Journal of Information Engineering and Applications
ISSN 2224-5782 (print) ISSN 2225-0506 (online)
Vol.5, No.1, 2015


# The Definition of Intelligent Computer Aided Software Engineering (I-CASE) Tools

Ayad T. Imam [1*] Ayman J. Alnsour[2] Aysh Al-Hroob[1]

1. Faculty of Information Technology, Al-Isra University, Amman 11622, Jordan
2. Faculty of Engineering, Al-Isra University, Amman 11622, Jordan

* E-mail of the corresponding author: alzobaydi_ayad@iu.edu.jo

**Abstract**

The growing complexity of the software systems being developed and the use of different methodologies indicate the need for more computer support for automating software development process and evolution activity. Currently, Computer-Aided Software Engineering (CASE), which is a set of software systems aimed to support set of software process activities, does this automation. While CASE tools prove its importance to develop high quality software, unfortunately CASE tools doesn't cover all software development activities. This is because some activities need intellectual human skills, which are not currently available as computer software. To solve this shortcoming, Artificial Intelligence (AI) approaches are the ones that can be used to develop software tools imitating these intellectual skills.

This paper presents the definition of Intelligent Computer Aided Software Engineering (I-CASE). The definition encompasses two steps. The first step is a clear decomposition of each basic software development activity to sub activities, and classify each one of them whether it is an intellectual or procedural job. The second step is the addressing of each intellectual (un-automated) one to proper AI-based approach. These tools may be integrated into a package as an Integrated Development Environment (IDE) or could be used individually. The discussion and the next implementation step are reported.

**Keywords**: Software Engineering, CASE tools, Artificial Intelligence

## 1. Introduction

The aim of Software Engineering (SE) is to manage the development of software product via structured and systematic approach known as software development process. The software development process is a repeatable and predictable formation that comes in several different models or methodologies and used for improving the productivity and quality of producing software. Each model of software development process has its own organizing and coordinated set of activities that should be performed by engineers, managers and technical writers to produce the aimed software. We can name assorted common software development processes that are in use today, like a waterfall, prototyping, iterative and incremental development, spiral development, rapid application development, Agile, extreme programming (XP), Lean, and Scrum. No matter how they are coordinated in their software development model, the principle or basic activities that should be performed for software manufacturing are shared among these models, which are Requirement engineering, Software Architectural Design, Implementation, Software Testing, Software Documentation, Training and Support, and Maintenance (Sommerville, 2010) (Pressman, 2010).

Currently, SE is attaining effective techniques like unified modelling language (UML), programming languages like Java & C#, and standardization. Furthermore, Software development is typically achieved with a support from Computer-Aided Software Engineering tools, CASE tools for abbreviation. CASE is a set of software systems, which aimed to automate (fully or partially) some of software development process activities, especially on a large scale or complex projects. An example of the set of software development process activities included in CASE are Data dictionary to manage design entities, Graphical editors to develop system model, Graphical User Interface (GUI) builder to build a user interface, Debuggers to locate the program's fault, and automated translators (compilers and interpreters) to generate new versions of a program. These tools are available, as separately or as a package like Rational Rose, Figure1 shows the typical CASE toolset architecture (Sommerville, 2010) (Pressman, 2010).

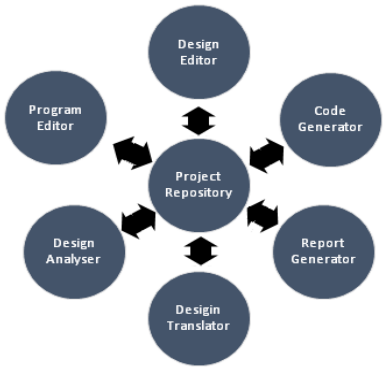
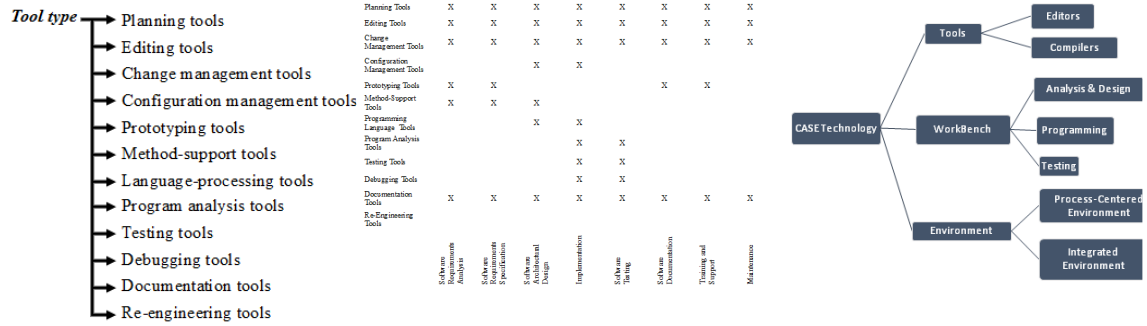



