International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-8 Issue-2, July 2019

Tracing Software Development with UML—A Pragmatic Workflow

Nalinee Sophatsathit

Abstract*: This paper proposes a software development tracing technique as a means for source of requirement verification. It is a pragmatic process since the methods employed in this research are well-known and practiced by software developers. The proposed technique helps trace any missing unimplemented or erroneously implemented requirements to insure their completeness. The underlying principles are governed by the proposed apriori algorithm and precedence relation that link all the cross-reference action items in the UML diagrams. These links are then recorded in a checklist that serves as the reference to requirement items and action sign-off confirmation. Thus, the contributions of this research work are two folds. First, the proposed tracing technique is simple, pragmatic, and inexpensive to learn and implement. Second, it helps reduce human errors caused by manual process that are often performed by different parties. The result statistics show that many errors incurred in the development process can be identified and rectified accordingly. Future work should focus on automating the proposed technique to be a development support tool for the benefits of software practitioners and technology.

***Index Terms*: action item, apriori algorithm, precedence relation, checklist.**

I. INTRODUCTION

Software development paradigms have evolved considerably during the past decades. Many programming languages and tools available such as html, java script, ionic, angular.js, node.js, MongoDB, Sublime Text intelligent language editor, Yii framework, and so on, are becoming more and more sophisticated. A couple of apparent stumbling blocks in coping with these tools are learning to use them effectively and make them work together seamlessly. The former is achievable as we follow the manuals properly and frequently practice using them, while the latter seems to be difficult to achieve if those tools are developed by different software companies, on different frameworks and platforms. As technology advances, tool complexity grows. Many users inevitably have to keep pace with it. Similarly, the learners have to adjust themselves to accommodate such a steep learning curve. Case in point, to create a file on the UNIX operating system, a simple 'vi filename' at the command prompt (provided one uses *vi* text editor) will get the job done. On the contrary, for modern window-based operating systems, a series of mouse clicks, excluding any changes in all the default settings, must be done before one can start editing. At any rate, these stumbling blocks can be overcome by the first solution—follow the manuals. This research will focus on the second stumbling block as to how technology can handle such a problem with the help of a pragmatic workflow. The work considers software development as it is one of the important learning areas and also serves as a pragmatic means for modern software development technology. If we take a closer look into software technology, one of the common difficulties encountered in software development process is code tracing. From design, coding, testing, and installation, errors and glitches are prevalent (assuming the Waterfall Model). These errors are, in many cases, unidentified from where they came. And the fixes sometimes are ad hoc in nature. That means they do not comply with the original user's requirements, which often are inaccessible by the testers or field installation engineers. According to proper software engineering principles, software should be developed *right* in the first place. That is, correct requirements and specification, correct design based on the correct requirements, correct test results according to test specifications, and correct installation/operation according to requirements specification. But how can we ensure that these processes are done right? The obvious answer is using traceable document to verify. This is one aspect why new story-development in agile paradigm fills in the above shortcomings by completing all activities of the current story within the cycle. Anyhow, tracing to verify and validate (V&V) is still a tall order for software developers as a whole, regardless of the development paradigm being deployed.

The proposed technique exploits some existing technologies and innovative techniques to make them work together. The main emphasis is on applying simple and straightforward methods that are easy to learn. Thus, the contribution will be significant at a comprehensible level by implementers and inexpensive cost for software development. This paper is organized as follows. The next section will discuss some relevant literature, followed by the proposed methodology. Some problems and their solutions, causes or effects, and linkage to the sources that are uncovered by this research will be presented in the results section. A few important findings and future work will be discussed in the last section.

II. RELATED WORK

Software has been one of the instrumental tools for modern technology, especially in this information age.

Revised Manuscript Received on 30 July 2019.
***** Correspondence Author
Nalinee Sophatsathit*, Computer Science Program, Faculty of Science and Technology, Suan Sunandha Rajabhat University, Bangkok, Thailand.

