Thoughts for Developing Sofware Quality Procedures

Contents

1. Overview

Every company, every software development team works somewhat differently. Each distinct product category involves its own set of development tasks. A “one size fits all” approach to software quality procedures therefore is not useful; software SOPs instead need to be tailored to the team and tasks while fulfilling regulatory expectations.

The questions and suggestions below were developed to help teams generate their own software SOPs. They outline many key topics in a typical set of software procedures (“XYZ” is your company’s name):

- Software Quality Planning
- Software Development Procedure
- Software Requirements and Traceability
- Software and Database Design
- Software Coding Standards
- Code Review
- Issue Tracking
- Software Configuration Management
- Software Testing and Validation
- Software Release and Deployment

Thoughts for each document are organized according to a typical SOP outline, but this outline is not mandatory.

Similarly, some companies combine or divide SOP topics differently.

If you would like additional help creating or revising your software quality procedures, or with developing validation documents, don’t hesitate to contact ShoeBar Associates!

2. Software Quality Planning

2.1. General

In planning a development project, how and where do you set out:

- quality objectives and requirements,
- processes to follow, documents to establish, and resources needed,
- required verification / validation / monitoring as appropriate, and acceptance criteria,
- records to be created,
- risk assessments and risk mitigation plans?

2.2. Relevant Standards

Consider including, for each relevant FDA guidance or international document, a table which lists the elements (required or recommended) in the reference, and where that information is covered in your project documents.

2.3. Responsibilities

Typically there are specific planning responsibilities for engineering, quality assurance, product management/marketing, regulatory affairs where appropriate, operations, purchasing, and R&D.

2.4. Definitions

Good terms for definitions:

- Project
- Procedure
- Quality Planning
- Quality System
- One reference also defines "Significantly Modified Product", which requires a specific quality plan

2.5. Project Initiation

- How is a project requested / initiated / approved?
- When does quality planning begin? (This is usually the transition from "RESEARCH" or "investigation" or "feasibility" to "Development")

2.6. Quality Plan Contents

What considerations must be addressed in a quality plan? Typical parts might include:

- Introduction and scope (general description of the project)
- Project Roles (who has which team role, i.e. Project manager, development, technical writer, QA, user or subject matter expert, etc.)
- Project plan (task breakdown, timeline, milestones, dependencies, equipment/resources needed)
- Standards Assessment (list of relevant regulatory documents, and for each, a table of requirements listed there and your corresponding project documents where each is addressed)
- Requirements / Scoping Documents (i.e. reference to wherever system requirements are set out)
- Project Deliverables (software? data? documentation? configured study?)
- Development Practices (each as applicable; may reference existing procedure or describe project-specific method: development method, code configuration management, problem report handling, code reviews, programming standards, backups/archives)
- Computer system environment (development, test, final environments; security needs)
- Reviews (what will be reviewed and how review will be documented/approved - reviews of: requirements or spec, quality plan, system description, design specs, critical code, test plan)
- Testing (high-level description)
- Installation/Implementation (as appropriate)
- Completion Criteria (checklist for "when is the project done?")

2.7. Other Quality Concerns

- What are the standard project management concerns: nature of the project, deliverable/s, personnel, tasks, schedule, equipment/resources?
- How, and how frequently, will the hazards in the product be evaluated or re-evaluated? What is the scale for deciding to accept, document, mitigate?
- Are there ever instances where a customer / client organization's procedures will be followed, rather than XYZ's?

3. Software Development

3.1. Purpose

This statement needs to be terse but extremely well thought-out. What purpose is your software development method intended to serve? In most medical device companies, the ***software*** development method is all about who they are and what value they offer the marketplace. Typically this states "The software development method employed at XYZ is intended to achieve . . . ", with a list of key goals (such as maximum flexibility, maximum end-user input, high level of quality, complete but efficient documentation, auditable process control).

3.2. Scope

Is there any software development activity you carry out which would be exempt from what you outline in this procedure? (For example, do you ever act as a subcontractor for some other organization which takes all responsibility for the design and testing activities?)
If not, simply state that the document applies to all software development carried out at XYZ where the resulting software will be used for therapeutic or diagnostic (i.e. not academic research or prototype) purposes.

3.3. Responsibility

You could take the approach of listing the typical tasks expected of each of the key players:

- Project Manager
- Development Manager, if a separate person
- Developers
- Quality Assurance
- Technical Writer

Another approach would be to create a table listing each of the documents you generate in the course of development, note the key players on the other axis, and use the table to show who is responsible for writing, reviewing, and approving each of those documents.

3.4. References

- Your Quality Manual
- FDA General Principles of Software Validation
- FDA Guidance for Content of Premarket Submissions for Software Contained in Medical Devices
- FDA Guidance on Off-the-Shelf Software Use in Medical Devices
- IEC 62304
- IEC TIR 80002-1
- ISO 14971
- Your design control SOP
- Your risk management SOP
- Your Software Change Control SOP
- Your Software Testing and Validation SOP

You could also reference your other major software engineering SOPs - Software Requirements & Traceability, Software and Database Design, Configuration Management, Coding Standards, Code Review. Recommended: provide those references if those SOPs are mentioned in this one.

3.5. Attachments

Do you have any other standard forms or lists or how-to explanations?
Typically, some of the more detailed procedures have attachments, but not the top level one.

