Software Design Specification

CxOne Guide

CxGuide_SoftwareDesignSpecification.doc

November 4, 2002

Construx
SOFTWARE

Advancing the Art and
Science of Commercial
Software Engineering

Contents

Copyright Notice

For further information or support visit www.construx.com.

1 Introduction

This CxOne user guide discusses how to use ***CxTemp_SoftwareDesignSpecification*** to create a software design. It provides additional detail, insight, and examples to the outline and comment text of the template. See ***CxStand_Design***, ***CxCheck_Design,*** and the other design checklists for further detail on how CxOne defines design activities and recognizes design quality. See ***CxGuide_Architecture*** and other architecture artifacts for more in depth support when creating architectures.

1.1 Design Template Purpose

One size fits all is not the way to approach software design. Design works best when the design process and artifacts are determined by expert engineers as part of early project planning. CxOne provides *CxTemp_SoftwareDesignSpecificatio*n only as a starting point, it should be customized and augmented as appropriate.

As noted in the template introduction, most projects should not rely on a single design document, or feel compelled to closely follow *CxTemp_SoftwareDesignSpecificatio*n. Even though this template is laid out in manner that suggests support of an entire project's design from start to finish, it is expected that most projects will have a family of customized design artifacts. *CxTemp_SoftwareDesignSpecificatio*n can serve as a blueprint for creating that family of artifacts, with different document sections broken out into a group of documents.

1.2 When to Use

The Software Design Specification is a good starting point when there are no readily available materials for reuse from previous projects. Completed design artifacts from previous projects that cover similar issues are an excellent place start design work from.

Design is normally started in parallel with the initial project planning. The architecture along with high and low level designs are then developed and refined throughout the project. When design activities occur will be highly dependent on the project lifecycle and project plan.

1.3 Capturing Design Information

*CxTemp_SoftwareDesignSpecificatio*n focuses on providing structure and content for design artifacts, but does not describe techniques for capturing design information itself. Design information is normally captured in text form, diagrams, or a combination of both. As discussed in *CxStand_Design*, CxOne assume engineers will rely on proven software engineering techniques like UML for capturing designs. Instead of adding new techniques for capturing design information, CxOne focuses on materials that assist getting started on designs, defining design processes, packaging designs, and ensuring the quality of those designs.

There are many sections in this guide that describe placeholders for design descriptions; it is assumed the engineer will use techniques familiar to them for capturing this information.

1.4 How to Use the Design Template

1.4.1 Packaging the Design

The packaging of the design will vary widely depending on the nature and scope of a project. Small projects may have a single document covering the high level and detailed designs. Larger projects may have a hierarchical family of documents and diagrams that cover information in the specification. Each project should organize and package content to best meet the needs of the project.

One good way to start is to create a single top-level document and then break out elements into separate documents based on complexity or CM needs.

As an example, a common way to break *CxTemp_SoftwareDesignSpecification* into a family of design documents would be to package sections from the template into documents:

System Overview / Design Process – template sections 1-2

Architecture – template section 3 (also see *CxTemp_Architecture*)

High Level Design Documents – for each major section under template section 4

Low Level Design Documents – for each major section under template section 5

User Interface Design Document – for each major section under template section 6

1.4.2 Trade-offs and Alternatives

In design documents is always a good idea to document the thinking that goes into significant trade-offs or alternative decisions that were considered but discarded.

1.4.3 Pointers in the Design

Due to the potential for a family of artifacts and redundancy of information in the design, pointers should be used liberally. Pointers allow an easy mechanism to include detailed information by reference. Pointers can also be used to cross reference information within an artifact. This can be useful when creating overlapping views of information.

1.4.4 When to Customize

As with all CxOne materials, design materials encompass critical issues that should be addressed for most projects. A nominal path is provided by the templates and supporting material that will work well for most projects. To gain greater efficiency the exact organization, content, and presentation can be tailored to best meet project needs.

2 Design Specification Content Guide

This section is the detailed guide for *CxTemp_SoftwareDesignSpecification.* The sections here match the template organization.

2.1 Introduction

If appropriate use this space to summarize any background, context, or audience targeting issues related to understanding this design specification or related design artifacts.

2.1.1 System Overview

This section provides an entry point for understanding the system and the environment it will operate in. This section may provide brief high level descriptions of system structure, functionality, interactions with external systems, system issues, etc. A single diagram that shows major system components and the external systems they interact with is often useful.

If the system is complex, this is a good candidate for splitting into a separate document. Much of the work in this area is often done as part of the requirements analysis, which also works well if this is a separate document.

The difference between the system overview and the architecture sections is one of formality, scope, and intent. They are both top-level views of the system, but the architecture view is more rigorous, precise, and detailed than the system overview needs to be. The system overview often provides a larger view of the system including external systems whereas software architectures focus more on the software. Audience is another good differentiator. The system overview should be understandable by a domain expert, while the architecture is targeted at software engineers. See the *CxGuide_Architecture* for details software architectures.

2.1.2 Design Map

A table is often useful to capture all major design artifacts and summarize them.

2.1.3 Supporting Material

(Optional) - When appropriate, provide references for any other pertinent documents such as:

- Related and/or companion documents
- Prerequisite documents
- Documents which provide background and/or context for this document
- Documents that result from this document (e.g. a test plan or a development plan)

2.1.4 Definitions and Acronyms

(Optional) – CxOne encourages centralization of definitions and acronyms to support defining items in only one place.

