traversing the previous 4 environments. The production system contains a load balanced setup with multiple nodes executing the newly developed functionality. Any deployments to this environment should verify that the configuration across all nodes are identical to ensure stability. It my difficult to troubleshoot issues at times if one server on the node is experiencing issues. Only a subset of the end-users will actually reach this node with a load-balanced setup. Once any deployment is complete, we will periodically merge the production setup back to QA to ensure these two environments are synced.



1.2 Tag Progression and Deployment

Now that we have discussed the various environments, tagging code becomes especially important. During several iterations of development and QA, units of code are likely revised and re-worked. Source code control become difficult if there isn't a careful tagging structure isn't in place as development traverses from one environment to the next.

Source Code Tagging shall take the following format:

PROJCODE-RELEASE-ENVIRONMENT-DESCRIPTION-DATE-VERSION

Let's take the following tag as an example:

COMM-20120731-V2

COMM = Project Code for Communities

20120731 = Release Date

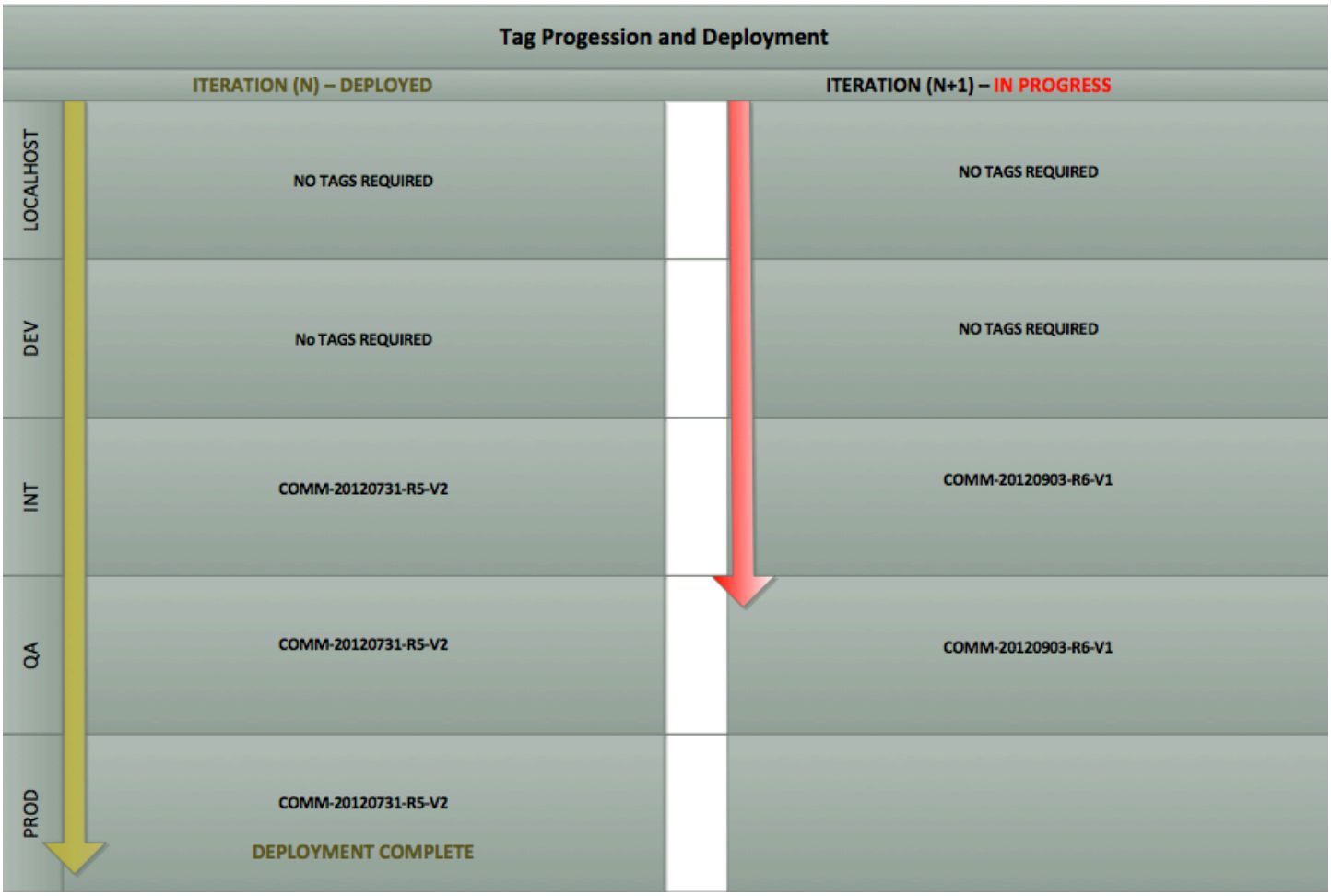
R5 = Release Iteration

Tag Number = Incremental Project Tag Number. Value reset per project.

V2 = Version (only for Hotfixes)

In the left column of the chart below, you will see the tag progression of source code completed for the WordPress Recent Activity FED. As the code traverses from environment to environment, several variables (such as Environment, Date, and Version) are being updated in the process. It may also be very common that concurrent next-iteration development is in progress, as shown in the right column.

After development is complete and the code has been tagged, the developer is responsible for deploying the code or arranging deployment (depending on environment). As outlined previously in the document, only a subset of individual *"gatekeepers"* are given deployment access.

