How Much does integrating this Commit Cost? -A Position Paper

Yujuan Jiang, Bram Adams
MCIS, Polytechnique Montreal,
Email: {yujuan.jiang, adams.bram}@polymtl.ca

Daniel M. German
University of Victoria, Canada
Email: dmg@uvic.ca

***Abstract*—Integration is a common development activity that fuses multiple pieces of source code together, either developed in-house or acquired from a third-party (e.g., Git commit). Integration essentially combines software that was developed in parallel by different developers in different teams or organization. Thus it requires to iron out inconsistent and conflicting changes. Inconsistencies are costly and can only be identified a posterior. It is very hard to judge if it is worth integrating a code commit, or which of two equivalent commits or libraries to integrate. If there is an indication about the effort the integration will take, it will help the integrators to make dicisions easier. In this paper, we propose the ISOMO model to quantify the projected cost of integrating a code commit. This will help people to evaluate how much effort a commit will cost and decide if it is worth being integrated.**

I. Introduction

Consider the following two scenarios. On the one hand, Marius is a Debian package maintainer tasked with the job of packaging version 3.0 of the BLOB open source project. Having found two security holes in version 2.0 that have not been acknowledged by the project, Marius is anxious about the new version. Will it incorporate the Debian-specific fixes he made or will he have to modify and re-integrate them himself? For sure, he is reluctant to glance through this version's diffs to analyze every possible change compared to the old version. On the other hand, John is a Linux kernel developer who would like to adopt recent optimizations made to the Linux kernel memory subsystem by an Android vendor. The Android project's changes are not automatically ported back into the main kernel version, and John is foreseeing many integration issues caused by hardcoded dependencies to Android code. Fixing those inconsistencies may be too expensive. Although both cases are different, they share the same problem: how to estimate the cost of software integration?

Integration is the common activity in software developement that merges multiple pieces of source code into an existing project to result into consistent whole. Integration is necessary to reuse a library or change in a project, as illustrated by the Debian and Linux kernel examples. Instead, source code developed in one particular context must be adapted to work in another context. As such, integration must make this adaptation to avoid incompatibilities or, in the worst case, failure of the host project.

To achieve this, integrators (i.e., anyone performing some kind of integration activity) potentially might need to spend a lot of effort in seemingly mundane activities like trying to understand changes to a particular region in the code (Marius) or the dependencies of an interesting code change (John). Before spending this effort, they would like to know how much effort they should expect? In this paper, we assume that the business case for integration has been made and the project has been convinced that a particular commit's functionality and implementation deserves a look. However, is this integration worth the effort, i.e., are the expected benefits higher than the cost? Will this integration cause problem like, for example, additional library dependencies?

Currently, the answers to these questions are unknown before integrating a commit. The developer can only judge by her personal experience, which is not a measurable and reproducible source. There is some work on predicting the risk of software changes [2], but those techniques can only tell an integrator whether or not a commit may be risky, but not how much exactly this integration will cost and why. In addition, risk is a much more general concept, encompassing bugs, integration effort, and other kinds of impact like customer happiness. Hence, even though an integration may be risky, a developer might still decide to do it because it brings more benefit in the long term. However, the current techniques cannot measure such kind of cost and benefit quantitatively.

In this paper, we propose a cost model that quantifies how much integration will cost. As such, it captures the effort of integrating a commit (integration concerns various "artifacts" like external libraries, components, code changes, we are going to talk about the case of "commits") into a project. The outcome is a number based on which the integrator can predict his workload and trade-off between the cost and benefit of integrating a new piece of code change (a positive value indicates the cost while a negative value indicates the benefit). To our knowledge, our work is the first to **quantify** the effort of integration activity. Apart from the cost model and its uses, we discuss challenges that we encountered in making a concrete implementation of the model. For this is a position paper, we look forward feedback by the workshop participants to find solutions for these challenges and further develop the cost model.

II. ISOMO Model

To evaluate the cost of integrating a commit into a project, we propose the ISOMO model (Integration of Software cOst

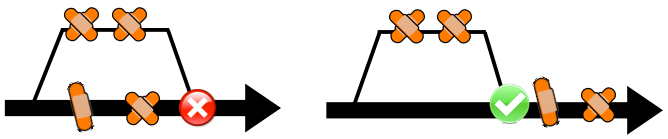


Fig. 1. Rebasing two commits in the bottom branch to update them to the new state of the project after merging the top branch's commits.

MOdel). When a particular commit has to be integrated, ISOMO computes the expected integration cost by analyzing the characteristics of the commit and the host project. The resulting cost gives a quantitative indication of whether it is worth further pursuing the integration (e.g., there might be "cheaper" alternatives).

