Quality System: Design Control Procedure

CORP Medical Products

1. Overview

1.1. Objective

This document describes the details of the **CORP** Design Control Procedure for Project Change Requests (PCR).

1.2. Scope

This is required for any PCR.

1.3. Responsibility

This document shall be maintained by CORP Management personnel.

This document shall be followed by all CORP personnel involved with PCRs.

1.4. Procedure and Outputs

The following sections describe the operation of the CORP Design Control Procedure for PCRs in detail sufficient to allow a software developer or project manager to understand its use on a day-to-day basis.

Sections that follow

1.5. Definitions

FDA Definitions

DCP	Document Change Procedures
DHF	Design History File
DMR	Device Master Record

CORP Definitions

RDRC	CORP Design Review Committee (MGMT personnel)
MGMT	Person(s) in management position
PP	Point Person
TAGSYS	System for tying all code and document versions to SCR state
SCR	Software Change Request (this is really a Product Change Request, but for historical reasons, we continue to call it an SCR).

1.6. References

CORP Quality System: Design Control Standard Operating Procedure (CORP.820.30.sop.doc)

CORP Quality System: Design Control Procedure Appendix (CORP.820.30.app.doc)

CORP Quality System: Design Control Procedure Tools (CORP.820.30.tools.txt)

CORP Quality System: Introductory presentation (CORP.820.30.intro.ppt)

CORP Quality System WEB SCR Tool

2. Overview of the Design Control Procedure

2.1. What Initiates a Product Change Request?

A product change request can be initiated for many reasons, including hazard analysis results, customer problem reports, or suggestions to improve accuracy, user friendliness, functionality, or error handling. Such suggestions can come from alpha or beta test sites, university staff, CORP employees, customers, or even prospective customers. Change requests can range from simple cosmetic modifications to complex functional changes. Regardless of the complexity or type of change, all change requests are subject to the CORP Design Control Procedure.

2.2. The Design Control Procedure

This is an overview. All of these issues are described in later sections. Refer to the appendix for more details. In order to avoid ambiguity, we refer to states as either “previous” or “next” relative to a transition of interest; there is no “current” state.

Figure 1 (next page) shows our cyclic development state machine. Cycles can occur from any state back to any previous transition (cycles are not shown). Transitions are named with trailing => (e.g., Create Concept=>), and States are named with trailing # (e.g., Concept#). Transitions are actually processes, and are where all the work gets done. During a transition process, the state following it is being created, and can change until the transition finishes. States exist entirely as the information and other tangibles that define them (e.g., electronic files, references to paper documents, source code). When a transition process finishes, the State following it is reached. When the machine has reached a State, no action is occurring. Reviews=> are actually “controls” on the state machine, determining if the previous state is acceptable, and whether to continue on or to cycle back to a previous transition. To set these off from the fundamental development process, both Review=> and Review Output# are put on one line and indented. Only the uppercase REVIEWs are required to occur.

Figure 1: Development State Machine

Create Concept=>
Concept#
Create Initial Design=>
Initial Design#
REVIEW=> Review Output#
Create Design Input=>
Design Input#
Review=> Review Output#
Design Process=>
Design Output#
Review=> Review Output#
Verification=> ---> Verify Conformance with Design Input
Verification Output#
Review=> Review Output#
Validation=> ---> Validate Conformance with User Needs
Validation Output#
REVIEW=> Review Output#
Transfer=>
Final Product# (identical to Design Output#)

Figure 2 shows the correspondence with the simplified phases recorded in our high-level web-based SCR tool for overall project tracking (shown on right side).

Figure 2: Correspondence with Web Tool

Create Concept=> --. IDEA
Concept# --‘
Create Initial Design=> --. EVAL
Initial Design# |
REVIEW=> Review Output# --‘
Create Design Input=> --. PROG
Design Input# |
Review=> Review Output# |
Design Process=> |
Design Output# |
Review=> Review Output# |
Verification=> |
Verification Output# |
Review=> Review Output# --‘
Validation=> --. TEST
Validation Output# |
REVIEW=> Review Output# --‘
Transfer=> --. FINI
Final Product# --‘

2.2.1. Cyclic or Spiral Development

Most software development follows a cyclic (or spiral) design process. Cycles can occur among the above steps, especially among Design Input, Process, and Output. The output of one cycle serves as the foundation of the input for the next, with each getting closer to the final product (thus the term “spiral”). Each such cycle is called a “phase” of development.

For example, the initial Design Input might consist of only the requirements specifications. In this development phase, the Design Process might produce high-level design specifications as the Design Output. These design specifications, combined with the original requirements specifications, become the Design Input of the next development phase. In that phase, the Design Process might lead to prototyping or modifying existing software, which is used to further evaluate needs and ideas. This can lead to a Design Output that prescribes modifications to the Design Input specifications for the next development phase. Entirely new requirements may be added, others may be modified or removed, or the high-level design specification may be modified. These iterations will occur as many times as necessary until the Design Input is translated into a Design Output which conforms to those requirements.

At any iteration, the Design Output will likely contain more information than is needed for the next Design Input, while also lacking information which is needed, such as the plan for following phases. The process of selecting the relevant information, adding the new information, and creating the ongoing plan, takes place in the next Create Design Input process. For example, one design phase may result in multiple prototypes being created, which all become part of that Design Output. However, the next Create Design Input process might simply select the most appropriate approach, suggest refinements to that approach, and lay out the plan for this and following phases. Although the previous Design Output remains available for future reference, only the relevant parts of it need to propagate to the next Design Input.

