Automating Measurement for Software Process Models using Attribute Grammar Rules

Rodziah Atan
Information Systems Department
Faculty of Computer Science and Information Technology
University Putra of Malaysia
Serdang 43400, Selangor, Malaysia
rodziah@fsktm.upm.edu.my

Abdul Azim Abd. Ghani
Information Systems Department
Faculty of Computer Science and Information Technology
University Putra of Malaysia
Serdang 43400, Selangor, Malaysia
azim@fsktm.upm.edu.my

Mohd. Hasan Selamat
Information Systems Department
Faculty of Computer Science and Information Technology
University Putra of Malaysia
Serdang 43400, Selangor, Malaysia
hasan@fsktm.upm.edu.my

Ramlan Mahmod
Multimedia Department
Faculty of Computer Science and Information Technology
University Putra of Malaysia
Serdang 43400, Selangor, Malaysia
ramlan@fsktm.upm.edu.my

Abstract

The modelling concept is well accepted in software engineering discipline. Some software models are built either to control the development stages, to measure program quality or to serve as a medium that gives better understanding of the actual software systems. Software process modelling nowadays has reached a level that allow software designs to be transformed into programming languages, such as architecture design language and unified modelling language. This paper described the adaptation of attribute grammar approach in measuring software process model. A tool, called Software Process Measurement Application was developed to enable the measurement accordingly to specified attribute grammar rules. A context-free grammar to read the process model is depicted from IDEF3 standard, and rules were attached to enable the measurement metrics calculation. The measurement metric values collected were used to aid in determining the decomposing and structuring of processes for the proposed software systems.

Keywords: Software process modelling, Process measurement, Attribute grammar rules.

1. INTRODUCTION

Developing reliable software within time scheduled and cost estimated is a difficult task for many software development companies. Any flaws or late delivery of a system means a great deal for many individuals involved. It is indeed vital to produce reliable software right on schedule to avoid inconveniences for the developers, vendors and users. The software community places great hope

on software modelling notations and techniques to ease various software development challenges. One of the challenges is the requirement to creatively analyse and design problem-solving technique with a highly coordinated development team within a complex environment.

Software process modelling (SPM) is one of the techniques used to creatively define and analyse significant aspects, which can be adapt into convoluted application development and also can be used to structure a strategic co-ordination for the development team. The intellectual tool set available for software developers has steadily been enriched with more powerful and comprehensive models. There have been many approaches introduced to this particular field of software engineering. It started from the basic structure of software designing model and evolved throughout the time.

Software process modelling nowadays has reached a level that allow software designs to be transformed into programming languages, such as architecture design language (ADL), and unified modelling language (UML). These kinds of process modelling languages (PMLs) proved that people in software development team can execute their designs. There are many more existing software process notations and enactions that give much more choices of method for software developers to improve their process models. Above all the benefits offered by these known techniques, one factor differentiates their efficiency, which is measurement.

This paper will discuss on the approach of combining modelling standard in business process environment, software process modelling measurement and attribute grammar approach for an automatic software process metric measurements. The end result of the system will be a collection of measurement attributes that prescribe the process model designs size. The objective of this study is mainly to enhance the process modelling measurement effort in software engineering field in terms of predicting the design size, automatically.

2. SOFTWARE PROCESS MODELS

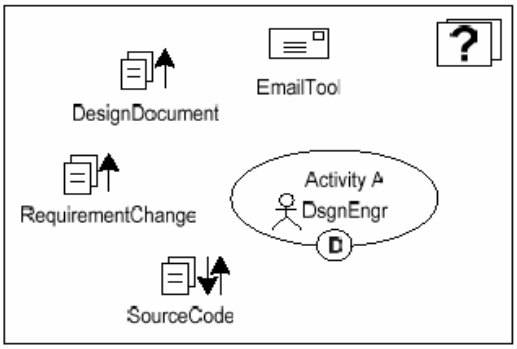
A software process models is an abstraction of the framework of process architecture within which project-specific software processes are defined [1]. It formalizes the structure, standards and other related process elements in a form of architectural standard that can be use as a framework of software process definition. The need for a standard process framework is important for compelling reasons such as; to permit training, management, review and tool support. It also useful to contribute to overall process improvement in the organization and it provide a structured basis for measurement.

