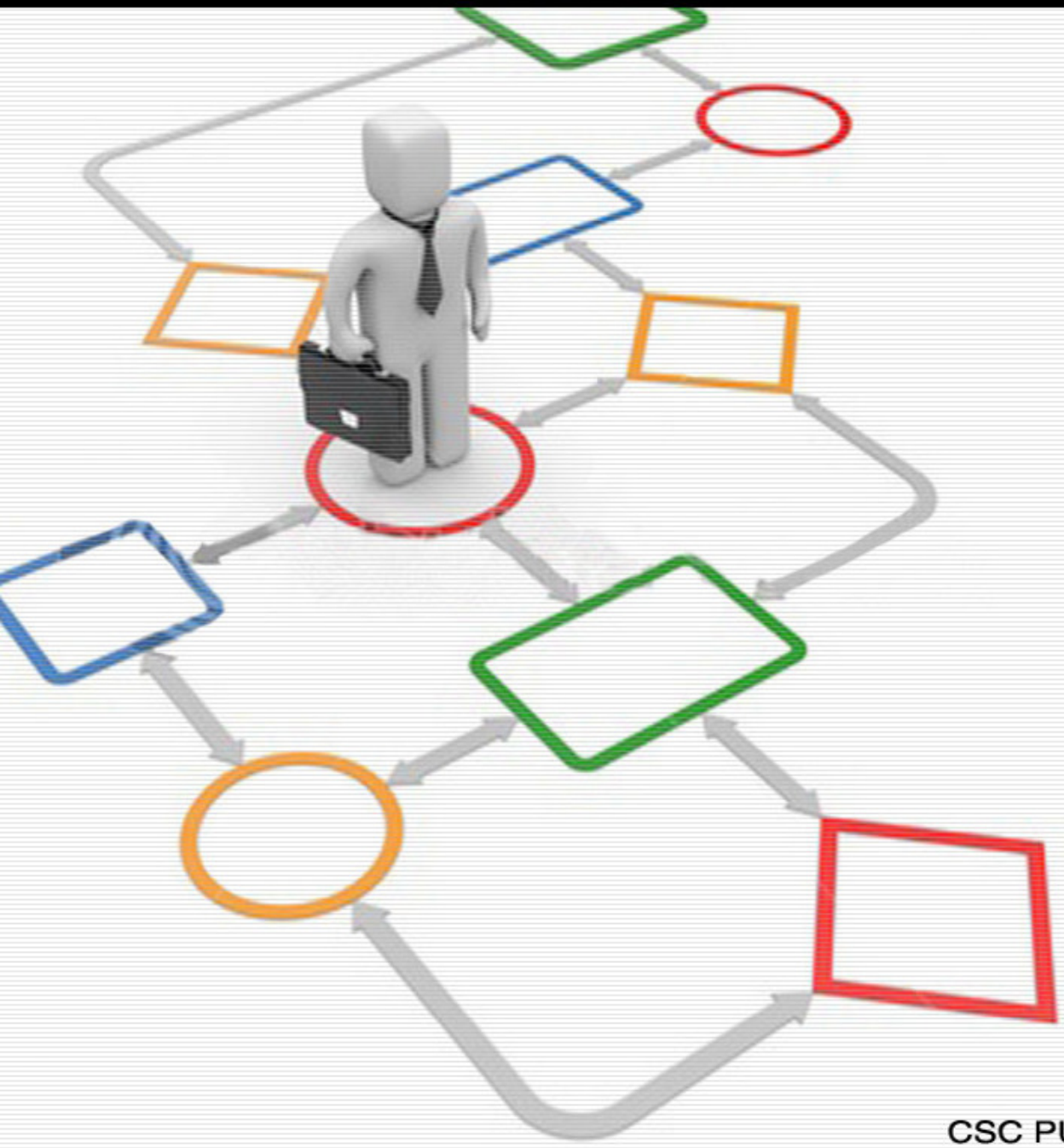
Volume 4 ▪ Issue 1 ▪ September 2013

INTERNATIONAL JOURNAL OF

SOFTWARE ENGINEERING (IJSE)

ISSN : 2180-1320

Publication Frequency: 6 Issues / Year



CSC PUBLISHERS

http://www.cscjournals.org

INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING (IJSE)

VOLUME 4, ISSUE 1, 2013

EDITED BY
DR. NABEEL TAHIR

ISSN (Online): 2180-1320

I International Journal of Software Engineering (IJSE) is published both in traditional paper form and in Internet. This journal is published at the website http://www.cscjournals.org, maintained by Computer Science Journals (CSC Journals), Malaysia.

IJSE Journal is a part of CSC Publishers
Computer Science Journals
http://www.cscjournals.org

INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING (IJSE)

Book: Volume 4, Issue 1, September 2013
Publishing Date: 15 - 09 - 2013
ISSN (Online): 2180-1320

IJSE Journal is a part of CSC Publishers
http://www.cscjournals.org

Published in Malaysia

Typesetting: Camera-ready by author, data conversation by CSC Publishing Services – CSC Journals, Malaysia

CSC Publishers, 2013

EDITORIAL PREFACE

The International Journal of Software Engineering (IJSE) provides a forum for software engineering research that publishes empirical results relevant to both researchers and practitioners. It is the First Issue of Fourth Volume of IJSE and it is published bi-monthly, with papers being peer reviewed to high international standards.

The initial efforts helped to shape the editorial policy and to sharpen the focus of the journal. Started with Volume 4, 2013, IJSE appears with more focused issues. Besides normal publications, IJSE intend to organized special issues on more focused topics. Each special issue will have a designated editor (editors) – either member of the editorial board or another recognized specialist in the respective field.

IJSE encourage researchers, practitioners, and developers to submit research papers reporting original research results, technology trend surveys reviewing an area of research in software engineering, software science, theoretical software engineering, computational intelligence, and knowledge engineering, survey articles surveying a broad area in software engineering and knowledge engineering, tool reviews and book reviews. Some important topics covered by IJSE usually involve the study on collection and analysis of data and experience that can be used to characterize, evaluate and reveal relationships between software development deliverables, practices, and technologies. IJSE is a refereed journal that promotes the publication of industry-relevant research, to address the significant gap between research and practice.

IJSE gives the opportunity to researchers and practitioners for presenting their research, technological advances, practical problems and concerns to the software engineering. IJSE is not limited to a specific aspect of software engineering it cover all Software engineering topics. In order to position IJSE amongst the most high quality journal on computer engineering sciences, a group of highly professional scholars are serving on the editorial board. IJSE include empirical studies, requirement engineering, software architecture, software testing, formal methods, and verification.

International Editorial Board ensures that significant developments in software engineering from around the world are reflected in IJSE. The submission and publication process of manuscript done by efficient way. Readers of the IJSE will benefit from the papers presented in this issue in order to aware the recent advances in the Software engineering. International Electronic editorial and reviewer system allows for the fast publication of accepted manuscripts into issue publication of IJSE. Because we know how important it is for authors to have their work published with a minimum delay after submission of their manuscript. For that reason we continue to strive for fast decision times and minimum delays in the publication processes. Papers are indexed & abstracted with International indexers & abstractors.

Editorial Board Members
International Journal of Software Engineering (IJSE)

EDITORIAL BOARD

TABLE OF CONTENTS

Volume 4, Issue 1, September 2013

Pages

Quality Attributes and Software Architectures Emerging Through Agile Development: Pursuit or Overlooking?

G. H. El-Khawaga *Ghada.elkhawaga@ieee.org*
Teaching Assistant, Department of information systems,
Faculty of computers and information,
Mansoura University,
Mansoura, Egypt

Prof. Dr. Galal Hassan Galal-Edeen *Galal@acm.org*
Computer Science Department,
School of Sciences & Engineering,
American University in Cairo,
Cairo, Egypt

Prof. Dr. A.M. Riad *amriad2000@mans.edu.eg*
Dean of the faculty of computers & Information,
Mansoura University,
Mansoura, Egypt

Abstract

Software architectures play an important role as an intermediate stage through which system requirements are translated into full scale working system. The idea of what a system does, what it does not, and different concerns and requirements can be negotiated and expressed clearly through the software architecture. Software architectures exist to enhance and provide quality attributes, while they are quality attributes and their required level of achievement which can offer numerous number of software architectures for a single software system.

We believe that the agile approach to architecting is problematic because of agilists' beliefs about how to architect a software system, and how critical quality attributes are to achieve a stable yet flexible architecture. Through this research we clarify these issues, and discuss consequences of agile architecting on achieved level of quality attributes. We are going to pursue the answer to how to architect to achieve required level of quality attributes, while adopting an agile process.

Keywords Quality Attributes, Software Architecting, Agile Software Development, Refactoring, Clean Architecting, Light Architecting.

1. QUALITY ATTRIBUTES AS BEING ENABLER OF SOFTWARE ARCHITECTURES VARIANCES

Are software architectures there to answer certain quality attributes-related questions? Have we got to care about arrangements and relationships between software components in response to quality attributes-related needs? Have the concept of software architectures emerged after being involved into long era of deficient software resulting from unstructured development? Do software architectures exist to enhance quality attributes of software systems, or they are quality attributes which distinguish software architectures? If the answers to these questions are all "yes", then there are more questions to ask. Do software architectures emerging through paradigms like agile software development achieve their purpose of reaching a certain level of quality attributes defined through a product's context and concerns' analysis? Can we truly offer longevity of a

software product and its ability to absorb frequent changes all over its production time without paying attention to how its architecture is formed to offer quality attributes? To find an answer, we need to begin tackling and defining the relation between a software architecture and quality attributes.

1.1 Criticality of Quality Attributes

The intent of designing the architecture for a system is to transfer system required functionality, quality attributes, business goals, and system context into an intermediate state before being transformed to full-scale developed system. Software architecture is an arrangement of software building blocks into differentiated types, or categories that are grounded in or derived from the problem domain, and the way the software might be used and later adapted as an artefact [1]. This definition mainly referred to system requirements as a main driver of an architecture. Therefore; through architecture creation, architects are supposed to elicit and understand the received requirements so as to reach clear view of what the system should do, and to begin on making decisions that shape how the system will work to achieve desired goals. However, it is emphasized that a software architecture differs from building architecture in that it can't be limited to one structure [2]. In civil engineering, structure is one category of the architecture; while in software engineering, a system can have thousands of forms, which differ in quality attributes satisfaction levels, not in the functionality associated and achieved through these forms. If it were only about functionality, a software system would have been composed of a single module with no internal structure [2]. Functionality drives the initial decomposition of a system architecture into a set of components that together perform the functions of the system [3], but it is the mapping of a system's functionality into software structures that determines the architecture's support for quality [2]. A quality attribute is a constraint on the manner in which the system implements and delivers its functionality [4]. Systems are redesigned not only due to functionality dissatisfaction, but also due to lack of consideration of quality attributes like security, performance, maintainability, and reliability [2]. Quality attributes are advanced to functionality considerations, and this can be argued for by the idea [3] that one of the motivations for creating an architectural design (addressing quality attributes) before detailed design (addressing functionality) and coding is to enable improving, measuring, observing the quality of the system, and predicting whether the system to be built will exhibit certain quality attributes while addressing risks and potential defects earlier where they are cheaper, easier, and faster to fix. At the same time, software architecting is a major strategy for enhancing quality attributes of software systems [1]. Architecture plays a central role in realizing many qualities in a system. While we believe that an architecture embodies decisions about quality priorities and tradeoffs, and represents an early opportunity to evaluate these decisions, it is argued that an architecture provides only the foundation for achieving quality; but without paying attention to the details, this foundation will be in vain [2].

1.2 Challenges Associated With Quality Attributes' Specification

Considering, expressing, and evaluating quality attributes is not an easy mission. Challenges of adopting quality attributes can be categorized into two paths, so as to enable recognizing how to consider and deal with a system's desired quality attributes. A path or a category is about *what* these quality attributes are, and the other is about *how* they are considered into a system. The first category is related to the natural characteristics of quality attributes themselves. Many quality attributes naturally have architectural and non-architectural aspects. Performance, for example, has architectural aspects like functionality allocation to components, and communication between components; while it has non-architectural aspects like the choice of algorithms to achieve functionality, and how these algorithms are coded [2]. Ignoring this confusing nature of quality attributes raises many pressures and challenges, like the difficulty of ensuring that a specific quality attribute has stemmed of nontechnical issues [4]. Much attention should be paid to architectural and non-architectural aspects of a quality attribute so as to decide how to handle it while it is affecting other attributes.

Whether positively or negatively, quality attributes affect each other. So they cannot be handled in isolation. While making an architectural design decision, interactions between quality attributes

should be put into consideration, and a decision is to be made based on affected and interacting quality attributes relative priorities. Conflicts between quality attributes should be discovered as early as possible, and desired quality attributes achievement levels should be available early so as to help make a decision about a certain quality attribute preference whenever a conflict exists. However, this depends on how a development team handles quality attributes; and this is shown through the second category of challenges in dealing with quality attributes.

Another challenge that stems of natural characteristics of quality attributes is how to measure and evaluate an architecture's achievement of certain quality attributes. This challenge is due to that many quality attributes are qualitative in nature, rather than being quantitative [3]. For example, a software system into operation can be tested for its performance by quantitative measures, while maintainability of a system should be observed and reasoned about through qualitative measures like questionnaires. Considering a qualitative or a quantitative quality attribute for assessment is critical to deciding when to carry out an evaluation phase.

The second category of challenges is related to how quality attributes are handled through the development process, and where they are located into development participants' consideration. There is a wide agreement that modelling methods are weak in representing quality attributes [3], and that architectural analysis techniques focusing on quality attributes are rare [4]. This drives software architects to deal with quality attributes with an informal process [2]. However, informal and incomplete specifications of quality attributes increase dependability on the architect to fill in blanks and mediate the conflicts, and increases possibilities of redesigning the system to meet missed quality attributes. It is confirmed that quality, cost, and schedule are not independent as poor quality affect cost, and schedule [5].

Another challenge stems from that architects and developers –especially agilists- tend to deal with quality attributes as an afterthought [4]. This was attributed to the development team's attention to business stakeholders rather than technical ones, and to the team's belief that some quality attributes don't have direct impact on the cost-benefit for a system [6]. Business stakeholders won't be able to ask questions other than those about functionality, and they won't be aware of these questions that can help in analyzing and assessing the desired system's architecture [3]. The way of handling quality attributes raises technical future risks which if not handled early, they can break the system, and consequently will impact the cost-benefit of obtaining and operating the system threatened. It is argued that the costs for maintaining and extending an application will account for most of the cost of the application over its lifetime [3].

Agilists architect software in a way that exposures resulting architectures to risks associated with the challenges defined through this section. We are going to explain this more through the coming sections.

