Supporting Architecture Evolution by Mining Software Repositories

Vanya, A.

2012

document version
Publisher's PDF, also known as Version of record

Approximating Change Sets at Philips Healthcare

2

A single development task such as solving a bug or implementing a new feature often involves changing a number of entities, also known together as a change set. Change sets can be approximated from the version control system. They are then used by the architects and developers to take important decisions. So change sets need to be approximated carefully. It is common to assume that two entities checked-in less than a small time interval from each other, and having the same meta-data associated with them, belong to the same transaction. Transactions may be good approximations of change sets if developers commit change sets in one go and if the required meta-data is available. This is however not the case in the industrial environment (Philips Healthcare) we study. This chapter presents a case study in which we investigated how change sets can be approximated in an environment with a complex workflow and limited meta-data in the version repositories. By doing so, we execute Step 1 and Step 2 of the process described in Section 1.7 to help the software architect. We found that, dependent on the commit practices used, a suitable time intervals between check-in timestamps of files has to be determined and leveraged to reliably approximate change sets.

2.1 Introduction

A single development task such as resolving a bug or implementing a new feature often involves changing a number of files in a series of changes in the software system. Together, these series of changes associated with a single task are referred to as a *change set*. Knowledge about such change sets help architects and developers in evolving a software system. For instance, it may help architects assess the decomposition of a software system and analyze the dependencies between different parts of that decomposition to initiate any necessary restructuring activities. But such decisions can be costly and have far-reaching consequences, so it is crucial that they are taken with care and are based on reliable change set data. Actions based on grossly inaccurate change sets may result in undesirable consequences that may be costly to fix.

Changes to software systems are captured in version control systems like CVS, SVN, or IBM ClearCase. Previous research such as that by Zimmerman and Weißgerber [2004] and Fis-

Parts of this chapter has been published as:

Vanya, A., Premraj, R., and van Vliet, H. Approximating Change Sets at Philips Healthcare: A Case Study. In *15^{th} European Conference on Software Maintenance* (CSMR '11), pages 121–130, IEEE Computer Society, 2011.

cher et al. [2003b] proposed methods to process meta-data from such version control systems to extract sets of changes made to the software system. However, their research has focussed on leveraging available meta-data (developers' names, commit messages, and timestamps) to extract individual *transactions*. Transactions, also commonly known as *commits* [Alali et al., 2008], are different from change sets in that they only capture the set of new or modified files that were *checked-in* together into the version control system. Change sets, on the other hand, refer to a set of transactions that together represent all changes to the software system to complete a specific development task. Indeed, it is quite possible that a single transaction may also be the change set for a task. The availability of the set of transactions inferred from version control systems has spawned several new research initiatives such guiding further changes [Zimmermann et al., 2005], improving programmer productivity [Kerstin and Murphy, 2006], studying team development culture [Weissgerber et al., 2007], and the like.

A similar area of interest at Philips Healthcare in the Netherlands (our case-study environment) is improving the control over the evolution of their software systems by studying past changes. At Philips Healthcare, software architects are transitioning their systems such that development tasks can be completed with as few changes as possible to files across different parts of the system decomposition (e.g. across different sub-systems). Such organisation is expected to help them manage their development processes better, increase efficiency, and cut costs. A starting point to initiate the planned transition is to study and audit the composition of change sets to date and identify the causes behind cross sub-system changes. Architects can then consider maintenance activities (such as moving files from one subsystem to another) to better facilitate the evolution of the system [Vanya et al., 2010].

The nature of development processes at Philips Healthcare, and the task at hand (analyse past changes to their system) requires that change sets, and not individual transactions, be studied to get a reliable picture of the past changes (more details in Section 2.3). While state-of-the-art version management systems (such as IBM Synergy) can be used to recover change sets directly, which in turn can be queried and analysed, costs associated with switching to them have daunted Philips Healthcare, like many other organisations, to upgrade. Thus, change sets have to be approximated from the version control system in use.

In this chapter, we present a case study conducted at Phillips Healthcare to support them in recovering approximated change sets from their version control systems. Our work differs fundamentally from that of Zimmermann and Weißgerber in the following two ways:

- While Zimmermann and Weißgerber focussed on recovering transactions, our goal is to recover change sets. These change sets most often comprise several transactions that are separated by varying gaps of time (Section 2.4).
- Furthermore, developers at Philips Healthcare rarely explain the rationale behind changes by adding commit messages to their transactions. So, in our case, the change sets had to be approximated without the use of commit messages as compared to Zimmermann and Weißgerber's approach that heavily relied on these messages.