Although these cyclic phases can be planned, such plans often change due to what is learned during the process. For example, software design often combines both top-down (driven from the desired functionality) and bottom-up (driven from the implementation details) techniques. This can apply even to the definition of the initial requirements specifications. Desired high level requirements specifications may have to be changed in the face of implementation constraints such as subsystem initiations or limited resources.

It is not necessary that all such cycles be explicitly recorded. Consider the situation where implementation details or limited resources lead to modification of the requirements specification. This could all occur as part of the “Create Design Input” process, with the reasons for the decisions simply recorded in the “Design Input” documents. This could also occur as two explicit phases, where the first Design Input contained the original requirements specification, and the resulting Design Output shows the need to modify the input based on what was learned. The next phase would create the new requirements specification as the Design Input, and proceed from there.

2.2.2. Transitions and States

Each “Transition Process” is simply everything that needs to occur to create what is required of the next State. For example, the “Design Process” transition is everything that needs to occur to “realize” the Design Input requirements, thereby creating the Design Output.

Each “State” will have a corresponding Checklist that must be reviewed during the transition to that state. The Checklist itemizes the issues that must be considered during the transition to this state, and the information that must be recorded in this state. The checklists will be based on FDA requirements, past experience, and knowledge of this application and its users, so they will evolve over time.

The necessary State information must be recorded and updated throughout the transition process. It is not something that just occurs “at the end.” The state must be an ongoing repository for all information related to the transition to that state. This way, things are not forgotten, especially if the project is delayed or there is a change in personnel.

Each State exists entirely as a collection of files in the SCR directory. These files can refer to other locations, such as file folders, literature references, files that exist in other State directories (e.g., results of some Review), or the CVS source code tree. Each state will have a set of fundamental files that are required, and a directory in which the developer can put all additional supporting information (e.g., email bags, images, random discussion documents, code fragments, whatever).

A project may be in a Transition Process, even though no actual work is being done. For example, there may be no personnel available to work on it. This issue is external to this design control process. It is part of the larger picture in which company resources are allocated and company projects are managed. So, for example, after the Concept is reached, the project is technically in the Create Initial Design transition. However, unless it is assigned to someone and made a priority, it could sit there forever, with no further progress being made. To help track this aspect, an “activity” indicator is provided (e.g., active, delayed, hold).

2.2.3. Recording States

As each State is reached, the current version of each file in this SCR area is tagged in a way that ties them to that State (the process is referred to as TAGSYS). If the state refers to other information that is also under version control (e.g., source code), then that will be tagged as well. If such other information is outside of our version control (e.g., printouts, references to 3rd party articles, paper notes in file folders), we will provide as much information as we can (e.g., indicating dates of printouts or articles, or labeling separate file folders with the state). Thus, at any later time, the state of the design process at any step and any phase of the SCR can be recreated, and the information reviewed.

2.2.4. The Status File

One document common to all states is the Status file, which contains 4 sections; To Do, In Progress, Waiting, and Done. This file is to help the developer track what needs to be done and what is in the works. As items are completed and moved to the Done section, it provides a convenient historical record. As we often switch among projects, with gaps of weeks or months, or as we change developer assignments, this provides a valuable tool for getting back into an older project.

As long as items remain in the To Do, In Progress, or Waiting sections, the state is not stable. The previous transition process is still occurring and the contents of that state subdirectory are changing. When the previous transition process is finished and all checklist items associated with this state have been addressed, the state is considered stable, and only the Done section should then have entries. At this point in time, the next transition process can begin.

The reason for having both the In Progress and Waiting sections is to make it easy to see what you are currently waiting for others to do, as opposed to what you are doing yourself. "Waiting" issues are those you are waiting on, but which are out of your control (e.g., waiting on someone else to finish some code, waiting to receive new documentation from some company, or waiting for a user site to test some new software). "In Progress" issues are those you are currently working on, which are primarily under your control. By having the separate section, we can easily create a tool that will show all the Waiting issues for all the SCRs in your area, thus allowing you to readily follow up on things that aren't happening.

2.2.5. Reviews and Cycles

The Review after any state evaluates whether the previous state has met its Checklist requirements in terms of both (1) the transition process which created that state, and (2) the required output of that state, and whether these have been met in an acceptable manner. Based on the Review, either the next process can begin, or a cycle back to a previous process is required, along with recommendations on what needs to change. This is all recorded in the Review Output.

If all requirements are acceptable, the next step is set to the next sequential transition. However, if problems are discovered, the next step will be set back to one of the previous transitions, with specific instructions on what needs to be addressed. For example, the Design Output Review might force a cycle back to Create Design Input to reevaluate the requirements specification.

When cycles occur back through previous States, the previous State contents are always preserved for future reference; they are never altered or removed. All changes to the previous documents are managed through the document change process, which insures control over the changes and that all previous versions are maintained and available for review.

Only the following Reviews are mandatory:

- Review of Initial Design
- Review of Validation

The other Reviews are optional. For example, we will usually conduct the Reviews of both Design Output and Verification Output at the end of Verification Output. The rationale is simply that the Design Output may need to be modified based on Verification results; further, developers often conduct Verification in parallel with creating the Design Output. Simpler projects may conduct Design Input, Design Output, and Verification Reviews all at the end of Verification. The simplest projects may limit the Validation Review to only review the Validation, not requiring the Input, Output, and Verification reviews to occur at all.

The Review Schedule is created in the Initial Design, so is always subject to initial Review. It can then only be modified by MGMT.

2.2.6. Standard Review Output

The Review Output of any Review always includes the following

1. A record of the review results and who participated.
2. A record of the next step that is to occur.
3. The possible addition of action items to other states.

