Software Engineering Using Artificial Intelligence Techniques: Current State and Open Problems

Article · March 2012

CITATIONS
28

READS
14,629

3 authors:

Hany Ammar
West Virginia University
159 PUBLICATIONS **3,339** CITATIONS
SEE PROFILE

Walid Abdelmoez
Arab Academy for Science, Technology & Maritime Transport
49 PUBLICATIONS **681** CITATIONS
SEE PROFILE

Mohamed Salah Hamdi
Ahmed Bin Mohammed Military College, Qatar
63 PUBLICATIONS **567** CITATIONS
SEE PROFILE

Software Engineering Using Artificial Intelligence Techniques: Current State and Open Problems

Hany H Ammar[1,2], Walid Abdelmoez[3], and Mohamed Salah Hamdi[4],
[1] The Lane Department of Computer Science and Electrical Engineering, West Virginia University, USA,
[2]Computer Science Department, Faculty of Computers and Information, Cairo University, Egypt
[3]Arab Academy for Science, Technology and Maritime Transport, Egypt
[4]Ahmed Bin Mohamed Military College, Qatar

Abstract: This paper surveys the application of artificial intelligence approaches to the software engineering processes. These approaches can have a major impact on reducing the time to market and improving the quality of software systems in general. Existing survey papers are driven by the AI techniques used, or are focused on specific software engineering processes. This paper relates AI techniques to software engineering processes specified by the IEEE 12207 standard of software engineering. The paper is driven by the activities and tasks specified in the standard for each software engineering process. The paper brings the state of the art of AI techniques closer to the software engineer, and highlights the open research problems for the research community.

Keywords: Automated Software Engineering, Artificial Intelligence Techniques.

1. Introduction

The software intensive systems we develop these days are becoming much more complex in terms of the number of functional and nonfunctional requirements they need to support. The impact of low quality can also have a catastrophic impact on the mission of these systems in many critical applications. Moreover, the cost of software development dominates the total cost of such systems.

Research in applying artificial intelligence techniques to software Engineering have grown tremendously in the last two decades producing a large number of projects and publications. A number of conferences and journals are dedicated to publish the research in this field. The AI techniques are proposed in order to reduce the time to market and enhance the quality of software systems. Yet many of these AI techniques remain largely used by the research community and with little impact on the processes and tools used by the practicing software engineer.

The recent survey papers published in this field are mainly targeted to the research community. They are driven by the specific AI techniques used rather than the software engineering activities supported. They are also focused on a specific software engineering process such as software design [28]

This survey paper attempts to close the gap between the research and practice of applying AI techniques to the software engineering processes. It also highlights open practical problems to the research community in applying such techniques by surveying the recently proposed work in this area.

We use the terminology and the processes defined by the IEEE 12207 standard of software engineering. We then map the current state art of AI art techniques proposed in the literature to specific tasks and activities of some of the software processes define by the 12207 standard. These AI techniques attempt to automate or semi-automate these tasks and produce optimal or semi-optimal solutions in much less time. .

The paper is organized as follows. In section 2, we give an overview of the IEEE 12207 standard of software engineering, and describe the most important AI techniques. We survey the current AI techniques proposed for the primary processes of development in sections 3. We highlight the open problems in section 4.

2. Background

This section briefly introduces the primary processes of the IEEE 12207 standard for software life cycle processes. This standard is well documented and widely used by the industry and has been adopted by the major standards organizations.

2.1 The IEEE 12207 standard for Information Technology-software life cycle processes

This standard establishes a framework for software life cycle processes, and provides well-defined terminology to be used by all the stakeholders. It defines processes, activities, and tasks that are to be applied during the acquisition of a system that contains software, a stand-alone software product, and software service. The standard covers the tasks for the overall the life cycle phases during the supply, development, operation, and maintenance of software products.

