International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org
Volume 6, Issue 6, June 2017 **ISSN 2319 - 4847**

An Object-Oriented Design and Analysis Environment

Deepak Kumar

Assistant Professor, Department Of Computer Science
And Engineering, Guru Gobind Singh Educational Society's
Technical Campus, Bokaro Steel City, India

Abstract
Although object-oriented systems development (OOSD) has greatly matured over the past decade, the jury is still out on its ultimate impact on IS organizations. It's a structured method for analyzing, designing a system by applying the object-orientated concepts, and develop a set of graphical system models during the development life cycles of the software. While OOAD is viewed by many as the best available solution to the ongoing "software crisis," some caution that OOSD may be so complex that it will never become a mainstream methodology. Of particular importance to successful OOSD is object-oriented analysis and design (OOAD), the cornerstone of any serious systems development project. This paper reviews a wide range of empirical studies on OOAD involving human subjects, many with conflicting results. A critique of the research methodologies employed and a discussion of future needed research are presented.

Keywords:OOAD, Domain Modeling, Trade Show, Modeling, Conclusions and Future.

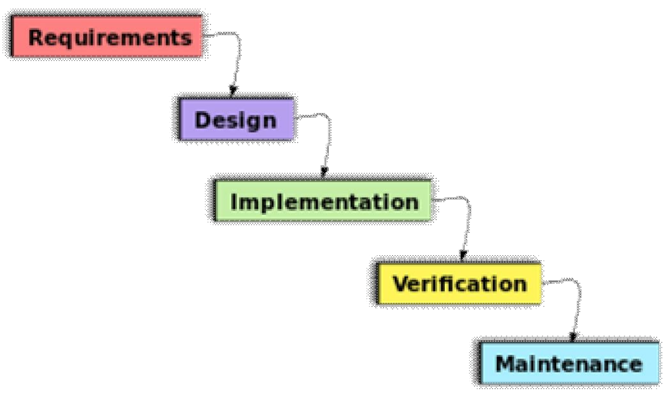
1. INTRODUCTION

Object-oriented (OO) analysis and design (A&D) are important activities of the software development life cycle. OO analysis aims to identify the real problem of the organization for which the application software is built or enhanced, and specify requirements or system capabilities to solve the real problem. OO design develops ideas and software solutions to be implemented to deliver the capabilities. This is sometimes referred to as "programming in the large" — design of system architecture and high-level algorithms. OOAD are crucial activities because the software, if implemented according to poor requirements, will not meet the organization's business needs. Poor design will drastically increase testing and maintenance costs.

2. INTEGRATED OOAD ENVIRONMENT OVERVIEW

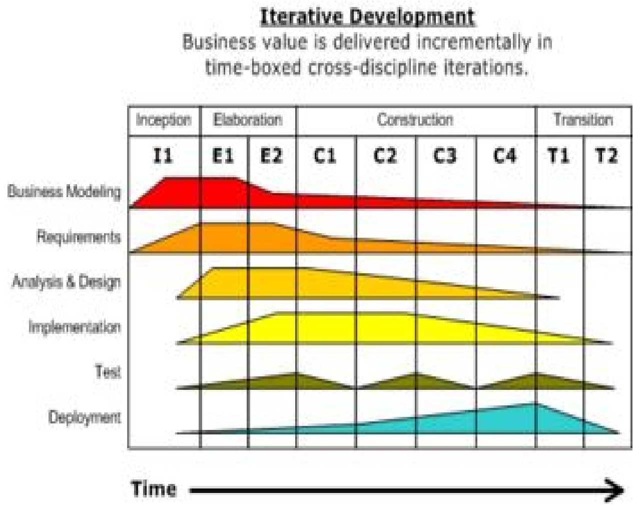
The software life cycle is typically divided up into stages going from abstract descriptions of the problem to designs then to code and testing and finally to deployment. The earliest stages of this process are analysis and design. The analysis phase is also often called "requirements acquisition".

In some approaches to software development—known collectively as waterfall models—the boundaries between each stage are meant to be fairly rigid and sequential. The term "waterfall" was coined for such methodologies to signify that progress went sequentially in one direction only, i.e., once analysis was complete then and only then was design begun and it was rare (and considered a source of error) when a design issue required a change in the analysis model or when a coding issue required a change in design.



