Logical Coupling Based on Fine-Grained Change Information

Romain Robbes Damien Pollet Michele Lanza
REVEAL @ Faculty of Informatics — University of Lugano, Switzerland

Abstract

Logical coupling reveals implicit dependencies between program entities, by measuring how often they changed together during development. Current approaches use coarse-grained change information extracted from the version control history of the software system under study. Entities that are registered as having changed during a commit transaction have their coupling increased by the same amount, regardless of how and how much they actually changed. We present several new logical coupling measures taking into account fine-grained semantic changes. We evaluate their respective accuracy compared to the classical logical coupling measure on two case studies; in particular, we evaluate how well the new measures can estimate logical coupling with less data. Results show that our approach based on fine-grained information greatly ameliorates the state-of-the-art of logical coupling detection.

1 Introduction

Coupling is a measure of how much entities in a software system depend on each other. Knowing the coupling between entities is useful for a variety of development or management activities, such as maintenance effort prediction or program comprehension [2]. There is a variety of coupling measurements, usually related to the structure of the system under study. Coupling can be based on various program relationships, such as the number of calls between two entities, variable accesses, or inheritance relationships. Over the years, several alternative measures of coupling have been proposed, such as dynamic coupling [1], logical coupling [4] or conceptual coupling [7].

We focus on logical coupling, which is based on the change history of entities. The rationale behind logical coupling is that *"entities which have changed together in the past are bound to change together in the future"*. Logical coupling has been used for change prediction [15, 14], and reverse engineering [3, 6]. While useful, logical coupling is not as accurate as it could be: It is computed from the versions of the system archived in a SCM such as CVS or Subversion (From now on, we refer to this measure of logical coupling as *SCM logical coupling*). In previous work, we have shown that using versioning systems for software analysis has shortcomings [9].

We believe a finer-grained description of the evolution of a system provides a more accurate measure of logical coupling. SCM logical coupling is imprecise because the finest resolution is the commit transaction. All files in a transaction have their coupling increased by the same amount, regardless of how and how much they changed.

In this paper, we investigate how much logical coupling can be improved when using information gathered at development time. In previous work, we developed a tool platform [11] which records *all changes* made on a system while it is being implemented, and stores it in a change repository [8]. This change information retains the precise time information of when each change was made, and concerns program-level entities such as classes and methods, not only files.

The contributions of this paper are:

- Several novel measures of logical coupling at the class level, using fine-grained change data. These measures take into account the amount of changes and the precise date when the changes were performed.
- A comparative evaluation of the new measures with respect to the initial SCM logical coupling, using the change history of two case studies, and assessing how well these measures can estimate logical coupling with less data.

Structure of the Paper. Section 2 explains the shortcomings of SCM logical coupling, describes its usages and the approaches that address its shortcomings. Section 3 recalls our approach —Change-Based Software Evolution— to record fine-grained changes as they happen in an IDE. Section 4 describes the various alternatives to logical coupling we defined based on the fine-grained change information. Section 5 details our evaluation procedure to measure the respective accuracy of the logical coupling measures. Section 6 discusses our approach, while Section 7 concludes and outlines future work.

2 Logical Coupling

Gall *et al.* first introduced the concept of logical coupling [4] to analyse the dependencies in 20 releases of a telecommunications switching system. The concept was soon adopted by other researchers in the context of reverse engineering and program comprehension. Pinzger used logical coupling as part of his Archview methodology [6] for architecture reconstruction. D'Ambros visualized logical coupling with an interactive visualization called the Evolution Radar [3]. Logical coupling has also been used for change prediction. Zimmermann *et al.* [15] presented an approach based on data mining in which co-change patterns between entities are used to suggest relevant changes in the IDE, when one entity in the relationship is changed by the programmer. Ying *et al.* employed a similar approach [14], although at a coarser granularity level (Ying's approach suggests files, while Zimmermann's employs lightweight parsing to recommend finer-grained entities).

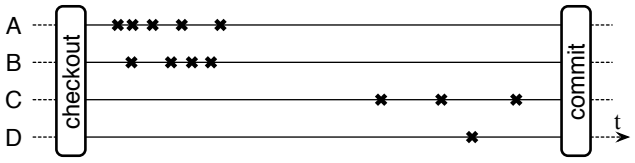

Figure 1. Example development session.