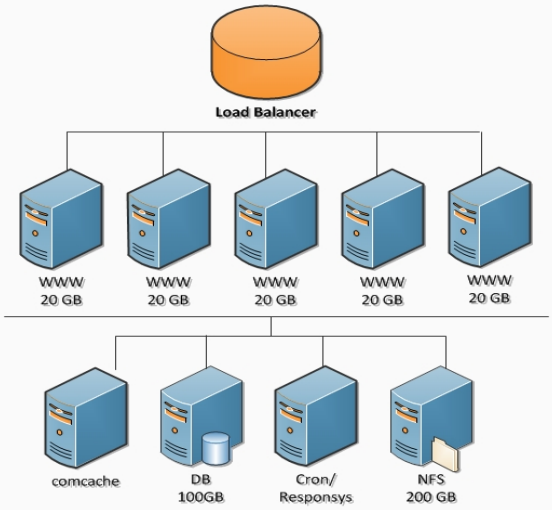


1.3 Current Infrastructure (Communities)

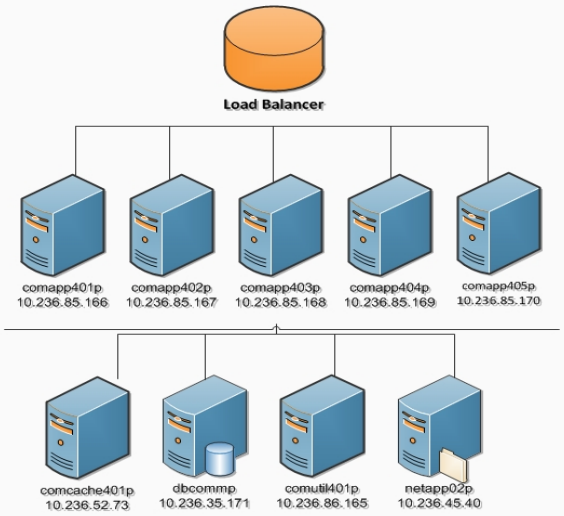
Production Servers / VMs / Storage

1. comutil401p.prod.ch4.s.com – Stand-alone server for crons, responsys and reporting.
2. comapp401p.prod.ch4.s.com – Web Server - 1 Core | 4 GB RAM | 20 GB HDD
3. comapp402p.prod.ch4.s.com – Web Server - 1 Core | 4 GB RAM | 20 GB HDD
4. comapp403p.prod.ch4.s.com – Web Server - 1 Core | 4 GB RAM | 20 GB HDD
5. comapp404p.prod.ch4.s.com – Web Server - 1 Core | 4 GB RAM | 20 GB HDD
6. comapp405p.prod.ch4.s.com – Web Server - 1 Core | 4 GB RAM | 20 GB HDD
7. comcache401p.prod.ch4.s.com – memcache - Physical server 16 GB RAM
8. dbcommq.prod.ch4.com – Dedicated database cluster with 100 GB allocation
9. netapp02p.ch4.s.com – NFS - *comm_sys_prod* with 200 GB allocation

Production Infrastructure



-prod4.ch4.s.com-



SERVER HOSTNAME	DESCRIPTION
comutil401p.prod.ch4.s.com	Stand-alone server for crons, responsys and reporting.
comapp401p.prod.ch4.s.com	Web Server - 1 Core | 4 GB RAM | 20 GB HDD
comapp402p.prod.ch4.s.com	Web Server - 1 Core | 4 GB RAM | 20 GB HDD
comapp403p.prod.ch4.s.com	Web Server - 1 Core | 4 GB RAM | 20 GB HDD
comapp404p.prod.ch4.s.com	Web Server - 1 Core | 4 GB RAM | 20 GB HDD
comapp405p.prod.ch4.s.com	Web Server - 1 Core | 4 GB RAM | 20 GB HDD
comcache401p.prod.ch4.s.com	memcache - Physical server 16 GB RAM
dbcommq.prod.ch4.com	Dedicated database cluster with 100 GB allocation
netapp02p.ch4.s.com	NFS - *comm_sys_prod* with 200 GB allocation

SERVER HOSTNAME	Physical/VM	IP ADDRESS
comutil401p.prod.ch4.s.com	VM	10.236.85.165
comapp401p.prod.ch4.s.com	VM	10.236.85.166
comapp402p.prod.ch4.s.com	VM	10.236.85.167
comapp403p.prod.ch4.s.com	VM	10.236.85.168
comapp404p.prod.ch4.s.com	VM	10.236.85.169
comapp405p.prod.ch4.s.com	VM	10.236.85.170
dbcommp.prod.ch4.com	VM	10.236.35.171
comcache401p.prod.ch4.s.com	Physical	10.236.52.73
