Towards Generation of Software Development Tasks

C. Albert Thompson
Department of Computer Science
University of British Columbia, Vancouver, B.C. V6T 1Z4 Canada
Email: leetcat@cs.ubc.ca
Website: http://c.albert-thompson.com
Advisor: Gail Murphy

***Abstract*—The presence of well defined fine-grained sub-tasks is important to the development process: having a fine-grained task context has been shown to allow developers to more efficiently resume work. However, determining how to break a high level task down into sub-tasks is not always straightforward. Sometimes developers lack experience, and at other times, the task definition is not clear enough to afford confident decomposition. In my research I intend to show that by using syntactic mining of past task descriptions and their decomposition, I can provide automatically derived sub-task suggestions to afford more confident task decomposition by developers.**

I. Introduction

A developer's day typically involves working on a number of different tasks [1], [2]. Descriptions for these tasks currently come from the developers' analysis of the task problem. Larger sized tasks, such as user stories, are broken down by developers into smaller sub-tasks. Sometimes these tasks and sub-tasks are explicitly captured in a repository, such as an issue repository. Other times, some or all of the tasks remain implicit.

The efficiency and quality by which development occurs is affected by the task breakdown and representation. Studies by Mark et al. show that developers have troubles returning to a task when they are interrupted [3]. Other studies by Mark finds that multi-tasking introduces more stress into a developers environment [2]. These effects impact the efficiency of the development process. The efficiency can also be impacted by inappropriate task breakdowns that might negatively impact the parallelism of development. The quality of a system can also be affected by missing appropriate tasks, such as potentially missing testing tasks that should be performed.

My thesis is that the efficiency and quality of software development can be improved through the automatic generation and recommendation of task breakdowns to developers based on patterns learned from development tasks specified in projects. To illustrate the basic idea, consider the following user story (a task), CONN-998 in Table I, from the `Connect` project [1] that is building open source software to support an exchange of health information across organizations. Given this task, I aim to recommend sub-tasks that break a task down into specific work units. The existing sub-tasks for CONN-998 provide an idea of the kinds of sub-tasks that must be generated. CONN-1033 describes a sub-task to do research before making an implementation, and CONN-1034, on Table I, is more descriptive and describes what direct configuration service the developer should implement. My aim is to generate these sub-tasks with a focus on the generating the first kind of sub-task that leverages patterns in the task for `Connect`.

[1]http://www.connectopensource.org verified on November 2014

TABLE I
Tasks taken from the Connect issue repository

Task ID	Task Title
CONN-998	As a CONNECT developer I want to create direct configuration services which fit into the CONNECT tech stack.
	Sub-Tasks Titles
CONN-1033	As a CONNECT developer I want to implement some Direct configuration web services.
CONN-1034	As a CONNECT developer, I want to implement Spring/Hibernate Direct configuration services as part of CONNECT so that I can change Direct config settings through the Admin GUI.

The contributions of this research are anticipated to be:

- An approach to learn patterns about task structure and contents from existing project task (issue) repositories.
- An approach to recommend sub-tasks based on learned patterns.
- A tool that delivers sub-task recommendations effectively to a developer.

II. Related work

The idea that steps could be generated to solve a problem has been considered in the field of AI planning. In planning, computers are given a goal and a set of rules that they can follow and are asked to find a way to accomplish the goal using AI algorithms such as Markov chains, or decision trees [4], [5]. The research revolves around ways to understand the similar set of rules needed to accomplish a goal [6]. My research is in that there is a goal, a task, but differs from planning in lacking a predefined set of rules to determine how to archive a goal.

A number of approaches consider how to have a human generate the sub-tasks to be worked on. In Soylent a writer can create tasks to be preformed later, such as editing and proof

reading short texts while writing [7]. CrowdForge uses crowd sourcing to help break down audio transcription or writing task into smaller tasks [8]. In the domain of software engineering, microtask programming semi-automatically decomposes a task into microtasks [9]. The task generation happens when a developer, while programming, inserts a comment containing pseudo code, that is then detected and used to create a microtask. All these approaches assist the developer in making sub-tasks that have been explicitly identified by the developer. This research attempts to determine and automatically generate sub-task.