As mentioned before, it is crucial that the approximated change sets are as correct as possible in order to reliably use them in decision making. Hence, in addition to presenting details

on inferring change sets with limited meta-data from our study environment, we also present the methods we used to evaluate the approximated change sets and gauge their accuracy (Section 2.5). Our evaluation setup is fairly general in that it can be adapted with little effort to other study environments to measure the accuracy of approximated change sets.

Thus, through this work, we make the following contributions:

1. Present a method used in our case-study environment to recover approximated change sets (which are different from transactions) from version control systems, that too with limited availability of meta-data.

2. Present an evaluation setup to assess the accuracy of approximated change sets in the environment that can be adapted and reused in other environments for the same purpose.

The results of our investigations show that it is possible to approximate change sets with reasonable accuracy even in environments with limited meta-data. We discuss our results in detail (Section 2.6), then present the threats to validity (Section 2.7), and bring this chapter to a close with conclusions and consequences of our work (Section 2.8).

2.2 Related Work

2.2.1 Change Set Definitions

Reviewing earlier work makes it clear that the phrase "change set" is rather overloaded. The definitions provided have in common that they describe a set of changes, but in which way the changes are related and what the changed entities actually are makes quite a difference in those definitions.

A considerable portion of the related work defines change sets as all the files that have been checked-in together into the version repository [Moser et al., 2008, Kagdi et al., 2007a,b]. Rastkar and Murphy [2009] extend this definition by requiring that all the files checked-in together should belong to the same development task. We have experienced that in practice, unless a strict development process is applied, developers often check-in file modifications together which relate to more than a single development task.

Hassan and Holt [2004] leave the technicality of commits aside and define change sets as all the entities which have to be modified because of a single development task. Such a change set may come from a single commit but it does not necessarily do so. The authors define a development task to be a new feature addition, feature enhancement or a bug fix.

According to McNair et al. [2007] a change set comprises logical commits. A logical commit is understood to be all the files modified because of a single modification request (MR). Change sets are then defined as any set of logical commits developers or architects consider to be logically related.

Estublier et al. [2005a] describe change sets as the first class entities in advanced versioning. It is emphasized that although a change set may contain any set of changes, a change set is found to be practically applicable when it is associated with a feature or task.

In this chapter we will refer to change sets as defined by Hassan and Holt. The reason to do so is that we cannot safely assume the commits of developers coincide with the submission of task implementations. Also, we believe that change sets as defined by Hassan and Holt are the ideal input for many evolution-related work. Our decision is further supported by the observations of Estublier et al. [2005a].

2.2.2 Change Set Approximation

When change sets are not directly available from the version control system applied then they have to be approximated. According to our best knowledge, so far only commits have been approximated based on the meta-data stored in version management systems. The seminal work on such approximations include the work of Zimmermann and Weißgerber [2004] and Gall et al. [2003]. They refer to such approximated commits as *transactions*.

In case developers check-in all the modifications together which relate to a single development task, transactions may suffice to approximate change sets. However this is not always the case (see Section 2.3).

2.2.3 Using Transactions

As stated, given a strict development process transactions can be considered as approximated change sets. Predicting future changes is one of the primary applications of such transactions [Zimmermann et al., 2005, Hassan and Holt, 2004]. Kagdi and Maletic [2007a] combine the evolutionary dependencies derived from transactions with static dependency couplings to further improve the prediction of future changes. Shiraban et al. [2003] predict changes by using artificial intelligence techniques on the transactions, whereas in [Zimmermann et al., 2005] and in [Ying et al., 2004] rule mining is applied on the same type of input.

Further, transactions have been used to:

- identify sub-systems that fulfill the requirement of independent evolution [Beyer and Noack, 2005]
- find strong inter-module dependencies [Ratzinger et al., 2005a, Gall et al., 2003]
- identify sets of co-changed files [Vanya et al., 2008, Antoniol et al., 2005]
- relate software entities and non-program artifacts and find traceability links [Zimmermann et al., 2003, Zimmermann and Weißgerber, 2004, Kagdi et al., 2007c]
- identify cross cutting concern candidates [Breu and Zimmermann, 2006]

Transactions, as we see, are used for several reasons by many earlier works. In general, these studies are using transactions to derive which entities changed together and leverage this information to support the decisions of architects and developers. In contrast, we focus on recovering change sets in our case study environment.

