Aberrant Open-ISM™

SOFTWARE DEVELOPMENT LIFECYCLE

Property	Description
Document Version	1.0
Status	DRAFT
Last Update	2022-03-25
Document Owner	Risk Management
Next Scheduled Review	

PROPRIETARY

Document Approvals		
Approver Name	**Title**	**Date**
???	???	???

Revision History			
Version	**Date**	**Description of Changes**	**Revised by**
1.0	2022-03-25	DRAFT	Aberrant

TABLE OF CONTENTS

1 OVERVIEW

The Software Development Lifecycle (SDLC) is a framework that describes the activities performed at each stage of a software development project. Likewise, the Software Testing Life Cycle (STLC) defines testing as a process that is executed in a systematic and planned manner in conjunction with software development. Given the interrelated nature of both lifecycles it seemed practical to address both concepts within the scope of same document.

2 SCOPE

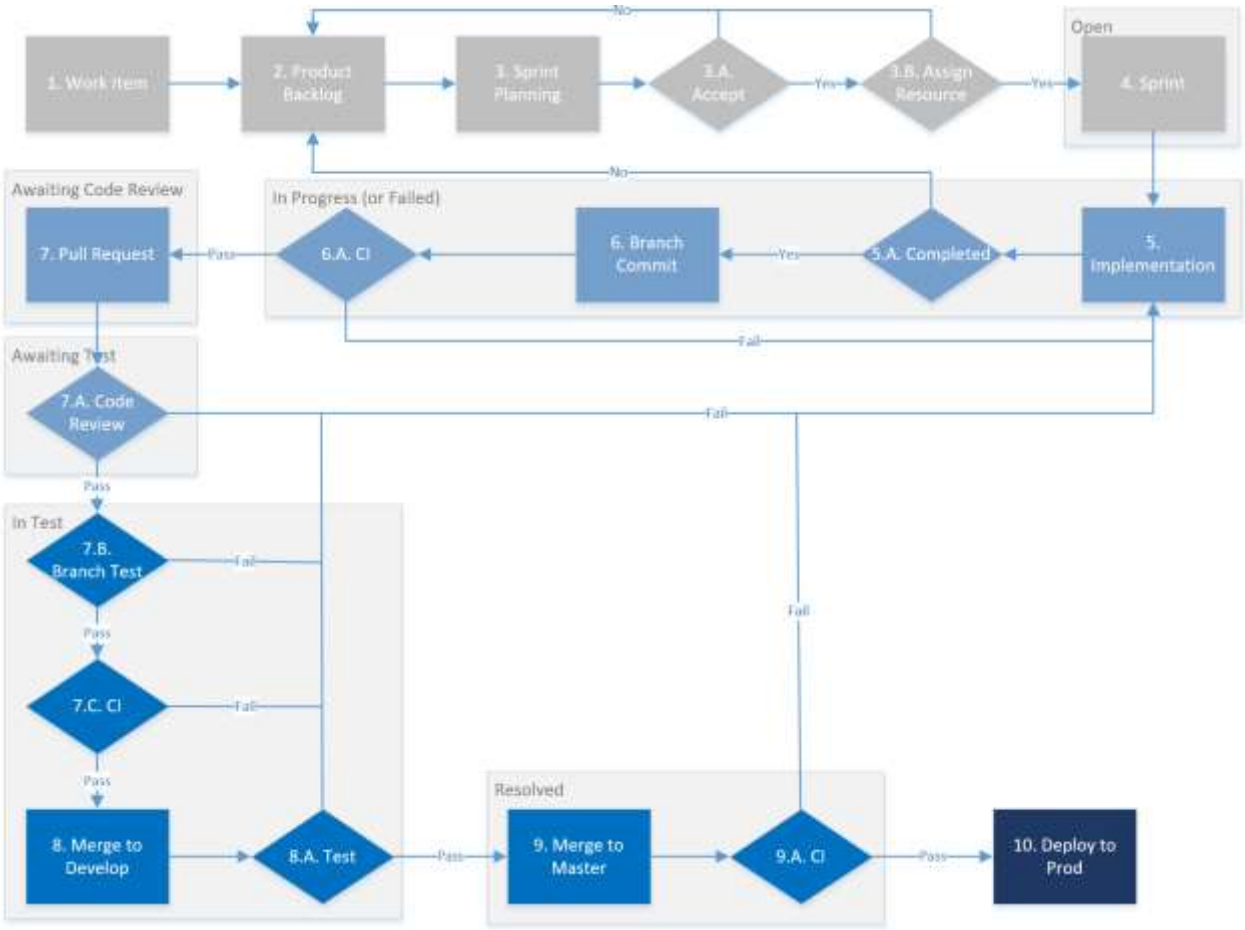
Changes the occur in code that are held in a source control repository that is administered by <<Company Name>> fall within the scope of this policy. Changes to batch files or scripts not held in source control are the purview of change control.

