VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
ÚSTAV INFORMATIKY

FACULTY OF BUSINESS AND MANAGEMENT
INSTITUTE OF INFORMATICS

THE APPLICATION OF FUZZY LOGIC FOR TEST CASE PRIORITIZATION

APLIKACE FUZZY LOGIKY PRO URČENÍ PRIORITY TESTŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Ing. ANDREA STAROSTOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR
prof. Ing. PETR DOSTÁL, CSc.

BRNO 2014

Brno University of Technology
Faculty of Business and Management

Academic year: 2013/14
Institute of Informatics

Diploma Thesis Assignment

Ing. Andrea Starostová

Information Management (6209T015)

Pursuant to Act. No. 111/1998 Coll., on Higher Education Institutions, and in accordance with the Rules for Studies and Examinations of the Brno University of Technology an Dean's Directive on Realization of Bachelor, Master and Doctoral Degree Programs, the director of the Institute of is submitting you a diploma thesis of the following title:

The Application of Fuzzy Logic for Test Case Prioritization

In the Czech language:

Aplikace fuzzy logiky pro určení priority testů

Instructions:

Literature / Sources:

DOSTÁL, P. Pokročilé metody analýz a modelování v podnikatelství a veřejné správě. 1. vyd. Brno: CERM, 2008. 340 s. ISBN 978-80-7204-605-8.
DOSTÁL, P. Advanced Decision Making in Business and Public Services. Brno : CERM, 2011. 168 s., ISBN 978-80-7204-747-5.
HANSELMAN, D. a B. LITTLEFIELD. Mastering MATLAB7. Pearson Education International Ltd., 2005. 852 s. ISBN 0-13-185714-2.
KLIR, G.J. a B. YUAN. Fuzzy Sets and Fuzzy Logic, Theory and Applications. Prentice Hall, New Jersey, 1995. 279 s. ISBN 0-13-101171-5.

The supervisor of diploma thesis: prof. Ing. Petr Dostál, CSc.

The deadline for submission for the diploma thesis is given by the Schedule of the Academic year 2013/14.

L.S.

doc. RNDr. Bedřich Půža, CSc.
Director of the Institute

doc. Ing. et Ing. Stanislav Škapa, Ph.D.
Dean of the Faculty

Brno, 24. 3. 2014

Abstract

The master's thesis focuses on determination of Test case priority using Fuzzy logic. As principle of Fuzzy logic is a convenient way to turn given inputs to final output according to defined rules, a Fuzzy based model for assigning Test case priority has been chosen. In order to fulfil the aim of the thesis, firstly particular criteria along with parameters set to each Test case and its weights needs to be defined accordingly. So as to come to the conclusion and evaluate input data, the solution for computing in the program MS Excel and MATLAB is used herein.

Abstrakt

Diplomová práce je zaměřena na stanovení priority testovacích případů s využitím fuzzy logiky. Vhodným přístupem k získání výstupu na základě definovaného vstupu a stanovených pravidel byl zvolen fuzzy model přiřazující prioritu testovacím případům. K dosažení cíle práce byla nejprve stanovena kritéria, parametry a poté určena jejich váha pro jednotlivé testovací případy. Na závěr jsou vyhodnocena vstupní data s využitím řešení v programu MS Excel a MATLAB.

Key words

Fuzzy Logic, Fuzzy Logic Toolbox, Software development, Software testing, Test cases

Klíčová slova

Fuzzy logika, Fuzzy Logic Toolbox, Vývoj softwaru, Testování softwaru, Testovací případ

Bibliographic citation

STAROSTOVÁ, A. *The Application of Fuzzy Logic for Test case Prioritization.* Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2014. 77 s. Vedoucí diplomové práce prof. Ing. Petr Dostál, CSc.

Declaration of originality

I hereby declare that this master´s thesis originated entirely from me. Information derived from the published work has been acknowledged in the text and references are given in the list of reference. I also declare that I did not breach of copyright in the sense of Act. No. 121/200 coll. on Copyright Law and Rights Related to Copyright and on the Amendment of Certain Legislative Acts.

Brno, 30th May 2014

.....

Ing. Andrea Starostová

Acknowledgements

I would like to thank my tutor doc. Ing. Petr Dostál, CSc. for supervising my master's thesis, for his guidance and for providing feedback with this work.

In particular, I would like to express my thanks to Ing. Věslav Zahradník CSc. for his availability at all times to discuss particular aspects of the thesis. I greatly appreciate his time and insight. After all, much of this thesis would not have been possible without the assistance and support of my family and close friends.

CONTENT

INTRODUCTION

As a matter of fact, self-learning approaches are applied in software computations comprising statistics, machine learning, neural networks and Fuzzy logics. Artificial intelligence has featured as promising, instrumental and practical technique of soft computing technologies in science and engineering domains. Nevertheless, progression from bivalent logic to Fuzzy logic is a significant positive step in the evolution of science. In large measure, the real-world is a fuzzy world. To deal with fuzzy reality what is needed is fuzzy logic. In coming years, Fuzzy logic is likely to grow in visibility, importance and acceptance.

From the tests in software engineering point of view, fuzzy expert system provides a better way of prioritizing the Test cases. Moreover, prioritization of Test cases becomes all the more important due to fact that it is not feasible to run all the Test cases after each and every change. Once a change is made it is not possible to retest all the Test cases of the test suite as it will consume lot of time. Therefore, prioritization of the Test cases has been widely proposed and used in recent years as it can improve the rate of fault detection during the testing phase.

More specifically, prioritization is used when the time for the testing is limited. So, in order to attain maximum coverage, the more important cases are tested. However for this purpose, Fuzzy expert system should be selected because of better decisions made by it in comparison to the normal expert system.

In this regard, the master's thesis aimed at determination of Test case priority using Fuzzy logic. The work is organized into three main parts as follows. The first part describes the basis of theoretical background concerning testing phase in software engineering that is essential for understanding of subsequent parts. The second part represents analyses focusing on profile of BIAC's company, BIAC services and process of software testing in BIAC.

Finally, the last part is dedicated to evaluation of Test case priority in the program MS Excel and Fuzzy Logic Toolbox in MATLAB. Nevertheless, in order to meet the objective of the master's thesis, the last chapter reflects approach of determination of particular criteria along with parameters set to each Test case and its assigned weights.

AIM OF THE THESIS

The aim of master's thesis is to determine prioritization of Test case using Fuzzy logic based model. The output of the proposed model will be determination of Test case priority order in the program MS Excel and Fuzzy Logic Toolbox in MATLAB which would in fact increase among other things the test effectiveness and fault detection rate. Nevertheless, the objective of the proposed solution is concentrated on definition of input variables along with parameters set to each Test case and assigning its particular weights based on testing environment.

1 THEORETICAL BACKGROUND

1.1 Fuzzy logic

According to Zadeh (2008, p.2753), fuzzy logic is a precise logic of imprecision and approximate reasoning. More specifically, fuzzy logic may be viewed as an attempt at formalization/mechanization of two remarkable human capabilities. First, the capability to converse, reason and make rational decisions in an environment of imprecision, uncertainty, incompleteness of information, conflicting information, partiality of truth and partiality of possibility – in short, in an environment of imperfect information. And second, the capability to perform a wide variety of physical and mental tasks without any measurements and any computations. In fact, one of the principal contributions of fuzzy logic – a contribution which is widely unrecognized – is its high power of precisiation.

To be more precise, fuzzy logic deals with the concept of partial truth theory and provides a methodology to model uncertainty and the human way of thinking, reasoning and perception. Fuzzy logic systems are rule-based or knowledge-based systems first formulized by Zadeh in 1965. Since the fuzzy set, a class of objects with a continuum of grades of membership, is descriptive of vague impressions than numerical, variables are therefore better described by linguistic terms.

Fuzzy logic sets are characterized by membership functions, also known as characteristic functions that assign to each object a degree of membership varying between zero and one. Variety of membership functions are in practice such as S-shaped, Z-shaped, Triangular, and Trapezoidal shaped functions. The triangular membership functions are formed using straight lines. These straight line membership functions have the advantage of simplicity. Because of their smoothness and concise notation, Gaussian membership functions are popular methods for specifying fuzzy sets. These curves have the advantage of being smooth and nonzero at all points (Taghavifar and Mardani, 2013).

In fact, fuzzy logic measures the certainty and uncertainty of how much the element appertains to the set. Due to the principle of fuzzy logic, it is practicable to figure out the solution of a given task better than by conventional methods (Dostál, 2011).

1.1.1 Operations on Fuzzy Sets

Basically, a fuzzy set is a class of objects with a continuum of grades of membership. Such a set is characterized by a membership (characteristic) function which assigns to each object a grade of membership ranging between zero and one. The notions of inclusion, union, intersection, complement, relation, convexity, etc., are extended to such sets, and various properties of these notions in the context of fuzzy sets is proved without requiring that the fuzzy sets be disjoint (Zadeh, 1965, p.338).

The notion of fuzzy sets aimed at mathematically modelling vague concepts was first introduced by Zadeh in connection with the representation and manipulation of human knowledge automatically. As Zadeh (1965, p.339) described, the theory of fuzzy sets is a generalization of classical set theory, making use of the notion of partial degrees of membership. Practically, the theory of fuzzy sets provides a systematic framework for dealing with complex phenomena in describing the behaviour of systems which do not lend themselves to analysis by classical methods based on probability theory and bivalent logic.

Since its inception, the mathematical foundation as well as extensive application of the theory too many different areas have already been well established (Zadeh, 1965). The examples of fuzzy sets are illustrated in the Figure 1 (ESRU, 2014).

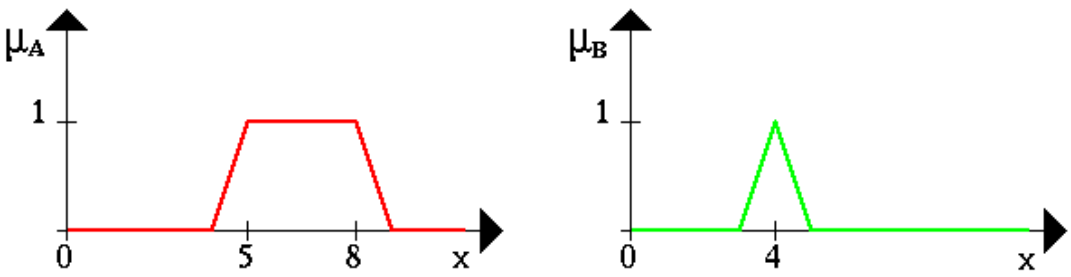


Figure 1 Fuzzy sets μ_A, μ_B. Adopted from ESRU (2014)

The following Figure 2 gives an instance of intersection of the fuzzy set between 5 and 8 **AND** about 4 (ESRU, 2014).

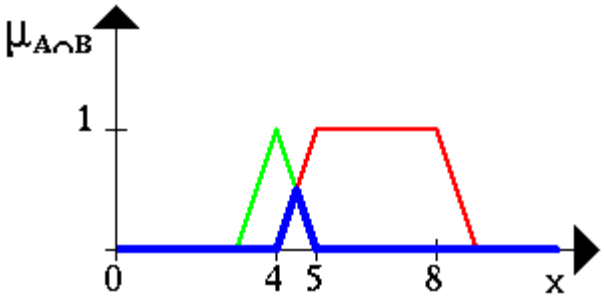


Figure 2 Intersection of two fuzzy sets. Adopted from ESRU (2014)

In Figure 3, the union of two fuzzy sets is shown. Besides that, the negation of the fuzzy set A is represented by blue line (ESRU, 2014), (see Figure 4).

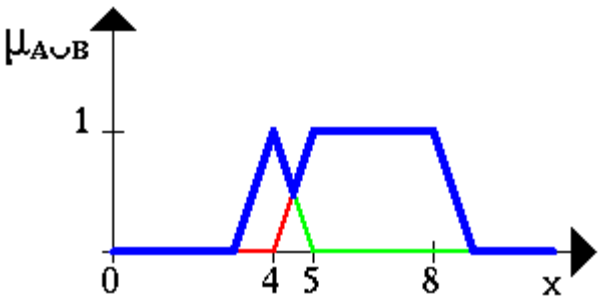


Figure 3 Union of two fuzzy sets. Adopted from ESRU (2014)

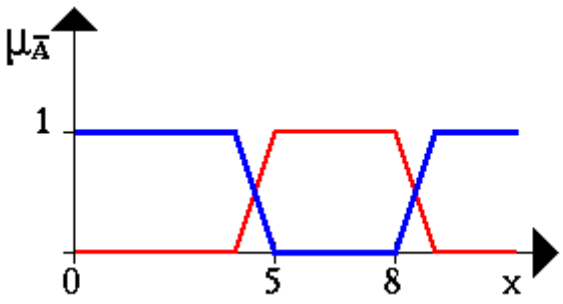


Figure 4 Negation of the fuzzy set A. Adopted from ESRU (2014)

1.1.2 Process of fuzzy logic system

The fuzzy logic system compose of three basic steps: fuzzification, fuzzy inference, and defuzzification (Dostál, 2011), (see Figure 5).

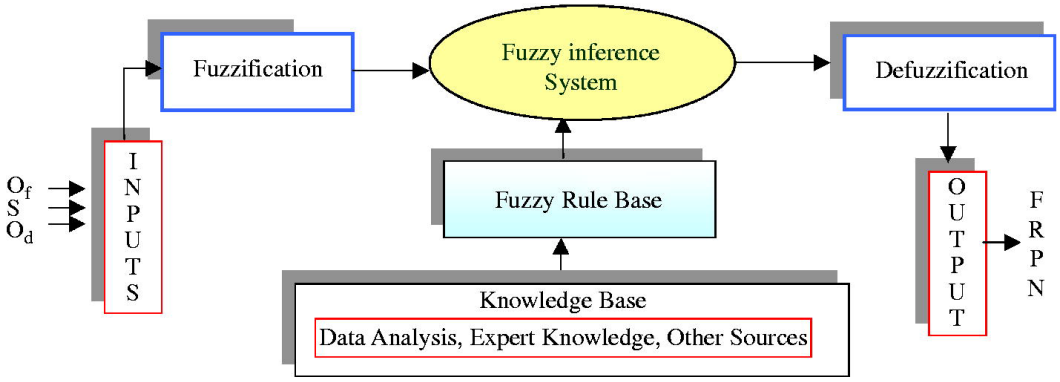


Figure 5 Architecture of fuzzy decision making system. Adopted from Emerald Insight (2014)

The first step (fuzzification of data) represents the transformation of language variables into numerical values. For instance, the variable could be characterized as very low, low, medium, high and very high. It is usually defined by three to seven attributes (terms). The degree of membership of attributes is determined by mathematical functions.

Nevertheless, there are many shapes and types of membership functions that are used. Both input and output variables are defined by attribute and membership functions (Dostál, 2011).

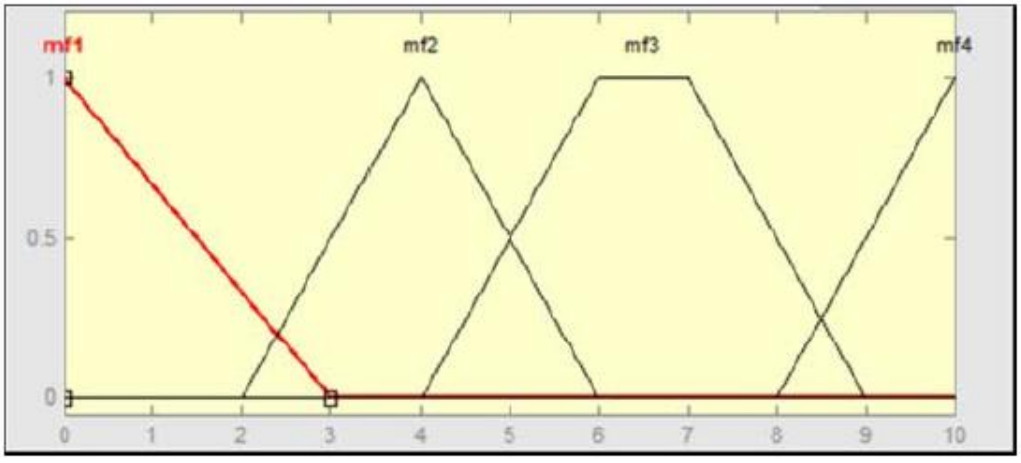


Figure 6 The types of membership functions Λ, Π. Adopted from Dostál (2011).

The second step (fuzzy inference) defines the system behaviour in terms of the rules such as <IF>, <THEN>, <WITH>. The fuzzy sets are essential to perform the fuzzy model based on that rule using an implication function. This implication functions, however, known as If-then true rule or called linguistic rule. The rules determine the input

and output membership functions that will be used in inference procedure. The fuzzy rules determine the fuzzy expert system. Furthermore, it is required to determine the weight of the rule in the system. It is allowed to change the weight rules during the process of optimization. The fuzzy rule base is usually constructed from the experience of the decision maker (Dostál, 2011).

For example, IF there is excellent software quality with a strong analyst capability THEN there must be less number of errors in the software. In this case excellent, strong, and less are fuzzy sets qualifying the variables software quality, analyst capability and number of errors respectively (Srivastava, Kumar, Singh and Raghurama 2010).

The third step (defuzzification) represents the reverse process of fuzzification. Defuzification is necessary to convert the output fuzzy values to linguistic values in order to present verbally the results of a fuzzy computing cycle. In the process of entering the data, the fuzzy logic system works as an automat (Dostál, 2011).