2. THE AGILE WAY TO TACKLE QUALITY ATTRIBUTES

Agilists regard architecting in light of traditional development as being associated with heavy-weighted practices which don't yield value on the short term. Of course we are totally against these beliefs, but it is out of scope to discuss and argue about how far these claims from reality. What we are concerned about here is to discuss architecting practices that agilists use and have influence on quality attributes. The main agile techniques to tackle quality attributes are architectural spiking, and refactoring. *Architectural spiking* is about implementing a feature that the development team believe to be exposed to and affected by the highest number of architectural design decisions. We believe architectural spikes are not efficient at evaluating architecture design decisions, because those decisions were originally made to satisfy certain quality attribute concerns. Quality attributes cross-cut a software architecture, while quality attribute concerns differ across various parts of an architecture. To take a vertical slice of an architecture as a means to judge the level of achievement of a quality attribute, while knowing that this quality attribute would be heterogeneous across the whole architecture; this doesn't seem to be a viable way to evaluate an architecture's conformance to its basic role. Agilists claim

they do only practices that add value, and we strongly believe conducting architectural spikes is a practice that missed its basic value. Agilists use architectural refactoring to make high-level changes to achieve quality attributes. However, not all quality attributes such as security can be accommodated later in implementation through refactoring [9]. Some quality attributes' components and mechanisms must be designed early in the life cycle. Issues associated with the way agilists handle changes through architectural refactorings and these issues' implications are explored through the coming section.

Agilists believe in simplifying design to achieve a barely good enough design to begin with. The point here is that while software architecture is believed to be the magical work for achieving system qualities such as performance, security, and maintainability [7], agilists consider designing for system qualities to be heavy work about unforeseen changes, and this work should be eliminated if not avoided. While adopting this attitude; they ignore foreseen changes that would come up on the long term. As a consequence; agile methods are accused of ignoring quality attributes such as reliability, scalability and changeability [8]. As change is inevitable, mechanisms should be employed to enable software to smoothly be adapted to changing circumstances in the development game.

3. AGILE ARCHITECTURAL REFACTORINGS: INTENDED TO PROVIDE A CHANCE, AND RESULTING IN A THREAT

Adding quality attributes through a software system's life cycle introduces new requirements, thus it can be considered some sort of perfective changes because they introduce new requirements and they aim at non-functional optimizations [10]. Lientz et al. –as cited in [10]- reported that 60.3% of the maintenance effort was categorized as perfective. This percentage is close to the results reported by Mockus & Votta's study [11] conducted which concluded that perfective changes accounted for 45% of all the modification requests. The challenges accompanying quality attributes' accommodation -whether these challenges are in general or are attributed to the usage of agile methodologies in software development- have resulted in having perfective changes to be of the highest percentage of the total maintenance efforts. The study conducted in [11] revealed that perfective changes -as well as being the highest to add more lines of code- are more time consuming than adaptive and corrective changes.

To study a change's implications on cost and schedule; the proposed change shouldn't be attached only to the code level. Instead, and with the aid of a big picture of the system under consideration; a proposed change should be studied at a global level rather than being localized only at the code level. A proposed change to code shan't be left till it violates the principal architectural design decisions that govern the application. In the way of identifying how change can affect a system's architecture, practitioners [4] tried to borrow some architectural concepts from physical buildings' literature. They were inspired by Stewart Brand's Shearing Layers of change. Brand categorized elements that make up a building into six categories. Brand's layers of change [4]:

1.Site: the geographical setting, and legally defined lot.

2.Structure: the foundation and load-bearing elements which are expensive to change.

3.Skin: exterior surfaces; they change so frequently to keep up with technology or for repair.

4.Services: the working guts of a building like electrical wiring.

5.Space plan: the interior layout; like doors, and floors.

6.Stuff: all the things that can be changed on a daily to monthly basis.

This categorization is organized in a manner that reflects the velocity and the hardness of changing the elements classified, from the slower and harder to change to the faster and easier.

Practitioners tried to make benefit of this categorization in software [4]. According to [4], the site layer in software denotes the usage context which may be an organization; the structure layer denotes the software system architecture as it identifies a system's load-bearing elements; the skin layer denotes user interfaces; the stuff layer may denote user settings. We believe that grouping elements by their similar change rates can help separate concerns, localize changes, and hence increase a software systems' responsiveness to changes. Such categorization can aid in identifying the necessary techniques to apply a given change, as well as the time and cost to achieve it [4]. Adhering to these groupings, it can be concluded that changes to software system architecture are to be the most expensive, difficult, and complex to implement.

Besides the important role an architecture plays in preparing for how the system will change and in localizing the effects of change, the profound changes to a system's architecture are induced by quality attributes' accommodation [4]. As mentioned before, agilists use refactoring as a main technique for adding quality attributes late in development lifecycle. Refactoring to introduce or modify quality attributes can imply modifying a component's internal specification; for example, introducing new components to increase performance implies changes to connectors [4]. Therefore, the consequences of making changes that can affect architecture elements – especially those resulting from making changes to accommodate quality attributes- should be studied carefully. Quality attributes are prevailing and affecting huge portions of code and functionality, thus modifying quality attributes is believed to be costly [7]. Not accommodating these changes early in the development process is sufficient to tear down the myth of having better quality using agile methods.

Frequent non-systemic modifications to requirements can result in architectural degradation, which leads to a mismatch between the actual functions of the system and its original design [12], and subsequently upgrades and fixes become expensive to implement. This case is called architectural erosion [11]. Architectural erosion is defined as the regressive deviance of an application from its original intended architecture resulting from successive changes [4]. Architectural erosion leads to increasing resistance to change and subsequently high cost of maintenance [13]. Architectural degradation causes are mainly mapped to late-lifecycle changes which are considered to be the most crucial, risky, and expensive when they are changes to requirements [12]. Therefore, the earlier to make changes is the better, and the earlier to consider quality attributes is the best. The difficulty, the choice of suitable technique, and the cost of supporting a given change are all deeply influenced by the development level at which a change is implemented [4]. As a result, late-lifecycle refactorings affecting the architecture of a software system are considered to be the most risky and expensive changes.

Among the important triggers of architectural refactoring are architectural smells which are believed to be negatively impacting system quality [14]. Architecture violations are considered to be the main architectural smells' type for which architectural refactorings are carried out [15]. This way we can conclude that refactoring to overcome certain architecture violations is likely to produce other architecture violations, and even they can be of a greater number than the ones these refactorings were carried out to overcome. Therefore, refactoring to reduce or eliminate an architectural smell can be risky and complex [14]; as it requires decisions that seem to be local while they have broad effects and involve uncertain consequences. The problem is more complex and risky in case of the absence of a well-defined architecture, and this may be the case while adopting an agile method in software development.

Architectural refactoring effectiveness for achieving quality is another issue that rises here. Architectural refactoring is effective in increasing an application's maintainability and consequently reducing costs [15]. However, architectural refactoring's effect on other quality attributes like performance, and security should be considered as well. Also, mutual influences of quality attributes and sometimes conflicts are critical aspects to be considered. Not all quality

attributes can be achieved in the same time; their achievement is proportional and they can't be treated in isolation of each other. Thus for example, refactoring to increase performance can affect reliability negatively, and so on.

We believe that architectural refactoring can alter a product's perceived behavior whenever these refactorings are conducted to incorporate quality attributes. This claim sounds reasonable as long as the main aim of refactoring is to alter internal structure without changing external behaviour, and it also raises critical questions about the viability of refactoring –in the context of agile development- to leverage a system's architecture and alter it later to insert missed quality attributes. Refactoring to fix architectural problems was firmly emphasized to be inefficient [16]. Refactoring, as considered to be a small activity with limited effect, is almost a local activity, whereas architecture is a global concern.

The discussion above highlights two issues; the *first* is that the need for spending some time planning architecture upfront is not something to be ignored. The second issue is that depending on refactoring to bring good code structure and hoping that code units together will form a good architecture that will stand and accommodate all upcoming changes won't be a viable development strategy in most cases.

4. SALVATION THROUGH CLEAN LIGHT ARCHITECTING

It is now clear that the way software architectures developed in the context of agile development is deficient regarding how quality attributes are accommodated. Agile architecting begins with overlooking quality attributes' accommodation and ends with risky and expensive pursuit. Problems discussed through this research are the main inspiration for our suggested recipe here to achieve a framework to architect in the context of agile software development. The ingredients of the proposed recipe are clean architecting; light architecting.

- *Clean Architecting*: actually the morals of this trend are similar to those which triggered clean coding. Clean coding aims at enabling readability of code and hence backward tracing of a solution. This is exactly the same aim of clean architecting. A clear rationale of architectural decisions whenever being traceable through an architecture would guide through highlighting architecturally significant requirements (ASR)s. These ASRs include functional requirements, quality attribute requirements, design constraints, and any requirement that can influence architectural design decisions made to form an architecture. Clean coding aims at facilitating testing and discovering refactoring positions. We argue that clean architecting is about providing forward traceability of potential changes to be conducted. As changes are irresistible for an agile software system, and -as explained- changes have critical effects on architecture; there is a need to conduct change impact analysis. Change impact analysis is about analyzing potential consequences of changing a factor, component, connector, configuration upon other components, connectors, configurations, or upon the quality attribute achieved through the previous state before change. Change impact analysis also enables defining potential conflicts between various quality attributes. This way, clean architecting should also enable early evaluation of architectural design decisions; and this is aligned with agile software development mindset which encourages short feedback cycles and early changes' discovery.

- *Light Architecting*: it complements and enables clean architecting. To enable architecting while saving agile values, a light architecting process should be revolving around creating an initial minimal architecture at the preproduction or chartering level of a product development process, and leaving non-critical architectural decisions -that are more potential to changes and aren't about cross-cutting decisions- to be made incrementally and iteratively at the release and iteration levels. This highlights again the need for impact analysis to decide which decisions can affect a broader portion of software features. To eliminate the gap between customer requirements captured informally and architectures which are believed to be captured explicitly; software architects should be involved through the development life process. This way we can consider software architecting as a continuous process which is about role collaboration, and

which enables collaboration and communication among team members. Communicating "what a software product is" is a basic moral of architecting. Therefore; choosing the suitable way to share information among team members and to keep it for further usage, is purely a team free choice. This way we can consider informal diagrams on a whiteboard to be a viable document. Light architecting facilitates developing clean architectures thanks to two reasons. First, time constraints which result in dirty architecting are halted through incremental and iterative architecting. Second, when architecting becomes a shared responsibility among team members, it is easier to increase learning curve and enable making benefit of all team members' skills; therefore, there is more possibility to come up with a clean architecture.

A few approaches were suggested to overcome the absence of a mechanism to create flexible yet static architectures in the context of agile development. Most of suggested approaches revolve around systemizing and providing a context for conducting architectural refactorings. Among these approaches are developer stories writing [17], and Continuous Architectural Refactoring (CAR) & Real Architecture Qualification (RAQ) [18]. These approaches are criticized for accrediting refactoring as the only way to introduce quality attributes in resulting architectures, while ignoring the need for designing initial architectures upfront depending on careful analysis of concerns about quality attributes.

To achieve clean light architecting while planning for quality attributes in the context of agile software development, we suggest an architecting process which is comprised of a hybrid of three complementing methods. The first is *Quality Attribute Workshop (QAW),* because it facilitates capturing quality attribute requirements through collaborative brainstorming sessions, in the form of scenarios which is light enough to be placed into the product backlog. This way we argue this method is qualified to be integrated into a development process obtaining the agile mindset. The second method is *Attribute-Driven Design (ADD)*, because it enables developing an initial architecture incrementally based on quality attributes. The initial version will be based on highest priority requirements, and it will evolve through product development releases and iterations till the architecture reaches its final form. This way ADD also enables incorporation of requirements changes as they come up. ADD contains checkpoints where design is checked for being consistent with customer requirements. The third method is *Architecture Tradeoff Analysis Method (ATAM)*, which is a collaborative architecture evaluation method which early detection of architectural design decisions which are inconsistent with customer requirements. This method facilitates discovering conflicts and tradeoff points between quality attributes, and risks that can results whenever an architectural design decision is changed. This way, change impact analysis is facilitated and a team can be aware of their architectural decisions implications on various quality attributes. A proposed framework to achieve clean light architecting is under development and will be demonstrated in upcoming papers.

Considering quality attributes early while designing translates into business value, and we know that agile teams are pursuing business value in all their decisions and practices. By designing for including quality attributes right from the beginning, resulting architecture is shaped around a long term goal rather than short-sighted goals; besides, the number of architectural refactorings that would be needed over time is expected to be reduced. Agile methods would be more qualified for developing safety-critical systems, where performance and reliability are a must. Agile teams won't be able to go for large-scale products without an architecture that offers maintainability, reusability, scalability, interoperability, and other quality attributes that can be achieved through having a light clean architecture developed incrementally and iteratively.

5. CONCLUSION

Agile architects should advocate a development culture that values making architectural design decisions based on careful analysis of requirements and give a due care to quality attribute requirements in advance, especially that they do not change as rapidly as functional requirements. There is also a need for analyzing resulting architecture carefully to assess its adoption of needed quality attributes, and to deal with conflicts between several qualities at the earliest possible development level. Planning for quality attributes in advance not only prevents

problems of missed quality attributes and implications of redesigning a system to incorporate these quality attributes, but also provides a more stable basis for the architectural design as well. Planning an architecture based on quality attributes while keeping the process light and agile is not a myth. Comprising an architecting process which harmonizes both clean and light architecting is a dream that can be easily achieved if architecting and agile development morals are well-absorbed and tackled.

6. REFERENCES

[1] Galal, G. H., (1998), "Software architecting: from requirements to building blocks within an architectural style", *Workshop W2: Techniques, Tools and Formalisms for capturing and assessing Architectural Quality in Object-Oriented Software, the 12th European Conference on Object Oriented Programming (ECOOP'98),* Brussels, Belgium, 20-24 July.

[2] Bass, L., Clements, P. & Kazman, R., (2003), Software Architecture in Practice, Addison-Wesley Professional, Boston, USA.

[3] Albin, S. T., (2003), *The Art of Software Architecture: Design Methods and Techniques,* Wiley publishing, Indianapolis, Indiana, USA.

[4] Taylor, R., Medvidovic, N. & Dashofy, E., (2009), *Software Architecture: Foundations, Theory, and Practice,* Wiley publishing, Indianapolis, Indiana, USA.

[5] Barbacci, M. R., Klein, M. H. & Weinstock, C. B., (1997), "Principles for Evaluating the Quality Attributes of a Software Architecture", CMU, Software Engineering Institute, Pittsburgh, PA, USA.

[6] Mcgovern, J., Ambler, S. W., Stevens, M. E., Linn, J., Sharan, V. & JO, E. K., (2003), *A Practical Guide to Enterprise Architecture,* Prentice Hall, Upper Saddle River, New Jersey, USA.

[7] Faber, R., (2010), "Architects as service providers", *IEEE Software*, vol. 27, no. 2, pp. 33-40.

[8] Sharifloo, A. A., Saffarian, A. S. & Shams, F., (2008), "Embedding Architectural Practices into Extreme Programming", *proceedings of the 19th Australian Software Engineering Conference (ASWEC 2008),* Perth, Western Australia, Australia, 26-28 Mar., IEEE Computer Society, pp. 310-319.

[9] Barbacci, M., Ellison, R., Lattanze, A., Stafford, J., Weinstock, C. & Wood, W., (2003), "Quality Attribute Workshops (QAWs)", CMU Software Engineering Institute, Pittsburgh, PA, USA.

[10] Mohagheghi, P. & Conradi, R., (2004), "An empirical study of software change: origin, acceptance rate, and functionality vs. quality attributes", *Proceedings of the 2004 International Symposium on Empirical Software Engineering, (ISESE '04),* Redondo Beach, CA, USA, 19-20 Aug, IEEE, pp.7-16.