Software development process has gone through many transformations, starting from single instruction execution, spaghetti bowl code, structured programming, object-oriented programming, even-driven programming, and cross-platform development which is known by XaaS paradigms [8][9]. The popularity of XaaS as noted by Y. Duan et al. [8], M. R. Joshi and B. V. Tikar [9] reported how software development process would fit on different platforms. Many modern programming languages are created based on this very notion such as C++, C#, Java, Python, etc. From the developers' standpoint, guidelines and procedures to write code on such platform can be performed straightforwardly with the help of the Unified Modeling Language (UML) standards to support the development process and communicate among members of the development team [3]. During the development process, the relationship of code and its precedence items could be traced procedurally along their requirements and design transformation. To set up the relation precedence, we employed UML, in conjunction with the Precedence Diagramming Method (PDM), to express the connectivity among related items [5]. Thus, analysis could be performed using apriori algorithm [1], [6], [7]. Some modifications in the algorithm were made to tailor the algorithm suitably for requirements and code connectivity traces, for instance, chronological ordering, refinement by precedence relation, etc [4]. In the meantime, some forms of verification had to be carried out to ensure that all the necessary changes were taken care of. A checklist was devised using guidelines provided by Lutz [2]. Consequently, the proposed technique could be commissioned with practical development environment technology.

III. METHODOLOGY

The proposed technique focuses primarily on using existing algorithms, methods, well-established and practiced techniques, namely, apriori algorithm, PDM, UML, and checklist. In this research, we will use a combination of apriori algorithm and precedence relation to construct cross reference association among various item sets. We focus on items (hereafter will be alternately referred to as action items) that involve interaction with users or system as they might incur errors, in particular, caused by the notorious human errors. By referring to the written documents, we can assure that the design and subsequent implementation will abide by the stated requirements and specification. In so doing, every succeeding item can be traced back to the sources such as Software Requirements Specification (SRS), design, and test documents that relate to the action items. The procedure can be described step-by-step as shown in the proposed algorithm below.

The procedure starts by selecting a designated story to be considered (step 1). The set of task items that make up the story along with their corresponding precedence relation is input (step 2) and built (step 3 and 4). Then fill out the story details such as duration to complete (how many man-hours it takes to complete the story), reference origin (where the story is originated typically from SRS document), and the person in charge of the story as shown in Table 1 (step 5). Create the UML diagrams to describe the process of how the story would be carried out. Explore if the activities in the story can be further broken down into finer-grained. If so, proceed immediately. This step 5 will successively create a reference from the point of preceding activities to the last activity of the story. Note that the first activity is frequently referenced by its successors according to the apriori association. Their precedence relation mandates the order of description details as the story unfolded.

(1) **Proposed algorithm 1**: //story S, precedence relation P
(2) Input: S, P
(3) S → $\{u_1, u_2, \ldots, u_n\}$ //build set of task items from story S
(4) P → $\{p_1, p_2, \ldots, p_k\}$ //build set of precedence relation
(5) foreach $u_i \in S$
    $D_{ui} \leftarrow t_j$ //duration of task t_j
    foreach $a_i \in P$ //activities
        if a_i = human interaction
        $a_i \leftarrow L_i$ //label it as an action item
        traverse P to find predecessor of a_i
        $L_i \leftarrow M$ //link to manual/document source
    endfor
endfor

In order to guard against errors, omissions, left-out items, a verification checklist is also introduced in the form of *checklist* to insure proper procedural precedence preservation. This checklist is generated from the label obtained above, where the action items to be checked are extracted from the reference (or link) field as shown in Table 1. The registration requirements are the first document in line, version a0.5, that has been approved on 02/15/2019. The revised version (i0.9) of input specification based on its predecessor ($Xref(L_i)$ = a0.5) was approved on 03/04/2019. Data design specification (d0.5) based on input specification document (i0.9) and form layout (u0.6) constructed from registration requirements and revised input specification (a0.5, i0.9) were *pending* for approval, and so on. This extra precaution will serve as a document trace for process completion. We will explain how this procedure works with the help of an illustrative case of software design example.