The “next step” is either the next sequential transition, or a cycle back to a previous transition if the Review discovers issues that need to be addressed. If the previous transition is more than one step back, the Review may discover issues that need to be addressed at multiple previous states. For example, a Verification Review may cause a cycle back to Design input, with issues to be addressed at Design Input, Design Output, and Verification.

When the next step is a cycle back, each issue that must be addressed is ALWAYS recorded in the Review Output. Then, one of two situations exists:

1. The issue is simple enough to be addressed during the Review:

   In this case, the changes are made to the previous state during the review process.

2. The issue is bigger than can be addressed during the Review:

   In this case, the issue MUST be added to the previous state Status file To Do list. It is not sufficient to just record it in the Review Output. The reason is simple. Long time delays or personnel changes can occur before the cycle back actually takes place. If the items are not in the Status file To Do list, how would one later know they needed to be done?

   If the issue cannot be immediately added to the previous state Status file, an entry must be immediately made in the Review Output Status file To Do list, indicating that this must be added to the previous state Status file. As long as the issue is on the Review Output To Do list, the Review is not complete, and it will not be forgotten.

   The issue added to the previous state Status file must provide specific instructions on what must be done. To keep the Status file entry short, it can refer to other files that contain the detailed information. For example, during the review process, detailed information about the issue might be recorded in the Review Output area. There is no reason to replicate this information elsewhere; simply refer to the Review Output area. However, if it is more appropriate to record this information back into the previous state, it can be added to the previous state files, or new files can be created in the previous state, and then it would refer to these files.

Before the Review process begins, a TAGSYS snapshot of all states is created. When the review is complete, another TAGSYS snapshot is created. Thus, regardless of whether the review forced a cycle back, or modified a previous state during the review, the different phases of the previous state will be captured via TAGSYS. Both situations will show a cycle back. The only difference is that in the latter case, the cycle back occurred during the review process, and is finished when the review is finished.

Note that, even if next step was the next sequential transition, action items can be added to later states, based on Review Output. For example, a Design Output Review may discover an issue that should be considered during Validation. It can be added to the Validation Status file.

2.2.7. Other Ways to Cycle

In addition to Reviews, the output of any process (i.e., the state which follows it) can cause a cycle back to any previous process. This could be a planned cycle, or an unplanned cycle due to what was discovered in the process. For example, if Verification fails in any phase, then either the Design Output or Design Input must be altered. The former requires a cycle back through Design Process and Design Output to correct the problem, while the latter requires a cycle back to Create Design Input, to change the specifications. Of course, if the Design Input was subject to Review, then

this modified Design Input will also be subject to Review before further work can proceed. Or, as another example, if Validation fails, the process is the same as with Verification, with the added possibility of a cycle back to Create Concept or Create Initial Design.

As a final example, consider that development on any project may need to be curtailed due to limited resources and other pressing needs. At that point, a reassessment of Design Input requirements specifications may remove certain requirements, perhaps moving them to a later project for a future release. This can either be planned, where primary functionality is developed first, then resources reassessed for secondary functionality, or it can happen asynchronously at any point in the development process due to external issues.

2.2.8. Larger Projects and Hierarchical SCRs

Larger projects will generally be decomposed into smaller subprojects, any of which may merit its own SCR. In these cases, we create a Master SCR for the main project, which then refers to the subproject SCRs in its state files.

For ease of visibility and management of SCRs, only one level of sub SCR is allowed (i.e., a sub SCR can not spawn a sub SCR). Any necessary control on the dependencies or ordering of subprojects (e.g., SCR N must be completed before SCR M can begin) is specified in the Master SCR state files.

Care must be taken in such decompositions, since interactions among the sub SCRs can be difficult to track and understand (e.g., modifications to Design Input Specifications of a sub SCR may impact another sub SCR, but might not be visible to that other sub SCR). These must be carefully thought out, and any decision to create a new sub SCR must driven entirely by the question:

> Will this help provide the most effective development path, or would it be better to stay a part of the current single SCR?

Although partitioning into subprojects is best done during Initial Design, this isn't always possible. The need might not be seen until the project is better understood, which might take many design cycles, or unexpected events can occur later in the design process. In such cases, a new Master SCR must be created, which then refers to the initial SCR, and any new sub SCRs that must be created. The existing "first" SCR is not allowed to just spawn sub SCRs. The reason is that we need to keep the Master SCR as clean as possible, only dealing with the management of the sub SCRs.

3. Design Steps in More Detail

Each "Step" (any State or Transition) in the above state machine is now discussed in more detail. See Appendix for much more info on the FDA views on these issues. However, the FDA does not discuss states or transitions. For example, they talk about the Design Input, but both as a process and as a state.

3.1. Create Concept *[transition]*

This creates the initial writeup on whatever change is desired (anything from a simple bugfix, to a new feature, to an entirely new program). This should capture input from everyone with an interest (ideally, as many people as possible) on the goals of this project.

Checklist:

- Title.
- Description.
- Name of primary person/institution, entering this.
- Names of persons/institutions with interest or which should be involved.

Additional info on this process and its output:

This writeup can range from extremely simple (for small modifications or bugfixes), to extremely detailed (for large functional changes or new programs). Whether it is detailed enough will be determined in the next process, Create Initial Design. At that time, the assigned person may need to communicate with those who created the Concept to get more information, even modifying the Concept as required. All of this will be Reviewed at the Review of Initial Design.

3.2. Concept *[state]*

All that is required in Checklist.

In this state, the SCR is unattached to any project, and in no phase of development. It is simply one of many possible projects that are awaiting management approval to be entered into the system.