3 REQUIREMENTS GATHERING

When an 'Epic' is initially scoped functional, non-functional, and security requirements are evaluated and included in the relevant work items that are generated and stored in <<Company Name>>'s issue tracking system.

4 SDLC WORKFLOW

Figure 1: SDLC Work Flow



4.1 WORK FLOW DETAIL:

1. Work Item: A work item represents an item that can be inputted into our Product Backlog—<<Company Name>>'s Product Backlog is a web-based ticketing system. Work items are represented as virtual "tickets" and are categorized as either bugs or features. Work items have a title, description, case study, and scope. For scoping we use a Fibonacci based pointing system that is popular with the Agile methodology.
2. Product Backlog: A collection of Work Items. Work Items are periodically "Groomed" to ensure that the Backlog is an accurate representation of upcoming development work. Recently identified customer issues must be evaluated and triaged during this phase of the workflow.
3. Sprint Planning: A meeting where items from the Backlog are matched to business priorities.

A. Items are accepted or rejected from inclusion in the current sprint based on an evaluation from the team.

B. Items are assigned to available resources. In the event, resources are not available the item is rejected from the current sprint and returned to the backlog.

1. Sprint: A collection of work items that results from the Sprint Planning Process—a sprint represents two weeks' worth of work that will be 'delivered' at the conclusion of the sprint. Delivered items are demonstrated to the company at large during the 'show me' meeting prior to sprint planning. Assigned work items that exist in the context of a sprint are represented in the Kanban board as in an 'Open' state. Architectural guidance is provided when required to the ticket owner during this phase. Special consideration is given to security, performance and industry canonical standards.
2. Implementation: A work item selected for implementation by a developer transitions into the 'In Progress' state on the Kanban board. The developer performs development tasks with secure coding practices in mind. The developer branches from the "develop" branch and begins work. During implementation, the developer may annotate or modify the ticket representing the work item to assist the product manager or QA.

A. Items that are completed are committed and pushed to their branch in the source control system. If the ticket is not able to be completed by the developer because of a blocking issue or under scoping the ticket is moved to the backlog where is will be re-evaluated.

1. Branch Commit: The branch is submitted to the source control system. This triggers a branch build by the continuous integration (CI) system.

A. If the branch build compiles without errors and all the tests in the suite of regression tests pass the developer will create a pull request. If for any reason the branch fails CI the work item goes back to its implementation state.

1. Pull Request: The developer submits a pull request and assigns relevant reviewers for a code review. Reviewers are other members of the development team that participated in the sprint. The status of the work item is changed to "Awaiting Code Review" on the Kanban board.

A. The pull request is reviewed by one or more reviewers. Reviewers evaluate the pull request against existing security standards, scale considerations and industry best practices. Based on the code review the pull request is either passed or failed. If the code review is passed the ticket moves to "Awaiting Test" on the Kanban board. If the work item is failed the pull request is annotated and the work item is moved back to implementation to correct the issues.