Figure1: CASE Toolset Architecture

Traditionally, to help project managing, CASE tools are classified using three criteria, namely: functional-type, process activity, and integration environment, which all illustrated in Figure2-a, Figure2-b, and Figure2-c respectively (Sommerville, 2010).




(a) CASE Tools Classification based on using Functional Type Criteria

(b) CASE Tools Classification based on Process-Activity Criteria

(c) CASE Tools Classification based on Integration Environment Criteria

Figure2: Classifications of CASE tools

Easy to note that the use of CASE software tools has led to considerable advancement in the software industry; in that it helps getting product of high quality, minimum faults, and with time and cost competent manner. Yet, not all of the activities of software development process are supported by CASE software technology. This is because some software process activities in the software engineering need creative thought to be achieved, which not easy to be automated by software (Harman, 2012).

To solve this problem, we suggest in this paper, the using Artificial Intelligence (AI) approaches to develop software tools that imitate human intellectual skills like creativity, Natural Language Processing (NLP), and inference. I-CASE, which stands for Intelligent Computer Aided Software Engineering, is the name we give to this suggested technology.

This paper briefly reviews the previous AI works for SE. Using this reviewing, we are going to show what AI approaches can be used to develop I-CASE tools. The paper concludes the suggested I-CASE tools as a solution for the criticisms of CASE tools and an overview of the solution's elements.

## 1. AI in SE

AI and related topics, viz: Knowledge Engineering and Knowledge Based Systems (KE & KBS), Knowledge Management (KM), Expert Systems (ES), Fuzzy Logics (FL), Artificial Neural Networks (ANN), and others share with SE same goals. The first shared goal is the solving of a complex problem via utilizing similar sequence, namely: problem definitions, discovering problem features, searching for already defined solution of analogous problem, and conclude result. The second shared goal is that both AI and SE handle the modelling of real world objects like process models, business processes, or expert knowledge (Meziane & Vadera, 2012).

SE remains a very skilful knowledge and experiences human activity, which is known as the problem solving skill. Therefore, AI and related topics will continue playing a main role in automating activities of software development. The existing AI's works have already confirmed that there are substantially benefiting for the SE, shown by the amazing range of achievements in surpass humans in some software engineering activities

(Rodriguez, et al., 2011). AI is used to get insight into the properties of SE problems as well as the solutions' domain spaces rather than abstract solutions to individual SE problems. Exciting and interesting examples can be seen in certain situations which are the well understood of miss estimation risk of requirements and ending time of the software project under developing, problems related to program comprehension, and finally, the developing of quality, faults, effort and performance predictive models (Harman, et al., 2009). More elaborated review about AI's work for SE are found in (Meziane & Vadera, 2012) and (Harman, 2012). Examples of AI's techniques used in the SE area:

- AI's searching techniques using heuristic and optimisation in the Search Based Software Engineering (SBSE) field of SE (Harman, 2007).
- AI's reasoning techniques in the presence of uncertainty using Fuzzy and probabilistic techniques to cater for ill-defined, fuzzy, incomplete and noisy information in SE development process (Krogmann, et al., 2010) (Danilchenko & Fox, 2012).
- AI's classification, learning and prediction techniques to model software reliability, analysis of users, and predict software costs as part of project planning (Antoniol, et al., 2009).
- Rule-based systems, Experience-based systems, and Case-based systems have been utilised to support agile methods like RAIS designing system (Ramachandran, 2008) and ECG-RF system for composing code (Imam, et al., 2014)
- Service-oriented and product line using AI techniques (Ammar, et al., 2012)

Through the following paragraphs, we are going to make review of current AI techniques applied to each basic activity of the SE development process as reported by a number of researchers. The review is accompanied by analysing each activity to its sub activities and suggesting a suitable AI approach that may be used to automate it.