SCM Logical Coupling Loses Information. In this example, four entities, A, B, C and D, are modified during a single development session. The figure shows a timeline for each entity, with a mark every time the entity was changed during the session. It is obvious that entities A and B have a very strong relationship, while entity C and D have a moderate relationship. In addition, the relationships AC, BC, AD or BD, are weak at best. However, based only on the information recovered from the version repository, an SCM-based logical coupling algorithm will give equal values to each relationship. This means that *a large amount of data is needed before the measure can be accurate*. Gall *et al.* used the mean co-change as the threshold to establish logical coupling between two entities —in their case, five co-change occurrences [5]. Similarly, change prediction works much better for projects with a large history, in "maintenance mode", rather than in active development [15].

Addressing Information Loss. Zou, Godfrey and Hassan introduced Interaction Coupling [16]. Interaction coupling is based on IDE monitoring, like our approach, and records navigation and edition events during development sessions. These events are counted at the file level, and the number of *context switches* between two files is the measure of interaction coupling. The measures are also classified in three categories: co-change (the two files changed at least once together), change-view (one of the files changed, while the other was consulted) and co-view (the two files were viewed together). Although the sequence of events is taken into account, the exact date of each event is not. In the same fashion, the nature of the edits is not considered either.

Our approach bears some similarity with Zou's, as they both require IDE monitoring. However our approach considers only changes to the system, but records both the content of the change, and the exact date at which the change occurs. Thus we can replicate their measurement if we only consider co-change relationships, which would enable us to compare it with standard logical coupling.

3 Change-Based Software Evolution

Our logical coupling measures are based on data gathered during our previous work on change-based software evolution (CBSE) [10]. CBSE models software changes as first-class entities. Instead of files, CBSE models the evolution of actual program entities; and instead of relying on the developer to take snapshots, CBSE monitors changes in the IDE *as they happen*.

Our tool silently records the history of abstract syntax tree changes, down to individual expressions. Atomic changes record apply/undo data, author, and precise timestamp. The tool also groups them into larger-grained changes, from editions leading to legal code —the granularity considered thereafter— up to automated transformations or developer sessions. Space lacks to describe the approach, but a more complete account can be found in [10]. Of note, our recording is non-instrusive: The only interaction with the user is when our monitoring plug-in occasionaly asks the user to upload some data.

4 Logical Coupling Measurements

In the following, for any program entity a and session s, we note δ_a for any change concerning a —directly or through child entities, *i.e.,* changes to a method concern its class— and $s_a = \{\delta_a \in s\}$. We compute the long-term coupling $a \leftrightsquigarrow b$ between two entities by aggregating a per-session coupling measure over the history or an interval of sessions:

$$a \overset{\text{hist}}{\leftrightsquigarrow} b \overset{def}{=} \sum_{s \in \text{hist}} a \overset{s}{\leftrightsquigarrow} b$$

The various coupling measures each define their own $\overset{s}{\leftrightsquigarrow}$, as shown in Table 1; we now explain their intuitive meaning.

SCM Logical Coupling (LC). This is the logical coupling measurement introduced by Gall *et al.*. Two entities

Logical (LC): Occurrences of co-change of two entities in a session. $a \overset{LC}{\rightsquigarrow} b \overset{def}{=} \begin{cases} 1 & \text{if } a \text{ and } b \text{ changed during } s; \\ 0 & \text{otherwise.} \end{cases}$
Change-based (CC): How much entities co-changed during a session. $a \overset{CC}{\rightsquigarrow} b \overset{def}{=} \left(\prod_{s_a \times s_b} \lvert s_a \rvert \cdot \lvert s_b \rvert\right)^{1/\lvert s_a \times s_b \rvert}$
Interaction (IC): Interleaving of sequential changes. $a \overset{IC}{\rightsquigarrow} b \overset{def}{=} \lvert s_a \times s_b \rvert$ with δ_a and δ_b successive
Time-based (TC): Proximity in time of changes in a session. $a \overset{TC}{\rightsquigarrow} b \overset{def}{=} \max\left(0,\ 1 - \frac{1}{\lvert s_a \times s_b \rvert} \sum_{s_a \times s_b} \lvert \Delta t(\delta_a, \delta_b) \rvert\right)$

Table 1. Per-session coupling contributions.

are related if they change during the same session. A threshold of five co-change occurrences is often used to qualify entity as logically coupled.