Adding measurement into process modelling is another interesting research area that can be expanded abroad. Software measurement also covers a big portion in software engineering. Each of these measures has its very own class and schemes in accordance to its creator. One of the widely accepted classification schemes is from Fenton et al. [2]. They classify software measures in the classes of resources, process and product measures. The process and product measures are used to measure attributes of the documentation, code, characteristics of the activities, method, practices and transformation employed in developing the products. Another important measure is the one connected to programs, flow graphs or models, which is called the intra-modular software measures. This kind of measurement will be the main concern and consumed heavily throughout this particular study.

The means of interactively browse and symbolically execute process models can be a great help to software model designers. As an example, the precedence structure of sub-tasks or steps specified in the modelled process instance can be executed and lists of measurement metrics can be produced accordingly. Agents and tasks can then use or consume simulated time, budgeted funds and other resources along the way [3].

Virtual Reality Process Modeling Language (VRPML), for instance, is a visual PML that has been developed to include support for the integration of a virtual environment and dynamic creation and assignment of tasks and resources at the PML *enaction* level. The main objective of VRPML development is to be the research vehicle to address a research hypothesis that a PML, which exploits a virtual environment is useful to support software processes for distributed software engineering teams [4].

The VRPML exploits virtual environment at PML enactment level, which allows work context for a particular activity to be defined and later be opened as a workspace in a virtual environment [5]. The said activity will later be enabled using the *task-centred mapping* whereby each activity in a software process corresponds to a room in a virtual environment [6]. Figure 1 shows an example workspace in VRPML system.



ActivityName = Activity A, 2,
ActivityType = General Purpose,
Role = DsgnEngr
AssignedEngineer = Unspecified,
Artefact = Design Document, Path/Url for Modified Design, Read, Path/Url for tool,
Artefact = Requirement Change, Path/Url for Req. Change, Read, Path/Url for tool,
Artefact = Source Code, Path/Url for Source Code, Read/Write, Path/Url for tool,
Tool = Email Program, Email, Path/Url for tool,
Transition = D, Transition Done, Non-Decomposable, 5,
Description = Put the description of the activity here.

FIGURE 1: Example Workspace

A role specific process model (i.e. view) might be developed to formalise process models, which leads to different views of the processes. Because the roles collaborate, some information is common in views of different roles. Thus the software process models related to several roles should be integrated in order to allow for better coordination on basis of a consistent and less redundant software process models. Such an explicit representation of processes performed by multiple roles is called a *comprehensive software process model*. Comprehensive software process model can be used to represent important processes of a software development project. In this case, it serves as a basis of a central information system to guide, coordinate, and support the different roles.

Developing software systems is not an easy task. Many software systems face the risks of having flaws and malfunctions. Errors found during delivering the software system is highly potential been caused by the failure while coding the system, or it should be happening while designing the product. Repairing the 'completed' software system costs a lot. The best opportunity for short-term software cost reduction is to eliminate rework or fixing defects, which is more than 33 percent of developing new software systems [7].

The problem of reworking a software system can be avoided by tackling the problem far before the system is developed. How is it possible? Some would answer by strictly outlined the system requirements, or choosing the programming approach that flawless, or employ a highly competent programmers. Another question will arise, is the approach really going to ensure that the system is error free? The second question should be harder to answer than the first one. Software process modelling and process definition is not a new topic of interest in software engineering community. The said quality and productivity of software often can be improved by a well defined and managed processes, together with estimated and measured results of designed processes. Software process modelling and definition offered many benefits to the practitioners. It supported many objectives such as facilitating human understanding and communications, support process management and to provide automated execution support.

The prototype tool described in this paper use a context-free grammar to read the process model, which was adapted from part of Backus-Naur Form (BNF) of the process definition standard used – the IDEF3 standard. The proposed prototype is able to count process models' measurement metrics, which can be exploit to measure physical decomposition and structuring strategy of software systems' designs.