This is a good example of the notion of “activity” of a project. Once this state is reached, the project is technically is the Review Concept transition. But nothing may actually be happening. Management would have to allocate resources and make this a priority before any work will proceed on the next transition. This issue is external to the Design Controls, and is part of the higher level project management process.

3.3. Create Initial Design *[transition]*

The process of evaluating the Concept and creating the Initial Design plan and documentation. This is the real starting point for every SCR, whether a small bugfix or minor feature addition, to a complete new program or product. This is always subject to Review before any further work can occur. This must contain a high level Requirements Specification, Master Plan, Risk Analysis, and Review Schedule.

Checklist:

- Requirements specification.
- Requirements specification captures general goals comprehensively.
- Requirements specification is written unambiguously.
- Master Plan exists and is appropriate.
- Risk Assessment exists and is appropriate.
- Review Schedule exists and is in line with complexity and risks.
- Objectives are in line with user, company, and safety needs.

Additional info on this process and its output:

All of these outputs are initial and tentative. They will very likely change as the project progresses, based on what is learned downstream. Subsequent Reviews will ensure that they have not changed in ways inappropriate for our design control procedure.

Requirements Specification: High-level requirements of project, which can include functional, performance, and interface requirements. For small projects such as simple

bug fixes, this might be detailed and comprehensive. For large projects, this will very high level and broad brush, just the starting point for more refinement in possibly many Design Input/Output cycles.

Master Plan: Roadmap of how the project will proceed. For simple bugfixes, it might show a single pass through Design Input, Design Output, Verification, and Validation. For large projects, it may indicate multiple cycles through Design Input/Output to achieve continued refinement of the problem and solution (e.g., first cycle creates detailed requirements specification, next cycle creates proposed high level architecture, next cycle produces prototypes, etc.).

Risk Assessment: Evaluation of risks associated with this project, and consequences. By risks, we mean the possibility of incorrect operation and danger to the end user or patient. At this point in the project this is used for two purposes: (1) to determine whether the project is in line with user, company, and safety needs, and (2) to determine the necessary Review schedule. For example, the simple addition of an “Add New Beam” button to a program GUI, which simply replicates the function of an already existing button elsewhere in that GUI, may be shown to have very low risk of incorrect operation and danger to the patient. Given the added user functionality, it is in line with user, company, and safety goals. Further, a minimal Review Schedule can be prescribed. On the other hand, the addition of a new dose display to an existing program may be shown to have a higher risk of incorrect operation. However, since the dose display will be used to create more accurate treatment plans, it can still be in line with user, company, and safety goals, as long as the risks are addressed. This would require a more rigorous Review Schedule, and more rigorous Verification and Validation criteria downstream.

Review Schedule: Prescribes which of the optional Reviews will occur. This must be based on the complexity of the project and the risks involved. Simple projects with low risk will have fewer reviews than complex problems with high risk. For example, a simple bug fix involving non-life critical software might schedule only one review at Verification Output, whereas the development of a new program might schedule all reviews in the state machine. Of course, the Review Schedule must be approved in the Review of Initial Design.

3.4. Initial Design *[state]*

All that is required in Checklist.

** NOTICE ** All of this output goes into the Design Input state files, since this is nothing more than the very first Design Input. The only thing special about it is the Checklist. Thus, there are no “Initial Design” state files.

If, instead, we created Initial Design state files, then at the conclusion of Initial Design we would have to copy all of that information to the Design Input files as a starting point. There is no value in this, as there is no reason to ever return to or modify the Initial Design. It is simply the first of possibly many Design phases. It will always be available for review (as is any phase of any state), but we will simply build on this design through multiple phases, growing the Design Input until it is adequate to produce the final Design Output.

3.5. Review (of Initial Design) *[transition]*

Ensure that Checklist requirements were met for both (1) previous state, and (2) transition that created previous state.

REQUIRED for all SCRs, carried out by RDRC.

NOTE: This reviews the Design Input state files, since that is where all of the Initial Design output was recorded. See Initial Design.

3.6. Review Output (of Initial Design) *[state]*

Standard Review Output described in Introduction, resulting in next step being set, either to next transition, or back to a previous transition.

For example, if the Risk Assessment was flawed, the Review Output would record what needs to be modified, new action item(s) would be added to the Initial Design "To Do" list (these could make reference to the issues in the Review Output), and the step would be set back to Create Initial Design. When a new Concept state is reached, another Review will be scheduled.

3.7. Create Design Input *[transition]*

This is the process of creating the Design Input. Eventually the Design Input must provide all the items in the Checklist, but these can occur as a result of multiple cycles back through Design Input, as described in the Master Plan.

Checklist:

- Requirements specification.
- Master Plan.
- Risk Assessment.
- Review Schedule
- High level architecture specification.
- Verification criteria.
- Validation criteria.
- Record of who was involved with each aspect of this activity.

Additional info on this process and its output:

Design input is the starting point for product design, and for each subsequent iterative design phase. The requirements that form the design input establish a basis for performing subsequent design tasks and validating the design. These must eventually reach an engineering level of detail; they are not just concept documents.

Virtually every product has input requirements of three types:

- Functional: what it does, processing of inputs into outputs...
- Performance: speed, accuracy (this could be functional), response time...
- Interface: user I/F plus compatibility with existing components...

Design Input is often arrived at iteratively. Initial Design Input might consist of only the requirements specifications. Then, the Design Process might produce high-level design specifications as the Design Output, along with time and resource estimates (evaluation). These design specifications, combined with the original requirements specifications, become the Design Input of the next development phase. The Design Process might then lead to prototyping, which can lead to a Design Output that prescribes modifications of the Design Input specifications for the next development phase. These iterations will occur as many times as necessary until the Design Input is

translated into a Design Output which conforms to those requirements. All of this must be documented.