Change-based Coupling (CC). Entities that change many times during a session are more coupled than those which only changed occasionally. We define a session as a period of continuous programming activity without breaks lasting longer than one hour. Change-based coupling CC is similar to the previous LC measure except that the number of changes for each entity is factored into the measure.

Interaction Coupling (IC). This is the coupling introduced by Zou *et al.*, although we consider only the code changes and ignore the navigation events. Each time an entity changes, it becomes the entity in focus. The coupling between A and B is equal to the number of times the focus switched from A to B or from B to A. It is then rounded between zero and one, based on whether the number of context switches is more or less than the average of context switches.

Time-based Coupling (TC). If two entities changed simultaneously, their relationship is stronger than if one changed at the beginning of the session and the other at the end. The coupling linearly decreases with the average delay between changes, from 1 if all changes happened simultaneously to zero if they happened one hour apart or more —this delay being the minimum session gap.

5 Comparative Evaluation

Prediction. The amount of data needed by logical coupling is one of the reasons logical coupling is used more for retrospective analysis, rather than forward engineering. In plain words, if there is not enough data, the measure is useless. For example, Zou *et al.* mention in their study, that the classic measure was unable to find any coupling relationship from a one-month period of data. On the other hand, the coupling they defined did work on shorter periods than the classic logical coupling.

Since we have a longer period of data, we can compare more formally the predictive power of the coupling measures by proceeding as follows:

- Measure the SCM logical coupling (LC) between classes of the system, ignoring relationships below the threshold coupling value used in [5], which is 5. This constitutes the expected set E.
- For each measure m (CC, IC and TC), measure the coupling of each relationship for each session. This coupling is between 0 and 1. If a relationship's coupling is above a certain threshold tr, put the relationship in a candidate set $C1_{m,tr}$. We filter out relationships where an entity has changed less than 5 times overall, since we can not predict anything for these.
- Repeat this procedure for two or three sessions, *i.e.*,, the threshold for a relationship has to be crossed in two (respectively three) sessions in the history. These constitutes candidate sets $C2_{m,tr}$ and $C3_{m,tr}$.

We define the precision P and recall R for a candidate set C, with respect to the expected set E, as: $P = |E \cap C|/|C|$ and $R = |E \cap C|/|E|$. Precision and recall come from information retrieval and give an idea of the accuracy of a prediction [13]. The recall expresses the number of false negatives given by the measure: It is the proportion of expected entities that were predicted. If all the expected entities are predicted, the recall is 1. The precision evaluates the number of false positives in the prediction: It is the proportion of predicted entities that were wrong. If only entities in the expected set are predicted, the precision is 1.

These measures allow us to determine which coupling is the best at estimating the logical coupling with a limited amount of information.

Convergence. Another useful indicator is the rapidity of convergence of the measure: We want to estimate how well a measure identifies coupling from a few recent sessions as opposed to the whole history up to that point in time. We thus compare two measures *based on the entities they identify as most coupled to a set of entities of interest*, in a three-step process:

1. We extract a sample of "interesting" entities among the most coupled according to the few recent sessions;
2. For each interesting entity, we get two "neighbours" lists of the entities it is most coupled with, one from only the recent sessions and the other from all the past history;
3. We compute the Spearman correlation [12] between pairs of neighbour lists, averaged over the set of interesting entities.

Since the Spearman correlation measures differences in order, this method indicates how much two measures agree

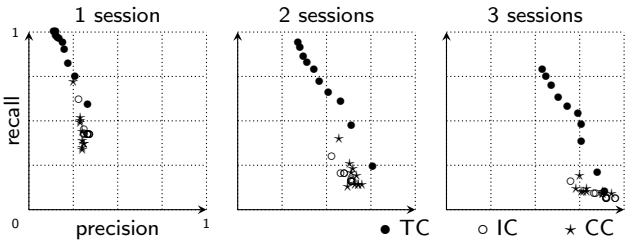

Figure 2. Precision and recall of time, interaction and change coupling.

about the relative intensities of interesting–neighbour couplings. This matches the scenario of a developer who needs to compare couplings with the entities (s)he's working on, while accommodating the constraint of partial information. At various points along the development history, we thus measured the coupling using only a few recent sessions, to compare it with the coupling measured using all past sessions at that point. If both couplings agree, it means that the measure quickly approximates the final coupling from limited data. Of course, a limited number of sessions will only concern a subset of all entities, so the coupling approximation can only be correct for this subset of entities.