### *1.1 Requirements Related Tools*

Requirements engineering is the most important stage among software development process stages, as a small error at this stage may cause huge deflection in a software designs and implementations, which in turn will result different software from that defined by the stakeholder. Software requirements come from a diversity of sources like to relate stakeholders, standards, laws… etc., in natural language form. These requirements are re-produced in classes, specific, and clear set of requirement form. With these properties, requirements will support the generating of correct, consistent and fault-tolerant software models. As it is known, natural language has a number of challenges to be faced. Thus, the automation of this activity should be able to solve the incomplete, ambiguous, and contradictory requirements. These challenges need semantic handling of natural language communication skill. The automation of this preliminary stage will elongate a standing challenge of automated software development activities. As shown in Figure 3, effectual requirements analysis includes four tasks (Sommerville, 2010) (Umber, et al., 2011) ((P&R), n.d.):

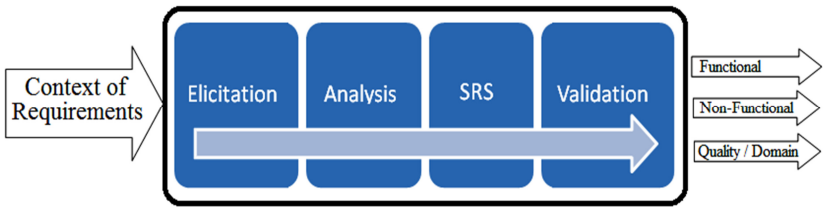



Figure 3: Tasks of Requirements Engineering

- Requirements elicitation: gathering the requirements of a system for customers, users, administrators and other stakeholders.
- Requirements analysis: checking the gathered requirements against completeness, clearness, reliable and vagueness. Also resolving perceptible conflicts
- Software Requirements specification (SRS): classifying the analysed requirements into functional requirements, non-functional requirement, and domain requirements. The specifications possibly include a number of use cases.
- Requirements Validation: ensures the agreement between the input context information and the output

requirements artefacts of the total requirements engineering process.

Currently, the software tools used in requirements engineering are of supported type rather than fully automated type. Full automation of requirements engineering tasks is achieved via AI's approaches such as logic based approach, Artificial Neural Network (ANN), Natural Language Processing (NLP) and ontologies' knowledge based systems. These approaches are able (with certain challenges from one natural language to another) to handle the problems accompanied with the natural language input to requirements engineering in its four tasks, like ambiguity, contradiction, and modelling problem domains. The Commercial-Off-The-Shelf (COTS) NLP software tools facilitates developing NLP-based software for any purpose like extracting different types of software requirements from NL context (Umber, et al., 2011) (Ormandjieva, et al., 2007) (Brown, 2005) (Meziane & Vadera, 2012).

### *1.2 Design And Code Generation Related Tools*

Design activity gives an abstract representation of a software system. Software architectural design should make sure that the software system under development would meet the requirements specified by the clients and have the flexibility to accommodate any future requirements. Software architectural design is an intellectual activity that needs a human skill of creativity to handle it. This activity is concerned with converting the software design to program code. Converting a design to code could be the clearest activity, among other software engineering activities, where code generator programs are the software type that automates this activity. Currently, deductive and inductive AI's elementary approaches are used as suitable techniques to tackle this intellectual task; i.e. Design of a solution (Brown, 2005) (Danilchenko & Fox, 2012).

Code generation is a method used for fast developing of software by using Automatic Code Generation (ACG) software tools, which save time, save effort, improve software quality, improve software accuracy, and free software developers from dreary routine jobs. Different types of code generator are exist like code wizard, the forward-engineering tool set integrated with modelling tools (convert a description of solution into code), the reverse-engineering tools and compilers. There are two types of code generator. The first type is called a passive code generator, in which the code produced from it needs to be adjusted or modified by developers. The second type is called an active code generator, which are incorporated in the developing process and rerun to regenerate the new code while developing the software (Kitzelmann, 2010) (Imam, et al., 2014). While the current ACG software are of notable achievements, it is also notable that the most of these software tools, particularly the forward-engineering tool set integrated with modelling tools, still need considerable input information from humans that is the design of the system. This weakness is considered normal since design is an exceptionally creative (none routine) task that is a very hard aspect to be automated by software. Hence, current ACG software tools freelance the software developers to work on non-routine tasks rather than fully replacing the design and implementation stage (Danilchenko & Fox, 2012).