3.6. Definitions

Here are some terms typically defined in Software Development procedures:

- Add-On
- Completion Statement

- Design documentation
- High-level design
- Detailed design documents
- Functional Specification
- Level of Concern
- Major Revision / Minor Revision
- Maintenance Release
- Patch
- Regression testing
- Traceability Matrix

3.7. General Information

Topics you could cover here are:

- Process model (or if you maintain flexibility in process methodology, a statement to that effect)
- Project team roles (if not specific to individuals)
- Risk evaluation levels
- Where certain processes (not specific to any phase of development) are defined: issue tracking, software configuration management, coding standards and code review.
- Considerations and requirements for off-the-shelf software.

3.8. Procedure

Classically, the ***software*** development SOP outlines the phases of development, specifies the documents or other deliverables to be created in each phase, indicates which project role carries out what general activities, and defines the "checkpoints" at which the project moves from one phase to the next.

- This document should point out what activities happen in each phase, what output is produced in that phase, and who (or what team) makes the key decision to move to the next phase. (A development iteration is not a "phase" in the sense I'm using here. Rather, all the iterations make up the design/development phase.)
- How do you set forth the overall goals of a new release, and divide them up among iterations? (You will likely have a separate SOP for Requirements Definition and Traceability - reference that one but don't repeat its contents here.)
- If your specific design emerges as part of development, you need to cover how / where / by whom the emerging design is documented.
- For final test, recall you have a separate Testing and Validation SOP; reference that here but don't repeat it. Instead, you can outline the key activities and outputs of this phase.
- An extremely important topic to address (in a clearly labeled section) is Change Control. A separate Change Control SOP may not be necessary. However, you need to make the Change Control topic stand out in this SOP as well as the Requirements Definition SOP. (Talk about change control of developed ***software*** here, and change control of Use Cases in the Requirements & Traceability SOP.)
- Here, the key points will be that software changes will have been evaluated for their impact on the overall product (regression testing, among other things) and will be approved before release.
- How and by whom are user documentation and help materials developed? How do you handle review/approval, and modification, of these materials?
- Somewhere, you'll need to list the outputs / deliverables to be generated in a complete development project (you'll have listed the appropriate ones in each phase). These are considered quality records.

3.9. Release

- Do you define your release requirements and procedure here, or have a separate SOP for that?

- Do you have any abbreviated process for maintenance releases? What would qualify a project to be considered a "maintenance release"?
- Do you have any need to define a process for add-on software? (extra features initially released as a separate entity)

4. Software Requirements and Traceability

4.1. Purpose

Suggestion: state what ends you want to achieve, and what criteria you want to meet, in your requirements gathering process - and follow with a statement to the effect that establishing traceability ensures that all agreed-upon requirements are implemented when a software version is tested and released.

Your goals can be: "to enable continuous refinement of development priorities based on stakeholder input and active risk management strategies". This priorities list can include:

- regulatory requirements and guidelines
- industry standard product expectations
- known issues with existing products
- human factors engineering

4.2. Scope

Should be pretty simple - the procedure applies to all software development activities, including maintenance, at XYZ.

4.3. Responsibility

Use this section as a general outline of how the key roles divide up the areas covered in the procedure. Key roles could be (as applicable):

- Project Manager
- Change Control Board (CCB)
- Analyst
- QA Manager

Alternatively, you could reference responsibilities outlined in the general SOFTWARE development SOP.

4.4. References

At minimum:

- Software Development Method SOP
- Software Change Control SOP
- Software Issue Tracking SOP
- Possibly also your Testing and Validation SOP

4.5. Attachments

- Template for Software Requirements document
- Template for Traceability Matrix

4.6. Definitions

Best – reference definitions in the ***software*** Development SOP but do not repeat them here. Only if necessary, add other definitions here, such as:

- Iteration
- Milestone
- Release (both as a verb and as a noun)
- Use Case

4.7. General Information

You need to make clear that the processes you outline are independent of the tools used to implement them. This procedure, and the others in the Software Engineering category, should describe what gets carried out, but not be too specific about the system used for that activity. If you use a requirements management tool, so much the better - but the important topics are what a requirement must contain, and how you review them.

4.8. Procedure

Recommended: separate out different stages:

- Sources of requirements (who they come from and how they arrive)
- Review and prioritizing (Requirements Group)
- Selection of features / corrections (Change Control Board)
- Organization into iterations (Project Manager)
- That should be as far as you need to go here, since this is only the Requirements/Traceability document. What you describe here is a broken-out subsection of the whole development process, which is outlined in the Software Development Method document.
- Absolutely vital - how do you ***document*** requirements? Where in the documentation do you indicate which ones are approved at what point?
- Change Control of Use Cases is a fundamental topic. The Software Design SOP will go into more detail about what happens to Use Cases in the course of development, so you don't need to address that here. However, it will be vital to specify how changes to Use Cases are reviewed and approved. Make sure the system is "light" enough that it will be followed, but strong enough that you prevent bootleg or cowboy modifications to Use Cases.
- How do you document traceability of requirements to design? To testing?

5. Software and Database Design

5.1. Purpose

- The issue here is not how you design software, but how you document your designs.
- Declare a purpose along lines of "XYZ documents design of all product software, at various levels appropriate to the entity, to help assure consistency and maintainability."
- Note that "documenting" something doesn't necessarily mean creating some long, verbose Word file with a bunch of embedded block diagrams or whatever. Your documentation can take whatever form is meaningful (to whomever needs to consult it six months or a year later), flexible, and controlled - don't forget that last part!

5.2. Scope

It's worth thinking seriously about what kind of software you document this way - are there any projects where this isn't done, and if so, what kinds of projects? If there are no exceptions, then stating that the procedure applies to all software development at XYZ is all you need here.