The CONTENT of the Design Input will change dramatically depending on the phase of development. The specific needs for a given phase of any project will be spelled out in the Master Plan and the Status file “To Do” list. The latter is modified as required by the previous process that brought us to this state. For example, initially that previous process would be the Design Review of the Concept, but afterward, it could be any later Design Review, or even the developer just deciding they need to cycle back after Design Output.

The FORM of the Design Input information will also vary depending on the phase. Initial requirements specifications might be plain text, whereas high-level design specifications might require flow charts, state machines, or other system diagrams. Even requirements specifications might require images or screen snapshots, or spreadsheets.

The Verification and Validation Criteria should employ fault tree analysis or risk analysis, to demonstrate how they achieve their goals. For example, if a new use of an existing program can be shown to be identical to its current use, and the current use is already being verified and validated elsewhere, then there is no need to verify and validate it here. Or, if two functions of a new program are shown to be independent of one another, then the verification and validation procedures can address each independently, rather than having to address the permutations of their operation. Refer to “Testing RFC” for detailed information on our long term plans.

3.8. Design Input *[state]*

All that is required in Checklist.

3.9. Review (of Design Input) *[transition]*

Ensure that Checklist requirements were met for both (1) previous state, and (2) transition that created previous state.

OPTIONAL.

3.10. Review Output (of Design Input) *[state]*

Standard Review Output described in Introduction, resulting in next step being set, either to next transition, or back to a previous transition.

For example, if the Design Input is inconsistent, or not at a sufficient level of engineering detail (e.g., too much like a concept document), these issues would be recorded in the Review Output, new action items would be added to the Design Input “To Do” list (these would make reference to the issues in the Review Output), and the step would be set back to Create Design Input.

3.11. Design Process *[transition]*

The process of producing the Design Output, employing whatever tools and methods are appropriate. This is everything that needs to occur to “realize” the Design Input requirements for this cycle of design. What happens during this step varies depending on the phase of development, but output can include evaluations, meetings, risk/fault analysis, prototyping, testing, and, in the final design phases, the source code itself.

Checklist:

- Record of who was involved with each aspect of this activity.
- Meet each requirement stated in the Design Input for this cycle.
- Provide documentation to support that each requirement has been met.
- CORP Pair Programming is employed.
- Provide list of possible design output documents (see below).
- Follow guidelines for documentation.
- Follow guidelines for coding.

Additional info on this process and its output:

An item is Design Output if it is a work product, or deliverable item, of a design task listed in the design and development plan, and the item defines, describes, or elaborates an element of the design implementation. Examples include block diagrams, flow charts, software high-level code, and system or subsystem design specifications. The design output in one phase is often part of the design input in subsequent phases. In the final phases, the design output includes the source code, comments in the source code, and any documents in the source code area.

Documenting design output in terms that allow an adequate evaluation of conformance to design input requirements is a significant requirement and design activity. The requirements for Design Output can be separated into two parts:

- Should be defined and documented [expressed] in terms that allow adequate assessment of conformance to design input requirements.
- Should identify the characteristics of the design that are crucial to the safety and proper functioning of the device. ... shall contain or make reference to acceptance criteria and identify those design outputs that are essential for the proper functioning of the device.

Documentation shall also be maintained as appropriate for the complexity or criticality of the problem. Such documentation can include:

- Software source code
- Production and process specifications
- Software machine code (e.g., diskette or master EPROM)
- Work instructions
- Quality assurance specifications and procedures
- Installation and servicing procedures
- Flowcharts or diagrams
- Risk or Fault analysis
- Ad hoc discussion notes
- Meeting notes
- Email exchanges
- Verification activities
- Test procedures and results
- Instructions on use
- QA specs and procedures

CORP Pair Programming means that each programmer shall communicate with their designated partner on a regular basis about all significant design or programming

decisions, and shall document those communications in the SCR HISTORY file. MGMT will communicate with each programmer, or review the HISTORY files, on a regular basis to verify that this communication is occurring.

Guidelines for documentation and coding are described later in this document. We try to minimize such constraints to those that are believed to create a better result in terms of correctness, safety, robustness, and maintainability.

3.12. Design Output *[state]*

All that is required in Checklist.

3.13. Review (of Design Output) *[transition]*

Ensure that Checklist requirements were met for both (1) previous state, and (2) transition that created previous state.

OPTIONAL.

3.14. Review Output (of Design Output) *[state]*

Standard Review Output described in Introduction, resulting in next step being set, either to next transition, or back to a previous transition.

For example, assume that one Design Input requirement was to produce a high level design (architecture) that addresses all known failure modes through a fault tree analysis. During the Review, the fault tree analysis is seen to have missed some possible faults. These missing faults would be recorded in the Review Output, an action item would be added to the Design Output "To Do" list (referring to the Review Output if more information is recorded there), and the step would be set back to Design Process.

However, it is also possible that, as a result of the problems seen in the fault tree analysis, the Review determines that the Design Input requirements are also incorrect (e.g., conflicting goals, or issues were left out). This would also be recorded in the Review Output, with action items being added to the Design Input "To Do" list (referring to the Review Output if necessary). In this case, the next step would be set back to Create Design Input, rather than Design Process.

Now, when back at the Create Design Input transition, the developer would find the To Do list issues regarding the Design Input requirements. Then, when eventually in the Design Process, the developer will find the To Do list issues regarding the flawed fault tree analysis. Thus, nothing from the Review will be forgotten, no matter how much time elapses.