The fact that designing a solution is extremely intellectual duty, which is hard to be directly automated makes the efforts of developing designing software tools looks like myth. Deductive programming (DP) implements deductive reasoning to develop software that uses both UML diagrams and program synthesis to generate algorithmic portions used later by ACG. Inductive programming (IP) implements inductive reasoning to develop software that assists the designing of algorithm that will be used later by ACG for assembling of executable programs, which includes loops or recursion. Also, there is a semi-automatic induction of programs that uses exemplary behaviour to learn recursive policies and end-user programming in intelligent agents (Kitzelmann, 2010) (Danilchenko & Fox, 2012). Example of this strategy is the utilizing of automated algorithms with machine learning to do repair assignment (Ammar, et al., 2012). Another example is the using of rule-based system to develop code generator software that gets used a pre-written chunk of programs to compose a new code for a new program (Imam, et al., 2014).

Converting requirements to architectural design is an easier said than done problem. Actually, this area desires a lot of research to address the growing complexity of system's requirements either functional or non-functional.

### *1.3 Verifying & Validation (Testing) Related Tools*

In this activity, the resulted software is to be tested by parts and as a whole. This is to make sure the consistency of the software parts and the soundness of the software results. Currently there are certain programs that work as tester for other software. The software under development needs to be confirmed whether it meets its specifications and judge the correctness of its outputs. This confirmation is achieved by software testing, which encompass of Validation and Verification tasks (V&V). Verification is the evaluating of work elements, which are plans, requirement specs, design specs, code, and test cases by using reviews, walkthroughs, and inspection methods. Validation is the evaluating of the resulted software either during or at the end of the development

process by using methods to discover errors, faults, and failures (Sommerville, 2010) (Gebhardt & Kaske, 2011). Some of the testing methods used either in verification and validation, achieved manually, where the software tester arranges test cases for different levels and sections of the code, carries out the tests, and reports the results to his supervisor. There are automated software test tools that support the achieving other verification and validation methods. Obviously, manual testing has limitations like the consuming of time and resource, and the confirmation of the rightness of the used test cases. Certainly, these limitations of the manual testing can be overcome using automated test tools (Gebhardt & Kaske, 2011).

Notably, the generation of test cases is arduous and the techniques used to do that are encumbered by the properties of the software to be tested. Examples of the challenges are:

1. Ignore or fail to make active interactions with the operating system, network access and databases results weak interacting with the environment of the software under testing.
2. In AI heuristic dependent testing software, sufficient supervision upon the search is not applied in some cases
3. Scalability of the testing software of constraint-programming approach.

Steps forward achieving the aim of entirely automated design of test case can be shown in the recent works like Search-Based Software Testing (SBT), which uses AI' heuristic search, and Constraint-Based Testing (CBT) techniques, which uses Constraint Programming technology (Ammar, et al., 2012).

### *1.4 Software Documentation Related Tools*

Documentation of software can be defined as a written text aims to file the inner design of the software for any future maintenance and improvement, and to explain how to use the software. Software documentation could represent different things to different role people. Documentation is a significant activity of software engineering that makes the reviewing a more smooth job to accomplish its anticipated goals. Stakeholders of the software identify the properties and functions of the software via documentation's five types, which are (Pressman, 2010) ((P&R), n.d.):

1. Requirements documentation type: the groundwork of the software development process that identifies the functions, characteristics, or qualities of a software system.
2. Architectural design documentation type: pilot view of software that includes the units and their relationships as well as the system's relations to its environment.
3. Technical documentation type: documentation of algorithms and code.
4. End user documentation type: manuals used by the end-user including system's administrators, maintenance and other related staff.
5. Marketing documentation type: instructions for marketing and market's product.

Documentation is a composition (intellectual) skill activity. Yet, we can see examples of software automated documenters (known also as document assemblers and documentation generator) like Sandcastle, Doxygen, EiffelStudio, javadoc, AutoDocs, ROBODoc, and others. These documenters perform a number of limited editing jobs like extracting headers using three steps: scanning, analysing, and generating that by including the documentation in the source code. Nevertheless, human touch is necessary and can't be avoided (Pressman, 2010). This shows that the efforts went for creating automated software documentation didn't reach the final goal, which is fully automated documentation software.

The above five types of documentation can be automated via the using of different approaches, AI is definitely among them. The documenters, available nowadays, support the different five types of documentation listed above. A worth to see comparison is available in (Anon., n.d.).