ISOMO computes the cost of a commit based on the following four cost categories (see Table I) covering ten sub-factors. The ten cost sub-factors are based on previous work about integration conflict detection [2], risky software evolution [7], and guidelines to users of version control systems [6].

- **Merge cost (MG).** This cost happens at the start of integrating a new commit, such as analyzing the possible risk, changing the integrated commit to adapt it to the host project (cherry-picking and–or making a local patch) or changing the host project (i.e., updating the platform version) to adapt it to the integrated change (rebasing, see Figure 1).
- **Update cost (UD).** After integrating a commit, newer versions might require revising the integration and (in the worst case) updating local patches or cherry-picking parts. For example, the cherry-picking parts have dependencies to the original whole patch, so we need to complete more parts of the patch to update the integrated commit.
- **Maintenance cost (MT).** This cost is about the effort of maintaining the integrated code as the host project evolves (in contrast to update cost).
- **Removal cost (RM).** It is possible that an integrated change will be removed either because it caused too many problems or the project evolved in time. Depending on how strongly coupled an integrated change is to the host project, the cost of removal might be high or low.

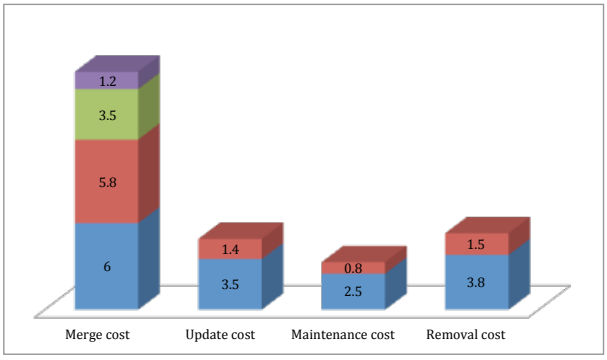
Table I lists each of these cost factors and their sub-factors in more detail. To calculate the total cost, we propose to use the following formula:

$$C = \sum_{i=1}^{10} a_i \times F_i$$

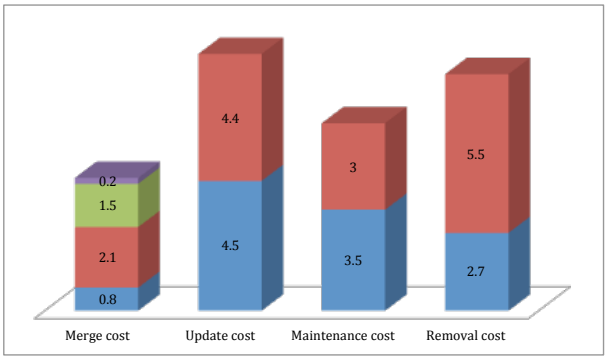
F_i are the factors in Table I and a_i are the weights of the cost factors. This formula is an initial proposal, we still need to improve our model in future.

III. ISOMO, WHAT IS IT GOOD FOR?

With the ISOMO model, we can quantify multiple activities in the software development process.



(a) The cost profile of a commit of high merge cost.



(b) The cost profile of a commit of high update cost.

Fig. 2. Profiles of two commits.

A. Building an Integration Profile

For each commit that an integrator wants to integrate, we could predict its cost profile by ISOMO, i.e., the effort for the integrator integrating it into his project, measured in terms of the ten cost sub-factors. Such a profile not only allows the integrator to have an initial impression of the effort of integration for a particular commit, but also to compare the effort of integrating two different commits.

Figure 2(a) and Figure 2(b) show two different integration profiles. On the one hand, the one in Figure 2(a) has a higher merge cost up front but lower maintenance and update costs: the initial integration may need a lot of work but the integrated code does not require many updates or maintenance. On the other hand, the profile in Figure 2(b) has a lower initial merge cost but rather high maintenance and update costs. It is easily adapted into the project, but is potentially incompatibile in the long run and will continuously force the integrator to fix future issues. For example, if a commit adds new dependencies on five libraries, it is coupled with these external libraries. Afterwards, if the functions of an external library change, then every related statement in this commit source code should change.

If the integrator can foresee the effort, she could make different decisions at the beginning to avoid future loss. For example, in the motivational example in the introduction, if Marius had predicted the cost of his local patch, he could have decided to put more effort into getting his local patch accepted upstream by the BLOB project.

TABLE I
FOUR CATEGORIES OF INTEGRATION COST.