1.1.3 Contribution of fuzzy logic

As Zadeh (2008, p.2753) described, the most visible, the best understood and the most widely used contribution of fuzzy logic is the concept of a linguistic variable and the associated machinery of fuzzy if–then rules. But there are other equally important contributions which are much less visible and much less well understood. What is needed to understand the significance of these contributions is fuzzy logic in its non-traditional setting.

The machinery of linguistic variables and fuzzy if–then rules is unique to fuzzy logic. This machinery has played and is continuing to play a pivotal role in the conception and design of control systems and consumer products. However, its applicability is much broader. A key idea which underlies the machinery of linguistic variables and fuzzy if–then rules is centred on the use of information compression. In fuzzy logic, information compression is achieved through the use of fuzzy granulation (Zadeh 2008, p.2753).

In conclusion, fuzzy logic is described as an approximation process, in which crisp inputs are turned to fuzzy values based on linguistic variables, set of rules and the inference engine provided (Omran, 2010).

1.2 MATLAB

MATLAB is a high – level programming language and technical computing environment developed by MathWorks. MATLAB allows analysing data, developing of algorithms, plotting of functions and data and creating models and applications. The language and tools in MATLAB enable to use several approaches and reach a solution faster than with classical programming languages including C/C++ or Java (MathWorks, 2014a).

Moreover, the MATLAB deals with a range of applications, such as signal processing and communications, design and video processing, control systems, test and measurement. The language of technical computing is used by many engineers and scientists in industry and academia (Matlab, 2014a).

1.2.1 Fuzzy Logic Toolbox

The Fuzzy Logic Toolbox deals with functions, apps and a Simulink block focused on providing analysis, design and simulating systems built on fuzzy logic. The product is able to develop and analyse fuzzy inference systems and several methods like adaptive neurofuzzy inference systems and fuzzy clustering (MathWorks, 2014b; MathWorks, 2014c).

Basically, the toolbox is aimed at modelling comprehensive system behaviours using simple logic rules and shift these rules to a fuzzy inference system accordingly. Furthermore, the toolbox uses fuzzy inference blocks in Simulink and simulate the fuzzy systems within a complex model of the whole dynamic system. It is also possible to generate C code from Simulink for use in embedded applications that involve fuzzy logic. As all toolboxes in MATLAB, Fuzzy Logic Toolbox can be adjusted as well. Such revisions incorporate modifying of source code and algorithms, adding own membership or using defuzzification techniques (MathWorks, 2014b; MathWorks, 2014c).

Key features

- Fuzzy Logic Design app for setting up fuzzy inference systems and viewing and analysing results
- Membership functions for building fuzzy inference systems

- Support for AND, OR, and NOT logic in user-defined rules
- Standard Mamdani and Sugeno-type fuzzy inference systems
- Automated membership function shaping through neuroadaptive and fuzzy clustering learning techniques
- Capability of including a fuzzy inference system in a Simulink model
- Capability of generating embeddable C code or stand-alone executable fuzzy inference engines (MathWorks, 2014d).

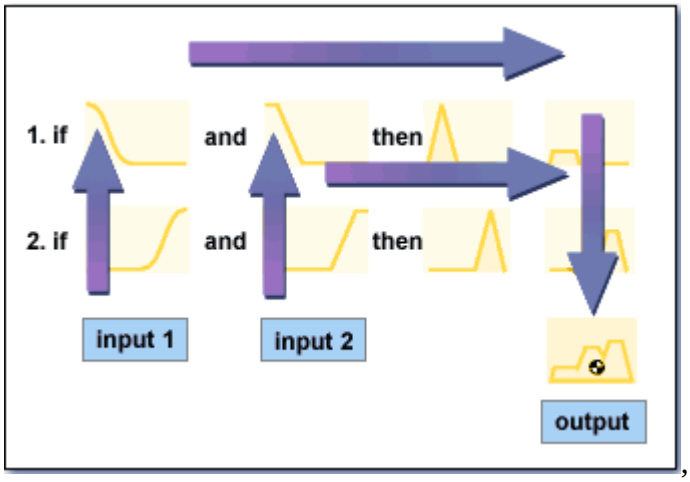


Figure 7 Fuzzy Inference Diagram. Adopted from MathWorks (2014e)

1.2.2 Fuzzy Inference System

Fuzzy inference is a method that interprets the values in the input vector and, based on user-defined rules, assigns values to the output vector. Using the editors and viewers in the Fuzzy Logic Toolbox, the rules set, definition of the membership functions and analysis of the behaviour of a fuzzy inference system (FIS) can be built (MathWorks, 2014f), (see Figure 8). The following editors and viewers are provided:

FIS Editor displays general information about a fuzzy inference system. Furthermore, the input and output membership functions, the rule base and the fuzzy operators can be defined, with the FIS editor (Klingenberg 2014; MathWorks, 2014f).

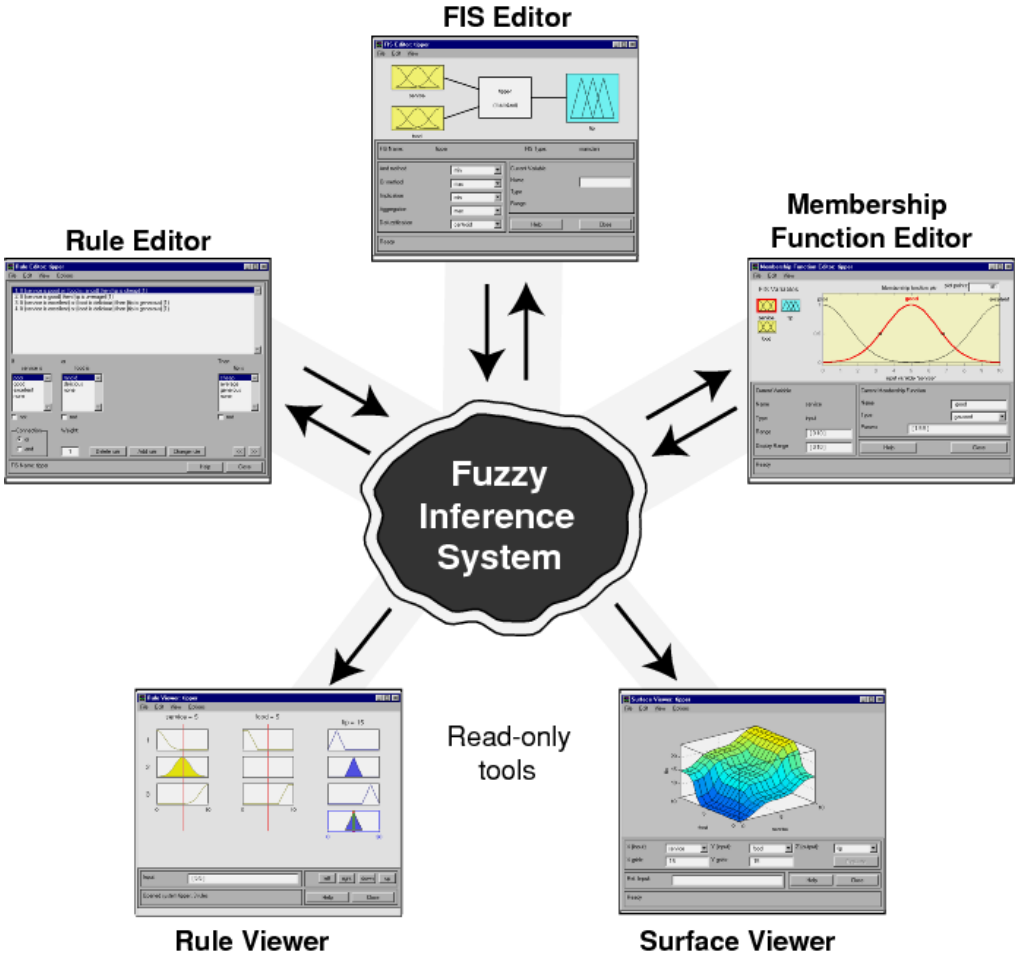


Figure 8 Fuzzy inference system. Adopted from MathWorks (2014g)

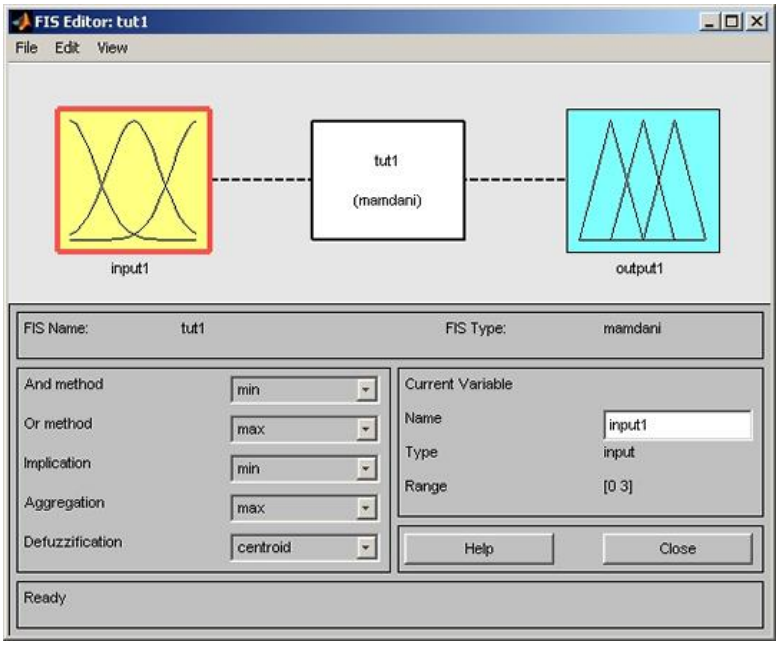


Figure 9 FIS Editor. Adopted from Klingenberg (2014)

Membership Function Editor demonstrates and edits the membership functions associated with the input and output variables of the FIS (MathWorks, 2014f).

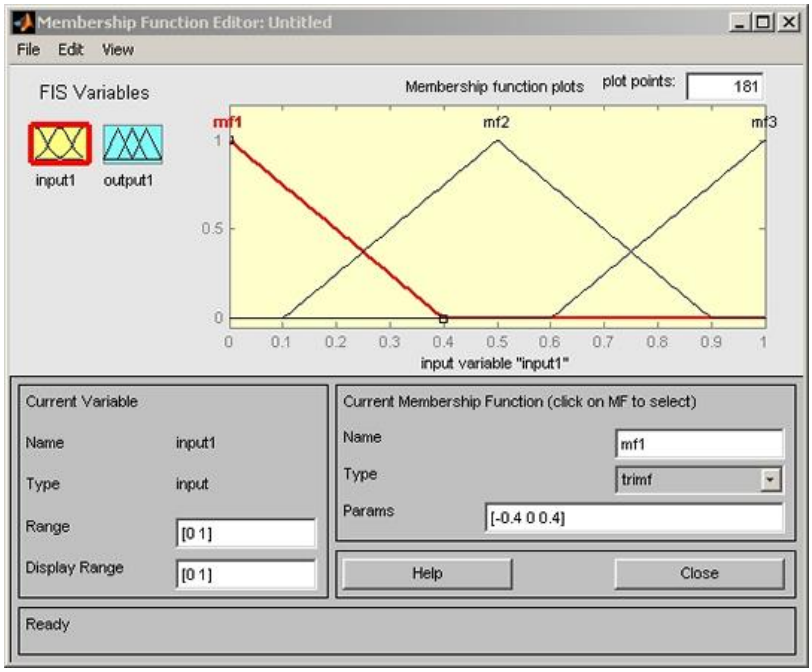


Figure 10 Membership Function Editor. Adopted from Klingenberg (2014)

Rule Editor enables to view and set up the fuzzy rules (Dostál, 2011).

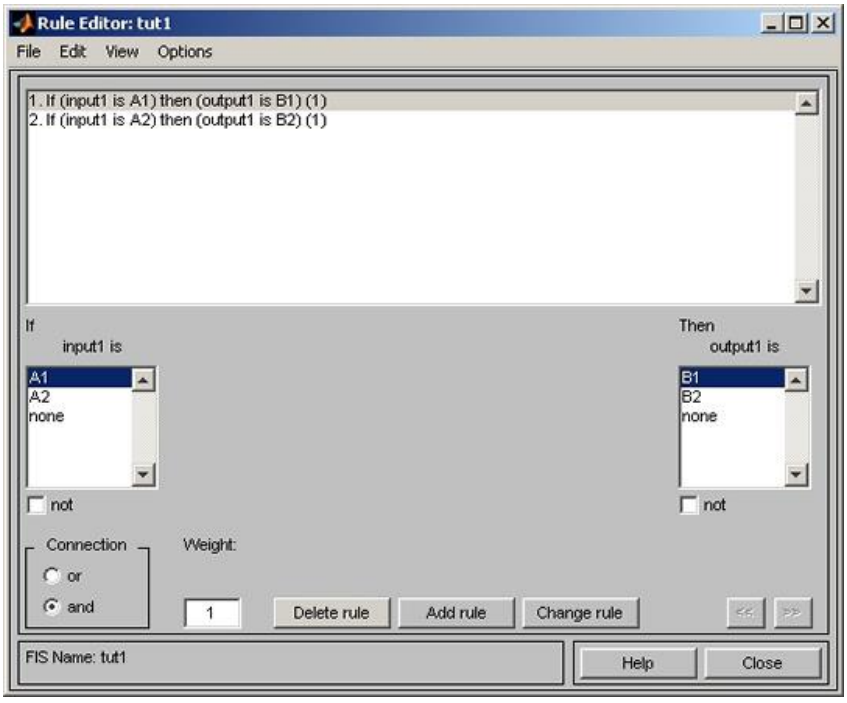


Figure 11 Rule Editor. Adopted from Klingenberg (2014)

Rule Viewer illustrates detailed behaviour of a FIS to help diagnose the behaviour of specific rules and enables evaluation of the dependence of the output on the values of inputs (Dostál, 2011; MathWorks, 2014f), (see Figure 11).

Surface Viewer generates a 3-D surface from input variables and the output of an FIS and displays dependence of single variables created by the rules (Dostál, 2011; MathWorks, 2014f), (see Figure 12).

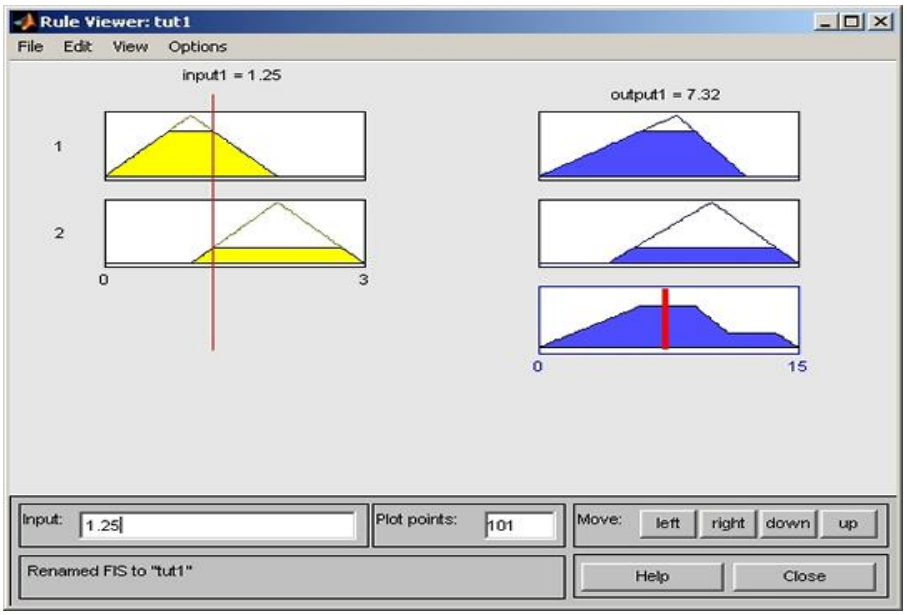


Figure 12 Rule Viewer. Adopted from Klingenberg (2014)

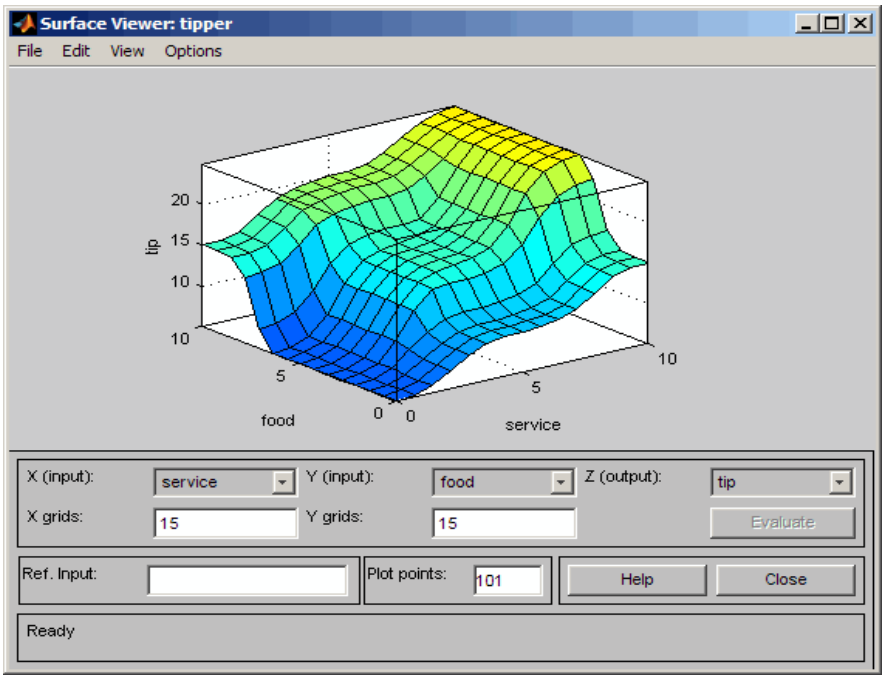


Figure 13 Surface Viewer. Adopted from MathWorks (2014g)

1.3 Software development

Basically, main objective of software development is customer satisfaction. According to Srivastava, Kumar, Singh and Raghurama (2010, p.183), software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software. The main aim of software engineering is to produce software at low cost with higher efficiency. Apart from the fact that the field is still relatively young compared to its sister fields of soft computing, there is still much discussions around what software engineering indeed is, and if it the title engineering is used properly. Software development area composes of several phases. However, software testing is defined one of the most important in all the phases of Software Development Life Cycle (SDLC).