### *1.5 Other Activities*

- Training and Support: this activity is a part of the deployment stage. It is important to make users accommodate with the software prior to using it. Usually, users will have many questions about the functions and the way of using the software. Learning strategies and knowledge transfer are human cognition skills (Wallace, 2012) (Antoniol, et al., 2009). Computer Aided Instructions (CAI) or Electronic Learning (e-learning) software form a class of software that can be used to automate teaching and training activity. Examples of such applications are MOODL and Blackboard.
- Maintenance: this is an after delivery stage activity. The maintaining is an important activity to keep the software coping with lately discovered requirements, changes, or problem. Maintenance is a partial

development for selected portion of the software, and it may take longer time than the initial development of the software. Reused component is the strategy used for maintenance old software (and even developing new software). Man hand stills required to gather the reused components either for maintenance or for developing (Weimer, et al., 2009) (W.B. & Harman, 2010).

- Project planning: a broad assortment of software tools are developed to help in the planning, control and management of projects, and communications between team members. Effective project management tools encompass software for information transmission between all team members and between different software tools. Example of project management tools are planning, scheduling, and risk management (Antoniol, et al., 2009) (Wallace, 2012).

## 2. The Definition Of I-CASE Tools

Based on the above review, we conclude the current reported SE activities that use AI approaches and techniques in Table 1.

Table1: SE Activities use AI Approaches and Techniques

| SE Activity | AI approach |
|---|---|
| Project planning & Scheduling | • Knowledge Based System (KBS)<br>• Case Based Reasoning (CBR)<br>• Genetic Algorithm (GA)<br>• AI's classification, learning and prediction techniques |
| Requirement engineering | • Knowledge Based System (KBS)<br>• Computational Intelligence<br>• Probabilistic reasoning<br>• AI's classification, learning and prediction techniques |
| Testing | • Knowledge Based System (KBS)<br>• Genetic Algorithm (GA)<br>• AI planning methods |
| Coding | • Expert system (ES)<br>• Case Based Reasoning (CBR)<br>• Rule induction |
| Risk Management | • Artificial Neural Network (ANN)<br>• Probabilistic reasoning |
| Evolving & Maintenance | • Artificial Neural Network (ANN)<br>• Genetic Algorithm (GA) |
| Search Based Software Engineering (SBSE) | • AI's heuristic and optimisation searching techniques |

As shown in Table1, the main three approaches of AI that are logic based approach, connectionist approach, and GA where notably used, as well as the heuristic search and optimization applications of AI. Based on that, and up to our knowledge, other AI key applications like recognition, NLP and creativity are not used until now; where there are many SE activities need such automation tools.

The AI-based tools, used for developing new software, are used here to define an extended version of the current CASE tools, and will be called Intelligent Computer Aided Software Engineering (I-CASE). I-CASE tool set aims to fully automate the software development process activities rather than supporting the development process as CASE tools do. The software development process activities aimed to be automated by I-CASE tools include the activities that require human skills like recognition, analysis, creativity, prediction. Examples of these tools are applications that can understand natural language (recognition and analysing), can create design of information system, can predict changes and can make auto-fixing errors (evolving system). AI algorithms, methods and techniques are very suitable to develop significant and successful software applications, which affect those intellectual activities of software engineering.

Classifying of I-CASE tools is based on considering software development activities as main classes. This means that we shall have four sets of I-CASE tools: Requirements related tools, Design and Code-generating related

tools, Software Testing (V&V) related tools, and Documentation related tools. Other activities may form a group that encompasses training, maintenance and project planning activities. As it is obvious, this classification makes no need for extra classifications as defined now. Also, each software development model (like water fall) specifies its software tools that imitate the steps of that model. This classification will facilitate the planning of the development of software project and can be used as a framework for developing an I-CASE integrated development environment. Figure 4 shows the classification of the suggested I-CASE toolset.