[11] Mockus, A. & Votta, L. G. (2000), "Identifying reasons for software changes using historic databases", *Proceedings of the International Conference on Software Maintenance,* San Jose, CA, USA, 11-14 Oct., IEEE, pp. 120-130.

[12] Williams, B. J. & Carver, J. C., (2007), "Characterizing Software Architecture Changes: An Initial Study", *proceedings of the First International Symposium on Empirical Software Engineering and Measurement, (ESEM'07),* Madrid, Spain, 20-21 Sept., IEEE, pp. 410-419.

[13] Perry, D. & Wolf, A., (1992) "Foundations for the study of software architecture", *ACM SIGSOFT Software Engineering Notes,* Vol. 17, No. 4**,** pp. 40-52.

[14] Garcia, J., Popescu, D., Edwards, G. & Medvidovic, N., (2009), "Identifying Architectural Bad Smells", *proceedings of the 13th European Conference on Software Maintenance and Reengineering, (CSMR '09),* Kaiserslautern, Germany, 24-27 March, Winter, A., Ferenc, R. & Knodel, J. (Eds.), IEEE, pp. 255-258.

[15] Bourquin, F. & Keller, R. K., (2007), "High-impact Refactoring Based on Architecture Violations", *Proceedings of the 11th European Conference on Software Maintenance and Reengineering, (CSMR '07),*Amsterdam, Holland, 21-23 Mar., IEEE, pp. 149-158.

[16] Coplien, J. O. & Bjornvig, G., (2010), *Lean Architecture: for Agile Software Development,* Wiley Publishing, Indianapolis, Indiana, USA

[17] Jensen, R. N., Moller, T., Sonder, P. & Tornehoj, G., (2006), "Architecture and Design in eXtreme Programming; Introducing Developer Stories", *proceedings of Extreme Programming and Agile Processes in Software Engineering, 7th International Conference (XP 2006),* Oulu, Finland, 17-22 June, Springer Verlag, pp. 133-142.

[18] Sharifloo, A. A., Saffarian, A. & Shams, F., (2008), "Embedding Architectural Practices into Extreme Programming", *proceedings of the 19th Australian Conference on Software Engineering (ASWEC'08)* Perth, WA, Australia, 26-28 March, IEEE, pp. 310-319.

Determining The Barriers Faced By Novice Programmers

Pranay Kumar Sevella *pranay_kumar.sevella@students.tamuk.edu*
Dept. of Electrical Engineering and Computer Science
Texas A&M University–Kingsville
Kingsville, 78363, U.S.A

Young Lee *young.lee@tamuk.edu*
Dept. of Electrical Engineering and Computer Science
Texas A&M University–Kingsville
Kingsville, 78363, U.S.A

Jeong Yang *jeong.yang@tamuk.edu*
Dept. of Electrical Engineering and Computer Science
Texas A&M University–Kingsville
Kingsville, 78363, U.S.A

Abstract

Most of the novice programmers find glitches at various phases while trying to complete a program in their Computer Science programming course. These phases can be while constructing the code, finding errors in the code at the time of compilation of the program, debugging these errors while executing the program. Novice programmers are unable to understand some of the concepts in programming. Computer Science programming course instructors are experiencing difficulty in finding these barriers faced by the students. These barriers are forcing students to drop programming course from their degree plan and becoming a concern to the professors teaching programming course. In this research ActivePresenter software is used. This software recorded the full motion video with crystal clear quality and helped in capturing screen shots automatically with a click of a mouse or pressing any key on the keyboard of the students who are trying to complete a programming assignment. By analyzing all the recordings collected from different students, these barriers are determined.

Keywords: Novice Programmer, Programming Barrier, Programming Education.

1. INTRODUCTION

Computer Science education researchers have tried since the 1980s to find how the Computer Science professors can effectively serve their students. Computer Science students who are considered as novice programmers registering in introductory programming courses are facing difficulties in programming and dropping from the course after attending few lecture classes. Computer Science programming instructors are facing difficulty in recognizing these barriers.

Purpose

The main purpose of this research is to find the barriers faced by the novice programmers. These barriers are considered to be challenges to the computer science programming instructors.

Method

All the students are given a C programming assignment to complete in one hour in the computer lab. While students are trying to complete the given assignment, their computer screens are automatically recorded in video. These video files, which record student's activity on the assignments, are analyzed and the barriers are determined.

The method used for determining these barriers is categorized into three stages.

Stage 0: Before Experiment Conducted
- Identifying a problem
- Preparing Classified Barriers
- Preparing questions to be asked to students
- IRB approval
- Setup experiments
- Find and invite novice programmer students
- Setup video recording on the computer lab

Stage 1: At the time of experiment
- Give an instruction to the students get consent to attend this experiment
- Recording the screen of the students trying to complete programming assignment

Stage 2: After the experiment
- Analyzing the recordings of each student individually
- Finding the barriers
- Classifying each barrier

Result
By thoroughly analyzing the recorded data, all the barriers faced by the Novice programmers are determined. And even the concepts which the Novice programmers face difficulty in understanding are also determined.

1.1 About C Programming
C language is the most popular and widely used programming language. Dennis Ritchie developed C language between 1969 and 1973. Most of the other programming languages are derived directly or indirectly from C language. C is sometimes considered as “High-Level assembly language”.

1.2 Novice Programmer
Any person, who is a beginner in programming, is called a Novice Programmer. In this research, freshmen students who have knowledge about C programming which is one of the courses in their previous semester are considered to be novice programmers.

2. RELATED WORK AND SUMMARY OF REFERENCES
2.1 Background
Previously, several studies have been done to understand the behavior of a Novice programmer. One such study is [1], the authors estimate the final grade of the student based on how the students do their respective lab assignments and how they react to errors they face while programming [3]. Authors have recorded the data affective states and behavioral states of the students. Then, they applied the linear regression using each behavioral activity and affective activity. In this study, the teaching assistants collected the data by periodically observing the activities of each student.

The authors [5] have developed a syntax error preprocessing and integrated semantic system that helps novice programmers to decipher the compile-time error messages. By this system, novice programmers stress more on issues related to design rather than issues related to implementation. All the syntax errors that are checked are collected by a survey [6]–[7] of former and current teaching instructors. There are discrepancies observed between the errors identified by the instructors and the errors encountered by the students. In this paper, the authors have developed a real-time system with machine controlled error collection that records a hundred percent of java errors in a database. All the errors encountered by the students, faculty, and users using this IDE are collected in the database. Hence in this paper, it is concluded that there are

five errors .The instructors have identified that those errors are also collected by the automated system. The system also verified that there are discrepancies between error encountered by students and errors identified by instructors. And through the use of this system, most common errors faced by Novice programmers are also identified.

The authors [11] claim that there are many big gaps brought by present methods of programming education in understanding programming by novice programmers. Due to these big gaps, the novice programmers are losing confidence in continuing with the programming course. Hence to reduce these big gaps the authors have introduced an AtoP method. In this method, novice programmers are given a checklist along with the programming assignment. This checklist consists of small exercises, which help students to write source code for the assignment. Teaching assistants help students [12-15] to complete all the small exercises in the checklist along with the programming assignment. The authors have also developed a site called AtoP which records all the results from interactive assessments of Novice programmers. Novice programmers can retrieve their data at any time. With this AtoP method the big gaps for students are reduced.

2.2 Recording Programmers' Activity

For recording the activity of programmer in the computer, software for recording the screen of the student is required. Once software is started, it starts recording the screen of the students so that all the activities of students can be recorded. This data is saved in video format.

3. RESEARCH QUESTION

3.1 Problem

Many of the novice programmers take C programming as their Computer Science programming course. But some students are dropping this course after attending few lecture classes. Students who are continuing with the course are facing difficulties in the subject. Some students are easily able to complete their assignments on their own. On the other hand, some students are unable to complete given assignments on their own and instead, they take the help of their Instructor or Teaching Assistants to complete their assignments.

3.2 Goal

The goal is to find the specific phases or activities in the programming course where the novice programmers find difficulty in understanding and also implementing some C programming concepts in their program. Once the instructor finds the part or activity of the program where the novice programmer is facing problems, the instructor can guide the students to overcome this problem.

3.3 Hypothesis

The main hypothesis that is expected before the start of the research is that most of the barriers faced by the students are conceptual barriers rather than basic programming barriers.

Main concept oriented barriers can be as follows:

- Loops
  The barriers that can be faced in loops are understanding the concept of loops, understanding syntax, format, working, and control flow of program in *for*, *while* and *do-while* loops.

- Nested statements
  . The barriers that can be faced in nested statements are understanding the concept of nested statements *if*, *else-if* and *nested-if* statements. Understanding syntax, format, control flow of program in these statements. Novice programmers cannot understand when *if block* statements are executed and when the *else* block is executed.

- Switch concepts
  The barriers that Novice programmers can face in *switch* concept are the format of *switch* statement, the syntax, defining each case in switch statement, and understanding the control flow of program.

- Arrays concepts
  The barriers that Novice programmers can face in the concept of arrays are initialization of arrays, addressing any particular element in an array and conditions for applying basic arithmetic operations to arrays.

- Concept of functions
  The barriers faced by Novice programmers in the concept of functions are defining a function and if the function has arguments, passing arguments to that function and calling the function.

- Files concepts
  The barriers faced by Novice programmers in files concept can be initializing a variable to the file, accessing data present in the file, and reading or writing data from a file.

The basic programming barriers can be as follows:

- Header files declaration
  Declaring header files correctly can be a barrier to Novice programmers. This comes under basic programming barrier because in every programming code, header files have to be declared.

- Syntax errors
  Writing correct syntax is a necessity for all the concepts in C programming. Since every concept has particular syntax, writing correct syntax is a basic programming barrier for novice programmers.

- Basic programming format
  Writing basic format of programming can also be one of the barriers faced by Novice Programmer.

4. METHOD

This approach is implemented on Under-Graduate students of Electrical Engineering and Computer Science Department of Frank H. Dotterwhich College of Engineering at Texas A & M University - Kingsville who have taken Computer Science as their major studies.

In order to collect the screen recordings of students and analyze their recorded videos, an approval from Institutional Review Board (IRB) granted by the office of Research and Sponsored Programs at Texas A & M University – Kingsville is required because humans are considered as subjects in this research.

The students who participated in this approach were asked to sign and agree a consent form that was approved by IRB. Some of the important points from the consent form are as follows:

- Students do not get any benefits if they participate in this research
- Students can back out from participation at any moment
- Students agree to record their screen while they are completing their programming assignment
- Students agree to review their screen recording
- All the students participating in this experiment are above 18 years of age.

Twenty students came forward to participate in the experiment. Six different programming assignments were prepared and one randomly selected assignment among these six assignments was given to each student.

Six programming assignments cover different topics of C programming. The assignments were based on following C programming concepts:

- *if* and *nested-if* statements
- Iteration concepts (using *for* or *while* or *do-while*)
- *switch* statements
- files (manipulating data in files using C programming)
- functions (defining a function, calling that function and passing arguments to function)
- arrays (applying mathematical operations to arrays)

The categorization of each programming assignment distributed to each student in the respective programming concepts is explained in the Table 1.

Program	Programming Concepts
Program 1	*if* and *nested-if* statements
Program 2	Iteration concepts
Program 3	*switch* statements
Program 4	Files concepts
Program 5	Functions concepts
Program 6	Arrays concepts

TABLE 1: Categorization of Each Programming Assignment In The Respective Programming Concepts.

Students are given one hour to complete their programming assignment. Each student is assigned with one computer system in a one-to-one student to computer ratio in the Computer Laboratory.
The basic method of approach can be categorized into three different categories. They are as follows:

- Collecting data
- Method of collecting data
- Analyzing data

The control flow of these three categories can be represented as follows:

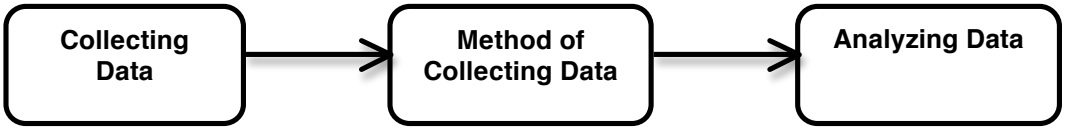


FIGURE 1: Control Flow of Method.

The detailed procedure of these three categories is explained below.

4. 1 Collecting Data
All the students from the Computer Science programming course who are interested in taking part in this research are given a C programming assignment. Students have to complete these programming assignments in the lab. Data is collected while the students are working on their

programming assignment. Data is collected individually from each novice programmer. The data collected is the screen recordings of the students.

4. 2 Method of Collecting Data

The data is collected by video recording and saved in video format. For Screen Recording no external hardware is required, as the software itself can record the screen of the novice programmers and the recorded data is saved in video format.

The software used to record the computer screen was ActivePresenter Free Edition Version 3.7.2 (Released 01.08.2013). It is a product of Atomi Systems, Inc. This software records the full motion video with crystal clear quality. This software also helps in capturing screenshots automatically with a click of mouse or by pressing any key on the keyboard. With this software, the collected data, which is in video format, can be exported to various other video formats like AVI, MP4, WMV, WebM.

Students use Dev C++ 4.9.9.2 version as their IDE (Integrated Developing Environment) for C programming. Dev C++ is used as IDE for both C and C++ Programming.

4.3 Analyzing Data

The assembled data is processed for analysis. The screen recording of each student is taken one at a time and properly analyzed.

Whenever any ambiguity is found, which is, when there is a long pause in student's programming activity, it means that student is facing some problem in writing further programming code. This pause can be starting from two minutes to any amount of time. This problem can be one of the barriers faced by the novice programmer.

Once the student completes programming code and tries to execute the program, compile time errors are found. When the student is unable to rectify these compile time errors, this can be considered as barrier faced by the novice programmer.

After the program is compiled, the student executes the program and gets unexpected output. This can also be considered as one of the barriers faced by the novice programmer.

Once all the barriers faced by the students are observed, then the programming code of the students is analyzed and the mistakes made by the student in their programming code are noted.

Later, how the student overcomes these barriers is observed. Students are allowed to utilize online resources by browsing the Internet and trying to correct their programming code.

This process is carried out for each individual student recording.

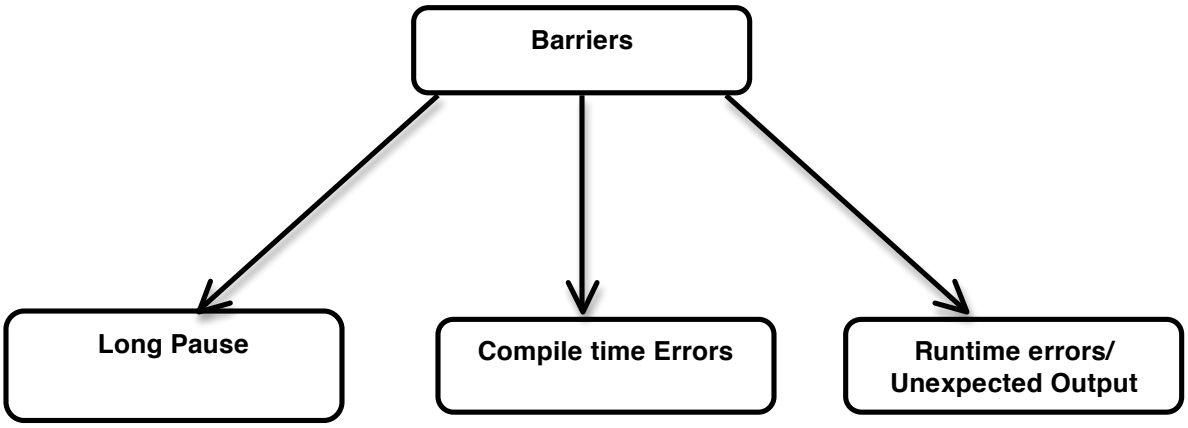


FIGURE 2: Classification of Barriers.

