Behavior Engineering Methodology Enhancement to Support Code Generation

Emerson C. Simbolon and Eko K. Budiardjo

***Abstract*—Behavior Engineering (BE) is a component and behavioral based system methodology. BE uses Behavior Modeling Language (BML) to models a system. Formal syntax in BML supports automated code generation to the built system. However, no tool exists yet to support it. This paper propose a method in which makes it reliable to support code generation. Rational Unified Process (RUP) terminology such as Workflow, Worker, Activity and Artifacts are used to familiarize the method to people who already familiar to RUP. The suported research target is to apply the BE, so a tool is produced as well to apply the method.**

***Index Terms*—Behavior engineering, code generation, software engineering**

I. Introduction

Software development reliability has been an important issue since the emerging of safety in software engineering. Pioneers such as Dijkstra [1], Hoare [2], and Milner [3] have raised more concerns and suggested several important ideas and methodologies to build system with possibility of minimum error. The idea of reliable software construction and verification is not a practical subject and targeted to build critical system. For example, B by Abrial is one of software methodology for reliable software development [4]. It has been used in many critical software developments such as Paris Transportation and Peugeot automobiles [5]. The process in B requires deep logical proof. The details of deriving specifications have made possible to generate the code automatically. The process starts with a formal specification through several logical refinements which yield a reliable generated source code. The formal specification and logical refinements are expensive, but paid off by the result of having a reliable generated source code.

In 2003, Dromey proposed another mechanism to create a dependable system which called Behavior Engineering (BE) [6]. Based on BE, software requirements are analyzed and designed to form an Integrated Behavior Tree (IBT or only BT). BT has a formal semantic, so code generation is possible [7].

BT diagram provides a possibility to semi-automatic check of requirements' consistency, completeness, and aliveness [8]. Moreover, if the check process produces some counter examples, it is also possible to trace the counter example visually from the original diagram. Those features could lead to another new methodology of software engineering, which focuses on reliability but maintain simplicity and easiness for software engineers in building critical system.

In this paper, we apply an enhancement to BE to support an executable and dependable code. It uses ABS model as middle level representation, while BT as the higher level and Java at the implementation level. ABS is a semi-formal language to model system specification abstractly and yet still executable [9]-[11] Some features of ABS are already lined up with BT features. For example, ABS supports parallel system interaction just like BT. ABS ensures parallel execution aliveness to avoid deadlock and starvation natively.

The rests of this paper are presented as follow: Section II describes BE process and how it models a system. Section III describes environment that is used to conduct the experiment. Section IV proposes enhancement to the BE including the example of enhancement result. Section V evaluates the result of the research and suggestion to future works. Section VI concludes the research. Section VII provides future works that will help enhance the current result.

II. Behavior Engineering

BE is an integrated discipline that supports a large scale system development based on components and its behaviors that exists in that system. BE uses Behavior Modeling Language (BML) which consists of: Behavior Tree (BT), Composition Tree (CT), and Structure Tree (ST). Those models respectively describe behavioral runtime, system composition, and hierarchical structure of the system. This paper concentrates on BT.

A. Behavior Engineering Process

The whole BE process described in Fig. 1. First step of BE is to formalize the requirements. Dromey described the formalization step as to remove redundant, inconsistent information, also to split the complex requirements into smaller and focused requirement [6]. The next step is to model the requirement using BT. There is one Requirement BT (BT) that corresponded to each requirement. After each RBT is valid with respect to its requirements in natural language, they are integrated into an IBT. The integration processes are explained by Winter *et al*. [12]. IBT models the whole system and is required for further development steps, i.e. model checking, simulation, and code generation.

B. Behavior Tree

BT is a tree-like structure that consists of Behavior Nodes (BN). BT models the behavior of the system through interaction of each component and its behavior. Semantic of BT is derived from process algebra so it is assumed to be

Manuscript received July 19, 2013; revised September 22, 2013 .

The authors are with the Faculty of Computer Science, University of Indonesia, Indonesia (e-mail: emerson.chan@ui.ac.id, eko@cs.ui.ac.id).

safe and sound when we operate two nodes or more to form a BT [14]. Node notation in BT is shown in Fig. 2.