Work done by Bettenburg et al. informs task creators what additional information should be included in a task to assist a developer [10]. This encourages task creators to submit better bug reports. This work focuses on improving a newly created task, I can use the good task creation concepts described in [10] when assisting in developers in task creation.

Scerri et al. and Mizoguchi et al. have both considered identifying actions found in natural language [11], [12]. Research by Scerri et al. automatically extracts action items from emails using natural language [11]. They create a model of classification using five grammatical, syntactical and linguistic features of natural language to capture action-object-subject tuples. Mizoguchi et al. describes the idea of a task ontology which includes goals, verbs, nouns, adjectives, and verbs that are constrained to object, calling them "constraint verbs" [12]. These results can be used as a basis for developing the proposed approach for sub-task generation.

III. Initial Analysis

To investigate whether there is promise in learning task patterns in a project, I consider the following three questions:

1) Do there exist patterns in tasks regardless of the hierarchical or other relationships in tasks?
2) Are there patterns in hierarchical (parent-child) relations of a task, between one parent and its children?
3) Can the hierarchical task patterns be categorized meaningfully?

To investigate these questions I use two open source issue repositories: The `Connect` and the `Mylyn` project[2]. I focus in this paper on the results from `Connect`, reporting the results from `Mylyn`, which were similar, in Table II without discussion.

1. Are there patterns between tasks regardless of any hierarchical or other relationship?

Issue repositories contain a rich set information that can be used to make informed recommendations of sub-tasks, such as description, comments, task type, etc. In my analysis to date, I have focused on the title of a task and its hierarchical relationship to other tasks. The regularity of task titles determined by Ko et al. [13] is the reason I have focused on titles.

Mizoguchi et al. found that objects related to verbs are important in the task ontology [12]. I use this approach to look for patterns in issue repositories. For each of the considered issues repositories, I apply the Stanford language parser[3] to determine all the verb-noun instance pairs that exist in each task title. 2736 verb-noun instances, of which 1831 are distinct, were found in `Connect`. This high percentage of distinct verb-noun instances is not unexpected. It is not likely a developer would repeat the same action on the same programming artifact multiple times. Rather the developer might repeat the same action, using the same verb, on different programming artifacts.

TABLE II
Data results from both `Connect` and `Mylyn` projects

	Connect	**Mylyn**
Repository Stats		
Tasks with hierarchical relation	1184	1503
Verb-noun instances	2736	3092
Distinct verb-noun instances	1831	2541
Distinct verbs	318	403
Distinct nouns	720	909
Rules		
Verb-noun rules	723	100
Verb rules	361	422
Noun rules	458	370
Categories		
Formulaic relationships	425	121
Feature→design relationships	359	664
Uncategorized relationships	400	718

A critical part of a task description is a verb, with the verbs being applied to different nouns related to different parts of a system. In `Connect`, there are 318 different verbs used, and 720 different nouns. I can leverage the similarity in verbs and nouns in an approach to generate and recommend sub-tasks.

2. Are there patterns in parent-child relations of a task, between one parent and its children?

It may be that finding patterns within task hierarchies may be more likely to occur than generally across a issue repository. To investigate this question, I used the SPMF data mining mining library[4] to do association rule mining to extract rules from within hierarchies. The first data set used the verb-noun instance to see what were the repeated actions on programming artifacts taking place. The second data set and third data sets were the verbs and the nouns separately analyzed using association rule mining.

Association rule mining on the first data set (verb-nouns) found 723 rules that in the `Connect` issue repository with support of at least two for each rule. Here is an example of one rule found: if a parent task contains the verb-noun, "*replace metro*" then child task will have "*use CXF*" with a support of 10 occurrences and a confidence of 11% see Table III. I manually examined 30 random rules and found the sub-tasks of each rule were nearly identical differing only in a few words. The rules covered 425 hierarchical relationships out of a total 1183 relationships in the `Connect` issue repository.