5. RESULTS

Out of twenty students who have participated in the experiment, screen recordings of only 16 students are collected. The rest of the four students recordings were not properly recorded and could not be used for the research purpose.

There are different problems that students faced while they were trying to complete their programming assignment. The problems faced by these novice programmers are mentioned in the following paragraphs.

Header Files

Header Files must be declared in all the programming code; hence the total number of students participated for this barrier are sixteen. And four students have made mistakes while declaring header files. Figure 3 indicates wrong declaration of header files.

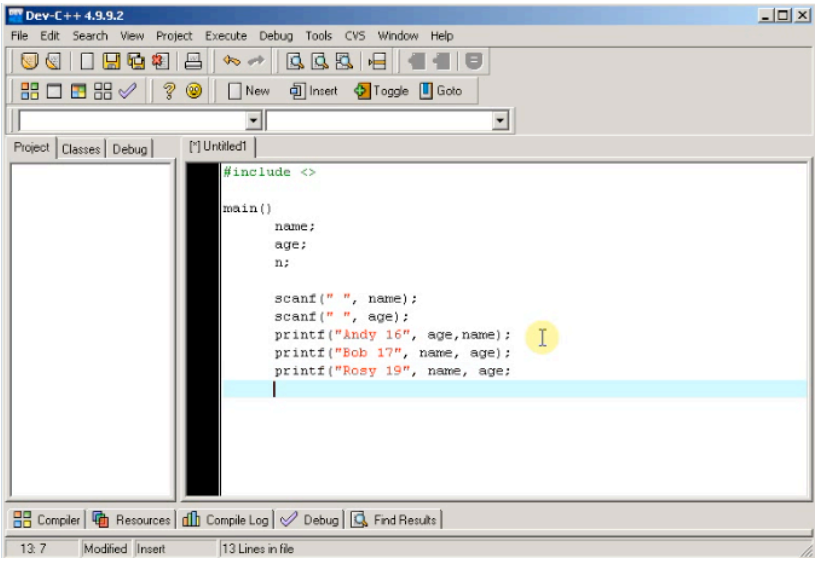


FIGURE 3: Screenshot 1.

Thinking for logic

Logic is required for writing any programming code; therefore, the total number of students participated in this barrier are sixteen. Four students have taken time to think logically for writing programming code.

Variables

Every programming code needs to have variables. These variables have to be declared with specific data types according to their usage in programming code; hence the total number of students participating in this barrier are sixteen. Three students declared wrong data types to the variables. One student did not assign data type to the variables. Another student is unable to use variables correctly in the program. Figure 3 indicates wrong declaration of variables.

printf* and *scanf

Every programming code that displays any output in the output screen has *printf* statements and every programming code that takes data from the user contains *scanf* statements. The total numbers of students participating in this barrier are sixteen. Two students were unable to write correct syntax for both *printf* and *scanf* statements. And four students were unable to write correct syntax for *scanf* statements. Out of four students, two students could not use *scanf* correctly. One student did not know how to take data given by the user and did not know the concept of *scanf*. Figure 3 indicates wrong declaration of *printf* and *scanf* .

Zero level

For this barrier, the knowledge of all the students is considered; hence the total number of students participating in this barrier are sixteen. Four students were unable to write basic programming code. Out of these four, two students tried to browse the Internet but were unable to understand the programming code found on the Internet.

Usage of Internet

All of the students were permitted to use the Internet for online resources to get basic syntax or any other programming related data from the Internet. Three students used online resources to get basic syntax and basic programming code. Out of these three, two students were able to complete their assignment successfully.

if* and nested *if

Three students got program1, which contained the concepts of *if* and *nested*-if statements. One student was unable to write conditional statements and could not use multiple *if* statements and put all *if* conditions in *printf* statements.

Iteration concept

Three students got program2, which contained the iteration concepts. All the students who got the programming assignment with iteration concept faced barriers. Two students faced difficulties in initializing the loop. One student got confused between the syntax of *for* loop and *while* loop and was unable to write looping statements. Another student completed his programming code but did not get the correct output and tried to manipulate code on variables in the code, but was unable to trace on which variable the operations were carried out. Figure 4 indicates wrong usage of while loop.

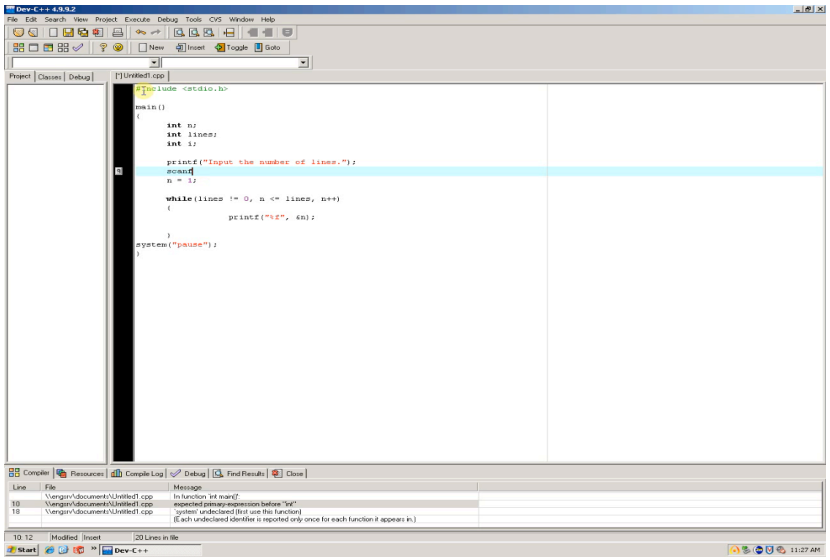


FIGURE 4: Screenshot 2.

***Switch* concept**

Three students got program3, which contained the *switch* concepts. All the three students were unable to define the *switch* statements and *switch* syntax and used Internet resources for format and syntax. Figure 5 indicates wrong declaration of *switch* statement.

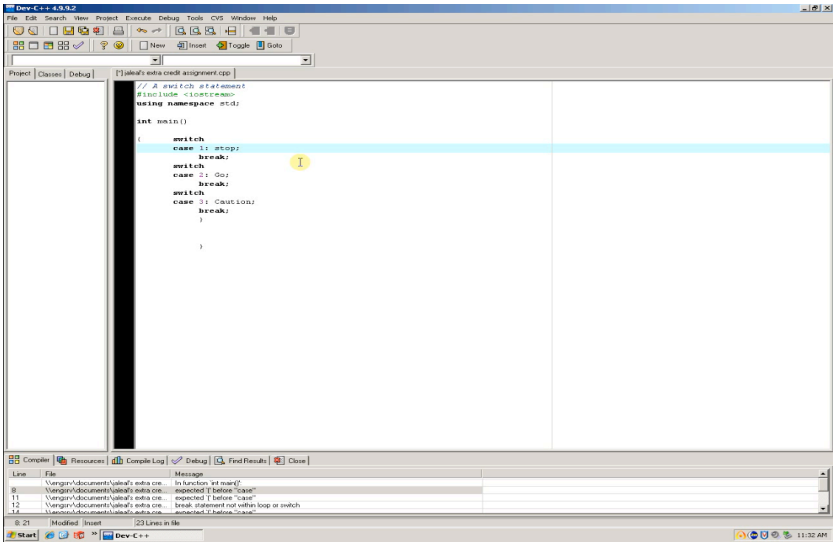


FIGURE 5: Screenshot 3.

Files concept

Two students got program4, which contained the concepts of files. There was only one student who faced difficulty in writing programming code and did not know the concept of files. Screen shot 4 indicates wrong usage of files concept.

Functions concept

Three students got program5, which contained the concepts of functions. Two students faced difficulties in function's concept. Out of these two, one student did not know the concept of functions and completed the program without using functions. The other student made mistakes while defining a function and was unable to call function with arguments.

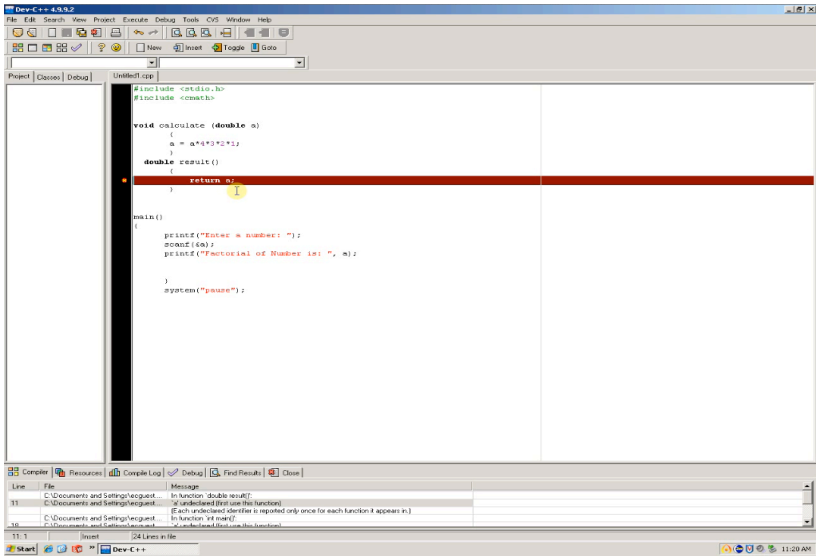


FIGURE 6: Screenshot 4.

Arrays concept

Two students got program6, which contained the concepts of arrays and both students were able to complete their programming assignment successfully . There were no barriers faced by the students in the arrays concept.

These were the barriers that were observed from the students programming recordings.

At the end of the experiment, it was observed that seven students out of sixteen students who participated in the experiment were able to complete their programming assignments successfully. Five students partially completed their assignment, and four students were in level zero and were unable to write the basic code.

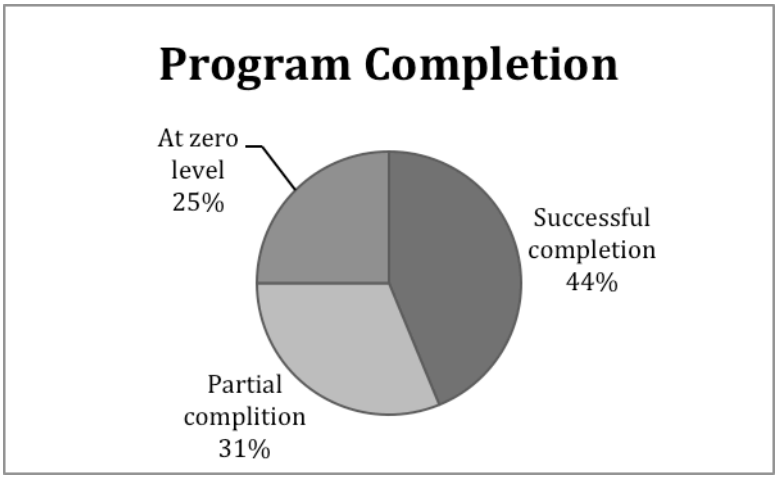


FIGURE 7: Pie Chart Representing Students to Program Completed.

There are 29 barriers that are observed in this research. Out of these 29, 19 barriers are basic programming based barriers and 10 are conceptual based barriers.

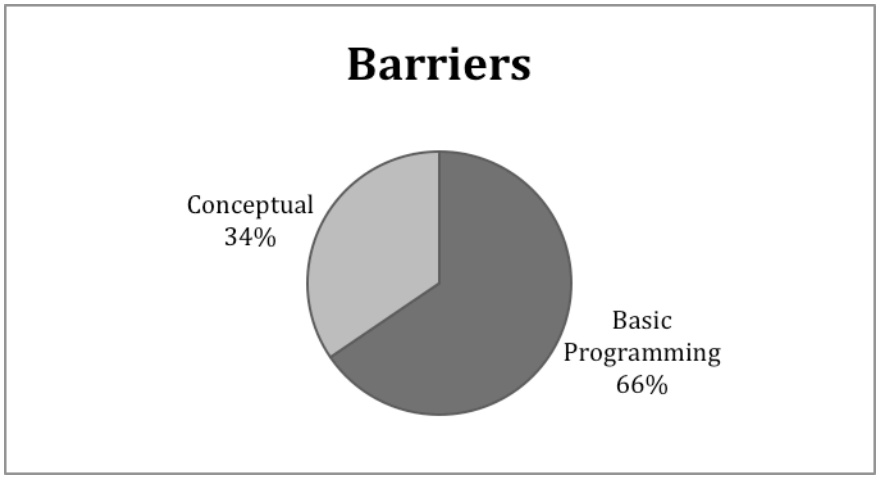


FIGURE 8: Pie chart of Barriers.

Table 2 represents the summarized list of all the barriers that are observed, the number of students who have faced these barriers, and the total number of students who were involved.

Barriers	Number of Students facing Barriers	Total Number of Students Participating
Header files	4	16
Thinking about logic	4	16
Variables	3	16
Issues with *printf* & *scanf*	8	16
Zero level	4	16
Usage of Internet	8	16
if and *nested-if*	1	3
Iteration concept	3	3
switch concept	3	3
Files concept	1	2
Functions concept	2	3

TABLE 2: Classification of All Barriers.

6. CONCLUSION

In this research, all the glitches that novice programmers face at different phases while completing a program in their Computer Science Programming course are determined. Also, the C programming concepts that novice programmers find difficulty in understanding and implementing in their C programming code are also determined.

After the successful completion of this research, it is found that the hypothesis predicted before the start of this is research is proved to be wrong. The predicted hypothesis at the start of this research is that most of the barriers faced by the novice programmers would be conceptual barriers than basic programming barriers. But after the research, it is proved that most of the barriers faced by novice programmers are basic programming barriers, rather than conceptual barriers.

The reason why most of the students drop from programming course can also be found out. This study even helps the programming instructors to teach C programming concepts in a more effective manner [16].

For future studies, the activities of the Novice Programmers while trying to complete the given assignment are also recorded with the help of an external video recorder. By this recording the facial gestures of the Novice Programmers is collected and, their behavioral states [1,18-21] can be determined. And with this recording, the other external resources that Novice Programmers use to overcome the barriers faced by them [17] are determined. These external resources can be referring to a Text Book or, referring to their classroom notes, or by taking help from their friend, or by taking help from Teaching Assistants.

7. REFERENCES

[1] Ma.Mercedes T. Rodrigo, Anna Christine M. Amarra, Sheryl Ann L.Lim, Ryan S. Baker, Thomas Dy, Sheila A. M. S. Pascua, Emily S. Tabanao, Matthew C. Jadud, Maria Beatriz V. Espejo-Lahoz, Jessica O. Sugay. *Affective and Behavioral Predictors of Novice Programmer Achievement*. In ITiCSE'09, July 6-9, 2009, Paris, France.

