UNIVERSITY OF GOTHENBURG

A Systematic Review of Automated Software Engineering

Gegentana

Master of Science Thesis in Program Software Engineering and Management

Report No. 2011:066
ISSN:1651-4769

University of Gothenburg
Department of Applied Information Technology
Göteborg, Sweden, July 2011

A Systematic Review of Automated Software Engineering

Gegentanta, June 2011.

Supervisor: Dr. Robert Feldt

Chlamers University of Technology
Department of Computer Science and Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Applied Information Technology
Göteborg, Sweden June 2011

Acknowledgement

This thesis project will not be possibly finished, without supports from many people, and I would like to thank them all here.

First of all, I would like to deeply thank my supervisor, Prof. Dr. Robert Feldt, who guided and encouraged me to strategically conduct this Systematic Review in the entire processes of the thesis project, provided me with a lot of valuable guidance, brilliant suggestions and constructive thoughts throughout this thesis. I am grateful to him for his enormous support and valuable comments.

Secondly, I would like to thank Jie Lin and Yi Xu, who are master students from Master program in Software Engineering and Technology of Chalmers University of Technology. They helped me to prototype the pilot data extraction in this study.

Thirdly, I would like to thank Qianxi Cao, who is working in Chalmers Open Communication Studio (CHOCS). She provided me with many helps on correcting my language issues in the thesis paper, which improved both the readability and correctness of this thesis.

Finally, I would like to thank my dear family who constantly encourage and support me to pursue my study in Sweden.

Table of Contents

A Systematic Review of Automated Software Engineering

GEGENTANA
Department of Applied IT
IT University of Gothenburg/ Chalmers University of Technology
Gothenburg, Sweden
Tata619@hotmail.com

Abstract

Context: *Automated software engineering is becoming an increasingly important part of Software Engineering. Both fully and partially automated approaches and methods can improve the productivity and quality of software development.*

Objective: *The goal of this study is to identify the current status of the automated software engineering field based on publications in the years 1999 to 2009. The results should be valuable for people who are assessing which automated approaches and methods to implement in their software development.*

Method: *The method used in this study is a systematic review. It is a well-defined method, which can be used to identify, analyze, synthesize, evaluate and compare available and relevant articles on a specific research topic. The attributes and characteristics to extract for each automated approach/method was based on a partial literature in the field and related software engineering fields concerned with automation of human activities.*

Results: *From the 122 published articles selected in the final stage of paper screening and filtering we found 127 automated approaches distributed on 9 areas of Software Engineering. We also provide analysis of these approaches based on the years of publication, automation level of the proposed automated approaches, human activity required for using each approach and their types.*

Conclusion: *Software design was the most prevalent area for research in automated software engineering from 1999 to 2009. Furthermore, 39.4% of automated approaches were deemed as having a low automation level, indicating that much manual work was still left for utilizing the technique. Meanwhile, only a total of 22 required human activities were mentioned for the 127 automated approaches, which indicates that researchers focus on the automation approaches themselves but neglect to consider the level of automation they supply as well as the human activities that are still needed when using them.*

KEYWORDS: systematic review, automated software engineering, Required Human Activity,automated approaches.

Acronyms: ASE=automated software engineering, SR=Software Requirements, SD=Software Design, SM=Software Maintenance, SEP=Software Engineering Process, ST=Software Testing, SC=Software Construction, SQ=Software Quality,SCM=Software Configuration Management, SEM=Software Engineering Management.

1. INTRODUCTION

In the computer and information era, Computer-based applications become an essential part of human life [1]. Software is the core of Computer-based application, therefore became a critical part of science research.

Software engineering has been the subject of a wide range of discussion over the last decade. It is about developing, maintaining and managing high-quality software systems in a cost-effective and predictable way. The studies within software engineering concern the real-world phenomena of it, which include: the development of new, or modification of existing, technologies (process, models, methods, techniques, tools or language) or support their activities, and the evaluation and comparison of the effect of using such technology in the often very complex interaction of individuals, teams, projects and organizations, and various of task and software system [2]. As the core of software engineering, it concerns the development and evolution of large and complex software-intensive system. Meanwhile, it is a concern for the production of quality software that meets reasonable requirements of its performance, reliability, maintainability, and cost [3].

Nowadays, the size of software becomes increasingly larger; therefore, massive software production and subtle software maintenance are needed. Consequently, the cost of advancing the relevant complimentary work is rising dramatically. As a result, the demand for high qualitative and reliable software while keeping reasonable cost of human resource is needed. As a consequence of the increasing complexity and the need for better-designed and user-friendly software products, more and more efforts need to be made to software development. A vast amount of work is involved in the different fields of software engineering such as software requirements, design, construction, testing and quality, etc.

Meanwhile, researchers focus on developing the techniques, which they use to use manually and they attach an importance to enabling the automation of the techniques, with the goal of partially or fully automating the activities in software engineering that can significantly increase both the software quality and productivity.

Without automated approaches, it is very labor intensive and time consuming when developing large software.

Automation stops people from doing redundant and repetitive work. For software developers, automated software engineering (ASE) has already become an important element of the standard development of software. Automated approaches increase productivity of developers and improve the reliability of new software [4].

When applying automated approaches into the real world if the automated approaches cannot afford full-automation, Required Human Activity are needed.

Systematic research synthesis has been wildly applied to medicine and health care field, which had been proved useful for helping the researchers to summarize large amount of information identifying gaps, beneficial and harmful interventions. Thereby the researchers can get the clear reporting and evidence to conduct the future planning in health care field. The successful use of the SR in different fields can adequately prove that it is an effective and efficient solution to performing the overview on specific topics. A systematic review is a well-defined method to identify, analyze, synthesize, evaluate, and to compare all available literature works relevant to a specific research topic [18]. It is a critical study work for the researchers to get vivid understanding about the status quo in the relevant field through this SR.

Systematic reviews have been gaining popularity in software engineering since Kitchenham published the paper in 2004 [5], with reviews of recently published articles on diverse topics, which includes: requirement elicitation [6], agile software development [7], etc. It is important for software engineering practitioners and researchers to constantly conduct research, since it is impossible for individuals to get familiar with some specific fields.

In this study, in an effort to summarize the ASE to enable industry practitioners to better assess that automated approaches, such as, tools, techniques and methods etc, are worth implementing in their organizations. All the 122 selected articles involved in this study are mapped into main taxonomy through systematic review.

Rest of this study is organized as follows. Section 2 introduces the previous researches both in ASE and the systematic reviews, as well as the definition and classification of automation. Section 3 presents a detailed introduction of research methodology used in this study. Furthermore, it presents how systematic review works, which includes its planning and conducting process. The results found via classification of the data extracted in the systematic review are presented in the Section 4. In Section 5, discussions with respect to the results are presented. The conclusion is given in the Section 6.

1.1. Research Questions

Research questions to be addressed by systematic review are:

- How many types of automated approaches were investigated in automated software engineering from 1999 to 2009?
- What were the types of automated approaches investigated in automated software engineering from 1999 to 2009?
- How many fields in software engineering were considered in the researches during 1999 and 2009?
- Which software engineering field, sub-category and sub-class was more widely studied by the researchers?
- Which level of automated approach was more popular to the researchers, autonomous level, informing level or decision support level?
- What kinds of Required Human Activity were needed when human applying the automated approaches?

1.2. Limitation

This study aims to consider and analyze the articles published within ASE field between 1999 and 2009. All the articles should be written for introducing new automated approaches used in this area with the purpose of improving the quality or productivity of software. Case study and papers that only include descriptions and researches of how to use the automated approaches are excluded from this study because of the consideration of the purpose of this study is researches on newly proposed automated approaches, rather than the further use of them.

Therefore, the selection of the papers and corresponding criteria, on which the selection is based, only focuses on the original published papers that have proposed new automated approaches used in ASE field. The detailed description of criteria is presented in Section 3.1.4.3.

2. RELATED WORK

2.1. Previous Research in ASE

Nowadays, automated approaches have been applied in many areas in software engineering field, such as requirements definition [8], architecture [9], implementation, modeling [10], testing and quality assurance [11], verification and validation [12]. Many articles have been published and widely mentioned previously finished tasks in the ASE. These researches and published paper are used in the source paper in this study.

These researches focused on how to apply the semi-automated and fully automated approaches into the development of software engineering to improve the quality and productivity of the software. In this study, the automation level of automated approaches will be considered as well.

2.2. Previous Use of Systematic Review

Nowadays, systematic review has been broadly used in researching psychology sciences, statistical sciences, education, industrial/organizational psychology, medicine, health sciences domain, and software engineering [13]. According to the medicine practice by using SR, Kitchenham et al [2004] evaluates the idea of Evidence-Based software engineering and proposes a guideline for systematic review that is appropriate for software engineering researchers [14]. Based on these guidelines, a lot of SRs had been done in the software engineering afterwards.

2.3. What is Automation

Automation has been used in many different ways. Oxford English Dictionary (1989) defines automation as

- Automatic control of the manufacture of a product through a number of successive stages;
- the application of automatic control to any branch of industry or science;
- by extension, the use of electronic or mechanical devices to replace human labor.

The purpose of applying automation (partially or fully) in different control systems, technology and process is to reduce the needs of human intervention. In controlling process, operators can use the automated devices to implement.

2.4. Classifying Automation in General

Automation can be applied to four broad classes of functions: information acquisition; information analysis; decision and action selection; and action implementation. It refers to full or partial replacement of a function previously carried out by the human operator, which implies that automation is not all or none, but can vary across a continuum of levels, from the lowest level of fully manual performance to the highest level of full automation [19].

Table 1 shows a 10-point scale, with higher levels representing increased autonomy of computer over human action, based on a previously proposed scale [19].

Table 1. Levels of Automation

High	10.The computer decides everything and acts autonomously, ignoring the human.
	9. The computer informs the human only if it decides to.
	8. The computer informs the human only if asked.
	7. The computer executes automatically, and then necessarily informs the human.
	6. The computer allows the human a restricted time to veto before automatic execution.
	5. The computer executes that suggestion if the human approves.
	4. The computer suggests one alternative.
	3. The computer narrows the selection down to a few.
	2. The computer offers a complete set of decision/ action alternatives.
Low	1. The computer offers no assistance: human must take all decisions and actions.

In order to make the description easier to understand, these 10-point automation scales have been re-divided and simplified into four levels, which is used in this project. (Detailed description can be found in Section 3.1.3).

3. RESEARCH METHODOLOGY

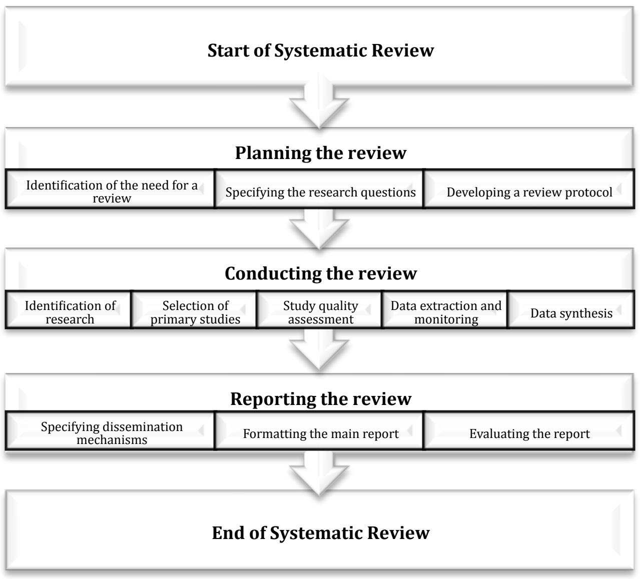
Systematic Literature Review (also referred to systematic review) is a form of secondary study that uses a well-defined methodology to identify, analyze and interpret all available evidence related to a specific research question in a way that is unbiased and (to a degree) repeatable [15].

The most distinctive point that a Systematic Literature Review differs from conventional literature review is that SR holds more scientific value in terms of credibility, systematization, and preciseness. This is the main reason that a systematic review has been undertaken. A search strategy must be predefined, and be in accordance with during the conducting the SR.

According to the advantages and disadvantages of systematic review described by Kitchenham (2004) [14], SR can be a effective method to help the researcher to get the information about the effects of some phenomenon across a wide range of settings and empirical methods with it is less likely the results are biased.

The systematic review conducted in this study followed the procedure described by Kitchnham [15]; designed three phases to conduct this SR study, which are described in the Figure 1 [15].

Figure 1. Stages of systematic review


Start of Systematic Review
Planning the review
Identification of the need for a review
Specifying the research questions
Developing a review protocol
Conducting the review
Identification of research
Selection of primary studies
Study quality assessment
Data extraction and monitoring
Data synthesis
Reporting the review
Specifying dissemination mechanisms
Formatting the main report
Evaluating the report
End of Systematic Review

3.1. Planning the review.

3.1.1. Reasons for Performing the SR in this Study

To begin with, bearing the purposes of implementing the following activities shown below, SR is selected to be the only methodology in this study.

- To summarize the existing empirical evidences of benefits and limitations in ASE.
- To identify how many activities are adopted in current research in order to help the researchers do the further research in ASE.
- To execute the SR on existing works in ASE, based on the predefined search strategy, which can reduce the biases of hypothesis from the researcher.

In order to avoid meaningless and unnecessary duplicated work in this study, a pilot search was conducted, before the design of the SR for ASE. Three databases, which are known as Inspec, IEEE and ACM digital libraries, have been searched with, to check whether an SR of this topic has been done or not.

The following search string has been used to search with subject, title and abstracts in the databases.

((“systematic review” OR “systematic overview”) AND (“automated software engineering” OR ASE))

According to the search, there was no relevant result found, which indicated that there was no SR being done in ASE field. Therefore, a SR of ASE needs to be undertaken, and won’t duplicate with any previous study.

3.1.2. Defining the Research Questions

Research questions are formed based on scanning relevant key articles in ASE, and detailed description can be found in the Table 2.

Table 2. Research Questions

Research Questions	Aim
RQ1: How many types of automated approaches were investigated in automated software engineering from 1999 to 2009 and what were they?	To summarize the types of automated approaches found in the articles from 1999 to 2009.
RQ2: How many fields in software engineering were considered in the researches during 1999 and 2009?	To summarize automated approaches based on the software engineering fields mentioned in the articles during 1999 and 2009.
RQ3: Which software engineering field, sub-category and sub-class were more popular to the researchers?	To summarize and classify the automated approach into the software engineering field, sub-category and sub-class, in order to see the popularity distributed in software engineering field.
RQ4: Which level of automation was more popular to the researchers?	To summarize the popularity of the level that the automated approaches were investigated in.
RQ5: What kind of Human Required Activity are needed when human applying the automated approaches?	To summarize the need of Human Required Activity, when applying different the automated approaches.

3.1.3. Defining Automation Levels

Ten levels of automation described in Section 2.4, are re-divided into four automation levels, which are applied in this study. Since automation level in this study is just one of the features that considered, the classification of them doesn’t have to be too detailed. Data will be generated based on these four new automation levels. The re-divided automation levels are displayed in Table 3.

Table 3. Automation Level

Levels	Level Description
LA-Autonomous	10.The computer decides everything and acts autonomously, ignoring the human 9. informs the human only if it, the computer decides to
LB-Informing	8. informs the human only if asked, or 7. executes automatically, then necessarily informs the human, and 6. allows the human a restricted time to veto before automatic execution, or
LC-Decision Support	5. execute that suggestion if the human approves, or 4. suggests one alternative 3. narrows the selection down to a few, or 2. The computer offers a complete set of decision/ action alternative, or
LD-No Automation	1. The computer offers no assistance: human must take all decisions and actions.

The highest level (LA) represents that computer can implement complete automatic activity, or inform the human if necessary. The computer controls whether the human interventions are needed or not.

The second level (LB) is middle level, which refers to a partial automatic approach by the computer. Human needs to make the decision based on the computer decision alternatives. It differs from LA in the point that human decision is more important in this level.

The third level (LC) represents that human completely control the decision-making and action. The only work that computer needs to finish is information collection.

The fourth level (LD) refers to non-automation approach. It is excluded in the study because of the scope of this study is within ASE field, which means automation has to be taken into consideration.

3.1.4. Developing a Review Protocol

The purpose of a pre-defined protocol is to guide researchers during the entire part of “conducting of SR” and further reduce the possibility of the emergence of researcher bias.

3.1.4.1. Searching Strategies

The correct search strategies are very important to guarantee the success of primary study identification. The aim of search strategies is to define the databases, where the primary resources are searched, and how to search them.

Four bibliographic databases list below were chosen as the target databases in this study:

- IEEE Xplore
- ACM Digital library
- Inspec
- Scopus

3.1.4.2. Exploring Search String

In this study, the search string should be composed by selected keywords. The following parts describe the process of making search string.

Step 1. Gathering Keywords

The purpose of summarizing keywords was to collect relevant terms and synonyms used in the ASE field. The combination of (“automated software engineering” and (productivity or quality)) was used as the initial search term in two bibliographic databases (Inspec and ACM Digital library). The reason of adopting Inspec database instead of the IEEE Xplore database is that IEEE Xplore is already covered by Inspec database since 1994 [19]. Therefore, These two databases were enough for me to conduct a pilot searching and gathering original keywords. 53 papers were found from the Inspec database, which included the Journal articles and Conference Proceedings. Afterwards, 93 articles were found in ACM Digital library with the same restrictions as the previous searching in Inspec. By scanning the title and abstract, 64 related papers were chosen from ACM Digital database finally. In total, 117 papers were studied to select keywords for this pilot search.

Step 2. Forming Checklist of Keywords by Frequency (Appendix A).

The contribution of this checklist was to sort out the keywords by frequency. (Frequency here means the times of keywords that occur in the amount of journal articles and conference proceedings.) The keywords were listed in the Table A in Appendix A, with the frequency from high to low. The percentages were calculated and listed as well. Two tables that used to summarize the results are displayed in Appendix A.

Step 3. Extraction of Keywords

The keywords whose percentages were higher than 13% in Inspec and 8% in ACM were chosen respectively. The reason for applying different standards for these two databases separately was to avoid the biases in selecting key terms, and to ensure the key terms chosen were as comprehensive as possible. From the research questions, 21 general keywords for the further search were extracted. All the keywords relevant to detailed concepts in specific research fields in ASE were abandoned because the research focused on studying in a broader sense of ASE area. Table 4 lists the entire 21 general keywords.

Table 4

Key terms
Automat*
software, system, program*
process, engineering, development
design, model*, comput*, generat*, test*, verification, requirements, analysis, tool*, management, pattern
quality, safety, productivity.

Step 4. Verification of Keywords

In order to make the key terms rigorous, the different combination of terms needed to be explored, in order to check how many hits for each search string. The “automat*” with different compounding was used as the search string to search in Inspec database. (e.g., automat* program, automat* system). From the results, there were 14 keywords out of 21 key terms mentioned in most of those articles. The result is displayed in Table 5.

Table 5

Compounding	Key terms
1. “automat* program*” 2. “automat* system” 3. “automat* software” 4. “automat* process” 5.“automat*software development”	design, model*, comput*, generat*, test*, verification, requirements, analysis, tool*, management, pattern, quality, safety, productivity.

