Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 1999 Proceedings

Americas Conference on Information Systems (AMCIS)

December 1999

Identification of Test Cases from Business Requirements of Software Systems,

Jonathan Maletic
University of Memphis

Khalid Soliman
University of Memphis

Manuel Moreno
William M. Mercer Inc.

Identification of Test Cases from Business Requirements of Software Systems

Jonathan I. Maletic, Ph.D.
University of Memphis
Department of Mathematical Science Division of Computer Science
maleticj@msci.memphis.edu

Khalid S. Soliman
University of Memphis
Fogelman College of Business
ksoliman@memphis.edu

Manuel A. Moreno
William M. Mercer, Inc.
manuel_moreno@mercer.com

Abstract

Many organizations are struggling between the fast delivery of new software and quality assurance. Software testing play a key part in the quality assurance of software systems. Formal testing techniques increase software quality and, at the same time, reduce software development cycle time. This article presents a methodology for the identification and definition of black box test cases based on the functional requirements of a software system. The methodology is applied during the initial phases of software development. The method involves analyzing system requirements and constructing a functional description graph to organize these requirements.

Introduction

Until recently, many software companies have focused on the fast time to market delivery of software products rather than quality assurance. The result is inadequately tested products that have unpleasant surprises for customers (LaMonica, 1995). Testing is a central concept to the construction of quality software. The cost and quality benefits of conducting testing from the very initial phases of a software development project are well known and documented (Kit 1997). If there is an error in the requirements or the functional design of the system, the cost of fixing such an error at a late stage in development is immense compared to finding and fixing the error at an earlier phase (Pressman, 1997). An effective process in place for software testing will resolve the dilemma of inadequately tested products. Software testing must be an integrated part of the entire software development process (Kit 1997).

The Process Overview

The methodology introduced refines information contained in the business requirement of a software system. The requirements are described as a set of functional requirements. In the general domain of information, business, or transaction processing these functional requirements reflect the individual requirements of the business process being modeled. The first product derived is a graphical model of the functional requirements. Each functional requirement is investigated separately. The attributes of each functional requirement are identified from domain analysis and requirements documentation. Attributes that are necessary for the functional requirement are on *AND-links* and optional attributes are on *OR-links*. The AND-links are represented with a horizontal arc connecting the links. OR-links are left unconnected.

The domain of each attribute is then examined. The number of possible values, types, or instances is noted. To compute an extreme upper bound on the total number of possible test cases the AND-links are multiplied and the OR-links are then added to this product. This number does not take into consideration duplicates, symmetries, interdependencies, independencies, or equivalence relationships.

With this functional requirement model and the requirements documentation, the inter-relationships between each attribute is assessed. A matrix (upper triangular) describing these relationships is constructed. (see figure 2 for an example) The matrix, R, is defined over $(attributes \times attributes)$. Each element of the matrix, $R_{i,j}$, has a value over a discrete range representing the level of interdependence. Typically this range is represented mnemonically as none, weak, mild, or strong. If two

attributes are strongly related in the problem domain, then this is noted. The matrix is therefore symmetrical and the diagonal values represent the trivial relationship of an attribute with itself.

These two products (functional description graph and relationship matrix) are a concise representation of the requirements describing relationships in an easy to understand graphical manner. From these representation the test cases necessary for adequate coverage of the concepts can be systematically derived. The relationship matrix is used in conjunction with the functional description graph to determine the necessary test cases.

First a model of Add-Registration is given describing the entire problem space. The functional description graph for Add-Registration is given in Figure 1. The graph describes the Add-Registration as having three necessary attributes, namely Registration Type, Vehicle, and Customer. The details of each of these attributes can be described themselves in detail by a functional description sub-graph. Each attribute may in turn be a complex concept composed of multiple attributes. This allows for specific levels of abstraction in the representation that are manifested in the domain knowledge and requirements documentation. Also as independent concepts become apparent, their