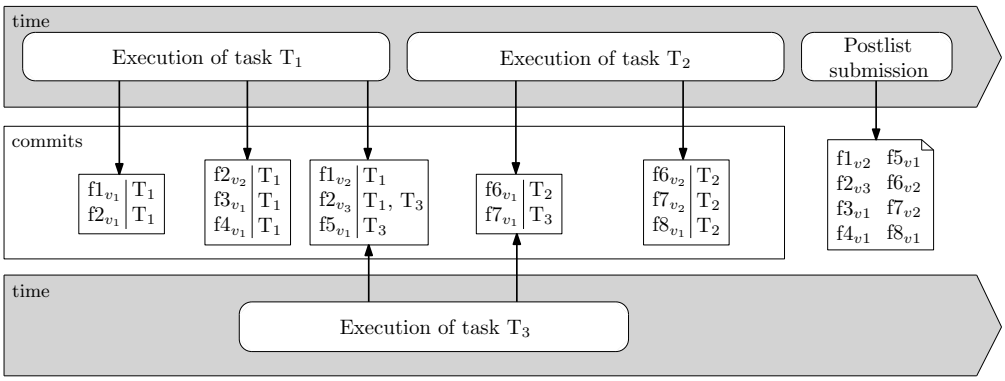

Figure 2.1: An example process instance executed by a developer

2.3 Case-Study Environment

In order to approximate the change sets, familiarity with the workflow in the case study environment is crucial. The manner in which changes are recorded depends on the software development process followed in the company. In the case of Philips Healthcare, the following practices are commonly followed in their the development process:

- During the execution of a single development task, developers typically commit files more than once.
- Developers do not need to make sure that each of their commits results in a buildable system.
- From time to time, developers work on more than one development task, especially in the problem report (PR) resolution phase of the software process. In such cases, a commit can concern more than one development task.
- After the execution of a few development tasks the summary of the modifications (a postlist) is sent to the integrator who checks if the system can be built with the modifications introduced.
- Especially when a complex feature is developed, several developers are working on the same development task.
- Developers are not obliged to annotate commits.

Figure 2.1 depicts a scenario where a developer executed three development tasks and submitted the related modifications to the system integrator (in form of a so called postlist). As we can see, during the execution of those development tasks files f1–f8 were committed.

In each commit Figure 2.1 indicates the files changed, the versions of those files and the id of the development tasks which served as a reason to modify those files. In the third commit, for instance, $f2_{v3} \mid T_1,T_3$ indicates that version 3 of file f2 was changed and committed because of development tasks T_1 and T_3. In the following paragraphs we elaborate further on some of the process characteristics listed. Note that the tasks (T_1,T_2, and T_3) presented in the figure are for illustrative purposes only and are not available in the version repository.

In the work culture at Philips Healthcare, software developers most often commit change sets to the repository via a series of intermediate commits with smaller changes rather than a single and complete commit. In Figure 2.1 we can see, for instance, that related to task T_1 three commits were made.

Many reasons prompt developers to complete change sets across a series of commits. One such reason that we identified by communicating with the developers is that their work is often disrupted by meetings, breaks, etc. and there is a general preference to commit all available changes before any such break to ensure against data loss. Another important reason identified is time to market pressure. Development tasks are parallelized in the company as much as possible. This encourages committing smaller iterations to maintain the workflow in the company. For instance, a developer may initially commit the interface of a component so as to enable development of other dependent components in parallel. Commits corresponding to the actual implementation of the component follow later. As a consequence, the complete set of changes to complete a single task may be separated over a number of commits. In the case of embedded software systems, the time interval between such commits may be much more than a few minutes because the life-cycle of some such changes involves hardware changes as well (which may take up to a month).

Furthermore, developers at Philips Healthcare are not obliged to annotate their commits; many of the commit messages in the version repository are left blank. As a result, it is often not possible to identify the motivations behind the check-ins by solely examining the repository. The only meta-data available for every check-in are at the file-level and include the developer who committed the change and the respective time-stamp.

Commits at Philips Healthcare may not only capture a part of a change set as described, but there are cases when the files modified because of more than one development task are committed together. The reason for inhomogeneous commits is that developers from time to time work on more than one development task in parallel. Working on more than one development task at the same time is done typically when the sets of files to be modified related to the developments tasks overlap. In Figure 2.1 we can see that the third commit (from left to right) is related to files which were modified because of tasks T_1 and T_3. There, file f2 was modified for both tasks.

As described, commits can be both incomplete and heterogenous with respect to change sets. This, however, does not introduce a problem when maintaining the consistency of the code. The reason is that the results of commits are not directly used to build and test the software system. The results of commits from one developer are not even directly accessible to other developers. Instead, typically after a series of commits a developer fills out a simple form where he indicates which files he modified and which the latest versions of those files are.