The alternative to waterfall models are iterative models. This distinction was popularized by Barry Boehm in a very influential paper on his Spiral Model for iterative software development. With iterative models it is possible to do work in various stages of the model in parallel. So for example it is possible—and not seen as a source of error—to work on analysis, design, and even code all on the same day and to have issues from one stage impact issues from another. The emphasis on iterative models is that software development is a knowledge-intensive process and that things like analysis can't really be completely understood without understanding design issues, that coding issues can affect design, that testing can yield information about how the code or even the design should be modified, etc.

Although it is possible to do object-oriented development using a waterfall model, in practice most object-oriented systems are developed with an iterative approach. As a result, in object-oriented processes "analysis and design" are often considered at the same time.



The object-oriented paradigm emphasizes modularity and re-usability. The goal of an object-oriented approach is to satisfy the "open closed principle". A module is open if it supports extension. If the module provides standardized ways to add new behaviors or describe new states. In the object-oriented paradigm this is often accomplished by creating a new subclass of an existing class. A module is closed if it has a well-defined stable interface that all other modules must use and that limits the interaction and potential errors that can be introduced into one module by changes in another. In the object-oriented paradigm this is accomplished by defining methods that invoke services on objects. Methods can be either public or private, i.e., certain behaviors that are unique to the object are not exposed to other objects. This reduces a source of many common errors in computer programming.

The software life cycle is typically divided up into stages going from abstract descriptions of the problem to designs then to code and testing and finally to deployment. The earliest stages of this process are analysis and design. The distinction between analysis and design is often described as "what vs. how". In analysis developer's work with users and domain experts to define what the system is supposed to do. Implementation details are supposed to be mostly or totally (depending on the particular method) ignored at this phase. The goal of the analysis phase is to create a functional model of the system regardless of constraints such as appropriate technology. In object-oriented analysis this is typically done via use cases and abstract definitions of the most important objects. The subsequent design phase refines the analysis model and makes the needed technology and other implementation choices. In object-oriented design the emphasis is on describing the various objects, their data, behavior, and interactions. The design model should have all the details required so that programmers can implement the design in code.

3.NATIONAL TRADE SHOW SERVICE EXAMPLE

A trade show is an exhibition organized so that companies in a specific industry can showcase and demonstrate their latest products and services, meet with industry partners and customers, study activities of rivals, and examine recent market trends and opportunities. In contrast to consumer fairs, only some trade fairs are open to the public, while others can only be attended by company representatives (members of the trade, e.g. professionals) and members of the press, therefore trade shows are classified as either "public" or "trade only". A few fairs are hybrids of the two; one example is the Frankfurt Book Fair, which is trade only for its first three days and open to the general public on its final two days. They are held on a continuing basis in virtually all markets and normally attract companies from around the globe. For example, in the U.S., there are currently over 10,000 trade shows held every year, and several online directories have been established to help organizers, attendees, and marketers identify appropriate events.

The importance of face-to-face marketing experiences has never been more important - they build valuable business relationships and drive revenues. Hargrove was built on the belief that strong relationships play a vital role in the growth of any company. Because of the depth and breadth of our trade show management experience, we also understand the challenges you face while planning your show as budgets get tighter and the demand to increase the return on investment escalates.

