Software Requirements and the IEEE

Introduction

Software Requirements is concerned with the elicitation, analysis, specification, and validation of software requirements. The IEEE has provided ten standards that deal with software requirements that are listed at the end of this work\(^1\). This paper attempts to bring those requirements together into a single document to promote better understanding across the scope of the ten requirements standards documents.

\(^1\) IEEE standards are updated on a schedule; the reader should take care to ensure that the standard being used is the most current version.
ACRONYMS

- DAG - Directed Acyclic Graph
- FSM - Functional Size Measurement
- SADT - Structured Analysis and Design Technique
- UML - Unified Modeling Language

Background

At its most basic, a requirement is a property that must be exhibited to solve some problem in the real world. Software requirements express the needs and constraints placed on software that contribute to the solution of some real-world problem. A software requirement is requirement that is satisfied, at least partially, in software.\(^2\)

An essential property of all software requirements is that they must be verifiable. It may be difficult or costly to verify certain software requirements, but it is important to keep track of which software is intended to satisfy which requirements. Typically, software requirements are uniquely identified so that they can be subjected to software configuration control, managed over the entire software life cycle, and can be identified with, and traced to, specific software components.

Kinds of Requirements IEEE Std 12207-96. The requirements types shown below are broadly congruent with the sections of IEEE 12207 that refer to requirements activities. IEEE 12207 inherently results in a view that is “waterfall-like” but dependence on that particular software development method (i.e., the waterfall model) should not be inferred and is not intended.

Product and Process Requirements.

A distinction can be drawn between product requirements and process requirements. Product requirements are requirements on software that is to be developed or acquired (for example, “An employee shall be authenticated before being allowed to access the system.”). A process requirement is a constraint on the way that the software is developed (for example, “The software shall be written in C++.”).

Some requirements generate other implicit process requirements. In particular the method that will be used to verify the requirement may implicitly (but

\(^2\) It is the case that requirements can be satisfied in many different ways such as human activity, hardware implementation, and software implementation. This work is looking only at requirements that are to be implemented through software.
preferably explicitly) require that certain methods be used. Process requirements may also be imposed directly by the development organization, the customer, or a compliance section such as security or records management.

Functional and Nonfunctional Requirements.

Functional requirements describe the procedure that the software is to perform; for example, adding a column of numbers, or printing a report. Functional requirements are also known as capabilities; however the term “capabilities” is a highly overloaded term in Information Technology and should probably be avoided.

Nonfunctional requirements constrain a solution, and are therefore sometimes called constraints.

Requirements can be typed by area of impact. For example there are performance requirements, maintainability requirements, safety requirements, reliability requirements, security requirements, records management requirements, and many other possibilities.

Emergent Properties

Some requirements represent emergent properties of software. Such requirements cannot be addressed by a single component, but are satisfied by the way several (or all) of the software components interoperate. Emergent properties of software can be captured as elements of overall software architecture rather than a characteristic of a single software component.

Quantifiable Requirements

Software requirements should be stated clearly and unambiguously and, where appropriate, quantitatively. This is particularly important for nonfunctional requirements.

System Requirements and Software Requirements

A system is an interacting combination of elements that work together to accomplish a defined objective. Elements include hardware, software, firmware, people, information, techniques, facilities, services, and other support elements.

System requirements are the requirements for the system as a whole and can be met through any of the elements of the system.

---

3 For example, a requirement for formal verification through mathematical proof may require that certain language constructs, or even certain languages, be used.
The Requirements Process.

The requirements process is not a discrete front-end activity of the software life cycle, but rather a process initiated at the beginning of a project and continually refined throughout the software life cycle.

Software requirements are configuration items, and they should be managed using software configuration management practices as are other products of the software life cycle process.

The requirements process is interdisciplinary. The requirements specialist needs to mediate between the domain of the stakeholder and that of software engineering. There may be many people involved besides the requirements specialist, each of whom has a stake in the software. The stakeholders will vary across projects, but always include users and customers. Typical examples of software stakeholders include:

- Users: People who will use the software. It is often a heterogeneous group comprised of people with a variety of roles and needs.
- Customer: The group who commissioned the software or who represent the software’s target business.
- Regulators: Federal, State, Local, and Department regulatory authorities.
- Software engineers: People who profit from developing the software.

