Software Documentation Guidelines

Dr Fritz Solms

May 5, 2014

Contents

1 Overview

This document provides some guidelines for the documentation which can be used when generated documentation for modern software development projects.

All documents should be continuously updated and managed within the version control system. Published documents (e.g. for each demo and and documents made available for the client or on project day) are snapshots in time of these dynamic documents and should have a date and a version number.

1.1 Assumptions

The documentation guidelines make the assumption that the software development process followed is a hybrid process which has

1. an up-front software architecture requirements and design phase, followed by
2. an iterative/agile development process within which the required functionality is developed iteratively (e.g. using the Scrum development process).

1.2 Overview of documents and when they are created

Figure 1 provides an overview of the documents which are generated by the recommended hybrid development process. Within such a process one first specifies the *vision and scope* of the system, then captures the architectural requirements and subsequently generates an initial version of the software architecture specification.

Subsequently an agile methodology (e.g. Scrum) is used to iteratively develop each use case or system functionality. For each use case the detailed functional requirements, application design, implementation, functional testing are done. This leads to additions to the functional requirements document, application design document, functional testing document and user manual with the relevant aspects of each use case added to the respective manuals as the use case is developed.

Following the hybrid software development process the first document which is generated is a *Vision and Scope* document which specifies

- what the client wants to achieve with the software system and
- the high-level scope of the proposed software system.

Next the architectural requirements are captured in a *Architecture Requirements Specification* which contains the

- the access and integration requirements,
- the quality requirements (e.g. scalability, performance, reliability, security, ...),
- and potetially a set of architectural constraints specified by the client.

Once the software architecture has been specified, the application development can be done within an agile software development methodology like *Scrum*. For each iteration the functional or application requirements are elicited and added to the *Functional Requirements Specification* document, the application design for the functionality is added to the *Application Design* document,

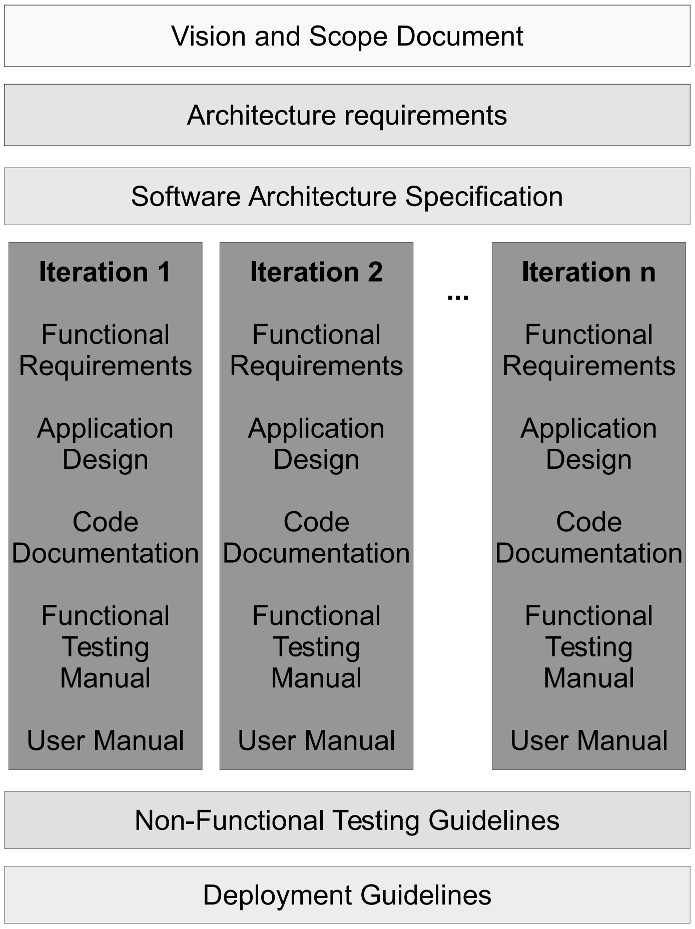

Figure 1: Overview of the documentation for a software project.

the functional testing instructions to the *Fucntional Testing* document, and the instructions on how to use the functionality developed in that iteration is added to the user's manual. If necessary, complex aspects of the code can be explained within code documentation which are embedded in the source files.