Table 1. Action item reference checklist.

Date	Version	Description	Author	$Xref(L_i)$	Action
02/15/2019	a0.5	Registration requirements (Use Case)	Tom	-	☒ pass ☐ fail ☐ n/a
03/04/2019	i0.9	Revised input specification	John	a0.5	☒ pass ☐ fail ☐ n/a
03/10/2019	d0.5	Data design specification	John	i0.9	☐ pass ☐ fail ☐ n/a
03/21/2019	u0.6	Form layout	John	a0.5, i0.9	☐ pass ☐ fail ☐ n/a

The following procedure describes the above algorithm by means of a user registration process to demonstrate the construction details. Fig. 1 illustrates the root item use case diagram. A user can either view the member list if he does not remember his membership information or add new information if he is a new user. Both actions include a manage data file function to handle the data repository. As design progresses, many artifacts must be created accordingly to accommodate the register process. Sample artifacts to be created are data repository (DB), application form (html), code to run the task (script for form, information display, verification, confirmation), and most important of this research, cross referencing for all related documents and diagrams. They furnish mapping sequences among these artifacts. Therefore, tracing sequence of development can be attained using apriori information and association from the respective UML diagrams.

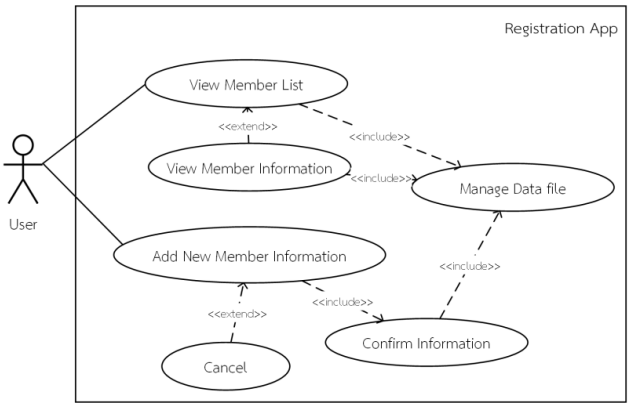


Fig. 1. User registration process.

Table 2. Activity detail.

Activity (task item)	Duration (man-day)	Reference	In charge by
View member list (a_1)	1	FR §2	John
view member info (a_3)	1	FR §2.1	John
manage data file (a_4)	2	FR §2.2	John
Add new member information (a_2)	3	FR §3	Tom
cancel (a_5)	1	FR §3.2	Tom
Confirm info (a_6)	1	FR §3.4	Tom

Remarks: FR – functional requirements, a_i denotes activity *i*

Table 2 shows the activities (task items) of the user registration process. The duration is the estimated completion time of each task, measured in man-day. Every activity can be referred to by the designated section or subsection of the Functional Requirements (FR) description. The last field shows the name of person in charge of the activity. Building a precedence relation according to FR field yields the activity referential relation a_1 through a_6 for the task items in Table 2. This is denoted by the dash lines as shown in Fig. 2.

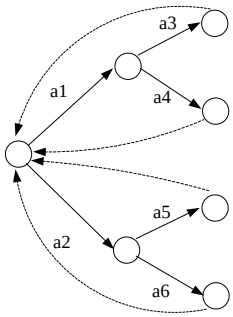


Fig. 2. Activity precedence relation.

Fig. 3 depicts the normal execution of task item a_2 in Table 1. The added tailing tags label the association of the artifacts in the cross-reference table to the pertinent documents. For example, the create_userinfo(+) task adds new user information to the save_DB(+). The process is recorded in the data repository for subsequent audition. Thus, save_DB(+) references create_userinfo(+) to insure all fields are created in the repository to hold the data from next input_userinfo(+) activity. In the meantime, check_passwd(?) is also applied as a security precaution. The edit_profile(#) activity is tagged to ensure that user information from the create_userinfo(+) activity can be edited accordingly. The update_DB($) will subsequently complete the task based on the same cross reference information.

