Software Design Specification

by Brad Appleton <bradapp@enteract.com>

http://www.enteract.com/~bradapp

Copyright © 1994-1997 by Bradford D. Appleton

Permission is hereby granted to make and distribute verbatim copies of this document provided the copyright notice and this permission notice are preserved on all copies.

Overview

The following is an attempt to put together a complete, yet reasonably flexible template for the specification of software designs. Wherever possible, I have tried to provide guidelines (instead of prescribing requirements) for the contents of various sections and subsections of the document. Some may prefer to require more detailed subsections of a particular section, choosing one or more of the subsection topics from the list of guidelines provided. In this sense, this document is really a template for a template.

In devising this template, I have gleaned information from many sources, including various texts on Software Engineering (Pressman, Sommerville, and Van Vliet), Object-Oriented Development (Booch, Rumbaugh, Berard, and Wirfs-Brock), various SEI reports, DoD-Std and Mil-Std documentation requirements (2167/2167A), and IEEE documentation standards (particularly IEEE-1016 for software designs, and IEEE-830 for software requirements). I have made every effort not to assume or impose a particular software development methodology or paradigm, and to place more emphasis on content than on format.

It is my desire that a completed Software Design Specification meet the following criteria:

It should be able to adequately serve as training material for new project members, imparting to them enough information and understanding about the project implementation, so that they are able to understand what is being said in design meetings, and won't feel as if they are drowning when they are first asked to create or modify source code.

It should serve as "objective evidence" that the designers and/or implementors are following through on their commitment to implement the functionality described in the requirements specification.

It needs to be as detailed as possible, while at the same time not imposing too much of a burden on the designers and/or implementors that it becomes overly difficult to create or maintain.

Please note that there are no sections in this document for describing administrative or business duties, or for proposing plans or schedules for testing or development. The sections in this document are concerned solely with the design of the software. While there are places in this document where it is appropriate to discuss the effects of such plans on the software design, it is this author's opinion that most of the details concerning such plans belong in one or more separate documents.

Document Outline

Here is the outline of the Software Design Specification template. Please note that many parts of the document may be extracted automatically from other sources and/or may be contained in other, smaller documents. What follows is just one suggested outline format to use when attempting to pres-

ent the architecture and design of the entire system as one single document. This by no means implies that it literally is a single document (that would not be my personal preference):

- Introduction
- System Overview
- Design Considerations
  - Assumptions and Dependencies
  - General Constraints
  - Goals and Guidelines
  - Development Methods
- Architectural Strategies
  - strategy-1 name or description
  - strategy-2 name or description
  - ...
- System Architecture
  - component-1 name or description
  - component-2 name or description
  - ...
- Policies and Tactics
  - policy/tactic-1 name or description
  - policy/tactic-2 name or description
  - ...
- Detailed System Design
  - module-1 name or description
  - module-2 name or description
  - ...
- Glossary
- Bibliography

The above outline is by no means exclusive. A particular numbering scheme is not necessarily required and you are more than welcome to add your own sections or subsections where you feel they are appropriate. In particular you may wish to move the bibliography and glossary to the beginning of the document instead of placing them at the end.

The same template is intended to be used for both high-level design and low-level design. The design document used for high-level design is a "living document" in that it gradually evolves to include low-level design details (although perhaps the "Detailed Design" section may not yet be appropriate at the high-level design phase).

The ordering of the sections in this document attempts to correspond to the order in which issues are addressed and in which decisions are made during the actual design process. Of course it is understood that software designs frequently (and often fortunately) don't always proceed in this order (or in any linear, or even predictable order). However, it is useful for the purpose of comprehending the design of the system to present them as if they did. Frequently, one of the best ways to document a project's design is to keep a detailed project journal, log, or diary of issues as they are mulled over and bandied about and to record the decisions made (and the reasons why) in the journal. Unfortunately, the journal format is not usually organized the way most people would like it for a formal review. In

such cases, for the purpose of review, the journal can be condensed and/or portions of it extracted and reorganized according to this outline. However, if this is done then you need to choose whether to update and maintain the design document in the journal format, or the formal review format. It is not advisable to try and maintain the design document in both formats. (If you have an automated method of converting the journal into a formal document, then this problem is solved.)

Detailed Document Description

This section describes the contents of each section of the Software Design Specification.

Introduction

Provide an overview of the entire document:

- Describe the purpose of this document
- Describe the scope of this document
- Describe this document's intended audience
- Identify the system/product using any applicable names and/or version numbers.
- 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)
- Define any important terms, acronyms, or abbreviations
- Summarize (or give an abstract for) the contents of this document.

Note:

For the remaining sections of this document, it is conceivable (and perhaps even desirable) that one or more of the section topics are discussed in a separate design document within the project. For each section where such a document exists, a reference to the appropriate design document is all that is necessary. All such external (or fragmented) design documents should probably be provided with this document at any design reviews.

System Overview

Provide a general description of the software system including its functionality and matters related to the overall system and its design (perhaps including a discussion of the basic design approach or organization). Feel free to split this discussion up into subsections (and subsubsections, etc ...).