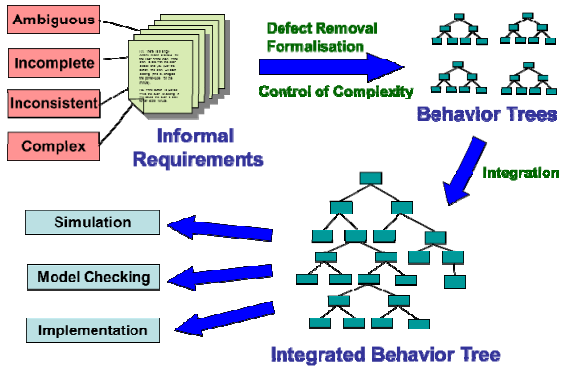


Fig. 1. Behavior engineering process[13]

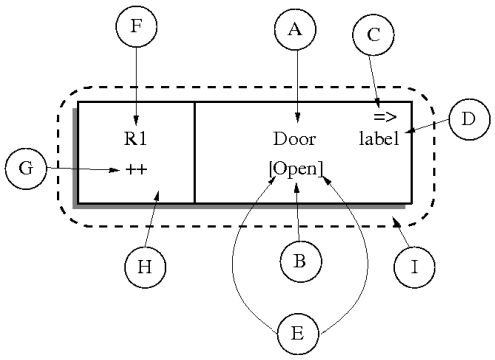


Fig. 2. Node notation in BT[15]

Legend:

A: component, B: behavior, C: operator, D: identifier label,E: behavior type,F: traceability link (with respect to requirement number),G: traceability status (with respect to requirement change), H: sign, I: node.

Information that required in code generation is: A, B, C, E, and I. Component information is needed to generate the object code. Behavior name and its type information is generated into the object's method. Operator information is used to control the execution process. The node information is used as the execution flow.

C. Model Check and Code Generation

Research about model check in BT is based on the need of model correctness before the development phase [16] The model check is applied to a well-defined model specification such asSymbolic Analysis Laboratory (SAL). SAL is a model language that is used to model concurrent system and already support model check, simulation, deadlock detection, and automated testing generation. Translation of BT model to SALis already conducted by Grunske *et al.*, [8]. Another verification method is also conducted by transform the BT model into Alloy [17].

BT transformation into SAL defines translation rule in variable handling, inter node translation, and behavior type execution. These schemes also can be used to transform BT to another programming language. In 2011, Emerson used this scheme to translate BT to an executable model language that called Abstract Behavioral Specification (ABS) [18]. Through the translation model to an executable model the BT model is also executable.

D. Behavior Runtime Environment

BRE is an execution environment of BE model. Each behavior type and operator is contributed to the runtime process. Fig. 3 explains the state of BN execution.

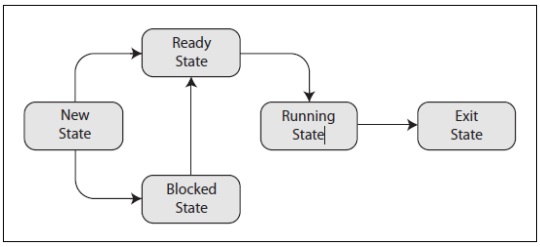


Fig. 3. State transition in BN execution [7]

The execution process of each behavior type, operator, and edge type are shown in Table I.

TABLE I: BEHAVIOR NODE EXECUTION PROCESS [7]

	Node Type	Path	Running/Blocked Behavior
Behavior Type	State Realisation	Ready	Call component function pointer and execute user-defined code.
	Selection	Ready	Evaluate Expression. Only add children to new if expression is true.
	Guard	Blocked	Evaluate Expression. Unblock if expression is true.
	Input	Blocked	Wait to receive message from eBRE (Internal) or a component or the environment (External)
	Output	Ready	Output message to eBRE (Internal) or environment (External)
Operator Edges	Reference	Ready	Add Destination to process control model with new state
	Reversion	Ready	Remove all descendants of Destination from process control model regardless of state. Add Destination node to process control model with new state.
	Branch-Kill	Ready	Remove all descendants of Destination from process control model regardless of state.
	Synchronisation	Blocked	Check if all destination nodes are in blocked state, if so unblock.
Parent Edge	Atomic Composition	-	Child nodes are added with ready state at start of FIFO queue, instead of with new state.
	Alternative Branching	-	If node is unblocked or becomes the running node then remove all siblings from process control model regardless of state.

E. Software Development Tool

Research in BE is collected and integrated to form a Software Development Tool. The tool consists of BT text editor (TextBE) [7], BT graphical editor (Graph BT), code generator [18], simulator [19], and model-checker [20].

