# Description of Harmonia’s Usage and Implementation of Agile/Scrum

Our management approach begins by understanding the business case for the system to be designed or updated, to enumerate requirements (although the requirements may be a living document if features are added through Agile sprints), and to create user interfaces (UIs) with high usability. We outline our steps below.

*Use of Agile Scrum:* We apply a structured process of planning, execution, and documentation. We use the Agile Scrum software development methodology with two-week sprints. Two weeks affords our team enough time to develop features to completion while still allowing us to be responsive to changing priorities because we can reshuffle work quickly. Sprint duration can be customized to the client requirements for individual task orders. A best practice we developed is to review Acceptance Criteria for completed stories in Sprint tag-up meetings to remind developers of the story’s requirements and ensure that stories are complete before the Sprint Review meetings. Our well-defined and disciplined Agile Scrum software development methodology is documented on our internal Wiki for all developers to learn, and they are evaluated annually on their success in applying the methodology.

The work to be done in each sprint is defined as a set of user stories. Each user story consists of a name, description, and set of acceptance criteria. User story descriptions include who (benefitting actor), what (action to be performed), and why (reasoning for the action benefitting the actor). Acceptance criteria defines the story’s finished state and is agreed upon by the team and the product owner. Stories are assigned story points, a rough estimate of effort and risk required for the team to complete the story. Stories are then broken into tasks with associated estimates in hours. We maintain all Sprint information in the iceScrum sprint management tool (left of Figure 1), and view metrics such as burn down of story points (right of Figure 1).

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

Figure 1: Sprint Management Tool Views

We prefer to produce either a prototype of new features or a release quality product at the end of each sprint, which is presented to stakeholders for regular feedback and course correction if needed. Sprints also help teams limit time spent on each task to avoid getting stuck on a problem. This facilitates collaboration with the customer and internal course corrections to meet the overall product goals and timeline.

We work transparently with our stakeholders in our software development lifecycle (SDLC), and invite and encourage a representative from the client to serve as the product owner role in the Agile Scrum process. The benefit is a high degree of communication and assurance that everything our team does aligns with and adds the highest value to the client’s needs. The product owner is asked to interface with our team in a sprint planning meeting at the start of each two-week sprint to approve what user stories go into the upcoming sprint, and then interface again in a sprint review meeting at the end of the sprint to decide if work performed is acceptable.

*Architecture and Design:* In this phase the team focuses on the following tasks: analyze design objectives, use cases & requirements ([https://github.com/HarmoniaHoldings/medfinder/blob/master/documentation/Templates/Use Cases & Requirements - v1.1.dotx](https://github.com/HarmoniaHoldings/medfinder/blob/master/documentation/Templates/Use%20Cases%20%26%20Requirements%20-%20v1.1.dotx)), data models, identify industry standard design and architecture patterns as well as industry best practices, and create a solution architecture that complies with the organization’s enterprise architecture (e.g., complies with the service, data, and technical reference models, and follows the recommended technology stack). The solution architecture may be represented as Unified Modeling Language (UML) diagrams or equivalent formats (e.g., activity, sequence, class diagrams or equivalent formats). Next, design follows by evaluating programming frameworks that best meet the design objectives, maximize quality of the final application, and minimize labor to build and maintain the application. The architecture is then mapped to a software design, for example by definition of a set of interfaces between modules. The team creates a document describing the solution architecture and system design in accordance with templates that are part of our CMMI processes. The design undergoes peer review before development starts.