Step 5. Forming Search String

Finally, these key terms listed in Table 5 were used to form search string with logical operator (AND, OR). The search string is given in Table 6.

Table 6

Search string
Automat* AND (software OR system OR program*) AND (engineering OR process OR development) AND (design OR model* OR comput* OR generat* OR test* OR tool* OR verification OR requirements OR analysis OR management OR pattern) AND (quality OR safety OR productivity)

3.1.4.3. Study Selection Criteria

The articles should be selected based on inclusion and exclusion criteria by considering the title and reading abstract. As long as the contents of articles are related to any research question of this SR study in ASE, it has been considered. The inclusion and exclusion criteria are given below:

- The selected articles should be published works.
- The articles should be journal articles.
- The articles should be published between 1999 and 2009.
- The context of articles should be within automated software engineering field, i.e. the key purpose of the articles and proposed automated approach should be improving the development of software.
- The studies, which were in both academic and industry environments, should be considered and included.
- The issues of articles need to be related to any of the research question listed in Table 2.
- The language of articles should be English.
- The articles should be available in full text.
- General discussions, descriptions, experiments, case studies and reviews of techniques, tools and methods without empirical evaluation should be excluded. The detailed descriptions of what automated approaches are and how customer can use specific automated approaches in software engineering field should be considered.
- The articles should directly describe a method /technique /tool /process that automates (or automates more/ to a higher degree) a software development activity which was previously done manually.

3.1.4.4. Study Selection Procedure

In order to identify the correctness of the selection work, and check the correctness of criteria, a pilot selection was executed before conducting of the main selection work, with the purpose of measuring the consistency of our understanding about the study scope and selection criteria to ensure the quality of the selection.

30 papers out of the searching result were randomly selected as samples of the pilot selection. Then, selection works were conducted on these sample articles separately by the authors, following the selection criteria pre-defined in Section 3.1.4.3.

After finishing the individual selection, the results were compared, and the amount of selected papers that matched was checked. The result from individuals showed there were 14 of them unmatched, which was a very high ratio.

Some negotiations were made on correcting selection criteria and few changes were made in accordance to understanding of the criteria based on our agreements.

In order to check the correctness of the negotiated understandings, another 14 papers were randomly chosen again. The result showed that there were only three articles unmatched, which was acceptable. After these two pilot selection works conducted sequentially, we finally consolidated the standard understanding of the study selection criteria.

3.1.4.5. Study Quality Assessment Criteria

Articles searched should follow the inclusion criteria (Secion 3.1.4.3.) by reading the full text. The purpose of checklist is to assess the quality of selected papers. The detailed description is given in Table 7.

Table 7. Quality assessment criteria

Criteria	Yes/No/Uncertain
Is the abstract relevant to ASE field?	
Does the introduction clearly state the research question and the result of the automated approaches?	
Is the method innovative or not?	
Is the method used in the research paper appropriate?	
Is the content adequate to support the research?	
Are the validity threat mentioned in the research?	
Are the results explicit stated?	
Is the conclusion appropriately drawn?	

3.1.4.6. Data Extraction Strategy

The extraction work was conducted through a full text reading, and the data of the articles were mapped into the main taxonomy in the Table B (Appendix B), which were used for the further mapping work of articles into specific field and analysis work.

3.2. Conducting the Review

3.2.1. Identification of Research

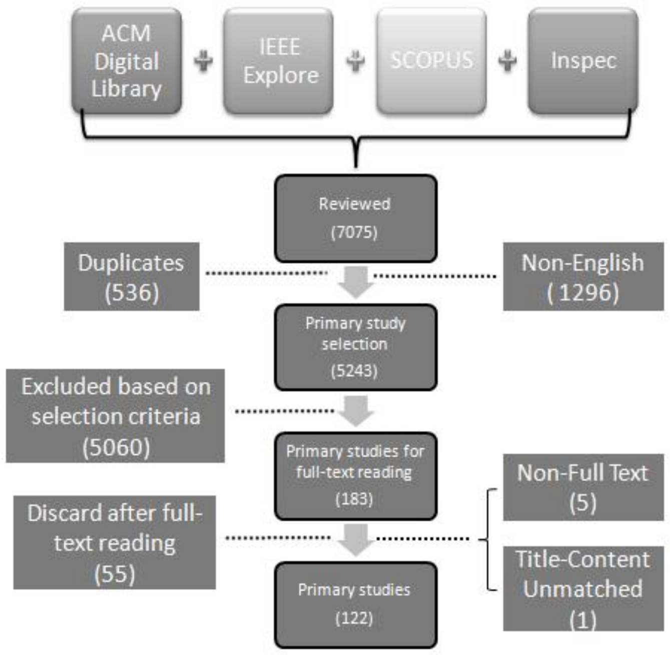
SR was conducted to find as many primary studies as possible, which were relevant to my research questions in ASE field by following the unbiased search strategy described in the review protocol in previous section. A big amount of efforts were made to search the articles from 4 bibliographical databases. Meanwhile, Zotero [17] was used in this research in order to help the author to manage the large amount of research papers, which can reduce the time spent on organizing the objects. This tool can support adding notes, tagging, and personal metadata through the in-browser interface. It can filter some articles automatically by checking the tags, which can avoid unnecessary duplication of articles.

3.2.2. Selection of Primary Studies

The purpose of performing paper selection was to identify the relevant papers, which can be relevant to the subject in agreed scope and suffice the objective of SR as well.

The search string in Table 6 was used to find the related articles in this study. A total of 7075 articles were found from 4 selected bibliographic databases. The steps of filtering papers are described fully in Figure 2.

Figure 2. Study Selection


ACM Digital Library
IEEE Explore
SCOPUS
Inspec
Reviewed (7075)
Duplicates (536)
Non-English (1296)
Primary study selection (5243)
Excluded based on selection criteria (5060)
Primary studies for full-text reading (183)
Non-Full Text (5)
Discard after full-text reading (55)
Title-Content Unmatched (1)
Primary studies (122)

3.2.3. Study Selection Procedure

7075 papers were found based on the searching string of this study, the papers, which were duplicated and non-English version were excluded. The usage of Zotero tool [17] to store the papers made it easier to remove the papers. In the primary study selection, 5243 papers were collected. Then screening of papers was conducted abided the exclusion selection criteria. Afterwards, 5060 papers were excluded according to these study criteria. Meanwhile, 183 papers were kept after this exclusion process. 183 publish papers were downloaded successfully. These papers had been checked based on using the quality assessment criteria table, which were expressed in Table 7. During reading the sample articles, 6 papers were excluded, which included 5 non-full texts and 1 inconsistence between content and title. By reading the details of the full-text papers, 55 papers, which were not relevant to this study, according to the quality assessment criteria, were found, during the study quality assessment phase, which was described in Section 3.2.4. Finally, 122 papers were extracted as the data used in this study, via data extraction procedure. A list of the selected papers was ordered according to the years they were published and displayed in Appendix D (Table L), whose number start with 'P', referring to Paper.

3.2.4. Study Quality Assessment

In this stage, the pilot study was assessed through the quality criteria checklist (Table 7). It was found that some studies were not well organized enough to interpret new automated approaches. Such as case study, survey empirical study, which no innovation automated approaches is proposed in the research.

3.2.5. Data Extraction

Data extraction work was conducted based on the aspects that described in Table B, given in the Appendix B. The entire 122 papers were read thoroughly, with full-text. And the detailed information in the articles used in data synthesis phase, was mapped into main taxonomy.

3.2.5.1. Generating Main Taxonomy

Main taxonomy was generated according to the research questions made at the beginning of this study. It was used to extract the data about automated approaches proposed in 122 selected papers. Since the scope of this study was within software engineering field, when generating the taxonomy, all the aspects of entire software engineering should be considered. Different sub-category and sub-class were used to determine the location of automated activities and corresponding techniques.

Table B (Appendix B) is the main taxonomy used for mapping the entire study with 122 papers. It is a systematic detailed description of all the information collected from the data extraction phase. There are 9 tables illustrated in taxonomy along with the main taxonomy (Table 10 to 16 in Section 4, and Table C&D in Appendix B and Section 4). They were used to further address the studies based on the sub-category and sub-class that the automated activities and the approaches belonged to. The reason to sort the tables in this sequence was due to the consideration of facilitating the readers from different fields in software engineering as much as possible. They had different points of emphasis and may search the information they need from the software engineering field they concern. In that case, the readers can go through the specific table from those 9 tables, from which they may find interesting and useful, as well as to find the detailed information from the main taxonomy based on the categories described in Table B, (Appendix B), in which the all the aspects of the paper explained.

3.2.5.2. Description of Main Taxonomy Table

Main taxonomy table was used to collect and classify data provided by all the articles studied in the research. Paper number was used to determine the number of the automated approaches explained in corresponding papers, for example, P1 refers to automated approach in paper 1. In case there were more than one automated approaches proposed in one paper, the combination of paper number and automated approach number was used to refer the automated approach. When more automated approaches described in one paper, it can be displayed in the form of P1-1 and P1-2. More detailed examples can be found in Section 4.1. This main taxonomy table is exhibited in Appendix B (Table 8).

Data were sorted based on the years when the papers were published from 1999 to 2009. Each article was reviewed and mapped with the following four aspects:

Article:

It includes the number of papers, and the published year of them.

Automated Approach:

When explaining the automated approach the paper mentioned, seven different categories were used to address data, including:

- AA NO.: automated approach number.
- Activity: a short description of relevant activity.
- Approach: automated approaches used to perform this automated activity.
- Relevant SE Area: which software engineering automated approach belongs to.
- Automation Level: which automation level automated approach belongs to.

Required Human Activity:

- RA NO.--Required Human Activity number.
- Activity--detailed explanation of the Required Human Activity, which aims at achieving the automated activity.
- Relevant SE area--which software engineering area human activity belongs to.

Setup Cost:

- Type--what kinds of type human needed before performing the automation activity.
- Effort--which level (Low, Medium, High) setup cost belongs to.

3.2.5.3. Mapping the Existing Literature into Main Taxonomy

3.2.5.3.1. Prototype of Data Extraction

In order to guarantee the correctness of the data extraction phase, two Master students were required to affiliate to validate, both of who were studying in software engineering fields as well. A pilot data extraction was exerted before performing the entire extraction work in the thesis project. Extracting the data into main taxonomy by reviewing 5 papers randomly chosen from 122 papers separately. Through the discussion of tiny differences between the results, the further explicit extraction criteria were formed.

3.2.5.3.2. Examples of Data Extraction

For mapping the correct data from relevant articles into the main taxonomy, a pre-defined description about how data varies from each other needs to be considered. Three papers with top citation numbers as examples are described below.

127 papers were searched in Google Scholar, aimed to judge the citation numbers of them. Through performing an analysis on the numbers of citation, 3 papers found with top citation number (P3 1999, P50&P56 2005). Names of these papers can be found in the Appendix D. The cited numbers of them were 94, 296 and 90 respectively.

The reasons why P3 was the second cited paper can be divided into two parts: firstly, after 10-years' publication, the research paper is still popular to study on because of its valuable topic; secondly, the concepts in the paper touched upon the fundamental theories, which had been considered important for today's research in this field. An interesting point was about P50, even it was published in the middle of period of this study considered, still got high amount of citation.

In terms to extracting the name of automated approach, the automated activity it can perform, the relevant field it belongs to, which can be easily and clearly found in the literatures, SWEBOK was used as a handbook to pilot the work on specifying the software engineering fields.

In contrary, how to classify the automation level that automated approach belongs to be flexible. Three different automation levels should be followed. Description of how to identify the automation levels of the proposed automated approach (the numbers refer to relevant automated approaches are illustrated in Section 4.1) in example papers are showing as blow:

- P3-1, an automated approach, was a framework, which comprised by Model-Checking and Abstract Interpretation. Abstract model checking was used in automated analysis of software. The researchers proposed two improvements on applying abstract modeling checking to infinite abstract transition systems. This activity belonged to the sub-class of model validation of Software Requirement field. This automated framework can perform automated information analysis, which could be classified into Informing Automation Level (LB). Since it could inform human only if needed (invalid model appearing). The researchers didn't explicitly mention any Required Human Activity.

- P50, an automated approach, was Dynamic software-updating framework. This automated activity was automatic generation of patch files, which used in Software Maintenance field. It exerted fully automated date acquisition, analysis, decision-support and action implementation. Therefore, this automated activity belonged to Autonomous Automation Level (LA).
  Meanwhile, Required Human Activity was mentioned in this paper. Programmers need to fill in the parts of state transformer and stub function. The level of human intervention belonged to high level, which represented that this automated approach (Dynamic software-updating framework) could not be exerted without programmers filling work. After human intervention, this automated approach would exert fully automated activity, which belonged to Autonomous Automation Level (LA).

- P55, an automated approach, was XML (Extensible Markup language) based on WSAMI (Web services for ambient intelligence). In this paper, researchers introduced XML-based WSAM declarative language and associated SOAP-based WSAMI middleware, which could be used in development of ambient intelligence system. It could automatically perform dynamically retrieving instances of services and further achieve dynamic composition of applications, according to environment. This automated activity belonged to sub-class of construction language in Software Construction field. This automated approach could exert fully automated activity, which belonged to Autonomous Automation Level (LA).

3.2.6. Data Synthesis

According to the statement that Brereton et al. mentioned in the article [21], which described the way of applying systematic review process within software engineering domain, the nature of systematic review is on qualitative issue. Therefore qualitative data synthesis method is more appropriate to be used in this study. Based on the explanation that George W. Noblit and R. Dwight Hare provided in their book, there were three types of the data synthesis methods: Reciprocal Translations as Syntheses, Refutational Synthesis, and Line-of-Argument Synthesis [22]. In this study, the last method was used to carry out the synthesis of the extracted data, which allowed researchers to analyze individual studies as well as group them due to the similarities that were repeatedly compared among studies. Then the analysis of the group of related studies would be accomplished as a whole. The results of the data synthesis phase are detailed stated in next section.

4. Results

Findings of this thesis project are presented in this part, based on the main taxonomy generated and the existing literatures mapped in it during data extraction phase. Four main issues were considered, when going through this study: the first one was Field from the main taxonomy, which included the analysis of 9 different software engineering fields; the second one was Automation Level from main taxonomy, in which the detailed analysis of automated approaches based on three different automation levels that the automated approaches belonged to were explained; the third one was Required Human Activity from main taxonomy, in which the detailed description of the efforts that human needs to spend when applying the automated approaches; the last one was Types of automated approach used in ASE, which stated detailed explanation of three most popular types of automated approaches out of the entire ten types of automated approaches.

4.1. Field from Main Taxonomy

In this study, apart from general analysis of the approaches found in this study based on the fields they belonged to and the years, when they were proposed, the analysis of the data extracted from each field are detailed described.

Generally, paper's number represents the index of the automated approach, but due to the case that there were more than one automated approach being proposed in one article, it's impossible to clearly refer to the approaches from the same article only by the article number. Therefore, the combination of paper number and approach number was used to represent index of automated approach.

For example, if the index of the approach is P1, it means the approach is from the paper P1, and there was only one approach in this paper. If the index of the approach is P3-2, which represents this approach is from paper P3, and it is the second approach proposed in this article.

Table 8 and 9 contributed to overview of this study based on the fields and years the approaches were published in.

The purpose of Table 8 was to summarize the number of approaches that have been found in each software engineering field, which was displayed and sorted, based on their percentages from high to low. The percentage described as how much approaches found in each field occupies in the total 127 automated approaches. This table consisted nine essential fields in software engineering field. Full references for 122 articles, where 127 automated approaches found, could be found in Appendix C.

According to the data explained in this table, it was obvious that the most popular field, where ASE researches focused on Software Design field. There were 41 automated approaches relevant to creating innovative automated approaches to improve the design of software. It accounted for 32.3% out of 127 automated approaches in total. Through exerting systematic review on 122 articles, the most popular research field in Software Design was detailed design. 8 automated approaches were proposed in this sub-class and account for 19.5%. The detailed description of this part was stated in the Section 4.1.1.

Software Requirements was the second largest field according to Table 7. In total, 27 automated approaches were found, which account for 21.3% in the 127 automated approaches. In Software Requirements field, Model Validation was the most popular sub-category been focused. 12 automated approaches were proposed, occupied 44.4% out of the total 27 automated approaches in Software Requirements field. The detailed description of data in this field was given in the Section 4.1.2.

Software Testing revealed the third highest percentage (16.5%) in this table, which included 21 automated approaches. Detailed description could be found in Section 4.1.3.

The rest 38 automated approaches found in the following four fields were quite less compared with the first three fields. Numbers and percentages of these approaches were: Software Quality field (12 and 9.4%), Software Construction field (11 and 8.7%), Software Maintenance field (9 and 7.1%), and software engineering Process field (6 and 4.7%). The detailed descriptions could be found from Section 4.1.4 to 4.1.7 respectively.

There was no automated approach found in either the Software Configuration Management field or Software Engineering Management field. Section 4.1.8 and 4.1.9 provide detailed descriptions.

Table 8. The percentage of each field in software engineering

Field	Approaches	Total	Percentage
Software Design	P1, P5, P6, P9, P11, P12, P13, P14, P16, P18, P22, P23, P25, P30, P34, P37, P38, P46, P47, P59, P62, P66, P67, P68, P69, P70, P90, P91, P97, P99, P102, P103, P112, P114, P115, P116, P120, P121, P3-2, P26-1, P26-2	41	32.3%
Software Requirements	P2, P17, P24, P35, P39, P44, P49, P51, P56, P76, P79, P84, P85, P88, P94, P98, P104, P108, P109, P111, P119, P3-1, P27-1, P27-2, P27-3, P53-1, P53-2	27	21.3%
Software Testing	P4, P20, P21, P33, P41, P42, P45, P48, P57, P60, P61, P64, P65, P75, P77, P93, P96, P101, P110, P117, P118	21	16.5%
Software Quality	P8, P15, P29, P32, P58, P74, P86, P89, P100, P105, P107, P113	12	9.4%
Software Construction	P19, P28, P54, P55, P71, P78, P80, P82, P83, P87, P122	11	8.7%
Software Maintenance	P10, P31, P36, P40, P50, P72, P92, P95, P106	9	7.1%
Software Engineering Process	P7, P43, P52, P63, P73, P81	6	4.7%
Software Configuration Management	0	0	0%

Software Engineering Management	0	0	0%

Table 9 displayed the numbers of automated approaches proposed in each software field based on the years. It was a supplemental table for Table 8. The horizontal rows were ordered by different 9 fields based on the popularities of them according to the Table 8. The vertical columns were sorted based on the years and divided into two categories under each year--number of automated approaches proposed (N) and the percentage (P) of automated approaches found in the corresponding year when they were published. The bold numbers highlighted in the table figure the highest percentage in each year separately. The reason why exerted comparing based on the number not only the percentage was that the numbers shown on the table were not necessarily enough to demonstrate the popularity of the field itself, which can avoid some bias when analyzing the data. Some of the detailed descriptions about Table 9 are displayed below:

Firstly, it was apparently seen that Software Design field was the most popular one in which the automated approaches was applied through the 8 years out of 11 in total. Especially, during the first 4 years, it occupied more than half of the amount of published articles in each year. All of this information reveals that Software Design was the most popular field, which attracted researches to explore automated approaches in ASE field.

