redin Revista Facultad de Ingeniería, Universidad de Antioquia, No. 77, pp. 75-87, 2015

An analysis of tools for automatic software development and automatic code generation

Un análisis de herramientas para el desarrollo y generación automática de software y de código

Viviana Yarel Rosales-Morales[1], Giner Alor-Hernández[1]*, Jorge Luis García-Alcaráz[2], Ramón Zatarain-Cabada[3], María Lucía Barrón-Estrada[3]

[1]División de Estudios de Posgrado e Investigación, Instituto Tecnológico de Orizaba. Av. Oriente 9 # 852, Col. Emiliano Zapata. C. P. 94320. Orizaba, México.
[2]Departamento de Ingeniería Industrial y Manufactura, Universidad Autónoma de Ciudad Juárez. Av. del Charro # 450 Norte, Col. Partido Romero. C. P. 32310. Ciudad Juárez, México.
[3]División de Estudios de Posgrado e Investigación, Instituto Tecnológico de Culiacán. Av. Juan de Dios Bátiz 310 Pte. Col. Guadalupe. C. P. 80220. Culiacán, México.

ARTICLE INFO

Received February 27, 2015
Accepted June 24, 2015

KEYWORDS

Software development, code generation, automatic code generation, CASE, IDE

Desarrollo de software, generación de código, generación automática de código, CASE, IDE

ABSTRACT: Software development is an important area in software engineering, which is why a wide range of techniques, methods, and approaches has emerged to facilitate software development automation. This paper presents an analysis and evaluation of tools for automated software development and automatic code generation in order to determine whether they meet a set of quality metrics. Diverse quality metrics were considered such as effectiveness, productivity, safety, and satisfaction in order to carry out a qualitative and quantitative evaluation. The tools evaluated are CASE tools, frameworks, and Integrated Development Environments (IDEs). The evaluation was conducted to measure not only the tools' ability to be employed, but also their support for automated software development and automatic source code generation. The aim of this work is to provide a methodology and a brief review of the most important works to identify the main features of these works and present a comparative evaluation in qualitative and quantitative terms of quality metrics. This would provide software developers with the information they need to decide the tools that can be useful for them.

RESUMEN: El desarrollo de software es una importante área en la ingeniería de software, por tal motivo han surgido técnicas, enfoques y métodos que permiten la automatización de desarrollo del mismo. En este trabajo se presenta un análisis de las herramientas para el desarrollo automático de software y la generación automática de código fuente, con el fin de evaluarlas y determinar si cumplen o no con un conjunto de características y funcionalidades en términos de calidad. Dichas características incluyen eficacia, productividad, seguridad y satisfacción, todo a través de una evaluación cualitativa y cuantitativa. Estas herramientas son 1) herramientas CASE, 2) marcos de trabajo (*frameworks*) y 3) ambientes de desarrollo integrado (IDEs). La evaluación se llevó a cabo con el fin de medir no sólo la capacidad de uso, sino también el apoyo que brindan para el desarrollo de software automático y la generación automática de código fuente. El objetivo de este trabajo es proporcionar una metodología y una breve revisión de los trabajos más importantes para, de esta forma, identificar las principales características de éstos y presentar una evaluación comparativa en términos cualitativos y cuantitativos, con la finalidad de proporcionar la información necesaria para el desarrollador de software que facilite la toma de decisiones al considerar herramientas que le pueden ser útiles.

1. Introduction

Software engineering is an engineering discipline whose goal is the cost-effective development of software systems [1, 2]. In addition to source code generation, there are various techniques, approaches, programming paradigms, and tools for software development and automatic source code generation. Some of the most relevant approaches are FDD (Feature Driven Development), MDA (Model Driven Architecture), UML-based development (Unified Modeling Language-based), and RAD (Rapid Application Development). On the other hand, among the most known programming paradigms for software development are AOP (Aspect-Oriented Programming), OOP (Object-Oriented Programming), Structured Programming, and Components-based Programming. In the same way, the tools for software development involve IDEs (Integrated

* Corresponding author: Giner Alor Hernández
E-mail: galor@itorizaba.edu.mx
ISSN 0120-6230
e-ISSN 2422-2844

DOI: 10.17533/udea.redin.n77a10

Development Environment), CASE tools (Computer-Aided Software Engineering) and IREs (Integrated Reverse-Engineering Environments). All these approaches, programming paradigms, tools, and techniques facilitate software development, and thereby, they improve the profitability of software systems [3]. Two important parts that have been recently implemented are the tools for software development and the tools for automatic code generation. A software development tool is a computer program that software developers use to create, debug, maintain, or otherwise support other programs and applications [4]. The term usually refers to relatively simple programs, which can be combined together to accomplish a task. For instance, one might use multiple hand tools to fix a physical object. The ability to use a variety of tools productively is one hallmark of a skilled software engineer [4].

There are various types of software development tools, such as design software tools, modeling software, encoding software, management software, and reverse engineering tools, among many others. On the one hand, some of the most important and used tools are CASE tools; CASE stands for Computer-Aided Software Engineering. It covers a wide range of different components used to support software process activities, such as requirements analysis, system modeling, debugging, and testing. CASE tools may also include a code generator which automatically generates source code from the system model and some process guidance, which gives advice to the software engineer on what to do next [1, 2]. In general terms, CASE is the application of a set of tools and methods to a software system with the desired end result of high-quality, defect-free, and maintainable software products [5]. On the other hand, in addition to CASE tools, there are other kinds of tools, such as IDE's (Integrated Development Environments) that combine the features of many tools in one package. For instance, these other tools facilitate some tasks, such as searching for content only in files in a particular project. IDEs may, for example, be used for the development of enterprise-level applications [4]. It is important to review these tools to identify their characteristics and evaluate them according to established criteria and desirable characteristics based on quality metrics. This evaluation is highly significant since it will allow software developers to broad their perspective regarding which characteristics and quality metrics are covered by a given tool.

In the field of software development and automatic code generation, several overviews and comparative analysis have been proposed in [3, 6-10]. In addition, some researches and systematic reviews have been conducted regarding these topics [11-19]. Moreover, previous works have addressed software systems development and, in some cases, software automatic generation. On the one hand, both quantitative and qualitative evaluations are highly needed for quality metrics of automatic code generation tools. These quality metrics include Functionality, Reliability, and Usability, among others. On the other hand, although some evaluations have been reported, there is no record of any research or initiative covering the aforementioned quality metrics. Therefore, the aim of this paper is to present a brief review on existing automatic code generation tools, (CASE tools, IDEs, frameworks, and academic tools) in order to propose a hybrid evaluation in quantitative and qualitative terms of quality metrics for each tool analyzed.

For this reason, this paper carried out a literature review of works that currently have an impact on the different techniques and methods for software development and automatic code generation. Thus, one hundred research works were analyzed and ten of them were selected for evaluation. These ten works were considered the most relevant according to the selection criteria; they address its use in software development and automatic code generation area from 1989 to 2014.