Design Considerations

This section describes many of the issues which need to be addressed or resolved before attempting to devise a complete design solution.

Assumptions and Dependencies

Describe any assumptions or dependencies regarding the software and its use. These may concern such issues as:

Related software or hardware

Operating systems

End-user characteristics

Possible and/or probable changes in functionality

General Constraints

Describe any global limitations or constraints that have a significant impact on the design of the system's software (and describe the associated impact). Such constraints may be imposed by any of the following (the list is not exhaustive):

- Hardware or software environment
- End-user environment
- Availability or volatility of resources
- Standards compliance
- Interoperability requirements
- Interface/protocol requirements
- Data repository and distribution requirements
- Security requirements (or other such regulations)
- Memory and other capacity limitations
- Performance requirements
- Network communications
- Verification and validation requirements (testing)
- Other means of addressing quality goals
- Other requirements described in the requirements specification

Goals and Guidelines

Describe any goals, guidelines, principles, or priorities which dominate or embody the design of the system's software. Such goals might be:

- The KISS principle ("Keep it simple stupid!")
- Emphasis on speed versus memory use
- Working, looking, or "feeling" like an existing product

For each such goal or guideline, unless it is implicitly obvious, describe the reason for its desirability. Feel free to state and describe each goal in its own subsubsection if you wish.

Development Methods

Briefly describe the method or approach used for this software design. If one or more formal/published methods were adopted or adapted, then include a reference to a more detailed description of these methods. If several methods were seriously considered, then each such method should be mentioned, along with a brief explanation of why all or part of it was used or not used.

Architectural Strategies

Describe any design decisions and/or strategies that affect the overall organization of the system and its higher-level structures. These strategies should provide insight into the key abstractions and mechanisms used in the system architecture. Describe the reasoning employed for each decision and/or strategy (possibly referring to previously stated design goals and principles) and how any design goals or priorities were balanced or traded-off. Such decisions might concern (but are not limited to) things like the following:

- Use of a particular type of product (programming language, database, library, etc. ...)
- Reuse of existing software components to implement various parts/features of the system
- Future plans for extending or enhancing the software
- User interface paradigms (or system input and output models)
- Hardware and/or software interface paradigms
- Error detection and recovery
- Memory management policies
- External databases and/or data storage management and persistence
- Distributed data or control over a network
- Generalized approaches to control
- Concurrency and synchronization
- Communication mechanisms
- Management of other resources

Each significant strategy employed should probably be discussed in its own subsection, or (if it is complex enough) in a separate design document (with an appropriate reference here of course). Make sure that when describing a design decision that you also discuss any other significant alternatives that were considered, and your reasons for rejecting them (as well as your reasons for accepting the alternative you finally chose). Sometimes it will be most effective to employ the "pattern format" for describing a strategy.

System Architecture

This section should provide a high-level overview of how the functionality and responsibilities of the system were partitioned and then assigned to subsystems or components. Don't go into too much detail about the individual components themselves (there is a subsequent section for detailed component descriptions). The main purpose here is to gain a general understanding of how and why the

system was decomposed, and how the individual parts work together to provide the desired functionality.

At the top-most level, describe the major responsibilities that the software must undertake and the various roles that the system (or portions of the system) must play. Describe how the system was broken down into its components/subsystems (identifying each top-level component/subsystem and the roles/responsibilities assigned to it). Describe how the higher-level components collaborate with each other in order to achieve the required results. Don't forget to provide some sort of rationale for choosing this particular decomposition of the system (perhaps discussing other proposed decompositions and why they were rejected). Feel free to make use of design patterns, either in describing parts of the architecture (in pattern format), or for referring to elements of the architecture that employ them.

If there are any diagrams, models, flowcharts, documented scenarios or use-cases of the system behavior and/or structure, they may be included here (unless you feel they are complex enough to merit being placed in the Detailed System Design section). Diagrams that describe a particular component or subsystem should be included within the particular subsection that describes that component or subsystem.

Note:

This section (and its subsections) really applies only to newly developed (or yet-to-be developed) portions of the system. If there are parts of the system that already existed before this development effort began, then you only need to describe the pre-existing parts that the new parts of the system depend upon, and only in enough detail sufficient to describe the relationships and interactions between the old parts and the new parts. Pre-existing parts that are modified or enhanced need to be described only to the extent that is necessary for the reader to gain a sufficient understanding of the nature of the changes that were made.

Subsystem Architecture

If a particular component is one which merits a more detailed discussion than what was presented in the System Architecture section, provide that more detailed discussion in a subsection of the System Architecture section (or it may even be more appropriate to describe the component in its own design document). If necessary, describe how the component was further divided into subcomponents, and the relationships and interactions between the subcomponents (similar to what was done for top-level components in the System Architecture section).

If any subcomponents are also deemed to merit further discussion, then describe them in a separate subsection of this section (and in a similar fashion). Proceed to go into as many levels/subsections of discussion as needed in order for the reader to gain a high-level understanding of the entire system or subsystem (but remember to leave the gory details for the Detailed System Design section).