Nevertheless, drawing on Rodrigez, Vizcino, Piattini and Beecham (2012, p. 664), Global Software Engineering (GSE) has become an increasing area of research, besides being an expanding trend in the Information Technology environment. GSE requires software tools (management tools, development tools, etc.) to encourage the specific features that this area has, and which have mainly come about as a consequence of the distance factor (temporal, geographic and socio-cultural distance).

Furthermore, modern software development, such as globally distributed teams, makes up particular challenges and risks (despite the benefits that can be gained) for the software field, which is essential to take into account. Actually, developing software systems through collaboration with other partners and in distinct geographical locations is a good opportunity for firms (Rodrigez, Vizcino, Piattini and Beecham, 2012).

Software tools for GSE should hence help to mitigate problems e.g.:

- **Geographic Dispersion,** which sometimes causes a loss of synchronous communication or team interactions, since the sites are in different time zones.
- **Control and Coordination Breakdown**, due to the difficulties created by a distributed environment.
- **Loss of Communication** - this is the case in this type of environment, if we consider that the richest communication medium is face-to-face communication.
- **Loss of Team Spirit** and trust among team members.

- **Cultural Differences** which occur when people from different cultures work together in a global environment (Rodrigez, Vizcino, Piattini and Beecham, 2012).

Tools designed to alleviate the challenges stated above should hence comprise unique characteristic, for instance supporting the interaction of distributed teams by applying communication and collaboration technology, supporting the development of real-world projects, minimizing the cost of the tools and infrastructure needed, together with their maintenance effort or helping to make up a feeling of trust between the members, and facilitating the knowledge of team ethics within the others.

However, there is lack of information considering which tools are able to help in the aforementioned challenges, or about which specific tools offer characteristics that are appropriate to allow them to be used in a GSE area. The most that we can state is that certain surveys exist in which some of the existing tools, usually those with regard to collaboration, are shortly demonstrated. A good instance of this is in which the authors present a set of collaboration tools for GSE, classified by the fields in which they can be used (Rodrigez, Vizcino, Piattini and Beecham, 2012).

As Kelkar (2009) described, software represents both computer programmes and related documentation together. It has impact on all areas of knowledge. In general, the system plays a dual role. It is a product by itself (information ''transformer''). It represents the ''vehicle'' for delivering other products (supporting system functionality, controlling other programmes).

Characteristics of a good software:

- Maintainability
    - The information system must allow for changing requirements.
- Dependability
    - Software must be reliable.
- Efficiency
    - The system resources should be saved.
- Usability
    - Software should be applicable by the users for whom it is constructed.

1.3.1 Software development life cycle

The development of an information system demands the commitment of valuable company resources and time. Large projects often require much more effort and take years to complete (Everett and McLeod, 2009).

In fact, software development life cycle (SDLC) includes several phases which generally involves the planning, definition, requirements, design, building, implementation, testing and maintenance. Under each of these phases, IT professionals or project leader needs to come up with deliverables, which depend on specifications of the software system or the project itself. For instance, within the planning phase of SDLC, there are various deliverables which are needed. As the planning phase includes a high-level view of the software project, a set of aims is required to be written down. This also involves information about the financial resources. Therefore, the deliverables may include documentation like the SDLC templates (Lewis, 2008).

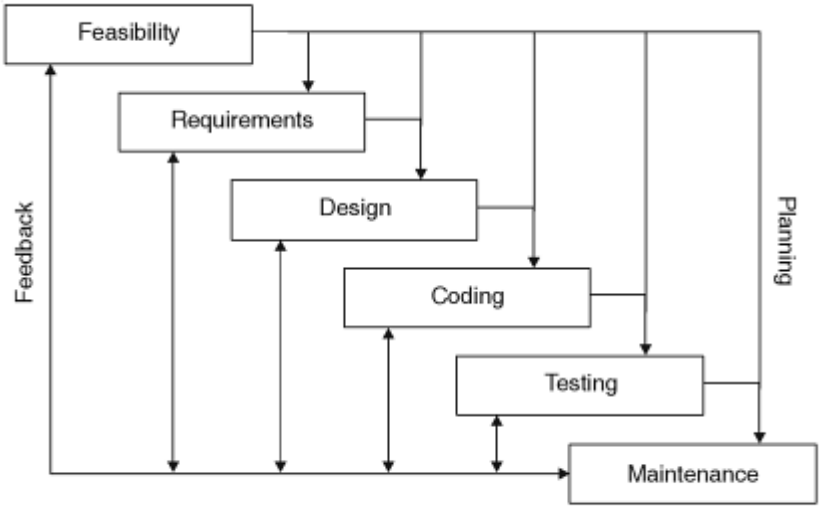


Figure 14 Iterative waterfall model. Adopted from (Kelker, 2009)

Within the definition stage, the deliverable represents a documentation which indicates the project plan. Within the requirements phase, the following deliverables might be demanded: a business process model (business proposal), requirements for information system, standards for the data architecture and analysis on how data will be transformed. The last two stages which involves building and implementation may need deliverables

as application forms, tested applications, site configuration, user training plan and software delivery (Lewis, 2008).

1.3.2 Specifying software requirements

Lewis (2008) described that the software system requirements may differ from company to company, but the main aim is clear. System's requirements aim to standardize the set of practices used in developing an information system, as that development will be both cost – effective and feasible.

Therefore, establishing the requirements for a software product is a significant undertaking and directs the course of action for the remaining software development effort. Traditionally, requirements specifications address the overall product under development and its external interfaces. However, an important practice employed by most engineering disciplines is the specification of requirements for every element of the product architecture or design. Therefore, there are significant implications with this practice that demand that the complete software architecture be formulated, including a specification for each element of the software product and associated post-development sustainment processes (Schmidt, 2009, p.10).

Drawing on Schmidt (2009, p.10), the software requirements specifications for the product guide the definition of the product architecture, software implementation, and software test and evaluation efforts. Requirements that are nonessential, over-specified, or introduce unacceptable risks place the project in jeopardy of being unsuccessful. This represents a situation where the software development team may attempt to do too much with too. Projects are constrained by the amount of resources available to produce a product. Project budget and schedule objectives must be the primary focus when establishing product requirements.

However, every software product is intended to serve a purpose and the software requirements should represent those product features and performance factors that enable the product to serve its purpose. Software products may support a business process, control the operation of a system or process, support data gathering and analysis activities, guide work productivity by automating mundane tasks, or provide some entertainment relevance. Thus, there exists a significant cost-benefit motivation for every software development undertaking that must be appreciated. Caution must be taken when

establishing software requirements that broaden the scope of the development effort beyond the means of the project to achieve its objectives. Improperly extending the software product scope sets the development effort on a path destined for failure. Every requirement implies a level of effort necessary to devise a suitable solution. Managing the scope of the software engineering undertaking is essential to the success of each and every development project (Schmidt, 2009, p.10).

1.4 Software Project Environment

The effective and profitable execution of a software engineering project involves an understanding of the complex interactions and dependencies inherent in the project environment. This knowledge must be fortified with a set of supervisory tools that provide information concerning the current status of tasks and work products. This information contains obscure symptoms of potential situations that threaten the project's success or software product's quality and competitiveness in the marketplace. Software engineering exploits this information to permit its attentive practitioners to recognize disruptive trends and react in a positive manner to neutralize the root causes of problematic conditions (Schmidt, 2009, p.55).

There are three fundamental management tools that are used to guide a project toward successful completion. The first is the integrated master plan (IMP), which identifies the organizational roles and responsibilities, tasks to be performed, and expected outcomes. The second is the integrated master schedule (IMS), which provides a timeline of key events, milestones, reviews, and decision points. And finally, there is the project budget, which identifies the resources that are allocated to each organization to enable the execution of planned tasks. However, these project management instruments must be properly developed, monitored, and adjusted to reflect the ambiguity inherent in task estimation. Initial planning forecasts of anticipated productivity, performance, and results must account for project uncertainty (Schmidt, 2009, p.55).

Software development projects are established with the aim of delivering a "new" software product to one or more customers. Therefore, until the software product definition is relatively complete, the project plans will always be imprecise. This implies that the project plans, schedules, and budgets are simply tools that direct the project team toward the definition, design, implementation, testing, documenting, and delivery of a

software product. The dilemma faced by the project team is determining how to define the software product in such a manner that the project goals and objectives can be satisfied. Inherent in this situation is the fact that project plans, schedules, and budgets are simply a means to an end to the successful delivery of a software product on time (according to schedule) and without exceeding authorized funding thresholds (according to budget). As long as the project team can define and deliver an acceptable software product by the delivery date and does not expend more resources than authorized, the project should be deemed successful (Schmidt, 2009, p.56).

Within the project environment there exists a variety of decision points that represent opportunities to maintain the project scope so that goals and objectives can be attained. Software engineering practices and tools are structured to recognize when the definition of the software product presents an opportunity to revisit the project plan. At each opportunity, a decision must be made on which way to proceed among alternative approaches. Making proper architectural design decisions involves the following factors:

- Understanding the product functions and characteristics that are important to stakeholders (requirements analysis).
- Determining how each product characteristic will be provided (functional analysis and design synthesis).
- Identifying which design approach best serves the current product stakeholders and the envisioned stakeholder community or customer base (trade-off analysis).
- Eliminating unknown conditions that improve the likelihood of achieving project and product objectives (risk assessment).
- Ensuring that every function or characteristic is necessary to the operation of the product and not in excess of what is needed (verification and validation).
  Controlling product complexity to simplify software operational and support costs (integrated product and process development, IPPD).
- Refining technical and project plans, schedules, and budgets to reflect the selected course of action (control) (Schmidt, 2009, p.57).

Fundamentally, the software product architecture determines the project effort necessary to successfully implement, test, deliver, and support the product throughout its life cycle. If the project definition is allowed to drive the software product definition, then the product may be less beneficial and noteworthy in a competitive environment.

The project scope must be aligned to provide the resources (personnel, facilities, equipment, tools, budget, schedule, etc.) necessary to define, design, implement, test, and deliver the software product to its customers. The software product must be developed to accommodate the needs and expectations of all stakeholders, including users, support staff, training staff, investors, and enterprise management. When the product definition and project scope are unbalanced, then the software engineering, technical, and project management teams must collaborate to stabilize the situation (Schmidt, 2009, p.57).

The software engineering effort represents the total technical effort within the project scope. As such, the software engineering leadership is responsible for defining the software product architecture in a manner that is consistent with the project scope. When it is perceived that the product value to its customers (consumers, operators, investors, etc.) can be enhanced with the application of additional project resources, then change proposals are generated to establish the merit of the enhancement. This occurs whenever the enhancement cannot be accommodated within the established project cost and schedule objectives. Figure 10 depicts the role of software engineering within a project environment (Schmidt, 2009, p.57).

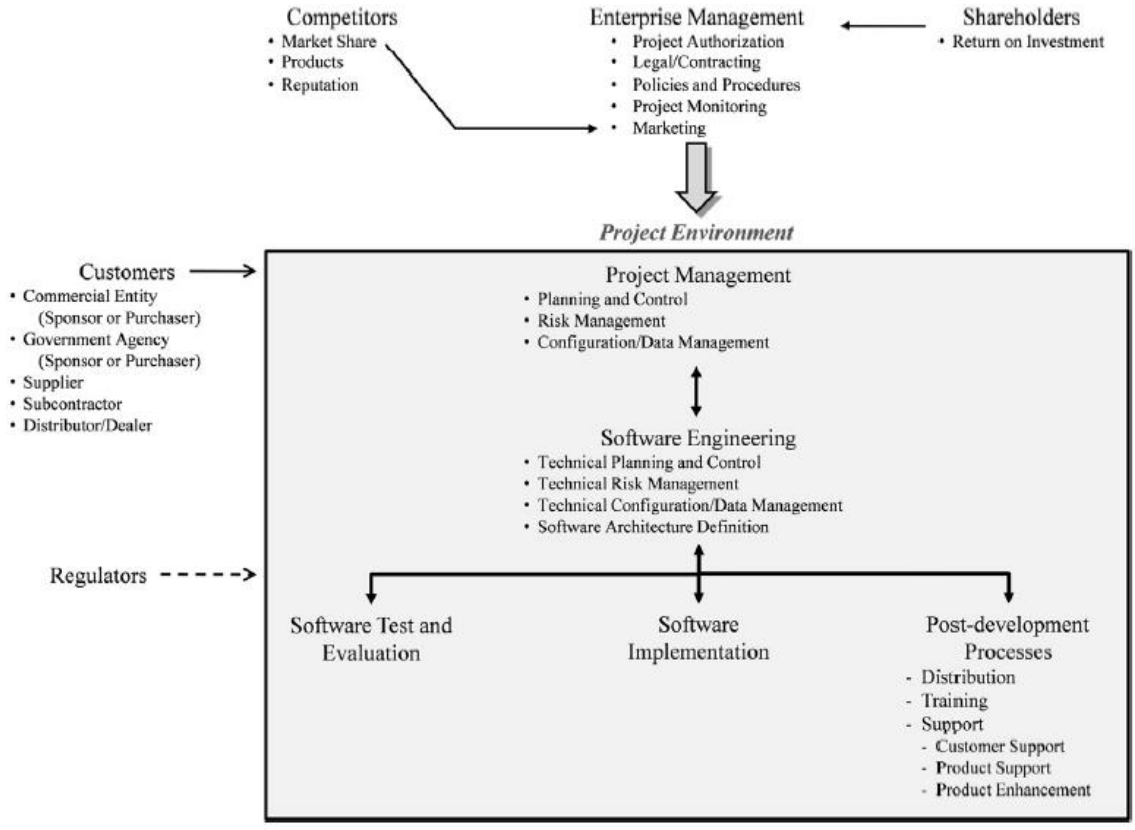


Figure 15 Role of software engineering within a project environment. Adopted from (Schmidt, 2009, p.58).

1.5 Software testing

Software testing can be defined as the execution of a program against Test cases with the intent of revealing faults. The different testing techniques are defined based on the artefact used to derive Test cases. Functional – or black-box – testing derives Test cases from the specification or description of a program; structural – or white-box – testing derives Test cases from implementations; fault-based testing derives Test cases from fault models based on common mistakes committed by programmers; and model-based testing derives Test cases from system specification models. To deem a software system *correct*, one could test every possible element of the system's input domain and check whether the output is consistent with the expected output (Lemos, Ferrari, Eler, Maldonado and Masiero, 2012).

However, even for simple programs this is usually infeasible, because the input domains tend to be very large (imagine, for instance, the input space of a compiler system). Therefore, a large portion of testing research focus on proposing ways to select meaningful subsets of Test cases to enhance the chance of revealing faults. Based on the categories of testing techniques described above, several testing selection criteria were proposed (Lemos, Ferrari, Eler, Maldonado and Masiero, 2012).

Besides testing techniques and criteria, there are many other aspects involved in the testing activity. For instance, in general, it is too expensive to test programs manually; therefore, software testing usually relies on tools to automate the Test case generation, execution, and results gathering. After faults are revealed while testing the programs, they must be localized and fixed. This activity is usually not included under the *software testing* activity, being called *debugging* (Lemos, Ferrari, Eler, Maldonado and Masiero, 2012).

Since it is closely related to testing, we decided to include papers concerned with it in our survey. Other topics that are important to software testing and were included are the following: *fault-injection*, which consists in intentionally introducing known failures into the system during its execution to evaluate if the system is robust enough to recover without crashing *regression testing*, which consists in selectively retesting a system to verify whether modifications have not caused unwanted effects and *testing strategy*, which consists in the way by which Test case design methodologies are combined to

provide an effective testing activity (Lemos, Ferrari, Eler, Maldonado and Masiero, 2012).

Software Testing is an important process of software development which is performed to support and enhance reliability and quality of the software. It consists of estimating testing effort, selecting suitable test team, designing Test cases, executing the software with those Test cases and examining the results produced by those executions. Studies indicate that 40-50 percent of the cost of software development is devoted to testing, with the percentage for testing critical software being even higher (Lemos, Ferrari, Eler, Maldonado and Masiero, 2012).

As such software testing is the process of validation and verification of the software product. Effective software testing will contribute to the delivery of reliable and quality oriented software product, more satisfied users, lower maintenance cost, and more accurate and reliable result in day to day working environment of software professionals. However, ineffective testing will lead to the opposite results, low quality products, unhappy users, increased maintenance costs, unreliable and inaccurate results.

