Requirements Analysis

Software Requirements

A *software (product) requirement* is a feature, function, capability, or property that a software product <u>must</u> have.

Software Design

A *software design* is a specification of a software system that programmers can implement in code.

What vs How

The traditional way to distinguish requirements from design:

What a system is supposed to do is *requirements*

How a system is supposed to do it is *design*

What vs How Problems

The what vs. how criterion is probably unworkable:

- **Other design disciplines do not make this distinction**
- **Understanding needs and constraints must always be the first step in design**
- **The what vs. how criterion does not hold up in practice**

Requirements vs Design

Determining requirements is really the first step of design.

Requirements **state client needs and solution constraints.**

Designs **state problem solutions.**

Requirements Phase Activities

Problem or requirements analysis

Working with clients to understand their needs and the constraints on solutions

Requirements Specification

Documenting the requirements

These activities are logically distinct but usually occur together.

Requirements Phase Problems

The requirements phase is HARD! There are several reasons for this:

- **requirements volatility**
- **requirements elicitation problems**
- **language problems**
- **requirements traceability problems**

Problem: Requirements Volatility

If requirements change during development, complexity increases and productivity drops, but bad requirements can be fixed and changing customer needs met.

⇕

If requirements are frozen in development, the product is produced more quickly and cheaply, but may not meet customer needs.

Problem: Requirements Elicitation

Eliciting requirements is difficult because:

- **It is often not clear who the users are**
- **Different customers want different things**
- **Customers don’t know what they want**
- **Engineers don’t understand the customers’ domain, so they can’t understand the customers**
- **Customers don’t understand computing technology, so they can’t understand the engineers**
- **Engineers are not trained to elicit requirements**
- **Requirements elicitation is a big job–everyone usually get tired of it, or run out of time and money for it, before it is really done**

Problem: Language

Natural language is expressive and understood by customers and developers, but it is imprecise and wordy.

⇕

Formal language is precise and concise and may be amenable to formal analysis, but it is inexpressive and difficult to understand.

Problem: Requirements Traceability

A single requirement may have consequences in many parts of a design, program, and data, and may need many test cases to verify.

How can requirements be traced through the life cycle to ensure they are met in the final product?

Problem Analysis Sources

- Customer documentation (market reports, process descriptions, feasibility documents, user profiles, etc.)
- Customer interviews and questionnaires
- Observation of users (process observation, UI studies)
- Joint design activities (JAD and PD)
- Reactions to demos, prototypes, and models

Software Requirements Specification

A *software requirements specification (SRS)* document is a statement of all the requirements for a software product.

Role of the SRS

1. Drives the rest of the life cycle
2. Forces good problem analysis
3. Serves as an agreement between customers and developers about the delivered product

SRS Quality Criteria: Completeness & Consistency

Completeness–All functions, features, capabilities, constraints, and other properties of the target system are described in detail.

Consistency–Requirements do not conflict with one another and all terms are used with the same meaning.

SRS Quality Criteria: Conciseness & Testability

Conciseness–No extraneous information is included in the requirements document: information about project history, costs, schedule, etc., appears elsewhere.

Testability–Each requirement is stated so that it can be tested.

SRS Quality Criteria: Readability & Traceability

Traceability–Some means is provided to verify whether the requirements are realized in design and code.

Readability–The requirements document is easy to read and understand, and each requirement is stated unambiguously.

SRS Quality Criteria: Feasibility and Changeability

Feasibility–All requirements can be satisfied using the tools, techniques, people, and budget available—the requirements specification phase is a good time to re-evaluate feasibility.

Changeability–The requirements specification document is written so that it is easy to change later in the life cycle.

Functional Requirements

Functional requirements specify what the system should do, that is, the services the system should provide, and the way the system should interact with its users.

Example Functional Requirements

- All screens, buttons, menus, and so forth, in the user interface, what they all do, and how they interact
- All data transformations, that is, all computations done by the system
- All files, databases, communication links, and other sources or sinks of information used by the system, and the way they are named, created, deleted, changed, and so forth
- System behavior in response to incorrect, incomplete, or inconsistent input, and to computational errors

Non-Functional Requirements

Non-functional requirements **specify constraints on how the system should operate, and standards for its operation.**

Example Non-Functional Requirements

- Response times to user operations
- Throughput (transactions per unit of time) and capacity (amount of data, users, processes, etc. handled)
- Environment requirements (processors, memory, OS, etc.)
- Reliability (rate of failure) and fault tolerance (ability to withstand and recover from bad input or processing errors)
- Security (ability to resist intruders and protect privacy)
- Development requirements (standards, deliverables, IV&V requirements)
- Maintainability (ability to accommodate)

Notations

Various notations are used during problem analysis and requirements documentation, including:

- **English**
- **Pseudo Code**
- **Data Flow Diagrams**
- **Data Dictionaries**
- **Structure Charts**
- **ER Diagrams**
- **SADT Diagrams**
- **Finite State Automata**
- **Regular Expressions**
- **State Charts**
- **Decision Tables**
- **Decision Trees**
- **Object Diagrams**
- **Z**

Methods

Various requirements analysis and specification methods are used:

Structured Analysis—Developed in the mid- to late-70's by Constantine, DeMarco, Yourdon, Gane, Sarsen, and others; still the most popular and widely used method

Structured Analysis Offshoots—methods by Jackson, Warnier, Orr, Bachman, and others; includes SADT and real-time system versions; used in selected and special-purpose shops

Object Oriented Analysis—A family of methods currently being developed by Coad, Yourdon, Rumbaugh, Schlaer, Mellor, Wirfs-Brock, Booch, and many others; this method become the new standard