B. Once the work item is accepted for testing the status of the work item changes to “In Test” on the Kanban board. The work item branch is tested by QA based on the case study in the work item ticket. If the test passes the ticket is merged to the “develop” branch. If the test fails the work item has its status updated to “Failed” on the Kanban board. The work item is then moved back to implementation.

C. When the branch is merged to “develop” this triggers a build by CI of the “develop” branch. The “develop” build should compile without errors and all of the tests in the suite of regression tests should pass. If for any reason the branch fails CI the offending work item goes back to its implementation state and the work item state is changed to “Failed” on the Kanban board.

1. Merge to Develop: Work items that pass branch testing are collected in the develop branch. At some point near the end of the sprint QA performs smoke testing—and security testing if needed.

   A. When QA is satisfied that everything is performing in-line with the work item case studies the branch is merged to master—the status of each work item is moved to “Resolved” on the Kanban board. If a particular work item fails it is sent back to implementation.

1. Merge to Master: Work items are merged to the “master” branch.

   A. When the branch is merged to “master” this triggers a build by CI of the “master” branch. The “master” build should compile without errors and all of the tests in the suite of regression tests should pass. If for any reason the branch fails CI an investigation is triggered and conflicts are resolved.

1. Deploy to Prod: A change control ticket is submitted with a list of all the tickets included in the build. The production site is smoke tested by QA. After deployment, the site undergoes dynamic scans, internal and external vulnerability scans, and annual penetration testing.

4.2 ADDITIONAL WORKFLOW

4.2.1 GROOMING

Grooming occurs outside the work flow. Prior to step 3. The Product Manager reviews the backlog with developers to ensure that the requisite work items have been added to the backlog for the next sprint and that the information in the tickets is updated and accurate. In the event the Product Manager identifies work items that are obsolete the team is consulted and the work items are switch to a “Closed” state in the ticketing system.

4.2.2 HOT FIXES

The process works as it does normally until step 7.C. The pull request is merged directly to master and not develop. Once CI passes on master, master is then merged to develop.

4.3 SYSTEMS OF RECORD

4.3.1 BACKLOG

<<Company Name>> uses <<Name of Issue Tracking System>> as it's system of record for the SDLC backlog. <<Name of Issue Tracking System>> provides visibility into work in progress and provides an immutable log of change that fall within the scope of the SDLC.

4.3.2 SOURCE CODE

<<Company Name>> uses <<Name of Source Code System>> for source code repository. <<Name of Source Code System>> is also used to document code reviews.

4.3.3 CI PIPELINE

<<Company Name>> uses <<Name of CI System>> for Continuous Integration.

4.4 SEPARATION OF ENVIRONMENTS

<<Company Name>> maintains strict isolation of the supported application environments in-line with canonical software engineering standards. <<Company Name>> supports three types of environments:

1. Development environments: Development environments are machine scoped, contain 'LOCAL' configuration, are branch based, and are hosted on dedicated hardware—e.g. the developer's laptop. No sensitive customer information is used in the development environment.
2. Virtual Environments: Virtual environments are programmatically generated on a secure cloud based virtual operating system built on a development branch. They are machine scoped and contain configuration used for testing. Virtual Environments are primarily used for testing and demonstrations—they contain no sensitive customer data. Once a virtual environment has been used it is recycled.
3. Production / DR Environments: Environments used for 'PROD' or 'DR' are deployed on both virtual and dedicated systems. Production/DR Environments are domain scoped and utilize sensitive customer data.

4.5 SECURE DEVELOPMENT ENVIRONMENT

Developer environments exist on dedicated machines that are self-contained and machine scoped. Access to the developer's machine requires authentication against our domain controller. Developer machines utilize full-disk encryption to ensure that development data is secure in the even the machine is stolen or misplaced. All data from PROD can be pulled to local and is totally anonymous; development and testing environments redact all sensitive data or use de-identified data. No sensitive customer information is present on development machines due to this "scrub."

The "scrub" job is to be reviewed no less than annually to ensure that customer information is being protected sufficiently. Due to how the data is protected locally on developer workstations in addition to the low risk associated with the scrubbed database, there is not a duration limit on how long a version of a scrubbed database can remain on a protected system.