Cost category	Sub-factor	Description
Merge cost	Risk Analysis	Physical analysis of a commit (typically during reviewing) to check whether it is worth being integrated into the project.
	Cherry-Picking	Picking up only part of a change for integration, the other part is abandoned.
	Rebasing	Ongoing (i.e., uncommitted) changes to the project have to be adapted to make use of the integrated change and be compatible with any local changes.
	Local Patch	Local modifications to a commit to make it work with the project at hand.
Update cost	Local Patch	Each time an integrated commit is updated upstream (i.e., by its owner), the host project's local patches might be invalidated and require modifications
	Cherry-Picking	Each time an integrated commit is updated upstream (i.e., by its owner), the parts cherry-picked (i.e., selected) by an integrator might have evolved and become dependent on the abandoned part. Hence, the integrator must reconsider the selected and abandoned parts.
Maintenance cost	Local Patch	Submit a new commit to make up for the flaws after discovering the problems caused by the previous commit.
	Adaptation Cost	Adapting/updating the way in which the commit is integrated.
Removal cost	Roll-Back	If crucial issues are identified shortly after integration, it is often still possible to some degree to roll back the integration (and other activities like rebasing) to avoid further loss.
	Refactoring	If rollback is not possible (e.g., a lot of changes have been made based on the commit), the project must be refactored to eliminate or neutralize the integrated changes.

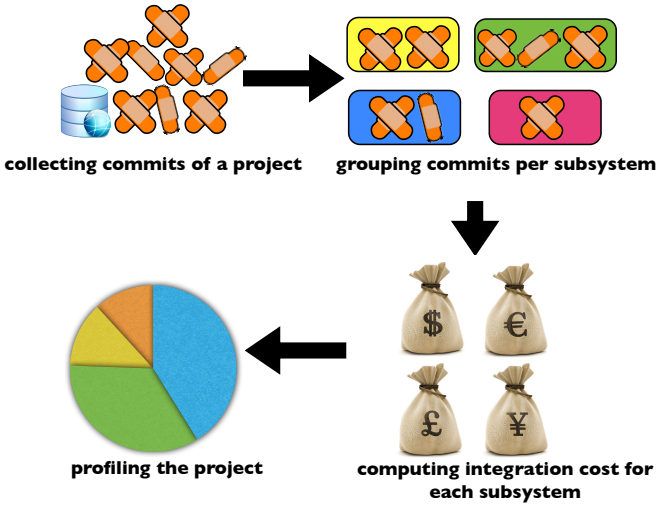


Fig. 3. The process of building a subsystem-level cost model.

B. Evaluating the Integration Cost of a Project

With the ISOMO model, we can also help a project evaluate its overall integration costs. This enables integrators to use concrete numbers when talking to their managers, even broken down across concrete cost sub-factors. If the project has a high merge cost, it indicates that it focuses especially on adding new functionality by merging continuously new commits, whereas if a project has a high maintenance cost, it indicates that integrated code plays a very important role in the project and its future evolution. Such a breakdown of costs may help a project rearrange its development and integration strategy according to a trade-off between benefit and effort.

In practice, the cost of a project is the sum of the cost of all integrated changes into a project. Similarly, the total cost of a subsystem is the sum of the cost sub-factors across all integrated changes.

C. Predicting the Cost of Reuse

Reusing part of an existing project can save time and take advantage of a lot of stable features. However, anyone trying to integrate part of a project during reuse also pays the effort of fixing integration issues when adapting the reused part to a new context. Knowing the integration cost that one should expect when integrating a particular subsystem may help integrators to judge which part is easier to integrate. If organizations who previously integrated a subsystem of a library could submit their integration profiles to the library's organization, the latter could advertise these profiles for future customers, and even make concrete predictions of integration effort for these customers.

Again, computing such a subsystem-specific cost only requires summing up the cost of all code changes previously integrated in the subsystem, as is shown in Figure 3.

IV. HOW TO IMPLEMENT THE ISOMO MODEL?

A. Collecting Integrated Commits

To implement the ISOMO model, we must obtain information about the commit to integrate (e.g., from version control system). If we want to build models for individual integrators, we collect all previous integrated changes of a particular integrator. If we want to build models for subsystems, then we must collect the related data and break it down for each subsystem.

B. Extracting Characteristics

To analyze the cost of an integration across different sub-factors, we must understand the background, the characteristics, and all related information of the integrated code and the host project, such as the size, the author, the commit date, the impacted subsystem and so on.

C. Calculating the Integration Cost

After collecting all the required characteristics, we could build the ISOMO model according to the intended usage scenario (see Section III) and different requirements, such as building models to predict the effort of an integrator or to measure the integration effort of a subsystem. In the latter case, for example, the specific steps of building the ISOMO model are:

- Grouping all past integrations of multiple integrators according to the impacted subsystems.
- Extracting the characteristics from each integration.
- Analyzing the possible cost in ten different sub-factors.
- Assigning weights to each sub-factor's total cost, generating the resulting total cost value.