5.3. Responsibility

- Who is responsible for ensuring documentation is created at which level?
- Do you create high-level design documents? Architecture documents? Component-level documentation?
- As appropriate, do you document database approaches or data models separately?

5.4. References

- Your Quality Manual
- Your Software Development Method SOP
- Your Requirements and Traceability SOP
- Your Testing and Validation SOP
- Any external reference books / articles that give helpful guidance on design documentation, applicable to your development environment

5.5. Attachments

Do you / will you provide standard outlines or templates for specific kinds of design documents? Even if those are web forms, images could be attached to the SOP. Not required - you should only attach something if it adds to understanding or makes something useful available.

5.6. Definitions

- If you use specific terms that could be interpreted various ways, use this section to clarify how ***you*** are using them. Examples are such fuzzy concepts as "high level design", "functional specification", "detailed design", "data model", or "component". (Not implying you use all these terms; they're just examples.)
- Further definitions could include UML, SVN, Wiki, JavaDoc, Doxygen, API, HTML, XML.

5.7. General Information

This section would catch whatever needs to be pointed out but doesn't fit anywhere else in the document.

5.8. Procedure

Here are some of the topics you could cover. Get rid of the ones which don't apply to you; make sure to add anything else that would be needed. (Each of these can be a separate section in the Procedure.)

- Architecture / High-Level Design
- Database Model
- Component Design
- User Interface Design
- If you use Use Cases, spend a section on these, since they are "kernel" inputs for development. It's vital to note that Use Cases as initially provided typically do not give sufficient detail for

complete development or modification of some component, and therefore the Use Cases need to be refined and/or corrected as development proceeds.

- It's also crucial to point out that a developer is responsible for obtaining clarification from the Use Case author, and for making sure that the refined or modified Use Case still fulfills the intention as originally set forth. (In the Requirements & Traceability SOP, we need to address approval of Use Cases).
- It would also be helpful to state that developers meet regularly in the course of a project (don't be too specific - this doesn't need to specify that the meeting is weekly) to review status, discuss issues, and obtain clarification of Use Cases where needed.
- If you have lists of required or typical information to put in any one of these levels, including a table of that information wouldn't be a bad idea. Consider this example of attributes to be specified for a component:

Identification	The name of the entity
Type	A description of the kind of entity
Purpose	A description of why the entity exists
Function	A statement of what the entity does
Subordinates	Identification of all entities composing this entity
Dependencies	Description of the relationship of this entity to other entities
Interface	Description of how other entities interact with this entity
Resources	Description of external elements used by this entity
Processing	Description of the rules used by this entity
Data	Description of data elements internal to this entity

- Other topics to cover:
  Risk mitigation
  Traceability to requirements
- At what point do your design documents need to be finalized and approved? Consider this question carefully!

6. Software Coding Standards

6.1. Purpose

Typical statement:-
"Software coding standards are intended to:

- Enhance the clarity of code, so that developers can later understand it, and therefore
- Assure the maintainability of code, so that developers can later correct errors and expand functionality."

6.2. Scope

- Unless you have any exception areas – software developed for special projects that doesn't fit into the company's key products – this would simply state that the procedure applies to all software developed in-house at XYZ.
- For an open-source development company, these standards would be strongly recommended to all community developers who wish to contribute to XYZ's code base.

6.3. Responsibility

- Clearly, all developers employed by XYZ, or contract employees working with XYZ, are responsible for adhering to these standards.
- Do you contract out some but not all of your software development? If so, who is responsible for coding standards in the code developed under contract?

6.4. References

- Your Software Development Method
- Your Testing and Validation document
- Your Code Review SOP, if separate from this one
- Whatever external source/s would be helpful (web site, book, or article)

6.5. Attachments

This document may not need any.

6.6. Definitions

Only you will know what terms need to be defined here. Example: the term "module" can be defined in a general way (in everyday speech, "piece" or "chunk" convey the same meaning), to keep from having that term confused with any language-specific use of the word.
Other possible terms -

- GUI
- Developer
- Doxygen (if you use it)
- CamelCase

6.7. General Information

Perhaps you need to explain that you're defining standards for a specific language – Java, or C++, or C, or possibly SQL. Other coding standards can be provided as needed for other languages.

6.8. Procedure

Here are broad, overall recommendations to include. Apply as appropriate.

- Use a uniform indenting style (it may be necessary to avoid tabs)
- Adopt and use consistent naming conventions for variables, parameters, GUI elements, and the like
- Perform all database access through a set of library routines, not directly
- Name constants in a way that describes their function
- Comment each procedure / function: purpose, passed parameters, returned values, possible error codes or messages, etc.
- Keep each procedure simple, short, easy to understand

A basic Java coding standards document lists the following headings - see if they apply to you -

- Data members
- Method Names
- Constants
- Parameter Names
- GUI Element Naming
- Database considerations
- Catching Exceptions
- Other headings I would include are:
- Naming Conventions (especially if important for distinguishing different kinds of entities)
- Coding Conventions (header blocks, placement of braces, fields, classes, local variables, etc.)
- Information to provide (for various kinds of entities)
- Comments (what to comment, comment style)

An interesting note one of my developers had me include in our "Programming Guidelines" SOP was as follows:

- "Developers are expected to bear in mind that source code may be a deliverable to the client, and therefore to avoid names, comments, or other material in the code that could cause concern or be considered offensive."
  This may or may not be applicable.
- Common errors to avoid are listed at the following sites:
  For C++: http://hissa.nist.gov/effProject/handbook/c++/
  For object-oriented languages: http://hissa.nist.gov/effProject/handbook/ooclass/
- If your team has trouble establishing a coding standard, conduct a web search to find candidates. Having a complete starting point helps avoid missing obvious issues; you can always adapt the standards to your own needs.