We briefly describe the processes defined by the standard and their dependencies in a layered architecture. The top layer contains the five primary life cycle processes defined by the standard. These are the acquisition, supply, development, operation, and maintenance processes. The standard also defines four project organizational processes that include the management process. Eight other supporting processes are defined in the middle layer are defined to support the primary processes and the management process in the top layer. The dependencies between the processes are specified. For example, the acquisition and the supply

processes interact to establish the contract for the project and start the management process, which in turn manages the development, operation, and maintenance processes. The operation process depends on the maintenance process to correct errors found, and how the later depends on the development process to redevelop components that require major changes.

3. The Development Process

In this section we focus on the major tasks of the development process based on the standard described above and then survey some of the AI techniques used in supporting the tasks of this process. In particular we focus on the tasks related to requirements analysis, architecture design, coding, and testing.

The system and software architectures play a major role in driving the management activities during the development and maintenance of software systems. The standard provides a flow of the development process activities and associated documents. The system architecture design activity which produces the *Software architecture and requirements allocation description (SARAD) document,* establishes the top-level architecture of the system and defines the software and hardware items of the system. These items are then developed concurrently. The software items development starts with the software requirements analysis activity that produces *the software requirements specification document (SRS)*. Then the software architecture design activity produces the documents related to *software architecture description (SAD), software interface design description (SIDD)*. This is followed by the software coding and testing activities. The application of AI techniques in support of the tasks of these activities is described in the following subsections.

3.1 Software requirements analysis

Requirement Engineering (RE):

Requirements are first expressed in natural language within a set of documents. These documents usually represent "the unresolved views of a group of individuals and will, in most cases be fragmentary, inconsistent, contradictory, not prioritized and often be overstated, beyond actual needs" [31]. The main activities of this phase are requirements elicitation, gathering and analysis and their transformation into a less ambiguous representation [36].

Problems arising during this phase can be summarized as follows:

- Requirements are ambiguous [27]
- Requirements are incomplete, vague and imprecise [34] , [35]
- Requirements are conflicting [35]
- Requirements are volatile [18]
- There are communication problems between the stakeholders [37]
- Requirements are difficult to manage [10]

In the following we explore the techniques used in the requirement engineering field

Processing Natural Language Requirements NLR

The transformation of NLR into specifications and design automatically, began in the early 1980s. In [1], Abbott drew an analogy between the noun phrases used in NL descriptions and the data types used in programming languages. In those days requirements and modeling were not as distinct activities as they are now. In [30];[18];[26], the author noted that verb phrases and to some extent adjectives describe relationships between these entities, operations and functions.

In the following, we give examples of some of the systems that have attempted to produce formal specification from NL Requirements:

In [30], the authors proposed a framework to translate specifications written in NL (English) into formal specifications (TELL). their system was not implemented but set the foundations for future systems. In [5], NL2ACTL system was introduced the, which aims to translate NL sentences, written to express properties of a reactive system, to statements of an action based temporal logic. In [18], the authors developed the FORSEN system which aims to translate NL requirements into the Formal specifications language VDM. This system allowed the detection of ambiguities in the NL requirements.

In the following, we give examples of some of the systems that have attempted to produce OO oriented models from NL Requirements

In [11], the authors defined a general framework for the automatic development of OO models from NL requirements using linguistics instruments. In [20], a Large-scale Object-based Linguistic Interactor Translator Analyser (LOLITA) NLP system was used to develop the NL-OOPS which aims to produce OO specifications from NL requirements in [11], the researchers developed an approach that linked the linguistic world and the conceptual world through a set of linguistic patterns. In [7], the authors developed the Class-Model Builder (CM-Builder), a NL based CASE tools that builds class diagrams specified in UML from NL requirements documents

Knowledge Based Systems (KBS):

In [13], the authors stated that *"The reuse of experts design knowledge can play a significant role in improving the quality and efficiency of the software development process".* KBS were used to store design families, upon the development of the requirements, input and outputs of the system's functionality. The system searches the KB and proposes a design schema which is refined by the user to fully satisfy the requirements. In [31], The READS tool supports both the front end activities such as requirement

discovery, analysis and decomposition and requirements traceability, allocation, testing, and documentation

Ontologies:

Ontologies are developed by many organizations to reuse, integrate, and merge data and knowledge and to achieve interoperability and communication among their software systems. In [34], the authors use semantic web and ontological techniques to elicit, represent, model, analyze and reason about knowledge and information involved in requirements engineering processes. In [4], the researchers have developed the Ontology-based software Development Environment (ODE) based on a software process ontology.