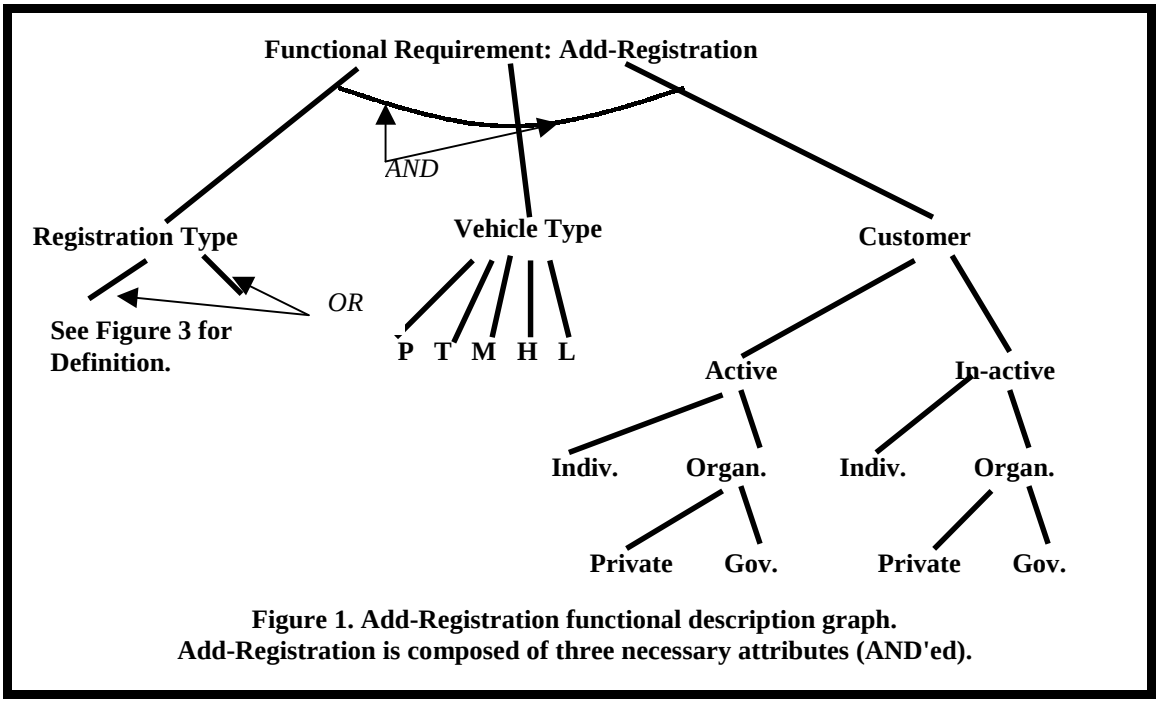

Figure 1. Add-Registration functional description graph. Add-Registration is composed of three necessary attributes (AND'ed).

Interdependencies and inter-dependencies can easily be uncovered and view between functional attributes. This allows for the reduction and fine-tuning of test sets. As the domains of attributes become clear, equivalence relationships can also be taken advantage of to reduce the number of test cases.

A Real World Example

The following example is from the domain of licensing and registration of motor vehicles. This example is taken from an actual software project implemented for a U.S. State department of transportation. In particular the business requirement dealing with adding a registration to the database will be investigated in this example for the development of test cases. A number of attributes and their domains for the *Add-Registration* are denoted.

corresponding functional description graphs can be examined independently. The associated relationship matrix for this example appears in figure 2.

The wide range of interdependency levels directly relates to the amount of testing that is required with respect to the two attributes. At the ends of the scale are high and none. The meanings of these two values are straightforward. Strong implies a very strong relationship between the attributes, that is one attribute can not be properly defined or used without the other. None describes the attributes as completely independent. The mild and weak values describe a varying degree of dependency. Typically these relationships result in the need for development of a subset of test cases. Sometimes an attribute may or may not need the existence of another attribute. This may relate to a medium relationship. Below describes the

general number of test cases that will be required to reasonably test the requirement:

Strong	100%
Mild	50% to 100%
Weak	0% to 50%
None	0%

The number of test cases needed to cover the Add-Registration will now be examined. An upper bound estimate in the number of test cases can be computed with respect to this model. A test case is considered as a specific set of attribute values that tests a given logical situation. For example, a test could be devised for a particular plate type and vehicle type. These sets of attribute values represent an instance of the entire possible input values.

The estimate is computed by multiplying the sizes of each attribute domain together for the necessary attributes (AND'ed links), then adding together optional (OR'ed) parts. One of the main objects here is to find the minimal set of test cases necessary. By identifying the relationships between attributes a large number of test cases can be thrown out as meaningless.

The authors have found this process of easy determination of meaningless tests to be very valuable in practice. Often time's developers did not see the relationships and thus conducted a large number of useless test cases. The relationship matrix labels the relationships between attributes within Add-Registration with a simple strong, weak, or none. Some of the relationships are labeled as weak-none which represent a very weak (or no) relationship between the attributes.

	Reg. Type	Vehicle Use	Tab Type	Vhcl Exmptn	Reg. Status	Customer	Cstmr Exmptn	Vehicle Type	Plate Type	Plate Status	Title
Reg. Type		None	None	None	Mild	Weak	None	None	Mild	Weak	Strong
Vehicle Use			None	None	Mild	None	Weak	None	Weak	Weak	None
Tab Type				None	Mild	Strong	None	None	None	Weak	None
Vhcl Exmptn					Mild	Weak	Mild	Weak	None	Weak	None
Reg. Status						Mild	Mild	Mild	Mild	Weak	Mild
Customer							None	None	Mild	Weak	Strong
Cstmr Exmptn								None	Strong	Weak	None
Vehicle Type									Strong	Weak	Strong
Plate Type										Weak	Strong
Plate Status											None
Title											

Figure 2. Relationship Matrix for the Add-Registration requirement. An identifier that describes the relative strength of the relationship between the two attributes labels each element. If there is a large amount of dependence between two attributes (e.g., Title and Plate-Type) then the corresponding element is labeled as such.

Conclusions

The testing methodology describe here represents a practical means of identifying the size and magnitude of the amount of testing needed for a software system based on its functional requirements. The method, having been applied to a number of problems in industry, is a robust and easily applied method that results in valuable information to the system developers.

References

Kit, Edward. *Software Testing in the Real World*, Addison-Wesley, 1997.

LaMonica, Martin, "Quality questions spur software testing," *InfoWorld*, (17:26), June 26, 1995, pp. 25, 31.

Levin, Rich, "Checking up on software," *Information Week*, (580), May 20, 1996, pp. 1A-6A.

Pressman, Roger S. *Software Engineering A Practitioner's Approach 4th Edition*, McGraw Hill, 1997.