***Development:***Once we begin development work, our team’s focus is on the tasks listed below.

* Develop (for new requirements) or update (in accordance with change management process) software. Developers are responsible for developing unit tests with pass/fail acceptance criteria.
* Developers post their source code changes to the open source Review Board code review system and peer review each other’s code to catch as may issues as possible and ensure quality before formal testing starts. A senior team member much approved the changes before the code can be contributed to the project.
* Developers contribute their source code by committing their changes to Git, our Configuration Management (CM) system, in a development branch. When committing, developers comment what they changed, including metadata like the Review Board review number and the iceScrum story number.
* Developers utilize Jenkins, the Continuous Integration (CI) server, which automatically retrieves the latest code from the CM system and executes jobs whenever new commits are found. Jenkins jobs check out the latest revision of the code, build the code, run unit tests, deploy the application to a test server, run automated UI tests, and report back to developers via email if a job has any failures.
* The CI server is set up to also subject code to nightly automated scans to find Common Weakness Enumeration (CWE) items (<https://nvd.nist.gov>) and other weaknesses to build quality in before testing.
* Developers are trained in secure coding practices to avoid introducing CWE items into the code base, and code scans are launched automatically by Jenkins.
* We provide incremental software updates to the client, via continuous deployment when permitted by the client or otherwise by running deployment scripts, and we regularly review progress with the client to get their feedback and improve the software design.
* We proactively manage risk by identifying problems early and bringing them to management’s attention.

At the end of the Development phase all of the software requirements defined during the requirements gathering phase should be satisfied and the software should be ready for testing. A Requirements Traceability Matrix ([https://github.com/HarmoniaHoldings/medfinder/blob/master/documentation/Templates/Requirements Traceability Matrix - v1.1.xltx](https://github.com/HarmoniaHoldings/medfinder/blob/master/documentation/Templates/Requirements%20Traceability%20Matrix%20-%20v1.1.xltx)) ensures that all requirements map to implementations as described in the Component Architecture document ([https://github.com/HarmoniaHoldings/medfinder/blob/master/documentation/Templates/Component Architecture - v1.1.dotx](https://github.com/HarmoniaHoldings/medfinder/blob/master/documentation/Templates/Component%20Architecture%20-%20v1.1.dotx)) and then to functional and performance test cases for complete test coverages.

We discuss more detail on several of the points above in the paragraphs below.

***Automated Code Scanning to Detect Weaknesses and Bugs:***We stated earlier that we use code scanning to find CWE items. Our team currently supports DISA (contract HC1028-13-F-0515) in installing, running, and reporting for code scanning for vulnerabilities and weaknesses using various tools and interfacing with the National Institute of Science and Technology (NIST) National Vulnerability Database (NVD). A best practice we derived from our DISA work is to incorporate code scanning for vulnerabilities when a developer commits a source code change to a code repository. We automate the scan using our Continuous Integration server. Thus developers are made aware of CWE items from the NIST NVD on a daily basis, and will correct these long before updates ever enter testing or deployment to production. The benefit is better software assurance and more resistance to cyber-attack.

***Source Code Review Process:***To build quality into our code, our process includes code reviews. Prior to committing code to version control, our developers are required to get approval from a senior developer using a peer-review process. Comments and issues found (e.g., code readability, correctness questions, maintainability concerns) are written down and recorded with the open source Review Board tool. Following code review, the developer updates their code, and records how the issues were addressed in Review Board. The developer then commits the code to, as noted earlier, the Git CM repository. As part of the commit process, two hooks are executed. The first one verifies that a comment was included as part of the commit. It is critical to know what the developer’s thoughts were when they made a particular change. The second hook identifies the review request on Review Board for the given code and ensures that the developer received approval for the code. Once verified, the hook allows the code to be committed to the repository and the review request is marked as submitted in Review Board.

***Testing Process:*** Automated Software testing is performed on a continuous basis as a part of Agile development and our use of a Continuous Integration server. We also perform formal quality control (QC) testing before a major release, as described in the next paragraph, along with user acceptance testing (UAT). We use a test-driven development approach in order to validate the requirements and obtain user acceptance. Our deliverables to stakeholders are thoroughly tested and ready for production deployment.

A formal test plan ([https://github.com/HarmoniaHoldings/medfinder/tree/master/documentation/Templates/Test Plan - v1.1.dotx](https://github.com/HarmoniaHoldings/medfinder/tree/master/documentation/Templates/Test%20Plan%20-%20v1.1.dotx)) is created for major releases. Software testing for all application updates or new releases follows a rigorous methodology, with the following steps:

* Test cases and scenarios are built to mirror new functional requirements, and are used to validate compliance and perform regression, simulation, and load-balance testing as applicable.
* Automated tools and procedures are used to ensure code meets requirements and to maintain source control integrity so that working features do not break when future code updates are made.
* Perform additional quality validation by creation of synthetic database content and random sampling for data migration and technical re-engineering from legacy to new technology platforms.
* Reporting all bugs found in the Bugzilla bug reporting web application.
* Fix any reported bugs, and repeat testing; file new bugs if necessary.
* Pilot implementation, when applicable, is used to verify functionality in a subset of end users prior to a whole system upgrade.