[2] Matthew A. Turk and Alex P. Pentland, *Face Recognition Using Eigenfaces.* In Computer Research and Development (ICCRD), 2011 3rd International Conference.

[3] Joni, S., Soloway, E., Goldman, R, and Ehrlich, K. 1983. *Just so stories: how the program got that bug*. SIGCUE Outlook 17,4(Sep. 1983), 13-26.

[4] Baker, R.S., Corbett, A.T., Koedinger, K.R., and Wagner, !A.Z. (2004) *Off-task behavior in the Cognitive Tutor classroom: When students "Game The System"*. ACM CHI 2004: Computer-Human Interaction, 383-390.

[5] James Jackson, Michael Cobb, Curtis Carver. 2004. *Identifying Top Java errors for Novice programmers*. 35th ASEE/IEEE Frontiers in Education Conference, Indianapolis, IN.

[6] Flowers, Thomas, Curtis Carver, and James Jackson. *Empowering Novice Programmers with Gauntlet*. Frontiers in Education, 2004.

[7] Hristova, Maria, Ananya Misra, Megan Rutter, and Rebecca Mercuri, *Identifying and Correcting Java Programming Errors for Introductory Computer Science Students*. ACM SIGCSE 2003. pp 19-23.

[8] Bruckman, Amy and Elizabeth Edwards. *Should we leverage natural- language knowledge? An Analysis of user errors in a natural-language- style programming language*. ACM SIGCHI 1999. pp 207-214.

[9] Chabert, Joan and T. F. Higginbotham, *An Investigation of Novice Programmer Errors in IBM 370 (OS) Assembly Language*. ACM 14th Annual Southeast regional Conference 1976. pp 319-323.

[10] Spohrer, James and Elliot Soloway, *Novice Mistakes: Are the folk Wisdoms correct?* Communications of the ACM 1986, pp 624-632.

[11] Dinh Dong Phuong, Yusuke Yokota, Fumiko Harada, Hiromitsu Shimakawa, (2010). *Graining and Filling Understanding Gaps for Novice Programmers*. 2010 International Conference on Education and Management Technology(ICEMT 2010)

[12] Paul Gross and Kris Powers, *Evaluating assessments of novice programming environments,* Proceedings of the first international workshop on Computing education research, USA , pages: 99 - 110 , October 2005.

[13] Charlie Daly, John Waldron, *Assessing the assessment of programming ability*, SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education, 2004, pages 210-213.

[14] Masoud Naghedolfeizi, Singli Garcia, Nabil Yousif, and Ramana M. Gosukonda, *Assessing long-term student performance in programming subjects*, December 2008, Journal of Computing Sciences in Colleges , Volume 24 Issue 2, page 241-247.

[15] Nghi Truong, Paul Roe and Peter Bancroft, Automated Feedback for “Fill in the Gap” Programming Exercises, January 2005, ACE '05: Proceedings of the 7th Australasian conference on Computing education - Volume 42 , pages 117-126.

[16] Douglas A. Kranch, Teaching the novice programmer: A study of instructional sequences and perception, May 2011, Springer Science+Business Media, LLC 2011

[17] Brad Myers and Andrew Ko, Studying Development and Debugging To Help Create a Better Programming Environment, 2003, CHI 2003 Workshop on Perspectives in End User Development.

[18] Matthew C. Jadud, An Exploration of Novice Compilation Behaviour in BlueJ, October 2006,

A thesis submitted to the University of Kent at Canterbury.

[19] Albert Lai and Gail C. Murphy, Behavioural Concern Modelling for Software Change Tasks, 2003, IEEE.

[20] Yuska P. C. Aguiar, Maria F. Q. Vieira, Edith Galy, ean-Marc Mercantini and Charles Santoni, Refining a User Behaviour Model Based on the Observation of Emotional States, 2011, COGNITIVE 2011 : The Third International Conference on Advanced Cognitive Technologies and Applications.

[21] Ben Shneiderman, Exploratory Experiments in Programmer Behavior, June 1975, Technical report No. 17 Submitted to Indiana University Bloomington.

Aspect Oriented Programming Through C#.NET

Harsha Bopuri *bopuri@gmail.com*
Director Business Applications/ IT Developments
IMATRIX Corp
North Brunswick, NJ 08902 USA

Prof. Dr. Raied Salman *rsalman.faculty@unva.edu*
University of Northern Virginia
Adjunct faculty, Computer Science Department
7601 Little River Turnpike, Annandale, VA 22003, USA

Abstract

.NET architecture was introduced by Microsoft as a new software development environment based on components. This architecture permits for effortless integration of classical distributed programming paradigms with Web computing. .NET describes a type structure and introduces ideas such as component, objects and interface which form the vital foundation for distributed component-based software development. Just as other component frameworks, .NET largely puts more emphasis on functional aspects of components. Non-functional interfaces including CPU usage, memory usage, fault tolerance and security issues are however not presently implemented in .NET's constituent interfaces. These attributes are vital for developing dependable distributed applications capable of exhibiting consistent behavior and withstanding faults.

Keywords: Aspect Oriented Programming, Cross Cutting Concerns.

1. INTRODUCTION

Aspect Oriented Programming (AOP) is a new development technology that permits separation of crosscutting concerns that have in the past proved difficult to implement using object oriented programming (OOP). According to [4], AOP is an elegant and simple construct with the ability of really altering the manner in which we develop software. It is a way of performing arbitrary code orthogonal to the primary purpose of a module, with the purpose bettering the encapsulation and reuse of the arbitrarily invoked code and the target module. Crosscutting concerns exists in most large systems; however, in others, the system may be redesigned to convert the crosscutting into an object. For aspect oriented programming though, the assumption is that crosscutting concerns exists in systems by default and cannot be transformed out of the system design.

1.1. Crosscutting Concerns

AOP divides crosscutting concerns into single parts referred to as aspects. An aspect represents a modular part of crosscutting implementation. Under AOP, we initially implement a project using an object oriented language such as Java or C# then independently handle crosscutting concerns by implementing aspects. In the end, an aspect weaver is used to integrate the both the code and the aspect into an executable file.

Component based programming is a simplistic method to compose systems out of units having contractually precise boundaries and unequivocal context dependencies. Since software components are developed by third parties, they can be deployed autonomously. Several distributed component frameworks exits including; Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), .NET framework among others. Despite the fact

that the implementation of intricate distributed systems is considerably simplified by these frameworks, there is limited support for techniques such as fault tolerance, reliability and security. Fault tolerance expansion for components needs to substitute abstraction and encapsulation with the execution explicit knowledge concerning a component's internal timing performance, memory usage, CPU usage, and communication and access models
AOP is best illustrated by example, the best one being event logging [20]. Let us say you have a class Foo and you want to write to a log file every time a particular system is called for auditing purposes or rudimentary performance statistics. You may normally write code like the following to meet this requirement:

```
public class Foo
        {
            protected EventLog eventLog;
            public Foo()
            {
                eventLog = new EventLog(); // create an event log
                eventLog.Source = "Foo Application"; // Name a Source
            }
            public void bar()
            {
                eventLog.WriteEntry("Bar method begin");
                // do bar()
                eventLog.WriteEntry("Bar method end");
            }
        }
```

Is there anything incorrect with this code? Historically, nothing is really incorrect. However, this is just because we are accustomed to writing codes like that. It is considered okay to incorporate EventLog code in the Foo class because before AOP there was no method of logging events without clearly calling event logging code from inside the class itself [12]. However, with the arrival of AOP, the code above would in fact be interpreted as very wrong, virtually prohibitive to incorporate in a Foo class. This is because everyone appreciates what should be integrated in a Foo class i.e. bar methods and not logging. Therefore, if the above was to be accomplished using AOP, it would look as follows:

```
       [EventLoggingAttribute]
        public class Foo : ContextBoundObject
        {
            public void bar()
            {
                // do bar()
            }
        }
```

The above shows that the code tangential to the bar() method is transferred to another place, particularly, the logging aspect. An aspect is executed without any more knowledge on the client's part and is functionality factored out of a client's module in an AOP - like approach. In the above example, the bar() technique does its job, regardless of other aspects. This is beneficial because it increases maintainability, improves reuse and encapsulation of both aspect and module code because of the introduction of decoupling.

2. METADATA AND REFLECTION IN .NET

Reflection refers to a programming language tool which permits access to type information during execution. This mechanism has been affected for various object oriented languages including java, C#.net and C++. .NET does not confine reflection to a single coding language but rather allows inspection of any .NET assembly using the reflection technology. Runtime type information

in .NET can be accessed in two different ways namely; language runtime library and the unmanaged metadata interfaces.

2.1 Reflection Through Runtime Library

Under this, the reflection classes are declared in System. Reflection namespace. The GetType method, which is a public method, has a return value object of the typeTypecontained in the namespace System. The following definitions are represented in each type-instance.

- Class definition
- Interface definition
- Value-class
-

Through reflection we are able to query about any type characteristic including the access modifiers. The structure of metadata is one of hierarchical nature in which the class System.Reflection.Assembly is at the highest level of the hierarchy. An assembly object relates to at least one dynamic libraries (DLLs) which forms the building block of the .NET unit in question. As indicated in the figure below, System.Reflection.Module is located on the second level of the hierarchy. Drilling down further the metadata tree represents type information for any of the foundations for the .NET virtual object system member.

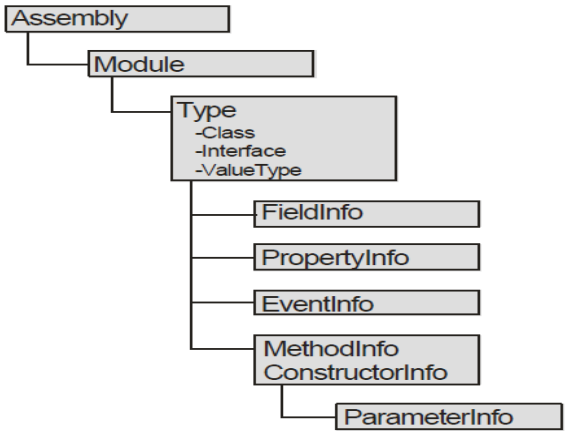


FIGURE 1: C#.NET Metadata Hierarchy.

In every circumstance, a class instance System.Reflection.MemberInfo stands for a single data element describing each of the below basic units constituting an object.

- Method (System.Reflection.MethodInfo)
- Constructor (System.Reflection.ConstructorInfo)
- Property (System.Reflection.PropertyInfo)
- Field (System.Reflection.FieldInfo)
- Event(System.Reflection.EventInfo)

2.2 Unmanaged Metadata Interface

These are an assortment of COM interfaces whose accessibility is external to the .NET environment. The interface definition is located in the COR.H, found in the Software development kit. The IMetaDataImport.IMetaDataAssemblyImportinterface aides in metadata accessibility on the .NET assembly level.

ImetadataDispenserinterface provides access to the metadata COM-interface IMetaDataImport.IMetaDataAssemblyImportinterface. The ImetadataDispenserinterface as the name suggests dispenses every types of additional metadata interfaces, permitting read and

write access to the .NET metadata. The dispenser is hence accessed through calls to the COM interface.

3. FAULT TOLERANCE REQUIREMENTS EXPRESSED BY C#

We shall demonstrate a simple calculator program in C# to explain how functional C# and non-functional C# (aspect) codes can be integrated together.

3.1 The Calculator Program

As shown by the below code snippet, the C# calculator has been accomplished within a class Calculator found in the namespace Calculate. Operands are stored as data-members Ope1 and Ope2. A public member method Add is implemented by the class.

```
namespace Calculate {
    public class Calculator {
        public Calculator() { Ope1=0; Ope2=0; }
        public double Ope1;
        public double Ope2;
        public double Add() { return Ope1+Ope2; }
    }
}
```

3.1.1 The Unmanaged Metadata Interfaces

The unmanaged metadata interfaces are a collection of COM interfaces that are accessible from “outside” of the .NET environment. You can access them from any Windows program. The interface definition can be found in the COR.H, which is contained in the platform software development kit (platform SDK).

IMetaDataImport.IMetaDataAssemblyImport interface is used for accessing metadata on the .NET assembly level. Access to this interface is obtained via a second interface, called IMetadataDispenser. As the name indicates, this interface “dispenses” all kinds of additional metadata interfaces, which allow read and write access to .NET metadata. Access to the metadata dispenser is obtained via calls to the COM system.

```
hr = CoCreateInstance(
    CLSID_CorMetaDataDispenser, 0,
    CLSCTX_INPROC_SERVER,
    IID_IMetaDataDispenser,
    (LPVOID*)&m_pIMetaDataDispenser );
    hr = m_pIMetaDataDispenser->OpenScope(
    wszFileName,
    ofRead,
    IID_IMetaDataImport,
        (LPUNKNOWN *)&m_pIMetaDataImport );
```

3.1.2 Tolerating Crash-Faults in the Calculator

The C# characteristic we are going to implement will entrench fault-tolerance to the calculator class we earlier wrote. The new modified class permits the independent creation and management of objects by clients. Due to the fact that we are using a simpler application, we assume that only crash faults occur at the object level thus we propose a proxy object for management of copies which makes up a single point of failure. Consequently, we assume that consistency of replicas can be maintained without the need of interaction with other replicas. We shall use C#.NET removing so as to spread the object copies across machine interfaces. This would create a distributed environment that tolerates both object and process faults. To maintain replica consistency, consensus rules such as voting scheme and master-slave replication scheme should be implemented. We outline C# attribute to define fault-tolerance requirements

```
[TolerateCrashFault (n)]
```

The parameter n denotes the number of objects crash faults that are likely to occur before the interruption of the component services. N+1 object replicas are needed so as to tolerate n crash-faults of objects. For our application an attribute has been used to expand the definition of the Calculator class.

```
[TolerateCrashFault (4)]
public class Calculator {
/* ... */
}
```

For our calculator application, five replicas would be created and the services continue running as long as one or more object persists.

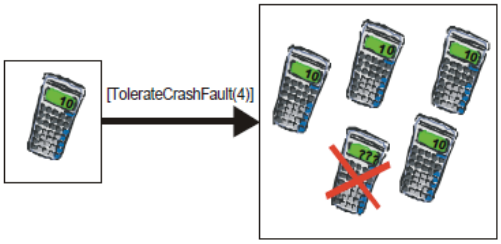


FIGURE 2: Replication in Space.