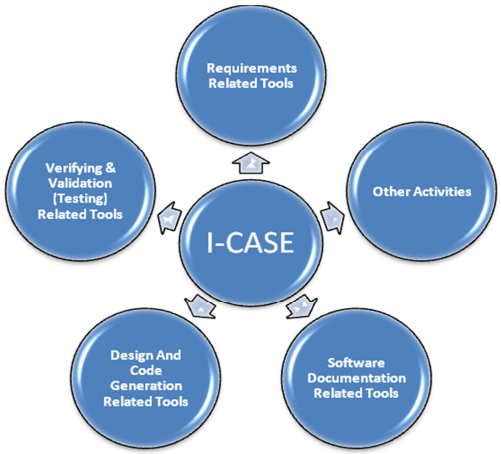



Figure 4: The Suggested I-CASE Toolset Architecture

Decomposition of each major activity to sub tasks facilitates computerisation and thus achieves the full automation of this major activity. In the previous paragraphs, we reported the sub-activities of each of the aforementioned five major sets, which some are already computerised and others are not yet. As the decomposition facilitates the computerisation of an activity, it is important to maintain integrity among the computerised sub activates to achieve the full computation of each of the major activities, which in turn should have integrity to achieve the ultimate goal that is I-CASE tool set.

Having such AI-based applications; i.e. I-CASE software tools, will surely add more to the legacy of AI works in SE. Also, the utilizing of I-CASE in different SE areas will give more flexibility tools to the software developers.

## 3. Expected Challenges and Recommended Solutions

AI techniques that have been used in SE development showed a number of problems and challenges. Recalling that both of AI and SE, are dynamic research fields that continually keep changing and improvements, makes us sure that there are more to come. Based on this fact, it is worthy to abbreviate here some recommendations as reported by different researchers. These challenges and recommendations would be of great help to face challenges lie ahead developing I-CASE tools:

- Embedding AI techniques into existing SE environment: Development and deployment of innovative AI-friendly software require both SE processes and software products to be integrated with AI environment, which encompasses the intelligent development tools along with the decision support systems (DSS). This environment can be used to sustain the methods defined for automation the largely human-intensive software development processes like analysis, building, and testing of software product (Filieri, et al., 2012).
- Define Approaches Instead of Solving Cases: it is notable that the recent using of AI in SE produces solution for individual problem rather than class of problems. To make these AI-based solutions more active, they should be moved up to be a solution approach for a problem's class rather than a solution for individual problem (Poulding & Clark, 2010) (Staunton & Clark, 2011).
- Creating Adaptive Software: AI evolving approaches and techniques like learning, prediction and optimisation can be used in certain activities of software development process like testing and evolution. Principally, evolving process can be used also to address the common challenges found in SE like self-adapting systems and autonomic computing. To do so, a number of techniques could be used like optimisation and genetic programming (Arcuri & Yao, 2008) (Weimer, et al., 2009) (W.B. & Harman, 2010).
- Utilization of Multi-core Computation to handle the computationally high-cost AI techniques: Fortunately, the parallelism nature of some AI techniques, like evolutionary algorithms, has been employed to solve large-scale problems like regression testing, software re-modularisation, and concept location. This means that methods

for converting existing sequential programming forms into parallel forms should be defined to fit the multi-core computing principles (Asadi, et al., 2010) (Yoo, et al., 2011) (Linderman, et al., 2008).

## 4. Conclusion

As the software products become large-scale, their maintenance becomes complex, and the growth in the software industry is increased, the demanding for fully automating of software development activities becomes necessary. Poor defined application domains, noisy, changing and conflicting objectives of the developed software are some of the problems' properties of the software development process. These properties would force the software developers to change the development and deployment methods and to get the support of intelligent software development tools for speeding up the development process and decreasing the cost of the development. AI techniques prove to be the best answer to these types of problem, since they based on imitating human intellectual skills.

In this paper, we surveyed and analysed research works, and define I-CASE tools that based on AI techniques. The ultimate goal of I-CASE is to speed up and facilitate the efforts of the software development. The survey, analysing, and definition of I-CASE tools presented in this research cover the development activities of requirements engineering, design, coding, testing, documentation, and other processes; highlighting the problems facing the automation of these activities and the required AI techniques.

## References

(P&R), O. (n.d.). Human Resources Management (HRM). Retrieved Dec. 2014, from Personal and readiness information management (P&R IM) : http://www.prim.osd.mil/cap/req-analysis-def.html?p=1.1.7.1.

Ammar, H., Abdelmoez, W., & Hamdi, M. S. (2012). Software Engineering using Artificial Intelligence Techniques: Current State and Open Problems. First Taibah University International Conference on Computing and Information Technology (ICCIT 2012), (pp. 24-29). Al-Madinah Al-Munawwarah, Saudi Arab.