3. PROCESS MODELLING TECHNIQUE

The prototype tool that was created, called the Software Process Model Measurement Application (SPMA) used a modelling technique which was adapted from Integrated Definition for Process Description Capture (IDEF3) [4] standard approach. Integrated Definition (IDEF) is a set of standardized methods for structuring and refining functional overview of an environment [8]. Starting from IDEF0 up to IDEF14, all these methods are highly consumed by many organizations and companies intending to upgrade the functional flow of their working environments. The specific IDEF3 or the Integrated Definition method for Process Description Capture can be used independently or combined with other family members' methods for documentation, analysis and improvement. IDEF3 is a description of the real world in a form of model structure.

Features and functions defined in this standard were highly employed by business process engineers in order to enhance the capability of their business process workflow settings. IDEF3 is divided into two parts of representing the knowledge acquisition of a process, namely process-centred and object-centred strategies. These two main categories of IDEF3 are for the flexibility of the users to model their environments in which one approach they know best. This research used the process-centred strategy to solve its complexity. The reason to choose IDEF3 process-centred strategy for process modelling is based on its organized way on modelling processes with temporal, causal and logical relation within a scenario of a modelled environment. Figure 2 shows the framework for SPMA model.

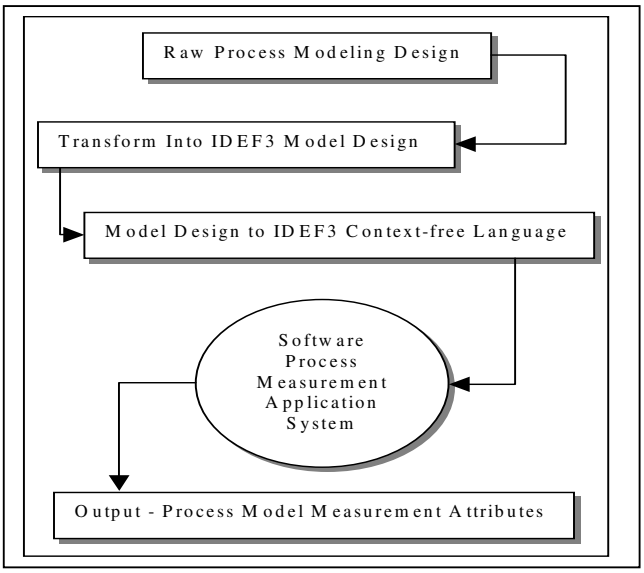


FIGURE 2: SPMA model framework

Although there exist many process modeling measurement applications, they usually have their very own measuring elements acting as additional features for their knowledge procurement for particular cases that they handled. In conjunction to this, SPMA fashioned its very own technique that collaborate business process modeling into software models development and process measurement. The software flow design which is created using IDEF3 method is converted into context free language that reads and interpret the whole process model design prior to analysis and measurement summary.

Attribute grammar element is also essential to SPMA model. It works as an agent that follow the flow of particular measurement metrics that has been assigned to the processes. The analysis of the attribute flow is then summarized and output a list of measurement attributes related to the software

process design. Some of the attributes examined are such as the process depth level, number of related sub-processes and the type of the design which basically horizontal or vertical.

3.1 SPMA Environment

As depicted from Figure 2, the process flow diagram created in IDEF3 structure should then be converted into IDEF3 language. The language consists of statements describing the declarations of sub processes, single processes, functional and junction statements and some other attributes such as the identifiers and the information flows either getting in the process or out from the processes, accordingly to IDEF3 structural design. Figure 3 shows an example of IDEF3 process-centred process schematic view of the scenario for *material purchase* process.

