Development of a Requirements Classification Scheme for Automated Support of Software Development

Dolly Samson

Computer Information Systems Department, Weber State College,
Ogden, Utah 84408-3804, USA

PROBLEM OF REQUIREMENTS ANALYSIS IN SOFTWARE ENGINEERING

Computer hardware technology has advanced to the point where it is feasible to develop software systems that exceed one million lines of code. Software engineering, the discipline concerned with developing these systems, typically includes the following phases in the software development cycle: requirements phase, design phase, implementation phase, test phase and the installation and checkout phase [IEEE, 1983]. This paper is concerned with a classification scheme to support knowledge-based analysis of software requirements.

Because requirements in these large systems can become quite numerous and complex, it is a very difficult task to analyze them in order to identify potential problems, i.e., problems that may arise in later phases, such as incompleteness, conflict, ambiguity, and absence of testability. Current software productivity tools such as CASE (Computer-Aided Software Engineering), executable requirements languages, prototyping tools, and test harnesses provide much assistance in later phases of software development, but not much for the requirements phase.

It is well-known in software engineering that the most costly and profound software errors are likely to occur at the requirements definition phase, early in the software development cycle. The productivity tools mentioned earlier either work from code which has been developed for the system, or work with user requirements statements which have been translated into a restricted, formal language. There are several problems with translating user requirements into a formal language very early in the project. First, it is difficult for the user to understand requirements as they have been translated and interpreted by the systems analyst. Second, the systems analyst may have made wrong assumptions or interpretations in the process of translation. Finally, formal languages do not have any provision for ambiguity, and early in the requirements phase, ambiguities may exist, to be worked out as more is learned about the system.

KNOWLEDGE-BASED SYSTEMS FOR REQUIREMENTS ANALYSIS AND TEST PLANNING

The focus of my work has been to use a knowledge-based approach to performing analysis of requirements to detect potential problems, primarily lack of testability. The knowledge-based system includes a classification scheme to represent software requirements as factual knowledge, and rules describing typical requirements characteristics as procedural knowledge. Terms in this classification scheme are used for indexing the requirements, and for providing access points used by the procedural part of the system. The classification scheme, including the structure and methodology used to develop it, will be described later on.

As an example of a requirements conflict problem, let us assume two requirements are indexed as *software portability* and *operating efficiency* (terms in the classification scheme). In most software systems, these two concepts of portability and efficiency would conflict. When hundreds of requirements such as these have been defined for a software system, this problem

ISSN: 2324-9773

might not be discovered until later in development when it is more costly to repair. The knowledge-based approach is designed to detect conflicts and other inconsistencies (including incompleteness and ambiguity) by encoding and applying rules that make use of classification-based indexing terms assigned to, and therefore descriptive of, requirements. A prototype has been developed as a proof of concept [Samson, 1988] demonstrating that it is possible to automate a requirements analyzer based on semantics of requirements statements as originally defined, rather than syntax of more formalized representations later in the software development cycle.

A second prototype was built for the U.S. Army that not only analyzed requirements, as did the previous prototype, but also served as a knowledge-based test plan generator [Samson, 1990a]. This system analyzes requirements for their testability. It then matches each requirement with appropriate metrics and test tools for the purpose of developing a preliminary test plan. For example, a requirement which states a maximum acceptable response time to a user query must be tested. To test whether the delivered software system meets this requirement, we might use a stopwatch to measure the elapsed time for the system to respond to a query. Here the elapsed time is the metric, and the stopwatch is the test tool.

A third project, completed for the NASA Jet Propulsion Laboratory during summer, 1990, expanded the requirements classification scheme to cover system requirements (i.e. hardware, personnel, and procedures in addition to software) [Samson, 1990b]. This project leveraged lessons learned from the previous two projects to develop a more flexible and useful classification scheme.

DEVELOPING CLASSIFICATION SCHEMES FOR KNOWLEDGE-BASED SYSTEMS

The remainder of this paper will discuss development of the classification schemes beginning with the NASA JPL one. This classification was developed from scratch, through analysis of what CASE tools do with requirements, study of requirements analysis in textbooks, 19 years of personal software engineering experience, indexing of over 100 requirements statements from several software- based systems, and a classification exercise given to JPL test planners. Our classification scheme at JPL, which we refer to as a requirements taxonomy, uses a faceted structure with five facets. This faceted structure was inspired by one used successfully in software reuse projects [Prieto-Diaz, 1987]. The five facets are:

Feature:	the basic service described by the requirement
Object:	the object supporting or facilitating the feature
Function:	what is being done to, with, or by the feature
Temporality:	temporal characteristic
Quality Attribute:	non-functional, subjective characteristic

Figure 1 shows an excerpt from this taxonomy. The following requirement shows how individual terms were obtained by indexing actual requirements. This requirement is from a ground data system, specifically, an earth-based receiving system which is capable of acquiring radar signals and processing those signals into meaningful data products, such as images, and of sorting, disseminating, and archiving data. The main component of the system is the database containing images accompanied by descriptive information about these images.