Antoniol, G., Gueorguiev, S., & Harman, M. (2009). Software Project Planning for Robustness and Completion Time in The Presence of Uncertainty Using Multi Objective Search Based Software Engineering. ACM Genetic and Evolutionary Computation Conference (GECCO 2009) (pp. 1673-1680). Montreal-Canada: ACM.

Arcuri, A., & Yao, X. (2008). A Novel Co-evolutionary Approach to Automatic Software Bug Fixing. The IEEE Congress on Evolutionary Computation (CEC '08) (pp. 162–168.). Hong Kong, China: IEEE Computer Society.

Asadi, F., Antoniol, G., & Gu´eh´eneuc, Y. (2010). Concept Location with Genetic Algorithms: A Comparison of Four Distributed Architectures. 2nd International Symposium on Search based Software Engineering (SSBSE 2010). Benevento- Italy: IEEE.

Brown, D. C. (2005). Artificial Intelligence for Design Process Improvement. In J. Clarkson, & C. Eckert, Design process improvement: A review of current practice (pp. 158-173). Springer.

Comparison of Documentation Generators. (n.d.). Retrieved Dec. 2014, from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Comparison_of_documentation_generators

Danilchenko, Y., & Fox, R. (2012). Automated Code Generation Using Case-Based Reasoning, Routine Design and Template-Based Programming. In S. Visa, A. Inoue, & A. Ralescu (Ed.), The 23rd Midwest Artificial Intelligence and Cognitive Science Conference (pp. 119-125). Cincinnati, Ohio, USA: Omnipress.

Filieri, A., Ghezzi, C., & Tamburrelli, G. (2012). A Formal Approach to Adaptive Software: Continuous Assurance of Non-Functional Requirements. Formal Aspects of Computing , 24 (2), 163–186.

Gebhardt, M., & Kaske, A. (2011, August). Tools and Methods for Validation and Verification as Requested by ISO26262. Softwaretechnik-Trends, 31 (3), Online. Germany: Gesellschaft für Informatik-GI (German Informatics Society).

Harman, M. (2007). Search Based Software Engineering for Program Comprehension. 15th International Conference on Program Comprehension (ICPC 07) (pp. 3–13). Banff-Canada: IEEE Computer Society Press.

Harman, M. (2012). The Role of Artificial Intelligence in Software Engineering. First International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2012 (pp. 1-6). Zurich: IEEE.

Harman, M., Krinke, J., Ren, J., & Yoo, S. (2009). Search Based Data Sensitivity Analysis Applied to Requirement Engineering. ACM Genetic and Evolutionary Computation Conference (GECCO 2009), (pp. 1681–1688). Montreal-Canada.

Imam, A. T., Rousan, T., & Aljawarneh, S. (2014). An Expert Code Generator using Rule-Based and Frames Knowledge Representation Techniques. 5th International Conference on Information and Communication Systems (ICICS 2014) (pp. 1-6). Irbid, Jordan: IEEE.

Kitzelmann, E. (2010). Inductive Programming: A Survey of Program Synthesis Techniques. In U. Schmid, E. Kitzelmann, & R. Plasmeijer, Approaches and Applications of Inductive Programming (LNCS) (Vol. 5812, pp. 50-73). Springer Berlin Heidelberg.

Krogmann, K., Kuperberg, M., & Reussner, R. (2010). Using Genetic Search for Reverse Engineering of Parametric Behaviour Models for Performance Prediction. IEEE Transactions on Software Engineering , 36 (6), 865–877.

Linderman, M. D., Collins, J. D., Wang, H., & Meng, T. H. (2008). Merge: A Programming Model for Heterogeneous Multi-core Systems. 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (pp. 287-296). Seattle, WA, USA: ACM.

Meziane, F., & Vadera, S. (2012). Artificial Intelligence in Software Engineering: Current Developments and Future Prospects. In Machine Learning: Concepts, Methodologies, Tools and Applications (pp. 1215-1236). Hershey, PA: Information Science Reference- IGI Global.

Ormandjieva, O., Hussain, I., & Kosseim, L. (2007). Toward A Text Classification System for the Quality Assessment of Software Requirements written in Natural Language. 4th International Workshop on Software Quality Assurance (SOQUA '07) (pp. 39-45). NY-USA: ACM.

Poulding, S. M., & Clark, J. A. (2010). Efficient Software Verification: Statistical Testing using Automated Search. IEEE Transactions on Software Engineerig , 36 (6), 763–777.