3.15. Verification *[transition]*

The process of producing the Verification Output, employing whatever tools and methods are appropriate. This is driven by the Verification Criteria, which must be specified in the Design Input, but can also employ more general verification approaches.

Checklist:

- Record of who was involved with each aspect of this activity.
- Address each item in the Design Input Verification Criteria
- Address items on "standard" verification criteria.

Additional info on this process and its output:

Verification is a three-pronged approach involving tests, inspections, and analyses. Any approach that establishes conformance with a design input requirement is an acceptable means of verifying the design with respect to that requirement. Complex designs require more and different types of verification activities.

Many of these practices are an integral part of the development process, and are routinely performed by developers. The objective of design controls is to ensure adequate oversight by making verification activities explicit and measuring the thoroughness of their execution.

- Each manufacturer shall establish and maintain procedures for verifying the device design.
- Design verification shall confirm that the design output meets the design input requirements.
- The results of the design verification, including identification of the design, method(s), the date, and the individual(s) performing the verification, shall be documented in the Design History File.

Following are a few examples of verification methods and activities (some of these may seem redundant):

- Walkthroughs
- Fault tree analysis of a process or design.
- Comparison of a design to a previous product having an established history of successful use.
- Static (paper) analysis
- Dynamic (runtime) analysis
- Code and document inspection
- Informal and formal design reviews
- Package integrity tests.

Verification is NOT production testing, which often only tests a small subset of the design. Verification is targeted at the specific project.

Documentation: Some verification methods result in a document by their nature. For example, a failure modes and effects analysis produces a table. Another self-documenting verification method is the traceability matrix. However, many verification activities are simply some sort of structured assessment of the design output relative to the design input, and one must document the completion of the activity by linking it with document signoff procedures.

3.16. Verification Output *[state]*

All that is required in Checklist.

3.17. Review (of Verification Output) *[transition]*

Ensure that Checklist requirements were met for both (1) previous state, and (2) transition that created previous state.

OPTIONAL.

Additional info on this process and its output:

Following is a brief discussion of the relationship of Design Review to Verification and Validation.

In practice, design review, verification, and validation overlap one another, and the relationship among them may be confusing. As a general rule, the sequence is: verification, review, validation, review.

In most cases, verification activities are completed prior to the design review, and the verification results are submitted to the reviewers along with the other design output to be reviewed. Alternatively, some verification activities may be treated as components of the design review, particularly if the verification activity is complex and requires multidisciplinary review.

Similarly, validation typically involves a variety of activities, including a determination that the appropriate verifications and reviews have been completed. Thus, at the conclusion of the validation effort, a review is usually warranted to assure that the validation is complete and adequate.

3.18. Review Output (of Verification Output) *[state]*

Standard Review Output described in Introduction, resulting in next step being set, either to next transition, or back to a previous transition.

See example from Review Output of Design Output for how cycles occur and are documented.

3.19. Validation *[transition]*

The process of producing the Validation Output, employing whatever tools and methods are appropriate.

Checklist:

- Record of who was involved with each aspect of this activity.
- Address each item in the Design Input Validation Criteria
- Address each applicable item in general system validation criteria (this would include fundamental safety issues)

Additional info on this process and its output:

Whereas verification is a detailed examination of aspects of a design at various stages in the development, design validation is a cumulative summation of all efforts to assure that the design will conform with user needs and intended use(s), given expected variations in components, materials, manufacturing processes, and the use environment.

While testing is always a part of validation, additional validation methods are often employed. These include analysis and inspection methods, compilation of relevant scientific literature, provision of historical evidence that similar designs and/or materials are clinically safe, and full clinical investigations or clinical trials.

“Testing” includes normal operation of the complete device; and this phase of the validation program may be completed first to make certain that the device meets the fundamental performance, safety and labeling specifications. Concurrently or afterward, the combined system of hardware and software should be challenged with abnormal inputs and conditions. As appropriate, these inputs and conditions include such items as:

- Operator errors;
- Induced failure of sensors and cables or other interconnects;
- Induced failure of output equipment;
- Exposure to static electricity;
- Power loss and restart;
- Simultaneous inputs or interrupts; and,
- As appropriate, deliberate application of none, low, high, positive, negative, and extremely high input values.

Planning for validation should begin early in the design process. The performance characteristics that are to be assessed should be identified, and validation methods and acceptance criteria should be established. For complex designs, a schedule of validation activities and organizational or individual responsibilities will facilitate maintaining control over the process.

3.20. Validation Output *[state]*

All that is required in Checklist.

3.21. Review (of Validation Output) *[transition]*

Ensure that Checklist requirements were met for both (1) previous state, and (2) transition that created previous state.

REQUIRED for all SCRs.

3.22. Review Output (of Validation Output) *[state]*

Standard Review Output described in Introduction, resulting in next step being set, either to next transition, or back to a previous transition.

See example from Review Output of Design Output for how cycles occur and are documented.

3.23. Design Transfer *[transition]*

The process of getting the verified and validated project to the end user.

Usually this is software, as with a new release or program updates. However, this can also be hardware, or entire system designs, as with the hardware, software, and configuration issues required to produce a high availability distributed CORP system.

Checklist:

- Record of who was involved with each aspect of this activity.
- Ensure that released software is identical to that which was verified/validated.
- Ensure that hardware and system configurations are identical, or equivalent, to those which were verified/validated.

Additional info on this process and its output:

The FDA says each manufacturer shall establish and maintain procedures to ensure that the device design is correctly translated into production specifications. They know that “manufacture” of software is not an issue; the tuff part is not duplicating software, but rather getting the master program correct in the first place.