The paper is structured as follows: First Section contains the Introduction, Second Section Research Methodology of this review, Third Section presents the review of tools and frameworks for automatic code generation. Section four concerns the Software quality characteristics for software development tools and Section five presents a review of evaluation models and the results obtained for qualitative and quantitative evaluation for Software Development and Automatic Code Generation tools. Finally, in section six conclusions and future work are discussed.

2. Research Methodology

The methodology is composed of three stages. The first stage presents a research of related works to Automatic Code Generation in several academic electronic databases. The second stage presents the classification of these works in the different kinds of tools. Finally, the third stage of the methodology involves the report of a comprehensive literature review that identifies technologies, tools, and frameworks in some of the papers reviewed. As it was previously mentioned, we have performed a detailed search of the major databases of electronic journals for a comprehensive bibliography on Automatic Code Generation. The digital libraries considered were: 1) ACM Digital Library, 2) IEEE Xplore Digital Library, 3) Science Direct (Elsevier) and 4) SpringerLink. Only the papers published by academic journals, workshops, and international conference proceedings were considered reliable and worthy.

Moreover, we also employed a keyword-based search in order to select the most relevant papers. The main keywords employed were: 1) Software Engineering, 2) Code Generation, 3) Software Development Tool, 4) Automatic Code Generation, 5) Source Code Generator Tool, and 6) CASE Tools. Papers that were not directly related to Automatic Code Generation or not suitable for the study were discarded. The research papers selected were manually classified by considering the following criteria: (1) CASE tools, (2) IDEs, and (3) frameworks for Automatic Code Generation. This selection Criteria is depicted in Figure 1.

The section below presents the selected and most relevant papers regarding Automatic Code Generation.

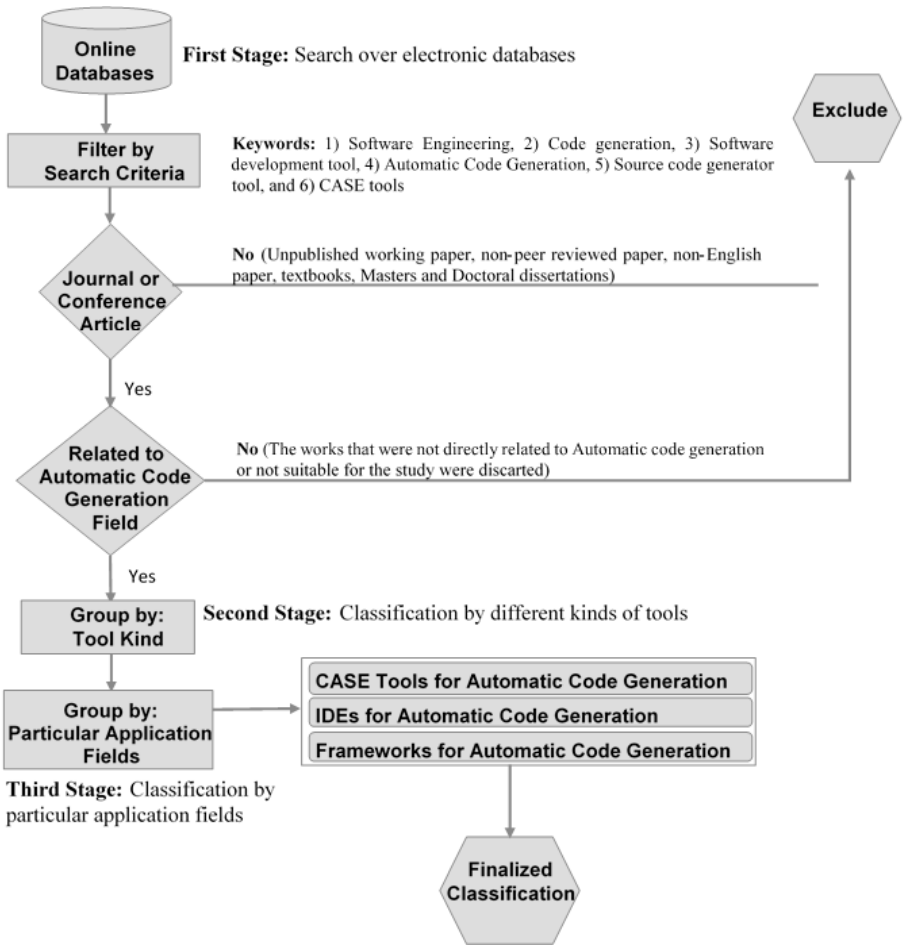

Figure 1 Selection Criteria Flow Diagram

It is worth mentioning that the tools to assess were selected from a large set of candidate tools based on the following criteria: 1) availability of documentation, 2) tools for a specific purpose and a general purpose, and 3) the code generated and programming languages supported.

3. Tools and frameworks for automatic code generation

Recently, several projects on tools for Automatic Code Generation have been developed, some examples of these initiatives are usually 1) CASE tools for Automatic Code Generation, 2) IDEs for Automatic Code Generation, and 3) frameworks for Automatic Code Generation. These initiatives have the purpose to facilitate working software developers. Some of the most important initiatives are presented below.

3.1. CASE tools for Automatic Code Generation

In [20] the determinants for a CASE tool implementation were presented. The findings indicated that an environment including the enforcement of a development methodology and the use of metrics contribute to perceived improvements in quality when using CASE. Furthermore, the use of metrics and consultants along with formal training contribute to perceived improvements in developer productivity. In [21], a survey on software evolution and computer-aided prototyping was provided. According to [21], Computer-Aided Prototyping will become a practical technique in the evolution process. CAPS (Computer-Aided Prototyping System) consists of an integrated set of tools that help design, translate, and execute prototypes. It includes a graph data model for evolution, evolution control system, change merging facility, and automated retrievals for reusable components and it supports the prototyping modeling process. In [18] a tool that generates a code with compatibility for design patterns to maximize reusability of design components was proposed. This tool constructs a library that stores explanation information of pattern and structure information of abstract type. Pattern structure information goes through the process of instantiation, which makes the patterns fit for specific applications. Instantiated structure information is generated as an XMI-based source code through a code generation template. XMI is

supported as a transformed format from most CASE tools, so it is sure for compatibility a code generation tool that is applicable to procedural language-based applications for distributed processing was described in [22]. The application programs along with the partition primitives were converted into independently executable concrete implementations. The process consisted of two steps, first translating the primitives of the application program into equivalent code clusters, and then, scheduling the implementations of these code clusters according to the inherent data dependencies. Furthermore, the original source code needed to be reverse engineered in order to create a meta-data table describing the program elements and dependency trees. The proposed code generation model was implemented using C and tested for various application programs for functional verification. In [23], a tool for programming using schemas was described. In order to solve a given programming problem, the user defined a recurrence relation system, selected the proper schema and the tool automatically generated the code that solved the problem in the target language. In this way, the tool allowed the integration of methodologies based on schemas into the subject of the course. Also, authors of [24] described EDEN as a CASE environment, whose objective is to integrate structure design methodologies, software module libraries, and rigorous testing through the entire software's life cycle. In addition to supporting each phase of the life cycle, EDEN will provide project management tools, such as metrics analysis, configuration management, and quality assurance compliance. Visual Paradigm [25] for UML (VP-UML) is a UML CASE Tool supporting UML 2, SysML and Business Process Modeling Notation (BPMN) from the Object Management Group (OMG). In addition to modeling support, it provides report generation and code engineering capabilities including code generation. It can reverse engineer diagrams from source code, and provide round-trip engineering for various programming languages. PowerDesigner [26] is a collaborative enterprise modelling tool produced by Sybase. PowerDesigner runs under Microsoft Windows as a native application, and runs under Eclipse through a plugin. PowerDesigner supports Model-Driven Architecture software design.