4. THE ASPECT WEAVER

This tool combines functional and aspect codes. For our case, we design a WrapperAssistant, which operates as our aspect weaver and generates snippets for replica administration. The Wrapper Assistant utilizes introspection and reflection techniques centered across the C#.NET CLR (Common Language Runtime) metadata to identify task signatures sent by a component and to create proxy classes for the exported classes. The TolerateCrashFault (n) attributecontrols the behavior of replica management scheme. The WrapperAssistantdialog provides the user with a list of classes that have been applied in a certain .NET assembly. Code will then be generated for the particular proxy class by the WrapperAssistantdepending on the class selected by the user in the list. The client programmer needs to make very few enhancements to the generated code; the programmer ought to modify just a line of code to utilize the added fault-tolerance enhancements.

```
using proxy;
// proxy namespace is imported by client
usingcalc;
//activates replica administration & fault-tolerance functionalities
void Calculate() {
    Calculator p = new Calculator (); // this comes from the proxy
    //namespace
    p.Ope1=4;
    p.Ope2=8;
    Console.WriteLine (c.Add ()); // writes to the console
}
```

4.1 Proxy Class Generation

Classes for replica management are generated by the WrapperAssistant inside the proxy namespace. The classes are instrumental in expanding the public classes employed in a particular component. For out calculator application, the below code is generated:

```
namespace proxy {
public sealed class Calculator:Calc.Calculator
    {
```

Every member role of the initial class is then overwritten with a version having an indistinguishable signature and routes the function calls to object copies instead of implementing them itself. The public variables of the initial class are declared as attributes in the tool-created proxy class. This would be as below for the

```
new public double Ope1 {
    get { /* ... */ }
    set { /* ... */ }
    }
```

The suitable count of base class interfaces has to be generated inside the constructor of the proxy class. The number is provided by the TolerateCrashFault attribute as shown below.

```
public sealed class
TolerateCrashFaults:System.Attribute {
    private int f_i;
    public TolerateCrashFaults(int i) {f_i=i; }
    public int Count
    { get { return f_i+1; } }
       }
```

The count of intolerable errors is internally recorded by the constructor. The count variable stores the number of copies that have to be created. Every overwritten member function in the class proxy routes its function-call to every occurrencereferenced in the collection. This would be represented as follows for the Add function.

```
public new double Add()
    { int i;
    double _RetVal=new double();
    for(i=0;i<_bc.Length;i++) {
    if(_bc[i]==null) continue;
    try { _RetVal=_bc[i].Add(); }
    catch(System.Exception) { _bc[i]=null; }
    }
       return _RetVal;}
```

4.2 Programmatic Tipping

Programmatic tipping is a technique used by high-level code weavers to assemble aspects from low-level devices. This technique allows addition of methods, types and fields programmatically. It is usually done using a compiled language. Below is an example of programmatic tipping.

```
public override void ProvideAspects(object targetElement,
    LaosReflectionAspectCollection collection)
{
  // Get the target type.  Type targetType = (Type) targetElement;
  // On the type, add a Composition aspect to implement
  // the IBindable interface.

  collection.AddAspect(targetType, new
AddBindableInterfaceSubAspect());

  // Add a OnMethodBoundaryAspect on each writable non-static property.
```

```
  foreach (PropertyInfo property in targetType.GetProperties())
  {
     if (property.DeclaringType == targetType &amp;amp;&amp;amp;
          property.CanWrite )
      {
          MethodInfo method = property.GetSetMethod();
          if (!method.IsStatic)
            collection.AddAspect(method,
                    new OnPropertySetSubAspect(property.Name, this));
       }
  }
   }
```

4.2.1 Custom Attributes

Custom attributes is an approach in which aspects are programmed as custom attributes and normally applied to fields, classes and methods. This example below implements transaction boundaries in .NET.

```
Imports PostSharp.Laos
Imports System.Transactions
 <Serializable>
Public NotInheritable Class TransactionScopeAttribute
    Inherits OnMethodBoundaryAspect
    Public Overrides Sub OnEntry(
         ByVal eventArgs As PostSharp.Laos.MethodExecutionEventArgs)
        eventArgs.State = New TransactionScope()
    End Sub
    Public Overrides Sub OnExit(
        ByVal eventArgs As PostSharp.Laos.MethodExecutionEventArgs)
        Dim transactionScope As TransactionScope = eventArgs.State
        If eventArgs.Exception Is Nothing Then
transactionScope.Complete()
        End If
       transactionScope.Dispose()
    End Sub
End Class
```

Transactional methods can be created using a new custom attribute as shown below.

```
<TransactionScope>
Sub Transfer(ByVal fromAccount As Account,
     ByVal toAccount As Account, ByVal amount As Decimal)
    fromAccount.Balance -= amount
    toAccount.Balance += amount
End Sub
```

It is required that custom attributes should be applied to each target explicitly but this is usually a challenge. This is because .NET languages do not offer the possibility to apply custom attributes to a set of code elements. A ‘multicast’ mechanism can be used to solve this problem. This is illustrated in the following code.

```
<assembly: TransactionScope(TargetTypes="MyNamespace.*")>
```

5. CROSS CUTTING CONCERN AND TANGLED CODE

Aspect Oriented programming is a technique used in programming to separate cross cut code across different modules in an application. Software applications are mostly developed to meet

business concerns. For example, a customer sales management software can have the requirements such as add, update, delete customer information, track sales and customers, ability to generate and print reports and a facility to send email. We will now use this requirement to elaborate cross cutting concern and tangled code.

The above business requirements are illustrated in the class diagram below.

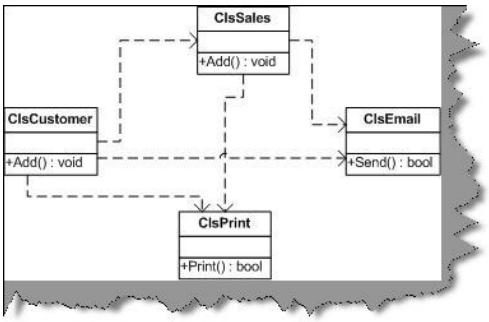


FIGURE 3: Business Concerns Class Diagram.

The diagram shows how the four concerns for application are met. According to object oriented programming (OOP), every object should only be concerned about its functionality. For example, "ClsSales" should only be concerned with maintaining sales information. From the above example, the core concerns are maintaining customer and sales records. The cross cut concerns are printing, sending email, logging and these spans across all the modules. This causes tangling of codes since there are several objects used across the modules. The codes are also called tangled in AOP methodology.

The cross cut code can be separated from the core modules by creating modules for cross cut and those for core functions separated. An AOP compiler can then generate a single executable even if the modules are separate. This process is called weaving and it is illustrated in the figure below.

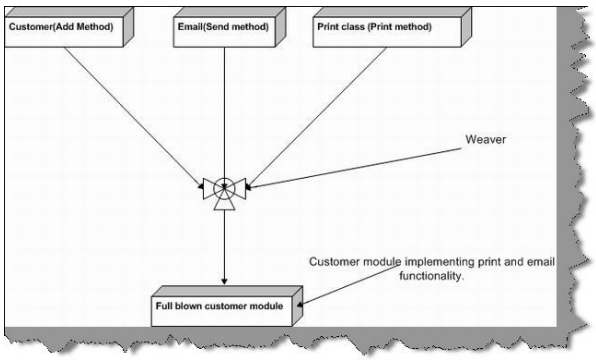


FIGURE 4: Weaving Modules.

AOP compilers are helpful in addressing the cross cut challenges. Types of AOP compliers are compile time weaving, link time weaving and run time weaving.

6. CONCLUSIONS AND FUTURE WORK

There is no good method of encapsulating without violating the integrity of the code. Aspect-Oriented Programming provides a solution to this challenge and enables better isolation of

responsibility, a more succinct code and encapsulation, all of which add to faster development times, increased comprehensibility and eased maintenance.

In this paper, it is pointed out, the pros and cons of the crosscutting concerns and the necessity of bringing the aspect oriented programming in to the limelight. This concept had a short term life in previous decade, but could not be extended, due to various reasons.

Though major software providers have chosen different approaches to achieve the above concept, this is the time to educate the IT world about efficiency of AOP using major .NET framework, and this paper does it to the best of my research.

As the support and implementation of AOP increases, the security concerns will also grow, which will increase the scale of fault tolerance. This will lead to further research to bring down the FT. Looking on the other side, there are few vendors who made attempts to integrate AOP with .NET framework and had also been successful to an extent. I believe this is the right time to make a smart move i.e. incorporate AOP in to corporate major programming concepts.

7. REFERENCES

[1] D. Box, “Essential COM”, 1998 Addison-Wesley, ISBN 0-201-63446-5

[2] K.Lieberherr, D. Orleans and J. Ovlinger. (2001). “Aspect-Oriented Programming with Adaptive Methods”, Communications of the ACM, Vol. 44, Issue 10.

[3] Groves, M. (2013). Aspect-Oriented Programming in .NET. Available: http://www.manning.com/groves/AOP.NETSampleCh01.pdf

[4] Clarke, S. & Jackson, A. (2004). SourceWeave.NET: Cross-Language Aspect-Oriented Programming. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.158.8736&rep=rep1&type=pdf

[5] G. Kiczaleset al. “Aspect Oriented Programming”, 1997. In proceedings of the European Conference on Object –Oriented Programming (ECOOP), Finland: Springer Verlag LNCS 1241.

[6] Kim, H. (2002). AspectC#: An AOSD Implementation for C#. Available: https://www.cs.tcd.ie/publications/tech-reports/reports.02/TCD-CS-2002-55.pdf

[7] Schult, W. & Polze, A. (2008). Design by Contract in .NET Using Aspect Oriented Programming. Available: http://www.tuplespaces.net/research/loom/Slides/DBC.pdf

[8] SUN Microsystems, “JavaBeans: The Only Component Architecture for Java Technology”, http://java.sun.com/products/javabeans/.

[9] Ferguson, D. (2004). Aspect. Net. Source Code… Available: http://www2.sys-con.com/itsg/virtualcd/dotnet/archives/0104/safonov/index.html

[10] Gnanasekaran, V. (2008). Rating of Open Source AOP Frameworks in .NET. Available: http://www.codeproject.com/Articles/28387/Rating-of-Open-Source-AOP-Frameworks-in-NET

[11] Miller, J. (2011). AOP with StructureMap Container. Available: http://weblogs.asp.net/thangchung/archive/2011/01/25/aop-with-structuremap-container.aspx

[12] Safonov, D. (2011). Aspect-oriented programming (AOP). Available: http://www.cs.helsinki.fi/en/event/58498

[13] Safonov, D. (2004). Aspect.NET: Concepts and Architecture. Available: http://www.aspectdotnet.org/articles/AspectDotNet2004_Article.pdf

[14] S. Hanenberg, R. Unland, “Concerning AOP and Inheritance”, Dept. of Mathematics and Computer Science University of Essen.

[15] Lee Breslau et al. (1999). Web caching and zipf-like distributions: Evidence and implications. In INFOCOM 1.

[16] Pei Cao and Sandy Irani.(1997). Cost-aware WWW proxy caching algorithms. In Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97), Monterey,CA.

[17] Constantinos A. Constantinides and Tzilla Elrad.(2000). On the requirements for concurrent soft-ware architectures to support advanced separation of concerns. The Workshop on AdvancedSeparation of Concerns in Object-Oriented Systems, OOPSLA.

[18] Li Fan, Pei Cao, Wei Lin, and Quinn Jacobson.(1999). Web prefetching between low-bandwidth clients and proxies: Potential and performance. In Measurement and Modeling of ComputerSystems.

[19] C. Fraleigh et al.(2001). Design and deployment of a passive monitoring infrastructure. Lecture Notes in Computer Science.

[20] Gustavo, A. & Grawehr, P. (2010). A Dynamic AOP-Engine for .NET. Available: ftp://ftp.inf.ethz.ch/doc/tech-reports/4xx/445.pdf

[21] Jangid, D. & Dave. R. (2012). Investigating the Web Application of AOP Using Aspect. Net Framework. Available: http://www.ijarcsse.com/docs/papers/8_August2012/Volume_2_issue_8/V2I800142.pdf

[22] Pérez, J. et.al (2010). Executing Aspect-Oriented Component-Based Software Architectures on .NET Technology. Available: http://www.sparxsystems.com/downloads/whitepapers/Aspect-Oriented_PRISMANET.pdf

Use of Cell Block As An Indent Space In Python

Hyung Jun Yoo *hjyoo760408@hotmail.com*
Department of Electrical Engineering and Computer Science
Texas A&M University-Kingsville
Kingsville, TX 78363, U.S.A

Young Lee *young.lee@tamuk.edu*
Department of Electrical Engineering and Computer Science
Texas A&M University-Kingsville
Kingsville, TX 78363, U.S.A

Abstract

Unlike most object oriented programming languages, Python does not use braces such as "{" and "}". Therefore, mixed tabs and spaces are used for indentation. However, they are causing problems in Python. Several approaches are applied to eliminate the problem that is not only for machine-readable form but also for proof reading for human. Often, characteristics of some programming behaviors are sometimes ambiguous. In such case, it is better for human to review what machines may not handle well, but the majority of python source code editors do not provide visually attracting environment. To provide the solution to both problems, concept of using cellblock in spreadsheet as an indent space for python source code is introduced.

Keywords: Source Code, Visualization, Spreadsheet, Python, Stereopsis Algorithm.

1. INTRODUCTION

Spreadsheet contains a table of values arranged in rows and columns where each value may have predefined relationship with values in other cells [1]. Python is an interpreted, object-oriented, high-level programming language that runs in major operating systems such as, Linux/Unix, Windows, OS/2, Mac, and Amiga [2]. It can be integrated with COM, .NET, COBRA objects, or implement Python for Java Virtual Machine (JVM) using Jython [3].

There are numbers of studies concerning software visualization, where pictures, graphs or animations can acquire information about specific program. Although significant numbers of software visualization products are released, most of those products are not compatible among each other when it comes to sharing their data. One approach to solve this problem is to use a widely used application with powerful feature that is already built in it [5]. This study suggests how to use such application to write and fix errors as a first step to use its source code as a data to visualize software. Calculating numbers with visualizing it result in graphs are magnificent, but if source codes can be analyzed and counted by given conditions, it can be a commonly used program which is easy to get access and since it is widely used, sharing data between users are not difficult. Python programming language was chosen in here since it had a problem that most languages did not have and thought it could benefit more from using this method. Indent style is unnecessary for most programming languages. Rather, it is used for more clear understanding of how the source code is constructed in human readable form. For indentation, using multiple spaces and tabs were common to programmers in most programming languages. However, spaces used for indentation may vary from time to time, different programming language may use different number of spaces, or they may diverge among source code editors which programmers use preferably.

Unlike other programming languages that use braces such as "{" and "}" for block of codes, Python relies on indentation. An issue arises when several programmers get involved with editing the same source code while they use different editors that has different spaces step up for indentation. This study analyzes the Python source code and displays its indentation level on spreadsheet to resolve the confusion in mixed indent style as well as, to examine potential problem.

UNIX users or old python programmers used adding 8 spaces for each indentation, but current recommendation for one indentation is using 4 spaces [4]. This can cause problematic python source code if a programmer decides to reuse old source code when they have different size of indent block. Moreover, numbers of Microsoft Windows based Microsoft Visual Studio programmers use tab as indentation, which is default indent block, which is used by Visual Studio 2005/2007.

2. USING CELL BLOCK IN SPREADSHEET AS INDENT SPACE FOR PYTHON

2.1 Motivation

This People from different background have different way of writing source code. However, when it comes to indent style, they should all follow the same rule. Changing a way of doing things, which have been done for a while, may not be easy to fix. Eventually, there are going to be errors made. When using the cell as an indented block, this problem can be solved. A program can be written in a way to count numbers of spaces for indentation in python source code. It should count the spaces for the indentation wherever it first occurs and save its one block of indentation information on a file, so that it can be imported from spreadsheet later. If indented spaces are 8 spaces, 16 spaces, 24 spaces which is multiple of 8, then 8 spaces should be marked as one block of indentation, 16 spaces as 2 blocks and so on.