Pressman, R. S. (2010). Software Engineering: A Practitioner's Approach, 7/e. McGraw-Hill Education.

Ramachandran, M. (2008). Software Components: Guidelines and Applications. NY, USA: Nova Science Publishers, Inc.

Rodriguez, D., Ruiz, R., Riquelme-Santos, J. C., & Harrison, R. (2011). Subgroup Discovery for Defect Prediction. 3rd International Symposium on Search Based Software Engineering (SSBSE). 6956, pp. 269–270. Szeged- Hungary: Springer.

Sommerville, I. (2010). Software Engineering (9th Edition). Addison-Wesley.

Staunton, J., & Clark, J. A. (2011). Finding Short Counter Examples in Promela Models using Estimation of Distribution Algorithms. In N. K. Lanzi (Ed.), 13th Annual Genetic and Evolutionary Computation Conference (GECCO 2011), (pp. 1923-1930). Dublin- Ireland: ACM.

Umber, A., Bajwa, I. S., & Naeem, M. A. (2011). NL-Based Automated Software Requirements Elicitation and Specification (Vol. 191). (A. Abraham, J. L. Mauri, J. F. Buford, J. Suzuki, & S. M. Thampi, Eds.) Kochi, Kerala, India: Springer Berlin Heidelberg.

V.S.Bagad. (2009). Electronics Product Design. Technical Publications.

W.B., L., & Harman, M. (2010). Evolving a CUDA Kernel from an nVidia Template. IEEE Congress on Evolutionary Computation, (pp. 1–8). Barcelona-Espan.

Weimer, W., Nguyen, T. V., Goues, C. L., & Forre, S. (2009). Automatically Finding Patches using Genetic Programming. International Conference on Software Engineering (ICSE 2009), (pp. 364–374). Vancouver, Canada.

Yoo, S., Harman, M., & Ur, S. (2011). Highly Scalable Multi-Objective Test Suite Minimisation using Graphics Cards. 3rd International Symposium on Search based Software Engineering (SSBSE 2011). 6956, pp. 219–236. Szeged, Hungary: Springer.

**Authors**

**Ayad T. Imam** received his Ph.D degree in computer science from De Montfort University, Leicester, UK in 2010. Currently, Dr. Ayad is an Assistant Prof. at Al-Isra University / Amman / Jordan. Dr. Ayad has a number of published papers in various Computer Science and Software Engineering topics. Dr. Ayad is a reviewer in a number of journals and conferences on Computer and Information related areas.

**Ayman J. Alnsour** received his Ph.D degree in Computer Engineering in 1995. Prof. Ayman has a number of published papers in various areas of Computer Science and Engineering. Prof. Ayman is a chairman of a number of conferences as well as a reviewer in a number of journals and conferences. Prof. Ayman is a member of IEEE, ACM, Computing Society (USA), and Jordan Computer Society (Jordan)

**Aysh Al-Hroob** received his Ph.D degree in Software Engineering from Bradford University, UK in 2010. Currently, Dr. Aysh is an acting head of the SE department / Faculty of IT / Al-Isra University / Amman / Jordan. Dr. Aysh has a number of published papers in various topics of Software Engineering. Dr. Aysh is a reviewer in a number of journals and conferences of software engineering field.

The IISTE is a pioneer in the Open-Access hosting service and academic event management. The aim of the firm is Accelerating Global Knowledge Sharing.

More information about the firm can be found on the homepage:
http://www.iiste.org

## CALL FOR JOURNAL PAPERS

There are more than 30 peer-reviewed academic journals hosted under the hosting platform.

**Prospective authors of journals can find the submission instruction on the following page:** http://www.iiste.org/journals/ All the journals articles are available online to the readers all over the world without financial, legal, or technical barriers other than those inseparable from gaining access to the internet itself. Paper version of the journals is also available upon request of readers and authors.

## MORE RESOURCES

Book publication information: http://www.iiste.org/book/

Academic conference: http://www.iiste.org/conference/upcoming-conferences-call-for-paper/

## IISTE Knowledge Sharing Partners

EBSCO, Index Copernicus, Ulrich's Periodicals Directory, JournalTOCS, PKP Open Archives Harvester, Bielefeld Academic Search Engine, Elektronische Zeitschriftenbibliothek EZB, Open J-Gate, OCLC WorldCat, Universe Digtial Library , NewJour, Google Scholar