4.6 INFORMATION SECURITY IN PROJECT MANAGEMENT

Items stored in the Product Backlog contain <<Company Name>> intellectual property (IP). As a result, additional security requirements have been put in place to safeguard <<Company Name>> IP.

4.6.1 PRODUCT BACKLOG SECURITY

<<Company Name>> uses a 3rd party web application product backlog that is internally hosted in our production environment. The product backlog is only available to users who have access to the <<Company Name>> network via our VPN. Data is encrypted in transport and at rest. Authentication utilizes two-factor authentication.

4.6.2 PRODUCT BACKLOG ACCESS

Access to the Product Backlog is controlled by <<Company Name>>'s Access Control Policy. The product backlog uses a role-based access control model (RBAC) to allow authorized users access to application features. Changes to user permissions are submitted via change control.

4.7 SECURE CODING PRINCIPLES

<<Company Name>> adheres to OWASP's Secure Coding Practices Guidelines and all developers are required to take secure coding training annually through our internal learning management system. Nightly, static scans of source code are performed against the development branch using OWASP's secure coding principles to ensure that code staged for release to our production environment adheres to secure coding principles. Architects review identified vulnerabilities and mitigated items based on their priority.

4.7.1 TECHNICAL REVIEWS

Technical reviews are conducted on all but the most trivial of pull requests. Software engineers have some discretion with this, however, the vast majority of pull requests are code reviewed by at least one developer. A log of all pull requests with their associated technical reviews is stored in <<Company Name>>'s source control system. Logs of pull requests are retained for at least one year or more.

4.7.2 CODE SIGNING

Restrictions on changes to software packages and artifacts are predicated on job function. Code signing should:

1. Ensure artifacts and code are signed by a trusted certificate authority (CA).
2. Use technical controls, such as digital signatures and version control, to ensure that only authorized artifacts are deployable.

4.8 SECURE DEVELOPMENT

Secure coding practices are incorporated into all life cycle stages of the application development process. The following minimum set of secure coding practices and architectural best practices are implemented when developing and deploying covered applications:

1. Requirements: Namely, functional and non-functional application requirements.
2. Architecture and Design: This incorporates low-level components such as algorithm design and high-level architecture design.
    0. Use only standardized, currently accepted, and extensively reviewed encryption algorithms.
    1. Leverage vetted modules or services for application security components, such as identity management, encryption, auditing, and logging.
3. Implementation: perform technical review consistently on pull request.
4. Code signing: To prevent injection of malware in deployed code.
5. Testing: Security testing is performed using both automated and manual methods.
6. Deployment: A change control ticket is submitted with a list of tickets included in the build.
7. Maintenance: Incorporates both enhancements and corrective fixes.
8. Static scans: To ensure OWASP design principles have not been violated.

4.9 SYSTEM SECURITY TESTING

<<Company Name>> does the following system based security related testing. Vulnerabilities that require remediation should adhere to Corrective Actions guidance in the Enterprise Risk Policy.

4.9.1 APPLICATION PENETRATION TESTING

The organization should perform both manual authenticated and unauthenticated penetration testing at least annually.

4.9.2 DYNAMIC APPLICATION SECURITY TESTING (DAST)

After deployment application security tests (DAST) should be performed.

4.9.3 INTERNAL AND EXTERNAL VULNERABILITY SCANS

At least quarterly review of internal and external vulnerability scans.

4.9.4 STATIC APPLICATION SECURITY TESTING (SAST)

Nightly static application security tests (SAST) of source code and third-party libraries to identify security related issues utilizing a SAST platform. Issues identified via SAST should be addressed prior to deployment.

4.9.5 SMOKE TESTING / USER ACCEPTANCE TESTING (UAT)

Smoke testing / User Acceptance Testing.

4.9.6 REGRESSION TESTING

Regression tests via continuous integration.