3.2. IDEs for Automatic Code Generation

In [27], authors described the research performed to analyze the requirements for the development of an IDE for embedded system design. The research considered the format and frequency of the data to be transferred within the system and finally the available communication mechanisms. The work concluded with a recommended approach to the development of an IDE for embedded system design. In [28], an overview of Cadena (Cadena stands for Component Architecture Development ENvironment for Avionics systems) - which is an integrated environment for building and modeling systems built using the CORBA (Common Object Request Broker Architecture) Component Model (CCM) was presented. Cadena provided facilities for defining component types using CCM IDL, specifying dependency information and transition system semantics for these types, assembling systems from CCM components, visualizing various dependence relationships between components, specifying and verifying correctness properties of models of CCM systems derived from CCM IDL, component assembly information, and Cadena specifications, and producing CORBA stubs and skeletons implemented in Java. In [29], the authors stated that software development could be eased with an IDE, which allows for using different individual tools from one single development platform. Unfortunately, when developing software for a particular embedded system, the development of an IDE for a certain device can be expensive, since the development of an IDE requires a lot of resources. Therefore, authors proposed the development of an integrated development for a mobile device, Nokia 770 Internet Tablet. The goal was to aim at a fully-fledged IDE with the lowest possible costs. In order to accomplish this, they turned to open source development communities, and targeted the effort to the integration of already existing components into a simple yet practical IDE. In [30], authors proposed an MB-UID (Model-Based User Interface Development) approach for semi-automatic generation of adaptive applications for mobile devices. An environment, called XMobile, offers a device-independent user interface framework and a code generation tool to provide fast development of multi-platform and adaptive applications according to device and platform features. In [31], a new modeling environment called MOSAIC was presented. It combines concepts such as equation-based modeling, use of symbolic mathematic language, and code generation. Moreover, the proposed tool followed a new modeling approach for the re-use of single equations and the support of different naming conventions. The modeling is done strictly in the documentation level. The model information is stored in XML and MathML, and code generation for different programming languages is used to transform the generally defined models into executable programs or suitable code fragments for the solution or use in various numerical environments. Furthermore, MOSAIC is provided as a Software as a Service. The result is a software tool that allows for modeling in the documentation level, promotes the reuse of model elements, and supports centralized cooperation on the Internet. IntelliJ IDEA [32] is a Java-based IDE for developing software. It was developed by JetBrains, and is available as an Apache 2 Licensed community edition, and in a proprietary commercial edition.

3.3. Frameworks for Automatic Code Generation

In [33], a framework for automatic graphical user interface code generation was developed. The authors also developed tools to support this framework: 1) a parser, 2) generation rules, and 3) target code production. The parser read specifications resulted from a reverse engineering process of a character-oriented user interface. These specifications were written in a language called AUIDL (Abstract User Interface Description Language). This language is based on the object-oriented paradigm, which means the use of class, object, attributes, and methods. The methods are used to describe the behavior of the user interface. In [34] the authors presented Tom and ApiGen that are two

complementary tools, which simplify the definition and the manipulation of abstract datatypes. Tom is an extension of Java that adds pattern-matching facilities independently of the used data-structure. ApiGen is a generator of abstract syntax tree implementations that interacts with Tom tool. The authors of Tom and ApiGen demonstrated the integration of an algebraic programming environment in Eclipse, by integrating a Tom editor, an automatic build process, and an error management mechanism. And they explain how Eclipse could be extended to support the development of Tom programs. In [35], the authors stated that Model-Driven Engineering (MDE) was considered one of the most promising approaches for software development. They presented an example based on state-machines that was used to demonstrate the benefits of this approach. After defining a modeling language (meta-model) for state machines, a graphical tool was presented, which was aimed at easing the description and validation of state-machine models. The generated models were used as inputs for the automatic Ada code generation tool, and testing including a simulation program to test the correctness and performance of the implemented application. In [36], a prototype tool called VULCAN that aimed to assist with the creation of high quality code through the use of design patterns was presented. This tool came in the form of a plug-in for Eclipse software development environment. VULCAN facilitates high quality code creation through the automatic generation of design pattern code templates, customized with user input, and integrated into pre-existing projects. By automating the design pattern generation process through the application of a practical and easily usable tool, the adoption of a model-driven engineering approach using design patterns can be substantially mitigated, resulting in improved system quality. In [37] an Automatic Coder using Artificial Intelligence (ACAI) was described. ACAI used an approach to solve automated code generation in routine programming domains. The main components of ACAI are considered the user goals and preferences, a library of abstract programs, and a library of generic code components. ACAI uses a combination of Case-Based Reasoning, Routine Design, and Template-Based Programming approaches to generate complete Java programs that satisfy users' requirements. Adobe Dreamweaver© [38] is a commercial Web development tool developed by Adobe Systems. Adobe Dreamweaver is available for OS©. X and for Windows©.

It is noteworthy that the aforementioned works are the most relevant from a wide range of papers reviewed. These works are of great importance for software development and the area of automatic code generation. However, these works have not been evaluated to quantify their quality and the benefits that provided to developers. In this regard, it is important to perform a quantitative and qualitative evaluation to identify the characteristics of each of the tested tools and present the results in a clear and orderly manner. The following section describes the features that were considered to measure the quality metrics of software development tools.

4. Software quality characteristics for software development tools

This section defines all the quality metrics considered in the evaluation process of software development tools. To carry out the evaluation process, we identified and reviewed the different Software Quality Models [39-41] and Philosophies [42-44]. However, contrary to this paper, these models do not consider the quality metrics of the ISO/IEC 9126 model, which are effectiveness, productivity, safety, and satisfaction. The ISO/IEC 9126 model was employed since it is a widely recognized international standard for quality software evaluation. ISO/IEC 9126 defines a quality model in terms of internal quality, external quality, and quality in use. Internal quality is evaluated by using internal attributes of software, such as modularity. External quality is evaluated when the software is executed, usually during formal testing activities. Quality in use refers to the users' view of the software's quality when they use it in a particular environmental context. ISO/IEC 9126 is composed of four sections. The ISO/IEC 9126-1 section classifies the external and internal quality of the software in a structured set of characteristics. These characteristics are further decomposed into sub-characteristics, which derive into specific attributes. ISO/IEC 9126-2 and ISO/IEC 9126-3 describe the software metrics for external and internal attributes, respectively. ISO/IEC 9126-4 defines the quality in use determined by four characteristics: effectiveness, productivity, safety, and satisfaction. The assessment method presented here is intended to measure the quality of each tool. Quality in use is defined as the capability of the software product for enabling users to achieve specific goals with effectiveness, productivity, safety, and satisfaction in specific contexts of use. The quality model presented in the first part of the standard, ISO/IEC 9126-1 [45] classifies software quality in a structured set of characteristics and sub-characteristics. The characteristics are:

Functionality: A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy explicit or implied needs [45].
Reliability: A set of attributes that bear on the capability of software for maintaining its level of performance under stated conditions for a stated period of time [45].
Usability: A set of attributes that bear on the effort needed for use and the individual assessment of such use by a stated or implied set of users [45].
Efficiency: A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used under stated conditions [45].
Maintainability: A set of attributes that bear on the effort needed to make specified modifications [45].
Portability: A set of attributes that bear on the ability of software to be transferred from one environment to another [45].