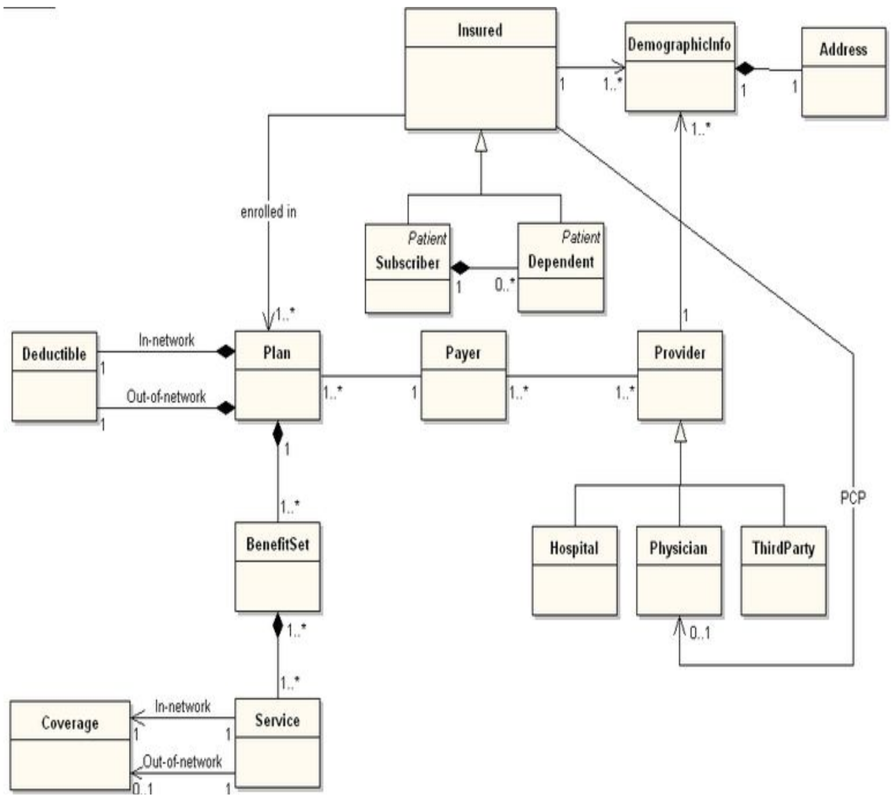
You need an experienced trade show manager willing to partner with you, identify solutions and deliver results. With a focus on personalized service, creativity and state-of-the-art resources, Hargrove is ready to bring your vision to life.

Learn how Hargrove can bring your trade show vision and goals to life.

4.DOMAIN MODELING

A domain model is a system of abstractions that describes selected aspects of a sphere of knowledge, influence or activity (a domain). The model can then be used to solve problems related to that domain. The domain model is a representation of meaningful real-world concepts pertinent to the domain that need to be modeled in software. The concepts include the data involved in the business and rules the business uses in relation to that data.

A domain model generally uses the vocabulary of the domain so that a representation of the model can be used to communicate with non-technical stakeholders.



5.USE CASE MODELING

A domain model is generally implemented as an object model within a layer that uses a lower-level layer for persistence and "publishes" an API to a higher-level layer to gain access to the data and behavior of the model.

In the Unified Modeling Language (UML), a class diagram is used to represent the domain model.

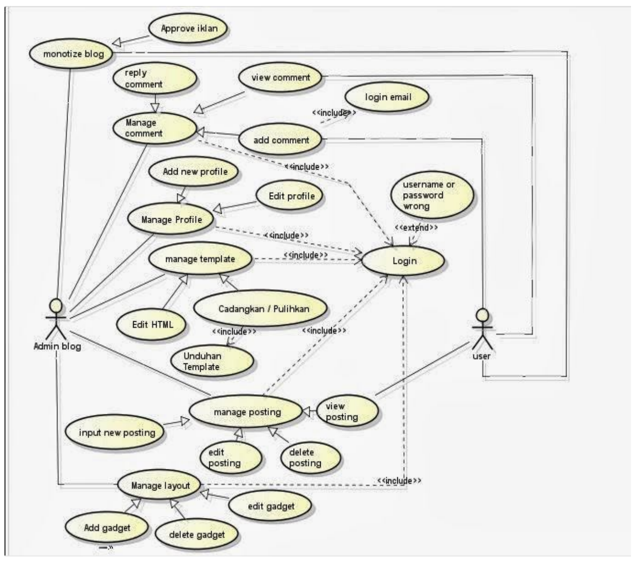
A use case diagram is a graphic depiction of the interactions among the elements of a system.

A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. In this context, the term "system" refers to something being developed or operated, such as a mail-order product sales and service Web site. Use case diagrams are employed in UML (Unified Modeling Language), a standard notation for the modeling of real-world objects and systems.

System objectives can include planning overall requirements, validating a hardware design, testing and debugging a software product under development, creating an online help reference, or performing a consumer-service-oriented task. For example, use cases in a product sales environment would include item ordering, catalog updating, payment processing, and customer relations. A use case diagram contains four components.