Intelligence Computing for Requirements Engineering:

In this section, we will discuss some of the systems developed using Computational Intelligence (CI) techniques to support requirements engineering The SPECIFIER system [21] can best be viewed as a case based system that takes as input an informal specification of an operation where the pre and post-conditions are given as English sentences. In [35], the authors used fuzzy logic and possibility theory to develop an approximate reasoning schema for inferring relative priority of requirements under uncertainty, to assess requirements priorities . This is to achieve an effective trade off among conflicting requirements so that each conflicting requirement can be satisfied to some degree.

In [12], an approach is presented that uses computational linguistics to analyze textual scenarios, to identify where actors or whole actions are missing from the text, to fill the missing information, and to generate a message sequence chart (MSC) including the information missing from the textual scenario. Then, the requirements analyst validates the generated MSC.

In Jose Del Salgado Martinez et al (Chapter 6 in [19]), the authors constructed a Bayesian network to predict whether a requirements specification has enough quality to be considered as a baseline. In order to structure and quantify the final model of the Bayesian network "Requisites", several information sources were used, such as standards, reports, and through interaction with experts. This Bayesian network represents the knowledge needed when assessing a requirements specification. Requisites were demonstrated on some use cases. After the propagation over the network of information collected about the certainty of a subset of variables, the value predicted determine whether the requirements specification has to be revised or not.

In [3], the authors present a collaborative and situational tool called MUSTER, that has been designed and developed for requirements elicitation workshops. The tool also offers an example of how a group support system, coupled with artificial intelligence, can be applied to very practical activities and situations within the software development process.

3.2 Software architecture design

One of the most important problems facing the software engineer is to develop quality architecture from the requirements model. In this section we describe recent work on software architecture design using AI techniques. Developing the software architecture starts by defining a hierarchy of subsystems and components with allocated responsibilities from the information provided by the requirements and analysis models. AI techniques uses quality attributes to define a goodness function over the space of possible architectures. Some of the most common quality attributes of architecture design used in developing the architecture are modularity, complexity, modifiability, understandability (or clarity), and reusability. Modularity is usually connected to the concept of coupling and cohesion, where designers strive for a modular design by developing the architecture using loosely coupled and highly cohesive subsystems and components. In an earlier work on using AI techniques for software architecture development, Robyn Lutz [14] used Genetic Algorithms (GAs) to search the space of possible hierarchical decompositions of a system. She introduced a fitness function using information theoretic metric capturing the data coupling and control coupling between components. The quality attribute used for the fitness function is related to the complexity and modularity of the produced architecture. Later on, she focused in her further research on Product Line Architectures (PLAs) [15,16] where variation points are explicitly defined to enhance reusability and modifiability of reference architecture that can be used to instantiate a family of architectures. Other work on hierarchical decompositions of a system is summarized in [28].

A promising recent work on synthesizing architecture from requirements using GAs is presented in [29]. Figure 1 shows the process of architecture generation. In this work, the requirements model based on use-cases that captures the functional requirements is used to develop a null architecture that gives the basic decomposition of the functionalities into components. The null architecture is represented by a UML class diagram that is generated from use-case sequence diagrams. The null architecture is used by the GA to first create an initial population of architectures. A fixed library of standard architectural solutions based on styles and patterns is used to produce new generations. The fitness function of the GA is defined by a weighted list of metrics of quality attributes. This function can also be optionally defined by scenarios capturing specific quality attributes. The work presented is restricted to modifiability scenarios because it can be easily formalized. Details of the GA technique used and the results of testing are given in the reference.