For software transfer, we employs our release and export process and programs, which guarantee a unique naming scheme for the release tarball, and unique version numbering in all programs. We do not rebuild anything after the final testing, even to just change a name or a CVS/RCS ident. The binary versions we test are the ones we distribute. Any rebuild would result on a different version name on the tarball.

For other aspects, more thought must be given to insuring that the specifications, hardware, software, etc., which are implemented later, are the same as those we verified/validated. This is handled via the system build and final quality assurance procedures.

3.24. Final Product *[state]*

All that is required in Checklist.

4. Implementation of the Design Process

Following are the details by which we will implement the state machine.

4.1. CVS Skeleton Files

Each CVS SCR directory will contain skeleton files for what is required in each possible state. All files are at the same level, to make it simpler to work with.

NOTE: I simplified the names (e.g., Input instead of DesignInput, or InitialReview instead of InitialDesignReview) since they got rather long. It's also easier to talk about them with shorter names. Here is an example (as of 1aug2002).

Concept.Document
InitialReview.Status
InitialReview.Output
Input.Status
Input.Requirements
Input.RiskAnalysis
Input.MasterPlan
Input.ReviewSchedule
Input.Design
Input.Verification
Input.Validation
InputReview.Status
InputReview.Output
Output.Status
Output.Design
OutputReview.Status
OutputReview.Output
Verification.Status
Verification.Output
VerificationReview.Status
VerificationReview.Output
Validation.Status
Validation.Output
ValidationReview.Status
ValidationReview.Output
Final

Final.Output
Final.Status

We will always just work in these files, regardless of what phase we are in. We rely on CVS and tags to snapshot the state of these files during the process (described later), and our tools to allow recreation of the previous files as required.

4.2. CVS State Directories

Each CVS SCR directory will also contain a subdirectory for each state, in which extra, supporting documents for any given state can be placed in these directories. There is no structure imposed on what is below the subdirectory. Here is an example (correct as of 11jul2002).

Concept	InitialDesignReview
DesignInput	DesignInputReview
DesignOutput	DesignOutputReview
Verification	VerificationReview
Validation	ValidationReview
DesignTransfer	
FinalProduct	

We will always just work in these directories, regardless of what phase we are in. We rely on CVS and tags to snapshot the state of the files during the process (described later), and our tools to allow recreation of the previous files as required.

4.3. Support Programs

4.3.1. StateChange Program

When a state is assumed to be stable (that is, the previous transition process has finished), the person responsible for reaching that state runs the “StateChange” program.

Functional goals:

- Keep current state
- Keep history of states, date each ended.
- Keep states and dates so they can be auto parsed to generate picture of what happened when.
- Ensure that reviews occur according to review schedule
- Ensure CVS tag of states
- Ensure tags are such that we can readily extract previous states with automatic program.

Implementation details

- Gets current state from StateHistory file.
  - StateHistory file lists states in reverse chronological order
  - Must checkout last known good version of this (based on tag names and dates?), to avoid problems if user modifies it.
- Tells user this current state.

- Asks user for the next state, or determines this from state machine.
- Checks to see if a Review is required. If so, forces next state to that Review, and notifies MGMT.
- Commits all files in this SCR directory (or tells the user to do so).
- Generates TAG name with today's date (to serve as "end date")
- Tags all files in SCR directory with TAG.
- Enters next state at top of StateHistory file.

4.3.2. UpdateViews Program

The "UpdateViews" program provides the ability to create the previous state of the SCR directory as any of the following "views":

1. Create the files as they were at the end of each state.
2. Create the files as they were before and after each Review.
3. Create the files as they were before any cycle back.

Within a given SCR directory, the top level files and directories are always the current working versions. The state history is maintained within CVS through tagging, such that we can recreate these previous states as desired.

Three types of tag will be provided by our tools:

- Tagging at the end of each state (mandatory).
- Tagging before and after any Review (mandatory).
- Tagging at any cycle back (optional, developer discretion).

We run UpdateViews, telling it which view we want, and it will checkout those versions of the SCR directory into a subdirectory with an appropriate view name (see below).

4.3.3. Example of UpdateViews Use

Assume an SCR in which the following events occurred:

- The first Design Output Review pushed it back to Design Input.
- The next Design Output Review pushed it back to Design Output.
- The next Design Output Review was successful.
- The first Validation Review pushed it back to Design Output.
- The next Validation Review was successful.

After running "UpdateViews Review", the SCR directory would contain a "reviews" directory containing the following directories, each containing a complete snapshot of the entire SCR directory at those Review points in development:

Reviews

- 01.OutputReview.Pre
- 01.OutputReview.Post
- 02.OutputReview.Pre
- 02.OutputReview.Post
- 03.OutputReview.Pre
- 03.OutputReview.Post
- 01.ValidationReview.Pre
- 01.ValidationReview.Post

- 04.OutputReview.Pre
- 04.OutputReview.Post
- 02.ValidationReview.Pre
- 02.ValidationReview.Post

After running “UpdateViews State”, the SCR directory would contain a “states” directory containing the following directories, each containing a complete snapshot of the entire SCR directory at those states in development (this clearly shows all the cycles which were due to Review results):

States

- 01.Concept
- 01.Input
- 01.Output
- 01.OutputReview
- 02.Input
- 02.Output
- 02.OutputReview
- 03.Output
- 03.OutputReview
- 03.Verification
- 03.Validation
- 03.ValidationReview
- 04.Output
- 04.OutputReview
- 04.Verification
- 04.Validation
- 04.ValidationReview
- 04.Transfer

A similar thing would occur for “UpdateViews Phase”, where the SCR directory would contain a “phases” directory, containing snapshots of the SCR directory at each phase (before each cycle back).