[2]http://eclipse.org/mylyn/ verified on November 2014

[3]http://nlp.stanford.edu/software/lex-parser.shtml verified November 2014

[4]http://www.philippe-fournier-viger.com/spmf/ verified November 2014

TABLE III
AN EXAMPLE FROM THE SYSTEM CATEGORY, WITH CHANGES UNDERLINED

Task ID	Task Title
GATEWAY-2302	Using Apache CXF and OpenSAML, I would like to replace Metro in XDR
	Sub-Tasks Titles
GATEWAY-2162	Using Apache CXF and OpenSAML, I would like to replace Metro in XDR Client
GATEWAY-2303	Using Apache CXF and WSS4J, I would like to replace Metro in XDR Server

TABLE IV
AN EXAMPLE FROM THE FEATURE→DESIGN CATEGORY

Task ID	Task Title
CONN-1065	As a CONNECT user I want the Admin GUI to display which remote HCID's the local gateway is communicating with so that I can analyze message traffic.
	Sub-Tasks Titles
CONN-1025	As a CONNECT developer, I want to be able to query event logs, to display information for message tracking.
CONN-1027	As a CONNECT developer I would like a Admin GUI page to display the remote gateways that the local gateway is communicating with.

Association rule mining found 361 rules with verbs and 458 rules with nouns with support of at least two. 30 rules from both sets were manually examined, in each rule many of the tasks in the hierarchical relationships covered a similar feature and the sub-tasks described steps to implement the feature. The rules I found covered 359 hierarchical relationships.

3. Can the hierarchical task patterns be categorized meaningfully?

I examined 30 rules from each data set to see if the rules fall into different categories: three categories were determined. The first category is a `system` category which had 425 relationships, where the hierarchical relations in the category were very similar to each other, differing only to which part of the system a sub-task pertains. Consider the example shown in Table III. In this example only Client/Server was added and WSS4J was change in the sub-tasks. The second category is `feature→design` which had 359 relationship. This category includes tasks were tasks that focused on implementation of a feature and the sub-tasks outlined the steps in design. An example is provided in Table IV. There was no obvious categorization for the remaining 400 relationships; for now, I have placed these in a miscellaneous category pending further investigation.

To date, my focus has been on understanding patterns that exist in issue repositories and on how those patterns might be used for sub-task generation.

IV. RESEARCH PROGRESS

I intend to mine historical issue data to create an approach to generate and recommend sub-tasks to developers. This approach can use information in a task a developer is assigned and the rules found in the project repository. Restricting recommendations based on a category and type of task may improve results. For example recommendations for feature tasks may be possible, while recommendation for bug tasks may not be possible. There may also be an opportunity to learn rules between different project repositories.

Before I can make recommendations to developers more analysis is needed on how patterns may be exploited and abstracted to compose a sentence to describe recommended tasks. To do this I will need enough project context to state generated tasks in the lexicon of the project. I may also need to put multiple verb-noun instances together to make something useful for a developer.

Even if appropriate tasks can be generated, there are still questions of how best to deliver recommendations to developers. Several approaches exist discussing ways to present information to a developer in an responsive application: a multi-layer user interface based on user expertise [14], an adaptive interface changeable by the user [15], or an adaptive interface automatically changed based on usage from a user [16].

V. EVALUATION METHODOLOGY

To evaluate recommendation of sub-tasks and tool developed as part of this research, there are 4 research questions to be answered.

1) Is it possible to recommend tasks and sub-tasks using patterns?
2) How can patterns be combined in a category to form meaningful sub-task titles and descriptions?
3) Given a recommendation how can they be ranked for a developer?
4) What ways can recommendations be effectively presented to a developer?