Each quality metric is divided in sub-characteristics. This division is presented below in Table 1 [45]:

Table 1 Quality Metrics and sub-characteristics of software quality by standard ISO/IEC 9126-1

Characteristics	Sub-characteristics
Functionality	1. Suitability
	2. Accuracy
	3. In17/2
	4. teroperability
	5. Security
	6. Functionality Compliance
Reliability	1. Maturity
	2. Fault Tolerance
	3. Recoverability
	4. Reliability Compliance
Usability	1. Understandability
	2. Learnability
	3. Operability
	4. Attractiveness
	5. Usability Compliance
Efficiency	1. Time Behaviour
	2. Resource Utilization
	3. Efficiency Compliance
Maintainability	1. Analyzability
	2. Changeability
	3. Stability
	4. Testability
	5. Maintainability Compliance
Portability	1. Adaptability
	2. Installability
	3. Co-Existence
	4. Replaceability
	5. Portability Compliance

Each quality sub-characteristic (e.g. adaptability) is further divided into attributes. An attribute is an entity that can be verified or measured in the software product. Attributes are not defined in the standard because, they vary between different software products [45]. The next section presents the evaluation model used in this study as well as the results obtained.

5. Evaluation and results

Software Engineering literature has proposed several works related to software development evaluation. In [13], a survey of existing research on aspect-oriented modeling and code generation was reported. An orchestrated survey of the most prominent techniques for automatic generation of software test cases was presented in [8]. In [19], a comparison between UML and SystemC was proposed, focusing on communication modeling. Although the absence of a standardized way to evaluate software development tools is still an issue, there are two main approaches: quantitative (objective) evaluations and qualitative (subjective) evaluations. On the one hand, according to [46], quantitative evaluations are based on identifying the effects of using a tool in measurable terms. On the other hand, qualitative evaluations — also known as feature analyses — are based on identifying the requirements that the user possesses to perform a particular task/activity and on linking these requirements to the tool's features that can support the task or tasks.

Moreover, there are other techniques, such as AHP (Analytic Hierarchy Process), which have great advantages when important elements in the decision are difficult to quantify or compare, or when communication among team members is impeded by their different specializations, terminologies, or perspectives. Some decision situations where the AHP can be applied include [47]: Choice, Ranking, Prioritization, Resource allocation, Benchmarking, Quality management, and Conflict resolution [48].

However, none of these situations meets the objective of this work. Therefore, it was decided to use a two-part evaluation of the tools and frameworks for automatic code generation: Tom and ApiGen, Laika, VULCAN, MB-UID, MOSAIC, ACAI, Visual Paradigm©, PowerDesigner©, IntelliJ IDEA©, and Adobe Dreamweaver©. The first part concerns a qualitative method to measure diverse aspects of these tools, by focusing on their usability as well as on the support for the standard features and the compliance with the principles of the use of best practices. The second part refers to a quantitative method to measure the quality in use of each tool. On the one hand, for the qualitative evaluation method, a set of 13 desired features for code generation tools were identified. These features were assessed in all the selected tools in order to determine their legitimacy and, at the same time, identify which of the tools would be best in specific circumstances. Because this qualitative analysis requires subjective measures, a discussion is presented in the following paragraphs with the aim of supporting the results. On the other hand, for the quantitative evaluation method, four software metrics were proposed based on the internal metrics defined by the ISO/IEC 9126 standard.

5.1. Evaluation design

This assessment method describes each of the three aspects that constitute the needs of a software developer; it also describes the features that a tool for software development, especially for source code generation, must possess in order to satisfy each of the identified needs.

The scale used for the measurement of the identified aspects is a 3-point Likert scale [49] in which "3" represents the best score and "1" represents the worst score as it is represented below:

- 3 points: strongly addressed (S.A.)
- 2 points: partially addressed (P.A.)
- 1 point: not addressed (N.A.)

For each of the three aspects, the final score will be the highest score assigned by a member of the evaluation team composed of two software engineers, two graphic designers, and two software developers. Finally, the overall evaluation for each software tool will be the sum of the final scores in the three aspects.

5.2. Qualitative evaluation

A usability evaluation approach is presented inspired by diverse proposals analyzed and based on a weighted matrix. This assessment method has a long tradition within software engineering and information systems literature [50]; moreover, it has been used for other quality evaluations [51, 52]. For this weighted matrix, we have proposed the evaluation of five aspects in order to assess the legitimacy of the proposed software tools. These aspects were selected from the field of software quality by the evaluation team. Each qualitative aspect had a score based on a 3-point Likert scale [49]. The aspects are presented in the following text.

Aspect 1: GUI design

Factor 1. GUIs with simplicity and predictability principles
Factor 2. User interaction based on the best practices

An appropriate GUI design encourages an easy, natural, and engaging interaction between a user and a system, and it allows users to carry out their tasks. Considering the type of software tool that this application is, we selected the 'simplicity' and 'predictability' GUI design principles proposed in [53] to evaluate the ease and naturalness of the GUI design of the code generation tools. The principle of simplicity allows users to understand and use a system easily, regardless of his or her computational experience and level of concentration when using the application. Predictability allows users to anticipate the natural progression of software development.

Aspect 2: Tools usability

Factor 1. Ease of use
Factor 2. Ease of learning
Factor 3. Technical knowledge and skills

ISO 9241-11 [54] defines usability as the extent to which a product can be used by specific users in order to achieve specific goals with effectiveness, efficiency, and satisfaction in a specific context. However, because of the quantitative nature of the 'effectiveness' and 'efficiency' factors, we have only considered the 'satisfaction' factor mentioned in the former definition in order to evaluate the usability of the application. ISO 9241-11 also defines satisfaction as the absence of discomfort, as well as the positive attitudes of users toward the use of a product.