Case Studies. We compared the coupling measures over the change histories of two projects. The first of those is SpyWare, the tool suite we have built in the recent years. For this study we selected the first two years of development of SpyWare, representing 360 sessions of development for a total of more than 16,000 changes across several hundred classes. The total code size of the prototype was around 20,000 lines of code at the time it was measured. The second project is Software Animator, a 5 kLOC system written in Java over 134 sessions and a period of three months, obtained via the Eclipse version of our plugin.

Prediction Results. We ran the prediction algorithm for TC, IC and CC, the expected set being computed with LC. We ran it with one, two or three sessions of information. We selected candidates using thresholds from 0.5 to 0.95, with a 0.05 increment. Figure 2 shows the results for SpyWare.

Increasing the threshold increases the precision and decreases the recall. Requiring more sessions of information also has the same effect. Time Coupling has far more predictive power than the other measures we defined, especially if considering that recall is more important than precision in practice —developers will ignore false positives easily but need the measure to identify as many coupled entities as possible. Time Coupling tends to be less precise but with much higher recall than the other measures.

To formally elect the best coupling measures, we combine precision and recall into the F-measure, defined as their weighted harmonic mean:

$$F_\beta = \frac{(1+\beta^2)\cdot P \cdot R}{\beta^2 \cdot P + R}$$

Common variations from $\beta = 1$ give a stronger weight to precision ($\beta = 0.5$), or to recall ($\beta = 2$). The coupling measures with the best F values are shown in Table 2. Time-based coupling is still the clear winner here, even according to $F_{0.5}$ in SpyWare.

		1 session	2 sessions	3 sessions
SpyWare	$F_{0.5}$	TC (0.95)	TC (0.90)	TC (0.75)
	F	TC (0.95)	TC (0.85)	TC (0.60)
	F_2	TC (0.90)	TC (0.60)	TC (0.50)
Software Animator	$F_{0.5}$	CC (0.60)	CC (0.55)	TC (0.50)
	F	CC (0.55)	TC (0.50)	TC (0.50)
	F_2	TC (0.50)	TC (0.50)	TC (0.50)

Table 2. Best coupling prediction according to the F-measure variants.

Convergence Results. Table 3 shows the agreement between coupling computed only from the three most recent sessions, and the same measure computed from all sessions back to the start of the history. The agreement values range from 1 (same entities, same order) down to -1 (same entities, reverse order). Zero agreement means there were no common entities or that their order was not correlated. Here too, time-based coupling has higher agreement on average.

Coupling Measure	SpyWare	Software Animator
Logical (LC)	0.33	0.31
Change (CC)	0.27	0.14
Interaction (IC)	0.34	0.30
Time-based (TC)	**0.79**	**0.83**

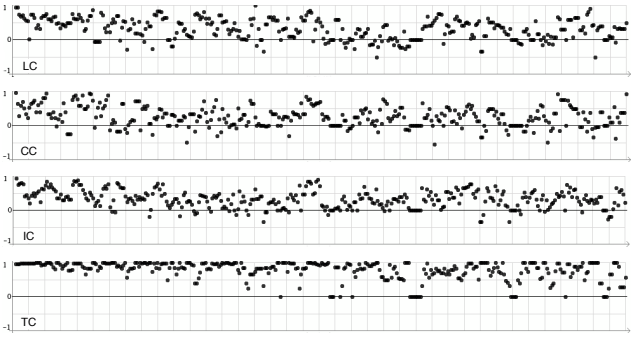

Table 3. Average convergence agreements, and graphs for SpyWare.

6 Discussion

Number of Systems. Our tests were performed on only two systems of moderate size. Still, we can somewhat generalize our results because they cover two different IDEs and languages.