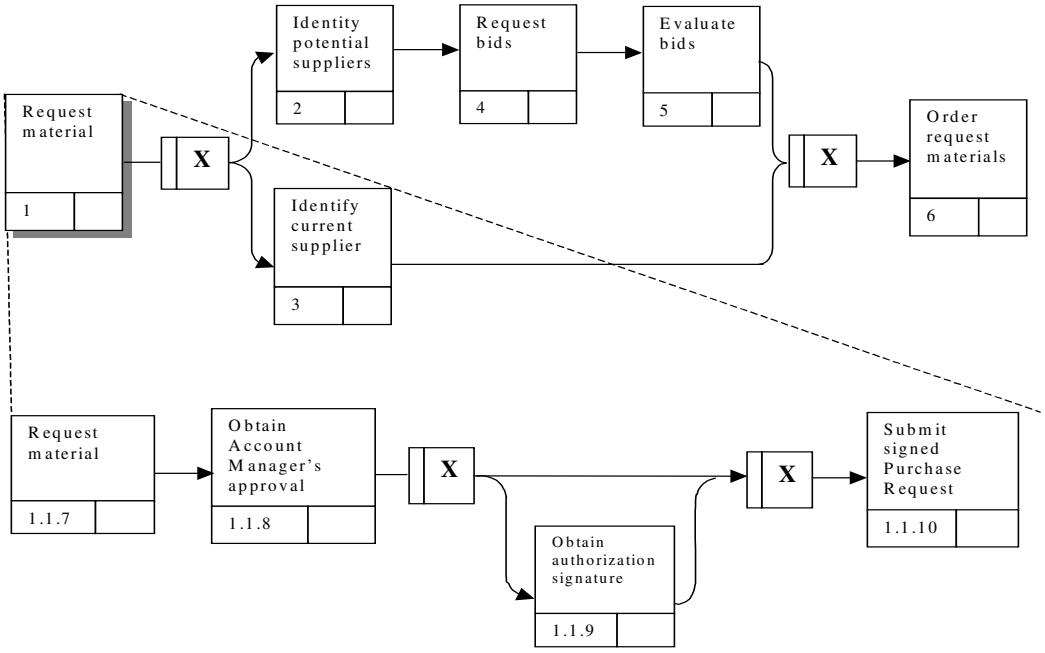


FIGURE 3: IDEF3 process model scenario

The idea of integrating software process modelling with business process modelling diagramming technique is a niche to this study. The stated design as shown in Figure 3 alone cannot be executed to produce lines of measurement attributes unless it is converted into a form that can be read automatically to produce specific metrics' calculation. This is the reason why the design has to be converted into context–free grammar form as shown in Figure 4, called the IDEF3-SPMA language.

```
<spmadl> : <dll> | error '\n'
<dll> : PROCESS IDENT';' <subprocesses> END
<subprocesses> : <subprocess_spec>
                    | <subprocesses> <subprocess_spec>
<subprocess_spec> : PROC IDENT io_data';' <dl>
                      END_PROC
<dl> : <sub_proc> | <bool_proct> | <sing_proc>
       | <dl><sub_proc> | <dl><bool_proc> | <dl><sing_proc>
<sub_proc> : IDENT <io_data> ASSIGN CALL '{'IDENT'}'';'
                  | IDENT <io_data> ASSIGN SUB '{'IDENT'}'';'
<bool_proc> : <junction><io_data> '{'<subjunc>'}'';'
```

FIGURE 4: IDEF3-SPMA language

3.2 Software Process Measurement

There are many existing effort of researches to deal with software process modelling, but there is still a lacking of process model measurement. Some of the examples are like Bassili and Weiss

(1984) [9], whom consider the measurement process and its validation, but do not couple the measurement process with software process.

Pfleeger and McGowan (1990) [10], associated sets of measures with the levels of the CMM, but do not define nor use it. The study use attribute grammar (AG) approach to measure process models. AG was selected because of its specification and automatic construction of language-based editors. Attribute grammar also provides a formal yet intuitive notation for specifying a static semantics of programming languages and has been variously used for constructing compiler generator systems. This unique characteristic of AG benefited much for this research.

Each semantic rule associated with a production rule either defines a synthesized attribute of the syntactic construct named on the left-hand side (*lhs*) or to define an inherited attribute of a syntactic construct on the right-hand side (*rhs*) of the production. In order to describe the occurrences of synthesising or inheriting attribute, shown in Figure 5 is an example of attribute grammar description specification.