III. EXPERIMENT

The experiment uses environment as shown in Table II. The experiment aims to show that the enhancement produce intended result to the case study. The experiment is not covering depth evaluation of model-checking environment because of the tool is not ready at the time this paper written. Through the experiment we show the proof that the enhancement works and also discovered many technical things that are needed to be added. Those technical things are including the practice of producing Graphical User Interface (GUI). We also include many preconfigured component to help generate a ready to use implementation code.

TABLE II: EXPERIMENT ENVIRONMENT

Criteria	Used Technology
IDE	Eclipse
Model Editor	GraphBT
Implementation language	Java
Model language	ABS
Targeted machine	PC
Operating system	Windows 8
GUI library	Eclipse SWT

IV. Enhancement to BE

The enhancement purpose is to suggest technical things that are not discussed previously to achieve code generation. The enhancement suggests terminologies that used in RUP such as worker, workflow, activity, and artifacts [21]. The purpose is to help many technician that already familiar with RUP to use BE.

The enhancement is not damaging the general concept of BE. It because the enhancement based on the experiment of applying the BE concept to a case study and make it more standardized and well defined guidance. The application of the methodology considers: work flow, worker, artifacts, and activity.

A. Methodology Practice

1) Requirement gathering

This phase is the initial inspection to the problem space of the system. Requirement is acquired through an interview to the Project Owner. The rest development processes depend on the clarity and the exhaustive information of the requirements.

- Artifact produced

Requirements (K1). Requirements are needed for analysis phase. Its contents are narration of system's problem space. Requirements should be amended if defects are found.

- Worker

Project Owner (W0). Project owner is the author who has the big plan of the whole project. She/he should confirm that the built software is fit to the requirement.

Project Manager (W1). Has task to bridge the gap between W0 and the development team. Has responsibility to deliver the project on schedule and to assign job to the development team.

Requirements collector (W2). Has task to collect and organize the requirements.

- Activity

Interview (A0). This activity is conducted by W2 to W0 to generate K1.

Organize the requirement (A1). After the requirements written to a document, W2 should be organized it with respect to specification of a requirement.Exampleof the organizationresult is based on the requirements described by Dromey [6].

R1 There is a single control button available for the use of the oven. If the oven door is closed and you push the button, the oven will start cooking (that is, energize the power-tube) for one minute.

R2 If the button is pushed while the oven is cooking, it will cause the oven to cook for an extra minute.

R3 Pushing the button when the door is open has no effect.

R4 Whenever the oven is cooking or the door is open, the light in the oven will be on.

R5 Opening the door stops the cooking.

R6 Closing the door turns off the light. This is the normal idle state, prior to cooking when the user has placed food in the oven.

R7 If the oven times out, the light and the power-tube are turned off and then a beeper emits a warning beep to indicate that the cooking has finished.

2) Analysis (F2)

Analysis is conducted through formulation of the requirements to BT diagram. The focus is to verify the requirement, remove ambiguity, complete information, and to preserve both consistency and clarity of the requirements.

- Artifact produced|

Component identification document (K2). This document consists of list of the components that identified from K1. Behaviors, attributes, and states are also included in this document.

RBT (K3). It is the RBT diagram that created with respect to each requirement.

IBT (K4). The integration result of the RBT.

- Worker

Analyst (W3). The role is to analyze the component, its behaviors, attributes, and states from K1 and put it to K2. W3 also creates RBT and the IBT.

- Activity

Component identi *fication* (A2). W3 identifies components and behaviors from K1. For example, the identification result from Oven system is shown in Table 3.

Build RBT (A3). After the components and behaviors are successfully identified, Analyst builds the RBT based on each requirement narration. The RBTsare created as much as the amount of requirements and identified with the same key of the corresponding requirement.

Build IBT (A4). IBT is an integration of all RBT. IBT represents the whole system behavior. Winter et. al. explains how to integrate RBT to IBT [12]. Integration result of Oven system is shown in Table III.

TABLE III: Component Analysis Result of Oven System

Id requirement	Identified components
R1	Button, Door, Power tube, Timer, Oven
R2	Button, Oven, Timer
R3	Button, Door
R4	Oven, Door, Lamp
R5	Door, Oven
R6	Door, Lamp
R7	Timer, Lamp, Power tube, Speaker, Oven

TABLE IV: Behavior Analysis Result of Oven System

Component	Behavior	Behavior type	Supported requirement
Button	Pushed	State Realization	R1, R2, R3
Door	Closed	Selection	R1
Door	Open	Selection	R3, R4
Door	Open	Guard	R5
Door	Closed	State Realization	R2, R3
Timer	1 minute count down	StateRealization	R1
Timer	Add extra minute	StateRealization	R2
Timer	Timeout	Guard	R3
Power tube	On	State Realization	R1
Power tube	Off	State Realization	R7
Speaker	Play sound	State Realization	R7

3) Design (F3)

This phase decides base architecture for the system. The system is implemented into a well-established language such as C or Java.