Recording. Our fine-grained coupling measures requires recording the change history of systems from the IDE. It can thus not be applied if that information has been lost. This limits the short-term applicability of the approach to new systems.

Precision. The disadvantages mentioned above are offset by the improved accuracy of the measurement. When more accurate information is taken into account, the logical coupling is more stable, and can thus be used earlier on to make predictions. SCM logical coupling is often used for retrospective analyses when the history is considerable. We provided initial evidence that more detailed measures provide useful results earlier.

7 Conclusion and Future Work

SCM logical coupling has shortcomings due to the lack of accurate information found in these archives. This forces SCM logical coupling to give the same importance to each entity changed in a given transaction. To have more precise data, a larger history is needed. By using fine-grained information recorded in the IDE, we were able to measure logical coupling with more precision, effectively giving a different weight to the relationships between entities in the same session.

We defined several distinct logical couplings measuring different aspects of the fine-grained change information and performed a comparative evaluation of several logical coupling measures. We showed that using data recorded in the IDE allows one to predict logical coupling relationships with a lesser amount of history. In particular, we found that the most stable measure among the ones we tested (*i.e.,* the one which gives the most accurate measure with the least amount of data), is the Time Coupling.

This work can be extended by comparing our measures with other ones like static, conceptual, or dynamic coupling; we could merge coupling values obtained from short-term data with navigation information to predict changes or assist IDE navigation.

Acknowledgments. We gratefully acknowledge the financial support of the Swiss National Science foundation for the project "REBASE" (SNF Project No. 115990).

References

[1] E. Arisholm, L. C. Briand, and A. Foyen. Dynamic coupling measurement for object-oriented software. *IEEE Transactions on Software Engineering*, 30(8):491–506, 2004.

[2] L. C. Briand, J. W. Daly, and J. K. Wüst. A Unified Framework for Coupling Measurement in Object-Oriented Systems. *IEEE Transactions on Software Engineering*, 25(1):91–121, 1999.

[3] M. D'Ambros and M. Lanza. Reverse engineering with logical coupling. In *Working Conference on Reverse Engineering (WCRE)*, pages 189 – 198, 2006.

[4] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *International Conference on Software Maintenance (ICSM)*, pages 190–198. IEEE Computer Society Press, 1998.

[5] H. Gall, M. Jazayeri, and J. Krajewski. CVS release history data for detecting logical couplings. In *International Workshop on Principles of Software Evolution (IWPSE)*, pages 13–23. IEEE Computer Society Press, 2003.

[6] M. Pinzger. *ArchView — Analyzing Evolutionary Aspects of Complex Software Systems*. PhD thesis, Vienna University of Technology, 2005.

[7] D. Poshyvanyk and A. Marcus. The conceptual coupling metrics for object-oriented systems. In *International Conference on Software Maintenance (ICSM)*, pages 469–478. IEEE Computer Society Press, 2006.

[8] R. Robbes. Mining a change-based software repository. In *International Workshop on Mining Software Repositories (MSR)*, page 15. ACM Press, 2007.

[9] R. Robbes and M. Lanza. Versioning systems for evolution research. In *International Workshop on Principles of Software Evolution (IWPSE)*, pages 155–164. IEEE Computer Society Press, 2005.

[10] R. Robbes and M. Lanza. A change-based approach to software evolution. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 166:93–109, Jan. 2007.

[11] R. Robbes and M. Lanza. Spyware: A change-aware development toolset. In *International Conference in Software Engineering (ICSE)*, pages 847–850. ACM Press, 2008.

[12] C. Spearman. The proof and measurement of association between two things. *American Journal of Psychology*, (15):72–101, 1904.

[13] C. van Rijsbergen. *Information Retrieval*. Butterworth, 2nd edition, 1979.

[14] A. Ying, G. Murphy, R. Ng, and M. Chu-Carroll. Predicting source code changes by mining change history. *Transactions on Software Engineering*, 30(9):573–586, 2004.

[15] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. In *International Conference on Software Engineering (ICSE)*, pages 563–572. IEEE Computer Society Press, 2004.

[16] L. Zou, M. W. Godfrey, and A. E. Hassan. Detecting interaction coupling from task interaction histories. In *International Conference on Program Comprehension (ICPC)*, pages 135–144, 2007.