CS360: Software Engineering

Chapter 4: Requirements Engineering

Requirements are descriptions of services that a system should provide and the constraints on it’s operation.

Requirement is not a standard, it’s definition is different from different perspectives, it can either be a high-level statement of a service or a formal definition of a system function.

The reason for this difference lies in the nature of contracts, contractors can vie for a contract and offer ways that they can build the system for the contractee.

Mnay issues lie in failing to create a solid divide between different levels of description. This can be remedied through the following definitions.

* User requirements – High-level abstract requirements
* Systems requirements – detailed description of what systems do.

User reqs. may vary from user to user, and can be either broad or detailed.

Sys. reqs. will be detailed of what a system will do in function, service, and operational constraints. The Sys. reqs. document is called a **functional specification** and it will define exactly what should be implemented.

User Reqs. and Sys. Reqs. both need to be written in a manner that is conducive to those who are reading it.

User Reqs. will be viewed more by managers, end-users, contractors, and system architects.

Sys. Reqs. will be viewed more by engineers, developers, system end-users, and system architects.

Stakeholders are those who are invested or will use the software.

Requirements engineering is one of the first phases of software engineering. It is important to establish high-level views of what a system may do. Then can they be considered in a feasibility study, which will assess whether the system is feasible in financial and technical manners.

Two types of requirement.

Functional and Non-functional

Functional consists of; Services, behavior, and other aspects of how the software should act.

Non-functional requirements are constraints on the services or functions offered by the system, they will typically be applied to the entire software system rather than individual components of the system.

These may provide some level of clarity, however, that clarity does not always sustain itself in the case of specific user requirements. Additionally, requirements may generate more requirements.

In the case that a specific off-the-shelf system meets the needs of an organization, then you instead work on getting **information requirements** which specify the information needed for people to do their work.

Imprecision and ambiguity in requirements can lead to changes in the software down the road, however, the further down the road, the greater the cost.

Ideally, reqs. are complete and consistent. All information and services should be defined and requirements should not contradict one another. Although a perfect requirements phase is normally only possible with smaller systems. For more work that needs to be done, more errors can be made. Additionally, the more stakeholders, the more different views there will be, often inconsistent views.

Non-func. Reqs. are often quite critical, more critical than functional reqs. Users can figure out how to work around broken func. Reqs. but often times they can’t use systems that have bad non-func. Reqs.

Non-func. May affect overall structure of a system rather than the individual components.

Individual non-func. Reqs like security may generate several related func. Reqs.

Non-func. Reqs. mostly come from user needs because of budget constraints, policies, or interoperability with other systems.

Common issue with non-func. reqs. is the generalized goals, this makes them ambiguous and the developers must discover scope and how to implement these things

Write non-functional requirements quantitatively so that they can be objectively tested.