- The boundary, which defines the system of interest in relation to the world around it.
- The actors, usually individuals involved with the system defined according to their roles.
- The use cases, which are the specific roles played by the actors within and around the system.
- The relationships between and among the actors and the use cases.

A use case diagram looks something like a flowchart. Intuitive symbols represent the system elements. Here's a simple example**:**



6.ACTOR-SYSTEM INTERACTION MODELING

The functionality provided by the system is documented in a use case model that illustrates the system's intended functions (use cases), its surroundings (actors), and the relationships between the use cases and actors (use case diagrams). The most important role of a use case model is to provide a vehicle used by the customers or end users and the developers to discuss the system's functionality and behavior.

The use case model starts in the Inception Phase with the identification of actors and principal use cases for the system. The model is then matured in the Elaboration Phases.

Like external entities in Data Flow Diagram, actors are not part of the system -- they represent anyone or anything that must interact with the system. An actor may

- only input information to the system
- only receive information from the system
- input and receive information to and from the system

7.OBJECT INTERACTION MODELING

Besides the relationships an object can have with other objects and its own behavior, an object may also interact with another object. For example, an object may send information to, request information from another object, it may alter another object or cause another object to do some action.

In OSA, we use the OIM (Object-Interaction Model) to describe the interaction among objects. Object interaction in OIM has three basic components:

- The objects that are involved in the interaction.
- The way that the objects act or react in the interaction.\
- The nature of the interaction.

Since we are able to identify objects in ORM, We use ORM components in OIM to show which objects are involved in the interaction.

Since we are able to define the behavior of objects with state nets, we use state nets in OIM to describe how objects act and react in interactions.

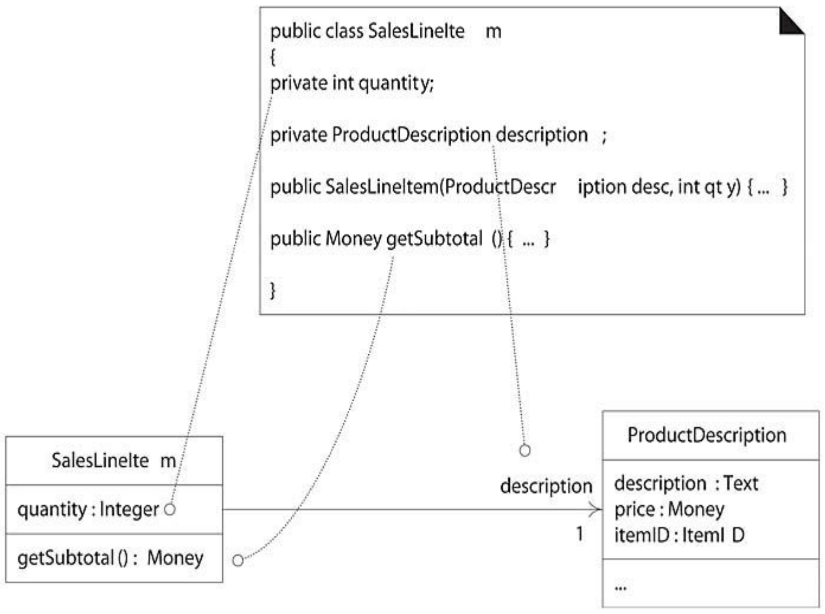
The nature of the interaction can be described by the activity that constitutes the interaction and the information or objects transmitted or exchanged in the interaction.

To describes the interaction and the objects exchanged in the interaction, we introduce a new feature, which create object-interaction models with an appropriate combination of ORMs and state nets.

8.GENERATING DCD AND SKELETON CODE

At the very least, DCDs depict the class or interface name, superclasses, operation signatures, and attributes of a class. This is sufficient to create a basic class definition in an OO language. If the DCD was drawn in a UML tool, it can generate the basic class definition from the diagrams.