Hence, software testing is a necessary and important activity of software development process. Myers states that "Software Testing is the process of executing a program with the intent of finding errors". The importance of testing can be understood by the fact that "around 35% of the elapsed time and over 50% of the total cost are involved in testing programs" (Srivastava, Kumar, Singh and Raghurama, 2010, p.183).

Practitioners are generally short of time or resources and tend to perceive systematic testing as not so very lucrative job. However, it affects overall software life cycle, because quality of software life cycle depend upon testing technique demanding adequate Test case preparation, modeling, and documentation which make the process complicated and challenging. These impending challenges have to be addressed by researchers and practitioners working closely together by estimating the amount of effort that is required to develop user-friendly software (Srivastava, Kumar, Singh and Raghurama, 2010, p.183).

1.5.1 Testing Types

Manual testing

Manual testing involves the testing of the software manually for instance without using any automated tool or any script. Therefore, the tester takes over the role of an end user and test the software to detect any unexpected behaviour or bug. Manual testing includes various levels like Unit testing, Integration testing, System testing and User Acceptance testing (Khan, 2011; Tutorialspoint, 2014).

In order to test the software and ensure the completeness of testing, testers use test plan, Test cases or test scenarios. Manual testing also includes initial testing as testers investigate the software to determine defects in it (Khan, 2011; Tutorialspoint, 2014).

Automation testing

Automation testing also known as "Test Automation" includes software testing when the tester writes scripts and utilizes another software for testing. This process incorporates automation of a manual process. Automation testing aimed at rerunning the test scenarios that were performed manually, quickly and repeatedly. Besides regression testing, automation testing is also utilized to test the application from load, performance and stress point of view. It growths the test coverage, improve accuracy, saves time and money by comparison to manual testing (Khan, 2011; Tutorialspoint, 2014).

However, it is impossible to automate everything in the software. Hence, the areas at which user can make transactions such as login form or registration forms etc., any area where large amount of users' can access the Software simultaneously should be automated. Moreover, all GUI items, connections with databases or field validations could be efficiently tested by automating the manual process (Khan, 2011; Tutorialspoint, 2014).

1.5.2 Levels of testing

Software testing is the process of accessing the functionality and correctness of a software through analysis. It also identifies most important defects, flaws, or errors in the application code that must be fixed. The system must be tested in steps with the planned build and release strategies. The key to successful testing strategies selecting the right

level of test at each stage in a project. The level of testing have a hierarchical structure which build up from the bottom-up where higher level assume successful and satisfactory completion of lower level test. Each level of test is characterized by an environment i.e. type of people, hardware, data etc. and these environmental variables vary from project to project. Each completed level represent a milestone on the project plan and each stage represents a known level of physical integration and quality. These integrated stages are known as level of testing (Khan, 2011).

Unit Testing

Unit testing represents the first and the lowest level of testing. In this level, respective components of software are tested. Unit testing is performed by individual developer on individual units of source code assigned areas. The aim of unit testing is to separate each part of the programme and show that individual parts are correct in terms of requirements and functionality. Therefore it helps to expose defects that might be hidden (Khan, 2011).

However, there are certain bounds of scenarios and test data that the developer can use to verify the source code. So when the developer exhausts all options there is no choice but to stop unit testing and unify the code segment together with other units (Tutorialspoint, 2014).

Integration Testing

Integration testing represents the level after unit testing where either the developer or an independent tester performs testing. The goal of integration testing is to test combined parts of a software and determine if they function correctly together. Furthermore, integration testing aimed at verifying functional, performance and reliability requirements placed on major design items. The importance of integration testing must not be overlooked due to the fact that approximately 40% of software bugs are exposed during testing. There are two types of integration testing:

- Bottom-Up Integration testing
- Top-Down Integration testing

In a comprehensive software development environment, Bottom-Up testing is usually done first, followed by Top-Down testing (Khan, 2011; Tutorialspoint, 2014).

System Testing

System testing begins after completion of integration testing. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets Quality Standards. System testing represents the first step in Software Development Life Cycle, where the application is tested as a whole. The application is tested in an environment which is very close to the production environment where the application will be deployed. System Testing enables us to test, verify and validate both the business requirements as well as the Applications Architecture. This type of testing is performed by a specialized testing team if there is one (Oladimeji, 2007; Tutorialspoint, 2014).

Acceptance Testing

In software engineering acceptance testing is a level of software testing where the system is tested for user acceptability. This is arguably the most important type of testing as it is performed by the Quality Assurance Team who will appraise whether the application meets the intended specifications and satisfies the client's requirements. Acceptance testing is performed after system testing and before making the system available for actual use.

Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors or Interface gaps, but also to point out any bugs in the application that will result in system crashers or major errors in the application. By performing acceptance tests on an application the testing team will deduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system (Khan, 2011; Tutorialspoint, 2014).

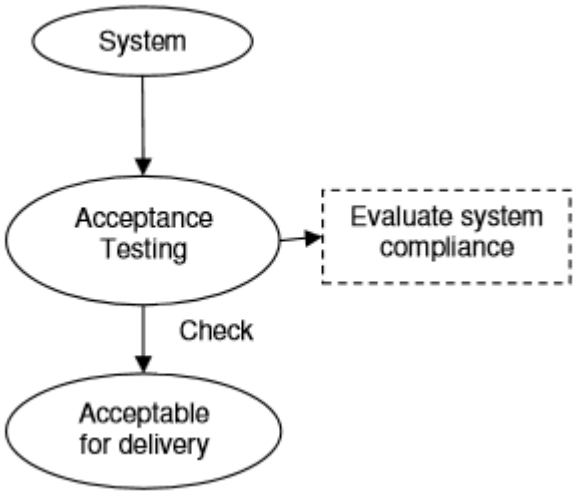


Figure 16 Acceptance testing. Adopted from Khan (2011).

Regression Testing

Unlike the previous levels of testing discussed, regression testing spans through the testing phase. Important reason for regression testing is that it is often extremely difficult for a programmer to find out how the changes in one part of the software effects the other part. Hence, regression testing is carried out whenever the system is modified either by adding new components during testing or by fixing errors. Its goal is to determine if modification to the system has introduced new errors in the system.

Therefore, the quality of a system is directly connected to good regression testing. Furthermore, regression testing is a very important aspect of the system maintenance. There are three types of regression testing techniques namely selection, prioritization and minimization (Oladimeji, 2007; Khan, 2011; Bhasin, Gupta and Kathuria, 2013).

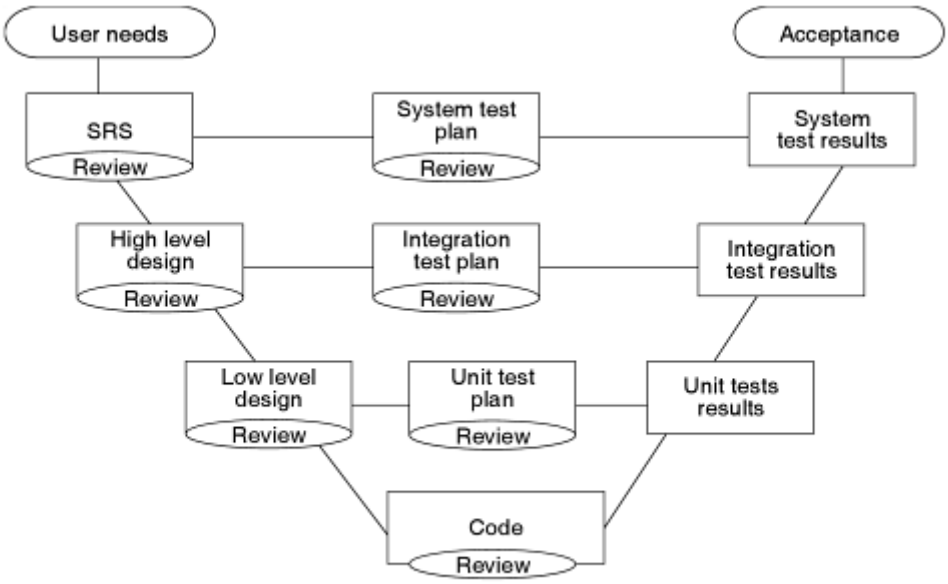


Figure 17 Verification, validation and testing: schematic. Adopted from Kelkar (2009, p. 31)

1.5.3 Test case Development

As a tester, the best way to determine the compliance of the software to requirements is by designing effective Test cases that provide a thorough test of a unit. So basically, a Test case represents a detailed procedure that fully tests a feature or an aspect of a feature. While the test plan describes what to test, a Test case describes how to perform a particular test. Therefore, it is needed to develop Test cases for each test listed in the test plan. Set of Test cases is called Test case suite (Bhasin, Gupta and Kathuria, 2013; Symbiosys Technologies, 2013).

General Guidelines

Moreover, various Test case design techniques enable the testers to develop effective Test cases. Besides, implementing the design techniques, every tester needs to keep in mind general guidelines that will aid in Test case design:

- The purpose of each Test case is to run the test in the simplest way possible.
- Concentrate initially on positive testing i.e. the Test case should show that the software does what it is intended to do.

- Existing Test cases should be enhanced and further Test cases should be designed to show that the software does not do anything that it is not specified to do i.e. Negative Testing
- Where appropriate, Test cases should be designed to address issues such as performance, safety requirements and security requirements
- Further Test cases can then be added to the unit test specification to achieve specific test coverage objectives. Once coverage tests have been designed, the test procedure can be developed and the tests executed (Symbiosys Technologies, 2013)

1.5.4 Prioritization of software testing

The prioritization of Test case becomes all the more important owing to the fact that it is not feasible to run all the Test cases after each and every change. Once a change is made it is not possible to retest all the Test cases of the test suite as it will consume lot of time. Therefore, prioritization of the Test cases has been widely proposed and used in recent years as it can improve the rate of fault detection during the testing phase (Chaudhary, Sangwan and Singh, 2012).

Generally, prioritization is used when the time for the testing is limited. In order to attain maximum coverage, the more important cases are tested. However for this purpose, fuzzy expert system should be selected because of better decisions made by it in comparison to the normal expert system. Basically, fuzzy expert system provides a better way of prioritizing the Test cases (Bhasin, Gupta and Kathuria, 2013).

In the work conducted by Zhewei Xu, Kehan Gao and Taghi M Khoshgoftaar, a fuzzy expert system has been proposed so as to select the Test cases when information of the source code is not available to testers. The system takes different Test cases as inputs and determines test importance accordingly (Bhasin, Gupta and Kathuria, 2013).

2 PROBLEM ANALYSIS AND CURRENT SITUATION

2.1 Company profile

BIAC GmbH (Business Insurance Application Consulting) is the IT and SAP Competence Centre primarily of the VIG (Vienna Insurance Group). The company supports the business processes of its clients of VIG and also for clients outside the VIG group with solutions based mainly on SAP. The service extends from consulting to the correct application up to development of specific solutions for the insurance industry, including planning, implementation, support and training and constant updating (BIAC, 2014a; BIAC 2014b).

BIAC basically offer holistic solutions with several complex services – from Customer consulting to Project Management to Hosting services and Service levels. The company enables to offer the services separately as well. The approach is based on the developing all systems in such a way as to offer the highest possible quality within a business framework. The main focus lies clearly on SAP Insurance and its surrounding systems (BIAC, 2014a; BIAC 2014b).

History

1985 - Separating of the IT division from WIENER STÄDTISCHE Versicherung and founding of Metropolitan Datenservice GmbH.

2005 - Foundation of BIAC Business Insurance Application Consulting as legal successor to Metropolitan Datenservice GmbH and to Central Point Insurance IT-Solutions (CP)

2008 - Foundation of the wholly-owned subsidiary B&A in Ostrava, CZ

2010 - Acquisition of 30 % of AIS s.r.o. in Brno, CZ

2010 - Integration of TBI-Info in Sofia, Bulgaria for the implementation of the SAP in Bulgaria (BIAC, 2014a)

2.2 BIAC Services

Project Management

In general, taking into consideration systematic project management, it is the vital requirement for the successful management and implementation of projects along the way of all their stages. With regard to BIAC's company, a project contract is developed for each project and mutually agreed to the company and the customer. The attainment of the project aims depends on all the phases of the project, from planning, through implementation and testing up to its successful conclusion.

Therefore, starting with the project requirements as defined by the client, a description of the project (a business case) with project plan, description of approach, extent, dependencies as well as a schedule with clear phases and milestones, including coordination of third parties and a technical design concept of the subject of the project is designed. The project management team thereby takes over control of the project with the establishment of the processes, the management of the extent of the project taking into account possible changes in requirements, quality control, schedules, costs and dependencies (BIAC, 2014c).

Test Management

Test management supports the activities of planning and test development. Hence, proper test management is critical to the success of a project. It covers the areas of test organisation, planning, automation and stand-alone active test implementation, test data management and reporting of the test results. The task of the cross function "Test Coordination and Test Automation" is the support of the development team in all test activities.

Within the framework of test organisation all activities are planned, documented and agreed with the development teams and specialist departments; the infrastructure is prepared, the testers are instructed about test scenarios and so on. It develops the information interfaces between the specialist department/s, BIAC and external providers. Besides that, other tasks include monitoring and reporting of test activities and test evaluation for the project leadership and overall product leadership (BIAC, 2014d).

Support services

All current requirements are covered with quick response times by the support services. Whole process starts with the ticket registration and processing at the support desk up to the Key User support and User Management. In addition, other important support services consist of change-request registration, its processing and evaluation and problem processing for production releases. However, the major concerns also dealt with IT security and ensuring of knowhow through training (BIAC, 2014e).

There are also further support services incorporated such as:

- First and Second Level Support
- MSA Application Support
- Third Level Support
- SAP Userline (BIAC, 2014e).

Business consulting

Business consulting offers support with the analysis and development of the specialist tasks and blueprints and, together with the client, defines the requirements in a prospective integrated solution. Superlative industry knowledge along with deep knowhow of process solutions in SAP applications together make up the basis for the realisation of sophisticated client solutions (BIAC, 2014f).

Solution consulting

In Solution consulting the step-by-step realisation of the IT technical client solution on the basis of the formulated specialist requirements takes place. SWD Software Development provides professional support and conduct in all software development phases. The services provided include support and advice in the creation of technical blueprints, SAP and non-SAP software development, test, quality and performance management, as well as migration support and the handover of the developed software into production systems (BIAC, 2014g).

Architecture

The function of the architecture is to view the application and project portfolio in its entirety and to highlight potential for improvement. Whilst doing so it also has in mind the TCO (Total Cost of Ownership) of IT to the VIG. This happens mainly in the framework of duties associated with projects under the title of the Architecture Board, when needed also through timely reviews in the framework of an Architecture Check. Concrete duties in close collaboration with the projects within the framework of a project include:

- Establishment of the current and desired architecture of the application affected by the project
- Impact analysis on associated applications
- Checking the list of applications affected by a project for its completeness
- Definition of cross over scenarios and the necessary interfaces for them
- Identification of intersections with already existing functionalities or functionalities developed in parallel in other projects
- Standardisation of the entire functionality and ensuring its use
- Separation of specialist requirements into components in special cases
- Checking the compliance of architecture standards
- Clarification of technical and organisational effects of requirements on operations and costs

Functions of architecture not directly involved in the project include: architecture standards, suggestions for process optimisation regarding project development (e.g. early definition of hardware requirements) and suggestions for the reduction of the Total Cost of Ownership (TCO) for projects (BIAC, 2014h).

Quality Management

Quality Management tasks include:

- Description of the enterprise from a systemic, integrated point of view (QM manual; guidelines, templates, BIAC Q-targets/standards)
- Creating of templates for QM documents (reports, procedure instructions, enquiry catalogue, checklists etc.)
- Carrying out audits (projects, QM systems etc)
- Quality assurance related to development (checks for project documents such as PHB, PMHB etc) coordinated with the relevant BIAC and SAP QA activities (QA KPIs)
- Providing an integrated description of the as-is situation of the enterprise (management review)
- Support with the development/derivation of Q targets/SLAs/KPIs, reporting, Q gates
- Platforms for improvements (e.g. KVP etc.)
- To provide a description of how our clients/contact persons see us (client feedback)
- Involvement in all activities and tasks in order to maintain the QM relevant points/tasks/topics
- Competent contact partner for enquiries about the relevant standards (ISO 9000ff; 19011; 10005ff)
- Supporting standardization. Input for the topic of Corporate Identity and formal minimal requirements for documents (e.g. version control)
- Interface to Q certification organisations
- Offers of the relevant training
- Monitoring easy traceability of relevant documents (guidelines, templates, lists of documents) and delivery of input for improvement
- Making tools for quality control (CAST, Code Inspector) in important core processes available and overall monitoring
- Contact for modifications and owner of the affected processes (BIAC, 2014h).

2.3 Modules in SAP Insurance system

FS-PM – Policy Management

SAP Policy Management is a division-overlapping policy management system which is suitable for both regional and global market oriented insurance enterprises. This is ensured by its inclusion within the SAP landscape (BIAC, 2014i).

msg.PM – Product Management

msg.PM (Product Management) is used for insurance product calculations (rates, tariff checks, balance sheets) across all lines of business for SAP Insurance modules. It is always used in combination with FS-PM (BIAC, 2014i).