7. Code Review

7.1. Purpose

Modify as you see fit -
"Code reviews are an important tool for detecting and removing defects in code to be released for managing clinical trial or adverse event data."

7.2. Scope

- A "scalpel" is the concept of "critical code": ". . . all code that controls the creation, modification to, or deletion of critical data", where the concept of "critical data" is defined as part of the project quality plan. Think carefully about this approach, and avoid creating more complication than the effort you might save.
- Without that distinction, the scope here is easy. The procedure applies to all software development performed at XYZ.
- Do you ever develop what could be termed "single-use software"? An example is data migration, which is specific to the given database schema - typically, the code doing the migration is created and reviewed, tested on a small sample, then used for the migration and archived away. The resulting data, i.e. the output, is reviewed carefully - so reviewing the code of the migration routine would have been wasteful and redundant. If a similar concern applies to you, it will be wise to write an exception into the Scope.

7.3. Responsibility

Key individuals are likely to be:

- Project Managers (responsible for ensuring that all code, or all critical code, in the project undergoes adequate review) and
- All employees or contractors involved in software development (responsible for following the review procedure, for documenting review dates and results, for making all necessary corrections to code, and for re-reviewing code when required).

7.4. References

- Your Software Development Method SOP
- Your Coding Standards SOP
- Your Configuration Management SOP
- Your Issue Tracking SOP
- Any "outside" references you want to cite for the process of reviewing code

7.5. Attachments

How do you record the fact that review of a piece of code occurred - what code and check-in version, who did the reviewing, when it occurred, whether any questions or issues arose? Do you capture this? Do you use any kind of form for this purpose?

7.6. Definitions

"Critical Code," if you use the concept. Otherwise, use your judgment on what terms to define.

7.7. General Concerns

Add whatever else you think is necessary -

- Reviewer(s) must be sufficiently knowledgeable to understand the code, but need not be a member of the project team.
- Project teams may choose to review code in face-to-face meetings or individually (reviewer examines code separately, and contacts author with questions as necessary).
- The Project Manager must build time for code reviews into the project plan.

7.8. Procedure

This section should address the following topics -

- Process of conducting code reviews
- Guidelines for reviewing

- Review criteria
- Followup

<u>Process of conducting reviews</u> (modify as you see fit):

- Code reviews are intended to uncover errors, as early as possible – they are often more effective than testing for removing errors. (As used here, the term “code review” corresponds to “walkthrough” or a less-formal “inspection.”)
- In a code review, the Author (of the code under review):
  - Provides the code to the reviewer(s), with any needed explanation of the process being carried out. For in-person meetings, the reviewer(s) should receive the code long enough in advance to become familiar with it prior to meeting.
  - Accepts observations and responds to questions from the reviewer(s).
- The Reviewer(s):
  - Ask(s) the author for additional clarification where necessary.
  - Look(s) for potential issues, and point these out to the author.
- In an open source development environment, what parts will you handle differently, and how, for contributed code?

<u>Guidelines for reviewing</u>:

- Each review should be a thorough examination of a specific part of the overall software product.
- If the review is face-to-face, each reviewer must prepare by becoming familiar with the code prior to the review. A valuable review aid is a line-numbered copy of the source code.
- A review is to focus on identifying errors, not on solving them.
- Reviews are held to examine the code, not the author.
- Length of review meetings, and number of review meetings in a day, should be planned so as to avoid overtaxing the participants’ concentration and effectiveness. (Many authors suggest two hours as a reasonable maximum.)

<u>Review Criteria</u>:

The reviewer examines the code for the following concerns:

- Fulfilling requirements and design specifications.
- Performing the correct operation
- Avoiding logic, program-flow, algorithm, array-handling, and other programming errors.
- Utilizing methods that provide optimal performance.
- Adhering to applicable coding standards (including whether comments are sufficient to make the code understandable).
- Note: It is highly recommended that the project team create or adopt an inspection checklist to aid in discovering errors.

<u>Followup</u>:

What kind of record will you create, to show that code was reviewed? Typically, this would be

- Project and/or software product
- Date of the review
- Reviewer(s) and Author of the reviewed item
- Item reviewed: (file name, procedure or routine name(s) as appropriate, version number – ref. your config management SOP)
- Issues – list all issues noted in the review.
- Re-review – agreement whether to review the rewritten code at a later date

<u>Other questions</u>:

- If true issues are discovered, should they be entered into the issue tracking system?

- You should state who is responsible for ensuring that issues discovered in code review are addressed.
- When revised code is checked in to configuration management, should the developer enter comments that the changes resulted from a code review on such-and-such date?
- If you generate code review reports, these need to be archived somewhere / somehow (another quality record - evidence to review in an internal self-audit).

8. Issue Tracking

8.1. Purpose

The purpose of the ***procedure*** is to provide a method for recording and tracking issues, and making summary information about those issues available. (Don't focus on the purpose of the document - it exists to outline the procedure, but any idiot can see that.)

8.2. Scope

Define the types of issues covered in the procedure - they include:

- Software problem observations and feature requests
- Customer/user support calls (if appropriate)
- Corrective actions (if appropriate)
- Preventive actions (if appropriate)

Also be sure to make clear the type/s of software to which this procedure applies.