ISSN: 2324-9773

Requirement: ***Load and maintain a directory that is searchable from a computer terminal.***

Indexing:

- *load and maintain* indicates the FEATURE data management
- *directory* is a specific OBJECT that facilitates data management
- *searchable* describes the search FUNCTION
- *computer terminal* indicates interactive TEMPORALITY
- there is no subjective characteristic for a QUALITY GOAL

As shown here, classification of individual requirements is accomplished through manual analysis of nouns and verbs. This is supported by an engineering thesaurus [U.S. Dept. of Defense, 1987] and a rudimentary thesaurus for ground data systems. Automation of the requirements indexing process will be essential for application of the classification scheme to be useful. We are currently investigating ways to implement automated indexing.

Context dependency is an issue that needs to be handled in the process of indexing requirements. Several examples include the word *remove*: "remove a file from a disk" is different than "remove a disk from a drive" (in the first case the file is destroyed, in the second, the disk still exists and can be re-inserted) and the word *power*: "computing power" refers to millions of instructions per second, where "electrical power" refers to watts. For each of these terms, different test strategies will prevail. For the ground data system example, domain knowledge is applied to understand the use of particular words. For example, ground data system requirements frequently describe a "product", which is actually an image of earth or another planet or star.

The first two classification schemes mentioned earlier in this paper both used a hierarchical structure for indexing requirements (see excerpt in Figure 2). They were developed in much the same way as the faceted scheme, without validation by experts. This scheme proved to be very difficult to modify when indexing many requirements, where new classes were discovered as new requirements were indexed. The hierarchical structure was very cumbersome, with many leaves (over 200 for the U.S. Army version), requiring structural changes as requirements from different systems were indexed.

CONCLUSION

The approach to characterizing software and system requirements through a classification scheme seems to provide enough information to analyze requirements to detect potential problems. The faceted structure better accommodates a more diverse set of requirements than a hierarchical structure. The next step in this work is to map combinations of terms to implementation and testing strategies in order to produce systems that are less trouble-free in implementation and can be better validated through testing.

Samson, Dolly. (1990). Development of a requirements classification scheme for automated support of software development. Proceedings of the 1st ASIS SIG/CR Classification Research Workshop, 149-154.
doi: 10.7152/acro.v1i1.12474

Figure 1. Faceted Requirements Taxonomy

Feature	*Object*	*Function*	*Temporality*	*Quality Attribute*
budget	data	archive	dynamic	accessibility
data acquisition	directory	acquire	interactive	adequacy
data communication	documentation	arithmetic	intersystem	availability
data compression	entire system	browse	near-realtime	compatibility
data management	external interface	catalog	parallel	compliance
data retrieval	firmware	connect	prioritized	consistency
documentation	image record	data transfer	random	efficiency
format	hardware	decode	realtime	interoperability
imaging	monitor	display	sequential	maintainability
input	operations	distribute	static	portability
media	performance	encode	periodic	reliability
output	personnel	enforce		timeliness
protocol	procedure	insert		
scheduling	record	manage		
security	report	predict		
system status	request	produce		
transformation	schedule	recover		
	software	report		
	standards	restart		
	status	route		
	text	search		
	user interface	select		
		sort		
		startup		
		track		
		update		

Figure 2. Excerpt from Hierarchical Requirements Taxonomy

I. Functional Requirements
- A. Software Utility
  - 1. Input events
    - a. data
    - b. format
    - c. message
    - d. source
    - e. timing
    - f. volume
  - 2. Processing events
    - a. access
    - b. algorithm
    - c. application
    - d. condition detection
    - e. condition action
    - f. exception handling
  - 3. Output events
    - a. data
    - b. data display
    - c. data storage
    - d. destination
    - e. format
    - f. timing
- B. Software Performance
  - 1. Static requirements
    - a. operating system
    - b. number of terminals
    - c. number of users
    - d. number of files
    - e. number of records
    - f. data accuracy
  - 2. Dynamic requirements
    - a. transaction rate
    - b. amount of data
    - c. error rate
    - d. response time
    - e. throughput
- C. External Interface Requirements
  - 1. Hardware
    - a. CPU
    - b. I/O devices
  - 2. Software
    - a. data communications
    - b. database management systems
    - c. message processing
    - d. operating systems
    - e. other applications
  - 3. Users
    - a. dialog style
    - b. evaluation method
    - c. frequency
    - d. expertise

ISSN: 2324-9773

REFERENCES

IEEE, *Standard Glossary of Software Engineering Terminology*, Institute of Electrical and Electronic Engineers, NY, 1983.

R. Prieto-Diaz and P. Freeman, "Classifying software for reusability," *IEEE Software*, pp. 6-16, Jan 1987.

D. Samson, *Automated Assistance for Software Requirements Definition*, Ph.D. Dissertation, George Mason University, Fairfax, VA, 1988.

D. Samson, "REQSPERT: Automated test planning from requirements," *Proceedings 1st International Conference on Systems Integration*, Morristown, NJ, Apr 1990a.

D. Samson, "Test planning from system requirements: a classification approach," *Information Systems Prototyping and Evaluation*, Jet Propulsion Laboratories, Pasadena, CA, Jul 1990b.

U.S. Dept. of Defense, *Thesaurus of Engineering and Scientific Terms*, 1987.

ISSN: 2324-9773

ISSN: 2324-9773