The requirements process plays a key role in terms of the cost and timeliness of a software product, and the customer’s satisfaction with the resulting system.

Requirements Elicitation.

Requirements elicitation is concerned with how the software engineer can collect requirements. It is the first stage in understanding the problem the software is to solve. It is a human activity where the stakeholders are identified and relationships established between the development team and the customer. Requirements elicitation is sometimes called “requirements acquisition.”

Requirements elicitation includes:

- Goals. The term goal or “business concern” or “critical success factor” refers to the high-level objectives of the software. Goals provide the motivation for the software, but are often vaguely formulated. Software engineers should pay particular attention to assessing the value and cost of goals. A feasibility study is a relatively low-cost way of doing this.
• **Domain knowledge.** The software engineer needs to acquire, or have available, knowledge about the application domain. This enables the software engineer to infer tacit knowledge, assess trade-offs that will be necessary between conflicting requirements, and to champion the effort within the customer and his own organization.

• **Stakeholders.** Software sometimes proves unsatisfactory because it has stressed the requirements of one group of stakeholders at the expense of others. The software engineer needs to identify, represent, and manage the “viewpoints” of all of the types of stakeholders. Whenever it is not possible to perfectly satisfy the requirements of every stakeholder, it is the software engineer’s job to negotiate trade-offs that are acceptable to the principal stakeholders and within budgetary, technical, and regulatory constraints.

• **Operational environment.** Some requirements derive from the environment in which the software will be used. Such requirements should be actively sought out, because they can affect software feasibility and cost, and restrict design choices.

• **Organizational environment.** Software requirements can be derived from structure, culture, and internal politics of the organization. The software engineer should be aware of these and should not try to force unneeded change on the organization.

Once the requirements sources have been identified, the software engineer can start eliciting requirements. It is particularly important to understand that elicitation is not a passive activity, and that, even if cooperative and articulate stakeholders are available, the software engineer has to work hard to elicit the right information. A number of techniques exist for doing this, the principal ones are interviews, scenarios, prototypes, facilitated meetings, and observation or immersion.

Requirements Analysis is the process of analyzing requirements to detect and resolve conflicts between requirements, discover the bounds of the software and how it interacts with its environment, and elaborate system requirements to derive software requirements.

The traditional view of requirements analysis has been conceptual modeling using some well documented analysis method. Analysis includes classification of requirements to help make trade-offs between requirements. Requirements should be described precisely enough to enable them to be validated, their implementation to be verified, and their costs to be estimated.
Requirements Classification.
Requirements can be classified in a number of ways, some common categories include:

- Functional or nonfunctional
- Derived or imposed.
- Product or process.
- Priority, the more essential the requirement is for meeting the overall goals of the software.
- Scope the extent to which a requirement affects other software and software components.
- Volatility/stability. The likelihood of change.

Other classifications may be appropriate, depending upon the organization’s normal practice and the application itself.

Conceptual Modeling IEEE Std 1320.1 and IEEE Std 1320.2.
Conceptual models are models of entities from the problem domain configured to reflect their real-world relationships and dependencies. Modeling the real-world problem is one key to software requirements analysis. Modeling aids in understanding the problem but is not meant to be the design solution. Several kinds of models can be developed, including data and control flows, state models, event traces, user interactions, object models, data models, and others. The factors that influence the choice of model include:

- **The nature of the problem.** Generally, control flow and state models are likely to be important for real-time software while data models are likely to be more important for management information software.

- **Expertise of the software engineer.** It is often more productive to adopt a modeling notation or method with which the software engineer has experience.

- **Customer requirements.** Customers may impose their favored modeling technique.

- **Availability of methods and tools.** Notations or methods that are poorly supported by training and tools may not achieve objectives even if they are the best choice for a particular type of problem.