3.3 Software coding and testing

Techniques learned from AI research make advanced programming much simpler, especially with regard to information flow and control as a result of advances in knowledge representation. In the following we focus on the AI techniques used in supporting the tasks of coding and testing.

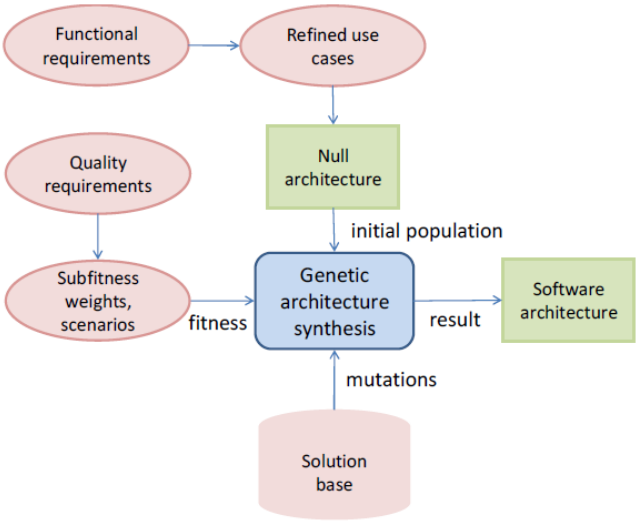

Figure 1. Evolutionary architecture generation Adopted from [29].

a) Coding:
Software engineers can apply AI techniques to help automate or assist the programming process.

Use of AI to help assist the programming process:
The main idea here is to create an expert system to assist software engineers during software development [23], [24]. In [23], this proposal is called the Programmer's Apprentice Project. The Programmer's Apprentice should have the capability of interacting with the human programmers exactly the same way as human assistants would, thereby hopefully increasing the productivity of the human programmers. At first, the Apprentice would only be able to handle "the simplest and most routine parts" of programming. As time progresses and research continues, the Apprentice should be able to deal with more complicated tasks. The human programmers will still be necessary to implement code of a 'tricky' nature (such as abstract reasoning or to better cater human preferences).

Use of AI to help automate the programming process:
The idea here is to have a completely automated program synthesis. This is done by having human specialists write a complete and concise specification of the desired software; so that, a system can generate "functions, data structures, or entire programs" directly from the specifications [8]. There are many possible AI technologies that could be applied.

Analogical reasoning in software reuse can be used. The idea is to find a system with similar requirements and modify it. Although this process looks feasible, it has not been demonstrated in software engineering to any great extent.
Closely related to analogical reasoning techniques is Case-based reasoning (CBR). CBR is based upon the premise that similar problems are best solved with similar solutions. CBR is argued to offer a number of advantages over many other knowledge management techniques. For program synthesis retrieval from component repositories and the reuse of successful past Experience is important. As an example, one application of CBR technology was to support the reuse of software packages within Ada and C program libraries.
The idea of experience reuse, the most ambitious form of CBR-supported reuse, is closely aligned with what is called Experience Factory. This field is also known as Organizational Learning, researches methods and techniques for the management, elicitation, and adaptation of reusable artifacts from software engineering projects. An Experience Factory is based upon a number of premises such as a feedback process, appropriate storage of experience, and support of reuse and retrieval [25].
Constraint programming is another AI technique that is applied in software engineering. Constraint programming has been, for example, used to design the PTIDEJ system (Pattern Trace Identification, Detection and Enhancement in Java. PTIDEJ is an automated system designed to identify micro-architectures looking like design patterns in object oriented source code. A micro-architecture defines a subset of classes in an objected oriented program. The main interest of PTIDEJ is that it is able to provide explanations for its answers. This is really interesting since coding and software engineering is often considered a form of art and where fully automated systems are not always appreciated by potential users (or programmers).
Search Based Software Engineering (SBSE) is an emerging research topic that focuses on representing aspects of Software Engineering as problems that may be solved using meta-heuristic search algorithms developed in AI. SBSE is the reformulation of software engineering tasks as optimization problems. One of the optimization and search techniques that can be used are genetic algorithms. Genetic algorithms are used for automatic code generation by optimizing a population of trial solutions to a problem. The individuals in the population are computer programs.