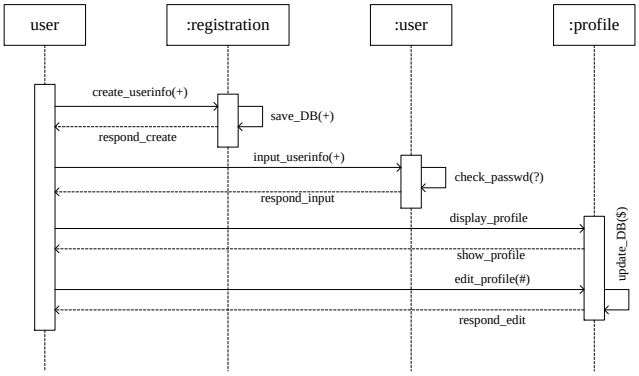


Fig. 3. Registration activity execution.

Fig. 4 depicts high level design of the add information process that takes the preceding registration process to work out all activity details till completion. The action items are tagged to refer their association relation to the preceding requirement items in SRS, i.e., a0.5 of the action item reference checklist. Execution commences at open app(%) that refers to the program to be invoked. As the information page is displayed, user selects proper function by clicking the corresponding button(+) to display the add information form. Now it is the user's responsibility to fill out all the pertinent information via input info(+) and submit button(#) at completion. From programmer's standpoint, these technical specifications, obtained from i0.9 of the action item reference checklist, will be used as the basis to design data file and repository template (DB),

denoted by the class diagram as shown in Fig. 5. Fig. 6 depicts the input formats and related specifications, particularly processing precedence relation, retrieved from the referenced document such as name (FN, LN), telephone number (+country code-local number), and so on.

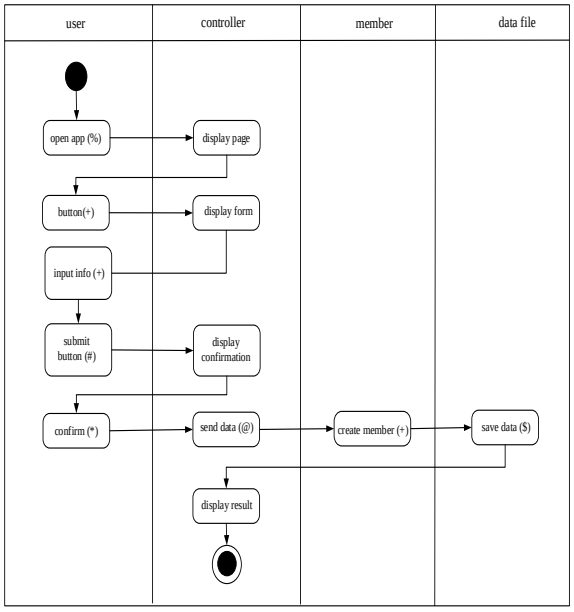


Fig. 4. Add information.

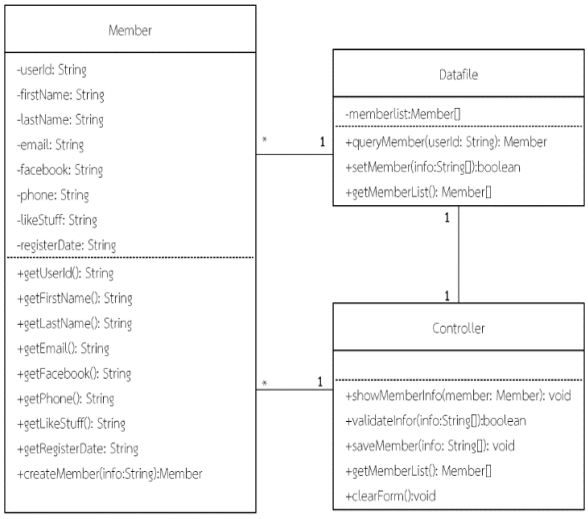


Fig. 5. Class diagram.