Defining a Class with Method Signatures and Attributes
From the DCD, a mapping to the attribute definitions (Java fields) and method signatures for the Java definition of SalesLineItem is straightforward, as shown in Figure.



```
public class SalesLineIte m
{
private int quantity;

private ProductDescription description ;

public SalesLineItem(ProductDescr iption desc, int qt y) { ... }

public Money getSubtotal () { ... }

}
```

9.ARCHITECTURAL DESIGN AND IMPLEMENTATION

Any real-world system is used by different users. The users can be developers, testers, business people, analysts, and many more. Hence, before designing a system, the architecture is made with different perspectives in mind. The most important part is to visualize the system from the perspective of different viewers. The better we understand the better we can build the system.

UML plays an important role in defining different perspectives of a system. These perspectives are –
- Design
- Implementation
- Process
- Deployment

The center is the Use Case view which connects all these four. A Use Case represents the functionality of the system. Hence, other perspectives are connected with use case.

Design of a system consists of classes, interfaces, and collaboration. UML provides class diagram, object diagram to support this.

Implementation defines the components assembled together to make a complete physical system. UML component diagram is used to support the implementation perspective.

Process defines the flow of the system. Hence, the same elements as used in Design are also used to support this perspective.

Deployment represents the physical nodes of the system that forms the hardware. UML deployment diagram is used to support this perspective.

It is very important to distinguish between the UML model. Different diagrams are used for different types of UML modeling. There are three important types of UML modeling.

10.STRUCTURAL MODELING

Structural modeling captures the static features of a system. They consist of the following –
- Classes diagrams
- Objects diagrams
- Deployment diagrams
- Package diagrams
- Composite structure diagram
- Component diagram

Structural model represents the framework for the system and this framework is the place where all other components exist. Hence, the class diagram, component diagram and deployment diagrams are part of structural modeling. They all represent the elements and the mechanism to assemble them.

The structural model never describes the dynamic behavior of the system. Class diagram is the most widely used structural diagram.

11. CASE STUDY AND EVALUATION

The purpose of a case study is to study intensely one set (or unit) of something—programs, cities, counties, worksites—as a distinct whole. What does this mean? For a program designed to encourage bars to observe the smokefree bar law, an evaluation must document the program's impact on the bars and on the behavior of people in the bars. In a non-case study design, one might decide to observe a series of randomly selected bars to see whether bartenders take some action to enforce the smokefree bar law when customers begin to smoke. This style of evaluation entails collecting data on bartender behavior from a random sample of bars large enough to be representative of the entire population of bars from which you sampled. In contrast, a case study design focuses on a hand-picked set of bars (sometimes even just one bar). Before the program begins, the evaluator spends time in the bar(s), observing behavior and talking with people. As the program progresses, the evaluator continues to make observations and to interview the owners, managers, employees, and customers. She might observe the bars at various times of the day to monitor compliance with other smokefree rules, such as the absence of ashtrays. At the completion of the program, the case study reveals in depth the experience of specific bars in implementing the new law, and the impact the program had on its efforts. Did the program either encourage or discourage compliance? Did new signage go up, and did bartenders begin to encourage compliance? Or did something completely unrelated to the program happen to change behavior? For example, did a bartender relapse during a quit attempt and resume smoking, thus encouraging others to smoke? Did, on the other hand, a favorite bartender have a heart attack, which made the customers more sensitive to smoking behavior? This kind of rich detail lets evaluators assess programs in a way that several data elements across a large variety of cases cannot. In a case study, note that some of the data collected might be quantitative, such as the number of instances of compliance at various times of the day. Case studies do not necessarily use qualitative data only. Overall, case studies are considered to be a qualitative technique, but they can contain quantitative information. However, the overall goal of a case study, which is to understand a select subset as a distinct whole in its particular context, distinguishes the case study from other designs. What one gains in richness by doing a case study evaluation, one loses in the breadth of generalizations about overall compliance. Put another way, a case study reveals a lot about the process and outcome at certain sites, and the ways in which these interrelate. It reveals less about a program's overall impact. One way to offset the lack of breadth in a single case study is to do multiple case studies and to compare the findings. For example, an evaluator could do complete profiles of several sets of bars, compare their implementations of the smokefree bar law, and look at the similarities and differences in implementation. This comparative study begins to clarify the impacts that your program either had or did not have, providing useful information for program revisions.