Aspect 3: Features promoting use

Factor 1. Use of modularity principles
Factor 2. Functions with well-defined purposes

This aspect first refers to the presence of features that promote the use of the application, such as modularity. Nevertheless, it also refers to whether the application was developed with a well-defined purpose. The factor of modularity reflects whether the software contains modules, each one representing a concrete set of concepts able to be reused in other environments. Modularity highly increases the number of possible combinations in a system, which makes it much more flexible [55]. Well-defined purposes guarantee the quality of the activities carried out through a software tool. This provides users the tools needed for a specific activity and avoids unnecessary functions that would not be used or missing features that hinder the use of the tool or reduce the acceptance and use of the application.

Aspect 4: Support for heterogeneous data sources integration

Factor 1. Interoperability with external software tools
Factor 2. Data security

ISO 9241-11 [54] defines interoperability as the ability of the software product to interact with one or more specified systems. The interoperability factor defines whether the software can be easily combined to enhance its capabilities. As far as data security is concerned, interoperability refers to the ability of a software tool to protect data in order to avoid unauthorized persons or systems to read or modify this data. Data security covers both data stored by the systems and data transmitted by the systems. Security is a critical factor to consider in software development because these applications can contain confidential information for both individuals and enterprises.

Aspect 5: Support and documentation

Factor 1. Documentation
Factor 2. Active roles of users
Factor 3. Inline help
Factor 4. On line developer documentation

This aspect refers to the presence of information, such as technical manual, user manuals, and other instructions that facilitate the use and operation of a tool. Documentation can be aimed at developers, and in this case, it contains information on the operations of the software system. However, it can also be aimed at end users with the purpose of facilitating the interaction between the end users and the tool (e.g., a training manual). Furthermore, Web applications emphasize on the active involvement of users in order to improve the tool. This involvement includes activities such as bug reporting, suggestion of new functions, and the implementation of new features through software development kits.

5.3. Quantitative Evaluation

As it was previously mentioned, quality in use is defined as the ability of the software product to enable users to achieve specific goals with effectiveness, productivity, safety, and

satisfaction in specific contexts of use. In order to measure the quality in use of each code source generation tool, the productivity characteristic was selected. Productivity refers to the ability of the software product to enable users to expend appropriate amount of resources in relation to the effectiveness achieved in a specific context of use. Some of the metrics covered by this characteristic are task time, waiting time, task efficiency, and help frequency, among others.

The metrics that we have proposed for this quantitative evaluation focus on the measurement of the quality of an automatic code generation tool to generate source code. Therefore, they are based on internal metrics defined in the ISO/IEC 9126 standard. They are described below:

Quality in use

Productivity
Task time: The estimated time of work spent to develop a basic application.
Help frequency: The frequency of use of the help and/or documentation tools.

Maintainability

Changeability
Modification complexity: The estimated work time spent on changing data sources of the applications already generated.
Analyzability
Inline documentation completeness: Ratio of the number of scripts, functions, or variables having documentation to the number of implemented scripts, functions, or variables.

In order to measure the quality metrics by each of the evaluated tools under the same circumstances, we have outlined a source code generation scenario as the 'hello world'. It is also a means to support the results obtained from the qualitative analysis, or at least to support the results corresponding to the aspects covered in this source code generation scenario. It is worth mentioning that this scenario does not involve the measurement of neither the software environmental adaptability nor the hardware environmental adaptability, since these metrics focus on evaluating standalone applications.

6. Results

Table 2 and Figure 2 depict the score obtained from the evaluation process of each aspect analyzed for the qualitative evaluation. Table 3 shows the results of the quantitative evaluation.

6.1. Discussion

The results obtained for the qualitative evaluation are presented below.

Aspect 1: GUI design

Most of the software development tools evaluated provide GUIs of different degrees of simplicity and predictability. On the one hand, tools such as Tom and ApiGen, Laika, XMobile and VULCAN provide interfaces that permit the design and modeling of system software, which are configured with the parameters required. Although these interfaces are too simple and do not provide anything beyond this, they allow users to perform easily and naturally the tasks that they need to accomplish. On the other hand, tools such as Visual Paradigm©, PowerDesigner© and Adobe Dreamweaver© provide a set of GUI components aesthetically pleasing that encapsulate business logic and implement interaction design patterns, such as wizards and input feedback, to mention a few. All these components allow users to easily understand and use the tool regardless of their experience and level of concentration when using it. As far as IntelliJ IDEA© is concerned, although it does provide a graphic user interface, this interface is neither intuitive nor easy to use. In fact, users require certain level of expertise in the use of IDEs and development tools. On the other hand, MOSAIC is a special-purpose tool; therefore, it is limited to users' experience.

Aspect 2: Tools usability

In general terms, the automatic code generation tools presented earlier have intuitive designs and demand little learning time for beginners. For instance, Adobe Dreamweaver© provides a wizard that permits dragging and dropping elements into the main panel. This characteristic is very easy to use since the developer does not need to know the programming language. Visual Paradigm©, however, can generate code in languages such as Java or C++ from a UML class diagram, and the user merely needs to know how to make UML diagrams. Also, on the one hand, Laika tool provides a graphical interface to generate the source code of whatever the user wishes to design; however, the code is generated for a particular device.

On the other hand, IntelliJ IDEA© and PowerDesigner© employ codification standards from the design environments, while MOSAIC is a more intuitive tool both in use and functioning, because it is a special-purpose system and has more limited options to generate source code. XMobile is focused on mobile applications development, and users must possess some knowledge of the applications they develop in order to use it. VULCAN is a plug-in for Eclipse, which makes it stick to the standards of the own IDE.

Aspect 3: Features promoting use

The use of modularity principles and functions with well-defined purposes is covered completely in the commercial tools; however, it is not addressed comprehensively in academic tools, since most of these remained in development or as prototypes; therefore, their use is not massively exploited. Software development tools such as Visual Paradigm© represent a clear example of modularity,

Table 2 Qualitative Evaluation Results

		Tom and ApiGen		Laika		VULCAN		MB-UID XMobile		MOSAIC		ACAI		Visual Paradigm©		PowerDesigner ©		IntelliJ IDEA©		Adobe Dreamweaver©	
Aspect		S	I	S	I	S	I	S	I	S	I	S	I	S	I	S	I	S	I	S	I
A1	F1	2	P.A.	3	S.A.	2	P.A.	2	P.A.	2	P.A.	2	P.A.	3	S.A.	3	S.A.	2	P.A.	3	S.A.
	F2	2	P.A.	3	S.A.	2	P.A.	2	P.A.	2	P.A.	2	P.A.	3	S.A.	3	S.A.	2	P.A.	3	S.A.
A2	F1	2	P.A.	2	P.A.	2	P.A.	2	P.A.	2	P.A.	2	P.A.	2	P.A.	2	P.A.	2	P.A.	3	S.A.
	F2			2	P.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	3	S.A.	3	S.A.	2	P.A.	3	S.A.
	F3	1	N.A.																		
				2	P.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	3	S.A.	3	S.A.	2	P.A.	3	S.A.
		1	N.A.																		
A3	F1	1	N.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	3	S.A.	2	P.A.	3	S.A.	3	S.A.
	F2																				
		1	N.A.	2	P.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	3	S.A.	2	P.A.	3	S.A.	3	S.A.
A4	F1	1	N.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	3	S.A.	2	P.A.	2	P.A.	2	P.A.
	F2																				
		1	N.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	3	S.A.	2	P.A.	2	P.A.	2	P.A.
A5	F1	1	N.A.	2	P.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	3	S.A.	2	P.A.	2	P.A.	3	S.A.
	F2			1	N.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	3	S.A.	2	P.A.	2	P.A.	2	P.A.
	F3	1	N.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	3	S.A.	2	P.A.	2	P.A.	3	S.A.
	F4																				
		1	N.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	1	N.A.	3	S.A.	2	P.A.	2	P.A.	3	S.A.
		1	N.A.																		
Results		16		22		16		16		16		16		38		30		28		36	