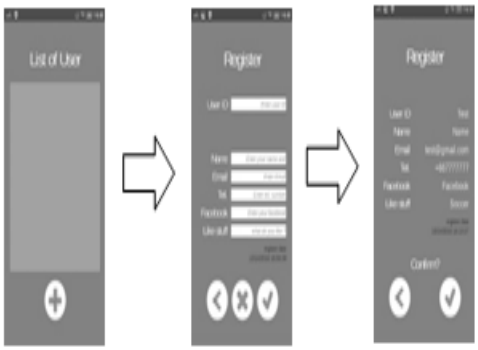
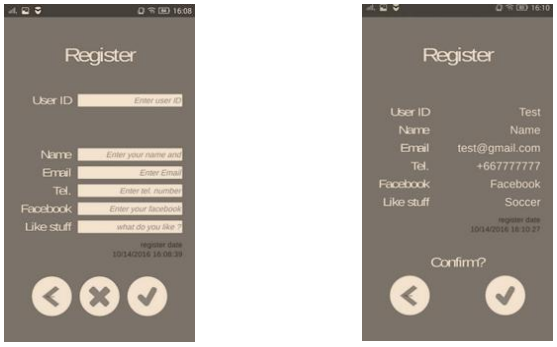


Fig. 6. Display pages.

main page register page confirmation page The final walkthroughs of registration cross-check yield another look of checklist verification procedure that serves as the confirmation activity (a_6 – confirm info). Fig. 7(a) shows the template of user registration with instructions displayed in the designated textboxes, while Fig. 7(b) depicts a filled form waiting for input confirmation.



(a) (b)

Fig. 7. Registration cross-check.

IV. RESULTS

From the beginning registration requirements document to the final registration app, the cross-reference detail that precipitated from several UML diagrams helped fine-tune the development process through the use of simple checklist. A number of common human errors were uncovered as shown in three consecutive summary tables. Table 3 demonstrates typical problems encountered, namely, incomplete username spec, data entry design difficulty, and erroneous diagrams. By virtue of cross-checking, the solutions to the above problems were derived with simple traces of document association. The table also contains addition links to elaborate on SRS and design association. Thus, many erroneous causes during the development process would not go undetected. Meanwhile, corrective solution could also be straightforwardly devised to handle the situations as shown in Table 4.

Table 5 collects the statistics on the above user registration apps. Many of those statistics were attributive to human errors, lack of support, and manual editing work. Fig. 8 depicts the overall cross-check results. Notice that misspelling and variable naming took the first and second causes of common errors that could have gone undetected during the development. The third spot—Poor writing, was something often overlooked by developer themselves and in worst case, project manager.

Table 3. Summary of problems, solutions, cause/effect, linkage

Problem	Solution	Cause/effect	Reported by	Link
Incomplete username spec	Missing char type definition	Errors in SRS	Tom	#1
Data entry design difficulty	Simplified SRS spec	User's permission	Tom	#2
Erroneous diagram	Revised workflow	Human errors	John	#3

Table 4. Association of story requirements (SRS) to design

Link	Description	SRS	Design
1	Username cannot contain special char	Character set to denote username	Missing char set domain
2	Sparsity of required fields	Order of input data fields	Regroup required fields
3	Error log recording	Missing requirement for error items	Revised log file design

Table 5. Result statistics of the development process

Error found	Frequency	Remarks
Misspelling	31	Human errors
Inconsistency of FR	2	SRS revision
No error recording	1	Document tracing
Poor writing of content, explanation, definition	14	Technical writer assistance
Poor variable/technical naming	26	Code walkthroughs
Programming style	9	Beautify code format
Pagination, indentation of document format	5	Reformat/editing

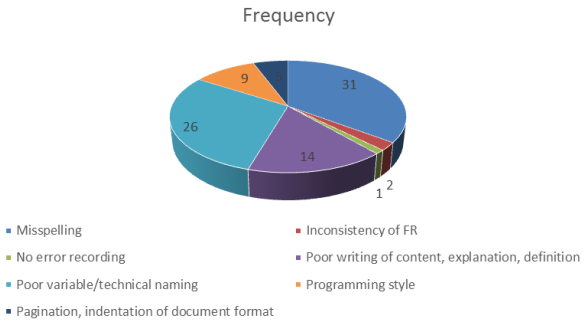


Fig. 8. Registration cross-check.