b) Testing: Software testing remains an expensive task in the development process and one of the main challenges concerns its possible automation. AI techniques can play a vital role in this regard. One of these techniques are constraint solving techniques. Since the seminal work of Offut and De Millo in the context of mutation testing [40], much attention has been devoted to the use of constraint solving techniques in the automation of software testing (Constraint-based testing). ATGen, for example, is a software test data generator based on

symbolic execution and constraint logic programming for ADA programs.

There are many other ways how AI techniques can support the testing process [19]. One of the earliest studies to suggest adoption of a knowledge based system for testing was by Bering and Crawford [2] who describe a Prolog based expert system that takes a Cobol program as input, parses the input to identify relevant conditions and then aims to generate test data based on the conditions.

A more active area of research since the mid-1990s has been the use of AI planning for testing. An AI planner could generate test cases, consisting of a sequence of commands by representing commands as operators, providing initial states, and setting the goal as testing for correct system behavior [9]. AI planning was also used for testing distributed systems [6] and for the generation of test cases for graphical user interfaces [17].

A study by Kobbacy, et al [38] has shown that the use of genetic algorithms for optimization has grown substantially since the 1980s. This trend is also present in their use in testing, with numerous studies aiming to take advantage of their properties in an attempt to generate optimal test cases. The authors in [33], for example, used genetic algorithms for testing object oriented programs where the main aim was to construct test cases consisting of a sequence of method calls.

Fuzzy logic is another AI technique that is applied in software testing to manage the uncertainty involved in this phase of software development [39].

4. Open Problems

Open problems that Artificial Intelligence can help in the requirements engineering phase include the following: [19]

- Disambiguating natural language requirements
- Developing knowledge based systems and ontologies to manage the requirements and model problem domains
- The use of computational intelligence to solve the problems of incompleteness and prioritization of requirements.

One of the most difficult problems is the problem of transforming requirements into architectures. Much research is needed in this area to address the ever increasing complexity of functional and non-functional requirements. Recent important research problems are developing product line architectures and service-oriented architectures using AI techniques.

Test data generation is notoriously hard. Recent work (including that one search based testing) has made progress towards the ultimate goal of fully automated test case design. However, the techniques that are being developed are often hampered by features of the programs under test.

One area that has received some attention is the use of automated algorithms with machine learning to make repair assignments. In any case, more studies with respect to the appropriate criteria for selecting assignment policy, reward mechanisms and management goals need to be undertaken.

One open problem with Search-Based Software Testing techniques, and Search-Based Test Data Generation techniques in particular, is lack of handling of the execution environment that the software under test lives within. Current state of the art in test data generation, for example, ignores or fails to handle interactions with the underlying operating system, the file system, network access and databases on which they may be dependent.

Another problem with Search-Based Software Testing techniques is: because fitness functions are heuristics, there are cases in which they fail to give adequate guidance to the search.

Constraint-Based Testing (CBT) is the process of generating test cases from programs or models by using the Constraint Programming technology. Scalability is the main challenge that CBT tools have to face to.

Dealing with more than hundred of thousands lines of code, with dynamic constructions such as huge dynamic data structures, with non-linear numerical constraints extracted from complex statements are some of the problems we have to deal with.

5. Conclusions

In this paper, we surveyed promising research work on applying AI techniques to solve some of the most important problems facing the software engineer. We surveyed research in the development activities of requirements engineering, software architecture design, and coding and testing processes. We summarized the most important open problems in these active research areas.

6. Aknowledgements

This research work was funded by Qatar National Research Fund (QNRF) under the National Priori-ties Research Program (NPRP) Grant No.: 09-1205-2-470.

7. References

[1] Abbott, R. J. (1983). Program design by informal English descriptions. *CACM*, *26*(11), 882–894.