By transversely comparing, in Software Design field, the highest number showed in 2006, which was 7. The odd point appeared in the 2007, there was no automated approach found related to Software Design field. In contrast, number of articles published in Software Construction field revealed the highest point in 2007, amount of which was 5 and percentage was 41.7%. It was strongly demonstrated that in 2007, investigation on ASE in Software Construction field attracted the most attention among researchers.

Secondly, automated approaches found in Software Requirement field appeared as the top numbers in 2003 (30%), 2005 (37.5%) and 2008 (27.2%). But it shared the top percentage with Software Design fields in two years, 2003 and 2008.

Thirdly, there was no top showing in Software Testing field. Even the articles relevant to Software Quality field appeared once as top percentage in 2000, it should be noticed that these results highly depended on the small amount of automated approaches proposed which was just 2.

Fourthly, only 9 and 6 automated approaches relevant to the Software Maintenance and Software Engineering Process fields respectively. Meanwhile, there were no articles investigating on automated approaches used in Software Configuration Management and Software Engineering Management fields.

Finally, two phenomena were found: one was that the diversity of software engineering fields increased from 1999 to 2009, much more innovative automated approaches were proposed by researchers; the other was that by comparing with the number of published articles before and after the year of 2004, the amount of proposed automated approaches was almost 3 times more than it was in the period between 1999 and 2004, which represented that there were increasingly attentions paid to the researches on ASE field from 1999 to 2009.

Table 9. The Numbers of automated approaches in each software field based on the years
N=number. P=percentage (%).

Year / **SE Field**	**1999**		**2000**		**2001**		**2002**		**2003**		**2004**		**2005**		**2006**		**2007**		**2008**		**2009**	
	N	**P**	**N**	**P**	**N**	**P**	**N**	**P**	**N**	**P**	**N**	**P**	**N**	**P**	**N**	**P**	**N**	**P**	**N**	**P**	**N**	**P**
Design	4	**50**	1	**50**	6	**60**	3	**50**	3	**30**	3	**33.3**	2	12.5	7	**36.8**	0	0	6	**27.2**	6	**46.2**
Requirements	2	25	0	0	1	10	1	16.7	3	**30**	2	22.2	6	**37.5**	0	0	4	33.3	6	**27.2**	2	15.4
Testing	1	12.5	0	0	0	0	2	33.3	0	0	2	22.2	3	18.8	6	31.6	1	8.3	3	13.7	3	23.1
Quality	0	0	1	**50**	1	10	0	0	2	20	0	0	0	0	2	10.6	1	8.3	4	18.2	1	7.7
Construction	0	0	0	0	1	10	0	0	1	10	0	0	2	12.5	1	5.3	5	**41.7**	0	0	1	7.7
Maintains	0	0	0	0	1	10	0	0	1	10	2	22.2	1	6.3	1	5.3	0	0	3	13.7	0	0
Process	1	12.5	0	0	0	0	0	0	0	0	0	0	2	12.5	2	10.6	1	8.3	0	0	0	0
Configuration Management	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Software Engineering Management	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Total	8	--	2	--	10	--	6	--	10	--	9	--	16	--	19	--	12	--	22	--	13	--

4.1.1. Software Design

The contribution of Table 10 is to make explicit classification on detailed of Software Design field. It is a detailed description of how to map entire 41 approaches, which belongs to the Software Design field into 10 Sub-categories in Software Design. This table consists of 6 aspects, which are fields in Software Design, sub-category, sub-class, Approach, Number (how many automated approaches were found in this field) and Percentage (what percentage of automated approaches occupies in the entire 41 automated approaches). The corresponding articles of each approach are displayed in Appendix D.

Detailed design is a sub-class that belongs to sub-category of Software Design Process. It was the most popular research area in the Software Design field during 1999 to 2009, since it was the highest percentage (19.5%) in the entire of table.

For demonstrating how to classify which sub-category or sub-class automated approach should belong to, an example is provided as below.

The title of article P22 was "A methodology for designing toolkits for specification level verification of interval-constrained information system requirements". In this article, authors focused on developing the basic structure of the Information System (IS) maintenance toolkits design model and detail modeling of Fault Inspecting sub-module design pattern. Researchers developed a more reusable and various IS verification algorithms and design components to achieve increasing structural quality and decreasing effort in building specific information system maintenance (ISM) toolkits. An automated approach they explored, which was Reuse-oriented UMP methodology that activity was generalized designs of Information System maintenance toolkits that maintain the requirements specification of IS.

According to the content of SWEBOK [20], automated approach P22 belongs to Detailed design sub-class of Software Design field, of which the definition is to describe the specific behavior of software components. The output of this process is a set of models and artifacts that record the major decisions that have been taken [20], which matched the corresponding activity performed by this automated approach.

Comparing with sub-class of Detailed design, Simulation and prototyping, and Static analysis were the second most popular sub-class. They belonged to sub-category of Quality Analysis and Evaluation. These two sub-classes both occupied 12.2% (5 approaches) out of 41 automated approaches.

The third most popular sub-category was Error and exception handling and fault tolerance, which accounted for 9.8% (4 approaches) out of 41 automated approaches.

The forth most popular sub-category was Architectural Structures and Viewpoints, Families of programs and frameworks, and one sub-class of Software design reviews. They all involved 3 automated approaches and accounted for 7.3% in the 41 approaches respectively.

The fifth popular sub-category was Distribution of Components, and two sub-classes of Architecture design, and Formal specification languages, which include 2, automated approaches (4.9%) separately.

The least popular sub-category was Quality, and three sub-classes of Coupling and cohesion, Behavioral patterns, and Object-oriented design measures. They only included 1 automated approach (2.4%).

Table 10. Automated approaches in Software Design field

Field	Sub-category	Sub-class	Approach	Number	Percentage
Software Design Fundamentals	Software Design Process	Architecture design	P23, P47,	2	4.9%
		Detailed design	P22, P62, P70, P91, P99, P112, P115, P120,	8	19.5%
	Enabling Techniques	Coupling and cohesion	P59,	1	2.4%
Key Issues in Software Design	Distribution of Components		P37, P68,	2	4.9%
	Error and Exception Handling and Fault Tolerance		P18, P69, P102, P121,	4	9.8%
Software Structure and Architecture	Architectural Structures and Viewpoints		P34, P90, P114,	3	7.3%
	Design Patterns	Behavioral patterns	P16,	1	2.4%
	Families of Programs and Frameworks		P6, P9, P97,	3	7.3%
Software Design Quality Analysis	Quality		P13,	1	2.4%
	Quality Analysis and Evaluation	Software design reviews	P12, P14, P25,	3	7.3%

		Static analysis	P1, P3-2, P11, P103, P116,	5	12.2%
		Simulation and prototyping	P26-1, P26-2, P30, P66, P67,	5	12.2%
		Object-oriented design measures	P46,	1	2.4%
Software Design Notations	Behavioral Descriptions (dynamic view)	Formal specification languages	P5, P38,	2	4.9%

4.1.2. Software Requirements

Table 11 details the percentage of automated approaches in Software Requirement of software engineering field. It was the second most popular research field in this study, which included 27 automated approaches out of 122 articles. This table indicates detail distribution of each automated approach in the entire Software Requirements field. It explicitly stated 9 Sub-categories of this field, which were Model validation (44.4%), Elicitation Techniques (14.8%), Software Requirements Specification (11.1%), Conceptual Modeling (7.4%), Measuring Requirement (7.4%), Requirements Sources (3.7%), Acceptance testing (3.7%), Requirements Attributes (3.7%), and Requirement Tracing (3.7%). The details of number of each automated approach were also mentioned in this table.

In detail, 12 automated approaches belonged to Model Validation, which occupied 44.4% in the entire 27 automated approaches, which demonstrated this sub-category was a main research objective. In terms of the types of these 12 automated approaches, it consisted of 5 frameworks, 6 methods, and 1 platform.

P3-1, as an example, described how to map automated approach into Model Validation. The name of article 3 was "Refine Model checking by abstract interpretation", in which two improvements of abstract model-checking were proposed, which could be applied to infinite abstract transition systems: one was a new combination of forward and backwards abstract fixed-point model-checking computations for universal safety; the other was using partial results of classical combination of forward and backward abstract interpretation analysis for universal safety. Both of these improvements were refinement of the abstract model-check for automated analysis of software. According to the classification explained in the SWEBOK [20], these automated approaches belonged to Model Validation of Requirements Validation in Software Requirement field.

The second most popular sub-category was Elicitation Techniques, but the number of automated approaches was only 4 (14.8%), which was much less than the one was found in Model Validation.

Table 11. Automated approaches in Software Requirement field

Field	Sub-category	Approach	Number	Percentage
Requirements Elicitation	Requirements Sources	P108	1	3.7%
	Elicitation Techniques	P53-1, P53-2, P98, P111	4	14.8%
Requirements analysis	Conceptual Modeling	P17, P119	2	7.4%
Requirements Specification	Software Requirements Specification	P27-1, P27-2, P27-3	3	11.1%
Requirements Validation	Model Validation	P24, P35, P56, P76, P79, P84, P85, P88, P94, P104, P109, P3-1	12	44.4%
	Acceptance Testing	P44	1	3.7%
Practical Considerations	Requirements Attributes	P39	1	3.7%
	Requirement Tracing	P49	1	3.7%
	Measuring Requirements	P2, P51	2	7.4%

4.1.3. Software Testing

In Table 12, it stated 21 automated approaches in the Software Testing field. There were 3 automated approaches found respectively in two different sub-classes in the top stage of this table, which were Unit testing and Conformance testing /Functional testing /Correctness testing. They were the most popular sub-class for the researchers in the Software Testing field. Each of them occupied 14.3%, concerning the 21 automated approaches. The detailed description about these 6 automated approaches are stated below:

In the sub-class: Unit testing, automated approach P93 (Assume-guarantee testing) was a technique to check requirements performed during testing of individual components. The second automated approach (P96) was Extended Learning framework applied L* algorithm, which synthesized assumptions that automate assume-guarantee reasoning for finite-state machines and safety properties. The third automated approach (P117) was eCrash (automated test case generation tool) that automatically generated high quality test cases for Object-Oriented Java software.

In the sub-class: Conformance testing /Functional testing /Correctness testing, automated approach P4 combined PURDOM's and EXTENDED PURDOM's algorithm with Extended Generate_Minimum_Statement algorithm. They were applied to test syntax and semantic coverage of JAVA language compilers. The second automated approach (P33) was KLAIML framework that automatically verified properties in mobile applications programmed in X-KLAIM. The third automated approach (P65) was Tool-set (Verifying compiler), which automatically proved that a program would always meet its specification, insofar as this had been formalized, without the need to run it.

Furthermore, 4 different sub-classes were mentioned, including 2 automated approaches in each sub-class separately, which displayed the same percentage (9.5%). These 4 sub-classes were known as Integration testing; Finite-state machine-based; Testing from formal specifications and Component-based testing.

The rest 7 different sub-classes only covered 1 approach in each. These 7 sub-classes were Test selection criteria/Test adequacy criteria, Reliability achievement and evaluation, Regression testing, Configuration testing; Data flow-based criteria, Fault density and Test results evaluation. The percentage of each was 4.8%, considering the 21 automated approaches in Software Testing field.

Table 12. Automated approaches in Software Testing field

Field	Sub-category	Sub-class	Approach	Number	Percentage
Software Testing Fundamentals	Key issues	Test selection criteria/Test adequacy criteria	P118	1	4.8%
Test levels	The target of the test	Unit testing	P93, P96, P117	3	14.3%
		Integration testing	P21, P110	2	9.5%
	Objectives of Testing	Conformance testing/ Functional testing/ Correctness testing	P4, P33, P65	3	14.3%
		Reliability achievement and evaluation	P42	1	4.8%
		Regression testing	P48	1	4.8%
		Configuration testing	P75	1	4.8%
TEST techniques	Specification-based techniques	Finite-state machine-based	P20, P61	2	9.5%
		Testing from formal specifications	P41, P64	2	9.5%
	Code-based techniques	Data flow-based criteria	P60	1	4.8%
	Techniques based on the nature of the application	Component-based testing	P45, P77	2	9.5%
Test-related measures	Evaluation of the program under test	Fault density	P101	1	4.8%
Test Process	Test Activities	Test results evaluation	P57	1	4.8%

4.1.4. Software Quality

In Table 13, it contained 12 automated approaches in the Software Quality field.

The highest percentage, which is the most noticeable, was 33.3% in the sub-category of Software Quality Measurement, which included 4 automated approaches. The first automated approach (P29) was a framework named Genetic classifier supported by Self-organizing maps and evolutionary-based developed decision trees. It automatically analyzed the quality-based software engineering data. The second automated approach (P32) was a tool named Data Model Quality Advisor (DMQA) that automatically performing high-quality conversion of legacy software from one database model to another using a small and fixed set of transformations. The third one (P86) was a HMSRM framework (Hierarchical mixture of software reliability models). It automatically selected the most appropriate lower-level model for the data and performances in prediction. The fourth automated approach (P107) was used to measure the common features of domain knowledge with object oriented and developing a set of new quality property metrics to measure the characteristics that were particular to different domain knowledge components. The name of this framework was DKM (Domain knowledge quality metrics) and domain knowledge quality-measuring tool.

Quality Improvement was the second most noticeable sub-category in this table. It included 3 automated approaches. The percentage here was 25%.

The third most noticeable sub-category was Software Quality Management Techniques, which was further divided into two sub-classes, which were Analytical techniques with 2 automated approaches found, and Testing with only 1 automated approach found. The percentages of them were 16.7% and 8.3% respectively.

The least noticeable sub-category was Verification and Validation, which contained 2 automated approaches. The percentage each was 16.7%, out of 12 automated approaches.

Table 13. Automated approaches in Software Quality field

Field	Sub-category	Sub-class	Approach	Number	Percentage
Software Quality Fundamentals	Quality Improvement		P8, P89, P105	3	25%
Software Quality Management Processes	Verification and Validation		P15, P58	2	16.7%
Practical Considerations	Software Quality Management Techniques	Analytical techniques	P74, P100	2	16.7%
		Testing	P113	1	8.3%
	Software Quality Measurement		P29, P32, P86, P107	4	33.3%

4.1.5. Software Construction

Table 14 comprised 11 automated approaches used in Software Construction field.

Two sub-classes occupied the highest percentage (18.2%) in the entire table respectively, which were Programming Languages and Debugging.

There were 2 automated approaches found in the sub-class of Programming Languages. One approach (P54) was StreamBit (a sketching methodology), which supported a compiler automatically sketch to be faithful to the input reference code. The other (P78) was Nemo (Language) that specifying a set of resources with usage constraints, a set of tasks that consume them according to various modes, and applications sequencing the tasks.

In the Debugging sub-class, 2 automated methods were included, names of which were Delta Debugging (P19) and SOBER statistical method (P71). Former one used the result of automated testing to systematically narrow the set of failure-inducing circumstances. The latter one was used to automatically localize software faults without any prior knowledge of the program semantics.

Apart from those 4 automated approaches described before, other 7 approaches were displayed in 7 different Sub-categories and sub-classes, which were Construction Planning, Construction Measurement, Integration, Construction Languages Tools, Unit testing and Integration testing. Then percentages of them were only 9.1% for each, out of 11 automated approaches.

Table 14. Automated approaches in Software Construction field

Field	Sub-category	Sub-class	Approach	Number	Percentage
Software Construction Fundamentals	Constructing for Verification	Programming Languages	P54, P78	2	18.2%
		Tools	P80	1	9.1%
Managing Construction	Construction Planning		P28	1	9.1%
	Construction Measurement		P87	1	9.1%
Practical considerations	Reuse	Unit testing	P82	1	9.1%
		Integration testing	P122	1	9.1%
	Construction Quality	Debugging	P19, P71	2	18.2%
	Integration		P83	1	9.1%
	Construction Languages		P55	1	9.1%

4.1.6. Software Maintenance

In Table 15, 9 automated approaches were explored in the Software Maintenance field in total.

The highest percentage appeared was 44.4%, which contained 4 automated approaches in the sub-category of Reengineering. According to these data, it indicated that reengineering was the most noticeable sub-category to the researchers in the Software Maintenance field. The detailed descriptions of these 4 automated approaches in Reengineering are displayed below:

The first automated approach (P31) was MIDAS (automatic system), which performing high-quality conversion of legacy software from one database model to another using a small and fixed set of transformations. The second automated approach (P40) was WAD abbreviated to Wizard for Application Dictionary based on Compiere. It automatically created a Web-based management information system following the Model-View-Controller pattern. The third automated approach (P72) was Principles of conventional control theory. It defined and improved a requirement engineering (RE) process control system. The forth automated approach (P92) was a framework named FTSyn (Fault-Tolerance Synthesizer). It automatically synthesized several fault-tolerant programs, including a simplified version of an aircraft altitude switch, token ring, Byzantine agreement, and agreement in the presence of Byzantine and fail-stop faults.

Reverse engineering was the second top sub-category in this table, which included 2 automated approaches with the percentage 22.2%.

The other three automated approaches were displayed in the Nature of Maintenance, Software configuration management, and Software quality separately, of which the percentage was 11.1% for each.

Table 15. Automated approaches in Software Maintenance field

Field	Sub-category	Sub-class	Approach	Number	Percentage
Software Maintenance Fundamentals	Nature of Maintenance		P50	1	11.1%
Maintenance Process	Maintenance Activities	Software configuration management	P10	1	11.1%
		Software quality	P106	1	11.1%
Techniques for Maintenance	Reengineering		P31, P40, P72, P92	4	44.4%
	Reverse engineering		P36, P95	2	22.2%

4.1.7. Software Engineering Process

Comparing with other 6 different fields mentioned above in software engineering field, Software Engineering Process field only included 6 automated approaches.

The sub-category of Process measurement revealed the percentage of 33.3%, which was the highest in Table 16. It included 2 automated approaches, the first automated approach (P7) was a method combining MPM (measurement process model), Object Oriented concepts and tools; the second one (P73) was PROM metrics collection tool. The automated activity of the former one was to provide flexible design, structure and automatic generation of efficient implementations of DSL programs. The latter one's activity was supporting to manage a large measurement program.

The other 4 automated approaches were found in sub-categories of Process infrastructure, Automation, Process assessment models and Software information models, each of which only accounted for 16.7%.