4.10 EXTERNALLY REPORTED SOFTWARE VULNERABILITIES

<<Company Name>> uses a web-based form that allows external users to report software issue or vulnerability. The form incorporates a 'captcha' that prevents a 'bot' from proliferating fake issues into the <<Issue Tracking System>>. When the user submits a software issue the form invokes an API that creates a ticket in the backlog and provides a ticket number to the user. The user is also provided with an email address that allows them to check on the status of their ticket.

Recently issues that have been identified by customers must be reviewed and triage during backlog grooming as part of the SDLC workflow process. Issues that identify process gaps that impact security, or security vulnerabilities may warrant inclusion in the Issue Queue as a corrective action. This should be evaluated by the development team during sprint planning.

5 OUTSOURCING

In some instances, it's necessary to work with software engineers outside the organization. When working with an outsourced engineering firm you must follow guidance from the "Service Provider Management Policy."

5.1 SOURCE CODE THAT IS MERGED

The SDLC accounts for source code that is developed externally and eventually merged into the company's repository. Source code in this case is subject to all the controls documented in this policy. Source code that is merged must also undergo UAT, review from a solution architect, and final approval from the product owner prior to being deployed into the production environment.

5.2 APPLICATIONS MAINTAINED BY A THIRD-PARTY

Applications maintained by a third-party are outside the scope of this document, however, if a third-party is hosting software on-behalf of the company it's incumbent on the company to adhere to "Service Provider Management Policy." Any source code or applications developed by third parties owned by the company fall within the purview of the CTO and the CISO regardless of where it is hosted. It's imperative that the CTO is included as a stakeholder in identifying non-functional requirements prior to signing any contract with a third-party application developer. It's important to note that third-party applications or services hosted on external systems may interoperate with company systems or network infrastructure in ways that impact service delivery, or impact the customer experience.

6 SYSTEM ACCEPTANCE

6.1 DEPLOYMENT PRE-FLIGHT CHECKLIST

The pre-flight checklist is performed after the build is merged to the master branch and ready for deployment. The following steps must be performed and attested to by the product owner prior to generating a change control request to deploy to production.

1. SAST has been performed and all issues with a severity rating above medium have been addressed and documented in the SAST tool.
2. All functional related to features or functionality have been tested. In the event UAT was required it was performed and is documented.

6.2 DEPLOYMENT

Deployment occurs after the pre-flight checklist is completed and the product owner has attested that the checklist was completed.

6.2.1 DEPLOYMENT LOG

A deployment log is generated and made available to customers via a web-based report that is available in the application at the conclusion of every software release. The deployment log includes the ticket numbers and the title of the issues from the backlog that were included in the release.

6.3 DEPLOYMENT POST-FLIGHT CHECKLIST

Post deployment an SRE, or a security analyst, should perform a post-flight checklist to confirm that the website is operational and no regressions have been introduced. The post-flight checklist includes:

1. A smoke test
2. DAST, all issues with a severity rating above medium should be addressed and documented in the DAST tool.

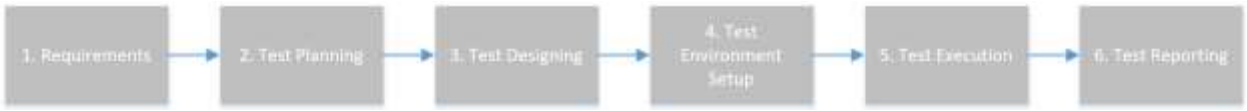
7 SOFTWARE TESTING

7.1 TEST DATA

Testing environments redact all sensitive data or use de-identified data in an environment that is segregated from the production environment. No sensitive customer information is present on in the test environment.

7.2 SOFTWARE TESTING LIFECYCLE (STLC)

Figure 2: STLC Work Flow



7.2.1 STLC WORKFLOW

1. Requirements: Discovery, analysis, and review of functional and non-functional requirements as they relate to testing. Security considerations are also reviewed.
2. Test Planning: This generally involves creating a "case study" in the work item's ticket. This responsibility generally falls to the product owner—this should also incorporate security test plans.
3. Test Designing:

   A. In the case of User Acceptance Testing (UAT) this would involve the creation of formalized test cases. The product owner is responsible for generating UAT test cases based on requirements gather from the business owner. QA and the Product Owner coordinate to ensure that test cases conform to business requirements. UAT test cases are stored in an auditable electronic form.