[2] Bering, C. A., & Crawford, M. W. (1988). Using an expert system to test a logistics information system. In Proceedings of the IEEE National Aerospace and Electronics Conference (pp. 1363-1368), Dayton, OH. Washington DC: IEEE Computer Society.

[3] Coulin, C., Zowghi, D., & Sahraoui, A. (2010). MUSTER: A Situational Tool for Requirements Elicitation. In F. Meziane, & S. Vadera (Eds.), *Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects* (pp. 146-165).

[4] Falbo, R. A., Guizzardi, G., Natali, A. C., Bertollo, G., Ruy, F. F., & Mian, P. G. (2002), Towards semantic software engineering environments. *Proceedings of the 14th international Conference on Software Engineering and Knowledge Engineering*, (pp. 477-478).

[5] Fantechi, A., Gnesi, S., Ristori, G., Carenini, M., Vanocchi, M., & Moreschini, P. (1994). Assisting requirement formalization by means of natural language translation. *Formal Methods in System Design*, *4*(3), 243–263.
[6] Gupta, M., Bastani, F., Khan, L., & Yen, I.-L. (2004). Automated test data generation using MEA-graph planning. In *Proceedings of the Sixteenth IEEE Conference on Tools with Artificial Intelligence* (pp. 174-182). Washington, DC: IEEE Computer Society.
[7] Harmain, H. M., & Gaizauskas, R. (2003). CM-Builder: A natural language-based CASE tool for object-oriented analysis. *Automated Software Engineering Journal*, *10*(2), 157–181.
[8] Hewett, Micheal, and Rattikorn Hewett (1994). 1994 IEEE 10th Conference on Artificial Intelligence for Applications.
[9] Howe, A. E., von Mayrhauser, A., & Mraz, R. T. (1995). Test sequences as plans: an experiment in using an AI planner to generate system tests. In Proceedings of the Tenth Conference on Knowledge-Based Software Engineering (pp. 184-191).
[10] Hull, E., Jackson, K., & Dick, J. (2005). *Requirements Engineering*. Berlin: Springer.
[11] Juristo, N., Moreno, A. M., & López, M. (2000). How to use linguistics instruments for Object-Oriented Analysis. *IEEE Software*, (May/June): 80–89..
[12] Kof, L. (2010). From Textual Scenarios to Message Sequence Charts. In F. Meziane, & S. Vadera (Eds.), *Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects* (pp. 83-105).
[13] Lubars, M. D., & Harandi, M. T. (1987). Knowledge- based software design using design schemas. *In Proceedings of the 9th international Conference on Software Engineering*, (pp. 253-262).
[14] Lutz, R. "Evolving good hierarchical decompositions of complex systems," Journal of Systems Architecture 47 (2001), 613–634.
[15] Lutz, R. "A Survey of Product-Line Verification and Validation Techniques," JPL-NASA Technical Report, 2007 http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/41221/1/07-2165.pdf
[16] Jing (Janet) J. Liu, Samik Basu and Robyn R. Lutz: Generating Variation Point Obligations for Compositional Model Checking of Software Product Lines. Journal of Automated Software Engineering, p. 29, vol. 18, 2011
[17] Memon, A. M., Pollack, M. E., & Soffa, M. L. (1999). Using a Goal Driven Approach to Generate Test Cases for GUIs. In *Proceedings of the Twenty-first International Conference on Software Engineering* (pp. 257-266).
[18] Meziane, F. (1994). *From English to Formal Specifications*. PhD Thesis, University of Salford, UK.
[19] Meziane, F. and Vadera, S., (2010). Artificial Intelligence in Software Engineering Current Developments and Future Prospects, In "Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects", IGI Global
[20] Mich, L. (1996). NL-OOPS: from natural language to object, oriented requirements using
the natural language processing system LOLITA. *Natural Language Engineering*, *2*(2), 161–187.