FS-CD – Collection and Disbursement for payments in/out

FS-CD manages collection and disbursement tasks across all lines of business, including current accounting, payment processing, incoming payment processing, correspondence and dunning. It also displays key account, corporate, broker and coinsurance business information (BIAC, 2014i).

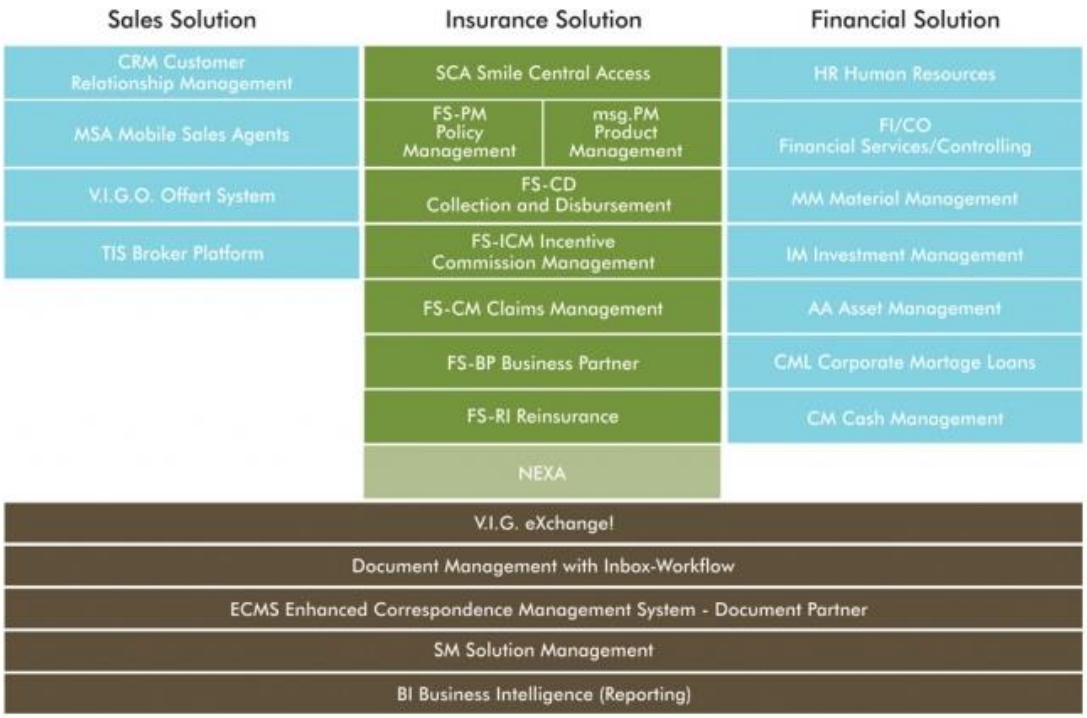


Figure 18 SAP Modules. Adopted from BIAC (2014i).

FS-ICM – Incentive Commission Management for commissions

The SAP Module FS-ICM is a cross sector solution that allows companies to manage all types of commissions and incentives paid both to employees and to partners. It provides up-to-date and transparent information about all previously earned and expected commissions and incentives. FS-ICM is a management instrument used to realise strategic corporate goals using monetary and/or non-monetary incentives, e.g. by increasing sales, by improving quality, reducing costs or other forms of adding value (BIAC, 2014i).

FS-CM – Claims Management for damages/indemnification

The SAP module FS-CM is used to set up and manage settlement claims for non-life insurance policies and benefit entitlement claims for life insurance policies (BIAC, 2014i).

FS-BP – Financial Services Business Partner for the management of partner data

FS-BP is used to store and manage information on business partners in a central application. This can be of particular interest when a company has more than one business relationship with a specific partner, e.g. as an existing supplier and a prospective customer. FS-BP utilises information technology benefits (e.g. data integrity and the elimination of redundant data) and, at the same time, facilitates customer service and new customer acquisition (BIAC, 2014i).

FS-RI – Reinsurance

The FS-RI module allows comprehensive management of active and passive reinsurance policies and contracts. Reinsurance is insurance for insurers. Reinsurance means that the direct insurer has transferred part of the policy risk, payments and premiums to another insurer (the reinsurer). The reinsurer in turn balances its risk by taking over the “risks” of several direct insurers. FS-RI has been designed for use in both active and passive reinsurance and offers direct insurers, reinsurers and agents flexible means of managing and administrating their reinsurance policies (BIAC, 2014i).

2.4 Process of software testing in BIAC

2.4.1 Test Management tasks

As already mentioned above, Test management in BIAC supports the activities of planning and test development as software testing is a necessary and important activity of software development process. In general, effective software testing will contribute to the delivery of reliable and quality oriented software product, more satisfied users, lower maintenance cost, and more accurate and reliable result in day to day working environment of software professionals.

Therefore, proper test management is critical to the success of a project. As a tester, the best way to determine the compliance of the software to requirements is by designing effective Test cases that provide a thorough test of a unit. So basically, a Test case represents a detailed procedure that fully tests a feature or an aspect of a feature. In other words, Test cases are flows or sequences of so-called Test Steps which are processed in the test object.

The test process, roles and responsibilities that are in place are defined with the support of renowned consulting firms and correspond to the customary international standards (ISO 9126). The members of the test management team are also trained and certified to the customary international standards (ISTQB). The test processes and tools used correspond to the standard recommendations for SAP implementation projects (BIAC, 2014j).

2.4.2 Test coordination

Moreover, there are a number of activities that must be carried to ensure tests are organised effectively. The careful preparation of test concepts and joint planning and agreement on test activities between the development team and the department is the first step in this process and should be based on defined and established test processes (guidelines smile test concept) (BIAC, 2014j).

Test organisation also extends to the preparation of the test infrastructure, which includes the organisation of rooms, equipment, systems, user rights and software installations. The nominated testers are offered wide support in the test preparation phase. This ranges from the provision of test documentation templates, advice on how to

describe test scenarios and test cases, risk analysis and test case prioritisation, roles and rights assignment, training in how to use the test tools (SAP Solution Manager, Support Desk), as well as guidance in preparing reports and preselecting test data.

With regard to test plans and test packages (work lists for testers), they are provided at the end of the test preparation phase. During the actual test process, the test coordination team serves as the information interface between the department(s), BIAC and the external providers.

Further tasks carried out during the test process include test status tracking, the monitoring of test and error resolution progress, the provision of test reports to the departments and project management teams, as well as the preparation of go-live recommendations and final test reports. Consolidated reporting of the results of the departmental tests and BIAC regression tests to release management forms the basis for a go-live decision (BIAC, 2014j).

2.4.3 Test automation

Test automation is an important aspect in comprehensive test management. In addition to the manual tests, automated regression tests are also an absolute necessity and are regularly required. Test automation activities include the establishment of an automated regression test portfolio to safeguard production applications and support projects, regular mechanical regression tests (“smoke tests”) to control quality in test systems after project changes or changes to production applications, as well as regular application of the automated regression test portfolio during the project and release test phases (BIAC, 2014j).

2.4.4 The execution of Test cases

From the BIAC point of view, all required input and verification parameters are specified in the Test cases, especially in the software tool for testing TOSCA Commander™. The execution of Test cases can be initiated by any user, no specific technical know-how is thus required. Therefore, this provides the advantage that everyone, even without special of knowledge TOSCA Commander™ (business unit) can create, administrate and execute Test cases, but the Test cases and results can nevertheless be administrated in a central tool (and together with automated Test cases) (BIAC, 2014j).

The execution of a Test case can be started directly in TOSCA Commander™ or by directly starting TOSCA Executor. This flexibility is based on the use of test set data files, which contain the actual information for the execution of a Test case. During the execution via TOSCA Commander™ these test sets are created automatically. If required, they can be created manually by experienced users.

Furthermore, each Test case is defined in MS-Excel according to the defining rules of the Test Management Team. This requires an Excel spreadsheet, which organizes the test data in rows and columns. Test cases are represented by columns; particular data sets are represented by rows. The Excel-sheets contain an identification so that a selected policy with the additional check criteria will be used for a certain test execution. The policy numbers are referenced analogue to the regression Test cases with previously generated policies (BIAC, 2014j).

3 PROPOSALS AND CONTRIBUTION OF SUGGESTED SOLUTIONS

In order to determine the Test case prioritization several software tools are proposed. The conceptual design can be implemented using any proper computer programming language and data base management technology. As fuzzy logic is a convenient way to map an input space to output space, a fuzzy based technique for assigning priority of Test case has been chosen. With this regard, the solution in the programme MS Excel and MATLAB is presented in master's thesis.

Taking into consideration proposed programmes, Microsoft Excel is a software programme included in the Microsoft Office suite of applications, which allow users to organize, format and calculate data with formulas using a spreadsheet system (Janssen, 2014). Moreover, using Microsoft Excel it is possible to integrate fuzzy logic decision-making with the existing source of data.

As was already mentioned in previous chapter, MATLAB is a high – level programming language and technical computing environment developed by MathWorks. MATLAB allows analysing data, developing of algorithms, plotting of functions and data and creating models and applications. In comparison to Microsoft Excel, MATLAB using Fuzzy Logic Toolbox is able to design and simulate fuzzy logic systems based on fuzzy logic principle.

So as to process and compute analysed data, firstly input data needs to be defined. Therefore, in the following chapter different criteria (variables) considered for assigning weight will be elaborated based on priorities of test coordinators in BIAC. Basically, each input variable represents one decision criterion. In order to assign its weights (attributes), input variables are classified based on their importance in the test environment.

3.1 Input variables

Test cases often differ in execution in terms of specifying specific values or environment information. In addition to business-based processes, information, which is derived from the execution environment and should be used in Test Steps and Test Step Values (or in Modules) plays a major role. Nevertheless, following 8 different criteria (variables) and its weight are determined and depicted below based on priorities of test coordinators in BIAC.

Number of Test steps

Test steps are executable actions which are executed in the test object. The input variable Number of Test Steps has been divided into five intervals according to executed steps count as shown in Table 1.

Number of verification steps

A Test case describes an elementary and functional sequence, which is used for the verification of one or several properties underlying a specification. However, the value that is expected according to the Test case specification does not always correspond to the value provided by the test object. Therefore, it is verified whether a particular state is reached or not with comparison to specified value. So basically, number of verification steps involves eventual number of stages (steps) necessary to verify results with desired values. This criterion is described by 5 categories depicted in Table 1.

Module integration complexity

Integration metric defines aspect of complexity of component-based software. This criterion takes into account number of interfaces (interactions) with other components. Module integration complexity variable is sorted into 4 intervals illustrated in Table 1.

Business priority

In general, all Test cases are specified by customer's business department. Therefore, each Test case specification is exclusively business-referred and is gradually adapted to the test object (application under test). Business priority of Test cases determine in fact the order of the Test cases to be executed and how they are assigned to tests. In order to

evaluate business priority one of the decision criteria for prioritization of tests, it has to be divided into 4 categories described as ‘Low’, ‘Medium’, ‘High’, ‘Very high’ (see Table 1).

Test case background

Test case background is divided into 2 categories: Test cases based on specification and Test cases based on daily basis. Both criteria have equal value in terms of determination of priority for testing.

Type of Test case

Type of Test case is split into 2 fundamental categories: Automated Test case and Manual Test case. Manual Test cases are able to manage in TOSCA Commander™ software tool for testing. Even without specific knowledge of the tool it is feasible to create, administrate and execute Test cases. Furthermore, existing manual Test Steps can be converted into automated Test Steps. However, there are specified 2 criteria mentioned below which are associated only with automated Test cases.

Execution time

Criterion execution time is related only to automated cases. It represents time for execution of Test case measured in minutes. Execution time is divided into 5 intervals (see Table 1).

Preparation effort for execution

As already mentioned above, variable preparation effort for execution is valid to take into account only for automated Test cases. It is related to effort made to prepare all necessary steps before execution of Test case and it also includes time spent when execution is changed. This variable is sorted into 3 main groups shown in Table 1 measured in minutes.

3.2 Solution in MS Excel

Solution developed in Microsoft Excel specifies all fuzzy variables and computations of total relative weighting regarding evaluating variables related to Test case prioritization. The output of the proposed solution represents per cent and written assessment of Test case's priority. Nevertheless, firstly input matrix and transformation matrix need to be defined.

Description of transformation matrix

The description of transformation matrix consists of 8 variables which represent input data needed for appraisal of Automated Test case prioritization. This matrix remains the same for each particular Automated Test case (ATC) used for assessment (see Table 1). In order to evaluate priority of Manual Test case, 2 last variables (Execution time, Preparation effort for execution) are not taken into consideration. These variables are related only to Automated Test cases.

Table 1 Description of transformation matrix. Constructed by author.

	Type of test case	Number of test steps	Number of verification steps	Module integration complexity	Business priority	Test case background	Execution time (only for ATC)	Preparation effort for execution (only for ATC)
1	Manual	<1,10 >	<1,5>	<1,3>	4 (Low)	Based on daily business experience	<0,2>	<0,10>
2	Automated	< 11,30 >	<6,12>	<4,6>	3 (Medium)	Based on specification	(2,5>	(10,20>
3		<31,50 >	<13,20>	<7,10>	2 (High)		(5,8>	(20,60>
4		< 51,100 >	<21,40>	<11,20>	1 (Very High)		(8,15>	
5			<41,60>				(15,60>	

State matrix

The state matrix describes which attribute of input variable belongs to particular analysed Test case. For each Test case is built one state matrix which corresponds to real values of attributes. There are 2 possible values of attributes – Yes (Y) or No (N). For further calculation computed in MS Excel, values Yes or No are replaced by binary values 1 for Yes or 0 for No. The following Table 2 gives an instance of state matrix.

Table 2 State (Yes - No) matrix. Constructed by author.

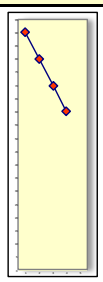
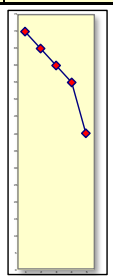
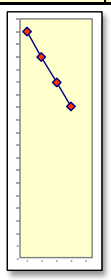
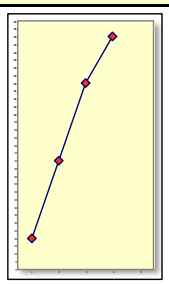
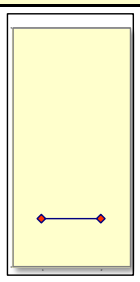
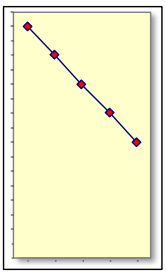
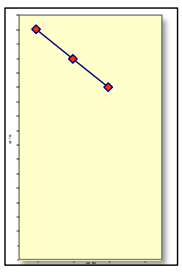
	Type of test case	Number of test steps	Number of verification steps	Module integration complexity	Business priority	Test case background	Execution time (only for ATC)	Preparation effort for execution (only for ATC)
1	N	N	N	Y	Y	Y	N	Y
2	Y	Y	Y	N	N	N	Y	N
3		N	N	N	N		N	N
4		N	N	N	N		N	
5			N				N	

Transformation matrix

Based on description of transformation matrix, the transformation matrix itself is evaluated. Each individual weight is determined according to importance of particular variables and its attributes which are set up by test experts in the company BIAC. Cells related to description of variables and its values need to correspond to each other.

In order to determine Test case priority, the function *SUMPRODUCT* is used. In general, the function *SUMPRODUCT* in MS Excel multiplies corresponding components in the transformation and state matrices and the sum of those products is returned. So calculation made of transformation matrix and state matrix gives an evaluation of Test case priority. The transformation matrix with illustrations of membership functions is shown in the Table 3.

Table 3 Transformation matrix. Constructed by author.

	Type of test case	Number of test steps	Number of verification steps	Module integration complexity	Business priority	Test case background	Execution time (only for ATC)	Preparation effort for execution (only for ATC)
1	No weights assigned	90	70	90	20	1	80	80
2		80	65	80	70	1	70	70
3		70	60	70	120		60	60
4		60	55	60	150		50	
5			40				40	
Max		**90**	**70**	**90**	**150**	**1**	**80**	**80**
Min		**60**	**40**	**60**	**20**	**1**	**40**	**60**
								

Retransformation matrix

The retransform matrix transforms the numerical values of Test case prioritization into linguistic values. The priority of Test cases is thus divided into 5 categories determined according to range of percentage relevant to final priority. Retransformation matrix as a result of solution in MS Excel is shown in Table 4. So when time is limited for execution of all hundreds of Test cases, determination of priority is essential in order to test maximum coverage of the most important Test cases.

Table 4 Retransformation matrix.
Constructed by author.fuzz

	Percentage	Priority
1	0%-20%	Very low
2	20%-40%	Low
3	40%-60%	Medium
4	60%-80%	High
5	80% -100%	Very High

3.3 Solution in MATLAB

As was already mentioned at the beginning of this chapter, fuzzy logic using Fuzzy Logic Toolbox, part of the MATLAB can be used for determination of Test case priority. For creating of fuzzy logic model is necessary to define input variables, their intervals, output variable and also membership functions. Nevertheless, the Fuzzy Logic Toolbox it is possible to trigger in a command window by the command *fuzzy*.

The fuzzy logic model is adjusted according to certain factors set up by test experts. To evaluate the attributes of variables, certain linguistic values are assigned such as *very low*, *low*, *medium*, *high* and *very high*. Once all the fuzzy inputs for each Test case are known and membership functions are set up, fuzzy rule base is constructed to arrive at the fuzzy output.