2.2 Proposal

If mismatch spaces of indent style is found, a program should correct it assuming a small mistake, but also inform the programmer that such error was found. When 8 spaces were used in an indentation and 7 spaces of indentation were found, it is easy to tell that one space is missing. However, some programmers use 4 spaces for indentation, which makes it when 6 spaces of indentation was found, it can be confusing to tell right away whether the indentation was meant for 4 or 8 spaces. One of the techniques to solve this problem is what we decided to call it a "Stereopsis" algorithm.

Stereopsis algorithm is similar to visualization spreadsheet where users lay out two data sets next to each other to compare to data groups [6].

One of the outputs, a "right eye view", of this program will mark indentation that if in 8 spacing indent style, a line with less than 8 spaces will be marked as no indentation, less than 16 spaces will be marked as 1 indented block, and less than 24 spaces will be marked as 2 indented blocks. The other output, a "left eye view", will show a line with spaces greater than 0 will get 1 indented block, greater than 8 spaces will get 2 indented blocks, and greater than 16 spaces will get 3 indented blocks. If source code has no error in indent style, both output will produce same result, but if there is an indent spacing that is different from others, error flag is raised which it can be check by the programmer later.

3. CASE STUDY

```
#Halstead Complexity output
#
if ((h == True and gotfile == True)or (s != True and c != True and h != True and gotfile == True)):

        print
        print '====================='
        print 'Halstead\'s Complexity'
        print '====================='
        print
        print ('Class Name').ljust(27),
        print ('Operators').ljust(12).rjust(4),
        print ('Operands').ljust(10).rjust(4),
        print ('Unique').ljust(10).rjust(4),
        print ('Unique')
        print ('').ljust(51),
        print ('Operators').ljust(10).rjust(4),
        print ('Operands')
        print '-------------------------------------------------------------------------'

        t_hal_operator_num = 0
        t_hal_operand_num = 0
        t_hal_unique_operator_num = 0
        t_hal_unique_operand_num = 0

        for key in result.keys():
                try:
                        print (key+'').ljust(29),
                except:
                        #print ('Total number of ').ljust(30),
                        dummy = 2
                if (key):
                        print (str(result[key].hal_operator_num)).rjust(5).ljust(11),
                        t_hal_operator_num += result[key].hal_operator_num
                        print (str(result[key].hal_operand_num)).rjust(4).ljust(10),
                        t_hal_operand_num += result[key].hal_operand_num
                        print (str(result[key].hal_unique_operator_num)).rjust(4).ljust(10),
                        t_hal_unique_operator_num += result[key].hal_unique_operator_num
                        print (str(result[key].hal_unique_operand_num)).rjust(4)
                        t_hal_unique_operand_num += result[key].hal_unique_operand_num
                 else
                        printf ('error occured')
        print '-------------------------------------------------------------------------'
        print ('Total Number').ljust(29),
        print str(t_hal_operator_num).rjust(5).ljust(11),
        print str(t_hal_operand_num).rjust(4).ljust(10),
        print str(t_hal_unique_operator_num).rjust(4).ljust(10),
        print str(t_hal_unique_operand_num).rjust(4)
        print
```

FIGURE 3.1: Python Source Code.

3.1 Limitation of Python Source Code Editor

Figure 3.1 shows one section of typical python source code, but there is an error, which might be hard to catch if condition statements or a loop contains several more lines of code. A programmer has to rely on the compiler to check the location of an error to find it. To be able to check errors in the source code without compiling save a lot of time, but before suggesting a solution to this, we will first go over with how an error is found and fixed just by using the source code and a compiler. After source code is ready, a programmer run compiler to see if there are any errors. If errors are found, the compiler usually returns line numbers. From compiling source code, we got an error and the line number where it points to else statement. Figure 3.2 shows where the error was in Figure 3.1. However, if this code is written or exported to a spreadsheet, it can be found fairly easy. Since spreadsheet has option to display vertical and horizontal line to clearly show the individual blocks of cells, we can use this to display the source code to catch an error.

3.2 Stereopsis Algorithm

3.2.1 Left Eye View

When this source code is passed through Stereopsis algorithm, since there is an error in the source code, two different results will occur. Following two figures will show how they appear and what can be done to fix it. These two figures will display mostly where the error occurred.

```
#Halstead Complexity output
#
if ((h == True and gotfile == True)or (s != True and c != True and h != True and gotfile == True)):

        print
        print '====================='
        print 'Halstead\'s Complexity'
        print '====================='
        print
        print ('Class Name').ljust(27),
        print ('Operators').ljust(12).rjust(4),
        print ('Operands').ljust(10).rjust(4),
        print ('Unique').ljust(10).rjust(4),
        print ('Unique')
        print ('').ljust(51),
        print ('Operators').ljust(10).rjust(4),
        print ('Operands')
        print '-------------------------------------------------------------------------'

        t_hal_operator_num = 0
        t_hal_operand_num = 0
        t_hal_unique_operator_num = 0
        t_hal_unique_operand_num = 0

        for key in result.keys():
                try:
                        print (key+'').ljust(29),
                except:
                        #print ('Total number of ').ljust(30),
                        dummy = 2
                if (key):
                        print (str(result[key].hal_operator_num)).rjust(5).ljust(11),
                        t_hal_operator_num += result[key].hal_operator_num
                        print (str(result[key].hal_operand_num)).rjust(4).ljust(10),
                        t_hal_operand_num += result[key].hal_operand_num
                        print (str(result[key].hal_unique_operator_num)).rjust(4).ljust(10),
                        t_hal_unique_operator_num += result[key].hal_unique_operator_num
                        print (str(result[key].hal_unique_operand_num)).rjust(4)
                        t_hal_unique_operand_num += result[key].hal_unique_operand_num
                else
                        printf ('error occured')
        print '-------------------------------------------------------------------------'
        print ('Total Number').ljust(29),
        print str(t_hal_operator_num).rjust(5).ljust(11),
        print str(t_hal_operand_num).rjust(4).ljust(10),
        print str(t_hal_unique_operator_num).rjust(4).ljust(10),
        print str(t_hal_unique_operand_num).rjust(4)
        print
```

Extra Space Causing Error

FIGURE 3.2: Line Drawn for Finding an Error.

```
if (key):
        print (str(result[key].hal_operator_num)).rjust(5).ljust(11),
        t_hal_operator_num += result[key].hal_operator_num
        print (str(result[key].hal_operand_num)).rjust(4).ljust(10),
        t_hal_operand_num += result[key].hal_operand_num
        print (str(result[key].hal_unique_operator_num)).rjust(4).ljust(10),
        t_hal_unique_operator_num += result[key].hal_unique_operator_num
        print (str(result[key].hal_unique_operand_num)).rjust(4)
        t_hal_unique_operand_num += result[key].hal_unique_operand_num
        else
        printf ('error occured')
```

FIGURE 3.3: Source Code from the Left Eye Method.

The left eye method inserts a block of indentation when default number of spaces in indent style is counted. Any spaces from 0 to 7 will get no block of indentation. Spaces from 8 to 15 will get one block of indentation. Spaces from 16 to 23 will get two block of indentation. This python code used 8 spaces for one block of indentation. The left eye method inserted 2 blocks of indentation when if statement had 16 spaces in front. However, 17 spaces where found before else statement started, thus it added 3 blocks of indentation. In the Left Eye method, one example of formula may be written as: Number of blocks of indentation equals spaces in front of current line of code divided by default indented spaces and add one if the remainder is greater than zero.

3.2.2 Right Eye View

Next figure is an output of the Right Eye method.

```
if (key):
        print (str(result[key].hal_operator_num)).rjust(5).ljust(11),
        t_hal_operator_num += result[key].hal_operator_num
        print (str(result[key].hal_operand_num)).rjust(4).ljust(10),
        t_hal_operand_num += result[key].hal_operand_num
        print (str(result[key].hal_unique_operator_num)).rjust(4).ljust(10),
        t_hal_unique_operator_num += result[key].hal_unique_operator_num
        print (str(result[key].hal_unique_operand_num)).rjust(4)
        t_hal_unique_operand_num += result[key].hal_unique_operand_num
else
        printf ('error occured')
```

FIGURE 3.4: Source Code from the Right Eye Method.

It is possible to find an error from examining from Figure 3.4 only but, when two different source codes are present, finding an error can be more convenient, because a programmer does not have to go through whole source code but find the difference in those source codes and make a correction. In this case, a programmer should select a result from the Right Eye method and choose it as code to be use. In the Right Eye method, formula can be written as: Number of blocks of indentation equals spaces in front of current line of code divided by default indented spaces.

3.2.3 Simple Error Correction

For simple error correction and handling, an error correction algorithm can be implemented while migrated to spreadsheet to remove problem in Figure 3.1. If the default indentation has 4 spaces, put no indentation to a code that has 0~1 spaces in front, level one indentation to a code with 3~5 spaces, and level two indentation to a code with 7~9 spaces in front. This will get rid of problems when a space bar is pressed a little bit more (or less) than intended. In Figure 3.1, default indentation is 8 spaces, which makes code with 0~2 spaces in front will have no indentation, code with 6~10 spaces will have level one indentation, code with 14~18 spaces will have level two indentation, and so on. This case, the result of simple error correction has same output as the right eye view. Indent spaces that is close to middle of the first indentation and next indentation should be alerted to a programmer, because it is risky to depend wholly on the computer to correct it automatically. These errors are sometimes ambiguous even to the programmer hence, the decision should be made manually by the programmer with clear information by presenting both result from Stereopsis algorithm.

3.2.4 Stereopsis Result

A python source code passed through Stereopsis algorithm will have 3 output files in Comma Separated Value (CSV) files and a message displayed on monitor screen. “_l” is added to file name which contains output through the left eye view whereas, “_r” and “_sec” will be added to output files passed through the right eye view and simple error correction, respectively. Messages on the monitor should contain location of indent mismatch from default indent spacing, suggested

indent space, and which indent style should be used. With this information, a python programmer should have sufficient enough knowledge to correct the source code. To demonstrate how the spaces were either indented or not, a sample file name “test.txt” was created. This file contains a sentence with tabs and whitespaces inserted in front.

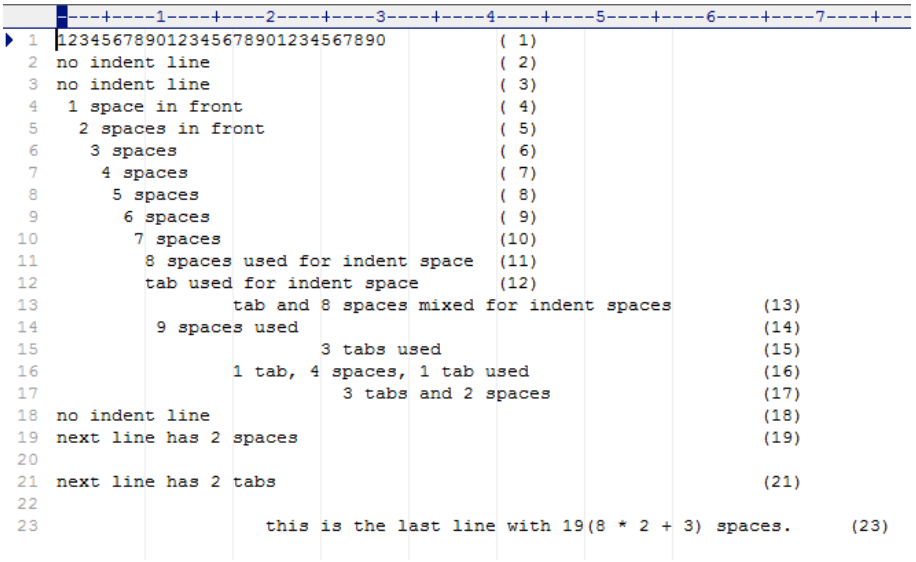


FIGURE 3.5: test.txt.

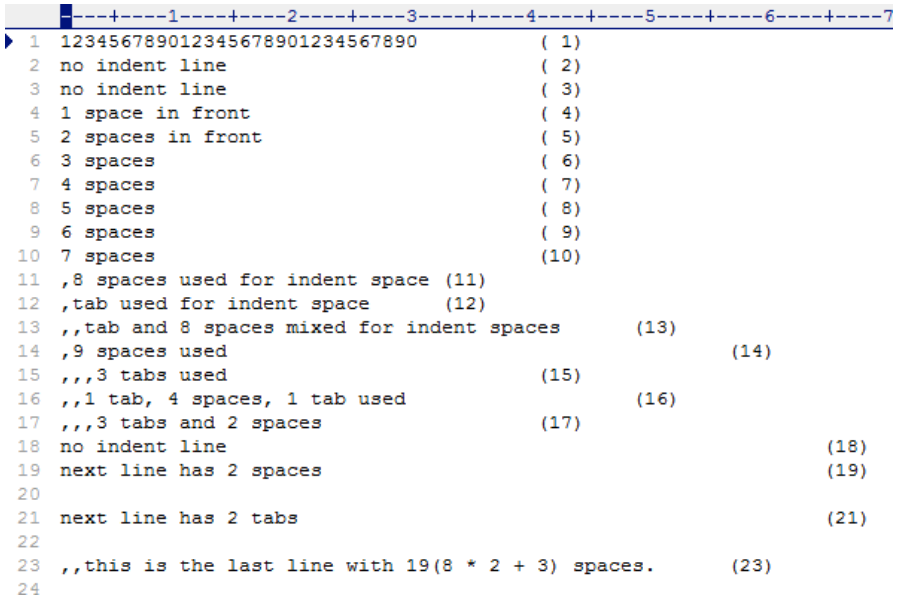
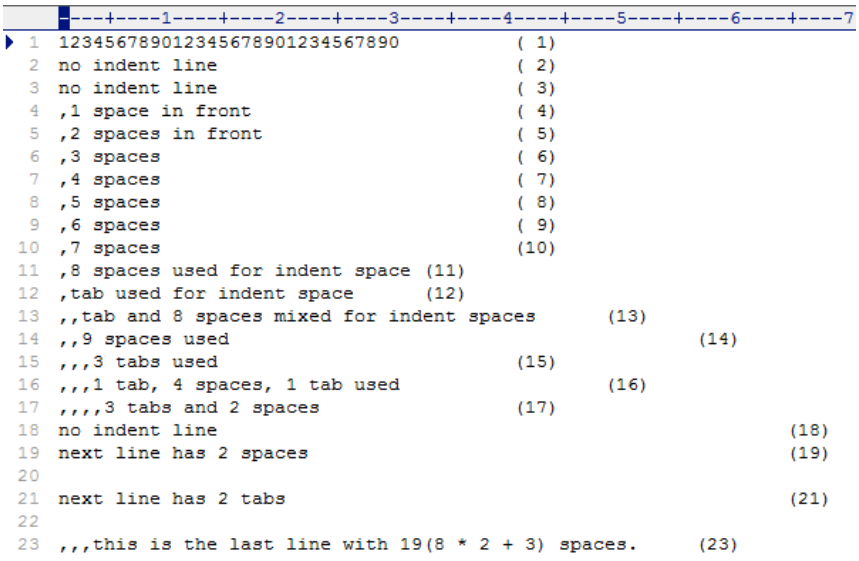


FIGURE 3.6: test_r.csv.

When passed through Stereopsis algorithm, 3 outputs are generated along with result message. In Figure 3.6, file “test_r.csv” will mark cell block when there are 8 or more spaces. Cell blocks are added if whitespaces are 8 to 15 spaces. Two cell blocks are added if whitespaces are 16 to 23. Tabs are treated as 8 spaces.