8.3. Responsibility

The key players whose responsibilities you should briefly outline are most likely

- Project managers
- Employees or contractors who develop, test, or evaluate software
- Employees who perform training, installation, or other customer services
- Employees who handle customer / user technical support contacts
- Quality Assurance

8.4. References

- Your quality manual
- Your software development procedure
- Your Software Testing / Validation SOP
- Your Corrective and Preventive Actions procedure (if appropriate)
- Possibly – your coding standards / code reviews SOP/s

8.5. Attachments

You typically don't need any

8.6. Definitions

- Corrective Action (if needed)
- Preventive Action (if needed)
- What classifications do you select for your issues? "Product / Category"? Severity? Priority? Each needs to be defined.
- Does your issue tracking system have a series of workflow "states" set up? What are they called, and what are the criteria for an issue being in each of those states?
- Assuming you select a severity for each issue, define the severity levels. An example of severity level definitions (applicable only to software bugs):

Critical:	Issue which could lead to death or serious injury of a patient, caregiver, or bystander; Data lost or destroyed; Incorrect statistical calculation or data transformation; System crash; Major feature not functional, no workaround; Performance issue that prevents a feature from being used under reasonable circumstances
Serious:	Issue which could lead to non-serious injury of a patient, caregiver, or bystander; Erroneous but believable assay result (for diagnostics); Minor feature or minor portion of major feature not functional; Feature not working as documented; discrepancies not easily recognized by user; Error condition incorrectly handled, or error attempting an operation that is clearly illegal; Performance issue that makes a feature unnecessarily difficult to use

Mild:	User interface malfunction or confusing interface; Feature not working as documented, but discrepancies easily reconciled by user; Reasonable workaround exists
Annoyance:	Generally, a user-interface error with simple workaround; Menus, dialogs, or program behavior unclear, unexpected, or inconvenient; Unexpected screen sizing, wrapping, scrolling, or related behavior
Cosmetic:	Screen labels misspelled, not grammatically correct, improperly placed, or typographically inconsistent

8.7. General Information

Use or remove this section as appropriate

8.8. Procedure

I see three sections to this.

- Information in an issue
- Problem/issue workflow
- Summaries / reports which must be available

Information in an issue (typical list):

- Originator: person who found the discrepancy or suggested the feature change.
- Project or Product to which the Problem Report relates
- Summary: a brief, one-sentence statement of what happens. This must be written to stand alone, separate from the full description and "steps to reproduce." (Most readers will only ever see the summary.)
  NOTE: It is extremely helpful to classify problems via a product-specific categories, to aid in grouping and analyzing Problem Reports.
- Version: Software version number that gave the error, or to which the observation applies.
- Type: Type of issue covered in the observation (Software Issue - problem or failure, Change Request, Document Issue, Support). If the project uses a different set, the Project Quality Plan must provide or reference their definition:
- Severity (define the scale somewhere)
- Priority (Work assignment)
- Computer Configuration
- Description: Nature of the failure, complete description of the error (including why it is an error, if necessary), error messages that result. The user who enters a PR is strongly encouraged to characterize the failure as best possible – type and number of tables or files open, client and server configurations, action in progress, sequence of actions that lead up to the failure, and so on.
- Steps to Reproduce: if not clear from the description, a sequence of steps that will generate the same failure.

Workflow:

- A Software Problem Reporting SOP describe a series of things for each workflow status (Open, Fixed, Closed):
- Definition of the status (e.g. Open = "to be acted upon")
- Entry: how a problem report (issue) gets into that status
- Actions: Who does what to an entry in that status (specific responsibilities of project manager, developer, QA, typically)
- Destinations: where a problem report / issue can be sent from that status (e.g. erroneous or duplicate reports can be closed directly; if a report is correct, it is typically sent to "Fixed" status). Note that an issue may have more than one possible destination, depending on the information it presents and on someone's analysis of the issue.

- Typical question - once an issue is set to Closed, can it ever be reopened? Or do you require a that a new issue be opened, even if it relates to the same item as before?

<u>Summaries</u>:

- This shouldn't be too explicit or prescriptive. Since you are using this system for corrective and preventive action (CAPA), and CAPA results feed into your management review, you should at least say a little about the general kinds of summaries you need to be able to generate. I jotted down a few:
  - Number of active issues, by product/group and date entered
  - Number of issues in each status for a selected product / group at a specific time
  - "Arrival Rate" - number of new issues per week, month, etc. for a selected group/product
  - Number of active issues for a selected group/product, by severity.

9. Software Configuration Management

Do you carry out any projects that would be considered strictly research, where the purpose is to explore a technology rather than to release a software version - and you will not use any of the code as part of a product?
Do you pursue different kinds of projects, where you employ different means for configuration management? That is, does this procedure need to have "elasticity" written into it, concerning how you achieve SCM?
Do your projects create any artifacts which NOT amenable to storing in a code-configuration system? (would GUI screens be an example?)

9.1. Purpose

State why you use SCM in the first place. Typically, SCM is intended to manage and track changes in source files so that:

- The complete configuration of a software deliverable can be known at any given time.
- XYZ can always reconstruct the released or delivered version of a software product.
- Changes to source code will be applied in a stable, controlled process.
- Developers can avoid inadvertent mix-ups or reintroduction of erroneous code.
- Developers can examine the trail of previous changes in a section of code.

9.2. Scope

See the questions above - do you need to exclude any types of development or types of files? Otherwise, state the types of files or artifacts to which this procedure applies.
The term "source files" includes but is not limited to:

- program source code
- installation command files
- system configuration files
- database schema definitions
- form or report definition files

9.3. Responsibility

- Clearly, all employees (and contractors, if applicable) who develop code are responsible for using the current configuration management system to retrieve code for modification, and to save code after modification (if a test step is associated with check-in, that is rightfully part of the Testing and Validation SOP, with a cross-reference to this one).
- Who is responsible for ensuring that the systems in use are adequate to identify, and control access to, any software element so as to prevent mistaken use of previous versions or reintroduction of previous errors? (Classically, this would be QA - your call.)