netapp02p.prod.ch4.s.com	Physical	10.236.45.40

2 Release Management

2.1 Release Structure

To provide additional structure into our deployments, we will be initiating a new release management structure. Work scoped and estimated from the *Statement of Work* will broken into several key core components. Each component will be evaluated and prioritized into releases. Our release cycles will be every 4 weeks, broken into 2 sprints. Each sprint iteration will initially be broken into 2-week cycles. Management will dictate the development for each sprint. The deadline for scheduled for an upcoming sprint should be submitted no later than EOB Wednesday the week prior to the beginning of the sprint. Production deployments will be executed on Tuesday after the end of the sprint. All “production-ready” code should be committed by EOB on the first Monday of the new release cycle.

2.2 Sprint Structure

At the start of every sprint, each team will be required to formally meet and review the user stories and tasks and determine a strategy of attack for completing the work on schedule. This mandatory meeting will take place at the start of the sprint on every other Monday at 10:00 a.m. - 12:00 p.m. CST.

General development updates will be provided on a daily basis in morning **Standups** at 9:30 a.m.

Individual team meetings (i.e. FED Team Meetings / BED Team Meetings) may be decided on a per-sprint basis by team.

On the last day of the sprint, an "End of Sprint Review" will be conducted with all members. Each member will be responsible for being prepared for the meeting, and providing a **functional** demo of their work to the group assigned in the sprint (if applicable). Potential issues, such as incomplete functionality or bugs, should be brought to the attention of the Release and Delivery Managers prior to the meeting. This mandatory meeting will be scheduled at the end of the sprint on every other Friday from 9 to 11 a.m. CST.