12.CONCLUSIONS AND FUTURE WORK

In this thesis, we addressed the problem of recognition of structures in images using graph representations and inexact graph matching. One of the main contributions of our work is to express this task as a combinatorial optimization problem with constraints, and to propose methods to solve it based on EDAs and their parallelization. A discussion on different representations of individuals has been provided. In particular, we proposed representations in both the discrete and continuous domains. Some of the constraints imposed to the matching could be introduced directly in the representations. Different types of fitness functions have been presented. Our contribution here is twofold. First an experimental comparison of their behavior has been performed, and second new fitness functions based on probability theory have been designed. The main focus of our thesis was on the optimization itself. A new approach based on estimation of distribution algorithms was introduced for solving the graph matching problem. Its foundations rely on an evolutionary computation paradigm that applies learning and simulation of probabilistic graphical models (i.e. Bayesian networks in the discrete domain and Gaussian networks in the continuous one) as an important part of the search process. Our contribution in this part was to adapt these algorithms to the inexact graph matching problem with constraints, which to our knowledge have never been addressed before. In particular we proposed original solutions to take the constraints into account. This contribution can certainly be exploited in other combinatorial optimization problems with constraints, thereby enlarging the potential application field of EDAs. Finally another contribution relies in the parallelization of EDAs. Up-to-date parallelization techniques have been applied to these algorithms, resulting in two different programs suitable for execution on multiprocessors with shared memory and cluster of workstations under windows or GNU-Linux systems. The use of shared memory libraries with threads –using pthreads– as well as high-level parallelization libraries based on message passing –such as MPI– have been analyzed in detail. The particular case of EBNABIC has been detailed, and each of its steps has been analyzed in terms of parallelization and computation costs. A parallel version of this algorithm is proposed for the BIC metric. This contribution allows now to use EDAs to solve problems with higher complexity. From an experimental point of view, our contribution lies in the comparison of the performance of EDAs in both discrete and continuous domains with other evolutionary computation techniques such as genetic algorithms and evolutionary strategies. These experiments were performed for the different types of individual representations, different types of fitness functions, and applied to synthetic and real graph matching problems. Results show that our approach obtains better results and that converge to a solution by having to evaluate less individuals than other more usual evolutionary computation methods such as genetic algorithms. These differences in the results have been proved to be statistically significant after applying non-parametric tests.

Many different adaptations, tests, and experiments have been left for the future due to lack of time (i.e. the experiments with real data are usually very time consuming, requiring even days to finish a single run). Future work concerns deeper analysis of particular mechanisms, new proposals to try different methods, or simply curiosity. There are some ideas that I would have liked to try during the description and the development of the fitness functions in Chapter 3. This thesis has been mainly focused on the use of EDAs for graph matching, and most of the fitness functions used to find the best result where obtained from the literature of adapted from these, leaving the study of fitness functions outside the scope of the thesis. The following ideas could be tested: 1. It could be interesting to consider the regions in the model and data images with different importance, depending on their size or their specific meaning with respect to the recognition process. This mechanism would for instance aid to distinguish in very complex problems which are the regions that are essential to be found, the ones that sometimes appear, and the ones that rarely do. 2. The way the model is constructed could be also changed: instead of using one typical image (prototype), it could be based on different images, in order to provide some information on the variability among the different images, and introduce it in the attributes. Unfortunately, in the type of images that we have taken as real examples the construction of a model from each image is a tedious task and no further study in this direction could be performed. Obviously, the use of other types of individual representations and fitness functions could be investigated since they have an important influence on the results obtained at the end. New approaches in this direction can be induced from techniques described in the literature such as [Bloch, 1999a,b, Rangarajan et al., 1999a, Sanfeliu and King-Sun, 1983]. The performances of all the fitness functions described in Section 3.4.3 have not been compared on a same problem. The main reason was that some fitness functions are very complex to compute and require a considerable execution time to evaluate each individual. Parallelization techniques have been applied to the learning step in EDAs, but not for the evaluation of individuals, and such a mechanism could help at reducing execution times. Nevertheless, we are already designing and running experiments to compare the performance of our newly proposed probability theory-based fitness function f4(h) and f5(h) to such of the fitness functions defined previously in this section. The preliminary results of these experiments do not seem to be satisfactory, and further study is still required in order to understand the behavior of these two fitness functions and improve it. Concerning the results for both applications (brain and facial features), we can also expect to improve them by having richer graphs, with more attributes. In the definition of the EDAs in Chapter 4, there are also many ideas that could be exploited to try to obtain a most effective convergence towards the best solution. An example of this is the use of a mechanism that could be understood as a learning depending on the fitness value of the individual: in the learning proposed for EDAs all the selected individuals are used for the learning equally regardless of their fitness value. This means that the fitness value is just considered for selecting the best individuals, but differences between the values among these individuals are not considered in the learning process. A similar idea to this is proposed in the Bit-Based Simulated Crossover algorithm (BSC) [Syswerda, 1993], but this idea could be extended to any EDA. One of the disadvantages that this new type of learning can have is that by accelerating the convergence the search is too focused to the main individuals, and therefore EDAs could lead to local maxima. However, this idea is still a possibility that could be analyzed in the future to check whether local maxima are avoided or not and how to improve it for specific problems such as inexact graph matching. The initial population in all EDAs has been built using a uniform distribution. Other methods could be also tested, as sometimes a pre-processing step could be added so that the search can also start with some specific individuals. Also, other types of statistical initializations such as greedy probabilistic methods could help at directing the search from the beginning, leading to less evaluations. Regarding the application of parallelism to EDAs, an extension for the near future is the use of more powerful multicomputers in order to improve the parallelization: the computers we used had at most only 2 processors, and therefore no more than 4 workers were created per computer so that all the workers do not compete for CPU use with the corresponding thrashing problem. An additional task to perform is the parallelization of other algorithms such as EGNAee and EMNA, which are also susceptible of being parallelized due to the high number of tasks that can be performed in parallel on different processors.