9.4. References

- Your Software Development Method
- Your Change Control SOP
- Possibly, your Testing and Validation SOP
- Possibly, your Release and Deployment SOP
- IEEE Standard 610.12, if you reference it for definitions you choose not to repeat here

9.5. Attachments

Probably not needed.

9.6. Definitions

Take a long, hard look at the specific terms which could mean different things to different people.
Typical terms would be:

- Release – v. The formal notification and distribution of an approved version.

- Major Revision – A revision, or version, of a specific software item that involves modifications to the product's Concept Definition. Major version changes are indicated by a version number change to the left of the decimal point (e.g. 3.47 to 4.00).
- Minor Revision – A revision, or version, of a specific software item that was created only in order to correct known issues or to refine existing functionality, and does not involve changes to the Concept Definition. Minor version changes are indicated by a version number change to the right of the decimal point (e.g. 2.65 to 2.68).
- Version – An initial release or a complete re-release of a software item or software element.
- Version number – A unique identifier used to identify software items and the related software documentation which are subject to configuration control.
- Configuration status report: document which lists, for a specific version or build of the software product, the names and version numbers of all source files used in the build.
- SCM: in general, software configuration management; for purposes of this document, source code configuration management.
- For definitions of the following standard terms, see IEEE Standard 610.12-1990: baseline, component, configuration, configuration control, configuration identification, configuration item, configuration management, configuration status accounting, version.

Use your own definitions and reference for other terms.

9.7. General Information

It may be useful to spell out that SCM must fulfill these criteria:

- Each configuration item is uniquely identified.
- Changed versions are clearly marked (e.g. by increasing the version number). A description of the change is highly recommended, to aid later maintenance.
- At any point, the Project Manager or designee can obtain a complete listing of all components that make up the software project, with their version numbers (i.e. a configuration status report).

9.8. Procedure

You have several main topics to address here - we're keeping this SOP fairly high-level, so it will still be applicable even if implementation details of your SCM tool change.

- General Concepts
- Version Numbering System
- Routine use
- SCM of items other than code (if this is an issue)
- Configuration Management of Supporting Software (which may not use the code control system at all)

General Concepts

- Configuration identification: unique identification of every source file or other configuration item.
- Configuration control: review (typically by consensus) of changes to each configuration item, and storage in such a way that changed versions are distinguishable and changes can be tracked.
- Status accounting: production of a configuration status report.
- Configuration audits and reviews: When appropriate (for example, in reviewing a maintenance release), Quality Assurance will compare the configuration status reports of the new version and the previous version.
- Version Numbering System - most important topic is how you decide something is a major or minor release. After that, it may be helpful to describe, in relatively high-level terms, how you handle major vs. minor releases in the SCM tool.
- What kind of records do you keep and for how long?

<u>Routine use:</u>

- This can be several bread-and-butter statements -
- Developers must work only with checked-out code, and must check in all changes that are to be kept;
- Project Manager is to assure that a configuration status report is produced for any configuration considered a baseline (at minimum, the configuration to be released).
- QA must, for interim test versions and maintenance releases, compare configuration status reports of the new and old baselined versions.
- An excellent habit - perhaps also be a developer responsibility - is to document what was changed or added, each time something is added or updated in the SCM system. This can take many forms; it's up to you how best to implement it.

<u>SCM of items other than code</u>

- This may or may not be a concern (some SCM tools can only store code as ASCII text). If you have to handle GUI screens or other artifacts differently, you need to set out criteria for what is managed and the purpose/s that management must serve.

<u>Supporting Software</u>

This section could cover the following types of topics:

- All supporting software used in development, even though not a part of the product, must also be controlled. Examples of such software would be:
    - Compilers and associated utilities
    - Third-party libraries
    - Installation packagers
    - Operating system
    - Database management software
- The term "control," in this context, only implies maintaining the affected item in a way that resists unplanned, unauthorized, or inadvertent changes. Use of a source control system is therefore not required for supporting software, if some other method achieves adequate control. In case of doubt, the question should be discussed with the Quality Assurance Manager.

10. Software Testing and Validation

An SOP on testing / validation of your software should be relatively high-level, stating in general terms the why / who / what and leaving to project plans the specifics of how.

10.1. Purpose

Consider a statement like this:

"Testing is planned, performed and documented to show both verification (written requirements are satisfied) and validation (product meets user needs)."

10.2. Scope

- This should be a pretty cut-and-dried statement, along lines of "This procedure applies to all software released by XYZ for use in any type of therapeutic or diagnostic application." (You'll probably want to refine the use statement.)
- Some companies make an exception for any software development that was pure research only, or where someone else took contractual responsibility for the testing / validation. If you choose to take this approach, be ***very*** cautious in applying it.

10.3. Responsibility

Start with these, and adjust them as fits:

- Project Manager: The Project Manager and QA Manager will collaborate on a QA plan, tailored for the features under development. The Project Manager must also coordinate with the developers to ensure that high-quality unit testing is being conducted in parallel with development. When necessary, the Project Manager can reassign defect resolution among developers.
- Developers: Developers will be responsible for writing JUnit test cases for all code they develop, and for performing regular and frequent regression tests on their code.
- QA Manager: The QA Manager will be responsible for designing the QA plan and related test materials, including test cases and procedures for acceptance testing. This may include coordination with clinicians to perform acceptance testing with actual users. The QA Manager will also be responsible for administration of the bug tracking system. This includes assigning defect resolution duties to developers.
- Testers: QA staff will be assigned specific QA tasks, and will be responsible for reporting defects found. In the absence of specific QA tasks, Testers will engage in exploratory testing in a manner which generates additional test cases and report all defects.