As the last stage of the sprint, each team should conduct a sprint post-mortem meeting to determine the successes of the sprint, as well as any pitfalls. Our goal is to prevent any repetitive issues. Any issues that occur frequently should indicate lack of general process. These should be notated and updated in Mingle.

2.3 Backlog

It may be common that work scoped for sprints does not complete by the end of the sprint. These incidents may occur frequently in the infancy of this process, however should be handled with discretion (depending on project schedule). This may be due to a variety of dependencies, such as underestimating scope of task, lack of tools or environment, or simply lack of focus. After several iterations of development, the team should adjust and become more accurate on work scoped by sprint.

2.4 Release Notes

After a major release (2 sprints) has been deployed, the Release manager is responsible for documenting all the feature additions, bug fixes, and any other additional release information into Mingle under “Release Notes”. This will be required for every release so that we can keep track of projects in progress, and also keep documented references of available functionality for potential future projects.

Below you find a calendar with the typical sprint schedule (including Sprint Planning meetings, End of Sprint Reviews, as well as Daily Standups)

AUGUST

Sun	Mon	Tues	Wed	Thurs	Fri	Sat
			1	2	3	4
5	6 1st Sprint Sprint Planning Meeting	7 Daily Standup	8 Daily Standup	9 Daily Standup	10 Daily Standup	11
12	13 Daily Standup	14 Daily Standup	15 Daily Standup	16 Daily Standup	17 End of 1st Sprint Sprint Review	18
19	20 2nd Sprint Sprint Planning Meeting	21 Daily Standup	22 Daily Standup	23 Daily Standup	24 Daily Standup	25
26	27 Daily Standup	28 Daily Standup	29 Daily Standup	30 Daily Standup	31 End of 2nd Sprint Sprint Review	

3 User Story Management

20 Step Guideline

1. All development tasks are to be broken into User Stories

2. User stories and tasks should be created within Mingle by BA, DM, or Reporter

3. Each user story should have initial acceptance tests created by BA, DM, or Reporter

4. User stories are placed into team buckets and prioritized

5. Teams are required to review their bucket of assigned mingle stories at the start of every sprint

6. Teams are required to distribute the work amongst team resources

7. Mingle stories are assigned to the developer at the start of every sprint

8. Developer is responsible for reviewing the story, promptly communicating any questions back to reporter, updating the mingle story if necessary, and creating additional acceptance tests

9. Developer is responsible for identifying and reporting any dependencies or blockers during development and notifying management (typically within 24-48 hours)

10. Once development on an environment is complete, the developer is responsible for verifying acceptance criteria have been met, updating the mingle story, and arranging deployment to the next environment

11. Once the code is believed to be “production-ready”, then the developer is responsible for providing a demo of the functionality to the team in the end of sprint review

12. The developer is responsible for closing out the cycle process by re-assigning the card in mingle for the next iteration/QA phase (if necessary), or back to the BA, DM, or Reporter if deployment is complete

13. The BA, DM, or Reporter is responsible for coordinating QA efforts after deployment

14. QA is responsible for verifying code unit is tested and reports results within test plan

15. QA is responsible for creating bug in Jira and assigning to corresponding developer

16. Developer is responsible for working with QA to identify issue and resolve bug

17. Developer is responsible for updating source code on all environments, re-tagging, and arranging re-deployment of newly updated code

18. Developer is responsible for updating Jira bug with status and notes and assigning Jira to QA for verification of bug resolution

20. QA is responsible for updating Jira as fixed/verified and notifying BA, DM, or Reporter

4 Executing Quality Assurance

4.1 What is User Acceptance Testing?

User Acceptance Testing (UAT) - also called beta testing, application testing, and/or end user testing - is a phase of software development in which the software is tested in the "real world" by the intended audience or a business representative. Whilst the technical testing of IT systems is a highly professional and exhaustive process, testing of business functionality is an entirely different proposition.