REFERENCES

[1] Peters, L. 1981. Software Design. New York, NY: Yourdon Press, p. 22.
[2] F. April 1987. No Silver Bullet: Essence and Accidents of Software Engineering. IEEE Computer vol. 20(4), p. 12.
[3] Panias, D. july 1985. Software Aspeas of Strategic Defense System Victoria, Canada: University of Victoria, Report DCS-47-IR.
[4] Gall, J. 1986. Systemantics: How Systems Really Work and How They Fail. Second Edition. Ann Arbor, MI: The General Systemantics Press, p. 65.
[5] Tsai, J. and Ridge, J. November 1988. intelligent Support for Specifications Transformation. LEEE Software vol. 5(6), p. 34
[6] Rentsch, T. September 1982. Object-Oriented Programming. SIGPL¿1NNoticesvol. 17(12), p. 51.
[7] Shankar, K. 1984. Data Design: Types, Structures, and Abstractions. Handbook of Software Engineering. New York, NY: Van Nostrand Reinhold, p. 253.
[8] Macintosb MacApp 1. 1. 1 Programmers Reference. 1986. Cupertino, CA: Apple Computer, p. 2.
[9] Bhaskar, K. October 1983. How Object-Oriented Is Your System? SIGPLANNotices vol. 18(10), p. 8.
[10] Kavi, K. and Chen, D. 1987. Architectural Support: for Object-Oriented Languages. Proceedings of tbe nirty-second 1EEE Computer Society Intemational Conference lEEE.
[11] iAPY 432 Object Primer 1981. Santa Clara, CA: Intel Corporation.
[12] Dally, W. J. and Kajiya, J. T. March 1985. SIGARCH Newsletter vol. 13(3).
[13] Dahlby, S., Henry, G., Reynolds, D., and Taylor, P. 1982. The IBM System/38: A High Levei Machine, in Computer Structures: Principles and Example& New York, NY: McGraw-Hill.
[14] Gane, C. and Sarson, T. 1979. Structured Systems Analysis. Englewood Cliffs, Nj: Prentice-Hall.
[15] Ward, P. and Mellor, S. 1985. Structured Developmentfor Real-Time Systems Engle-wood Cliffs, Nj: Yourdon Press.
[16] Hatley, D. and Pirbhai, 1. 1988. Strategiesfor Real-Time System Specification New York, NY: Dorset House.
[17] Atkinson, M., Bafley, P., Chishoim, K., Cockshott, P., and Morrison, R. 1983. An Approach to Persistent Programming. 7be Computerjournal vol. 26(4), p. 360.
[18] Khoshafian, S. and Copeland, G. November 1986. Object Identity. SIGPLANNotices vol. 21(11), p. 409.
[19] Vlissides, J. and Linton, M. 1988. Applying Object-Oriented Design to Structured Graphics. Proceedings of USENIX C++ Conference Berkeley, CA: USENIX Association, P. 93 May, R. September 16, 1988. How Many Species Are There on Earth? Science vol. 241, p. 1441.