S = Score; I = Interpretation; S.A. = strongly addressed; P.A. = partially addressed; N.A. = not addressed

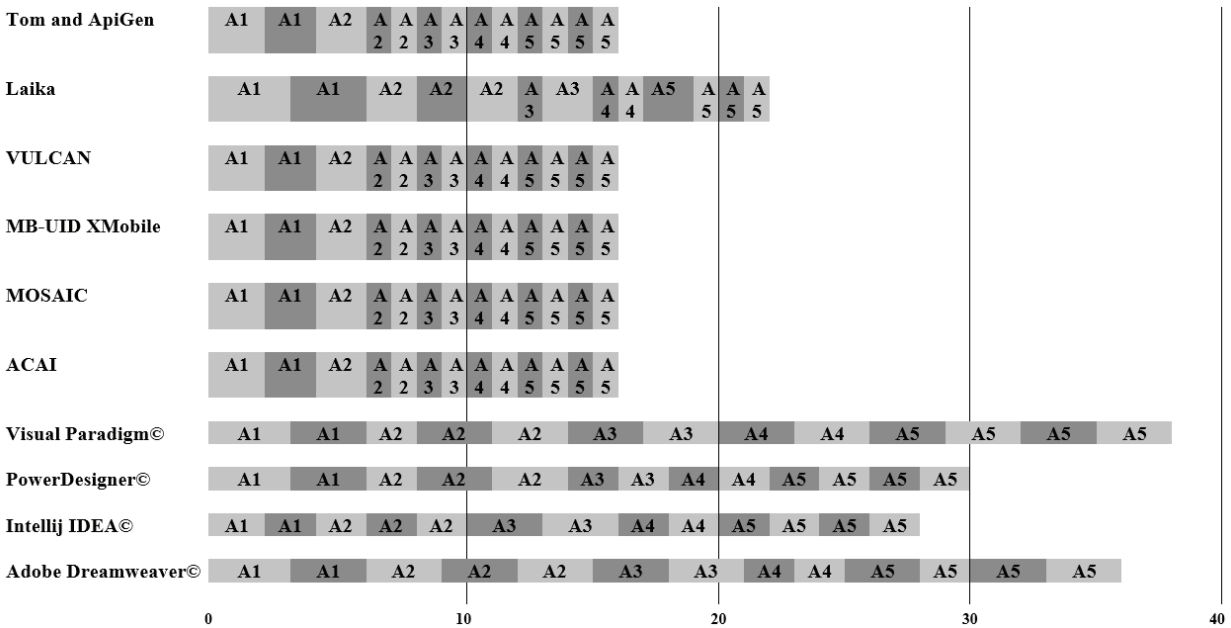

Figure 2 Qualitative Evaluation Results

since different modules are in charge of different tasks and all together form a complete tool. For instance, whilst one module may be responsible for modeling in UML, another one can generate the source code and a third one may be responsible for reverse engineering. On the other hand, some tools such as VULCAN were developed as plug-ins for other tools such as Eclipse. However, this does not mean they are independent since they completely rely on those tools for which they were designed, and this breaks the principle of modularity, to some extent. In the cases of MOSAIC, VULCAN, XMobile, and Laika, they represent tools for a well-defined purpose. MOSAIC focuses on Mathematics, VULCAN on design patterns, and XMobile and Laika are targeted to applications for mobile devices.

Aspect 4: Support for heterogeneous data sources integration

The tools analyzed provide support for several data sources, such as XML, XMI and Java. Tools like Visual Paradigm© can generate source code in different programming languages and this allows them to integrate with other tools such as IDEs like NetBeans© or Eclipse©. Laika,

however, is not supported since it is a special-purpose tool; it was developed for a particular device and programming language. Also, VULCAN cannot integrate itself with other systems, because it was developed as a plug-in for Eclipse. Interoperability is a difficult aspect to cover in these tools, especially if they are special-purpose tools such as Laika, MOSAIC, and XMobile.

Aspect 5: Support and documentation

For most of the automatic generation tools analyzed, a great deal for information could be gathered, such as their user manuals, introductory videos, tutorials, technical manuals, and published research papers. This facilitates both software learning and development. Moreover, the tools emphasize on the active involvement of their users in activities such as bug reporting, suggestion of new functions and the implementation of new features through the software development kits provided. In order to carry out these tasks, tools such as Visual Paradigm©, PowerDesigner©, IntelliJ IDEA©, and Adobe Dreamweaver© provide help and support to community through blogs, forums, and wikis. It is noteworthy that it is easier to find information about commercial tools. For the other tools that are Tom and ApiGen, Laika, VULCAN, XMobile, MOSAIC, and ACAI, it is more difficult to find information and help, especially because they are special-purpose tools. MOSAIC, for instance, has less impact because of this.

According to these results and as can be seen in Table 2, Visual Paradigm© is the tool with the best features for automatic software development, since it was rated 38 of 39 in the qualitative evaluation. As it can be inferred, Visual Paradigm© completely meets most of the features that an automatic software development tool must possess in order to integrate data from heterogeneous data sources and have attractive visual interfaces. Also, Visual Paradigm© can be used by both non-experienced and experienced users, as well as by enterprises demanding a high level of security for network and data. In addition, Visual Paradigm© emphasizes on the active involvement of its users, and this enables tools designers to know the customers' problems and needs in order to develop components that can help them solve their problems.

The results obtained for the quantitative evaluation are presented below.

Table 3 shows the results obtained for each quality metric from the quantitative evaluation. The tools presented earlier contain different components for the generation of source code. In order to generate the source code for a set of Java-based classes by using Visual Paradigm©, it is only required to design a UML class diagram representing the desirable software. As far as Adobe Dreamweaver© is concerned, fewer steps are required in the software development, since at the same time that the components are dragged to the main panel, the corresponding code is displayed. Adobe Dreamweaver© is able to generate codes in various programming languages, such as HTML or PHP. Nowadays, automatic code generation plays an important role in software development, since it helps save time and facilitates the use of tools and programming languages. However, it is also important that the tools meet a set of quality characteristics so the developer can feel confident when using them. In many cases, tools for automatic code generation have been proposed and developed in the form of prototypes, but these prototypes have failed to develop commercially, making them obsolete if they are compared to other tools that are being constantly developed and updated. Therefore, the analysis presented in this paper can demonstrate with the results obtained that Visual Paradigm© is the tool that meets more features than any other tool analyzed. Visual Paradigm© is a commercial tool.