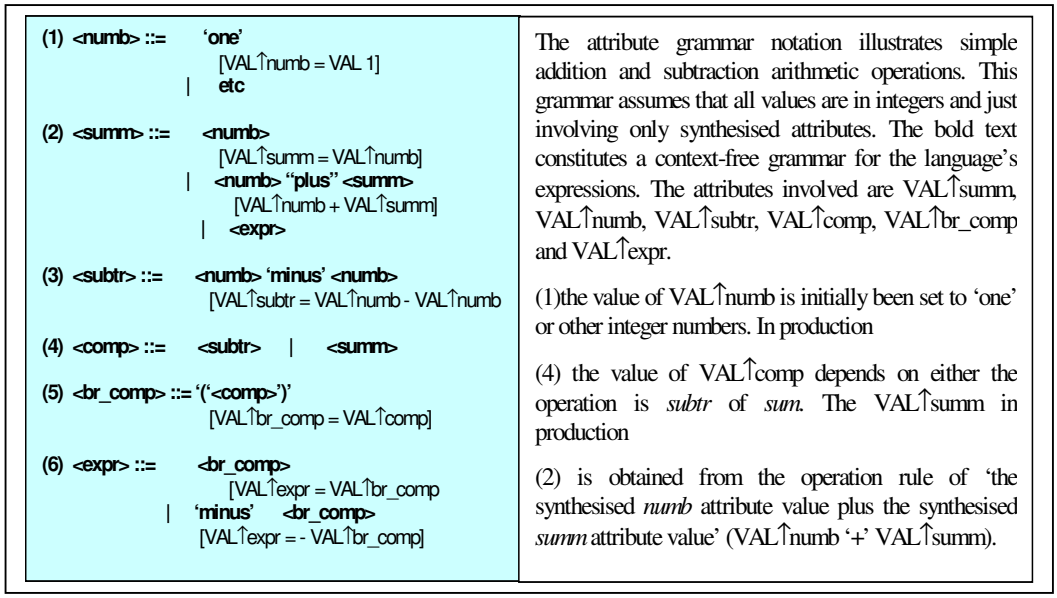


FIGURE 5: An attribute grammar description specification

3.3 IDEF3-SPMA Language

Formal definition of IDEF3-SPMA language is developed in order to give users a precise description of how to create acceptable design input as well as providing the instructors a reference model. There are two phases of language definition; the syntax definition and semantic description. A set of production rules is used to specify the syntax of IDEF3-SPMA language.

Each production specifies the manner in which a particular syntactic category (e.g. a clause) can be formed. Syntactic categories have names, which are used in productions and are distinguished from names and reserved words in the language. The syntactic categories can be mixed in productions with terminal symbols, which are actual symbols of the language itself. Thus, by following the productions until terminal symbols are reached, the set of legal programs can be derived. IDEF3-SPMA language is small and it has 14 described production rules, as follows;

1. <spmadl> ::= <dll>
2. <dll> ::= PROCESS <ident>';' <subprocesses> END
3. <subprocesses> ::= /*empty*/ | <subprocess_spec>
| <subprocess_spec> <subprocesses>
4. <subprocess_spec> ::= PROC <ident> <io_data>';' <dl> END_PROC
5. <dl> ::= <sub_proc> | <bool_proc> | <sing_proc> | <dl> <sub_proc>
| <dl> <bool_proc> | <dl> <sing_proc>
6. <sub_proc> ::= <ident> <io_data> ASSIGN CALL '{'<ident>'}'';'
| <ident> <io_data> ASSIGN SUB '{'<proc_list>'}'';'

7. <bool_proc> ::= <junction> <io_data> '{'<subjunc>'}'';'
8. <subjunc> ::= '['<proc_list>']' <io_data>',' CALL '{'<ident>'}'';'
9. <junction> ::= AND | OR | XOR
10. <sing_proc> ::= <ident> ASSIGN '{''}'';' | <ident> <io_data> ASSIGN '{''}'';'
11. <proc_list> ::= <ident> | <proc_list>',' <ident>
| <proc_list>',' <junction> '('<proc_list>')'',' <ident>
12. <io_data> ::= '('<var_inout>')'
13. <var_inout> ::= <ident> <iodata> | <var_inout>',' <ident> <iodata>
14. <iodata> ::= IN | OUT | INOUT

The defined IDEF3-SPMA language is able to gather and summarize information from the input process design. Source code metric definition using attribute grammar can be produced directly from the input source code. Design metric should have representation, which is able to abstractly show the process design at the early stage of the development. To this extend, the representation used is the design language specification.

4. IDEF3-SPMA INTERPRETER

The IDEF3 language is compiled using a C routine that was created using Flex and Bison tool. Flex and Bison are tools that can be used to help write compilers and interpreters or any program whose input has a well-defined structure [10]. Flex reads a specification file containing regular expressions for pattern matching. Diagram in Figure 6 shows the interpreter function of SPMA model during its execution.