10.4. References

- Your Quality Manual
- Your Software Development Method SOP
- Your Configuration Management SOP
- Your Issue Tracking SOP
- Your Software Release and Deployment SOP
- External references on testing, as you see fit – for example, IEEE standard 829.

10.5. Attachments

You may want to consider three attachments:

- Test Plan Template.
- Test Procedure Template
- Test Summary Report Template

10.6. Definitions

If definitions are adequate in the Software Development SOP, just reference that. Otherwise, focus on terms that tend to mean different things to different people, or which are more or less unique to your product or methodology. Just as examples, consider the following (if these are terms you feel need to be defined, use definitions that fit ***your*** use):

- Acceptance Criteria: Specific, objective criteria for determining that a software product or customized system fulfills its objectives. These may be expressed in such terms as (for example):
  - Specific top-priority functionalities that must be implemented.
  - Percent of all initially-requested features implemented.
  - Maximum number of open, active Problem Reports with Serious or Mild severity.
  - Zero open, active Problem Reports with Critical severity.
- Beta Evaluation: Execution of the software or system by one or more independent persons out of house, intended to reveal errors, problems in performance, or failure to meet user expectations prior to release.
- Functional Specification: A document intended to communicate, in terms understandable to the Client or User, intended features and characteristics of the software to be built.
- Release Testing: Execution of the software or system by one or more independent persons in house, under specified conditions, intended to determine whether the software or system will fulfill its specifications and perform according to user expectations.
- Requirements document: A comprehensive, written set of required software features, based on marketing, regulatory, and user input.
- Software Verification: The process of confirming that the software conforms to all its specifications.
- Software Validation: The process of showing that the software fulfills customer expectations and is suitable for the intended use in the customer setting.
- Traceability Matrix: A document, generally presented as a table, which relates all the elements at one step of the development process with all the elements at another step. Often generated to relate requirements to tests, or system functions to tests.
- Test Procedure: Document which specifies the steps to follow in testing a given functionality in the software.

Other terms used here ("release," "revision", "unit testing," "version") are as defined in IEEE Standard 610.12-1990. [Use whatever reference you think best suits your situation.]

10.7. General Information

Probably not needed

10.8. Procedure

You need to address several major topic areas:

- Types of testing
- Test Planning
- Release testing
- Reporting and Archiving Test Results

Types of testing:

- This includes unit testing, integration/system testing, performance testing, usability testing, and user scenario evaluations. Some information beyond the definition of each would "set the stage," which is all you want to do here.

Test Planning:

- During the planning phase of each development iteration, a test plan will be created. The test plan will detail the specifics of the testing process, including testing environment, schedules, personnel, automated unit tests, regression tests, and test cases, tailored to the nature of the features developed in the iteration.
- The contents of the test plan should include:
  - Minimal Operating and Testing Environment
  - Personnel
  - Basic System and Integration Testing
  - Data Set Creation and Validation

  - ♦ Performance and Load Tests
  - ♦ Application Modules Tests (Unit Testing)
  - ♦ Hardware/Software Test Environment
  - ♦ Exploratory Testing Procedures
  - ♦ Acceptance Testing Procedures
- Whether you spell all this out in the SOP, or list these things as examples in the Test Plan Outline attachment, is up to you.

Consider the following overall Test Plan outline:

- Project Information (Software / version under test, Test Plan version & date, Test Coordinator)
- Introduction / System Overview (can reference a separate document)
- Reference Documents (List of key documents related to the Test Plan: PQP, Requirements document, Functional Specification, Design Specification)
- Scope and Approach (Functional areas to be included and excluded from testing, as appropriate; type(s) of testing to be used. Provide risk analysis where appropriate)
- Tasks, Roles and Responsibilities (Tasks associated with testing, persons taking part in the test process, and roles they will perform [test coordination, testing, review, test development / revision])
- Testing Sequence (Planned order and approximate schedule of performing the planned tests. May refer to a separate testing-project plan document, if one exists.)
- Testing Environment (list of all the hardware/software environments in which the package will be tested)
- Qualification Tests (A list of all test procedure / test cases to be carried out in formal testing activities – this may reference a separate document. Each test must indicate the requirements to be met, and describe the test to be performed.)
- Test Inputs and Results (type of input to be used to perform the tests; where and how testers will record results and observations)
- Discrepancies (Planned approach for recording, tracking, evaluating, resolving, and retesting any anomalies revealed in the testing – tailor as needed)
- Acceptance Criteria (Provide clear criteria for deciding that the product passes testing. [For example, "All top priority requirements / functionalities implemented, no outstanding critical or severe problem reports against any of the top-priority functions, 95% of all functions implemented, and no critical problem reports against any functionality in the software."] Specify who will participate in judging that Acceptance Criteria have been met: Project Manager, Test Coordinator, Client or User as appropriate, others as necessary.)

<u>Release testing</u>:

- This is where the process becomes "official." The concerns are
  - ♦ Logging test results - remember that the completed test cases are a quality record your auditors will want to see.
  - ♦ Reporting anomalies - this sets out the requirement that issues be recorded, and points to the Issue Tracking SOP
  - ♦ Version replacement and retesting - this may not be as much of an issue for you, if you run automated regression tests and have complete functional coverage or nearly so. If all testing is manual (time consuming), this can be a concern. If you choose not to repeat certain test procedures on every new version because you are convinced that the functionality in question is not changed, then justify this choice by comparing sequential Configuration Status Reports.