Authors wish to express we do not intent to state which development tool is better or to tell software developers which tool they must employ. The merely purpose of this research paper is to emphasize the main features of each development tool.

Table 3 Assessment Results of Quality Metric

Tool	Components required	Steps necessary to generate source code	Time spent on generating source code (sec)	Steps necessary to modify the source code generated	Time spent on modifying the source code generated (sec)
Tom and ApiGen	5	3	<30	5	<35
Laika	3	5	<50	6	<45
VULCAN	2	6	<45	6	<50
MB-UID XMobile	3	7	<30	5	<65
MOSAIC	4	5	<35	5	<90
ACAI	5	4	<30	3	<30
Visual Paradigm©	7	4	<20	3	<20
PowerDesigner©	6	5	<20	3	<25
IntelliJ IDEA©	7	5	<20	4	<20
Adobe Dreamweaver©	5	3	<15	2	<10

7. Conclusions and future work

This paper has presented an evaluation of tools and frameworks for automatic software development and automatic source code generation. These kinds of reviews and subsequent tools assessments are extremely important issues for software developers to identify the characteristics and functionalities that cover the tools evaluated. During the evaluation and review of the tools, it was possible to identify their characteristics and determine their quality according to the quality model of the ISO/IEC 9126 standard. This evaluation is crucial since it involves different types of tools: CASE tools, IDE's, frameworks and academic tools. All of them were evaluated under the same parameters and compared with one another. The limitations of this paper could be addressed as future work by including more databases - such as Scopus (Elsevier), Web of Science, or CiteSeerX - that would allow for the evaluation of a larger amount of works. Also, a future study could expand the evaluation of tools by including other qualitative and quantitative characteristics in the evaluation process. Some features that could be considered are: correctness of the generated code, required computer resources, and integration with other tools. Another aspect that could be improved in upcoming research is the evaluation of the source code generated by the tools evaluated. This way, researchers can perform a particular assessment according to the tool type, such as assessments by the type of application generated, by supported programming languages, or by input/output parameters. Finally, this study can be extended by considering other IDEs and tools - such as Aptana© and Komodo© - as well as other tool features such as a debugger, a compiler, or a data dictionary, among others.

8. Acknowledgements

This work is supported by Tecnológico Nacional de Mexico (TecNM). This paper was also sponsored by the National Council of Science and Technology (CONACYT) and the Ministry of Public Education (SEP) through PRODEP.

9. References

1. I. Sommerville and P. Sawyer, *Requirements engineering: a good practice guide,* 1st ed. New York, USA: John Wiley & Sons, Inc., 1997.
2. I. Sommerville, *Software Engineering. International computer science series,* 8th ed. Boston, USA: Addison-Wesley Longman Publishing Co., Inc., 2004.
3. C. Zapata and J. Chaverra, "Una mirada conceptual a la generación automática de código", *Rev. EIA. Esc. Ing. Antioq,* no. 13, pp. 155-169, 2010.
4. B. Kernighan and P. Plauger, *Software tools in Pascal*, 1st ed. Boston, USA: Addison-Wesley, 1981.
5. D. Kuhn, "Selecting and effectively using a computer aided software engineering tool," in *Annual Westinghouse computer symposium*, Pittsburgh, USA, 1989.
6. L. Agner, I. Soares, P. Stadzisz and J. Simão, "A Brazilian survey on UML and model-driven practices for embedded software development", *Journal of Systems and Software*, vol. 86, no. 4, pp. 997-1005, 2013.
7. S. Pierucci and E. Ranzi, "A review of features in current automatic generation software for hydrocarbon oxidation mechanisms", *Comput. Chem. Eng.*, vol. 32, no. 4-5, pp. 805-826, 2008.
8. S. Anand *et al.*, "An Orchestrated Survey of Methodologies for Automated Software Test Case Generation", *J. Syst. Softw.*, vol. 86, no. 8, pp. 1978-2001, 2013.
9. R. Novais, A. Torres, T. Mendes, M. Mendonça, and N. Zazworka, "Software evolution visualization: A systematic mapping study", *Inf. Softw. Technol.*, vol. 55, no. 11, pp. 1860-1883, 2013.
10. S. Thomas, B. Adams, A. Hassan and D. Blostein, "Studying software evolution using topic models", *Sci. Comput. Program.*, vol. 80, Part B, pp. 457-479, 2014.
11. U. Kanewala and J. Bieman, "Testing Scientific Software: A Systematic Literature Review", *Inf. Softw. Technol.*, vol. 56, no. 10, pp. 1219-1232, 2014.
12. A. Magdaleno, C. Werner and R. Araujo, "Reconciling software development models: A quasi-systematic review", *J. Syst. Softw.*, vol. 85, no. 2, pp. 351-369, 2012.
13. A. Mehmood and D. Jawawi, "Aspect-oriented model-driven code generation: A systematic mapping study", *Inf. Softw. Technol.*, vol. 55, no. 2, pp. 395-411, 2013.
14. D. Alonso, J. Pastor, P. Sánchez, B. Álvarez and C. Vicente, "Generación Automática de Software para Sistemas de Tiempo Real: Un Enfoque basado en Componentes, Modelos y Frameworks", *Rev. Iberoam. Automática e Informática Ind. RIAI*, vol. 9, no. 2, pp. 170-181, 2012.
15. C. Yang, V. Vyatkin and C. Pang, "Model-Driven Development of Control Software for Distributed Automation: A Survey and an Approach", *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 44, no. 3. pp. 292-305, 2014.
16. H. Liao, J. Jiang and Y. Zhang, "A Study of Automatic Code Generation", in *2010 International Conference on Computational and Information Sciences* (ICCIS), Chengdu, China, 2010, pp. 689-691.
17. M. Jiménez and M. Piattini, "Problems and Solutions in Distributed Software Development: A Systematic Review", in *2nd International Conference on Software Engineering Approaches for Offshore and Outsourced Development* (SEAFOOD), Zurich, Switzerland, 2008, pp. 107-125.
18. Y. Seo and Y. Song, "A Study on Automatic Code Generation Tool from Design Patterns Based on the XMI", in *2006 International Conference on Computational Science and Its Applications* (ICCSA), Glasgow, UK, 2006, pp. 864-872.
19. P. Andersson and M. Höst, "UML and SystemC - A Comparison and Mapping Rules for Automatic Code Generation", in *Embedded Systems Specification and Design Languages*, E. Villar (ed). Amsterdam, Netherlands: Springer, 2008, pp. 199-209.