4.3.4. Example of Review

Here’s an example of a non-trivial Review, discovering problems at 3 previous states.

Assume that the Verification Output Review (which spanned multiple days as issues were discussed and thought out) determines that certain verification procedures were not carried out as specified. In addition, the review determines that certain design input specifications were ambiguous, and need to be changed. Further, the review determines that part of the implementation of the valid input specifications is flawed.

This requires a cycle all the way back to Create Design Input, but also requires that we record the information needed at each of the subsequent steps. At this point in the review, the Review Output Status file would look like this:

Review Output Status file:

To Do:

Design Input: Requirements spec change. See file newspec.

Design Output: Implementation problem. See file codeproblem.

Verification Output: Procedure 2 not done right. See file proc2prob.

The files that these notes refer to (newspec, codeproblem, and proc2prob) were created in the Review Output directory as the Review proceeded, to record all the necessary details, and were added/committed to the CVS repository. With the one-line action items in the Status file, these issues will not be forgotten at the end of the Review.

When the Review is over, the reviewer must now address each of these “To Do” items. For each one, it must be recorded in the appropriate state’s Status file, and when done, it is moved to the “Done” section of the Review Output Status file. Remember, no state can be done and stable if items remain in its Status file “To Do” or “In Progress” sections. When done, the following Status files will now have these entries (note that each entry just refers to the details file which already exists in the VerificationReviewOutput directory):

Design Input Status file:

To Do:

Requirements spec change. See VerificationReviewOutput/newspec.

Design Output Status file:

To Do:

Implementation problem. See VerificationReviewOutput/codeproblem.

Verification Output Status file:

To Do:

Procedure 2 not done right. See VerificationReviewOutput/proc2prob.

When all items are gone from the To Do list, the Review is done, and the reviewer runs StateChange, setting the step back to Create Design Input. When the developer encounters this state and each subsequent state, these review items will be clearly listed. In this way, review items will not be forgotten, no matter how much time elapses between the review and subsequent work.

Once the new Design Input, Design Output, and Verification Output are complete, and it is time for the next Verification Review, the reviewer can always look back at the original Status files for each state that was cycled back through, by selecting the versions with the previous Review tag. Remember, when StateChange was run when the previous Review was done, it tagged ALL other entries in this SCR dir with that Review tag. Thus, nothing from the previous review can be lost or forgotten.

5. Additional Information

5.1. Design Changes

5.1.1. What is it?

Each manufacturer shall establish and maintain procedures for the identification, documentation, validation or where appropriate verification, review, and approval of design changes before their implementation.

5.1.2. CORP Implementation

An SCR is assigned to any individual for work only via MGMT. Further, no SCR can get beyond the initial Design Input state without a Review. This provides two checkpoints to ensure that changes are approved, and further, that they are following the Design Control Process. This is all recorded in the DHF.

5.2. Design History File: DHF

5.2.1. What is it?

Each manufacturer shall establish and maintain a DHF for each type of device. The DHF shall contain or reference the records necessary to demonstrate that the design was developed in accordance with the approved design plan and the requirements of this part.

5.2.2. CORP Implementation

For us, this includes everything contained in our CVS source tree, CVS/SCR tree, and SCR database, and any other information these refer to (e.g., lab notebooks, file folders, test plots, email bags, etc.). This other information need not be consolidated at one location. It just needs to be referenced, and available, and a procedure must be in place to ensure it exists. However, the more we can consolidate this information into the SCR area (e.g., moving email bags into the SCR area), the better off we all will be.

Our CVS SCR approach with TAGSYS ensures that we have records of the SCR state at each design phase. This is all managed by the revision control system. We have tools that will allow extraction of each document associated with each phase, for later review.

5.3. Device Master Record: DMR

5.3.1. What is it?

Many of the design output documents are documents that directly form part of the DMR. The remaining DMR documents are created by quality assurance, production engineering, process engineering, technical writing, installation, servicing, etc., using design output data and information. For example, the finished device final-test methods and some installation and/or servicing test methods and data forms may be derived from the design verification protocol(s). When all of these design and documentation activities are completed, the DMR is complete. When the DMR is complete and initial production units, including packaging, meets all specifications, the total finished design output exists.

5.3.2. CORP Implementation

This goes at a higher level in our company wide process, as it uses information from all over the place. It would refer to the various outputs of the Design Control Procedures, as well as many other things.

5.4. Guidelines for Documentation

All of the required SCR documents, and all of the required information within, must be plain text (ASCII) files. However, these can refer to other documents of any type (images, web pages, word documents, spreadsheets, etc.) for additional supportive information.

5.5. Guidelines for Programming

Avoid #ifdefs in favor of runtime conditionals, unless there are good arguments for the #ifdefs, such as compiling on multiple architectures;but even then, other solutions exist, such as creating macros or functions in which the #ifdefs are localized.

Avoid leaving large amounts of old code in-line and commented our or #ifdefed out if it obscures the understandability of the code. If you must keep it in the file, then move it to the end of the file in a large block that is simply #ifdefed out (#ifdef NEVER).

In general, in order to keep chunks of old code around for future reference, use one of the following approaches. In any of these approaches, put newer code at the top, with a date and your initials and comment about why it is there. When appropriate, reference the code snippet from within the main code:

1. Create a file of the same name, with a .notes extension (e.g., if the working file is prog.c, old code would go in prog.c.notes).
2. Create a single file in the directory, called snippets, in which code from all other files goes, with appropriate comments about where it came from.