- Artifact produced

Architecture plan (K2). It contains configuration of the big plan of each component's implementation.

Component interface (K3). It represents each component look. Only component that visible to user is needed to be designed. The dimension of each component design should consider other aspects such as: relation to other component, aesthetics, and ergonomics.

System look (K4). It pictures the whole system look. The look is a result of the component interface placement.

- Worker

System designer (W4). The role of this worker is designing the system based on analyzed requirements.

- Activity

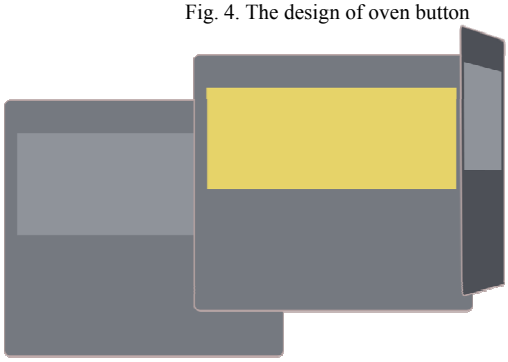
Design system architecture and implementation (A5). Designer has responsibility to plan the realization of every behavior. Designer also design the detail needed such as libraries that are needed for the implementation phase. The designed library needs to be fully tested to ensure the validity of its functionality.

The design result to oven system is as follow:

We will use "Button" preconfigured component to represent Button and Door component. Speaker needs to play sound, so we need a library to play a sound. We also need a library to implement the Timer capability.

Design component interface (A6). This activity gives clear picture to each component look. The look should consider what state that the component realizes to have the look. The example design result of the component Button and Door are shown in Fig. 4 and Fig. 5.

(a) Button when the door is opened (b) Button when the door is closed
(c) Button when the Oven is cooking
Fig. 4. The design of oven button



(a) Button when the Oven is “Off” (b) Button when the Oven is “On”
(c) Door in open condition
Fig. 5. The design of oven door

4) Verification (F4)

Verification is needed to ensure the correctness of the analysis result. With verification, we can make sure that a condition will or not will be met or whether the model already exhaustive. Verification conducted using model checker.

- Artifact produced

Verification result. It consists of the verification result and the test case to repair the model.

- Worker

Verifier (W5). Has task to verify the model.

- Activity

Formulate temporal logic (A7). Verifier identifies every condition that should not or should be met using temporal logic. Those formulas are identified through requirements and should be satisfied by the model before the system deployment.

Modelcheck IBT (A8). Verifier checks correctness of the model based on the temporal logic formula. Model check result is used to evaluate and repair the IBT or the requirements based on the test case that caused it to fail.

5) Implementation (F5)

In this phase, every behavior should be implemented to meet the expected behavior in its execution. For example, when Speaker realize ``play" behavior, it really plays a sound when it is executed. Differ with the conventional methodology, implementation in BE tries to focus implementation in each behavior and not the whole system code at once.

- Worker

Implementer (W6).Implementer has task to implement the system based on the design. Implementer can request a new library if there is no existing library support the needed functionality.

- Activity

Add technical detail (A9). Technical detail adding purpose is to add chunk of code that satisfies the expected behavior functionality. The code is more concrete than BE syntax and should be executable. Technical details that are added to Oven system behaviors are shown in Table IV.

Organizing layout (A10). To applies the designed layout of the system. The layout result of the system oven is shown in Fig. 6.

TABLE IV: TECHNICAL DETAIL OF OVEN SYSTEM BEHAVIORS

Component	Behavior	Behavior Type	Technical detail
Timer	1 minute countdown	StateRealization	timer_var! start();
Timer	Add extra minute	StateRealization	timer_var! addMinute(1);
Timer	Timeout	Guard	timeout_var = timer_var! timeout()
Speaker	Play sound	StateRealization	sound_var! play(“resource/ beep.wav");

6) Testing (F6)

Testing ensures that all system functionality run without defects. Testing is based on the scenario that identified through requirements. The result is consists of testing detail and the status of each scenario whether it is accepted or rejected and the reproduction step to reproduce the failure.