2.2 Design Considerations

This section is used to formally set the stage for a design.

2.2.1 Assumptions

Assumptions, background, and dependencies for the system and project are probably already captured elsewhere. This section shouldn't repeat those issues verbatim. Instead it should deal with previously stated issues in the context of design, if appropriate, or bring up new issues that have only come up as part of the design.

2.2.2 Constraints

Drivers are goals the system must achieve while constraints are external limiting factors. Like assumptions and dependencies, these are likely already captured elsewhere, so should be repeated here only if extra detail or specific context is added.

2.2.3 System Environment

If it has not already been covered in other project documentation, describe the required system environment including hardware and software for the operations environment, and any other environment if necessary (e.g., discuss development environment if different). If these issues have been covered elsewhere, but more detail is necessary to fully understand the design perspective, cover that additional detail here.

If multiple disparate environments are supported cover any global issues relating to this.

2.2.4 Design Methodology

Briefly describe the methods, processes, approaches, techniques, or conventions used for this software design. If one or more formal/published methods were adopted or adapted, include a reference to a more detailed description of these methods.

Describe any issues outside the scope of the requirements that affect the creation or details of the design. Areas for consideration include:

- Externally imposed processes or policies
- Global engineering trade-offs
- Algorithm or design pattern choices for the system
- Plans for requirements traceability, testing, or maintenance

Issues discussed here should impact multiple areas of the system. Elements that impact only a specific component should be discussed as part of more detailed designs for the component.

2.2.5 Risks and Volatile Areas

Discuss the most likely sources of change and risks (new requirements, technology, etc.) that would impact the design of the system. If appropriate, describe how the system will be designed to allow timely response to changes or what the contingency path is for changes.

2.3 Architecture

See ***CxTemp_Architecture*** and ***CxGuide_Architecture*** for support in creating software architectures.

2.4 High Level Design

Describe the behavior of each major element called out in the architecture or other upstream high-level designs. High-level design should focus on describing behavior through one or more views of the system. Each view can make use of either or both textual descriptions and models. Depending on the needs of a design it there may be several distinct levels of high-level design. Decomposition of the subsystem will usually fall out naturally from the views.

Areas for consideration include:

- Interfaces
- Data flow
- Key algorithms and processing
- States
- Data management (storage, distribution, persistence)
- Concurrency and synchronization
- Reuse

2.4.1 View/Model 1..n

Describe a view or model of a software element. Depending on the needs of a design it may make sense to describe design elements separately, or in groups, or in various combinations.

It can often be difficult to determine when to include information in this section verse the low-level design section that follows later. Final decisions on this are left to the designer. The general guideline is that high level design forms the bridge between the architecture and low level design and focuses on the problem domain design.

Common views include:

- Interaction models (sequence or collaboration models)
- High level class and interface diagrams
- Database schemas

2.5 Low Level Design

Describe the design details that will allow a module to be implemented. This section is often best organized as a series of individual documents for each group of related modules (this often falls out naturally from the architecture or high-level design). There may also be a hierarchy of low-level designs, unlike the flat list implied by the template.

It can often be difficult to determine when to include information in this section verse directly in the code. Final decisions on this are left to the designer, but in general low level design should provides sufficient details to enable another engineer to develop the code. Depending on the project size, team size, and technology used it can often be productive to package significant portions of detailed design in source code file, class, and interface comments.

2.5.1 Module 1..n

Describe each software module including where appropriate diagrams of detailed component structure, behavior, information/control flow, etc.

Areas for consideration include:

- Classification (module, class, package, function, file)
- The specific purpose and semantic meaning of the component
- Responsibilities and/or behavior (roles, services, tasks)
- Constraints (timing, storage, state, interaction rules)
- Interactions (unit is used by, unit uses)
- Resources (managed, affected, needed, deadlock potentials)
- Processing (algorithms, concurrency, states, exception handling)
- Data (types, ranges)

Common outputs include:

- Detailed class and interface diagrams or descriptions
- Psuedo code
- State diagrams
- Expansion of views from high level design

2.6 User Interface Design

During the design activities, the design of the user interface should occur. This section should contain user interface mockups and screen shots. Common look and feel details such as menus, popup menus, toolbars, status bar, title bars, drag and drop mouse behavior should be detailed. Illustrations of major screens should be created. The designer should strive for consistency of UI behavior.

The user interface design activity straddles the boundary between requirements gathering and design. On one hand the user interface has requirements as to how screens should look and behave, and as such, should be combined with other requirements. On the other hand the user

interface is highly dependent on the technology selected for the project, which is an application of the business requirements. As such, the user interface specification should be part of the design artifacts. CxOne defines the creation of user interface specifications as a design activity and should be managed separate from requirement artifacts. Any discussion of user interfaces in the requirement artifacts should be done if it is general enough or as a design note if it is for a specific screen.

2.6.1 Application Control

This section details the user interface functionality common to all application screens. Common look and feel details such as menus, popup menus, toolbars, status bar, title bars, drag and drop mouse behavior should be detailed here.

2.6.2 Screen 1..n

This section details the design of the major screens in the application. A screen transition diagram or table could be created to detail the control flow through the screens.