3.3.1 Input variables scheme

Fuzzy model consists of 3 inputs and 1 output. Each input is composed of set of variables as shown in Figure 19 and Figure 20. However, it is essential to take into consideration the fact, that for determination of priority of automated Test cases, 2 additional variables as ‘Execution time’ and ‘Preparation for execution’ time are needed for getting the proper results (see Figure 19).

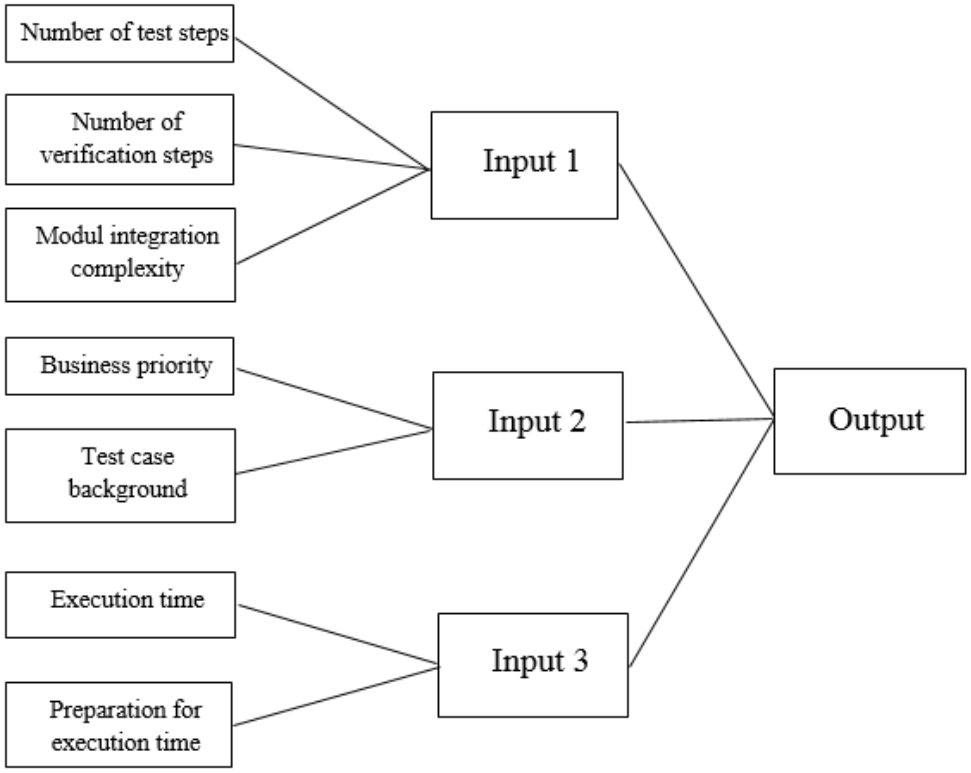


Figure 19 Input variable scheme for Automated Test cases. Constructed by author.

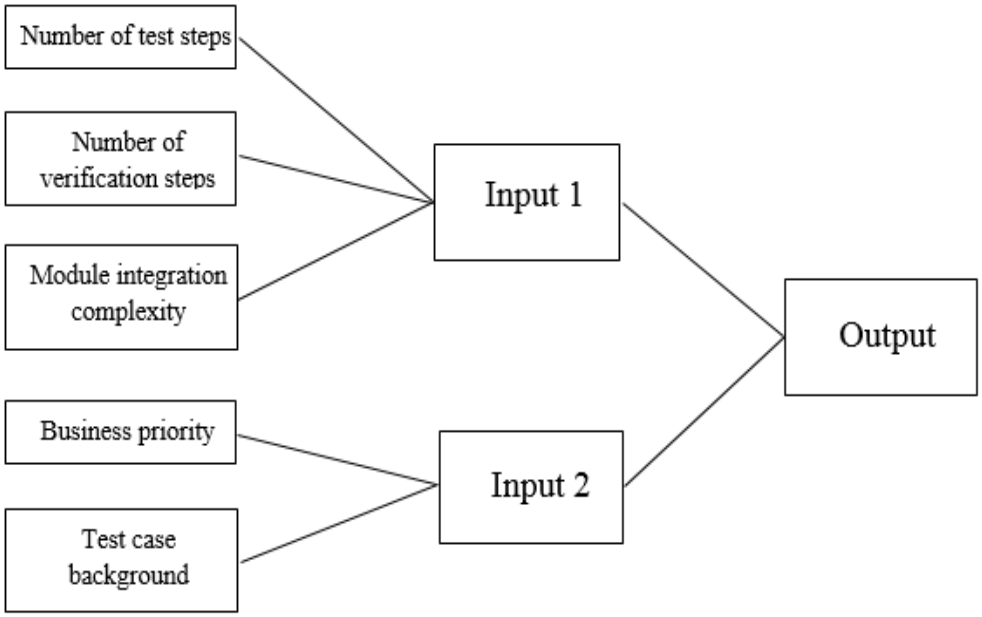


Figure 20 Input variable scheme for Manual Test cases. Constructed by author.

Fuzzy logic process:

- Input data are defined as a result of the number of years of experience gained by test experts in the company
- Input data are fuzzified using membership functions
- Fuzzy rule base is applied on fuzzy input to evaluate the fuzzy output
- Fuzzy output is defuzzified in order to get final value

3.3.2 FIS Editor

Fuzzy Inference System (FIS) is the process of formulating the mapping from a given input to an output using fuzzy logic. This will use Mamdani's fuzzy inference method which is most commonly seen fuzzy methodology. FIS Editor is shown in Figure 21 where variables of Input 1 are taken into consideration.

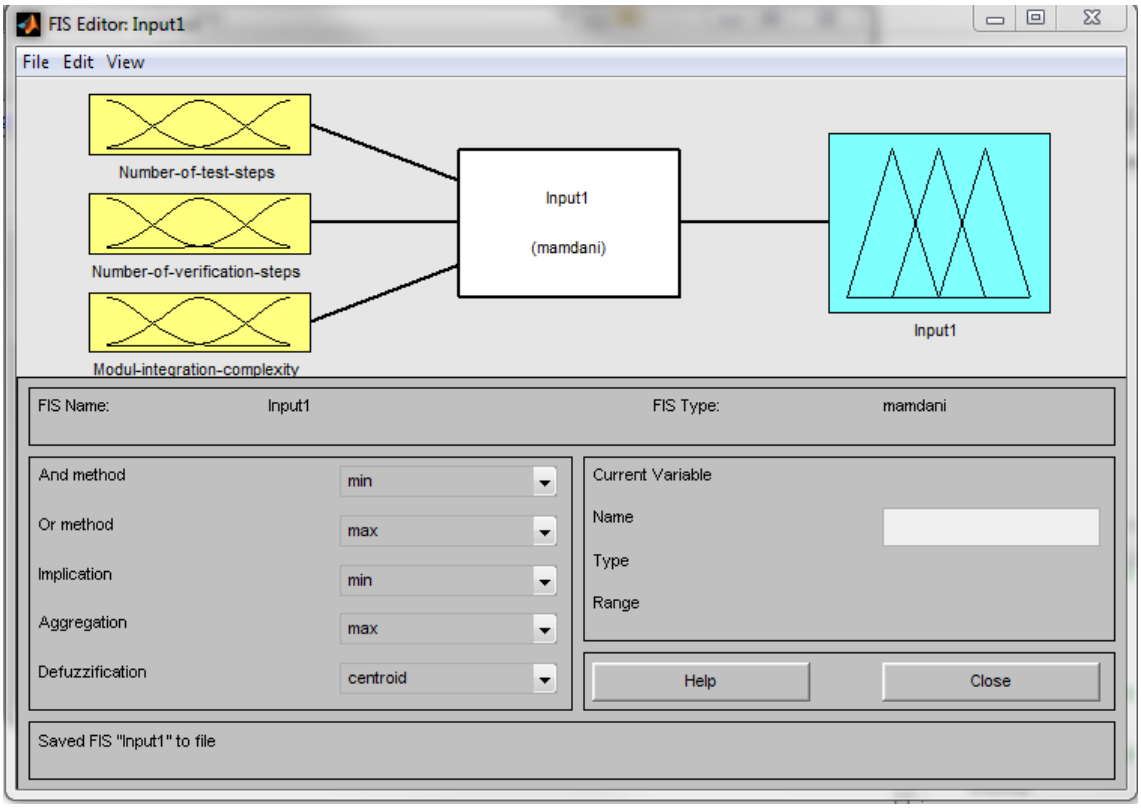


Figure 21 FIS Editor – Input1. Constructed by author.

3.3.3 MF Editor

With the Membership Function Editor it is possible to display and revise all of the membership functions associated with all of the input and output variables for the entire fuzzy inference system. Hence, the tool can be used to define the number, type, range and parameters of membership functions in case of each variable. In order to open MF Editor it is mandatory to double-click on the input variable. MF Editor for the variable *Number of Test cases* and its membership functions is illustrated in the Figure 22.

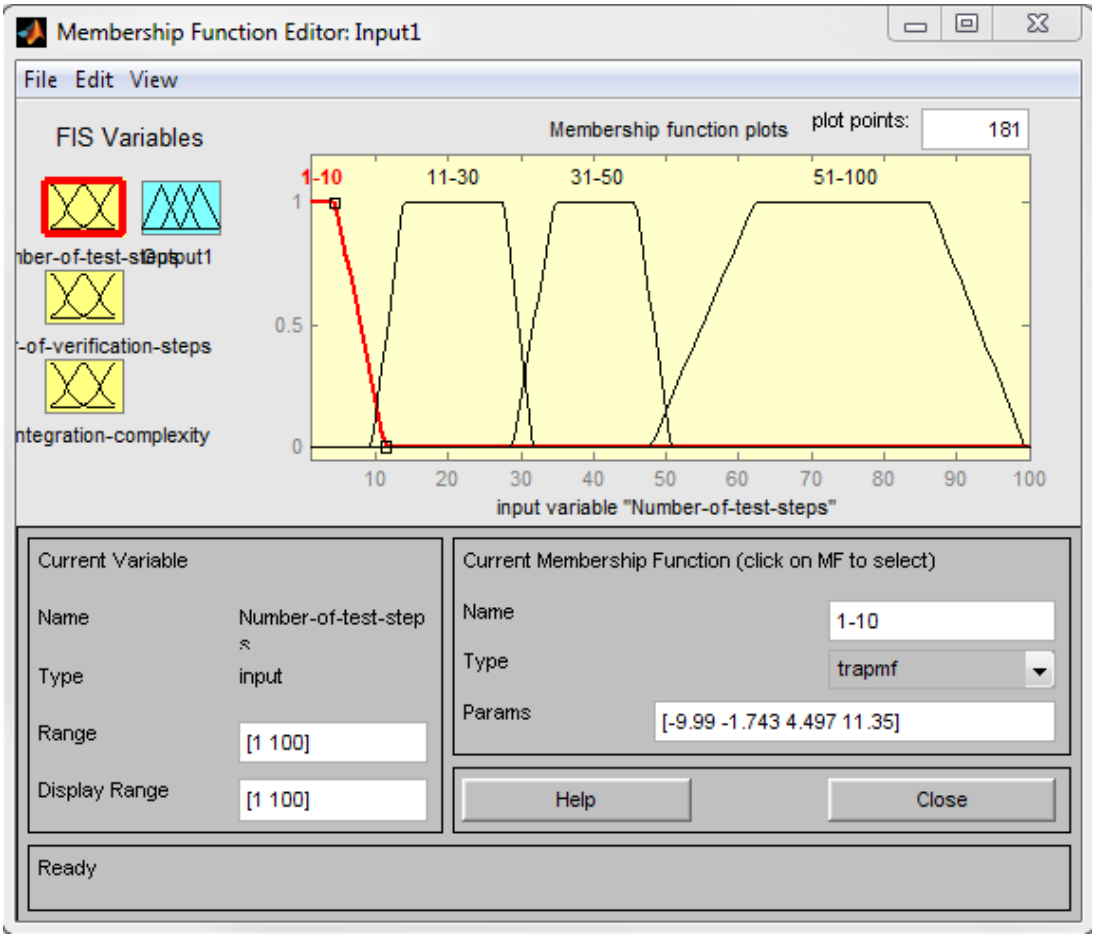


Figure 22 MF Editor – Input1. Constructed by author.

3.3.4 Rule Editor

The fuzzy logic based model presented herein specifies each parameter of Test case using membership values and uses fuzzy rule base for calculating Test case priority. Based on the descriptions of the input and output variables defined with the FIS Editor,

the Rule Editor allows to construct the rule statements. Total of 81 different rules have been formulated in order to analyse the results for Input1, as shown in Figure 23.

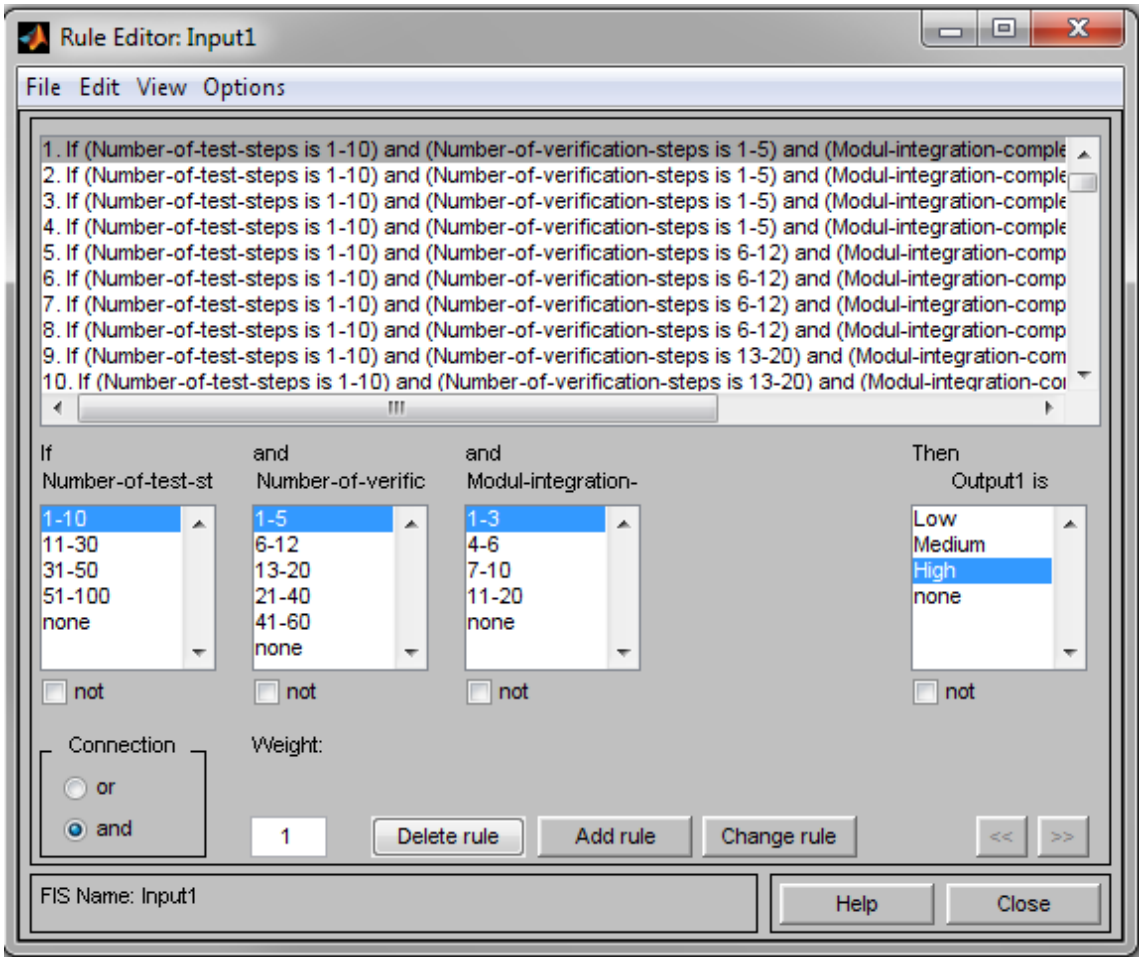


Figure 23 Rule Editor – Input 1. Constructed by author.

3.3.5 Rule Viewer

Further graphical user interface (GUI) tool in Fuzzy Logic Toolbox is Rule Viewer which enables to view the fuzzy inference diagram. Moreover, it is possible to see which rules are active or how individual membership function shapes affect the results. Rule Viewer displaying rules for Input 1 is shown in Figure 24.