Table 16. Automated approaches in Software Engineering Process field

Field	Sub-category	Approach	Number	Percentage
Process implementation and change	Process infrastructure	P81	1	16.7%
Process definition	Automation	P63	1	16.7%
Process assessment	Process assessment models	P43	1	16.7%
Process and product measurement	Process measurement	P7, P73	2	33.3%
	Software information models	P52	1	16.7%

4.1.8. Configuration Management

Table C in Appendix C stated the detailed description of what sub-categorizes displayed in Configuration Management field. The information in this table was referred to the SWEBOK [20]. Systematic review was exerted on the entire 122 articles, based on 17 sub-categorizes of Configuration Management fields, which were displayed in Table C. It turns out that there was no automated approach involved into these 17 sub-categories. The definition of Software Configuration Management (SCM) is a supporting software life cycle process (IEEE12207.0-96) which benefits project management, development and maintenance activities, assurance activities, and the customers and users of the end product [20]. SCM is closely related to the software quality assurance (SQA) activity. As defined in the Software Quality KA, SQA processes provide assurance that the software products and processes in the project life cycle conform to their specified requirements by planning, enacting, and performing a set of activities to provide adequate confidence that quality is being built into the software. SCM activities help in accomplishing these SQA goals [20]. Therefore, the researchers concentrated on exploring the automated approach in the SQA field rather than on the SCM.

Based on the result, it demonstrated that SQA was more popular to the researchers to explore the automatic approaches, comparing with the SCM.

4.1.9. Software Engineering Management

Table D in Appendix C displayed 24 Sub-categories in the Software Engineering Management field.

The result of exacting data phase showed that there was no automated approach found in this field, which was as the same as the situation in the Configuration Management field. Software Engineering Management can be defined as the application of management activities-planning, coordinating, measuring, monitoring, controlling, and reporting-to ensure that the development and maintenance of software is systematic, disciplined, and quantified (IEEE610.12-90) [20]. In the 122 papers, no relevant paper focused on Software Engineering Management field.

4.2. Level from Main Taxonomy

In Table 17, the entire 127 automated approaches were involved based on three different automated levels. (Detailed description of each level was mentioned in Section 3.1.3)

All the data were sorted according to the levels and years. In Table 17, only three different levels were included, in which Level D was excluded. The scope of this study focuses on automated approaches; therefore, Level D, presenting no automated approach, is apparently out of the study scope.

In this table, Number represented how many automated approaches were found in relevant levels. Research results were displayed according to the levels (Vertical) as well as the years (Horizontal). The percentages illustrated the corresponding numbers in the proportion of the total numbers of automated approaches found in that year (Horizontal).

According to the results in main taxonomy, LC is the most popular level, which included 50 automated approaches. Therefore, when computer exerted these 50 automated approaches, the Decision Support could be automatically conducted by generating suggestions and performing actions, but human needed to choose decisions. Software working on LC level cannot support fully automatic actions. Meanwhile, there were 45 and 32 automated approaches found respectively in the LA and LB level.

The largest numbers appeared in 2008 in Level A, which were 11, and it accounted for 50% out of the total number of the 22 automated approaches in that year. Oppositely, there was no automated approach found in neither Level A nor Level B in 2000, and the total number of automated approaches found in that year was only 2. In 1999, even the number of approaches classified in Level B was only 6; it still occupied 75% among the automated approaches found in that year.

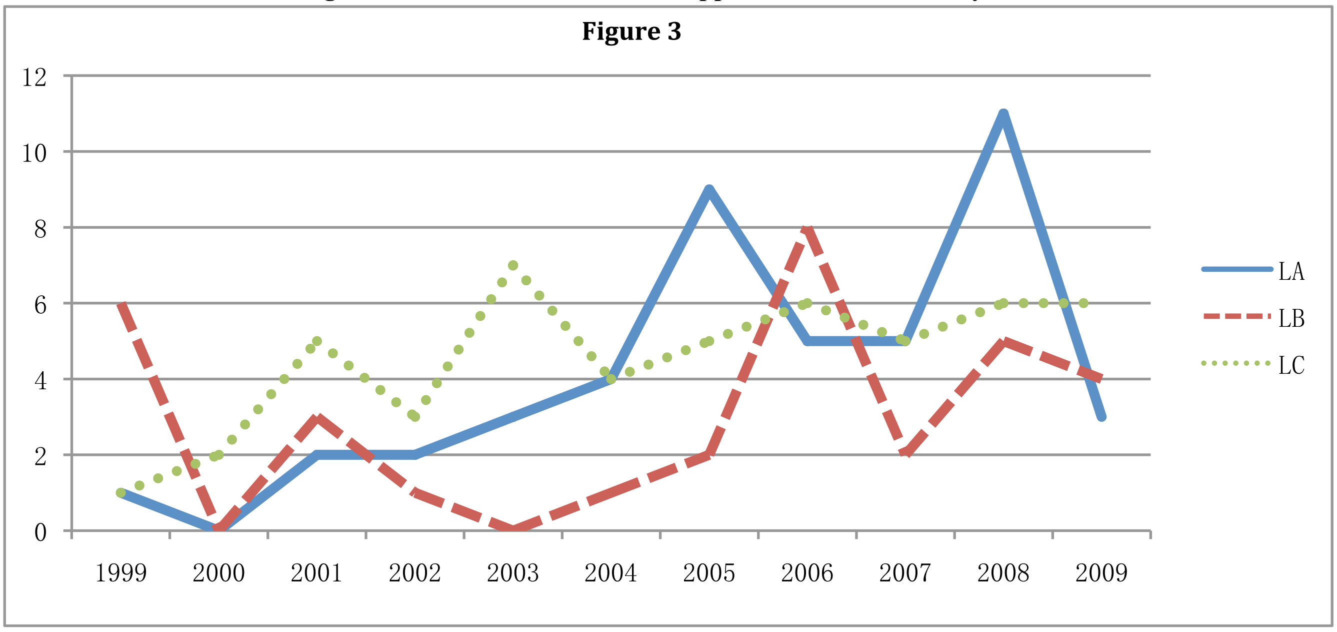
The contribution of Figure 3 is to make it easier to summarize this study regarding to an explicit tendency of these three levels from 1999 to 2009.

The vertical axis represented the numbers along with corresponding percentage of automated approaches found in different years (horizontal axis) in three levels separately. The data of the period during 2000 and 2004 displayed that more researchers have focused on LC level than other two levels in ASE. Afterwards, the figures of LA and LB levels were increasing dramatically, compared with the moderate increase of the figure in LC. Even during 2006 and 2007 the figure of LA and LB represented some decrease, the data of LA in 2008 could more explicitly explains the tendency of the levels of automated approaches found, which was that there was increasing tendency of Autonomous Level. Researches conducted more studies in this LA in order to find the automated approaches that could achieve full automation in ASE field.

Table 17. The Level of automated approaches based on the years

Year	Level A -Autonomous		Level B-Informing		Level C-Decision Support		Total Number
	Number	Percentage	Number	Percentage	Number	Percentage	
1999	1	12.5%	6	75%	1	12.5%	8
2000	0	0%	0	0%	2	100%	2
2001	2	20%	3	30%	5	50%	10
2002	2	33.3%	1	16.7%	3	50%	6
2003	3	30%	0	0%	7	70%	10
2004	4	44.4%	1	11.1%	4	44.4%	9
2005	9	56.3%	2	12.5%	5	31.3%	16
2006	5	26.3%	8	42.1%	6	31.6%	19
2007	5	41.7%	2	16.7%	5	41.7%	12
2008	11	50%	5	22.7%	6	27.3%	22
2009	3	23.1%	4	30.8%	6	46.2%	13
Total	45	35.4%	32	25.2%	50	39.4%	127

Figure 3. The Level of automated approaches based on the years



4.3. Required Human Activity from Main Taxonomy.

Table 20, 21, 22 were parts of main taxonomy, which comprised 22 Required Human Activities for entire 127 automated approaches (Detailed description about title of each column can be found in the Section 3.2.5.2). These three tables displayed 22 Required Human Activities separately, which based on the different levels of human effort they need - High, Medium and Low. These three different levels differ from each other based on the amount of the human efforts needed while using the automated approaches.

The contents in this table were sorted according to the years they were investigated. There are four main Sub-titles in the table: Required Human Activity, Relevant SE Area, Type, and Effort. To be specific, they referred to the activities that human need to do, in order to enable the usage of corresponding automated approaches; the software engineering fields, in which the activities will be carried out; what kind of activities they belong to; and the amount of the effort needed to spend on the execution of the activities.

The contributions of these three tables were to summarize how many automated approaches needed by human intervention and efforts, when applying the automated approaches with the purpose of making the approaches more effective. And the table showed whether these human activities needed were in the same field as the automated approaches or not.

Only 22 articles, which included 23 automated approaches, clearly stated the Required Human Activities, when introducing how to apply the associated automated approaches.

There were 15 out of 22 articles in total, which needed high requests of the human activities in order to satisfy the requirement of applying the automated approaches. There were 6 articles considering adding human activities, which belonged to the Medium Level. Only 1 article needed Low human intervention when applying the automated approach.

4.3.1. Summary of Required Human Activity

The articles mentioned Required Human Activity accounted for a proportion of 17.3% (22/127) in the entire automated approaches in this study, which demonstrated only few researchers focused on needed human efforts when proposing automated approaches in ASE. The reason might be that they concentrated more on both probing the automated approaches and how they work internally, instead of concerning what human needed to do to improve the efficiency of using automated approaches. Meanwhile, another reason could be the limitation of inclusion criteria (Detailed descriptions can be found in Section 3.1.4.3). For instance, the articles, which merely introduced how to use automated techniques, such as introducing case study conducted on using the techniques, were eliminated.

There was 15 out of 22 Required Human Activity needing high effort, which might indicate that the researchers would mention the necessity of the human intervention when it was really needed. Otherwise, if the usage of automated approach required just low efforts, the researchers might skip it in their articles, which might be the reason why only one paper found mentioning Low effort in Required Human Activity.

In addition, the primary purpose of including Required Human Activity was to see the software engineering area that human need to work on, in order to enable the usage of the automated approaches. The last finding showed that all the 22 Required Human Activity that researchers introduced were identical with the software field that the corresponding automated approaches were applied to. For example, in the paper 7, whose name was "Measurement processes are software tool", researchers proposed an automatic method (P7-1) combining measurement process model, Object Oriented concepts and tools which guiding the definition implementation and operation of measurement. It was marked in the field of Software Engineering Process/Process and product measurement/Process measurement. According to this paper, human needed to collect information manually when they used this automated approach. Required Human Activity exerted in the same field as the field that the automat ED approach applied to. Based on the paper 21, users needed to select testing criteria and extraction of UML Model of a web application in the integration testing manually. And then, the automated approaches ReWeb and TestWeb tools could perform automatic integration testing of the web application. They both belonged to the same Software Testing field.

Table 18. The High Level of Required Human Activity based on automated approaches

Year	Approach	Required Activity	Relevant SE Area	Type	Effort
1999	P2	Making the definition by following the multiple-criteria decision (MCDA) steps.	Software requirements/ Practical considerations/ Measuring requirements	Definition	High
1999	P7	Manually collecting information.	Software engineering process/ Process and product measurement/ Process measurement	Collecting	High
2001	P15	Estimating if the application contains the necessary features	Software quality/ Software quality management process/ Verification and validation	Preparing	High
2003	P31	Human programmers analyze the original program to find patterns of database accesses, such as filtering, joins, and aggregative operations.	Software maintenance/ Techniques for maintenance/ Reengineering	Analyzing	High
2004	P39	Manually categorizing requirements into software attributes.	Software requirements/ Practical considerations/ Requirements attributes	Categorizing	High

2005	P50	The programmer needs fills in the parts of the state transformer and stub functions.	Software maintenance/ Software Maintenance Fundamentals/ Nature of maintenance	Filling	High
2006	P64	Process feedback phase is provided by project team members who accomplish the task in person.	Software engineering process/ Process definition/ Automation	Providing	High
2006	P66	Coding using OWL language.	Software design/ Quality analysis and evaluation/ Simulation and prototype	Coding	High
2006	P71	Human prepare the exclusive test scripts for each component.	Software testing/ Key issues/ Test selection criteria/ Test adequacy criteria	Preparing	High
2007	P79	Failure modes of components must be injected by a safety engineer into the system model before model checking can be performed.	Software requirements/ Requirements validation/ Model validation	Modeling	High
2007	P82	Developers need to provide additional information such as class names, attribute names and types, and method names.	Software construction/ Practical considerations/ Reuse/ Unit testing	Providing	High
2007	P84	People set the target height in the control panel and the controller conducts the steeve to stop at the right place.	Software requirements/ Requirements validation/ Model validation	Setting	High
2008	P102	Designers specify the functionality and the faults that it may exhibit.	Software design/ Key issues in software design/ Error and exception handling and fault tolerance	Specifying	High
2009	P111	1. Eliciting MAS (Multiagent System) requirements with the REG.	Software requirements/ Requirements elicitation/ Elicitation techniques	Eliciting	High
2009	P112	Translating models to AOSE (Agent-oriented software engineering)	Software requirements/ Requirements elicitation/ Elicitation techniques	Translating	High

Table 19. The Medium Level of Required Human Activity based on automated approaches

Year	Paper No.	Required Human Activity	Relevant SE Area	Type	Effort
1999	P4	Considering and decide which semantic cases can be automated.	Software testing/ Test level/Objective of testing/Conformance testing, Functional testing, Correctness testing	Preparing	Medium
2000	P8	Carrying out the definition	Software quality/ Quality improvement/ The quality improvement process	Defining	Medium
2001	P19	Programmer must follow the causality chain and decide where to break it.	Software construction/ Construction quality/ Debugging	Deciding	Medium
2002	P21	1.User selects testing criteria 2. Extraction of UML Model of a web application	Software Testing/ Test levels/ Integration testing	Selecting and extracting	Medium
2006	P73	Human needs to specify which applications are installed in the target system and, consequently, which client-side components have to be installed.	Software engineering process/ Process and product measurement/ Process measurement	Specifying	Medium
2007	P81	Software engineer needs to combine manual tracing with the tool support during the software process.	Software engineering process/ Process implementation and change/ Process infrastructure	Tracing	Medium

Table 20. The Low Level of Required Human Activity based on automated approaches

Year	Paper No.	Required Activity	Relevant SE Area	Type	Effort
2003	P26	Producing derivatives without truncation error.	Software Design/quality analysis and evaluation/Simulation and prototype	Derivative production	Low

4.4. Types of Automated Approaches Used in ASE

Table 21 showed 10 types of automated approaches found from 1999 to 2009 in this study, which were Tool, Framework, Method, Technique, Model, System, Language, Platform, Theory, and Process. (Detail descriptions of these automated approaches can be found in the Appendix C).

The contribution of Table 21 is to display:

- How many automated approaches were found in this study?
- Which automated approach was more popular?
- Which year was more important for the automated approach during 1999 to 2009?
- What was the situation for automated approach?
- Which articles did include more than one automated approach?

Based on the figures summarized in Table 21, 127 automated approaches were specified into 10 types out of entire 122 articles. The top 3 popular types were Tool, Framework and Method, which the numbers of automated approaches included were 32, 28 and 26 respectively. Contrarily, Platform, Theory and Process appeared as the 3 least popular types of automated approach, whose amount was only 4.

In terms of the differences between years, 2008, 2006 and 2005 represented the top 3, the numbers of which were 22, 19 and 17 respectively.

Comparing with the figures displayed in the top 3 types, it was obviously showed that the development trend of tool and framework was the same, which was increasing from 1999 to 2009, but the one in Method was opposite.

From this table, it could be explicitly showed that the diversity of automated approaches became more.

The fourth from the bottom was Language, the number of which was only 5. The reason was that languages were frequently reused in other automated approaches.

In table 21, all the numbers marked refer to the highest numbers of automated approaches in each type, based on the years that they were proposed.

4.4.1. Tools used in ASE

These 32 tools used in 8 different software engineering fields. (Detail descriptions of tools can be found in Appendix C, Table E).

In Table E, 7 tools were located in the Software Design field, and 6 tools belonged to Software Testing field. In Software Requirements and Software Quality field the number was the same, which was 5. Software Maintenance field included 4 tools. Of both Software Engineering Process and Software Construction fields, it was 2 respectively. Only 1 tool belonged to Software Testing field.

In terms of the tools used in the Software Design field, 7 tools were mapped into 5 kinds of sub-classes, which were Detailed Design, Behavior description, Simulation and prototype, Architectural Structures and viewpoints, and Behavior patterns.

The most special article was P27, whose name was “Application of linguistic techniques for use case analysis”. It explained how to perform a quality evaluation of the requirements documents to ensure the quality of the documents from the linguistic perspective and the tools used to support the metrics. There were three tools mentioned and used in this article, which were Quality analyzer for requirements specifications (QuARS), Automated requirement measurement (ARM) and SyTwo. Their automated activities all belonged to Software Requirements field.

4.4.2. Frameworks used in ASE

There were 28 frameworks used in 7 different software engineering fields, the detailed distributions were: Software Design (11), Software Requirement (7), Software Test (3), Software Quality (3), Software Management (2), Software Engineering Process (1), Software Construction (1).

In terms of 11 frameworks used in sub-category and sub-class of Software Design field, 4 of them located in Quality Analysis and Evaluation; 3 used in Detailed design; 2 used in Software Structure and Architectural; only 1 used in Distribution of Components, and Coupling and cohesion separately.

In terms of 7 frameworks applied in sub-category and sub-class of Software Requirement, 6 used in Requirements Validation. There was only 1 used in Requirements Elicitation.

4.4.3. Methods used in ASE

The amount of Methods used in automated software engineering was 26. Similar as the frameworks mentioned in last section, they were distributed in 7 different software engineering fields as well. More precisely, Software Design (12), Software Requirements (5), Software Construction (3), Software Engineering Process (2), Software Engineering Technique (2) Software Test (1), Software Engineering Management (1).

The most popular was Software Design field (12), half of which used in sub-category of Quality Analysis field. The rest of 6 methods were equally divided into sub-class of Detailed design, and Error and Exception Handling respectively.

The second most popular was Software Requirement field (5). All of these 5 methods were used in Requirements Validation area.

It is apparent that the automated approaches were most frequently used in either Software Design or Software Requirement field. And the sub-classes like Quality Analysis and Evaluation, Detailed design, and Model Validation attracted more attentions from researchers to investigate.

Table 21. The Numbers of each automated approach based on the years

Year	Tool	Framework	Method	Technique	Model	System	Language	Platform	Theory	Process	Total
1999	1	3	1	1	0	1	1	0	0	0	8
2000	0	1	0	0	0	2	0	0	0	0	3
2001	3	0	5	0	1	0	0	0	0	0	9
2002	2	0	4	0	0	0	0	0	0	0	6
2003	6	2	1	0	0	1	0	0	0	0	10
2004	3	3	0	1	0	0	1	0	0	0	8
2005	2	3	4	3	3	1	1	0	0	0	17
2006	4	2	3	3	2	3	0	0	1	1	19
2007	4	2	2	0	1	0	2	1	0	0	12
2008	6	7	3	2	2	1	0	1	0	0	22
2009	1	5	3	1	2	1	0	0	0	0	13
Total	32	28	26	11	11	10	5	2	1	1	**127**

5. DISCUSSION

The systematic review conducted in this study found a total of 122 primary articles, which together proposed 127 different automated approaches in software engineering in the years 1999 to 2009 (inclusive). The data summarizing the results in these articles have been presented in Section 4, particularly in Tables 7 to 20, Figure 3 and in Appendix C. Below we discuss our results based on the major types of analysis conducted.