[21] Miriyala, K., & Harandi, M. T. (1991). Automatic derivation of formal software specifications from informal descriptions. *IEEE Transactions on Software Engineering*, *17*(10), 1126–1142.
[22] Moreno, C. A., Juristo, N., & Van de Riet, R. P. (2000). Formal justification in object-oriented modelling: A linguistic approach. *Data & Knowledge Engineering*,*33*, 25–47.
[23] Partridge, Derek, ed. (1991). Artificial Intelligence and Software Engineering. New Jersey: University of Exeter, 1991.
[24] Phil B. (1999). The Use of Artificial Intelligence for Program Development, http://www.philforhumanity.com/The_Use_of_Artificial_Intelligence_for_Program_Development.html
[25] Shepperd, M. J. (2009). Case-based reasoning and software engineering. Empirical Software Engineering. Springer. Retrieved from http://hdl.handle.net/2438/3049.
[26] Poo, D. C. C., & Lee, S. Y. (1995). Domain object identification through events and functions. *Information and Software Technology*, *37*(11), 609–621.
[27] Presland, S. G. (1986). *The analysis of natural language requirements documents*. PhD Thesis, University of Liverpool, UK.
[28] Outi Räihä, A survey on search-based software design," Computer Science Review, 4 (2 0 1 0) 203 – 249.
[29] Outi Räihä, Hadaytullah , Kai Koskimies and Erkki Mäkinen "Synthesizing Architecture from Requirements: A Genetic Approach" UNIVERSITY OF TAMPERE DEPARTMENT OF COMPUTER SCIENCES SERIES OF PUBLICATIONS – NET PUBLICATIONS , AUGUST 2010
[30] Saeki, M., Horai, H., & Enomoto, H. (1989). Software development process from natural language specification. In *Proceedings of the 11th international Conference on Software Engineering*. (pp. 64-73), Pittsburgh, PA.
[31] Smith, T. J. (1993). READS: a requirements engineering tool. *Proceedings of IEEE International Symposium on Requirements Engineering*, (pp. 94–97), San Diego.
SSBSE (2010). http://www.ssbse.org, checked 10.5.2011.
[32] Vadera, S., & Meziane, F. (1994). From English to Formal Specifications. *The Computer Journal*, *37*(9), 753–763.
von Mayrhauser, A., France, R., Scheetz, M., & Dahlman, E. (2000). Generating test-cases from an object-oriented model with an artifical-intelligence planning system. *IEEE Transactions on Reliability*, *49*(1), 26–36. doi:10.1109/24.855534
[33] Wappler, S., & Wegener, J. (2006). Evolutionary unit testing of object-oriented software using strongly-typed genetic programming. In Proceedings of the Eighth Annual Conference on Genetic and Evolutionary Computation (pp. 1925-1932), Seattle, WA. New York: ACM Press.
[34] Yang, Y., Xia, F., Zhang, W., Xiao, X., Li, Y., & Li, X. (2008). Towards Semantic Requirement Engineering. *IEEE International Workshop on Semantic Computing and Systems* (pp. 67-71).
[35] Yen, J., & Liu, F. X. (1995). A Formal Approach to the Analysis of Priorities of Imprecise Conflicting Requirements. In *Proceedings of the 7th international Conference on Tools with Artificial intelligence*. Herndon, VA , USA
[36] Young, R. R. (2003). *The requirements Engineering Handbook*. Norwood, MA: Artech House Inc.
[37] Zave, P. (1997). Classification of Research Efforts in Requirements Engineering. *ACM Computing Surveys*, *29*(4), 315–321.
[38] Kobbacy, K. A., Vadera, S., & Rasmy, M. H. (2007). AI and OR in management of operations: history and trends. The Journal of the Operational Research Society, 58, 10–28. doi:10.1057/palgrave. jors.2602132
[39] Nand, S., Kaur, A., Jain S. (2007).Use Of Fuzzy Logic In Software Development. Issues in Information Systems. Volume VIII, No. 2, pp. 238-244
[40] DeMillo, R.A., Offutt, A.J. (1991). Constraint-based automatic test data generation. IEEE Transactions on Software Engineering 17 (9), 900–910.