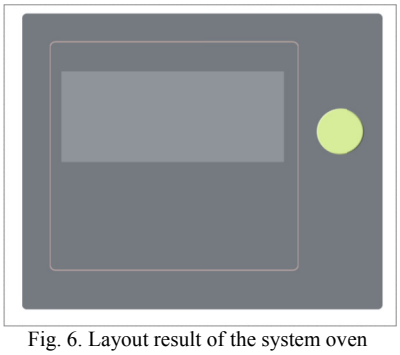


Fig. 6. Layout result of the system oven

- Artifact produced

Testing scenario (K6). Is a document in which each scenario should be accepted to mark the work as done. Information such as type of failure, reproduce steps, and who is responsible to the failure is noted in this document. Testing scenario of Oven system is shown in Table V.

TABLE V: OVEN SYSTEM TEST SCENARIO

Scenario	Expectation
Button is pushed when Door is open	Nothing happens
Button pushed when Door is closed and the Oven is idle	Start cooking for 1 minute
Button is pushed when the Oven is cooking	Adds 1 minute cooking time
Open the door when the Oven is cooking	The Door will be opened and the cooking ends
Cook for one minute	Cooking ends after waiting for 1 minute and the Speaker plays a "beep" sound

- Worker

Tester (W7). Has task to conduct the testing scenario and fill the required information regarding the test process.

- Activity

Testing (A11). Tester conducts the test to the generated code of an implemented IBT based on testing scenario. Tester takes note about the failure and reproducing steps.

7) Deployment (F7)

This is the last phase of a system development. After a system is being deployed, it is ready to be used by user.

- Artifact produced

Executable program (K7).It is the resulted program that generated from the implementation of BT.

Documentation (K8). This document contains technical detail of the development product such as description of each component and behaviors.

User Manual (K9). This document is used as a guidance to use the product.

Release notes (K10). This document is used to describe the system condition after development phase.

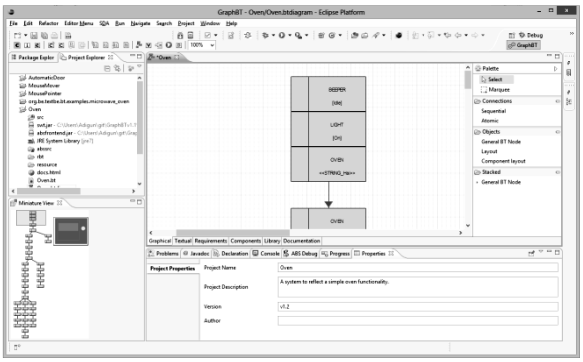


Fig. 7. GraphBT user interface

B. Tool Support

The whole development process utilized certain tools. The tools that are useful for development process are:

Pivotal Tracker (PT). PT is an Agile based requirement management. It helps communication between Project Owner in discussing requirement and problem space of the system. PT is useful in requirement gathering and to organize requirement.

GraphBT.Graph BT is an Eclipse plug-in that is used to apply the methodology described above. GraphBT is an integrated software development in BE, so it can be used to model-check, simulate, and generate code of a BT model. GraphBT is available as an open source project. GraphBT looks is shown in Fig. 7.

Git (or alike). This tool is used to track change. Git is integrated to Eclipse. Collaboration is achieved through Git

Adobe Photoshop (or alike). This tool is required to design component interface.

V. EVALUATION

A. BT result

The BT that produced by applying the guide (BT1) differs with BT that produced by the past (BT2). It is because BT1 is bounded by execution purpose. Another case also BT1 removes component User that operates the system and only concentrates on the system behavior for the execution purpose.

The change of the BT1 also makes the model-check process different. To conduct model check to BT1, there should be additional BTnode that has another property that yet not included in Fig. 2. That property is needed to distinguish whether a node should be included to the code generation or to model check only. In other words, that node will not be generated to avoid self-realization but the model checker always knows that the condition will be realized eventually.

B. Produced System

The produced system is not validated is open to contributor. The quality of the produced system is based on the quality of the development environment. For example, the combination of Java as the implementation language, ABS as the model language, and SWT library, a full operated desktop application is produced. A full critical system is able to be built as long as the environment is prepared well.

VI. CONCLUSION

BE enhancement is proposed, defined, and applied. The experiment result conclude that: BE supports a reliable system development that able to generate dependable code, quality of the produced system depends on the development environment, BE needs investment to mature the development process for commercial use. More advanced case study is needed to represent the capability of BE and to find another common practice in building system using BE.

ACKNOWLEDGMENTS

IMHERE for research grants. GraphBT contributors: Agung Pratama, Ardi, AtimasNurahmad, IkhsanulHabibie, and NurulQomariyah.