5.1. Field of software engineering

In terms of the fields that automated approaches belongs to, Software Design and Software Requirement were the top two most popular areas, where automated approaches were attempted most often. They included 41 automated approaches (32.3% in among total approaches) and 27 automated approaches (21.3%), respectively. Together they account for more than half of the ASE results. The overall data can be found in Table 7 while Tables 9 to 15 in Section 4 details the papers within each area.

The result indicated Software Design and Software Requirement were more important compared with Software Testing and other software engineering fields. The reason behind it can be that these two fields were located at the beginning of software development, and became the basis of further work in the entire development processes. The quality of Software Requirement and Software Design may affect the works afterwards, therefore more researches focus on these two fields.

One of the most reasons why Software Design became the most critical research field was that it was located in the beginning of software development processes. The quality of a Software Design can strongly affect the implementation of software, and influence the time and cost of entire software development, as well as the performance and quality of the delivered software.

Architecture generation is the first step in the design of software system [23] [24] [25], based on the result from this field, the papers involved architecture design, structure, architectural quality analysis, etc. Meanwhile, it considered embedded systems [P34], distributed systems [P69], real-time systems [P11, P14, P90], etc. The reason for many researches were found is that most of the qualities that the final software system possesses are usually decided at the architecture development stage itself [26].

A fundamental goal of software development is to deliver high quality products that are correct, consistent, and complete. There had been several examples of approaches to generate architectures based on functional requirements, whereas, almost no comprehensive automated approaches to consider non-functional requirements at the architecture development stage was mentioned. Since the current state of distributed system is ad hoc and manually, it should be considered and engineered into the architecture itself as well [P34]. It had driven the researchers to investigate automated approaches that support effectively designing and analyzing, in order to realize the non-functional requirements [P34, P67] at software architecture design level of producing automated approach in Software Design field. The significant benefits of these approaches included detecting and removing defects earlier, reducing development time and cost while improving the system's quality. Meanwhile, ensure non-functional requirements, such as adaptability during the process of architecture generation. For example, P34 presented an automated design method that helps develop adaptable architectures for embedded systems by developing a tool called Software Architecture Adaptability Assistant (SA3).

The other reason for most researches found in Software Design field is that the possibility and feasibility of developing automated approach is higher. For example, real-time systems were becoming increasingly widespread often in safety-critical application, and it is critical to ensure the concurrency and timing property of them [P11]. However, the behavior of them is often affected by the unpredictable factors, such as communication delays and the arbitrary interleaving of computations performed by different processes, which caused non-determinism significantly complicates system testing and debugging. As the result, testing of real-time system is difficult because erroneous behaviors may not manifest themselves during test runs even if the appropriate input test data are used [P90]. When comparing with the late stage of the development process where the final system has already been implemented and integrated together, it is much cheaper at design stage not only to fix bugs, if any, but also to make some changes. Additionally, testing and reliability requirements must be taken into account as early as possible in the design stage [P14]. Therefore, easier to apply automated approaches in design stage to guarantee the correctness, usability, and performance of the system before coding and testing.

Also, a good Software Design can be reused by similar software developments, which makes guaranteeing quality in Software Design area become even more important.

According to the results, Software Requirements played the second critical role in determining the overall software correctness and quality. It defines the documents of users' requirements of systems and serve as the baseline for the development of the software [27], in which, the errors are very costly, even impossible, to rectify at later stages of software development. Therefore, the validation of requirements definitions is of vital importance to software development [28]. Which can be one reason that it attracted many focuses.

Within Software Requirement, one of the most critical phases of software engineering is requirements elicitation and analysis. The quality of requirements and their associated analysis influence the success of a software project, since their outputs contribute to higher level design and verification decisions [P53]. Requirements analysis paves the way for high-level design, generation of test cases for verification, and supports early architecture reviews, which increases the importance of it. It was the basic reason that automated requirement elicitation techniques were proposed in papers P53, P99, P111.

Because reliable techniques for natural language understanding was not available, software developers rarely apply formal requirements specification techniques in practice, and it's generally infeasible to automatically identify conflicts and cooperation of requirements. Whereas, paper P39 adopted requirements traceability to help achieve better understanding and monitoring persistent software attributes, such as reliability, scalability, efficiency, security, and usability.

Among various automated approaches suggested for developing high-quality requirements specifications and conducting cost-effective analysis, formal methods were considered effective and promising [29].

Formal method combining abstract interpretation and model checking was considered for automated analysis of software [P3]. From the results in this study, model validation has been proven to be a powerful automatic verification technique [30]. There were 12 researches focused on the development model checking [P24, P35, P56, P57, P79, etc] indicated the prevalence of it. The feasibility and possibility of applying model validation in Software Requirement field therefore become another reason of why it is popular.

5.2. Automation Level

In terms of the automation level of automated approaches, Level C was the most popular level, which included 50 approaches (39.4% in among total approaches). The overall data can be found in Table 16 and Figure 3 in Section 4.

The reason why Level C, which is a low automation level, became the most researched level can be that ASE is still a new research field in software engineering, and most researches were conducted without previous relevant works being their basis, therefore, researches have to start at a lower level automation, and human interventions were needed on decision supporting in order to make sure that automated approaches can work successfully in software development.

Similar conjecture can be drawn from the result of Level A as well, which included 45 automated approaches (35.4% in among total approaches). During the first 5 years (From 1999 to 2003) the top researched levels were either Level C or Level B, while afterwards, there were more researches conducted on Level A. As a result, Level A became the top researched automation Level in other 4 years (2004, 2005, 2007, 2008). This indicated that researchers were trying to increase the automation level, with a few years' researches on LB and LC as their basis. It indicated that researches on fully automated approach are the trend of research on ASE, and the complete liberation of human is the ultimate aim of researches.

5.3. Required Human Activity

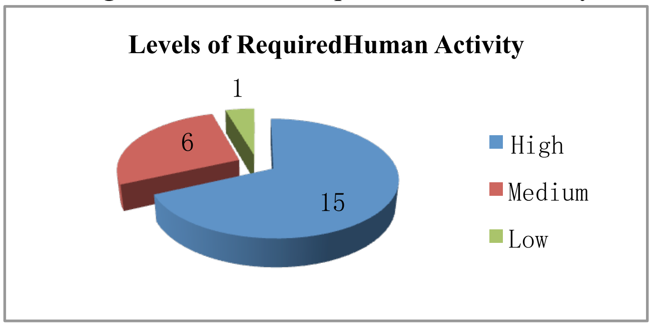
Required Human Activity is needed while applying the automated approaches, which cannot achieve full-automation. There were only 22 Required Human Activity clearly stated for corresponding automated approaches. The overall data can be found in Table 17, 18 and 19 in Section 4. The result proved the ignorance from researches on human interventions.

Comparing with 22 mentioned Required Human Activity automated approaches; researchers omitted them in the rest 105 automated approaches' human intervention. In fact, the description of Required Human Activity is still needed when using some automated approaches. In total, only 45 automated approaches can fully automation. Apart from them, still 82 automated approaches only support partly automation, which indicated Required Human Activity is still needed. For example, in Software Design field, 31 automated approaches cannot fully automation. 26 out of 31 automated approaches most conduct information analysis automatically. Depend on different automated approaches applied, human need to manually collect information (Acquisition information), make decision, or act implementation. These Required Human Activities need to be mentioned when researches describe the automated approaches. Since, giving clear description about Required Human Activity can help reader to understand how to adopt the automated approaches.

Some of researchers tended to represent Required Human Activity in the when describing the automated approaches. For example, in paper P13, the name of this automated approach was design units. It provided the basis for automatic generation of modular source code, and played a critical role in both code generation and test plan generation. Researchers only focused on describing the code generation process and how to apply the concept of design units into the code generation process. This automated approach belonged to LC, which indicated it couldn't provide fully automation. Therefore, Required Human Activity was needed when apply design units. There was little indication of Required Human Activity in this paper, whereas researchers tended to omit it. When applying this automated approach, Design and test engineers might have different options from at design time to carry out their tasks respectively. Therefore, the detailed description of Required Human Activity is needed to make readers and users have clear understanding about how to use this automated approach.

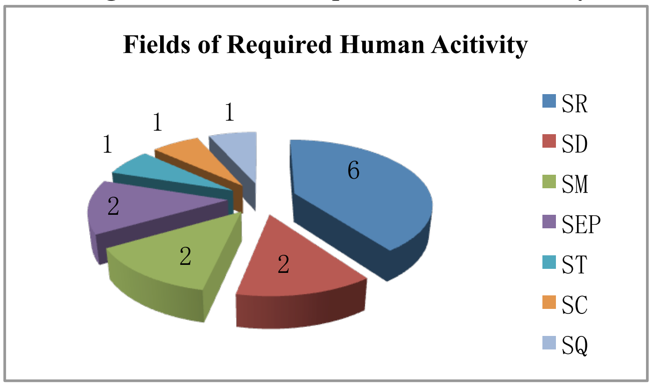
According to the summary of Level of Required Human Activity in Figure 4, there were 15 activities requiring High-Level human interventions, which further indicated that researchers tended to mention the Required Human Activity, only when the activities were really needed. Otherwise, even though some of the automated approaches cannot achieve full-automation, the researchers would skip their required human interventions.

Figure 4. Levels of Required Human Activity



According to the summary concerning software engineering field in Figure 5, Software Requirement field, was the most researched field, where Required Human Activity was mentioned, and the number was accounted for 6. The reason for this could be that more human activities were needed at the beginning of the entire software development processes.

Figure 5. Fields of Required Human Activity



5.4. Types of Automated Approaches

In terms of the type of automated approaches, there were 10 types of automated approaches found (The overall data can be found in Table 20 in Section 4).

The result indicated that automated Tool (32), Framework (28) and Method (26) were top three most researched types of automated approaches, the amount of which rated 68%, out of 127 automated approaches found in this study. It indicated that Tool, Framework, and Method were always the most popular and traditional types within the ASE field.

Based on the result, there were only 7 types of automated approaches researched before 2006. Afterwards, researches were conducted on three new types of automated approaches, Platform (2), Theory (1) and Process (1). It indicated that, besides considering the traditional types of automated approaches, researchers started investigating new types of automated approaches and tended to find new automated approaches in ASE field to meet the increasing needs of liberating developers from repeated work.

5.5. Limitation of the Research

This study only focuses on summarizing the current situation in ASE field. We focused on how many automated approaches applied in ASE and what they were; what corresponding activities of them were; which software engineering field and automation level the automated approaches belonged to; what kind of human intervention needed to be exerted when applying the automated approaches, and which level corresponding human interventions belonged to.

For the further research, it may focus on what kind of separate approaches can be reused in exploring new automated approaches. It will be useful for the researcher to explore new automated approaches.

5.6. Recommendation

There are a few recommendations for future researches, after systematically reviewing 122 articles:

To begin with, clear discussion of the automation level that automated approaches belong to should be included. It can make it easier for the user to understand the automated approach, as well as a clearer view of in what ways they can use them.

Furthermore, detailed description of the types of human activities needed, when users apply the corresponding automated approaches, should be involved. Meanwhile, it should consider not only what Required Human Activity is needed before applying automated approaches, but also what is needed afterwards. Additional human activities, which might enable the automated approach to be applicable and improve the usability of it, need to be included as well.

Finally, in this study, the result found that in Software Design and Software Requirement fields, it's more likely that the users can find proper automated approaches to facilitate their work. Resulting from the findings, the users are more encouraged to apply appropriate automated approaches. Meanwhile, the tool, framework, method are most researched types according to this study, when considering applying automated approach into software development, these mentioned types of automated approaches can be the best options.

6. Conclusion

In this study, a systematic review investigating the automated approaches in ASE field from the year 1999 to 2009 was presented. The purpose of this study is to summarize the researches conducted and the benefits and limitations of automated approaches in ASE field. There were 127 automated approaches found in 122 publish articles,

We summarized these automated approaches that included 10 types of automated approaches: tool, framework, technique, system, language, model, method, theory, process and platform. According to the result, Tool revealed the top popularity. These automated approaches have been classified into 7 different areas in software engineering fields including Software Design, Software Requirement, Software Testing, Software Quality, Software Construction, Software Maintenance and Software Engineering Process, which based on SWEBOK.

The result presented the current trend of researches in different fields in ASE area. Software Design and Software Requirements were the top two most researched areas. Detailed design and model validation were the top two most researched sub-classes of Software Design and Software Requirements respectively. The reason of if is to avoid bug to emerge in the steps afterwards, meanwhile, for it is more expensive to check the correctness of system after they are integrated together. Therefore, many automated approaches were proposed to apply in generation of architecture in Software Design fields, with the purpose of improving the overall quality of the architecture of system. Many automated approaches considering Model Validation and Requirement Elicitation were proposed to improve the quality in Software Requirement fields.

Meanwhile, we summarized the popularity of automation level of 127 automated approaches investigated as well. LC (Decision Support) was the most popular level in those three levels. Furthermore, 22 Required Human Activity were mentioned, in which there were 15 belonged to High Level, indicating that the researchers wouldn't introduce them until necessary.

We have observed that many automated approaches were developed in combination of different approaches, and made it fully automated. Future works can be undertaken to explore the possibility of combining these existing automated approaches to make more new automated approaches, which can be fully automated.

REFERENCE

[1] Hameed, S.A.; , "Toward software engineering principles based on Islamic ethical values," Computer and Communication Engineering, 2008. ICCCE 2008. International Conference on, vol., no., pp.379-385, 13-15 May 2008]

[2] Dag I. K. Sjoberg , Tore Dyba , Magne Jorgensen, The Future of Empirical Methods in Software Engineering Research, 2007 Future of Software Engineering, p.358-378, May 23-25, 2007

[3] Roy, G.G.; Veraart, V.E.; , "Software engineering education: from an engineering perspective," Software Engineering: Education and Practice, 1996. Proceedings. International Conference , vol., no., pp.256-262, 24-27 Jan 1996

[4] A.R. Dalton and J.O. Hallstrom, "nAIT: a source analysis and instrumentation framework for nesC," *Journal of Systems and Software* 82, no. 7, J. Syst. Softw. (USA) (July 2009): 1057-72.

[5] B. Kitchenham, T. Dyba, and M. Jorgensen. Evidence-based software engineering. In Proceedings of 26th International Conference on Software Engineering (ICSE'04), pages 273– 284, Edinburgh, Scotland, UK, May 2004. IEEE Computer Society.

[6] Davies, A., Dieste, O., Hickey, A., Juristo, N., and Moreno, A.M. (2006) Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review, Proceedings 14th IEEE International Requirements Engineering Conference (RE'06), IEEE Computer Society, pp. 179–188.

[7] Dybå, T. and Dingsøyr, T. (2008) Empirical Studies of Agile Software Development: A Systematic Review, Information and Software Technology, 50(9-10): 833–859.

[8] Américo Sampaio, Ruzanna Chitchyan, Awais Rashid, Paul Rayson,"EA-Miner: a Tool for Automating Aspect-Oriented Requirements Identification", ASE'05, November 7–11, 2005, Long Beach, California, USA

[9] Brandon Morel and Perry Alexander, "Automating Component Adaptation for Reuse", Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE'03)

[10] Egyed, A., Biffl, S., Heindl, M., and Grünbacher, P. 2005, "Determining the cost-quality trade-off for automated software traceability", In *Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering* (Long Beach, CA, USA, November 07 - 11, 2005). ASE '05. ACM, New York, NY, 360-363.

[11] Javed, A. Z., Strooper, P. A., and Watson, G. N, "Automated Generation of Test Cases Using Model-Driven Architecture", In *Proceedings of the Second international Workshop on Automation of Software Test* (May 20 - 26, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 3.

[12] Weifeng Xu, Dianxiang Xu, "Automated Evaluation of Runtime Object States against Model-Level States for State-Based Test Execution," Software Testing Verification and Validation Workshop, IEEE International Conference on, pp. 3-9, IEEE International Conference on Software Testing, Verification, and Validation Workshops, 2009

[13]Jorge Biolchini, Paula Gomes Mian, Ana Candida Cruz Natali, Guilherme Horta Travassos, "Systematic Review in Software Engineering", Rio de Janeiro, May 2005.

[14]Barbara Kitchenham, "Procedures for Performing Systematic Reviews", July 2004.

[15] Keele,staffs, Durham,"Guideline for performing Systematic Literature Review in Software Engineering" EBSE Technical Report, EBSE-2007-01.

[16] Md. Abdullah Al Mamun, Aklima Khanam, "Concurrent Software Testing: A Systematic Review and an Evaluation of Static Analysis Tools", 2009

[17] Zotero. [Online].Viewed 2010 March 26. Available: http://www.zotero.org/

[18] Kitchenham BA (2007) Guidelines for performing systematic literature reviews in software engineering. EBSE Technical Report EBSE-2007-001

[19] Raja Parasuraman, Thomas B. Sheridan, " A Model for Types and Levels of Human Interaction with Automation"

[20] "Guide to the Software Engineering Body of Knowledge (SWEBOK)" IEEE Computer Society. http://www.computer.org/portal/web/swebok/home

[21] P. Brereton, B. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," Journal of Systems and Software, vol. 80, Apr. 2007, pp. 571-83.

[22] G.W. Noblit and R.D. Hare, Meta-ethnography: synthesizing qualitative studies, Sage Publications Inc, 1988.

[23] IEEE, 1996. IEEE Standard 12207.0-1996, Industry Implementation of International Standard ISO/IEC 12207: 1995; ISO/IEC 12207 Standard for Information Technology—Software Life Cycle Processes.

[24] Shaw, M., Garlan, D., 1996. Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall, Upper

Saddle River, NJ. p. 19, 34, 149.
[25] Bass, L., Clements, P., Kazman, R., 1998. Software Architecture in Practice. In: SEI Series in Software Engineering. Addison-Wesley, Reading, MA. p. 94, 113.
[26] Bass, L., Klein, M., Bachmann, F., 2000. Quality attribute design primitives. Software Engineering Institute, Technical report CMU/ SEI-2000-TN-017.
[27] Stokes, D.A., 1991. Requirements analysis. In: Software Engineer's Reference Book. Butterworth-Heinemann, London.
[28]Boehm, B.W., 1981. Software engineering economics. Advances in Computing Science and Technology. Prentice-Hall, Englewood Cliffs, NJ.
[29] L. Chung et al., Non-Functional Requirements in Software Engineering, Kluwer, 2000.
[30] Clarke, E.M., and Grumberg, O.: 'Model checking'
(The MIT Press,1999)

APPENDIX A

Table A Frequency of Keywords

NO.	Keywords from Inspec	Frequency	Percentage	NO.	Keywords Found In ACM	Frequency	Percentage
1	program	27	51%	1	software	36	55%
2	quality	25	47%	2	model*	18	28%
3	tool*	19	36%	3	automat*	17	26%
4	software quality	18	34%	4	engineering	14	22%
5	system*	16	30%	5	develop*	13	20%
6	formal	15	28%	6	program*	13	20%
7	language*	15	28%	7	system*	12	18%
8	model*	15	28%	8	test*	12	18%
9	data	14	26%	9	analysis	10	15%
10	programming	14	26%	10	language*	10	15%
11	software engineering	14	26%	11	generat*	9	14%
12	code*	13	25%	12	design	8	12%
13	specification	13	25%	13	tool*	8	12%
14	program testing	12	23%	14	code	7	11%
15	testing	12	23%	15	reuse	7	11%
16	maintenance	11	21%	16	software engineering	7	11%
17	software maintenance	11	21%	17	component	6	9%
18	software tools*	11	21%	18	process	6	9%
19	automated	10	19%	19	embeded	5	8%
20	comput*	10	19%	20	pattern	5	8%
21	generat*	10	19%	21	product line	5	8%
22	verification	10	19%	22	requirements	5	8%
23	formal specification	9	17%	23	support	5	8%
24	process*	9	17%				
25	management	8	15%				
26	program verification	8	15%				
27	software development	8	15%				
28	object-oriented	7	13%				
29	object-oriented programming	7	13%				
30	program diagnostics	7	13%				
31	safety	7	13%				

Percentage = Frequency/Amount number of articles

Amount of articles found in:
Inspect (53)
ACM Digital Library (64)

APPENDIX B: Main taxonomy and two tables of Software Engineering fields omitted in Section 4.1.8 and 4.1.9.