The goal of User Acceptance Testing is to assess if the system can support day-to-day business and user scenarios and ensure the system is sufficient and correct for business usage.

4.2 Test Strategy

It is important to develop a testing strategy prior to initiating quality assurance (QA) on any project. Careful consideration must not only be placed on the functional requirements, but also the less obvious aspects, such as browser-based testing, UX and design, environment, as well as stress testing. With any enterprise level online site, we are destined to encounter end-users with various home and business setups. For example, it may be typical for a company to set corporate standards (for security purposes) on workstation configurations. These companies are often restricted to browse Internet site using older browsers. On the other hand, a start-up technology company might be completely utilizing tablets for their business operations. These are just examples of things that should be considered in regular QA testing.

Testing may be broken down into an initial outline or draft containing the various buckets, such as sections of the site, unit-testing, business category, or by priority. It is extremely important that the test plan capture everything from the most common end-user workflow, to the least likely. In doing so, we must verify all that each piece of the puzzle displays, functions, and interacts as it was intentionally designed. For these reasons, it is important that a test plan be outlined, drafted, created, and verified to accommodate each test and expected result.

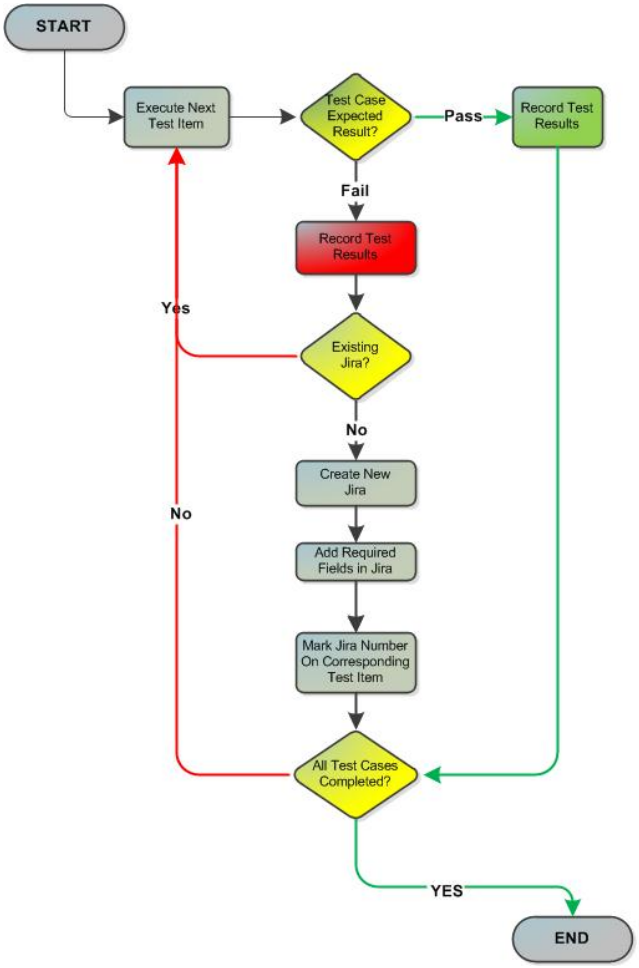
4.3 Creating the Test Plan

A test plan is absolutely necessary for large-scale development projects. During the QA lifecycle, a unit of code or page design may see several revisions in efforts of achieving the actual expected result. A test plan should contain a series of tests of the various components from user design to functionality, document the expected result, and report the actual findings. The test plan should contain test results (by environment), as well as any Jiras linked to the issue. IThe same process goes for any blockers or dependencies that must be addressed by other teams. In doing so, the test plan we will be provided greater visibility into the stability of the source code, and will ultimately help management check project health.