Finally the software product will be shipped with the above documents as well as *Non-Functional Testing* and *Deployment* guidelines.

Although most of the documents are incremental documents to which you will add with every new iteration, the documents you will have on project day should include

1. Vision and Scope
2. Architecture Requirements Specification
3. Architecture Specification
4. Functional Requirements and Application Design Specification

5. Functional Testing Guidelines
6. Non-Functional Testing Guidelines
7. Deployment Guidelines
8. User's Manual

1.3 The Title Page

All documents should contain a title page with the following information:

- Document type (e.g. Software Architecture Documentation)
- Project name
- Client name
- Group name
- Team members (names and student numbers)
- Publication date
- Document version (as per baselining)
- Change History (summary of changes across versions)

2 Vision and Scope Document

The document is developed at the beginning of the project, but can be modified and extended as the vision and scope for the project evolve.

A description of the vision of the project. This typically includes the main purpose of the projectand what the client aims to achieve with the project (i.e. the business or user needs the project aims to address).

The document often discusses what the alternatives are, e.g. how the functionality is done without the system and/or what the deficiencies of alternative products are.

2.1 Scope and Limitations/Exclusions

The document should specify the major features, high level scope and limitations of the proposed software product.

Use a high-level use case diagram with

- abstract use cases for services/responsibility domains of the system,
- concrete use cases (the leaf use cases in the specialization hierarchy) as the required concrete use cases/user services,
- optionally some include and extend relationships to show the core functional requirements, and

- actors showing the external systems which are not part of the scope of the system, but which the system integrates with.

List the exclusions/limitations, discussing any functionality which could erroneously be assumed within scope but which has been explicitly excluded from the scope of the system.

3 Architecture requirements

The software architecture requirements include the access and integration requirements, quality requirements and architectural constraints.

3.1 Access channel requirements

Specify the different access channels through which the system's services are to be accessed by humans and by other systems (e.g. Mobile/Android application clients, Restful web services clients, Browser clients, ...).

3.2 Quality requirements

Specify and quantify each of the quality requirements which are relevant to the system. Examples of quality requirements include performance, reliability, scalability, security, flexibility, maintainability, auditability/monitorability, integrability, cost, usability. Each of these quality requirements need to be either quantified or at least be specified in a testable way.

3.3 Integration requirements

This section specifies any integration requirements for any external systems. This may include

- the integration channel to be used,
- the protocols to be used,
- API specifications in the form of UML interfaces and/or technology-specific API specifications (e.g. WSDLs, CORBA IDLs, ...), and
- any quality requirements for the integration itself (performance, scalability, reliability, security, auditability, ...).

3.4 Architecture constraints

This specifies any constraints the client may specify on the system architecture include

- technologies which MUST be used,
- architectural patterns/frameworks which must be used (e.g. layering, Services Oriented Architectures, ...)
- ...

4 Software Architecture Documentation

This is template for a software architecture specification document. It is meant to provide some guidelines as an alternative to the Kruchten $4+1$ approach to documenting a software architecture and is more aligned with the approach of a software architecture providing the infrastructure within which application functionality is deployed and executed.

Throughout the document you are encouraged to use diagrams to illustrate aspects of your software architecture.

Note that you can leave out sections which are not relevant to you, add sections which are and generally modify the structure of the document to suit your specific project.

4.1 Architecture requirements

In this section extract the architectural requirments from the software requirements including

- scope of architectural responsibilities (e.g. persistence, reporting, process execution, ...),
- quantified quality requirements,
- integration and access channel requirements, and
- any architectural constraints.

4.1.1 Architectural scope

In this section discuss architectural responsibilities which need to be addressed by the software architecture. Typical examples include those of

- providing a persistence infrastructure (e.g. database),
- providing a reporting infrastructure,
- providing an infrastructure for process execution,
- ...