REFERENCES

[1] E. Dijkstra, "Letters to the editor: Go to statement considered harmful," *Communications of the ACM*, pp. 147-148 ,1968.
[2] T. Hoare, *Communicating Sequential Processes,* 1st ed. Prentice Hall International, 1985.

[3] M. Hennessy, R. Milner, "On Observing Nondeterminism and Concurrency," in *Proc. the 7th Colloquium on Automata, Languages and Programming*, London, 1980, pp.299-309 .
[4] R. Abrial, "The B Tool (Abstract)," in *Proc. the Europe Symposium on VDM - The Way Ahead*, 1988, 86-87.
[5] *Clearsy System Engineering*, Industrial Use of the B-Method, 2011.
[6] R. Dromey, "From requirements to design: Formalizing the key steps," *Software Engineering and Formal Methods*, pp.1-10 , 2003.
[7] T. Myers, "The foundations for a scaleable methodology for systems design," PhD Thesis, Griffith University , 2010.
[8] L. Grunske, P. Lindsay, and N. Yatapanage, " An automated failure mode and effect analysis based on high-level design specification with behavior trees," in *Integrated Formal Methods*, J. Romijn, G. Smith, J. Pol, eds. , 2005, pp. 129-149.
[9] D. Clarke, N. Diakov, R. Hahnle, E. Johnsen, I. Schaefer , J. Schafer, R. Schlatte, P. Wong, "Modeling Spatial and Temporal Variability with the HATS Abstract Behavioral Modeling Language," *SFM*, 2011.
[10] HATS Project: The ABS Language Specification. [Online]. Available: http://tools.hats-project.eu/download/absrefmanual.pdf
[11] The HATS Project: ABS: FLI. In: HATS Project Tool. [Online]. Available: Available at: http://tools.hats-project.eu/
[12] K. Winter, I. Hayes, and R. Colvin, "Integrating requirements: The behavior tree philosophy," *Software Engineering and Formal Methods*, Pisa, pp. 41-50 , 2010.
[13] *ARC Center for Complex System*, Building Dependable Systems, 2012.
[14] R. Colvin and I. Hayes, *Technical Report - A Semantics for Behavior Tree*, 2007
[15] *Behavior Tree Group*, Behavior Tree Notation, 2007.
[16] P. Lindsay, K. Winter, and N. Yatapanage, "Safety assessment using behavior tree and model checking," *Software Engineering and Formal Methods*, Pisa, Italy, pp.181-190, 2010.
[17] W. Itani and L. Logrippo, "Formal approaches to requirements engineering - From behavior trees to alloy," in *Proc. 2005 Canadian Conference on Electrical and Computer Engineering*, 2005, pp. 916-919 .
[18] E. Simbolon, "Translation of specification in behavior tree to an executable abstract behavioral specification," Bachelor Thesis, Universitas Indonesia, Depok , 2011.
[19] F. Dolot, "Rancangan dan implementasi bahasa simulasi untuk requirement engineering dalam bentuk animasi behavior tree pada textbe (textual editor for behavior engineering)," Skripsi, Universitas Indonesia, Depok ,2011
[20] N. Pratiwi, "Simulation of software requirement in BT using SAL," Skripsi, Universitas Indonesia, Depok , 2011.
[21] M. Priestley and M. Utt, "A unified process for software and documentation development," in *Proc. IEEE/ACM IPCC/SIGDOC,* 2000.

Emerson C. Simbolon was born in Jakarta on 1990. In 2013, he is graduated from master program in computer science – University of Indonesia. His research interest in Software Engineering is in particular of Behavior Engineering process and modeling. He is currently an engineer in a mobile software company in Jakarta.

Eko K. Budiardjo was born in Jakarta on 1959. He has been the faculty member of the faculty of computer science - University of Indonesia since 1985. Teaching, research, and practical services are aligned; give result in a full spectrum of academic achievement. Majoring in Software Engineering as professional track record, he has made some scientific contribution such as Software Requirement Specification (SRS) patterns representation method, ZEF Framework, and FrontCRM Framework. Graduated from Bandung Institute of Technology (ITB) in 1985, holds Master of Science in Computer Science from the University of New Brunswick – Canada in 1991, and awarded Philosophical Doctor in Computer Science from the University of Indonesia in 2007. Currently he is the Vice Chairman of ICT Technical Committee of The National Research Council (DRN), and Chairman of The Indonesian ICT Profession Society (IPKIN). He is reachable through eko@cs.ui.ac.id.