It is useful to start by building a model of the software context. The software context provides a connection between the intended software and its external environment, and aids understanding the software’s context in its operational
environment and it aids in understanding how the software interfaces with the environment.

Formal modeling using notations based on discrete mathematics that can be shown logically consistent, or correct have had an impact in some specialized domains. Formal mathematical modeling may be imposed by customers or standards and may offer advantages in analysis of critical functions or components.

**Methods.** The issue of modeling is sometimes confused with that of methods. A method is a notation (or set of notations) supported by a process that guides the application of the notations. There seems to be no empirical evidence for the superiority of one notation over another. However, the widespread acceptance of a particular method such as UML (Unified Modeling Language) can lead to beneficial industry-wide pooling of skills and knowledge.

**Architectural Design and Requirements Allocation IEEE Std 1471-2000.**

Architectural design is the point at which the requirements process overlaps with software and systems design. In many cases, the software engineer acts as software architect because the process of analyzing and elaborating the requirements demands that software (and other) components that will be responsible for satisfying the requirements be identified.

Requirements allocation is the assignment of requirements to components. Allocation is important to permit detailed analysis of requirements and for other operations and maintenance processes. Once a set of requirements has been allocated to a component, further requirements may be discovered having to do with how the component needs to interact with other components to satisfy the allocated requirements. In large projects, allocation generally stimulates a new round of analysis for each subsystem.

Architectural design is closely identified with conceptual modeling. The mapping from real-world domain entities to software components is not always obvious, so architectural design is sometimes identified as a separate topic. The IEEE standard suggests a multiple-viewpoint approach to describing the architecture of systems and their software items.

In most cases, it is wise for the software engineer to consult with the stakeholder(s) to reach a consensus on an appropriate trade-off. It is often

4 We apologize for introducing a new term here. A subsystem can be viewed as a set of components that work together to satisfy a sub-set of system goals or objectives. Another term that has come into use is service which is also a set of components that are “used” together to satisfy a sub-set of system goals or objectives.
important for contractual reasons that such decisions be traceable back to the customer and captured as part of the software architecture.

Requirements Specification.
In software engineering “software requirements specification” refers to the production of a document that can be systematically reviewed, evaluated, and approved⁵. For complex systems, particularly those involving substantial non-software components, as many as three different types of documents are produced: system definition, system requirements, and software requirements. For simple software products, only the software requirements definition is produced or a combined definition/requirements document may be produced.

The System Definition Document IEEE Std 1362-98.
This document is sometimes called the user requirements document or concept of operations. It records the system requirements and defines the high-level system requirements from the domain perspective. Its readers include representatives of the system users/customers so its content must be couched in terms of the domain. The document lists the system requirements along with background information about the overall objectives for the system, its target environment and a statement of the constraints, assumptions, and non-functional requirements. It may include conceptual models designed to illustrate the system context, usage scenarios and the principal domain entities, as well as data, information, and workflows.

Developers of systems with substantial software and non-software components often separate the description of system requirements from the description of software requirements. On this view, system requirements are specified and the software requirements are derived from the system requirements. Finally, the requirements for the software components are specified.

Software requirements specification establishes the basis for agreement between customers and developers about what the software product is to do and what it is not expected to do. For non-technical readers, the software requirements specification document is often accompanied by a software requirements definition document. Software requirements specification permits a rigorous assessment of requirements before design begins and reduces later re-design. It should also provide a basis for estimating product costs, risks, and schedules. Organizations can also use a software requirements specification document to develop their own validation and verification plans.

⁵ Traditionally in engineering disciplines, the term “specification” refers to the assignment of numerical values or limits to a product’s design goals.
Software requirements are often written in natural language but this may be supplemented by formal or semi-formal descriptions. Selection of appropriate notations permits particular requirements and aspects of the software architecture to be described more precisely and concisely than natural language.

The general rule is that notations should be used which allow the requirements to be described as precisely as needed. High precision is particularly crucial for safety-critical, security critical, and certain other types of dependable software.