4.1.2 Quality requirements

This section should state the quality requirements in order of priority/importance. Examples of quality requirements are scalability, reliability, performance, security, auditability, integrability, ...

Each quality requirement needs to be quantified. For example, scalability could be specified in terms of number of transactions per unit time or number of concurrent users.

4.1.3 Integration and access channel requirements

In this section you need to specify the different system which your system must integrate with and the integratation channels and protocols which need to be used.

You should also specify the different access channels through which the system functionality should be made available to humans and/or other systems.

4.1.4 Architectural constraints

In this section discuss the architecturl constraints (if any) which your client has placed on the software architecture of the system you are developing. Exampes of architecturl constraints include

- that the system architecture must be based on some or other reference architecture (e.g. SOA, Java-EE, . . .).
- particular technologies (e.g. programming languages, frameworks, protocols, . . .) you should be using, and
- operating systems and/or devices across which the system must be deployable.

4.2 Architectural patterns or styles

In this section discuss any architectural patterns or styles you have chosen to use together with the rationale for using them.

For example, you might have decided to use layering in order to have lower level layers reusable across different higher level layers (e.g. have a services layer usable by a web front-end and a mobile-device client).

Discuss the elements of the architectural pattern (e.g. the different layers) and how these elements are connected (e.g. the integration channels between the layers).

4.3 Architectural tactics or strategies

In this section discuss any architectural tactics or strategies you are using to concretely address any of the quality requirements.

For example, you could be using thread pooling and/or caching to achieve a higher level of scalability or performance.

Discuss how the strategy is realized within your software architecture.

4.4 Use of reference architectures and frameworks

In this section discuss any reference architectures and/or frameworks you might be incorporating within your software architecture. For example, you could be using an object-relational mapper or a particular adapter or even an application server or an enterprise services bus.

Disucss the reasons for choosing such frameworks for your software architecture.

4.5 Access and integration channels

Discuss in this section

- the different access channels enabling human and system users to use your system (e.g. a web and mobile front-end and a web-services access channel), and
- the integration channels through which your system will integrate with external systems (e.g. via a message queue or the database).

Discuss any protocols and/or technologies you are using for those access and integration channels.

4.6 Technologies

This section can be used to specify programming languages, operating systems and other technologies which you are using for your system.

5 Functional requirements and application design

5.1 Introduction

This section discusses the application functionality required by users (and other stakehodlers).

5.2 Required functionality

Use for each concrete use case a use case diagram with the required functionality in the form of includes and extends relationships to lower level use cases – this mauy be specified across levels of granularity.

5.3 Use case prioritization

Consider a simple three-level prioritization with

Critical: A use case which is absolutely essential (ask whether the project should be canceled if that functionality could not be provided).

Important: The system would still be useful without some of the important use cases, but the client would get quantifiably less value from the system.

Nice-To-Have: Its a requirement but the value to the client/business is insignificant/not quantifiable.

5.4 Use case/Services contracts

For each use case/service specify

Pre-Conditions: the conditions under which the service may be refused (usually there is an exception associated with each pre-condition).

Post-Conditions: the conditions which must hold true after the servcie has been provided.

Request and Results Data Structures: Use class diagrams to specify the data structure requirements for the request and result objects (i.e. the inputs and outputs).

5.5 Process specifications

For some of the use cases there may be requirements around the process which needs to be followed. If so, these requirements are typically specified via activity and/or sequence diagrams or alternatively via state charts.

5.6 Domain Objects

Use UML class diagrams to specify the data structure requirements in a technology neutral way. These can ultimately be mapped onto different technologies like ERD diagrams/relational databases, XML schemas, Python/Java/C++/…objects, paper based or UI forms, …But those are just different technology mappings and this would not be part of the requirements specification.

6 Glossary

The documents are to be read, understood and validated by a range of people from very different backgrounds (the client, domain experts/business analysts, the developers, software architects, users, …). Use a glossary to explain any terms which some parties may not be familiar with.