Table B. Main Taxonomy

Article	Year	
	Paper No.	
Automated Approach	AA NO.	
	Activity	
	Approach	
	Relevant SE Area	
	Automated Level	
Required Activity	RA NO.	
	Activity	
	Relevant SE Area	
Setup Cost	Type	
	Effort	

Table C. Automated Approaches in Configuration Management field

Field	Sub-category	Approach
Management of the SCM Process	Organizational Context for SCM	0
	Constraints and Guidance for SCM Process	0
	Planning for SCM	0
	Software Configuration Management Plan	0
	Surveillance of SCM	0
Software Configuration Identification	Identifying Items to be Controlled	0
	Software Library	0
Software Configuration Control	Requesting, Evaluating and Approving Software Changes	0
	Implementing Software Changes	0
	Deviations and Waivers	0
Software Configuration Status Accounting	Software Configuration Status Information	0
	Software Configuration Status Reporting	0
Software Configuration Auditing	Software Functional Configuration Audit	0
	Software Physical Configuration Audit	0
	In-Process Audits of a Software Baseline	0
Software Release Management and Delivery	Software Building	0
	Software Release Management	0

Table D. Automated Approaches in Software Engineering Management field

Field	Sub-category	Approach
Initiation and Scope Definition	Determination and Negotiation of Requirements	0
	Feasibility Analysis	0
	Process for the Review and Revision of Requirements	0
Software Project Planning	Process Planning	0
	Determine Deliverables	0
	Effort, Schedule and Cost Estimation	0
	Resource Allocation	0
	Risk Management	0
	Quality Management	0
	Plan Management	0
Software Project Enactment	Implementation of Plans	0
	Supplier Contract Management	0
	Implementation of Measurement Process	0
	Monitor Process	0
	Control Process	0
	Reporting	0
Review and Evaluation	Determining Satisfaction of Requirements	0
	Reviewing and Evaluating Performance	0

Closure	Determining Closure	0
	Closure Activities	0
SW Engineering Measurement	Establish and Sustain Measurement Commitment	0
	Plan the Measurement Process	0
	Perform the Measurement Process	0
	Evaluate Measurement	0

APPENDIX C: **Table used in the result**

Table E. Tool by Year

Year	Tool	Field	Description	Approach
1999	1. PURDOM's algorithm 2. EXTENDED PURDOM's algorithm 3. Extended Generate_Minimum_Statement algorithm	Software testing/ Test level/ Objective of testing/ Conformance testing, Functional testing, Correctness testing	Testing syntax and semantic coverage of JAVA language compilers.	P4
2001	Lusceta tool suite	Software maintenance/ Maintenance process/ Maintenance Activities/ Software configuration management	The current version of Lusceta tool suite provides support for editing, composing and simulating (stochastically enhanced) timed automata.	P10
2001	TR system (algorithm tools)	Software quality/ Software quality management process/ Verification and validation	Validation and maintenance process of Engineering requirement.	P15
2001	CASE tool built in a manner similar to the SORAC prototypes	Software design/ Software structure and architecture/ Design patterns/ Behavioral patterns	Automatically support pattern for the development of tools, for the specification of databases and for engineering design systems.	P16
2002	Simple Covering (Tool)	Software testing/ Test techniques/ Specification-based techniques/ Finite-state machine-based	Refinement of stream X-machine with expanding the input and output behaviors.	P20
2002	TestWeb	Software testing/ Test levels/ Integration testing	It is a research tool to support testing processes. This tool exploits a reverse engineered UML model of the Web application to generate and execute test cases, in order to satisfy the testing criteria selected by the user.	P21
2003	Automatic differentiation (AD) tool	Software design/ Quality analysis and evaluation/ Simulation and prototype	Accurate evaluating derivatives of functions described in a high-level programming language.	P26-1
2003	Quality analyzer for requirements specifications (QuARS) tool	Software requirements/ Requirements specification/ Software requirements specification	QuARS is a sentence analyzer aiming at reducing linguistic defects by pointing out those wordings that make the document ambiguous or unclear from a lexical point of view.	P27-1
2003	Automated requirement measurement (ARM) tool	Software requirements/ Requirements specification/ Software requirements specification	ARM is to providing measures that can be used to assess the quality of a requirements specification document.	P27-2
2003	SyTwo tool	Software requirements/ Requirements specification/ Software requirements specification	Sytwo is a tool that was developed as a web application to perform a lexical and syntactical analysis of English text.	P27-3

2003	CATSDL tool	Software design/ Quality analysis and evaluation/ Simulation and prototype	A coverage analysis tool for SDL specification	P30
2003	Data Model Quality Advisor (DMQA) tool	Software quality/ Practical considerations/ Software quality measurement	It provides a hypertext explanation facility for the constructs of the quality evaluation framework, and supports evaluation and comparison of up to three data models at a time.	P32
2004	Software Architecture Adaptability Assistant (SA3) tool	Software Design/ Software design notations/ Behavioral description	Selecting the architectural constituents that best fit the adaptability requirements for the architecture	P34
2004	MDRE (Using Specware code generator)	Software maintenance/ Techniques for maintenance/ Reverse engineering	Using formal specification and automatic code generation to reverse the reverse engineering process.	P36
2004	WAD tool (Wizard for Application Dictionary)	Software maintenance/ Techniques for maintenance/ Reengineering	Producing code from an application dictionary in a relational database.	P40
2004	PrUDE (Precise UML Development Environment) CASE tool	Software testing/ Test techniques/ Specification-based techniques/ Testing from formal specifications	PrUDE platform integrates the graphical UML notation as a front-end to the PVS (Prototype Verification System) verification tools.	P41
2005	RETRO (Requirements Tracing on Target) tool	Software requirements/ Practical considerations/ Requirement Tracing	Automatically providing predictive information before any code has been written.	P49
2006	C-Saw (Constraint-Specification Aspect Weaver) transformation Engine	Software Design/ Software Design Fundamentals/ Software Design Process/ Detailed Design	Modularizing crosscutting properties and replicate element of core model	P62
2006	Tool-set (Verifying compiler)	Software testing/ Test level/ Objective of testing/ Conformance testing, Functional testing, Correctness testing	Automatically proves that a program will always meet its specification, insofar as this has been formalized, without even needing to run it.	P65
2006	PROM metrics collection tool	Software engineering Process/ Process and product measurement/ Process measurement	Supporting to manage a large measurement program.	P73
2006	PLFaultCAT (Product-Line Fault Tree Creation and Analysis Tool). Safety analysis tool	Software quality/ Practical considerations software quality management techniques/ Analytical techniques	To aid software engineers in the application of product-line software SFTA (Software Fault Tree Analysis).	P74
2007	μCRL Toolset	Software testing/ Test techniques/ Techniques based of nature of application/	Automatically checking the secrecy of values inside components.	P77

		Component-based testing		
2007	GTB (the Grammar Tool Box)	Software construction/ Software construction fundamentals/ Constructing for verification/ Tools	Providing implementations of grammar transforms, automata construction algorithms, parsing and recognition algorithms, and a variety of visualization aids.	P80
2007	Traceability recovery tool based on Latent Semantic Indexing (LSI)	Software engineering process/ Process Implementation and change/ Process infrastructure	Identifying potential traceability links not traced yet (Suggested Links) and possible text description problems in the traced artifacts (Warning Links).	P81
2007	Diff-CatchUp tool	Software construction/ Managing construction/ Construction measurement	Inferring plausible replacements for the offending API that causes the API migration problem and examines the code base built on the evolved framework to select examples of how the potential replacements are used.	P87
2008	ConQAT (Continuous Quality Assessment Toolkit.	Software quality/ Software quality fundamentals/ Quality improvement	Identifying and resolve quality defects early in the development process, when implementing countermeasures is still inexpensive.	P89
2008	Planning based tools (Mixed-initiative planning algorithms: UCPOP and SHOP2)	Software design/ Software Structure and Architecture/ Architectural Structures and Viewpoints	Helping the developer to build a "design plan", based on the selection and articulation of a collection of "design operations" for each "design domain".	P90
2008	CODe-Imp	Software maintenance/ Techniques for maintenance/ Reverse engineering	Automatically refactoring object-oriented programs to improve quality as measured by well-defined quality models.	P95
2008	SMaRT (Scenario Management and Requirements Tool)	Software requirements/ Requirements elicitation/ Elicitation techniques	Improving scenario quality and providing effective automated support for work with scenarios.	P98
2008	Semantic metrics to analyse design specifications in NL-based program comprehension tool (Tool)	Software Design/ Software Design Fundamentals/ Software Design Process/ Detailed Design	Providing a consistent and seamless type of metric that can be collected through the entire lifecycle.	P99
2008	Metrics-Based Design selection tool	Software quality/ Software quality fundamentals/ Quality improvement	Automatically selecting the better AOD (Aspect Oriented Programming) from alternative designs of an application based on the proposed metrics.	P105
2009	eCrash (automated test case generation tool)	Software testing/ The target of the test/ Test levels/ Unit test	Automatically generating high quality test case for Object-Oriented Java software.	P117

Table F. Framework by Year

Year	Framework	Field	Description	Approach
1999	Combination of Model-Checking and Abstract Interpretation	Software requirements/ Software validation/ Model validation	Abstract interpretation based universal safety model checking for infinite abstract systems.	P3-1
1999	Abstract interpretation	Software design/ Quality analysis and evaluation/	Elimination the impossible potentially infinite behaviors.	P3-2

1999	Framework comprises two parts: definition of an abstract machine and definition of a DSL in terms of the abstract machine operations	Software design/ Software structure and architecture/ Families of programs and frameworks	Providing flexible design, structure and automatic generation of efficient implementations of DSL programs.	P6
2000	MCRDR (Multiple classification RDR (ripple-down rules)) /FCA Framework	Software design/ Software structure and architecture/ Families of programs and frameworks	Allowing user to view, exploring analyze, maintaining, manipulate and consulting the knowledge in a knowledge-based system.	P9
2003	Environment For Combining Optimization and Simulation Software (EFCOSS) framework	Software design/ Quality analysis and evaluation/ Simulation and prototype	This framework supports the interoperability of simulation and optimization software in an automated fashion and also provides an easy way for the integration of derivative code.	P26-2
2003	Genetic classifiers supported by (Self-organizing maps and evolutionary-based developed decision trees) framework	Software quality/ Practical considerations/ Software quality measurement	Analysis of quality-based software engineering data.	P29
2004	KLAIML framework	Software Testing/ Objectives of Testing/ Conformance testing, Functional testing, Correctness testing	Automatically verifying properties in mobile applications programmed in X-KLAIM	P33
2004	Verifiable Embedded Real-Time Application Framework (VERTAF) framework	Software requirements/ Requirements validation/ Model validation	Automatic design of embedded real-time system integrating functional and nonfunctional requirements	P35
2004	Co-operative connectors (Architectural entity) (Framework)	Software design/ Key issues in software design/ Distribution of components	Providing to describe the software components, the interactions between these components, and the properties that regulate the composition of components.	P37
2005	Formal Design Analysis Framework (FDAF)	Software requirements/ Requirements validation/ Acceptance testing	It is an aspect-oriented approach that supporting the automated translation of extended Unified Modeling Language designs for distributed real-time systems into existing formal notations, including Architecture Description Languages Rapide and Armani.	P44
2005	Dynamic software updating (based on dynamic patches) Framework	Software maintenance/ Software Maintenance Fundamentals/ Nature of maintenance	Automatic generation of patch files.	P50
2005	Metadata-driven framework	Software engineering process/ Process and product measurement/ Software information models	Automatically producing queuing-base performance models.	P52
2006	Framework combines OWL (Web Ontology Language) and SweDE (Semantic Web Development Environment)	Software design/ Quality analysis and evaluation/ Simulation and prototyping	Evaluation of the performance and QoS of ambient intelligent systems.	P66
2006	FDAF (Formal Design Analysis Framework)	Software design/ Quality analysis and evaluation/ Simulation and	Modeling and predicting the performance cost of security aspect.	P67

		prototyping		
2007	Q-algebras framework	Software requirements/ Requirements validation/ Model validation	Model checking for concurrent components.	P76
2007	Adaption approach framework combine computable safety criterion and interaction patterns (procedure calls and events publishing/subscribing.)	Software construction/ Practical considerations/ Integration	Specifying the coordination between components, handling and checking adaptations of this coordination.	P83
2008	Component substitutability check part of COMFORT reasoning framework	Software requirements/ Requirements validation/ Model validation	Localizing the necessary verification effort to only modified system components of evolving software, and reduce dramatically the effort to check substitutability after every system update.	P88
2008	SCE (System-on-chip environment) design framework	Software Design/ Software Design Fundamentals/ Software Design Process/ Detailed Design	Integration of automatic model generation, estimation, and verification tools enables rapid design space exploration and efficient MPSoC implementation.	P91
2008	FTSyn (Fault-Tolerance Synthesizer)	Software maintenance/ Techniques for maintenance/ Reengineering	Automatically synthesizing the several fault-tolerant programs including a simplified version of an aircraft altitude switch, token ring, Byzantine agreement, and agreement in the presence of Byzantine and fail-stop faults.	P92
2008	Extended Learning framework applied L* algorithm	Software testing/ The target of the test/ Test levels/ Unit test	Synthesizing assumptions that automate assume-guarantee reasoning for finite-state machines and safety properties.	P96
2008	SMF (Safety modeling framework)	Software design/ Quality analysis and evaluation/ Static analysis	Analyzing and developing safety-aware UML architectures.	P103
2008	Three-level framework (Feature model level, diagnosis level and implementation level)	Software requirements/ Requirements validation/ Model validation	Supporting automatic error detection and explanation.	P104
2008	Framework: DKM (Domain knowledge quality metrics) and domain knowledge quality-measuring tool.	Software quality/ Practical considerations/ Software quality measurement	To measure the common features of domain knowledge with OO and develops a set of new quality property metrics to measure the characteristics that are particular to different domain knowledge components.	P107
2009	REG (Requirements Elicitation Guide) based on the AT (Activity Theory) (Framework)	Software requirements/ Requirements elicitation/ Elicitation techniques	Guiding requirements elicitation and increases the productivity with the use of templates for a wide range of requirements.	P111
2009	Encompassing taxonomy of visual guideline for UML class diagrams. (Framework)	Software Design/ Software Design Fundamentals/ Software Design Process/ Detailed Design	Improving the aesthetic quality and thus the understandability of UML class diagrams.	P112
2009	Traceability reference model and Rule-based approach (Framework)	Software quality/ Practical considerations/ Software quality management techniques/ Testing	Support automatic generation of traceability relations between feature-based object-oriented documents.	P113
2009	BaVeL (Framework)	Software Design/ Software Design Fundamentals/ Software Design Process/	Verifying results obtained in semantic domains to different formats, including the context of the original language.	P115

		Detailed Design		
2009	An approach: Two coverage criteria (process coverage and modified condition/decision coverage) for LOTOS specifications (Framework)	Software testing/ Key issues/ Test selection criteria/ Test adequacy criteria	Allowing automatic generation of coverage based test suites and can be used to automatically exercise those aspects of the system that are missed by handcrafted test purposes.	P118

Table G. Technique by Year

Year	Techniques	Field	Description	Approach
1999	Enhanced Compositional Reachability Analysis (CRA) with Property Automata	Software design/ Quality analysis and evaluation/ Static analysis	To check safety properties which may contain actions that are not globally observable.	P1
2004	RT (Requirements traceability) (Technique)	Software requirements/ Practical considerations/ Requirements attributes	1.Automatically identifying conflicts and cooperation among requirements based on their attributes. 2.Automatically generating trace dependencies among the requirements.	P39
2005	DART (Daily Automated Regression Tester) (Technique)	Software testing/ Objectives of Testing/ Regression testing	Automate GUI smoke testing.	P48
2005	State chart model combine with timed automata	Software requirements/ Requirements elicitation/ Elicitation techniques	Time dependencies can be represented allowing a more thorough and accurate analysis.	P53-1
2005	Message sequence charts combine with timed automata	Software requirements/ Requirements elicitation/ Elicitation techniques	Providing a rich set of semantics and relationships to an abstract clock that could be used for real-time analysis.	P53-2
2006	NuSPADE (combine by proof planning and a program analysis oracle)	Software quality/ Software quality management process/ Verification and validation	Increasing the level of automation in high integrity software verification.	P58
2006	Program restructuring approach using the clustering techniques	Software design/ Software design fundamentals/ Enabling techniques/ Coupling and cohesion	Automated support for identifying ill-structured or low-cohesive functions and providing heuristic advice in both the development and evolution phases.	P59
2006	The technique based on compositional model checking and program analysis.	Software testing/ Test techniques/ Specification-based techniques/ Testing from formal specifications	Automatic verification of infinite families of systems.	P64
2008	Assume-guarantee testing (Technique)	Software testing/ The target of the test/ Test levels/ Unit test	Checking requirements is performed during testing of individual components.	P93
2008	Brokering algorithm which extends query processing techniques (Technique)	Software quality/ Practical considerations/ Software quality management techniques/ Analytical techniques	Automatically deriving an integer linear programming problem that returns an optimal matching of data providers to data consumers under realistic economic cost models.	P100
2009	Test-data generation approach: Suspicious statement selection and test-data generation based on the suspicious statements. (Technique)	Software construction/ Practical considerations/ Reuse/ Integration testing	Generating test data with high fault detection.	P122

