Software Maintenance Plan (SMP)

Project – SyncGallery +

Revision History

Date	Revision	Description	Author
15-Feb-12	1.0	Document Created	W. Ghorishi

Acronyms

Acronym	Definition
SMP	Software Maintenance Plan
SCR	Software Change Request
SDLC	Software Development Life Cycle
SRS	Software Requirements Specification
SQA	Software Quality Assurance

Table of Contents

1 Purpose

The purpose of the Software Maintenance Plan (SMP) is to document the process by which post-release defects and feature requests are handled. Highlighting the roles and responsibilities of the team during the maintenance phase, the document will also address how each Software Change Request (SCR), henceforth referring to changes resulting from either a defect or enhancement request will evolve through various stages of the Software Development Life Cycle (SDLC), for the syncGallery+ project.

2 Maintenance Process Overview

The following diagram depicts the high level process that will be applied to handling Software Change Requests, SCRs, triggered in a post-release maintenance phase.

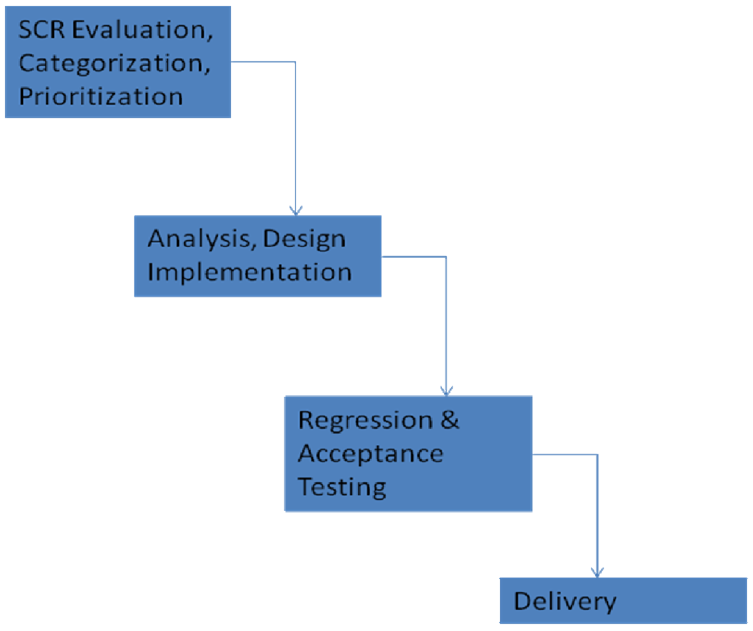
SCR Evaluation, Categorization, Prioritization
Analysis, Design Implementation
Regression & Acceptance Testing
Delivery

Fig 1: Maintenance Process Overview

While certain tasks need to be conducted in a sequential order, the overlap between each of these phases is to highlight the fact that maintenance phase of the SyncGallery+ project will also adopt the agile development model, where the underlying focus will be on concurrency of tasks and upfront testing of changes.

3 Maintenance Process

The following is a detailed description of how a SCR will trigger various workflows in various phases of the SDLC:

3.1 SCR Evaluation, Categorization & Prioritization, Scheduling

3.1.1 Evaluation

Often times, a customer's reporting of a defect may not be correct, accurate or descriptive enough. This may be due to deficient understanding of the software or inability to provide the relevant details required by the software development team to address their change request. Therefore it is critical for each change requested by the customer to be evaluated, documented and tracked. This process will rely on the change tracking tool, TRAC used during the initial development phase. Regardless of the validity of the change request, the request must be clearly documented and captured in TRAC. All relevant information associated with the SCR will then continue to be documented in the associated ticket in TRAC, represented through a unique ticket ID. The rejection, progress and/or completion status of the change request will be driven through updating of the ticket in TRAC.

3.1.2 Categorization

As described earlier, SCRs maybe triggered due to a defect or an enhancement request. The following describes the categorization process for both defect and enhancement driven change requests.

3.1.2.1 Defects

The following is a list of possible failure scenarios that can trigger a defect-based SCR:

- System Failure: Software hangs and/or crashes rendering the application inoperative
- Functional Failure: Functionality misbehavior where atomic functionality is not performing as required (e.g. cropping)
- Use-case Failure: Use-case misbehavior where sequence of operations are not performing as required (e.g. uploading captured images to Dropbox)
- Non-functional Requirements Failure: Inadequacies in performance and maintainability as specified in non-functional requirements. Other types of non-functional failures may include assumptions that may not always hold true in the application and hence may introduce undue risk to the execution of the application

The categorization process will require documentation and description of the change request to a point where it can be classified to one of these categories. Once categorized, it will be further classified in Trac using the relevant defect severities.

3.1.2.2 Enhancements

The following is a list of possible scenarios for an enhancement driven SCR:

- Requirements Change: The assumptions and requirements that the application was originally developed against have changed or are no longer valid
- Additional Functionality: The customer would like to build upon the requirements originally provided
- Non-functional requirements enhancements: Raising the bar on performance, maintainability, risk-tolerance and other non-functional requirements.