   B. For Smoke Testing we don't produce test cases—this is for expedience. The tradeoff is formalized documentation at the expense of throughput. We consider manual functional testing as a stop-gap for more formalized automated testing: unit tests, integration tests and block tests. In the case of automated testing the code is the documentation.

1. Test Environment Setup: Creation of an environment and context that emulates the production environment.

A. For branch testing as it relates to the SDLC each branch gets its own environment—the generation of branch test environments is completely automated and maintained by DevOps Engineer.

B. Environments for UAT may require additional manual modification. QA works with development to ensure that the manual environment is a close proxy of production. QA is responsible for the setup and maintenance of manual testing environments.

1. Test Execution: Invocation of tests in a test environment. The tester is responsible for documenting tests results.
2. Test Reporting: The result(s) of the test. Results are stored in our ticketing system which we use as our defect tracking system.

8 THREAT MODELING

Application architects are encouraged to map out the application, architecture, and infrastructure in a structured way to understand its weaknesses. A formal threat modeling methodology can be deployed if it's helpful. The specifics of this process are the purview of the CTO.

9 QUALITY ASSURANCE PLAN

9.1 DEFECT TRACKING

Tracking work items in the backlog is primarily the responsibility of the product owner and the developers. When a bug has been added to a sprint the responsibility for tracking the issue falls to QA—QA relinquishes responsibility when the work item moves to a "resolved" state on the Kanban board.

9.2 TEST AUTOMATION

<<Company Name>> invests heavily in test automation primarily because test automation is the most effective way to detect regression issues in the codebase. Developers are responsible for designing, creating and maintaining code used for regression tests. Regression tests are run at each and every check-in via continuous integration (CI). Exception reports from event logs are periodically correlated with code coverage reports to ensure that adequate code coverage exists for the codebase.

9.3 APPLICATION TESTING

Quality Assurance (QA) is responsible functional and non-functional testing for each work item. Product owners and developers are responsible for creating test cases. Developers are responsible for performing code reviews of prior to check-in.

In some cases, where the ROI justifies the investment of resources, QA will perform User Acceptance Testing (UAT). UAT testing is functional testing with formal test cases. At the conclusion of each test case the business owner is required to sign-off on the test.

9.4 SECURITY TESTING

9.4.1 STATIC APPLICATION SECURITY TESTING (SAST)

SAST is performed nightly on code to ensure that best practices are applied uniformly to vulnerable areas of the codebase. Vulnerabilities when they are discovered are triaged, documented, and mitigated. If an issue warrants a code change it is added to the backlog. Nightly execution of SAST acts as a control to ensure that issues are mitigated in a timely manner.

9.4.2 DYNAMIC APPLICATION SECURITY TESTING (DAST)

DAST is performed, and reviewed, on a quarterly basis. If an issue is detected it is triaged and mitigated or added to the backlog.

9.4.3 AUTHENTICATED APPLICATION PENETRATION TESTING

Authenticated Application Penetration Testing is performed by a third party and is done on a scheduled basis at least annually.

9.5 PERFORMANCE TESTING

Testing of the applications overall performance is periodically performed by members of the development team. Testing may include performance profiling, high-level load testing, or a review of web analytics.

9.6 LOAD TESTING

Testing is performed to validate system requirements and to ascertain the performance limits of the application.

- On at least an annual basis load testing of the production system should be performed.

- The output of load testing should serve as the foundation for the formal capacity management assessment.

9.7 PRODUCTION TESTING

The product owner is responsible for coordinating alpha and beta testing.

9.8 VERIFICATION

QA is responsible for evaluating and ensuring that the materials used for testing meet the specific requirements of the particular phase of the STLC—e.g. that specification are up to date and case studies accurately reflect business requirements.