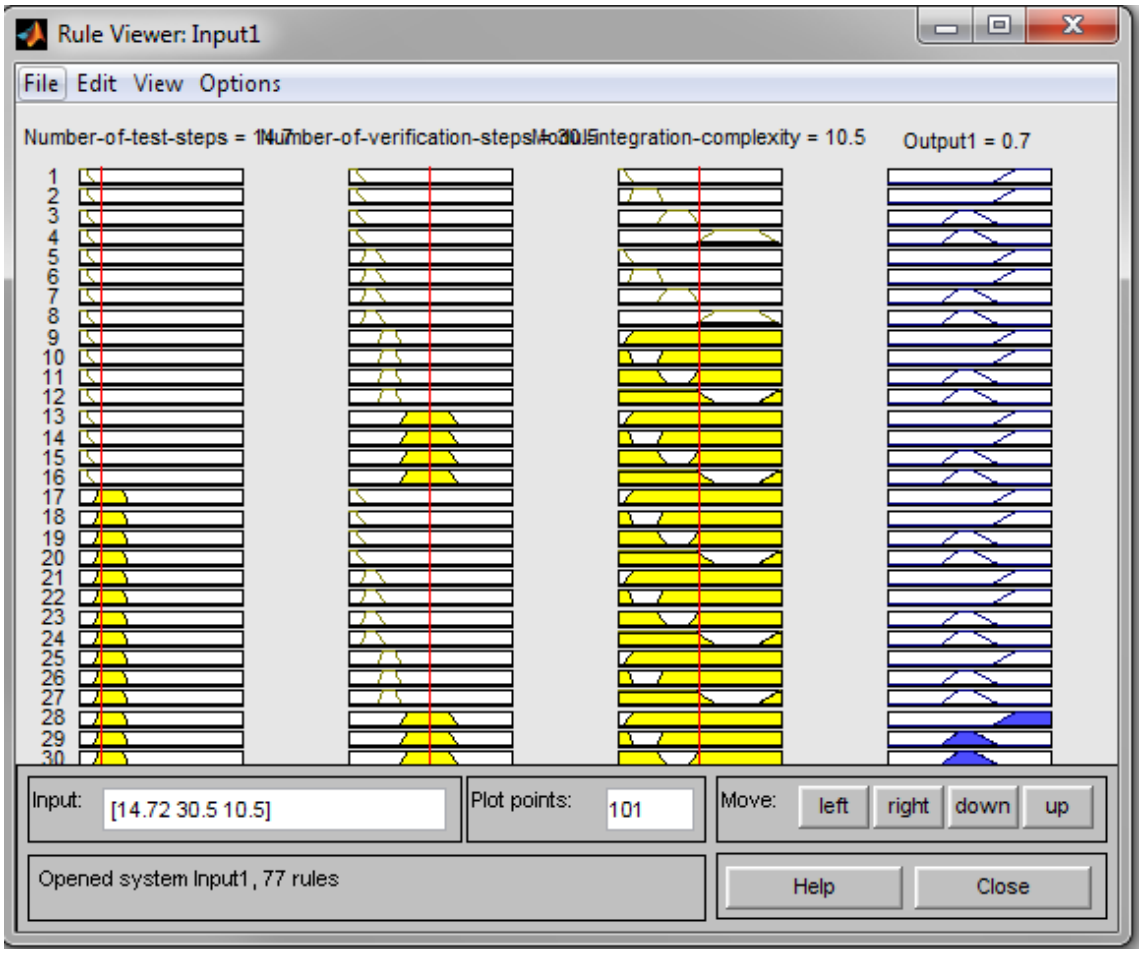


Figure 24 Rule Viewer - Input 1. Constructed by author.

3.3.6 Surface Viewer

The Surface Viewer can generate a three-dimensional output surface where any two of the inputs vary. The Figure 25 represents the surface view of the variable *Number of verification steps* and the variable *Number of test steps* and their relation to the evaluation.

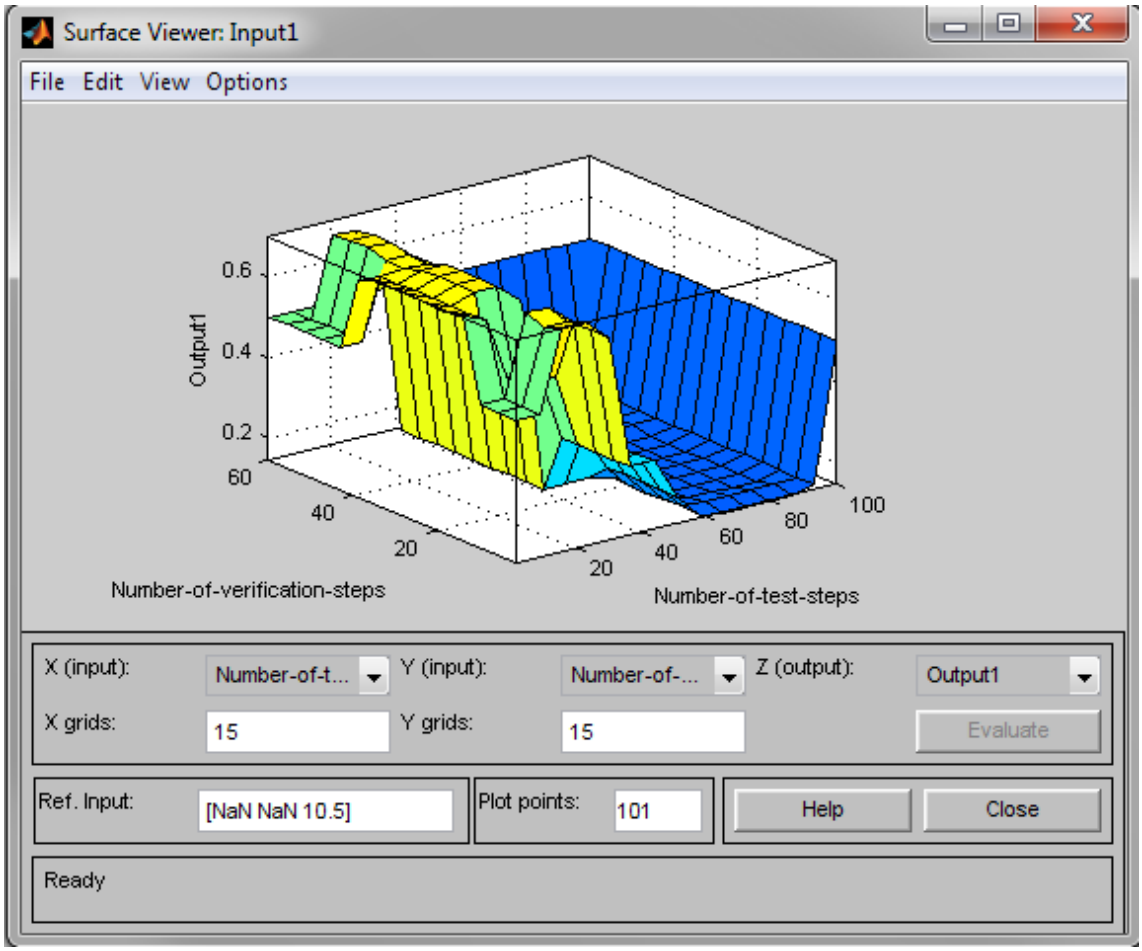


Figure 25 Surface Viewer – Input 1. Constructed by author.

3.3.7 Evaluation of Test case priority in both programs

So, in order to get evaluation of Test case priority in Fuzzy Logic Toolbox in MATLAB program, there are 2 possibilities how to come to the feasible solution. One of the possibility consists of entering input values of variables manually directly in Command Window. With that regard, firstly it is necessary to determine whether calculation is related to Manual Test case (see Figure 20) or Automated Test case (see Figure 19).

Moreover, for the purpose of evaluating priority of Automated Test cases, 2 more criteria as Execution time and Preparation for execution time are taken into account as illustrated in Figure 27. Source code used for both types is depicted in Appendix 1.

```
Enter type of test case in the form 0-Manual or 1-Automated:0
Enter input data in the form
[Number of test steps;Number of verification steps;Modul integration complexity ]:[20;10;2]
Enter input data in the form  [Business priority; Test case background ]:[4;1]

Input_man =

    0.6998    0.1485


Priority =

    0.3972


ans =

Low
```

Figure 26 Determination of priority for Manual Test Cases derived from Command Window.
Constructed by author.

```
Enter type of test case in the form 0-Manual or 1-Automated:1
Enter input data in the form
[Number of test steps; Number of verification steps; Modul integration complexity ]:[20;10;2]
Enter input data in the form  [Business priority; Test case background ]:[4;1]
Enter input data in the form  [Execution time; Preparation for execution time ]:[3;5]

Input_aut =

    0.6998    0.1485    0.8549


Priority =

    0.3972


ans =

Low
```

Figure 27 Determination of priority for Automated Test Cases derived from Command Window.
Constructed by author.

Second option how to get evaluation of Test case priority deals with loading input data directly from data source saved in MS Excel. It enables to load automatically as many Test cases (rows) as is needed for evaluation. In order to start with execution of input data in Command Window, firstly type of Test cases (0-Manual, 1-Automated) to be filled in and secondly name of Excel Sheet needs to be defined as shown in Figure 28.

```
Enter type of test case in the form 0-Manual or 1-Automated:1
Enter file name in the form [name.xls]written with an apostrophe :'DP.xls'

Input_aut =

    0.6998    0.1485    0.8549


Priority =

    0.3972


ans =

Low


Input_aut =

    0.2952    0.8551    0.5029


Priority =

    0.8002
```

Figure 28 Determination of priority for Test Case inputs derived from MS Excel. Constructed by author.

Nevertheless, input data saved in MS Excel (see Table 5) are sorted into columns in the same order as they are entered manually in Command Window. Final values considering priority of Automated Test cases are written automatically into last column named Output after calculation is made in Fuzzy Logic Toolbox. So, priority of Test cases is written both in Excel Sheet and in Command Window of Fuzzy Logic Toolbox.

Table 5 Input data for evaluation of Automated Test cases for computation made in MATLAB.
Constructed by author.

Input 1			Input 2		Input 3		Output
Number of test steps	Number of verification steps	Modul integration complexity	Business priority	Test case background	Execution time	Preparation for execution time	Priority [%]
<1;100>	<1;60>	<1;20>	<1;4>	<1;2>	<0;60>	<0;60>	
40	10	5	4	1	3	5	39,75%
40	10	2	1	2	6	5	80,11%
15	15	15	3	2	3	15	58,87%
20	10	5	2	1	3	15	79,99%
5	2	2	1	1	1	15	95,51%

In addition to solution derived from MATLAB, the computational process of the proposed decision support system is created in MS-Excel as well (see Table 6). The output of that solution represents Test case's priority expressed as a percentage and as linguistic variables according to final figures as shown in Table 7.

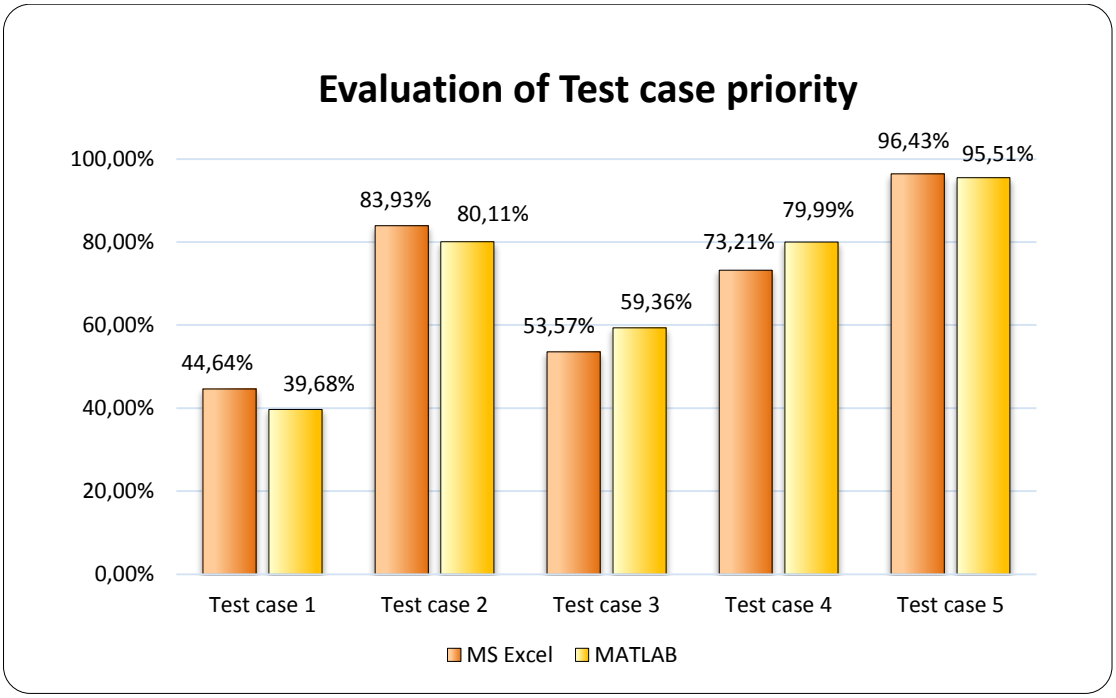
Table 6 Input data for evaluation of Automated Test cases for computation made in MS Excel.
Constructed by author.

	Type of test case	Number of test steps	Number of verification steps	Module integration complexity	Business priority	Test case background	Execution time (only for ATC) [minutes]	Preparation effort for execution (only for ATC) [minutes]
1	Automated	<31,50 >	<6,12>	<4,6>	4 (Low)	Based on daily business experience	(2,5>	<0,10>
2	Automated	<31,50 >	<6,12>	<1,3>	1 (Very High)	Based on specification	(5,8>	<0,10>
3	Automated	< 11,30 >	<13,20>	<4,6>	3 (Medium)	Based on specification	(2,5>	(10,20>
4	Automated	< 11,30 >	<6,12>	<4,6>	2 (High)	Based on daily business experience	(2,5>	(10,20>
5	Automated	<1,10 >	<1,5 >	<1,3>	1 (Very High)	Based on daily business experience	<0,2>	(10,20>

Table 7 Priority of Automated Test cases computed in MS Excel
Constructed by author.

Test case	Percentage	Priority
1	37,50%	Low
2	83,93%	Very high
3	53,57%	Medium
4	73,21%	High
5	96,43%	Very high

Comparison of results taken from MS Excel (see Table 7) and MATLAB (see Table5) gives evaluation of analysed Automated Test cases. Basically, final output values conducted in both programs do not differ so much from each other. For better comparison, both results are illustrated in the Graph 1.



Graph 1 Comparison of results for Automated Test cases computed in MS Excel and MATLAB. Constructed by author.

In order to proceed with solution for evaluation of Manual Test cases, the same approach is used analogously. The output for determination of priority derived from MATLAB is shown in the last column of Table 8. Apart from that, input data for computation made in Excel are shown in the Table 9. Final evaluated figures computed in MS Excel are expressed in percentage in the Table 10.

Table 8 Input data for evaluation of Manual Test cases computed in MATLAB. Constructed by author.

Input 1			Input 2		Output
Number of test steps	Number of verification steps	Modul integration complexity	Business priority	Test case background	Priority [%]
<1;100>	<1;60>	<1;20>	<1;4>	<1;2>	
40	15	8	4	1	14,50%
20	3	5	1	2	94,78%
40	15	8	3	2	39,72%
5	10	2	2	1	79,67%
70	15	2	1	1	79,67%

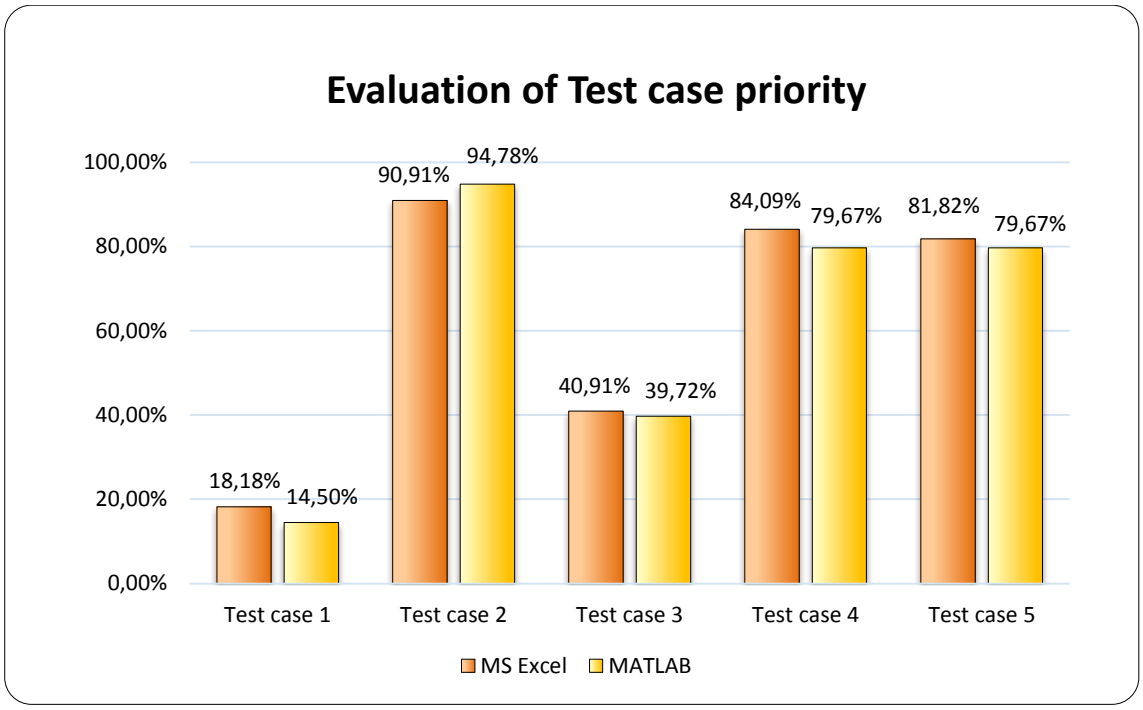
Table 9 Input data for evaluation of Manual Test cases for computation made in MS Excel.
Constructed by author.