3.1.3 Prioritization

From customer's perspective, both defect and enhancement driven SCRs are of high priority and need to be delivered with the highest urgency. Reality of a limited resource pool that can be assigned to accepted SCRs means that all effort should be prioritized. The following descending priority scheme will be used as a high-level guideline and the filtered results will be shared and discussed with the customer:

- Priority 0: System Failures
- Priority1: Functional Failures
- Priority2: Requirements Change
- Priority3: Use-case Failures
- Priority4: Non-functional Requirements Failure
- Priority5: Additional Functionality
- Priority6: Non-functional requirements enhancements

Of course, input from the customer as well as availability of resources with the required expertise will ultimately affect the actual priority of individual SCRs.

3.1.4 Scoping & Scheduling

Once an SCR is accepted during the evaluation phase, categorized and prioritized, it will then be scoped with respect to the magnitude/size of the effort involved. Based on the expertise required and availability of resources, it will then be scheduled into delivery queue and a target date for completion of the SCR will be set in Trac and communicated to the customer.

3.2 Analysis, Design & Implementation

3.2.1 Analysis

The idea behind conducting an analysis on approved SCRs is to understand how it will impact the ecosystem within the application, prior to proceeding with further resource investments. This will include feasibility of the requested change, its overall value-add to the application as well as new risks and exposures that will be triggered through the SCR. Short of major discoveries rendering the approved SCR non-feasible, the engineer assigned to this phase will also be responsible for updating the Software Requirements Specification (SRS).

3.2.2 Design

Upon completion of the analysis, the engineer assigned to a SCR will be developing a list of modules that are impacted, updating the design documentation and other related artifacts, re-publishing the public APIs that have changed and documenting the associated TRAC ticket with any design-level decisions, risks/constraints that may impact the viability of the request change.

3.2.3 Implementation

In addition to making the required changes to code, the engineer assigned will be integrating the changes to the head branch (main baseline), developing/updating the relevant unit test cases. Prior to submitting the code to SVN for further SQA testing, the developer is responsible for executing the relevant unit and sanity test cases and ensuring they all pass.

3.3 Regression & Acceptance Testing

3.3.1 Regression Testing

The first rule to making any changes to a software application that has already been released is to ensure that the changes do not negatively impact the quality of the existing software. Known as regression testing, the SQA (Software Quality Assurance) team will execute all functional, integration and system test cases to assert the quality and integrity of the application have been maintained. Reported regression failures will prevent the release and delivery of the SCR to the customer.

3.3.2 Acceptance Testing

SQA team is also responsible for developing and executing a set of functional, integration and system level test cases (as applicable and based on the nature of the SCR) to validate that the defect resolved or enhancement made is inline with the detailed requirements defined for the SCR, during the analysis phase. The test cases developed will be done under the context of the traceability matrix used to assess test coverage. Also, following the formal adoption of the SCR, the relevant test cases will be added to the regression suite.

3.4 Delivery

3.4.1 Software Configuration Management

Regardless of the type of release issued to the customer, all software must be under configuration management in SVN, base-lined and labeled prior to release. The following lists a number of release vehicles that can be used to issue an SCR to the customer:

- Engineering release: Issued to the customer with the intent of allowing them to assess the new behavior and/or implemented functionality, early on. Engineering releases may be delivered after significant unit and sanity level testing but without having undergone full regression and acceptance testing
- Patch releases: Patch releases are formally certified by SQA from an acceptance testing/sanity testing perspective but do not necessarily guarantee that the newly added SCR has maintained the quality/integrity of the original application (due to not having completed a full regression test cycle)
- Formal releases: These releases are fully certified by SQA and reflect the highest quality configuration of the software, including the latest defect resolutions and enhancements base-lined

3.4.2 Release Artifacts

All formal and patch releases will be delivered to the customer as an executable (unless source code release is negotiated and approved by the SyncGallery+ project team), along with the following artifacts:

- Release notes capturing the baseline software features, recent enhancements and defect resolutions
- User's guide
- Training/demonstration material

4 Resources & Responsibilities

To ensure continuity and excellence in support, all members of the original team that were involved in developing the SyncGallery+ application will continue to be part of the maintenance team for this product. Furthermore, the roles assigned to each team member will leverage their experience and contributions during the development phase. Below are the high-level responsibility assignments:

Responsibilities	Resources
SCR Evaluation, Categorization & Prioritization	Project Manager: Will G.
Analysis, Design & Implementation	Developers: Kobe S., Daniel L., Jacky T.
Regression, System & Acceptance Testing	Quality Assurance: Hitha D., Sarah Y.
Delivery	Project Team: Will G., Kobe S., Daniel L., Jacky T., Hitha D., Sarah Y.