4.4 Test Plan Execution Workflow / Bug Reporting

Please refer to the test plan execution workflow shown in the flowchart below. During QA testing, it is extremely important that developers are not reporting duplicate bugs. QA testing should be delegated by section (per QA lead), and will be responsible for thoroughly testing each assigned scenario. During this process, each tester should verify that any cases that do not meet the expected result do not have an existing Jira open. This may cause confusion, especially if the notes have all been placed in one Jira, and not the other. Any duplicate bugs found should be closed, and communicated back to the QA lead.

You will also find information on what prerequisite information to provide for all reported Jira issues. It is **mandatory** that all of the required information be populated into the Jira prior to initiation of any work.



Additional Notes

1. *Create a new Epic for Communites Q/A*
2. *Assign all Jira bugs to parent epic*
3. *All Jira bugs should have watchers*
4. *All team members should have Jira email notifications enabled*
5. *All bugs should be prioritized (minor, major, critical, blocker)*
6. *Bugs will be addressed based upon priority*
7. *Jira bugs should contain:*
    a. ***Appropriate Subject Header***
    b. ***Summary of problem*** *(short description)*
    c. ***Test case number*** *(from test plan)*
    d. ***Steps to replicate problem*** *(including URLs)*
    e. ***Relevant error messages*** *(verbose, if available)*
    f. ***Screenshot*** *(if applicable)*
    g. ***Dependencies*** *(links to other related or blocker Jiras)*
8. *Jiras bugs should be continuously be updated through lifecycle (from creation to resolution)*
9. *All corrective measures should be documented throughout process (for future reference)*
10. *All Jiras should be Closed once bug resolution has been confirmed*

PHP ENGINEERING | QA FLOWCHART | V1.0 - 20120806

5 Live Outages (Post-launch)

5.1 Incident Classification Priority

If there is any confusion on how to properly classify a reported incident after site launch, then please see the table below for assistance:

Priority (P_x) Classification	External?	Description/Example (Pulled from ESOC)
P1	Yes	**CRITICAL** Has substantial revenue impact or significant impact on customer experience. MUST communicate to Business Operations for ALL P1 priorities. Call or stop by during business hours (M-F 8am-5pm CST) or non-business hours to "bizops911@searshc.com".
P2	Yes	**HIGH** Has some immediate impact or potential impact for large business if not quickly resolved. MUST communicate to Business Operations for ALL P2 priorities. Call or stop by during business hours (M-F 8am-5pm CST) or non-business hours to "bizops911@searshc.com".
P3	Yes/No	**MEDIUM** Requires attention over next several business days. Requests will be prioritized by business impact and time required.
P4	Yes/No	**LOW** May not have sufficient business impact
P5	Yes/No	**MINOR** Non-customer facing (nice-to-haves / to-do's)

6 Engineering On-Call (Communities)

6.1 Document Outage

It is mandatory that a Jira is created prior to any engineering troubleshooting has been completed. This is required for incident tracking, analysis, documentation, and accountability. The reporter, or reporters, must have completed the necessary prerequisite troubleshooting, with strong confidence that the incident is properly assigned to the correct team resource.

Information required within the Jira Incident Report should contain all (or most) of the following fields:

Time/Date of Outage:

Reporter:

Team:

Assignee (Currently scheduled on-call engineer, as outlined in the Sears PHP Team Wiki) – https://wiki.intra.sears.com/confluence/display/PHPTM/home

Environment (Production, QA, Integration):

Description of Incident (including steps to replicate, relevant login information

Relevant Login Information (if required)

Screenshots (if applicable)

*Priority *(i.e. P1, P2, …, P5)*:

6.2 On-Call Rotation

Engineers will be on a weekly on-call rotation starting on Monday and ending on Sunday to address any potential outages on the Communities site.

Scheduled on-call engineers must always be within a 30-minute range of their mobile workstation during their weekly on-call schedule. Engineers must acknowledge the incident within 30 minutes of page. On-call engineers are not authorized to begin any troubleshooting without the formal creation of a Jira issue. The Jira issue should be assigned to the Communities engineer currently scheduled on call. If this person cannot be reached, then the reporting team may contact the emergency contacts sequentially by priority (from Emergency Contact 1 to Emergency Contact 3, shown in section 7, *Emergency Contact Information*). The responsible engineer (or contact) will be required to acknowledge the incident through the Jira issue created by the reporter for this particular incident.