We visualize (Figure 2) code coverage, or the percent of lines of code in each source code module that is exercised by an automated unit test, as an indicator of which parts of the source code require additional tests to ensure that nearly all lines of code are executed in testing. We do not want an end user to execute code for the first time in a production environment. It is not until most of this visualization is green that our team becomes confident in the quality of code delivered to the testing group for functional and integration testing.

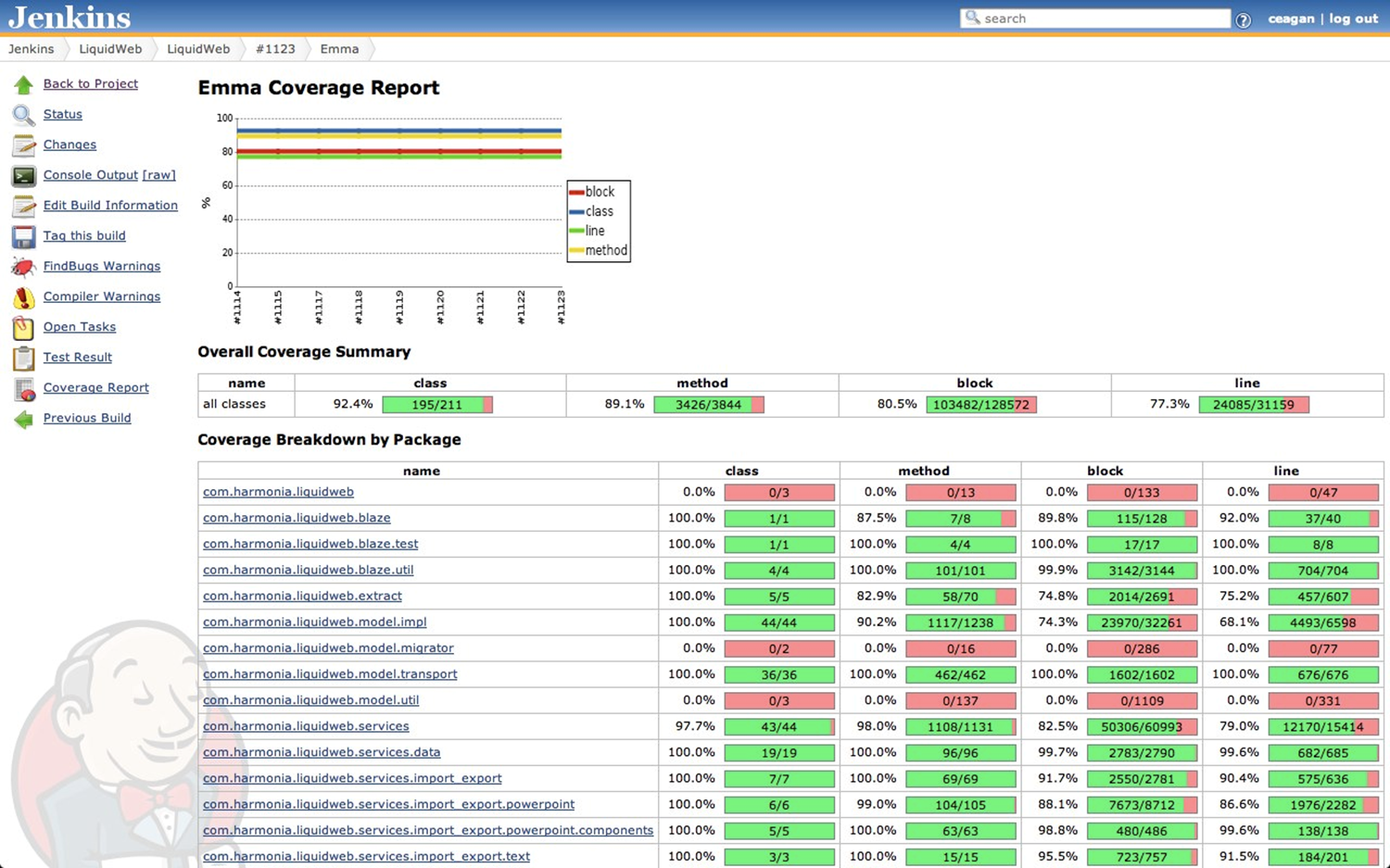
****

Figure 2: Browser Display of Unit Test Code Coverage; Red Shows Untested Code

***Continuous Delivery:*** We implement a process through our Continuous Integration (CI) and Configuration Management (CM) systems to support Continuous Delivery (CD) of our development products. Our Git CI system uses the Git Flow Workflow model for organization of our Git repositories which provides two main branches named master and develop. The develop branch contains code which is working towards the next release of the product. The master branch contains code from the most recent release of the product. Developers commit to the develop branch during their normal sprint work and the team merges that code into the master branch when a release is made.

Project teams create virtual machines to represent the customer’s production environment as closely as possible, matching operating system versions and simulating firewalls and network equipment. This environment is created in triplicate to create a test environment for formal testing by the QC team, a test environment built from the latest develop branch from Git, and a replica production environment built from the latest master branch from Git.

Jenkins, our CI system, automates building of deployment artifacts for each deployment environment from the proper branches in the Git repository. A successful build for the develop branch trigger automatic deployment to the virtual machine environment constructed for testing by developers. A successful build for the master branch triggers automatic deployment to the virtual machine environment