[20] Descartes, R. 1984. Rules for the Direction of the Mind. Vol. 31 of Great Books of the Westerri World. Chicago, IL: Eney-clopedia Britannica, p. 32.
[21] Shaw, M. May 1989. Larger Scale Systems Require Higher-Level Abstractions. SIGSOFT Engineenng Notes vol. 14(3), p – 143
[22] Brooks, F. 1975. The Mythical Man-Montb. Reading, MA: Addison-Wesley, p. 42.
[23] Gilb, T. 1988. Principles of Software Engineering Management. Reading, Massachusetts Addison-Wesley, p. 92.
[24] Mellor, S., Hecht, A., Tryon, D., and Hywari, W. September 1988. Object-Oriented Analysis: Theory and Practice, Course Notes, in Object-Ollented Programming Systems, Languages, and Applications. San Diego, CA: OOPSLA'88, p. 1.3.
[25] Dijkstra, E. May 1968. The Structure of the "THE" Multiprogramming System. Communications of the ACMvol. 11(5), p. 341.
[26] Kishicla, K., Teramoto, M., Torri, K., and Urano, Y. September 1988. Quality Assurance Technology in Japan. IEEE Software vol. 4(5), p. 13.
[27] Wirth, N. 1986. Algoritbms and Data Structures, Second Edition. Englewood Cliffs, NJ: Prentice-Hall.
[28] Wirfs-Brock, R. October 1991. Object-Oriented Frameworks. American Programmer vol. 4(10), p. 27.
[29] Date. An Introduction, p. 10.
[30] Hawryszkiewycz, 1. 1984. Database Analysis and Design. Chicago, IL: Science Research Associates, p. 425.
[31] Wiorkowski, G. and Kull, D. 1988. DB2 Design and Development Guide. Reading, MA: Addison-Wesley, p. 29.
[32] Meyer, B. 1988. Object-Oriented software Construction. New York, NY: Prentice Hall.
[33] Saunders, J. March/April 1989. A Survey of Object-Oriented Programming Languages. Journal of Object-Oriented Programming vol. 1(6).
[34] W. Rankl, W. Effing, “Smart Card Applications, Design Models forusing and programming smart cards,” Springer-Verlag, 2007.
[35] A readable and helpful overview of how to do case studies. Good interviewing section. Although the word, “Education,” is in the title, this book has more general application. Patton, M. Q. (1987).
[36] A. B. Mohamed, A. A. Hamid, K. Y, Mohamed,“Implementation of an Improved Secure System Detection for E-passport by using EPCRFIDTags,” 2009.
[37] R. Schoof, “Elliptic Curves over Finite Fields and the Computation of Square Roots mod p”, Mathematics of Computation, Vol. 44, No. 170, pp. 483-494, April 1985.
[38] F. Morain,“Building cyclic elliptic curves modulo large primes”, Advances in Cryptology – EUROCRYPT 91, Lecture Notes in Computer Science, 547: 328-336, 1991.
[39] N. Koblitz,“A Course in Number Theory and Cryptography”, Springer-Verlag, second edition, 1994.
[40] Padma Bh, D. Chandravathi, P. P. Roja, “Encoding And Decoding of a Message in the Implementation of Elliptic Curve Cryptography using Koblitz’s Method”, IJCSE, Vol. 02, No. 05, 1904-1907, 2010.T. C. Lee, B. K. Lee, “A HESSL (Highly Enhanced Security Socket Layer) protocol”, In: The Proceedings of the Seventh IEEE International Conference on E-Commerce Technology, July 19-22, 2005, Munich, Germany, 456-460.