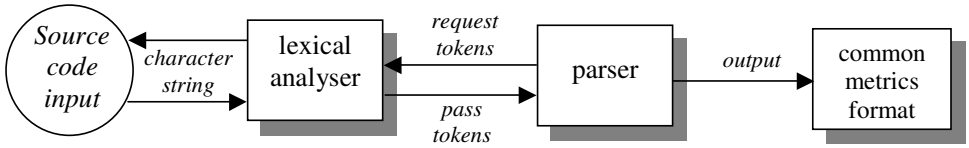


FIGURE 6: Interpreter function in SPMA model

Measurement within this study circles the area of process part of the system. The basic objective of the measurement is to measure the level of integration among the processes, the relationship between the stated unit of behaviors (UOBs) and the counts of hierarchy and UOBs used within a specified system or subsystems. AGs have a clear distinction between inherited and synthesized attributes, together with grammars that are quite visible [11].

5. SPMA EXECUTION

To execute SPMA tool, there are four stages of operation that should be followed sequentially, as described before. The first one is to get a problem or a requirement of a system, then the user must represent the process model in IDEF3 description before moving on to stage three, i.e. converting the representation into IDEF3-SPMA language accordingly to the defined syntax rules. After that, if there is no syntax error found in the input lines, SPMA tool executes and read the input to calculate its measurement metrics determined by the system. Figures 7 through 10 show the interfaces in SPMA model execution.

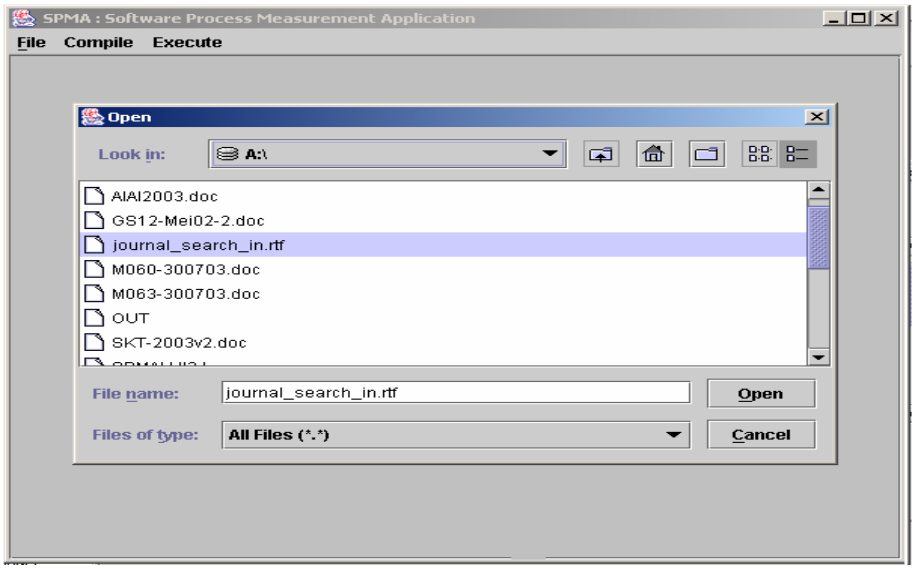


FIGURE 7: Choosing file function in SPMA model

Figure 7 show the case where user clicks on Open operation where a popup Open window will appear. User can search and select existing input file from the window.