9.9 VALIDATION

QA is responsible for evaluating the final product to check whether the software meets the business needs defined in the business requirements.

10 CONTACT INFORMATION

Name of Security Program Owner

Title of Security Program Manager

Phone Number

Email

11 DOCUMENT RACI

Responsible	Assigned to do the work	Security Program Manager
Accountable	Final decision, ultimately answerable	ISM Governance Committee
Consulted	Consulted BEFORE an action or decision is taken (proactive)	Executive Management
Informed	Informed AFTER a decision or action has been taken (reactive)	Named Participants in this document Other parties affected by the change

12 LICENSE INFORMATION

APPENDIX A: GLOSSARY OF TERMS

Alpha Testing: Testing with internal users or development partners.

Application Penetration Testing: A simulated attack against the application executed by a third party. <<Company Name>> performs Application Penetration Tests at least annually.

Beta Testing: Testing directly with customers. Users may be selected using stochastic methods that limit the audience to a statistically meaningful number.

Block Testing: Integration tests that incorporates the ability to dynamically inject test data. Block tests are idempotent.

Business Owner: An individual who defines functional requirements.

Code Coverage: The percentage of test automation coverage for each assembly or module in the codebase.

Code Reviews: Developers review code for every pull request. In the event an issue is identified the person checking in is required to correct the issue and submit a new pull request.

Continuous Integration (CI): Triggered when a check-in occurs on a branch. CI ensures that the code compiles and that all regression tests pass prior to a branch being merged.

Defect Tracking: Relates to the detection and accounting of defects in the production system. Defect tracking can be accomplished in different ways:

- Forensic analysis of logs provides insight into the location, quantity and severity of exceptions thrown by the system.
- The rate of bugs reported to the ticketing system within any given increment of time.

Developer: A person who implements application features or fixes bugs. Developers also create and maintain test automation.

DevOps Engineer: The person (or persons) who maintain continuous integration.

Dynamic Application Security Testing (DAST): Black box security testing that tests the application from outside the firewall and performs attacks against the system like what an actual attacker would do. DAST is run at least quarterly. <<Company Name>> uses <<Name of DAST system>> for DAST.

Epic: An organization group of Work Items in the Product Backlog.

Functional Testing: Testing business cases and permutations.
Integration Tests: A code based test that tests interdependencies. Integration test are idempotent.

Load Testing: Simulating production traffic to ascertain the performance limits of an application.

Non-Function Testing: A requirement that must be present for the application to be considered functional. Non-functional testing incorporates performance, load and scalability testing.

Performance Profiling: Performing a trace of the application at runtime to evaluate application performance.

Product Backlog: A repository of work items.

Product Manager: The individual who defines non-functional requirements.

Quality Assurance (QA): The person who ensures that the product meets functional and non-functional specifications.

Regression Testing: Incorporates automated testing from previous versions of the codebase: e.g. unit testing, integration testing and block testing. Regression tests ensure that the codebase continuous to respect functional rules after a bug is fixed or a feature is added.

Security Requirements: <<Company Name>> generates security requirements using the OWASP Top 10.

Smoke Testing: A combination of functional and non-functional testing.

Source-Code Tracking: The ability to view a history of all changes to a file and to associate those changes to a specific user.

Static Application Security Testing (SAST): SAST is used to validate that the 'develop' branch of the codebase is compliant with the latest OWASP application security guidelines. SAST is run nightly.

Tester: A person who plans and executes the tests.

Unit Testing: A code based test that tests a unit of work. A unit test has no dependencies that aren't mocked.

User Acceptance Testing (UAT): Functional testing with test cases. At the conclusion of the test the business owner signs-off that the test succeeded.

Web Analytics: A platform that tracks application performance and quantifies web traffic based on use. Web analytics utilize meta data from traffic to provide insight into users.

Work Items: A digital document, or collection of digital documents, that describes an application feature or bug. Work Items are organized into Epics.