1 Requirements

1.1 Structure and Purpose of System Documentation

The remainder of this section discusses the purpose and content of each section. Generally, each section develops a successively more concrete view of the target realization. In general, specifics are deferred to later sections insofar as it makes sense. For instance, the requirements should not prescribe design decisions; and the design should not prescribe representation details. However, there are no precise “boundaries” for what is specified where. It depends on the nature of the component being described. These sections, and they are all describing the same thing at differing abstraction levels.

1.1.1 Requirements

The Requirements section specifies what the component under design does in terms of its externally observable behavior. Requirement specifications may include such properties as:

- **Functionality**, the input-output relations.
- **Preconditions** or “assumptions,” are conditions for correct use.
- **Postconditions** or “guarantees,” including not only output values but also such things as effects on call-by-reference arguments, file space, etc.
- **Invariants**, such as safety and liveness conditions that are preserved by the executing component.

| 1. Requirements. What the system does. |
| 2. Design. How the requirements are satisfied. |
| 3. Implementation. Key representations, algorithms, etc. |
| 5. Testing. Purpose and procedures. |
| A. Developer Instructions. make procedures, etc. |
| B. User Instructions. End-user procedures. |

Figure 1: Organization of System Documentation
1 REQUIREMENTS

- *Constraints* on resources such as time, space, etc.

- *Validation.* The requirements may include a collection of specific observable (i.e. input/output) behaviors to which the delivered realization must comply. These may be thought of as being provided by the End User (or customer) for determining minimal satisfactory functionality.

The “formal” requirements statement consists of a numbered sequence of individual *requirement specifications*, like Requirement 1.1 at the beginning of Section 1. In critical applications, the subsequent sections are expected to address each of these individually.

1.1.2 Design

The *Design* section explains *how* the requirements are satisfied. For this reason, it is usually organized according to component functionality (rather than architecture, as is the case in the *Implementation* section).

The design is presented abstractly, and routine representation details are deferred. It typically includes, for example, graph depictions of data structures or control-flow. Key algorithms may be presented in abbreviated form (e.g. pseudo-code) the main goal being to show mathematically how requirements are addressed.

Ideally, the design description gives just enough information for someone with sufficient programming expertise to develop an equivalent implementation on their own.

Design *verification* is a comparison of two levels of *description* (as opposed to *testing* the actual realization, see Section 1.1.5). In critical applications, it is necessary to explain how each requirement specification is satisfied by the design. The means of verification ranges from demonstration by model simulation to machine-checked mathematical proof, although in common practice may be merely a careful, more-or-less rigorous English explanation.

1.1.3 Implementation

The *Implementation* section presents “key” representation details, including the overall architecture of the component. It should not include incidental coding details that can be readily understood by inspection of the source code. Instead, this section gives the “lay of the land,” that a competent programmer would need to know before delving into the low-level coding details.

Specific design specification statements, if any, should be addressed. This is another level at which the term “verification” is applicable.
1 REQUIREMENTS

1.1.4 Code

Source code is processed by a code documentation tool. Doxygen\(^1\) that generates navigation indices, such as call graphs. These tools often have comment formatting provisions as well, allowing source-comments to be integrated logically in the higher-level system documentation.

However, the primary purpose of source-comments is to provide local guidance in the immediate code context. Hence, these comments are generally insufficient for the higher purpose of the Implementation section.

1.1.5 Test

In contrast to validation (Sec. 1.1.1) and verification (Sec. 1.1.2), testing refers to execution of the component realization (hardware device, object code, etc.) against a selected sample of inputs and expected outputs.

The test of interest in this section do not include routine tracing for the purpose of programming, but rather, a cumulative suite of fixed tests whose purposes include final validation against end-user requirements, and regression testing against revisions, diagnoses of failures in the field, and so forth.

These tests should be automated for repeatability, and anomales in testing must be tracked and resolved prior to release of a component. This section includes both test specifications and documentation of repeatable test procedures.

1.2 Tagged Specifications

Specification 1.2 Formal specification statements are identified by locator tags indicating the section in which they appear and a sequence number.

\(^1\)http://www.stack.nl/~dimitri/doxygen/index.html is used in P545 unless it is superceded by an equivalent tool provided by the project development environment.