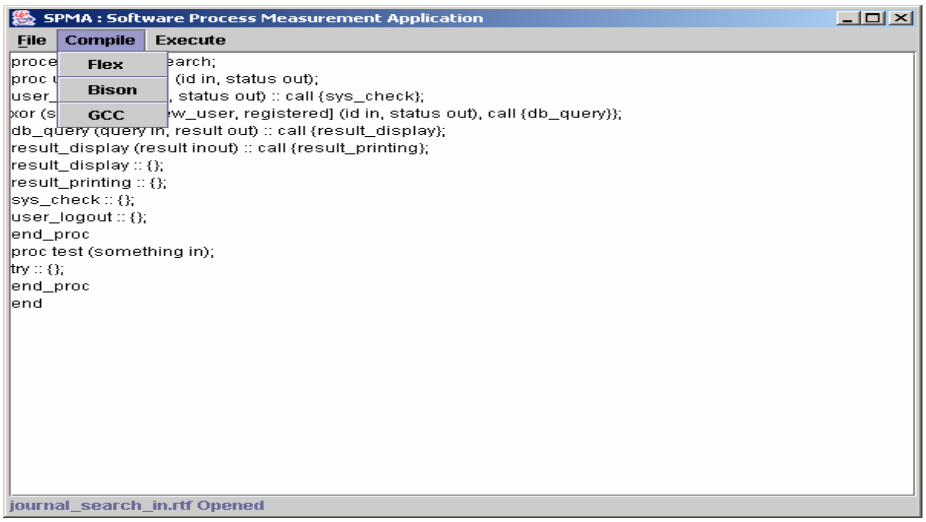


FIGURE 8: Compile function in SPMA model

Figure 8 depicted the second interface option in SPMA which is the Compile function. The operations are necessary each time before executing the system. This is to ensure that the parser and analyser used are the most current ones. The three compilation stages are the Flex, Bison and GCC.

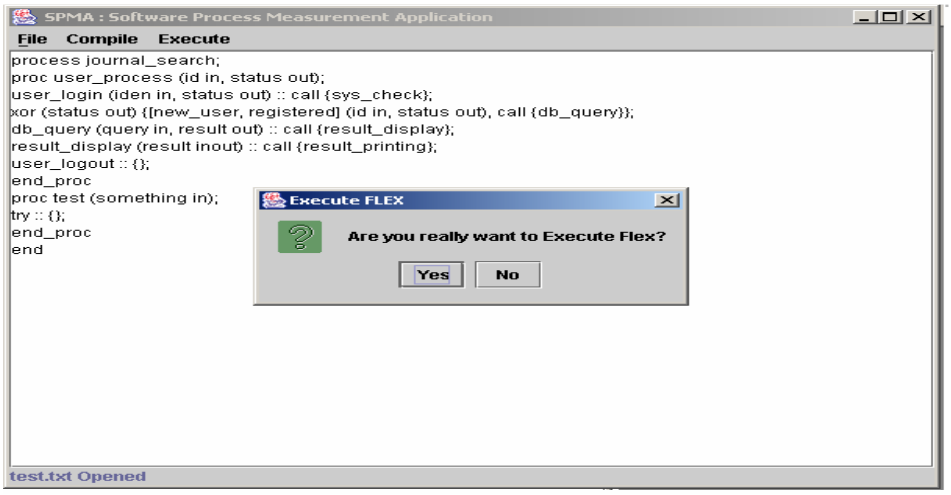


FIGURE 9: Execute function in SPMA model

Once a user tries to compile the analyser and parser, a popup message will appear (shown in Figure 9), to verify that the user is intentionally compiling the lexical analyser and the parser. Users just have to click on Yes or No to confirm on their action.

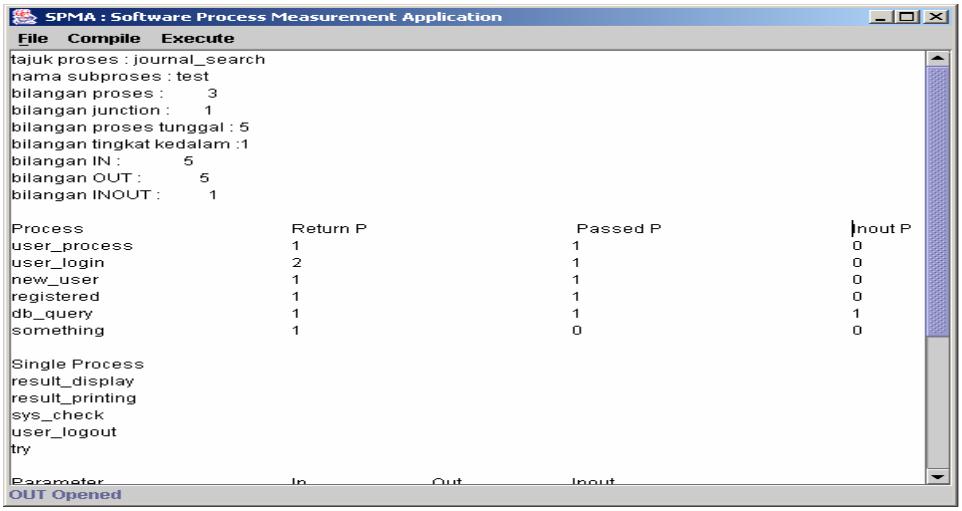


FIGURE 10: Output file generated by SPMA model

If the input design has error/s, a message error will appear telling there has been an error inside the input file and correction is needed. For an error free file, the users can open the output file (Figure 10) using the names they have defined before.