Test case	Type of test case	Number of test steps	Number of verification steps	Module integration complexity	Business priority	Test case background
1	Manual	<31,50 >	<13,20>	<7,10>	4 (Low)	Based on daily business experience
2	Manual	< 11,30 >	<1,5>	<4,6>	1 (Very High)	Based on specification
3	Manual	<31,50 >	<13,20>	<7,10>	3 (Medium)	Based on specification
4	Manual	<1,10 >	<6,12>	<1,3>	2 (High)	Based on daily business experience
5	Manual	< 51,100 >	<13,20>	<1,3>	1 (Very High)	Based on daily business experience

Table 10 Priority of Manual Test cases computed in MS Excel
Constructed by author.

Test case	Percentage	Priority
1	18,18%	Very low
2	90,91%	Very high
3	40,91%	Medium
4	84,09%	Very high
5	81,82%	Very high

Last but not least, the output of Test case priority calculated in both programs is depicted in the Graph 2. Input data compared in this graph are resulted from Table 8 and Table 10. Nevertheless, the eventual amount of Test cases which is considered to be evaluated depends mainly on type of executed tests.



Graph 2 Comparison of results for Manual Test cases computed in MS Excel and MATLAB.
Constructed by author.

CONCLUSIONS

The aim of master's thesis was to determine prioritization of Test case using Fuzzy logic based model. So as to get evaluation of Test cases in order of priority, Fuzzy based model was selected because of better decisions made by it in comparison to the additional normal expert systems. Moreover, Fuzzy logic allows the integration of numerical data and expert knowledge and can be a powerful tool when tackling significant problems in software engineering especially in testing environment such as determination of Test case priority.

In fact, the output of the proposed model is resulted from determination of Test case priority order in the program MS Excel and Fuzzy Logic Toolbox in MATLAB. In order to fulfil the aim, firstly, it was essential to determine input variables along with parameters set to each Test case and assigning its particular weights based on testing environment in BIAC's company.

From the overall point of view, prioritization is used when the time for the testing is limited. In general, determination of Test case priority can improve the test effectiveness and the rate of fault detection during the tests phase. Therefore, prioritization of the Test cases was widely proposed in the BIAC's company. The results obtained due to this process are very encouraging for better decision-making in whole Test Management.

However, one of the greatest difficulties in using the model is determination of proper fuzzy rules which depends on current priority of tests which are executed in that moment. These fuzzy rules express the information for interpretation of the nature of Test cases in testing department. The interpretation of each fuzzy rule is made by analysing its basis and its output finally provides a determination of Test case priority order. Besides that, additional measures, improvements and fine-tuning will be conducted in Test department in the foreseeable future.

With regard to the aim of master's thesis, whole concept was divided into three main parts. The first part was dedicated to literature review which consisted of theoretical knowledge concerning testing phase in software engineering. The second part represents analyses focusing on profile of BIAC's company, SAP modules and process of software testing in BIAC. Finally, the last part is dedicated to evaluation of Test case priority in the program MS Excel and Fuzzy Logic Toolbox in MATLAB. Furthermore, results derived from both solutions are depicted and compared in the graphs.

REFERENCES

BHASIN H., GUPTA S. and KATHURIA M., 2013. Regression Testing Using Fuzzy Logic. *International Journal of Computer Science and Information Technologies*. [online], 4(2), pp. 378-380. Available via: International Journal of Computer Science and Information Technologies [Accessed 11 November 2013].

BIAC, 2014a. *About us* [online]. Business Insurance Application Consulting GmbH. Available at: http://www.biac.at/en/about-us [Accessed 12 April 2014].

BIAC, 2014b. *Insurance Solutions* [online]. Business Insurance Application Consulting GmbH. Available at: http://www.biac.at/en/insurance-solutions [Accessed 12 April 2014].

BIAC, 2014c. *Project Management* [online]. Business Insurance Application Consulting GmbH. Available at: http://www.biac.at/en/project-management [Accessed 12 April 2014].

BIAC, 2014d. *Test Management* [online]. Business Insurance Application Consulting GmbH. Available at: http://www.biac.at/en/test-management [Accessed 12 April 2014].

BIAC, 2014e. *Support Services* [online]. Business Insurance Application Consulting GmbH. Available at: http://www.biac.at/en/support-services [Accessed 12 April 2014].

BIAC, 2014f. *Business Consulting* [online]. Business Insurance Application Consulting GmbH. Available at: http://www.biac.at/en/business-consulting [Accessed 12 April 2014].

BIAC, 2014g. *Solution Consulting* [online]. Business Insurance Application Consulting GmbH. Available at: http://www.biac.at/en/solution-consulting [Accessed 12 April 2014].

BIAC, 2014h. *Architecture and Quality Management* [online]. Business Insurance Application Consulting GmbH. Available at: http://www.biac.at/en/architecture-and-quality-management [Accessed 12 April 2014].

BIAC, 2014i. *SAP Insurance Modules* [online]. Business Insurance Application Consulting GmbH. Available at: http://www.biac.at/en/insurance-solutions#FS-PM [Accessed 28 April 2014].

BIAC, 2014j. *BIAC Service Catalogue*. Intranet, BIAC, Vienna.

CHAUDHARY N., SANGWAN O.M., and SINGH Y., 2012. Testcase Prioritization Using Fuzzy Logic for GUI based Software. *International Journal of Advanced Computer Science and Applications.* [online], 3(12), pp.222-227. Available via: International Journal of Advanced Computer Science and Applications [Accessed 12 November 2013].

DOSTÁL P., 2011. *Advanced decision making in business and public services.* 1st ed. Brno: Akademické nakladatelství CERM. 167 p. ISBN 978-80-7204-747-5

EMERALD INSIGHT, 2014. *Architecture of fuzzy decision making system* [online]. Emerald Group Publishing Limited. Available at: http://www.emeraldinsight.com/content_images/fig/1820240402017.png [Accessed 08 April 2014].

ESRU, 2014. *Operations on Fuzzy Sets* [online]. University of Strathclyde. Available at: http://www.esru.strath.ac.uk/Reference/concepts/fuzzy/operations.htm [Accessed 09 April 2014].

EVERETT G., MCLEOD R., 2009. *Software Testing: Testing Across the Entire Software Development Life Cycle*. New Jersey: John Wiley & Sons. 345 p. ISBN 0470146346

JANSSEN C., 2014. *Microsoft Excel* [online]. Techopedia. Available at: http://www.techopedia.com/definition/5430/microsoft-excel [Accessed 04 May 2014].

KELKAR S.A., 2009. *Software Project Management: A concise study.* 2nd ed. New Delhi: PHI Learning Pvt. 230 p. ISBN 8120336720

KHAN E., 2011. Different Software Testing Levels for Detecting Errors. *International Journal of Software Engineering.* [online], 2(4) pp. 70-80. Available at: http://cscjournals.org/csc/manuscript/Journals/IJSE/volume2/Issue4/IJSE-60.pdf [Accessed 8 April 2014].

KLINGENBERG B., 2014. *Beginners Tutorial* [online]. Calvin College Engineering. Available at: http://www.calvin.edu/~pribeiro/othrlnks/Fuzzy/tutorial1.htm [Accessed 13 April 2014].

LEMOS O.A.L., FERRARI F.C., ELER M.M., MALDONADO J.C. and MASIERO P.C., 2012. Evaluation studies of software testing research in Brazil and in the world: A survey of two premier software engineering conferences. *The Journal of Systems and Software.* [online], 86(2013) pp. 951-969. Available via: The Journal of Systems and Software. [Accessed 5 April 2014].

LEWIS J., 2008. *SDLC 100 Success Secrets - Software Development Life Cycle (SDLC) 100 Most Asked Questions, SDLC Methodologies, Tools, Process and Business Models*. Newstead: Emereo Publishing. 184 p. ISBN 1921523158

MATHWORKS, 2014a. *Matlab: The Language of Technical Computing* [online]. The MathWorks, Inc. Available at: http://www.mathworks.com/products/matlab/ [Accessed 12 April 2014].

MATHWORKS, 2014b. *Fuzzy Logic Toolbox: Design and simulate fuzzy logic systems* [online]. The MathWorks, Inc. Available at: http://www.mathworks.com/products/fuzzy-logic/index.html [Accessed 12 April 2014].

MATHWORKS, 2014c. *Fuzzy Logic Toolbox: Working with the Fuzzy Logic Toolbox* [online]. The MathWorks, Inc. Available at: http://www.mathworks.com/products/fuzzy-logic/description2.html [Accessed 12 April 2014].

MATHWORKS, 2014d. *Fuzzy Logic Toolbox: Key Features* [online]. The MathWorks, Inc. Available at: http://www.mathworks.com/products/fuzzy-logic/description1.html [Accessed 12 April 2014].

MATHWORKS, 2014e. *Interpreting the Fuzzy Inference Diagram* [online]. The MathWorks, Inc. Available at: http://www.mathworks.com/cmsimages/40305_wl_fl_mainimage_wl_3248.gif [Accessed 12 April 2014].

MATHWORKS, 2014f. *Fuzzy Logic Toolbox: Building a Fuzzy Inference System* [online]. The MathWorks, Inc. Available at: http://www.mathworks.com/products/fuzzy-logic/description3.html [Accessed 12 April 2014].

MATHWORKS, 2014g. *Documentation Center: Build Mamdani Systems (GUI)* [online]. The MathWorks, Inc. Available at: http://www.mathworks.com/help/fuzzy/building-systems-with-fuzzy-logic-toolbox-software.html [Accessed 12 April 2014].

OLADIMEJI P., 2007. *Levels of Testing*. [online], Swansea University. Available at: http://www.cs.swan.ac.uk/~csmarkus/CS339/dissertations/OladimejiP.pdf [Accessed 12 April 2014].

OMRAN H., 2010. *Fuzzinator: A Fuzzy Logic Controller* [online]. Code Project. Available at: http://www.codeproject.com/Articles/33214/Fuzzinator-A-Fuzzy-Logic-Controller [Accessed 09 April 2014].

RODRIGEZ J.P., VIZCAINO A., PIATTINI M. and BEECHAM S., 2012. Tools used in Global Software Engineering: A systematic mapping review. *Information and Software Technology*. [online], pp. 663-685. Available via: Information and Software Technology [Accessed 2 April 2014].

SCHMIDT R., 2009. *Software Engineering: Architecture-driven Software Development*. San Francisco: Morgan Kaufmann. 376 p. ISBN 0124077684

SRIVASTAVA P.R., KUMAR S., SINGH A.P. and RAGHURAMA G., 2010. Software Testing Effort: An Assessment Through Fuzzy Criteria Approach. *Journal of Uncertain Systems.* [online], 5(3), pp. 183-201. Available via: Journal of Uncertain Systems [Accessed 1 April 2014].

SYMBIOSYS TECHNOLOGIES, 2013. *Beginners Guide to Software Testing* [online]. Symbiosys Technologies. Available at: http://api.ning.com/files/5jbHWSIOQjGopyhKTRk3iXyBYhqM7zg7HF-ZRD9JLI8kUS2pIv1eJ9ieZ1CrfAzzlFXAlE0DpEACMZ6nLf5YF6wU536XIsWQ/GuideSoftwareTesting.pdf [Accessed 12 April 2014].

TAGHAVIFAR H. and MARDANI A., 2013. Fuzzy logic system based prediction effort: A case study on the effects of tire parameters on contact area and contact pressure. *Applied Soft Computing*. [online], 14(2014) pp. 390-396. Available at: ScienceDirect [Accessed 8 April 2014].

TUTORIALSPOINT, 2014. *Software Testing Tutorial* [online]. Tutorialspoint. Available at: http://www.tutorialspoint.com/software_testing/software_testing_pdf_version.htm [Accessed 08 April 2014].

ZADEH, L.A., 1965. Fuzzy sets. *Information and Control.* [online], 8(3), pp. 338-353. Available via: ScienceDirect [Accessed 1 April 2014].

LIST OF FIGURES

LIST OF TABLES

LIST OF GRAPHS

LIST OF APPENDICES

Appendix 1 - M-file derived from MATLAB

Source code for evaluation of Test case priority used for user who enters the input data manually.

```
Type_of_testcase= input ('Enter type of test case in the form 0-Manual
or 1-Automated:');

switch Type_of_testcase
    case 0
       a=readfis ('Input1.fis');
       Input1=input ('Enter input data in the form\n[Number of test
steps;Number of verification steps;Modul integration complexity ]:');
       evalI1=evalfis (Input1, a);

       b=readfis ('Input2.fis');
       Input2=input ('Enter input data in the form  [Business
priority; Test case background ]:');
       evalI2=evalfis (Input2, b);

       d=readfis ('Input_man.fis');
       Input_man(1) =evalI1;
       Input_man(2) =evalI2;
       Priority=evalfis (Input_man, d);
       Input_man
       Priority

        if Priority <=0.30 'Very low'
            elseif Priority <0.40 'Low'
                elseif Priority <0.60 'Medium'
                    elseif Priority <0.80 'High'
                        elseif Priority <1 'Very high'
        end

       %fuzzy (d)
       %mfedit(d)
       %ruleedit(d)
       %surfview(d)
       %ruleview(d)

    case 1
       a=readfis ('Input1.fis');
       Input1=input ('Enter input data in the form \n[Number of test
steps; Number of verification steps; Modul integration complexity
]:');
       evalI1=evalfis (Input1, a);

       b=readfis ('Input2.fis');
       Input2=input ('Enter input data in the form  [Business
priority; Test case background ]:');
       evalI2=evalfis (Input2, b);

       c=readfis ('Input3.fis');
       Input3=input ('Enter input data in the form  [Execution time;
Preparation for execution time ]:');
```

```
    evalI3=evalfis (Input3, c);

    d=readfis ('Input_aut.fis');
    Input_aut(1) =evalI1;
    Input_aut(2) =evalI2;
    Input_aut(3) =evalI3;
    Priority=evalfis (Input_aut, d);
    Input_aut
    Priority

     if Priority <=0.30 'Very low'
         elseif Priority <0.40 'Low'
             elseif Priority <0.60 'Medium'
                 elseif Priority <0.80 'High'
                     elseif Priority <1 'Very high'
    end

    %fuzzy (d)
    %mfedit(d)
    %ruleedit(d)
    %surfview(d)
    %ruleview(d)

    otherwise
      disp('Invalid value of type of test case');
end
```

Appendix 2 - M-file derived from MATLAB

Source code for evaluation of Test case priority used for automatic loading the input data from Excel Sheet to Fuzzy Logic Toolbox.

```
Type_of_testcase= input ('Enter type of test case in the form 0-Manual
or 1-Automated:');

switch Type_of_testcase
    case 0
       a=readfis ('Input1.fis');
       b=readfis ('Input2.fis');
       d=readfis ('Input_man.fis');

       File_name = input ('Enter file name in the form name.xls
written with an apostrophe :');
       Input1matrix = xlsread (File_name,
'Manual_testcases','A4:C99');
       Input2matrix = xlsread (File_name,
'Manual_testcases','D4:E99');
       Input1matrix_size = size(Input1matrix);

       for row=1:1:Input1matrix_size(1)
        Input1 = Input1matrix(row, :);
        Input2 = Input2matrix(row, :);

        evalI1=evalfis (Input1, a);
        evalI2=evalfis (Input2, b);

        Input_man(1) =evalI1;
        Input_man(2) =evalI2;
        Priority=evalfis (Input_man, d);
        Input_man
        Priority
        xlswrite(File_name, Priority, 'Manual_testcases', ['F'
num2str(3+row)]);

        if Priority <=0.20 'Very low'
            elseif Priority <0.40 'Low'
                elseif Priority <0.60 'Medium'
                    elseif Priority <0.80 'High'
                        elseif Priority <1 'Very high'
        end

        %fuzzy (d)
        %mfedit(d)
        %ruleedit(d)
        %surfview(d)
        %ruleview(d)
       end

    case 1
       a=readfis ('Input1.fis');
       b=readfis ('Input2.fis');
```

```
        c=readfis ('Input3.fis');
        d=readfis ('Input_aut.fis');

        File_name = input ('Enter file name in the form
[name.xls]written with an apostrophe :');
        Input1matrix = xlsread (File_name,
'Automated_testcases','A4:C99');
        Input2matrix = xlsread (File_name,
'Automated_testcases','D4:E99');
        Input3matrix = xlsread (File_name,
'Automated_testcases','F4:G99');
        Input1matrix_size = size(Input1matrix);

        for row=1:1:Input1matrix_size(1)
         Input1 = Input1matrix(row, :);
         Input2 = Input2matrix(row, :);
         Input3 = Input3matrix(row, :);

         evalI1=evalfis (Input1, a);
         evalI2=evalfis (Input2, b);
         evalI3=evalfis (Input3, c);

         Input_aut(1) =evalI1;
         Input_aut(2) =evalI2;
         Input_aut(3) =evalI3;
         Priority = evalfis (Input_aut, d);
         Input_aut
         Priority
         xlswrite(File_name, Priority, 'Automated_testcases', ['H'
num2str(3+row)]);

         if Priority <=0.20 'Very low'
             elseif Priority <0.40 'Low'
                 elseif Priority <0.60 'Medium'
                     elseif Priority <0.80 'High'
                         elseif Priority <1 'Very high'
         end

         %fuzzy (d)
         %mfedit(d)
         %ruleedit(d)
         %surfview(d)
         %ruleview(d)
        end
end
```