Table H. System by Year

Year	System	Filed	Description	Approach
1999	ESSE (expert system for software evaluation)	Software requirements/ Practical considerations/ Measuring requirements	A prototype expert system for software evaluation that embodies various aspects of the Multiple-Criteria Decision Aid (MCDA) methodology.	P2
2000	DSES (Decision Supporting Expert System)	Software quality/ Software quality fundamentals/ Quality improvement	Inspection for quality evaluation.	P8
2000	Deductive System	Software requirements/ Requirements analysis/ Conceptual modeling	Automated support to System Requirement Analysis in the development of time-and safety-critical computer-based systems.	P17
2003	MIDAS (automatic translation system)	Software maintenance/ Techniques for maintenance/ Reengineering	It uses temporal abstraction techniques to discover database access patterns in the host program and translate them to relational-database operations.	P31
2005	Multi-agent System (MAS)	Software testing/ Test techniques/ Techniques based of nature of application/ Component-based testing	Practicing agent-oriented software testing. (Effectiveness in selecting the appropriate assignment based on requirements).	P45
2006	WAT (Agent-based WA testing system)	Software testing/ Test techniques/ Code-based techniques/ Data flow-based criteria	Automatically generate and coordinate test agents to decompose the task of testing an entire WA into a set of subtasks.	P60
2006	DRE SEMANTIC DOMAIN (System)	Software design/ Key issues in software design/ Distribution of components	Verifying distributed non-preemptive real-time scheduling of embedded systems.	P68
2006	MODEST (Modeling and Description language for Stochastic Timed systems) modeling formalism	Software Design/ Software Design Fundamentals/ Software Design Process/ Detailed Design	A language to model real-time and stochastic concurrent systems.	P70
2008	ISFEA automated knowledge-based system	Software requirements/ Requirements elicitation/ Requirements sources	Intelligent supporting of the preprocessing stage of engineering analysis in the contact mechanics domain.	P108
2009	Symbolic Model Verifier (SMV) system	Software Design/ Software Design Fundamentals/ Software Design Process/ Detailed Design	Performing safety analysis of software requirement through generating fault tree and verifying safety properties automatically.	P120

Table I. Language by Year

Year	Language	Field	Description	Approach
1999	Real-time Estelle (Language)	Software design/ Software design notations/ Behavior description/ Formal specification language	Generating implementation and guarantee specified real-time quality-of-service requirements automatically.	P5
2004	TUG (Formal specification language)	Software design/ Software design notations/ Design patterns/ Formal specification languages	It supports an automatic derivation of a prototype in Prolog from a specification in the language via a set of transformation rules.	P38

2005	XML (Extensible Markup Language) based WSAMI (Web services for ambient intelligence)	Software construction/ Practical consideration/ Construction language	Developing ambient intelligence application based on Web services.	P55
2007	Nemo (programming language)	Software construction/ Software construction fundamentals/ Constructing for verification/ Programming languages	Specifying a set of resources with usage constraints, a set of tasks that consume them according to various modes, and applications sequencing the tasks.	P78
2007	RDL (Reuse Description Language)	Software construction/ Practical considerations/ Reuse/ Unit testing	Specifying object-oriented framework instantiation processes.	P82

Table J. Model by Year

Year	Model	Field	Description	Approach
2001	An extended I/O automata formalism (Model)	Software design/ Key issues in software design/ Error and exception handling and fault tolerance	Specifying fault tolerance in mission Critical Intelligent Systems.	P18
2005	SRGM (with generalized logistic TEF and change-point) Model SRGM-Software Reliability Growth Model	Software testing/ Objectives of testing/ Reliability achievement and evaluation	Describing the fault detection/removal process during software development.	P42
2005	OOSPICE metamodel (Object-Oriented and Component-Based Software Process Improvement and Capability Determination)	Software engineering process/ Process assessment/ Process assessment models	Automatically ensuring that their executed work conforms to the appropriate assessment model.	P43
2005	Probabilistic analysis (Model)	Software design/ Software design quality analysis/ Quality analysis and evaluation/ Object-oriented design measures	Evaluating the evolution of a design through successive generations and to identify "bad" classes that can cause changes to the rest of the system.	P46
2006	EARL (Evaluation and Report Language) Data model	Software testing/ Test process/ Test activities/ Test results evaluation	It builds on Semantic Web technologies in order to make use of already existing metadata vocabulary, APIs, repositories, as well as other tools and resources.	P57
2006	State characterization model	Software testing/ Test techniques/ Specification-based techniques/ Finite-state machine-based	Automated test generation of test inputs using model checking.	P61
2007	HMSRM (Hierarchical mixture of software reliability models)	Software quality/ Practical considerations/ Software quality measurement	Automatically selecting the most appropriate lower-level model for the data and performances in prediction.	P86
2008	ED3M (Estimation of Defects based on Defect Decay Model)	Software testing/ Test-related measures/ Evaluation of the program under test/ Fault density	Computing an estimate of the total number of defects in an ongoing testing process.	P101
2008	GCT (Goal Centric Traceability)	Software maintenance/ Maintenance process/ Maintenance activities/ Software quality	Explicitly link QAMs to goals, to identify initial impact points, and to provide executable links between QAMs and goals that support dynamic reevaluation during an impact analysis event.	P106
2009	SIGNALMETA metamodel	Software design/ Software Structure and	Automated transformations are defined and implemented in order to produce,	P114

		Architecture/ Architectural Structures and Viewpoints	analyze, statically verify and model-check programs obtained from high-level models.	
2009	Requirement metamodel	Software requirements/ Requirements analysis/ Conceptual modeling	Extending the conceptual models used by Web Engineering methodologies with the aim of allowing the explicit consideration of usability requirements along with the evaluation of quality metrics during the design of the system.	P119

Table K. Method by Year

Year	Method	Field	Description	Approach
1999	Method combine MPM (measurement process model), Object Oriented concepts and tools	Software engineering process/ Process and product measurement/ Process measurement	Guiding the definition implementation and operation of measurement.	P7
2001	STP set (Simple Time Petri net)	Software design/ Software design quality analysis/ Quality analysis and evaluation/ Static analysis	To model the behavior of a program specification and allow to automatically analyze larger programs.	P11
2001	SMV Model Checker	Software design/ Software design quality analysis/ Quality analysis and evaluation/ Software design review	Automatically analyzing the potential for model confusion of interactive system.	P12
2001	Design units	Software design/ Software design quality analysis/ Quality	Automatic generation of modular source code.	P13
2001	MOSYS (Methodology for Automatic Object Identification from System Specification) supported by SIM2SYS tool	Software design/ Software design quality analysis/ Quality analysis and evaluation/ Software design review	Providing a method for automatically generating alternative design objects architectures.	P14
2001	Delta Debugging (Method)	Software construction/ Construction quality/ Debugging	Using the result of automated testing to systematically narrow the set of failure-inducing circumstances.	P19
2002	Reuse-oriented UMP methodology	Software design/ Software design Fundamentals/ Software design process/ Detailed design	Building generalized designs of Information system maintenance toolkits that maintain the requirements specification of Information system.	P22
2002	Concurrent Designer's Assistant (CODA) (Method)	Software design/ Software design Fundamentals/ Software design process/ Architectural design	Largely automates the process of generating a concurrent design.	P23
2002	Testing tool combine three methods. (Data flow testing methods, State transition testing methods, Entity testing methods) (Method)	Software requirements/ Software validation/ Model validation	Automatically generate a set of task activity lists according to state transition based test criteria, and to measure the adequacy of the test set according a set of data flow adequacy criteria.	P24
2002	MEDI (Methodology for estimate design intent) supported by MGP (Multiple genetic programming)	Software design/ Software design quality analysis/ Quality analysis and evaluation/ Software design review	Automatically estimate or extract design intent based on the data recorded from the design process, without interrupting designer's normal design activities. It is a reasoning method.	P25
2003	Extension the Lustre with Mode-Automata (Method)	Software construction/ Managing construction/ Construction planning	Description of these running modes of regulation system.	P28
2005	Decomposition method	Software design/ Software design Fundamentals/	Automated decomposition of a system into IDEAL components.	P47

		Software design process/ Architectural design		
2005	BDSA (Bi-directional safety-analysis method) combine SFMEA (Software Failure Modes and Effects Analysis) and SFTA (Software Fault Tree Analysis).	Software requirements/ Practical considerations/ Measuring requirements	Performing safety analysis on a software product line.	P51
2005	StreamBit (a sketching methodology)	Software construction/ Software construction fundamentals/ Constructing for verification/ Programming languages	A compiler automatically sketch is faithful to the input reference code.	P54
2005	Automated method of translating SCR (software cost reduction)-style requirements into PVS input language.	Software requirements/ Requirements validation/ Model validation	Verifying functional properties in PFS using PVS.	P56
2006	Semi-automated process tailoring method	Software engineering process/ Process definition/ Automation	It uses the artificial-neural network-based learning theory to reduce the time.	P63
2006	X-frame (A template of GFTSA (Generic Fault Tolerant Software Architecture) based on XVCL methodology (XML-based Variant Configuration Language))	Software design/ Key issues in software design/ Error and exception handling and fault tolerance	Customizing the formal template of GFTSA to formal models of specific systems automatically.	P69
2006	SOBER statistical method	Software construction/ Construction quality/ Debugging	Automatically localizes software faults without any prior knowledge of the program semantics.	P71
2007	Formal method based on Safe charts model and SGM (State-Graph Manipulator)	Software requirements/ Requirements validation/ Model validation	Verifying if a safety-critical system is safe or not.	P79
2007	Method: Timed automata model and Uppaal (Symbolic model checker)	Software requirements/ Requirements validation/ Model validation	Validating the safety and time constraint properties of embedded system with programmable logic controller, which modeled by timed automata.	P84
2008	Architectural Risk Analysis methodology based on Security Patterns	Software requirements/ Requirements validation/ Model validation	Automatically extracting the risk of a software system by reading the class diagram of the system.	P94
2008	SCRAPE (Safety-Critical Real-time Applications Exploration) (Method)	Software design/ Key issues in software design/ Error and exception handling and fault tolerance	Automatically deducing the replication, mapping and scheduling embedded control Software.	P102
2008	Timed Behavior Tree	Software requirements/ Requirements validation/ Model validation	Checking timed behavior tree model of time-critical systems using UPPAAL to support FMEA (Failure Mode and Effects Analysis).	P109
2009	Method combines three techniques on: computing the PSEs; determining the relevant instances; redefining a constraint in terms of the best context type	Software testing/ Test levels/ Integration testing	Facilitating the efficient integrity checking of UML-based software specifications complemented with a set of integrity constraints defined in Object Constraint Language (OCL)	P110
2009	nAIT	Software design/ Quality analysis and evaluation/	Providing a foundation for extending the use of automated software	P116

		Static analysis	engineering methods to the domain of wireless sensor network.	
2009	Global-to-Local approach (Method)	Software design/ Key issues in software design/ Error and exception handling and fault tolerance	Given a well-formed global description, a set of peers can be generated automatically.	P121

Table L. Theory by Year

Year	Theory	Field	Description	Approach
2006	Principles of conventional control theory (Theory)	Software maintenance/ Techniques for maintenance/ Reengineering	Defining and improving a requirement engineering (RE) process control system.	P72

Table M. Process by Year

Year	Process	Field	Description	Approach
2006	Variable-strength covering-array (Process)	Software testing/ Testing levels/ Objectives of testing/ Configuration testing	Testing higher-level interactions only in subspaces needed, while keeping a low level of coverage across the entire space.	P75

Table N. Platform by Year

Year	Platform	Field	Description	Approach
2007	FSAP/NuSMV-SA (Platform)	Software requirements/ Requirements validation/ Model validation	Improving the development cycle of complex systems by providing a uniform environment that can be used both at design time and for safety assessment.	P85
2008	A reduced Diopsis tile (Platform)	Software design/ Software structure and architecture/ Families of programs and frameworks	Efficiently use the resource of the architecture and allowing easy experimentation of several mappings of the application onto the platform resources.	P97

APPENDIX D:

Table L. The references in the systematic review

Paper	Ref
S.C. Cheung and J. Kramer, “Checking safety properties using compositional reachability analysis,” *ACM Transactions on Software Engineering and Methodology* 8, no. 1 (1999): 49-78.	P1
I. Vlahavas et al., “ESSE: an expert system for software evaluation,” *Knowledge-Based Systems* 12, no. 4 (1999): 183-197.	P2
P. Cousot and R. Cousot, “Refining model checking by abstract interpretation,” *Automated Software Engineering* 6, no. 1, Autom. Softw. Eng. (Netherlands) (January 1999): 69-95.	P3
A.S. Boujarwah, K. Saleh, and J. Al-Dallal, “Testing syntax and semantic coverage of Java language compilers,” *Information and Software Technology* 41, no. 1, Inf. Softw. Technol. (Netherlands) (January 15, 1999): 15-28.	P4
S. Fischer, “Towards the automatic generation of quality-of-service-preserving implementations from formal specifications,” *Computer Communications* 22, no. 3, Comput. Commun. (Netherlands) (February 23, 1999): 211-23.	P5
S.A. Thibault, R. Marlet, and C. Consel, “Domain-specific languages: from design to implementation application to video device drivers generation,” *IEEE Transactions on Software Engineering* 25, no. 3, IEEE Trans. Softw. Eng. (USA) (May 1999): 363-77.	P6
M. Morisio, “Measurement processes are software, too,” *Journal of Systems and Software* 49, no. 1, J. Syst. Softw. (USA) (December 15, 1999): 17-31.	P7
E.P. Paladini, “An expert system approach to quality control,” *Expert Systems with Applications* 18, no. 2, Expert Syst. Appl. (UK) (February 2000): 133-51.	P8
D. Richards, “The reuse of knowledge: a user-centred approach,” *International Journal of Human-Computer Studies* 52, no. 3, Int. J. Hum.-Comput. Stud. (UK) (March 2000): 553-79.	P9
L. Blair et al., “Formal support for dynamic QoS management in the development of open component-based distributed systems,” *IEE Proceedings: Software* 148, no. 3 (2001): 89-97.	P10
U. Buy and R.H. Sloan, “Automatic real-time analysis of reactive systems with the parts toolset,” *Automated Software Engineering* 8, no. 3 (2001): 227-273.	P11
J.C. Campos and M.D. Harrison, “Model checking interactor specifications,” *Automated Software Engineering* 8, no. 3, Autom. Softw. Eng. (Netherlands) (2001): 275-310.	P12
Jaehyoun Kim and C.R. Carlson, “Design units-a layered approach for design driven software development,” *Information and Software Technology* 43, no. 9, Inf. Softw. Technol. (Netherlands) (2001): 539-49.	P13
O.P. Dias et al., “On identifying and evaluating object architectures for real-time applications,” *Control Engineering Practice* 9, no. 4, Control Eng. Pract. (UK) (April 2001): 403-9.	P14
T.L. McCluskey and M.M. West, “The automated refinement of a requirements domain theory,” *Automated Software Engineering* 8, no. 2, Autom. Softw. Eng. (Netherlands) (April 2001): 195-218.	P15
J. Peckham and B. MacKellar, “Generating code for engineering design systems using software patterns,” *Artificial Intelligence in Engineering* 15, no. 2, Artif. Intell. Eng. (UK) (April 2001): 219-26.	P16
A. Gargantini and A. Morzenti, “Automated deductive requirements analysis of critical systems,” *ACM Transactions on Software Engineering and Methodology* 10, no. 3, ACM Trans. Softw. Eng. Methodol. (USA) (July 2001): 255-307.	P17
T.S. Perraju, “Specifying fault tolerance in mission critical intelligent systems,” *Knowledge-Based Systems* 14, no. 7, Knowl.-Based Syst. (UK) (November 2001): 385-96.	P18
A. Zeller, “Automated debugging: are we close?,” *Computer* 34, no. 11, Computer (USA) (November 2001): 26-31.	P19
F. Ipate and M. Holcombe, “An integrated refinement and testing method for stream X-machines,” *Applicable Algebra in Engineering, Communications and Computing* 13, no. 2 (2002): 67-91.	P20
F. Ricca and P. Tonella, “Testing processes of Web applications,” *Annals of Software Engineering* 14, Ann. Softw. Eng. (Netherlands) (2002): 93-114.	P21
S.M. Schorling and D.C. Rine, “A methodology for designing toolkits for specification level verification of interval-constrained information systems requirements,” *Information and Software Technology* 44, no. 2 (2002): 77-90.	P22
K.L. Mills and H. Gomaa, “Knowledge-based automation of a design method for concurrent systems,” *IEEE Transactions on Software Engineering* 28, no. 3, IEEE Trans. Softw. Eng. (USA) (March 2002): 228-55.	P23
Hong Zhu et al., “Software requirements validation via task analysis,” *Journal of Systems and Software* 61, no. 2, J. Syst. Softw. (USA) (March 15, 2002): 145-69.	P24
Y. Ishino and Y. Jin, “Estimate design intent: a multiple analysis genetic programming and multivariate based approach,” *Advanced Engineering Informatics* 16, no. 2, Adv. Eng. Inf. (UK) (April 2002): 107-25.	P25
C.H. Bischof et al., “Solving large-scale optimization problems with EFCOSS,” *Advances in Engineering Software* 34, no. 10 (2003): 633-639.	P26
A. Fantechi et al., “Applications of linguistic techniques for use case analysis,” *Requirements Engineering* 8, no. 3, Requir. Eng. (UK) (2003): 161-70.	P27
F. Maraninchi and Y. Remond, “Mode-automata: a new domain-specific construct for the development of safe critical systems,” *Science of Computer Programming* 46, no. 3, Sci. Comput. Program. (Netherlands) (March 2003): 219-54.	P28