Other characteristic of this language-based metrics calculation tool is that it provides suggestions or advises for the users. The appropriate advice will be appended to the output file in terms of clarifying the meanings of the stated list of output. Advice in this context means to narrate the metric values and define what's "Good" with the produced metric values [12]. Based on survey to six software analysts and process design experts (expert here means more than 10 years of experience in software design and development), the process model design *size* produced by this study is divided into three categories. Corresponding advices are given to define the "Good" out of the *size* value produced. The advices for the three categories are defined as follows:

1. *Small*: This category is for designs with size ranged from 1 to 300 elements in process structure. The advice given to this range is "This design falls into small model design category. The design can be implemented by three (3) persons per team within four (4) months.

2. *Medium:* This category is for designs with size ranged from 301 to 1000 elements in process structure. The advice given to this range is "This design falls into medium model design category. The design can be implemented by three (3) persons per team within eight (8) months.
3. *Large:* This category is for designs with size ranged from 1000 and above elements in process structure. The advice given to this range is "This design falls into large model design category. The design can be implemented by three (3) persons per team within sixteen (16) months.

A set of questionnaire was used to gather expert view to validate the categories listed above.

6. CONCLUSION

The method is hoped to be able to facilitate process modelling environment with an executable measuring tool which can be used and ported anywhere. The executable software process model measurement tool will beneficial to software design analyst whom responsible to create a reliable, extensible and logical designs of software systems. The suitability between both business and software process models showed that there is not much difference between them as they were referring to the same set of process modelling objectives.

7. REFERENCES

1. W. S. Humphrey, *"The Software Engineering Process: Definition and Scope".* In Proceedings of the 4th International Conference on Software Engineering: Representing and Enacting the Software Process, ACM, 1988

2. N. E. Fenton and S. L. Pfleeger, *"Software Metrics: A Rigorous & Practical Approach",* 2nd Edition, PWS Publishing Company, pp. 21-79 (1997)

3. P. Mi, and W. Scacchi, "*A Knowledge –Based Environment for Modelling and Simulating Software Engineering Processes*", IEEE Transition Knowledge and Data Engineering, 2(3):283-294,1990

4. K. Z. Zamli, *"Supporting Software Processes for Distributed Software Engineering Teams"* PhD Thesis. School of Computing Science, The University of Newcastle Upon Tyne, 2003

5. K. Z. Zamli, N. A. M. Isa, *"Modeling and Enacting Software Processes: The How and Why Questions",* Technical Journal PPKEE, 10(1), 2004

6. J. C. Doppke, D. Heimbigner, and A. L. Wolf, *"Software process and Execution within Virtual Environments"*, ACM Transactions on Software Engineering and Methodology, 7(1) 1997

7. R. B. Grady, *"Successful Software Process Improvement"*, Prentice-Hall, (1997)

8. R. J. Mayer, et al., *"Information Integration for Concurrent Engineering (IICE): IDEF3 Process Description Capture Method Report"*, Interim Technical Report for Period April 1992 – September 1995, (1995)

9. R. A. Frost, "*Constructing Programs as Executable Attribute Grammars"*, The Computer Journal, 35(4) (1992)

10. T. Mason, and D .Bough, "*UNIX Programming Tools: lex & yacc",* O'Reilly & Association Inc., (1990)

11. A. A. A. Ghani, "*On Software Metrics Definition, Automated Data Collection and Related Standard Issues"*, PhD Thesis, University of Strathclyde, 1993

12. L. L. Westfall, *"Seven Steps to Designing a Software Metric"*, Benchmark QA: White Papers. http://www.benchmarkqa.com/index_resources_whitepapers.htm. Accessed on 13th January 2006