If this component is very large and/or complex, you may want to consider documenting its design in a separate document and simply including a reference to it in this section. If this is the option you choose, the design document for this component should have an organizational format that is very similar (if not identical to) this document.

Policies and Tactics

Describe any design policies and/or tactics that do not have sweeping architectural implications (meaning they would not significantly affect the overall organization of the system and its high-level structures), but which nonetheless affect the details of the interface and/or implementation of various aspects of the system. Such decisions might concern (but are not limited to) things like the following:

- Choice of which specific product to use (compiler, interpreter, database, library, etc. ...)
- Engineering trade-offs
- Coding guidelines and conventions
- The protocol of one or more subsystems, modules, or subroutines
- The choice of a particular algorithm or programming idiom (design pattern) to implement portions of the system's functionality
- Plans for ensuring requirements traceability
- Plans for testing the software
- Plans for maintaining the software
- Interfaces for end-users, software, hardware, and communications
- Hierarchical organization of the source code into its physical components (files and directories).
- How to build and/or generate the system's deliverables (how to compile, link, load, etc. ...)

Each particular policy or set of tactics employed should probably be discussed in its own subsection, or (if it is large or complex enough) in a separate design document (with an appropriate reference here of course). Make sure that when describing a design decision that you also discuss any other significant alternatives that were considered, and your reasons for rejecting them (as well as your reasons for accepting the alternative you finally chose). For this reason, it may frequently be convenient to use one of the more popular "pattern formats" to describe a given tactic.

For this particular section, it may become difficult to decide whether a particular policy or set of tactics should be discussed in this section, or in the System Architecture section, or in the Detailed System Design section for the appropriate component. You will have to use your own "best" judgement to decide this. There will usually be some global policies and tactics that should be discussed here, but decisions about interfaces, algorithms, and/or data structures might be more appropriately discussed in the same (sub)section as its corresponding software component in one of these other sections.

Detailed System Design

Most components described in the System Architecture section will require a more detailed discussion. Other lower-level components and subcomponents may need to be described as well. Each subsection of this section will refer to or contain a detailed description of a system software component. The discussion provided should cover the following software component attributes:

Classification

The kind of component, such as a subsystem, module, class, package, function, file, etc. ....

Definition

The specific purpose and semantic meaning of the component. This may need to refer back to the requirements specification.

Responsibilities

The primary responsibilities and/or behavior of this component. What does this component accomplish? What roles does it play? What kinds of services does it provide to its clients? For some components, this may need to refer back to the requirements specification.

Constraints

Any relevant assumptions, limitations, or constraints for this component. This should include constraints on timing, storage, or component state, and might include rules for interacting with this component (encompassing preconditions, postconditions, invariants, other constraints on input or output values and local or global values, data formats and data access, synchronization, exceptions, etc.)

Composition

A description of the use and meaning of the subcomponents that are a part of this component.

Uses/Interactions

A description of this components collaborations with other components. What other components is this entity used by? What other components does this entity use (this would include any side-effects this entity might have on other parts of the system)? This concerns the method of interaction as well as the interaction itself. Object-oriented designs should include a description of any known or anticipated subclasses, superclasses, and metaclasses.

Resources

A description of any and all resources that are managed, affected, or needed by this entity. Resources are entities external to the design such as memory, processors, printers, databases, or a software library. This should include a discussion of any possible race conditions and/or deadlock situations, and how they might be resolved.

Processing

A description of precisely how this components goes about performing the duties necessary to fulfill its responsibilities. This should encompass a description of any algorithms used; changes of state; relevant time or space complexity; concurrency; methods of creation, initialization, and cleanup; and handling of exceptional conditions.

Interface/Exports

The set of services (resources, data, types, constants, subroutines, and exceptions) that are provided by this component. The precise definition or declaration of each such element should be present, along with comments or annotations describing the meanings of values, parameters, etc. .... For each service element described, include (or provide a reference) in its discussion a description of its important software component attributes (Classification, Definition, Responsibilities, Constraints, Composition, Uses, Resources, Processing, and Interface).

Much of the information that appears in this section is not necessarily expected to be kept separate from the source code. In fact, much of the information can be gleaned from the source itself (especially if it is adequately commented). This section should not copy or reproduce information that can be easily obtained from reading the source code (this would be an unwanted and unnecessary duplication of effort and would be very difficult to keep up-to-date). It is recommended that most of this information be contained in the source (with appropriate comments for each component, subsystem,

module, and subroutine). Hence, it is expected that this section will largely consist of references to or excerpts of annotated diagrams and source code. Any referenced diagrams or source code excerpts should be provided at any design reviews.

Detailed Subsystem Design

Provide a detailed description of this software component (or a reference to such a description). Complex diagrams showing the details of component structure, behavior, or information/control flow may be included in the subsection devoted to that particular component (although, unless they are very large or complex, some of these diagrams might be more appropriately included in the System Architecture section. The description should cover any applicable software component attributes (some of which may be adequately described solely by a source code declaration or excerpt).

Glossary

An ordered list of defined terms and concepts used throughout the document.

Bibliography

A list of referenced and/or related publications.