20. R. Urwiler, N. Ramarapu, R. Wilkes and M. Frolick, "Computer-aided software engineering: The determinants of an effective implementation strategy", *Inf. Manag.*, vol. 29, no. 4, pp. 215-225, 1995.
21. Y. Jing, "Research on computer-aided prototyping system and software evolution", *J. Zhejiang Univ. Sci. A*, vol. 1, no. 4, pp. 384-387, 2000.
22. V. Sairaman, N. Ranganathan and N. Singh, "An automatic code generation tool for partitioned software in distributed systems", in *19th International Conference on VLSI Design. Held jointly with 5th International Conference on Embedded Systems and Design,* Hyderabad, India, 2006, pp. 477-480.
23. A. Gavilanes, P. Martín and R. Torres, "A Tool for Automatic Code Generation from Schemas", in *9th International Conference on Computational Science* (ICCS), Baton Rouge, USA, 2009, pp. 63-73.
24. S. Erdogan, S. McFarr and D. Maglidt, "EDEN: an integrated computer-aided software engineering environment", in *8th Annual International Phoenix Conference on Computers and Communications,* Scottsdale, USA, 1989, pp. 349-353.
25. Visual Paradigm International, *Company*. [Online]. Available: http://www.visual-paradigm.com/aboutus/. Accessed on: Jan. 20, 2015.
26. PowerDesigner, *SAP Sybase PowerDesigner 16.5,* 2013. [Online]. Available: http://www.powerdesigner.de/en/. Accessed on: Jan. 20, 2015.
27. L. Maguire, T. McGinnity and L. McDaid, "Issues in the development of an integrated environment for embedded system design: Part B: design and implementation", *Microprocess. Microsyst.*, vol. 23, no. 4, pp. 199-206, 1999.
28. A. Childs *et al.*, "Cadena: An Integrated Development Environment for Analysis, Synthesis, and Verification of Component-Based Systems", in *7th International Conference on Fundamental Approaches to Software Engineering* (FASE), Barcelona, Spain, 2004, pp. 160-164.
29. J. Jarvensivu, M. Kosola, M. Kuusipalo, P. Reijula and T. Mikkonen, "Developing an Open Source Integrated Development Environment for a Mobile Device", in *International Conference on Software Engineering Advances* (ICSEA), Tahiti, France, 2006, p. 55.
30. W. Viana and R. Andrade, "XMobile: A MB-UID environment for semi-automatic generation of adaptive applications for mobile devices", *J. Syst. Softw.*, vol. 81, no. 3, pp. 382-394, 2008.
31. S. Kuntsche, T. Barz, R. Kraus, H. Arellano and G. Wozny, "MOSAIC a web-based modeling environment for code generation", *Comput. Chem. Eng.*, vol. 35, no. 11, pp. 2257-2273, 2011.
32. JetBrains, *IntelliJ IDEA, The Most Intelligent Java IDE,* 2015. [Online]. Available: https://www.jetbrains.com/idea/. Accessed on: Jan. 20, 2015.
33. A. Elwahidi and E. Merlo, "Generating user interfaces from specifications produced by a reverse engineering process", in *2nd Working Conference on Reverse Engineering*, Toronto, Canada, 1995, pp. 292-298.
34. J. Guyon, P. Moreau and A. Reilles, "An Integrated Development Environment for Pattern Matching Programming", *Electron. Notes Theor. Comput. Sci.*, vol. 107, pp. 33-49, 2004.
35. D. Alonso, C. Vicente, P. Sánchez, B. Álvarez and F. Losilla, "Automatic Ada Code Generation Using a Model-Driven Engineering Approach", in *12th Ada-Europe International Conference on Reliable Software Technologies,* Geneva, Switzerland, 2007, pp. 168-179.
36. G. Frederick, P. Bond and S. Tilley, "VULCAN: A Tool for Automatically Generating Code from Design Patterns", in *2nd Annual IEEE Systems Conference*, Montreal, Canada, 2008, pp. 1-4.
37. Y. Danilchenko, "Automatic Code Generation Using Artificial Intelligence", Ph.D. dissertation, Northern Kentucky University, Kentucky, USA, 2012.
38. Adobe, *Adobe Dreamweaver CC,* 2015. [Online]. Available: http://www.adobe.com/mx/products/dreamweaver.html. Accessed on: Jan. 20, 2015.
39. J. Mccall, P. Richards and G. Walters, "Factors in Software Quality: Concept and Definitions of Software Quality", Air Force Systems Command, Rome Air Development Center, New York, USA, Final Tech. Rep. RADC-TR-77-369, Nov. 1977.
40. B. Boehm, J. Brown and M. Lipow, "Quantitative Evaluation of Software Quality", in *2nd International Conference on Software Engineering* (ICSE), Los Alamitos, USA, 1976, pp. 592-605.
41. R. Dromey, "Cornering the chimera", *IEEE Softw.*, vol. 13, no. 1, pp. 33-43, 1996.
42. P. Crosby, *Quality is Free: The Art of Making Quality Certain*, 1st ed. Michigan, USA: McGraw-Hill, 1979.
43. W. Deming, *Out of the Crisis*, 1st ed. Massachusetts, USA: The Mit Press, 2000.
44. A. Feigenbaum, *Total quality control: Achieving productivity, market penetration and advantage in the global economy*, 4th ed. New York, USA: McGraw-Hill Education, 2005.
45. International Organization for Standardization (ISO), *Software engineering - Product quality - Part 1: Quality Model,* ISO/IEC 9126-1:2001, 2001.
46. B. Kitchenham, S. Linkman and D. Law, "DESMET: a methodology for evaluating software engineering methods and tools", *Computing & Control Engineering Journal*, vol. 8, no. 3. pp. 120-126, 1997.
47. E. Forman and S. Gass, "The Analytic Hierarchy Process - An Exposition", *Oper. Res.*, vol. 49, no. 4, pp. 469-486, 2001.
48. T. Saaty and K. Peniwati, *Group Decision Making: Drawing Out and Reconciling Differences*, 1st ed. Pittsburgh, USA: RWS Publications, 2008.
49. R. Likert, "A technique for the measurement of attitudes", *Archives of psychology*, vol. 22, no. 140, pp. 5-55, 1932.
50. A. Sutcliffe, N. Maiden, S. Minocha and D. Manuel, "Supporting scenario-based requirements engineering", *IEEE Transactions on Software Engineering*, vol. 24, no. 12, pp. 1072-1088, 1998.
51. M. Paredes, G. Alor, A. Rodríguez, R. Valencia and E. Jiménez, "A systematic review of tools, languages, and methodologies for mashup development", *Softw. Pract. Exp.*, vol. 45, no. 3, pp. 365-397, 2015.
52. L. Colombo, G. Alor, A. Rodríguez and R. Colomo, "Alexandria: A Visual Tool for Generating Multi-device Rich Internet Applications", *J. Web Eng.*, vol. 12, no. 3-4, pp. 317-359, 2013.

53. W. Galitz, *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*, 2nd ed. Ed. New York, USA: John Wiley & Sons, Inc., 1997.
54. International Organization for Standardization (ISO), *Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11: Guidance on usability*, ISO 9241-11:1998, 1998.
55. A. Schilling, "Toward A General Modular Systems Theory and Its Application to Interfirm Product Modularity", *Academy of Management Review,* vol. 25, no. 2, pp. 312-334, 2000.