constructed for release testing and sprint review demonstrations. The QC team is able to deploy the latest successful build of the develop branch to their test environment by manually triggering a Jenkins job for this purpose. This allows the QC team to fully test a revision of the develop branch without the code changing in the middle of them performing tests.

Our automated deployment system is robust and capable of supporting continuous delivery to full production sites after confirmation from the customer at sprint review meetings.

***Continuous Monitoring:*** To maintain high levels of quality and security we implement Continuous Monitoring by configuring scheduled and ad-hoc scan jobs for systems at various stages of development. Developers specify the types and depth of scans depending on the requirements of the project. Scans are then scheduled and executed. The scan results can be viewed through a web browser (Figure 3) or as PDF reports. The reports provide a risk ratings and details of vulnerabilities found, which feed into future development cycles allowing for us to continually improve the quality and security of our software.

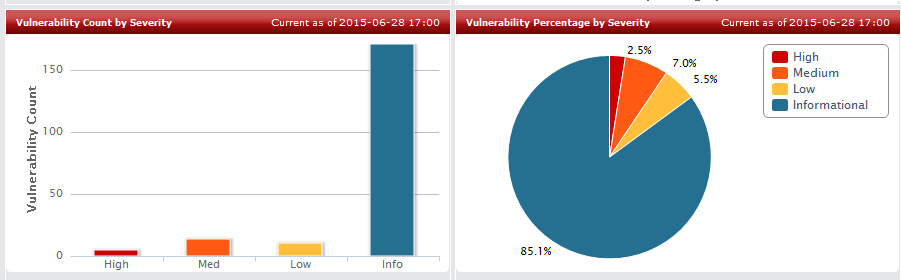


Figure 3: Continuous Monitoring Dashboard of Vulnerabilities

McAfee Vulnerability Manager (formally Foundstone), our tool of choice for Continuous Monitoring, provides web service API access to scan data in addition to a host of web based analysis and reporting tools. Developers can target systems at various levels with focus on vulnerabilities based on the latest directives from National Institute of Standards and Technology (NIST), Payment Card Industry (PCI), Open Web Application Security Project (OWASP) and others based on default or custom templates. Scan configurations are leveraged based upon the scope of the deployed system to ensure that focused information is available in order to streamline development and maintenance efforts.

Utilizing the robust capabilities of our Continuous Monitoring infrastructure we are able to effectively implement Continuous Monitoring to ensure that our developed products meet industry standards and are resilient to multiple attack vectors.