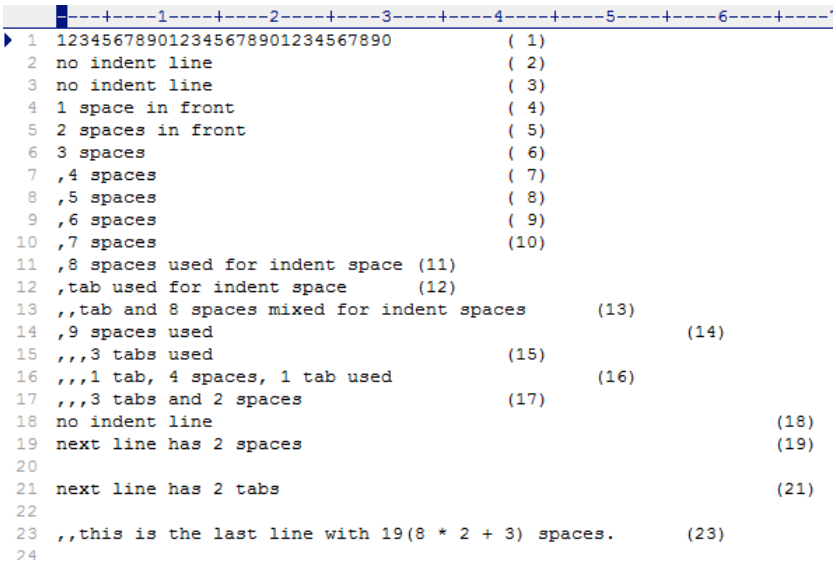
File “test_l.csv” will mark a cell block in front, if there is any space greater than one. Cell blocks are added if whitespaces are 1 to 8 spaces. Two cell blocks are added if whitespaces are 9 to 16. Tabs are treated as 8 spaces just like in the right eye view. Figure 3.7 shows the result of “test.txt” in “test_l.csv” with appropriate cell block inserted.

File “test_sec.csv” is the one that most programmers may like, since it corrects any little mistakes in intent spacing. This is not true if mistakes were made outside the correction range. One of the examples that simple error correction cannot correct is when indentation error exceeds more than 8 spaces. However, Stereopsis algorithm will still catch the mismatching indent style and inform the programmer where the error was made.



```
    ----+----1----+----2----+----3----+----4----+----5----+----6----+----7
 1  12345678901234567890123456789O          ( 1)
 2  no indent line                          ( 2)
 3  no indent line                          ( 3)
 4  ,1 space in front                       ( 4)
 5  ,2 spaces in front                      ( 5)
 6  ,3 spaces                               ( 6)
 7  ,4 spaces                               ( 7)
 8  ,5 spaces                               ( 8)
 9  ,6 spaces                               ( 9)
10  ,7 spaces                               (10)
11  ,8 spaces used for indent space (11)
12  ,tab used for indent space      (12)
13  ,,tab and 8 spaces mixed for indent spaces      (13)
14  ,,9 spaces used                                      (14)
15  ,,,3 tabs used                          (15)
16  ,,,1 tab, 4 spaces, 1 tab used                  (16)
17  ,,,,3 tabs and 2 spaces                 (17)
18  no indent line                                                (18)
19  next line has 2 spaces                                        (19)
20
21  next line has 2 tabs                                          (21)
22
23  ,,,this is the last line with 19(8 * 2 + 3) spaces.     (23)
```

FIGURE 3.7: test_l.csv.



```
    ----+----1----+----2----+----3----+----4----+----5----+----6----+----
 1  12345678901234567890123456789O          ( 1)
 2  no indent line                          ( 2)
 3  no indent line                          ( 3)
 4  1 space in front                        ( 4)
 5  2 spaces in front                       ( 5)
 6  3 spaces                                ( 6)
 7  ,4 spaces                               ( 7)
 8  ,5 spaces                               ( 8)
 9  ,6 spaces                               ( 9)
10  ,7 spaces                               (10)
11  ,8 spaces used for indent space (11)
12  ,tab used for indent space      (12)
13  ,,tab and 8 spaces mixed for indent spaces      (13)
14  ,9 spaces used                                       (14)
15  ,,,3 tabs used                          (15)
16  ,,,1 tab, 4 spaces, 1 tab used                  (16)
17  ,,,3 tabs and 2 spaces                  (17)
18  no indent line                                                (18)
19  next line has 2 spaces                                        (19)
20
21  next line has 2 tabs                                          (21)
22
23  ,,this is the last line with 19(8 * 2 + 3) spaces.      (23)
24
```

FIGURE 3.8: test_sec.csv.

Any mistakes equal to or smaller than 8 spaces or one tab will be corrected, but if indent error is greater than 8 spaces, the programmer may use output from the left or right eye view to make an adjustment.

Since spreadsheet eliminates and mismatch in indent style, this is a good way to write a python source code. However, direct python compiler is still needed to get the python to work, which is written, in spreadsheet format. This should be included in the future work, until then, python source code in spreadsheet can be exported to text file to be run.

Following figures 3.9, 3.10 and 3.11 shows each python source code in spreadsheet.

	A	B	C	D	E	F	G	H
1	123456789012345678901234567890(1)							
2	no indent line(2)							
3	no indent line(3)							
4		1 space in front(4)						
5		2 spaces in front(5)						
6		3 spaces(6)						
7		4 spaces(7)						
8		5 spaces(8)						
9		6 spaces(9)						
10		7 spaces(10)						
11		8 spaces used for indent space(11)						
12		tab used for indent space(12)						
13			tab and 8 spaces mixed for indent spaces(13)					
14			9 spaces used(14)					
15				3 tabs used(15)				
16				1 tab	4 spaces	1 tab used(16)		
17					3 tabs and 2 spaces(17)			
18	no indent line(18)							
19	next line has 2 spaces(19)							
20								
21	next line has 2 tabs(21)							
22								
23				this is the last line with 19(8 * 2 + 3) spaces.(23)				
24								

FIGURE 3.9: test_l.csv in Spreadsheet.

	A	B	C	D	E	F	G
1	123456789012345678901234567890(1)						
2	no indent line(2)						
3	no indent line(3)						
4	1 space in front(4)						
5	2 spaces in front(5)						
6	3 spaces(6)						
7	4 spaces(7)						
8	5 spaces(8)						
9	6 spaces(9)						
10	7 spaces(10)						
11		8 spaces used for indent space(11)					
12		tab used for indent space(12)					
13			tab and 8 spaces mixed for indent spaces(13)				
14		9 spaces used(14)					
15				3 tabs used(15)			
16			1 tab	4 spaces	1 tab used(16)		
17				3 tabs and 2 spaces(17)			
18	no indent line(18)						
19	next line has 2 spaces(19)						
20							
21	next line has 2 tabs(21)						
22							
23			this is the last line with 19(8 * 2 + 3) spaces.(23)				
24							

FIGURE 3.10: test_r.csv in Spreadsheet.

	A	B	C	D	E	F	G
1	123456789012345678901234567890(1)						
2	no indent line(2)						
3	no indent line(3)						
4	1 space in front(4)						
5	2 spaces in front(5)						
6	3 spaces(6)						
7		4 spaces(7)					
8		5 spaces(8)					
9		6 spaces(9)					
10		7 spaces(10)					
11		8 spaces used for indent space(11)					
12		tab used for indent space(12)					
13			tab and 8 spaces mixed for indent spaces(13)				
14		9 spaces used(14)					
15				3 tabs used(15)			
16				1 tab	4 spaces	1 tab used(16)	
17				3 tabs and 2 spaces(17)			
18	no indent line(18)						
19	next line has 2 spaces(19)						
20							
21	next line has 2 tabs(21)						
22							
23			this is the last line with 19(8 * 2 + 3) spaces.(23)				
24							

FIGURE 3.11: test_sec.csv in Spreadsheet.

```
#
if ((h == True and gotfile == True)or (s != True and c != True and h != True and gotfile == True)):
class ContainerType(object):
    '''ContainerType is a type to describe abstracted
    tokens that contain other tokens'''

    # a list of valid operators
    OPERATORS = reserved.keys() + ['+','-','*','/','++','--',';',':','[',']','(',')','{','}','&&','||','<=','>=','<','>','==','=','!']

    # a list of tokens that are neither operators nor operands
    IGNORE = [',']

    def __init__(self, name=None):
        '''ContainerType([str]) -> ContainerType'''
        self.name = name
        self.tokens = []
        self.counter = 0
        self.bodyless = False

    def work(self, tokens, index=0):
        '''self.work(list, [int]) -> int'''
        while True:
            if index >= len(tokens):
                break
            current = tokens[index]

            # track open braces and close braces
            if current.value == '{':
                self.counter += 1
                self.tokens.append(current)
            elif current.value == '}':
                self.counter -= 1
                self.tokens.append(current)
                if self.counter == 0:
                    break
```

Figure 3.12: Colored Indented Blocks

3.3 Coloring Cell Block

In information visualization spreadsheets, cells may have abstract data sets, selection criteria, viewing specifications and other information required to customize specific views, have been developed to allow end users access to rich visualizations of data [7]. An idea of containing information about surrounding cells can boost the visualization if coordinated carefully. With each blocks of cell represent an indentation, a programmer still need to count the number of blocks to check when two or more functions or statements are apart from each other but expected to be in same indentation. Using a built-in feature in Microsoft Excel 2005/2007, cell blocks can be colored when it meets given conditions. When block of cell is colored with pre-defined color, it will save time counting blocks of cells in front of the code.

In Figure 3.12, each *n*-th level of indented blocks is colored to show the location of the indented block in a source code. In this case, the yellow represents the first indent block, green the second, orange the third, purple the fourth, red the fifth and blue which represents it is on the sixth indented block. If empty line was inserted to a source code, none of the blocks were colored to avoid confusion. Coloring cell by condition statements are used to on Excel feature which allows user to fill the cell with data additionally, uses its data to give information for visualization [8].

4. CONCLUSION

Further study of this subject may be to add options work with spreadsheets with different types that are widely used, such as OpenOffice.org Calc, Apple Numbers, etc. Using spreadsheet features to analyze the source code, for example, count the number of classes, functions, variables, and lines of code to compute the complexity of the program. In Microsoft Excel VBA(Visual Basic Application) is provided when OpenOffice has StarOffice Basic in CALC(spreadsheets) to allow complex calculation using programming language based on the data from the spreadsheet [9]. Grouping and Outling in Excel as well as hiding cells features will provide grouping classes or structure when editing python source code which is in most object oriented language source code editor. This will allow programmers to look at the source code with abstract information where it only shows classes, functions and structures name.

When there are many programmers using numbers of different types of editor to work on one program, it is hard to maintain a single style of way of writing code. Thus, an application is needed to recognize different styles and synthesize them for ease of collaboration in work among programmers with different background. Such application should display the problem with more clear and in meaningful matter.

5. REFERENCES

[1] Byron S. Gottfried, **"**Spreadsheet Tools for Engineers Using Excel", McGraw-Hill, 2009

[2] Mark Lutz, "programming Python", O'Reilly Media, 2011

[3] "BeginnersGuide Overview", Python Official Site. Python Software Foundation, n.d. Web. 10

[4] Rossum, Warsaw, "Style Guide for Python Code", www.python.org/psf/, Python Software Foundation

[5] Martin Erwig, Robin Abraham, Steve Kollmansberger, Irene Cooperstein. “Gencel: a program generator for correct spreadsheets”, Journal of Functional Programming, Cambridge University Press, Vol. 16, Issue 3, (2006): 293-325.

[6] Ed Huai-hsin Chi, John Riedl, Phillip Barry, Joseph Konstan, Principles for Information Visualization Spreadsheets, Vol. 18, no.4 IEEE. (1998): 30-38.

[7] Robin Abraham, Margaret Burnett, Martin Erwig. “Spreadsheet Programming”, Encyclopedia of Computer Science and Engineering, (2009): 2804-2810.

[8] “Microsoft Conditional Formatting: Adding Customized Rules to Excel 2007”, Microsoft Developer Network (MSDN), Microsoft Corporation

[9] Solveig Hauglan, “StarOffice 6.0 Office Suite Companion”, Prentice Hall, 2002

INSTRUCTIONS TO CONTRIBUTORS

The International Journal of Software Engineering (IJSE) provides a forum for software engineering research that publishes empirical results relevant to both researchers and practitioners. IJSE encourage researchers, practitioners, and developers to submit research papers reporting original research results, technology trend surveys reviewing an area of research in software engineering and knowledge engineering, survey articles surveying a broad area in software engineering and knowledge engineering, tool reviews and book reviews. The general topics covered by IJSE usually involve the study on collection and analysis of data and experience that can be used to characterize, evaluate and reveal relationships between software development deliverables, practices, and technologies. IJSE is a refereed journal that promotes the publication of industry-relevant research, to address the significant gap between research and practice.

The initial efforts helped to shape the editorial policy and to sharpen the focus of the journal. Started with Volume 4, 2013, IJSE appears with more focused issues. Besides normal publications, IJSE intend to organized special issues on more focused topics. Each special issue will have a designated editor (editors) – either member of the editorial board or another recognized specialist in the respective field.

We are open to contributions, proposals for any topic as well as for editors and reviewers. We understand that it is through the effort of volunteers that CSC Journals continues to grow and flourish.

IJSE LIST OF TOPICS

The realm of International Journal of Software Engineering (IJSE) extends, but not limited, to the following:

- Ambiguity in Software Development
- Application of Object-Oriented Technology to Engin
- Architecting an OO System for Size Clarity Reuse E
- Composition and Extension
- Computer-Based Engineering Techniques
- Data Modeling Techniques
- History of Software Engineering
- IDEF
- Impact of CASE on Software Development Life Cycle
- Intellectual Property
- Iterative Model
- Knowledge Engineering Methods and Practices
- Licensing
- Modeling Languages
- Object-Oriented Systems
- Project Management
- Quality Management
- Rational Unified Processing
- SDLC
- Software Components
- Software Deployment
- Software Design and applications in Various Domain
-
-
- Software Engineering Demographics
- Software Engineering Economics
- Software Engineering Methods and Practices
- Software Engineering Professionalism
- Software Ergonomics
- Software Maintenance and Evaluation
- Structured Analysis
- Structuring (Large) OO Systems
- Systems Engineering
- Test Driven Development
- UML
-

CALL FOR PAPERS

Volume: 4 - **Issue: 2**

i. Paper Submission: November 30, 2013 **ii. Author Notification:** December 25, 2013

iii. Issue Publication: December 2013

CONTACT INFORMATION

Computer Science Journals Sdn BhD
B-5-8 Plaza Mont Kiara, Mont Kiara
50480, Kuala Lumpur, MALAYSIA

Phone: 006 03 6207 1607
006 03 2782 6991

Fax: 006 03 6207 1697

Email: cscpress@cscjournals.org

COMPUTER SCIENCE JOURNALS SDN BHD
M-3-19, PLAZA DAMAS
SRI HARTAMAS
50480, KUALA LUMPUR
MALAYSIA

PHONE: 006 03 6207 1607
006 03 2782 6991

FAX: 006 03 6207 1697
EMAIL: cscpress@cscjournals.org