Section III and IV described how the approach and mechanism might be developed. Assuming I am able to develop an approach and delivery a mechanism, I will also need to evaluate how the approach impacts the quality and efficiency of a development process.

To look at effectiveness, I will first determine over a number of project how many existing explicit sub-tasks can be found using our approach. I will also ask developers that work on those project to assess sub-tasks not found in repositories. Second, I will ask developers to use the tool and will compare use if more sub-tasks are being explicitly captures.

To look at quality, a longitudinal format study can be used. I will give the tool to developers to use for a month. I will then analyze the sub-tasks in the repository to see if more steps are being used in a process of software development. If the developers are capturing more sub-tasks then I can say a detailed process is being followed.

ACKNOWLEDGMENTS

This work was funded in part by IBM and in part by NSERC through the NECSIS project. Thank you to my adviser Gail.

REFERENCES

[1] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann, "Software developers perceptions of productivity," *SIGSOFT FSE, to appear, ACM*, 2014.

[2] G. Mark, Y. Wang, and M. Niiya, "Stress and multitasking in everyday college life: an empirical study of online activity," in *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*. ACM, 2014, pp. 41–50, difficulty in working late night and stress to developer.Describes different tasks Social media, email, web, Facebook, academic on stress.

[3] G. Mark, V. M. Gonzalez, and J. Harris, "No task left behind? examining the nature of fragmented work," in *in: Proceedings of ACM CHI 2005, ACM*. ACM Press, 2005, pp. 321–330, how developers cope with being distracted while working on tasks.

[4] Y. Freund and L. Mason, "The alternating decision tree learning algorithm," in *ICML*, vol. 99, 1999, pp. 124–133.

[5] W. R. Gilks, *Markov chain monte carlo*. Wiley Online Library, 2005.

[6] S. Russell, P. Norvig, and A. Intelligence, *A modern approach*. Citeseer, 1995, vol. 25.

[7] M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. Panovich, "Soylent: A word processor with a crowd inside," in *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '10. New York, NY, USA: ACM, 2010, pp. 313–322. [Online]. Available: http://doi.acm.org/10.1145/1866029.1866078

[8] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut, "Crowdforge: Crowdsourcing complex work," in *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '11. New York, NY, USA: ACM, 2011, pp. 43–52. [Online]. Available: http://doi.acm.org/10.1145/2047196.2047202

[9] T. D. LaToza, W. B. Towne, C. M. Adriano, and A. van der Hoek, "Microtask programming: Building software with a crowd," in *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '14. New York, NY, USA: ACM, 2014, pp. 43–54. [Online]. Available: http://doi.acm.org/10.1145/2642918.2647349

[10] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. SIGSOFT '08/FSE-16. New York, NY, USA: ACM, 2008, pp. 308–318. [Online]. Available: http://doi.acm.org/10.1145/1453101.1453146

[11] S. Scerri, G. Gossen, B. Davis, and S. Handschuh, "Classifying action items for semantic email." in *In Proceedings of the 7th International Conference of Language Resources and Evaluation*, 2010, pp. 3324–3330.

[12] R. Mizoguchi, J. Vanwelkenhuysen, and M. Ikeda, "Task ontology for reuse of problem solving knowledge," *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing*, pp. 46–59, 1995.

[13] A. J. Ko, B. A. Myers, and D. H. Chau, "A linguistic analysis of how people describe software problems," in *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*. IEEE, Sept 2006, pp. 127–134.

[14] B. Shneiderman, "Promoting universal usability with multi-layer interface design," in *ACM SIGCAPH Computers and the Physically Handicapped*, no. 73-74. ACM, 2003, pp. 1–8.

[15] S. Greenberg, *The computer user as toolsmith: The use, reuse and organization of computer-based tools*. Cambridge University Press, 1993.

[16] J. Mitchell and B. Shneiderman, "Dynamic versus static menus: an exploratory comparison," *ACM SIGCHI Bulletin*, vol. 20, no. 4, pp. 33–37, 1989.