D. Quantitative and Qualitative Analysis

After building cost models for the required context, we could do more quantitative and qualitative analysis based on the output depending on the specific use case.

V. Challenges Encountered

Implementing the ISOMO model faces multiple challenges.

- **Determing the metrics for measuring the costs.** The biggest challenge currently is how to determine the metrics to measure the four different kinds of costs. We propose to measure different cost factors with different metrics. For example, for "Cherry-Picking" and "Rebasing" of "Merge cost", we propose to use the number of changed lines of code (LOC) to measure the cost. For the "Roll Back" and "Refactoring" of "Removal Cost" we propose to use the number of impacted files to measure. However, there still exist some sub-factors that we cannot measure, such as the risk analysis sub-factor.
- **Assigning the weights to each sub-factor.** There are ten sub-factors to make up the whole integration cost. Each sub-factor might play a different role in the integration cost. Some may be more important while some are less important and commits may interact with each other, e.g., each commit may depend on others, which may impact the sub-factors. Hence, we must determine the weights of the different sub-factors.
- **Evaluating the ISOMO model.** A final challenge is how to measure the performance of our model. To evaluate a model, generally speaking, it needs a training set and a testing set, which is an oracle containing the known cost of a set of integrations. However, to the best of our knowledge, our research is the first to quantify the cost of the integration process, there is no existing data set yet. To solve this problem, we expect to get help from this workshop, hoping that some institution may have such kind of data or their own evaluation mechanism.

VI. Related work

The related work mainly includes work on software integration and cost models.

Jiang et al. [4] analyzed the integration process of Linux kernel. They found that the integration process usually takes 1-6 months to integrate a commit into the official release branch and only around 33% of the commits can be accepted into an official release. This work showed the relatively low efficiency of the integration activity, motivating us to propose a model to help integrators early on to identify potentially risky commits to merge.

Brun et al. [2] studied different kinds of merging conflicts, and concluded reasons why integration can be risky: (1) The integrator is only in charge of integrating, not developing the code, (2) the integrated change is too large and complex, (3) merging typically happens a relatively long time after the actual development, and (4) most of the serious conflicts are due to semantics instead of textual issues.

Bird et al. [1] proposed a what-if analysis that allows to distinguish harmful, redundant branches that could impact software quality. This simplification can avoid merge conflicts such as incompatibility in the integration process, which inflates integration time and brings trouble. Their empirical evaluation proved that such a simplification can reduce integration time by up to nine days. They only consider time as a cost, while our model considers a wider variety of metrics.

Research about cost models in software engineering has a long history. The classical model COCOMO [5] for evaluating the cost of the development process is a widely used parametric model estimating the cost at the beginning of a project.

Cost models are also used in many other fields such as software testing. Dalal et al. [3] conducted a large empirical study analyzing the effect of cost model-driven testing in software infrastructure and proved its performance. Our model is the first to quantitatively measure the integration activity for a project.

VII. Conclusion

In this paper, we propose the ISOMO model to compute the cost of software integration divided into four different dimension and ten sub-factors. To our knowledge, this model is the first one that quantifies the integration cost. Currently, this model is in a very early stage and must be improved further as well as evaluated in practice.

Acknowledgement

A special thank you goes to the reviewers and workshop participants for their valuable feedback.

References

[1] C. Bird and T. Zimmermann. Assessing the value of branches with what-if analysis. In *Proc. of the ACM SIGSOFT 20th intl. symp. on the Foundations of Software Engineering (FSE)*, pages 45:1–45:11, 2012.

[2] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin. Proactive detection of collaboration conflicts. In *Proc. of Foundations of Software Engineering (FSE)*, pages 168–178, 2011.

[3] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-based testing in practice. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 285–294, New York, NY, USA, 1999. ACM.

[4] Y. Jiang, B. Adams, and D. M. German. Will my patch make it? and how fast? – case study on the linux kernel. In *Proc. of the 10th IEEE Working Conf. on Mining Software Repositories (MSR)*, pages 101–110, 2013.

[5] C. F. Kemerer. An empirical validation of software cost estimation models. *Commun. ACM*, 30(5):416–429, 1987. Corrigendum: CACM 30(9): 770 (1987).

[6] J. Loeliger. Version control with git - powerful tools and techniques for collaborative software development, 2009.

[7] T. Mens and S. Demeyer. *Software Evolution*. Springer, 2008. ISBN 978-3-540-76439-7.