A number of quality indicators have been developed that can be used to relate the quality of software requirements specification to other project variables such as cost, acceptance, performance, schedule, and reproducibility.

“Lack of Quality” indicators for individual software requirements specification statements include imperatives, directives, weak phrases, options, and continuances.

Indicators for the entire software requirements specification document include size, readability, specification, depth, and text structure. IEEE Std 830-98 is a guideline for the production and content of the software requirements specification, and IEEE Std 1465-98 is a standard treating quality requirements in software packages.

Requirements Validation.

Requirements validation is concerned with the process of examining the requirements document to ensure that it defines the right software (that is, the software that the users expect).

Requirements may be validated to ensure that the software engineer has understood them, but it is also important to verify that a requirements document conforms to company standards, and that it is understandable, consistent, and complete.

Different stakeholders, including representatives of the customer and developer, should approve the document(s) and the documents should be subject to the same software configuration management practices as the other deliverables of the software life cycle processes.

It is normal to schedule one or more points in the requirements process where the requirements are validated. The intent is to identify any problems before resources are committed to building software.
Requirements Reviews IEEE Std 1028-97.

IEEE Std 1028-97 provides guidance for conducting requirements (and other software) reviews.

The most common means of validation is inspection or review of the requirements document(s). A group of reviewers is assigned to look for errors, mistaken assumptions, lack of clarity, and deviation from standard practice. The composition of the group that conducts the review is important because the document should be approved or accepted by the customer. It may help to provide guidance on what to look for in the form of checklists.

Reviews may begin upon completion of the system definition document, the system specification document, the software requirements specification document, the baseline specification for a new release, or at any other step in the process.

Other means for validation of requirements include:

- **Prototyping.** The dynamic behavior of a user interface can be better understood through a prototype than through textual description or graphical models. However, the users’ attention can be distracted from the core underlying function by cosmetic issues or quality problems with the prototype. For this reason, many recommend prototypes that avoid software. Prototypes may be costly to develop but when they avoid the waste of resources caused by trying to satisfy erroneous requirements, their cost can be more easily justified.

- **Model Validation.** Modeling is a recommended technique for eliciting requirements (see above). Those models can be validated through reviews. For object models it has been proven useful to perform a static analysis to verify that communication paths exist between objects that exchange data.

- **Acceptance Testing.** An essential property of a software requirement is that it should be possible to validate that the finished product satisfies it. Requirements that cannot be validated are just “wishes.” Identifying and designing acceptance tests may be difficult for non-functional requirements as they may not be expressed quantitatively.

**Practical Considerations**

The first level of decomposition presented above may seem to describe a linear sequence of activities, but that is an inadequate view. The requirements process spans the whole software life cycle. Change management and the maintenance of the requirements in a state that accurately mirrors the software
to be built, or that has been built, are a key to the success of the software engineering process.

Requirements should consist of a specification of what is required, and also of ancillary information that helps manage and interpret the requirements. This includes the classification dimensions of the requirement and the verification method or acceptance test plan. It may also include additional information such as a summary rationale for each requirement, the source of each requirement, and a change history. A critical requirements attribute is a unique identifier that allows the requirements to be unambiguously identified.

Requirements tracing is concerned with recovering the source of requirements, and predicting the effects of requirements. Tracing is fundamental to performing impact analysis when requirements change. A requirement should be traceable backwards to the stakeholders that motivated it. Conversely, a requirement should be traceable forward into the software components that satisfy it. The requirements tracing for a typical project will form a complex directed acyclic graph (DAG) of requirements.

**Measuring Requirements IEEE Std 14143.1-00.**

It is useful to have some concept of the “volume” of the requirements for a particular software effort. This number is useful in evaluating the “size” of a change in requirements, in estimating the cost of a development or maintenance task, as the denominator in other measurements. Functional Size Measurement (FSM) is a technique for evaluating the size of a body of functional requirements. IEEE Std 14143.1-00 defines the concept of FSM.
IEEE Software Requirements Standards.