Phones will be passed from the previously scheduled on-call engineer to the newly scheduled engineer on the first Monday. If the previous engineer is unable to attend work on the Monday exchange, then the previously scheduled engineer will be responsible for scheduling means for exchange, or assume responsibility for the additional day(s) during possession of the on-call phone/pager.

This process, as outlined in this document, will take effect on the Communities project soft launch on Wednesday, August 22nd at 12:00 a.m. CST and will cycle through both the duration and maintenance of the project.

**Failure to meet this mandatory process during a reported incident shall be addressed by lead manager on a per-incident basis.*

7 Emergency Contact Information

In cases where the scheduled on-call engineer cannot be reached, or in cases of serious outages, the emergency contact list may be contacted for resolution in the following order:

---

1. Name: Sebastian Frohm Title: Release Manager

   Phone: 773-706-9362 Email: sfrohm@searshc.com

---

2. Name: Shafeeg Karadsheh Title: Delivery Manager

   Phone: 773-780-1725 Email: skarads@searshc.com

---

3. Name: Brendan Gualdoni Title: Director of Web Development

   Phone: 224-565-3078 Email: bguald0@searshc.com

---

8 Documenting an Outage / Business Communication

8.1 Internal Notifications (Communities Dev Group)

All outages should be reported internally within the Sears PHP team wiki site in a newly created “Outages” section. It will be important in determining site stability until uptime reports are regularly recorded.

8.2 Client-facing Communication

In the case of an outage, a formal report to the BU leads will be necessary in describing the outage, the duration, and the steps taken to address the problem. This communication is typically sent from management and will also include information on the necessary measures required for preventing such an occurrence in the future.

Here is an example of previous outage communication:

SearsOutlet.com – P1 Production Issue (on May 21, 2012)
Summary of Issue with Root Cause and Resolution Details

Issue:
The Sears Outlet site Search functionality stopped working between 8.40a – 9a this morning (May 21st, 2012) and the product List page was not displayed due to search issues. However the site home page was still up that time. Attempts were made to bring the search back up for about 15min without taking a site splash. It was taking time to get back the search and so Outlet business made a call to splash the site. The site was splashed around 9.30a until 10.25a due to inconsistent and poor user experience related to site search

JIRA Information:

Search and Product List Page Failing in Sears Outlet:

http://jira.intra.sears.com/jira/browse/ESOC-43238

Root Cause:
Outlet is implementing Spring MVC Framework to improve reusability, and build a thin web layer decoupled from business layer. The search functionality in the Outlet site requires a particular version of the Spring MVC –> 3.0 to be available from a configuration file. On May 21st, around 8.40a, when search was initiated, the application was referring to the external Open Source Spring MVC site and that external site was also not available that time. So the Outlet site Search failed and all subsequent search requests from other customers started failing.

Resolution:
Temporary (already in place): The Outlet Dev team made the changes to the schema to have the older Spring MVC – Ver2.5 which is already cached in the system. The site was un-splashed and brought back at 10.25a. The fix brought back the complete functionality of the site.

Long Term Permanent Fix:
Outlet Dev team will put a permanent fix so that the code will not do an explicit look up for a Spring MVC version at runtime in future. This will be put in place in the next release this week (5/23)

Additional Measures:
ESOC monitoring is already in place for home page. The outlet codebase also sends alters for bottom of the funnel – checkout failures, order failures and no new orders for a preset time. With learning from today's production issue, the Outlet Dev team is planning to add additional alerts in the next 3-4 weeks on key sections of the site (Like Search, or any other critical areas) which will caution the delivery team instantaneously if any of the critical functions would fail on the site in future. This will cut down the reaction time and fix issues like this in future.

9 GO TEAM!!!