V.CONCLUSION

We have embarked on deploying conventional UML diagrams in conjunction with tagging or labelling technique to create development checkpoints. The process employed the proposed algorithm to describe the development activity details using standard UML diagrams (Table 1-2, and Figure 1), their precedence relation (Figure 1), and order of workflow (Figure 3). The proposed technique would help pinpoint requirements and specification construction which are tedious and error-prone human activities to undergo an effective harness. Related information can then be traced accordingly during the implementation. As a consequence, many careless or overlooked frequent mistakes are corrected and reduced as shown from the above experimental results. It is apparent from the contribution of this research work that proper use of pragmatic algorithms and/or procedures on well-known development tools such as UML can help reduce many human errors and improve the quality of software product. Future work will focus on tool development for document cross reference to reduce human errors of the development process. These tools should help simplify error reduction procedure of the manual work in the development process.

REFERENCES

1. M. Al-Maolegi, and B. Arkok, "An Improved Apriori Algorithm for Association Rules," *International Journal on Natural Language Computing (IJNLC)*, vol. 3, no. 1, February 2014.
R. R. Lutz, *Targeting Safety-Related Errors During Software Requirements Analysis*, Available: https://www.sciencedirect.com/science/article/pii/0164121295000771
3. Object Management Group (OMG), *OMG Unified Modeling Language Specification*, Available: https://www.omg.org/spec/UML/, version 2.5.1.
4. N. OKAZAKI, Y. MATSUO, and M. ISHIZUKA, "Improving Chronological Sentence Ordering by Precedence Relation," *Proceedings of the 20th International Conference on Computational Linguistics*. Aug 23–Aug 27, 2004, Geneva, Switzerland, pp. 750-756.
5. *Precedence Diagramming Method (PDM)*, Available: http://courses.ce.metu.edu.tr/ce434/wp-content/uploads/sites/48/2019/04/Lecture-Notes-PDM.pdf.
6. X. Yuan, "An improved Apriori algorithm for mining association rules," *AIP Conference Proceedings 1820*, 080005 (2017), Available: https://doi.org/10.1063/1.4977361, Published Online: 13 March 2017
7. J. Yabing, "Research of an Improved Apriori Algorithm in Data Mining Association Rules," *International Journal of Computer and Communication Engineering*, vol. 2, no. 1, pp. 25-27.
8. Yucong Duan, Guohua Fu, Nianjun Zhou, Xiaobing Sun, Nanjangud C. Narendra, and Bo Hu, "Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends," *IEEE 8th International Conference on Cloud Computing*, 2015, pp. 621-628.
9. M. R. Joshi, and Bhagyashri V. Tikar, "Cloud-Computing its Services and Resent Trends," *International Journal of Computer Science and Mobile Computing*, Vol.4, Issue.2, February 2015, pp. 136-143.

ACKNOWLEDGMENT

Nalinee would like to thank the students of CS Program for their contribution on this research. Suan Sunandha Rajabhat University and the Faculty of Science and Technology for the provision of research facilities.

AUTHORS PROFILE

Nalinee Sophatsathit receives her bachelor in Computer Technology from Rajabhat Nakorn Pathom University, Masters in Computer Science from Chulalongkorn University, and PhD in Innovation Management from Suan Sunandha Rajabhat University. She is a faculty member in Computer Science Program, Faculty of Science and Technology, Suan Sunandha Rajabhat University. Her research interests are innovation in IT, software usability, user-oriented innovation management. She can be reached at nalinee.so@ssru.ac.th