M. Reformat, W. Pedrycz, and N.J. Pizzi, "Software quality analysis with the use of computational intelligence," *Information and Software Technology* 45, no. 7, Inf. Softw. Technol. (Netherlands) (May 1, 2003): 405-17.	P29
W.E. Wong et al., "Coverage testing software architectural design in SDL," *Computer Networks* 42, no. 3, Comput. Netw. (Netherlands) (June 21, 2003): 359-74.	P30
Y. Cohen and Y.A. Feldman, "Automatic high-quality reengineering of database programs by abstraction, transformation and reimplementation," *ACM Transactions on Software Engineering and Methodology* 12, no. 3, ACM Trans. Softw. Eng. Methodol. (USA) (July 2003): 285-316.	P31
D.L. Moody and G.G. Shanks, "Improving the quality of data models: empirical validation of a quality management framework," *Information Systems* 28, no. 6, Inf. Syst. (UK) (September 2003): 619-50.	P32
L. Bettini, R. De Nicola, and M. Loreti, "Formulae meet programs over the net: A framework for correct network aware programming," *Automated Software Engineering* 11, no. 3 (2004): 245-288.	P33
L. Chung and N. Subramanian, "Adaptable architecture generation for embedded systems," *Journal of Systems and Software* 71, no. 3 (2004): 271-295.	P34
P.-A. Hsiung et al., "VERTAF: An application framework for the design and verification of embedded real-time software," *IEEE Transactions on Software Engineering* 30, no. 10 (2004): 656-674.	P35
S. Rugaber and K. Stirewalt, "Model-driven reverse engineering," *IEEE Software* 21, no. 4, IEEE Softw. (USA) (July 2004): 45-53.	P36
R. de Lemos, "Analysing failure behaviours in component interaction," *Journal of Systems and Software* 71, no. 1, J. Syst. Softw. (USA) (April 2004): 97-115.	P37
Chia-Chu Chiang, "Automated rapid prototyping of TUG specifications using Prolog," *Information and Software Technology* 46, no. 13, Inf. Softw. Technol. (Netherlands) (October 1, 2004): 857-73.	P38
A. Egyed and P. Grunbacher, "Identifying requirements conflicts and cooperation: how quality attributes and automated traceability can help," *IEEE Software* 21, no. 6, IEEE Softw. (USA) (November 2004): 50-8.	P39
N. Serrano et al., "Automated management of multicustomer code bases," *IEEE Software* 21, no. 6, IEEE Softw. (USA) (November 2004): 26-31.	P40
I. Traore and D.B. Aredo, "Enhancing structured review with model-based verification," *IEEE Transactions on Software Engineering* 30, no. 11, IEEE Trans. Softw. Eng. (USA) (November 2004): 736-53.	P41
Chin-Yu Huang, "Performance analysis of software reliability growth models with testing-effort and change-point," *Journal of Systems and Software* 76, no. 2, J. Syst. Softw. (USA) (May 2005): 181-94.	P42
C. Gonzalez-Perez, T. McBride, and B. Henderson-Sellers, "A metamodel for assessable software development methodologies," *Software Quality Journal* 13, no. 2, Softw. Qual. J. (USA) (June 2005): 195-214.	P43
K. Cooper, Lirong Dai, and Yi Deng, "Performance modeling and analysis of software architectures: An aspect-oriented UML based approach," *Science of Computer Programming* 57, no. 1, Sci. Comput. Program. (Netherlands) (July 2005): 89-108.	P44
D. Ponnurangam and G.V. Uma, "Fuzzy complexity assessment model for resource negotiation and allocation in agent-based software testing framework," *Expert Systems with Applications* 29, no. 1, Expert Syst. Appl. (UK) (July 2005): 105-19.	P45
N. Tsantalis, A. Chatzigeorgiou, and G. Stephanides, "Predicting the probability of change in object-oriented systems," *IEEE Transactions on Software Engineering* 31, no. 7, IEEE Trans. Softw. Eng. (USA) (July 2005): 601-14.	P46
D. Wang, F.B. Bastani, and L.-I. Yen, "Automated aspect-oriented decomposition of process-control systems for ultra-high dependability assurance," *IEEE Transactions on Software Engineering* 31, no. 9, IEEE Trans. Softw. Eng. (USA) (September 2005): 713-32.	P47
A.M. Memon and Q. Xie, "Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software," *IEEE Transactions on Software Engineering* 31, no. 10, IEEE Trans. Softw. Eng. (USA) (October 2005): 884-96.	P48
J.H. Hayes, A. Dekhtyar, and S.K. Sundaram, "Improving after-the-fact tracing and mapping: supporting software quality predictions," *IEEE Software* 22, no. 6, IEEE Softw. (USA) (November 2005): 30-7.	P49
M. Hicks and S. Nettles, "Dynamic software updating," *ACM Transactions on Programming Languages and Systems* 27, no. 6, ACM Trans. Program. Lang. Syst. (USA) (November 2005): 1049-96.	P50
R.R. Lutz and Qian Feng, "Bi-directional safety analysis of product lines," *Journal of Systems and Software* 78, no. 2, J. Syst. Softw. (USA) (November 2005): 111-27.	P51
A. D'Ambrogio and G. Iazeolla, "Metadata-driven design of integrated environments for software performance validation," *Journal of Systems and Software* 76, no. 2 (2005): 127-146.	P52
H. Saiedian, P. Kumarakulasingam, and M. Anan, "Scenario-based requirements analysis techniques for real-time software systems: a comparative evaluation," *Requirements Engineering* 10, no. 1, Requir. Eng. (UK) (2005): 22-33.	P53
A. Solar-Lezama et al., "Programming by sketching for bit-streaming programs," *ACM SIGPLAN Notices* 40, no. 6 (2005): 281-294.	P54
V. Issarny et al., "Developing ambient intelligence systems: a solution based on Web services," *Automated Software Engineering* 12, no. 1, Autom. Softw. Eng. (Netherlands) (January 2005): 101-37.	P55
Taeho Kim, D. Stringer-Calvert, and Sungdeok Cha, "Formal verification of functional properties of a SCR-style software requirements specification using PVS," *Reliability Engineering & System Safety* 87, no. 3, Reliab. Eng. Syst.	P56

Saf. (UK) (March 2005): 351-63.	
S. Abou-Zahra, “A Data Model to Facilitate the Automation of Web Accessibility Evaluations,” *Electronic Notes in Theoretical Computer Science* 157, no. 2 (2006): 3-9.	P57
A. Ireland et al., “An integrated approach to high integrity software verification,” *Journal of Automated Reasoning* 36, no. 4, J. Autom. Reasoning (Netherlands) (2006): 379-410.	P58
C.-H. Lung et al., “Program restructuring using clustering techniques,” *Journal of Systems and Software* 79, no. 9 (2006): 1261-1279.	P59
Y. Qi, D. Kung, and E. Wong, “An agent-based data-flow testing approach for Web applications,” *Information and Software Technology* 48, no. 12 (2006): 1159-1171.	P60
C. Robinson-Mallett et al., “Extended state identification and verification using a model checker,” *Information and Software Technology* 48, no. 10 (2006): 981-992.	P61
J. Gray, Y. Lin, and J. Zhang, “Automating change evolution in model-driven engineering,” *Computer* 39, no. 2, Computer (USA) (February 2006): 51-8.	P62
V. Sugumaran et al., “A semi-automated filtering technique for software process tailoring using neural network,” *Expert Systems with Applications* 30, no. 2, Expert Syst. Appl. (UK) (February 2006): 179-89.	P63
S. Basu and C.R. Ramakrishnan, “Compositional analysis for verification of parameterized systems,” *Theoretical Computer Science* 354, no. 2, Theor. Comput. Sci. (Netherlands) (March 28, 2006): 211-29.	P64
J.C. Bicarregui, C.A.R. Hoare, and J.C.P. Woodcock, “The verified software repository: a step towards the verifying compiler,” *Formal Aspects of Computing* 18, no. 2, Form. Asp. Comput. (UK) (June 2006): 143-51.	P65
I. Lera, C. Juiz, and R. Puigjaner, “Performance-related ontologies and semantic web applications for on-line performance assessment of intelligent systems,” *Science of Computer Programming* 61, no. 1, Sci. Comput. Program. (Netherlands) (June 2006): 27-37.	P66
Lirong Dai and K. Cooper, “Modeling and performance analysis for security aspects,” *Science of Computer Programming* 61, no. 1, Sci. Comput. Program. (Netherlands) (June 2006): 58-71.	P67
G. Madl, S. Abdelwahed, and D.C. Schmidt, “Verifying distributed real-time properties of embedded systems via graph transformations and model checking,” *Real-Time Systems* 33, no. 1, Real-Time Syst. (Netherlands) (July 2006): 77-100.	P68
Ling Yuan et al., “Generic fault tolerant software architecture reasoning and customization,” *IEEE Transactions on Reliability* 55, no. 3, IEEE Trans. Reliab. (USA) (September 2006): 421-35.	P69
H. Bohnenkamp et al., “MODEST: a compositional modeling formalism for hard and softly timed systems,” *IEEE Transactions on Software Engineering* 32, no. 10, IEEE Trans. Softw. Eng. (USA) (October 2006): 812-30.	P70
Chao Liu et al., “Statistical debugging: a hypothesis testing-based approach,” *IEEE Transactions on Software Engineering* 32, no. 10, IEEE Trans. Softw. Eng. (USA) (October 2006): 831-48.	P71
P. Sawyer, Hong Xu, and I. Sommerville, “Requirement process establishment and improvement from the viewpoint of cybernetics,” *Journal of Systems and Software* 79, no. 11, J. Syst. Softw. (USA) (November 2006): 1504-13.	P72
A. Sillitti, G. Succi, and S. De Panfilis, “Managing non-invasive measurement tools,” *Journal of Systems Architecture* 52, no. 11, J. Syst. Archit. (Netherlands) (November 2006): 676-83.	P73
J. Dehlinger and R.R. Lutz, “PLFaultCAT: a product-line software fault tree analysis tool,” *Automated Software Engineering* 13, no. 1, Autom. Softw. Eng. (Netherlands) (January 2006): 169-93.	P74
C. Yilmaz, M.B. Cohen, and A.A. Porter, “Covering arrays for efficient fault characterization in complex configuration spaces,” *IEEE Transactions on Software Engineering* 32, no. 1, IEEE Trans. Softw. Eng. (USA) (January 2006): 20-34.	P75
T. Chothia and J. Kleijn, “Q-Automata: Modelling the Resource Usage of Concurrent Components,” *Electronic Notes in Theoretical Computer Science* 175, no. 2 (2007): 153-167.	P76
T. Chothia, J. Pang, and M. Torabi Dashti, “Keeping Secrets in Resource Aware Components,” *Electronic Notes in Theoretical Computer Science* 190, no. 3 (2007): 79-94.	P77
G. Delaval and E. Rutten, “A domain-specific language for multitask systems, applying discrete controller synthesis,” *Eurasip Journal on Embedded Systems* 2007 (2007), http://www.scopus.com.proxy.lib.chalmers.se/inward/record.url?eid=2-s2.0-34247259625&partnerID=40&md5=f2e3190e0709202a9b3c706876ae6050.	P78
P.-A. Hsiung, Y.-R. Chen, and Y.-H. Lin, “Model checking safety-critical systems using safecharts,” *IEEE Transactions on Computers* 56, no. 5 (2007): 692-705.	P79
A. Johnstone and E. Scott, “Proofs and pedagogy; science and systems: The grammar tool box,” *Science of Computer Programming* 69, no. 1 (2007): 76-85.	P80
A.D. Lucia et al., “Recovering traceability links in software artifact management systems using information retrieval methods,” *ACM Transactions on Software Engineering and Methodology* 16, no. 4 (2007), http://www.scopus.com.proxy.lib.chalmers.se/inward/record.url?eid=2-s2.0-34648836593&partnerID=40&md5=e0cdd81589839fd29b8ad6e35681397a.	P81
T.C. Oliveira et al., “RDL: A language for framework instantiation representation,” *Journal of Systems and Software* 80, no. 11 (2007): 1902-1929.	P82

C. Sibertin-Blanc, P. Mauran, and G. Padiou, “Safe Adaptation of Component Coordination,” *Electronic Notes in Theoretical Computer Science* 189 (2007): 69-85.	P83
R. Wang, X. Song, and M. Gu, “Modelling and verification of program logic controllers using timed automata,” *IET Software* 1, no. 4 (2007): 127-131.	P84
M. Bozzano and A. Viliafiorita, “The FSAP/NuSMV-SA safety analysis platform,” *International Journal on Software Tools for Technology Transfer* 9, no. 1, Int. J. Softw. Tools Technol. Transf. (Germany) (February 2007): 5-24.	P85
Ping Guo et al., “A hierarchical mixture model for software reliability prediction,” *Applied Mathematics and Computation* 185, no. 2, Appl. Math. Comput. (USA) (February 15, 2007): 1120-30.	P86
Zhenchang Xing and E. Stroulia, “API-evolution support with Diff-CatchUp,” *IEEE Transactions on Software Engineering* 33, no. 12, IEEE Trans. Softw. Eng. (USA) (December 2007): 818-36.	P87
S. Chaki et al., “Verification of evolving software via component substitutability analysis,” *Formal Methods in System Design* 32, no. 3 (2008): 235-266.	P88
F. Deissenboeck et al., “Tool support for continuous quality control,” *IEEE Software* 25, no. 5 (2008): 60-67.	P89
J.A. Díaz-Pace and M.R. Campo, “Experiences with planning techniques for assisting software design activities,” *Applied Intelligence* 29, no. 1 (2008): 56-78.	P90
R. Dömer et al., “System-on-chip environment: A SpecC-based framework for heterogeneous MPSoC design,” *Eurasip Journal on Embedded Systems* 2008, no. 1 (2008), http://www.scopus.com.proxy.lib.chalmers.se/inward/record.url?eid=2-s2.0-49749151366&partnerID=40&md5=b1aa6b8e4be5ed469e67076eb79b1cdc.	P91
A. Ebnenasir, S.S. Kulkarni, and A. Arora, “FTSyn: A framework for automatic synthesis of fault-tolerance,” *International Journal on Software Tools for Technology Transfer* 10, no. 5 (2008): 455-471.	P92
D. Giannakopoulou, C.S. Păsăreanu, and C. Blundell, “Assume-guarantee testing for software components,” *IET Software* 2, no. 6 (2008): 547-562.	P93
S.T. Halkidis et al., “Architectural risk analysis of software systems based on security patterns,” *IEEE Transactions on Dependable and Secure Computing* 5, no. 3 (2008): 129-142.	P94
M. O'Keeffe and M. Ó Cinnéide, “Search-based refactoring for software maintenance,” *Journal of Systems and Software* 81, no. 4 (2008): 502-516.	P95
C.S. Păsăreanu et al., “Learning to divide and conquer: Applying the L*algorithm to automate assume-guarantee reasoning,” *Formal Methods in System Design* 32, no. 3 (2008): 175-205.	P96
K. Popovici et al., “Platform-based software design flow for heterogeneous MPSoC,” *Transactions on Embedded Computing Systems* 7, no. 4 (2008), http://www.scopus.com.proxy.lib.chalmers.se/inward/record.url?eid=2-s2.0-49449088727&partnerID=40&md5=7fed6a3df4f9b9305e3213c1e22f79eb.	P97
T.A. Alspaugh and A.I. Ant, “Scenario support for effective requirements,” *Information and Software Technology* 50, no. 3, Inf. Softw. Technol. (Netherlands) (February 2008): 198-220.	P98
C.S. Gall et al., “Semantic software metrics computed from natural language design specifications,” *IET Software* 2, no. 1, IET Softw. (UK) (February 2008): 17-26.	P99
A. Avenali et al., “Brokering infrastructure for minimum cost data procurement based on quality-quantity models,” *Decision Support Systems* 45, no. 1, Decis. Support Syst. (Netherlands) (April 2008): 95-109.	P100
S.W. Haider et al., “Estimation of defects based on defect decay model: ED3M,” *IEEE Transactions on Software Engineering* 34, no. 3, IEEE Trans. Softw. Eng. (USA) (May 2008): 336-56.	P101
C. Pinello, L.P. Carloni, and A.L. Sangiovanni-Vincentelli, “Fault-tolerant distributed deployment of embedded control software,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits* 27, no. 5, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. (USA) (May 2008): 906-19.	P102
M.A. de Miguel et al., “Integration of safety analysis in model-driven software development,” *IET Software* 2, no. 3, IET Softw. (UK) (June 2008): 260-80.	P103
P. Trinidad et al., “Automated error analysis for the agilization of feature modeling,” *Journal of Systems & Software* 81, no. 6, J. Syst. Softw. (USA) (June 2008): 883-96.	P104
C. Babu and R. Vijayalakshmi, “Metrics-based design selection tool for aspect oriented software development,” *Software Engineering Notes* 33, no. 5, Softw. Eng. Notes (USA) (September 2008): 27.	P105
J. Cleland-Huang, W. Marrero, and B. Berenbach, “Goal-centric traceability: using virtual plumblines to maintain critical systemic qualities,” *IEEE Transactions on Software Engineering* 34, no. 5, IEEE Trans. Softw. Eng. (USA) (October 2008): 685-99.	P106
D. Nabil, A. EL-Korany, and A. Sharaf Eldin, “Towards a suite of quality metrics for KADS-domain knowledge,” *Expert Systems with Applications* 35, no. 3, Expert Syst. Appl. (UK) (October 2008): 654-60.	P107
P. Wriggers et al., “Intelligent support of the preprocessing stage of engineering analysis using case-based reasoning,” *Engineering with Computers* 24, no. 4, Eng. Comput. (UK) (October 2008): 383-404.	P108
R. Colvin, L. Grunske, and K. Winter, “Timed behavior trees for failure mode and effects analysis of time-critical systems,” *Journal of Systems and Software* 81, no. 12, J. Syst. Softw. (USA) (December 2008): 2163-82.	P109
J. Cabot and E. Teniente, “Incremental integrity checking of UML/OCL conceptual schemas,” *Journal of Systems and Software* 82, no. 9 (2009): 1459-1478.	P110

R. Fuentes-Fernandez, J.J. Gomez-Sanz, and J. Pavon, “Requirements elicitation and analysis of multiagent systems using activity theory,” *IEEE Transactions on Systems, Man and Cybernetics, Part A (Systems and Humans)* 39, no. 2, IEEE Trans. Syst. Man Cybern. A, Syst. Humans (USA) (March 2009): 282-98.	P111
H. Eichelberger and K. Schmid, “Guidelines on the aesthetic quality of UML class diagrams,” *Information and Software Technology* 51, no. 12 (2009): 1686-1698.	P112
W. Jirapanthong and A. Zisman, “XTraQue: Traceability for product line systems,” *Software and Systems Modeling* 8, no. 1 (2009): 117-144.	P113
C. Brunette et al., “A metamodel for the design of polychronous systems,” *Journal of Logic and Algebraic Programming* 78, no. 4, J. Log. Algebr. Program. (USA) (April 2009): 233-59.	P114
E. Guerra et al., “Supporting user-oriented analysis for multi-view domain-specific visual languages,” *Information and Software Technology* 51, no. 4, Inf. Softw. Technol. (Netherlands) (April 2009): 769-84.	P115
A.R. Dalton and J.O. Hallstrom, “nAIT: a source analysis and instrumentation framework for nesC,” *Journal of Systems and Software* 82, no. 7, J. Syst. Softw. (USA) (July 2009): 1057-72.	P116
J.C.B. Ribeiro, M.A. Zenha-Rela, and F. Fernandez de Vega, “Test case evaluation and input domain reduction strategies for the Evolutionary Testing of object-oriented software,” *Information and Software Technology* 51, no. 11, Inf. Softw. Technol. (Netherlands) (November 2009): 1534-48.	P117
M. Weiglhofer, G. Fraser, and F. Wotawa, “Using coverage to automate and improve test purpose based testing,” *Information and Software Technology* 51, no. 11, Inf. Softw. Technol. (Netherlands) (November 2009): 1601-17.	P118
F. Molina and A. Toval, “Integrating usability requirements that can be evaluated in design time into model driven engineering of Web information systems,” *Advances in Engineering Software* 40, no. 12, Adv. Eng. Softw. (UK) (December 2009): 1306-17.	P119
Kwang Yong Koh and Poong Hyun Seong, “SMV model-based safety analysis of software requirements,” *Reliability Engineering & System Safety* 94, no. 2, Reliab. Eng. Syst. Saf. (UK) (February 2009): 320-31.	P120
Chao Cai et al., “Global-to-local approach to rigorously developing distributed system with exception handling,” *Journal of Computer Science and Technology (English Language Edition)* 24, no. 2, J. Comput. Sci. Technol., Engl. Lang. Ed. (China) (March 2009): 238-49.	P121
Dan Hao et al., “Test-data generation guided by static defect detection,” *Journal of Computer Science and Technology (English Language Edition)* 24, no. 2, J. Comput. Sci. Technol., Engl. Lang. Ed. (China) (March 2009): 284-93.	P122