<u>Reporting and Archiving Test Results:</u>

- A test summary report needs to tie up a release package - you would do well to specify what goes into that report.
- Records to be archived should also be spelled out.

- Example Test Summary Report outline:
    - Project Information (project name, test plan version/date, test coordinator)
    - Summary (Summarize the evaluation of the test items. Identify the items tested, indicating their version/revision level. Indicate the environment in which the testing activities took place.
    - Supply references to test plan and all test procedure documents.)
    - Variances (Report any variances of the test items from their design specifications. Indicate any variances in the test plan or test procedures. Specify the reason for each variance.)
    - Comprehensiveness Assessment (Evaluate the comprehensiveness of the testing against what was planned. Identify any features not sufficiently tested and explain the reasons.)
    - Summary of Results (Summarize the results of testing. Identify all problems resolved during final release testing. Identify any open problems.)
    - Evaluation (Provide an overall evaluation of each test item, including its limitations, based on the test results and the item level pass/fail criteria. The evaluation may include an estimate of failure risk.)
    - Summary of Activities (Summarize the testing activities and events. The summary should cover all resources: staffing level, machine use, and elapsed time for each of the activities associated with testing.)

<u>Records to be archived:</u>
Consider a statement such as the following:
The Test Coordinator is to archive final records of the testing, which will include:

- Test plan
- All test procedures
- All test logs (may be filled-in test procedure documents)
- All additional logs of ad-hoc testing
- Test Summary Report

11. Software Release and Deployment

11.1. Purpose

Suggestion

"XYZ has established build and release procedures to provide an orderly, controlled method for, and to help avoid errors in, transferring software from the development environment to end-user distribution."

11.2. Scope

Keep it simple. The procedure should apply to any software released by XYZ for use in medical therapeutic or diagnostic applications.

You may need to tweak that statement, but the point is to cover all uses that are not strictly research.

11.3. Responsibility

This should be completely black and white. Persons responsible for carrying out this procedure are:

- XYZ employees involved in developing, testing, or releasing software.
- All temporary and contract employees participating in software development at XYZ.

11.4. References

- Your Software Development Method SOP
- Your Configuration Management SOP
- Your Testing and Validation SOP

You may need to cite others, e.g. a build procedure document if it's separate, but these are the key ones.

11.5. Attachments

If you create a checklist specific to each release, then a template for that checklist should be attached. Otherwise, attachments aren't needed.

11.6. Definitions

Terms which could be defined:

- Build:
  - v. to create a customer installable software product out of source code from the developer(s). Traditionally, this process consists of "compile" and "link" steps, executed via a "make" file.
  - n. The product of a specific build event. Roughly synonymous with "version."
- Release: To approve a revision of software so that it may be made available to end users.
- Support: Providing means to contact XYZ with problem reports, and a method within XYZ for tracking and resolving reported problems.

11.7. General Information

If build procedure is combined with release procedure, then criteria for a build will be:

- Consistent and repeatable: a written checklist or a make file or script. Automating the build procedure (as far as possible) is strongly recommended, to ensure that all steps occur in the correct sequence. Note: The make file or script used for builds should also be controlled.
- Referenceable: builds can be referenced in the source control system (revision level can be determined for each source file used in a given software build).
- Distinguishable: each build has a unique identity.
- Secure: source code is derived from a controlled repository, where only authorized individuals can modify the contents.

If not in this procedure, these criteria need to be stated wherever the build process is set out. Note in particular the "referenceable" criterion - you can list all the files / pieces that went into a given build, and know the version of each piece. Having that list – I dubbed it a "Configuration Status Report" – makes it ***much*** easier to report on "what changed" in each new release.

- Do you have different projects where you use somewhat different processes for release? Do you want to point out that the project manager is responsible for documenting any project-specific release procedure?
- Do you prefer to document the build procedure as a document on its own, and reference that here?
- Do you, or should you, have some policy about how far back in your version history you'll support - or is this even an issue?

11.8. Procedure

What are the prerequisites for release? Adapt the following as appropriate:

- All required deliverables, as listed in the Project Quality Plan, are available. (These would include installable software and, as appropriate, installation instructions, user guide, and release notes.)
- The Project Team has reviewed and approved all documentation.
- The Project Team has prepared a Configuration Status Report for the final version, and for any previous versions as necessary to justify partial testing.
- Testing of the final software build is complete and Acceptance Criteria have been met.
- Note: If the final build was not subjected to the complete suite of tests, then the excluded tests should be justifiable by what functionality did not change from a previous version where that functionality was tested. Proof that modules were unchanged comes from the Configuration Status Report.
- The Project Team has reviewed results of testing and agreed that the version is suitable for release.

What then are final steps for a release? Example:

- Project Manager
  - Ensure that master copy has been created and scanned for viruses. (Note: Virus software must be up to date on the station used for scanning.)
  - Deal with sending code to escrow if this is required.
  - Ensure that Project Team has performed a "Readiness Review," and persons responsible have completed all the necessary actions.
  - Ensure that an offline copy of the distributed software (CD-ROM or diskette) is archived in offsite storage, and that a copy of any support documentation distributed in paper form is retained.
- Quality Assurance - ensure that acceptance testing has successfully completed from the distribution master copy.

Naturally, you can modify these as appropriate to your process.

- If a release checklist exists, it could be included in this document.
- If steps for release vary, then a checklist template with the basic elements should be provided.