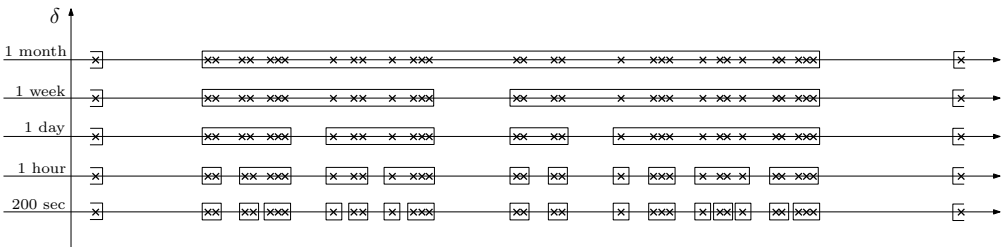

Figure 2.2: Change sets approximated with different time interval. A cross denotes a file checked-in to the version repository by a developer *D*

Such forms are also known in Philips Healthcare as *postlists*. In Figure 2.1 we can see that a postlist is created with all the latest versions of the files modified related to development tasks T_1, T_2 and T_3. Postlists are then sent to integrators who build and test the software system with the modifications in the postlist. If the system can be built and successfully tested then the modifications in the postlist will be consolidated. It mainly means that the modifications introduced to the files in the postlist will be made available to all developers. Postlists are further described in Section 2.5.2.

2.4 Approximating Change Sets

The most commonly adopted approach to approximate which files were checked-in together into a CVS-like version repository was proposed by Zimmermann and Weißgerber [2004]. They regarded files that were checked-in by the same *developer*, bore the same *commit message*, and updated in the repository within a *time interval* of 200 seconds from each other as a single *transaction*, i.e. an approximation of the result of a commit. The 200 second threshold was motivated by network latency arguments.

However, at Philips Healthcare, a single such transaction is unlikely to represent a complete change set because of their workflow as described in the previous section. Hence, Zimmermann and Weißgerber's approach was inapplicable to our environment since we are interested in change sets, and that too with no access to commit messages.

We hence adopted a different approach. Due to the lack of commit messages, we considered files checked in by a single developer within five different time intervals as a change set. These time intervals were chosen after consultation with the developers at the company and were selected as follows: 200 seconds, 1 hour, 1 day, 1 week, and 1 month. The longer time intervals are reflective of the workflow at Philips where change sets may spread across sometimes up to a month. Our task is to now investigate which one of the five time intervals can be used to approximate change sets as correctly as possible.

Our approach of approximating change sets is illustrated in Figure 2.2. On the time axis, a cross denotes a file checked-in to the version repository by developer *D*. Files that are checked-in to the repository within a set time interval (δ) are grouped together and considered as an

Table 2.1: The Approximated Change Sets (ACS)

		#Check-ins per ACS			
δ	#ACSs	Min.	Max.	Med.	Avg.
200 sec	115487	1	14002	2	8
1 hour	82571	1	14551	2	11
1 day	42447	1	14551	4	22
1 week	13568	1	19404	9	69
1 month	3408	1	27502	27	275

approximated change set. Figure 2.2 indicates approximated change sets identified with the five different δ values that we experimented with. As we can see, change sets approximated with a higher value of δ are often a union of change sets approximated with a lower δ value.

The number of approximated change set using each δ values is given in Table 2.1. As expected, the number of approximated change sets decreases for larger values of δ, while the average number of changes files included in the change set increases. From Table 2.1 we can also infer that certain change sets very likely do not refer to a single development task (e.g. the ones having around 14000 check-ins that may be results of merges).

But the approximated changes in Table 2.1 must be validated to be able to use them reliably for decision making. In the following section, we estimate the accuracy of the approximated change sets and investigate which time interval delivered change sets closest to the actual change sets.

2.5 Evaluation of Change Sets

After having approximated the change sets from the repository, we now perform an evaluation to judge their accuracy, i.e., how similar are the approximated change sets to the actual change sets. At Philips Healthcare, we had two options to conduct such an evaluation — these are presented below. We then reflect on the results from both evaluations to determine a suitable time interval.

2.5.1 Cross-checking with developers

The first option to evaluate the integrity of the approximated change sets is to check with the developers at Philips Healthcare. Given their experience and knowledge about the system, they are likely to be able to recall the development tasks performed and the associated changes.

We performed this evaluation by asking selected developers to participate in an on-line survey (screenshot in Figure 2.3). In the survey, developers were presented with the set of files (branch information and absolute file names) from an approximated change set. Those files were checked-in by them to the version repository. The task of the developers was to

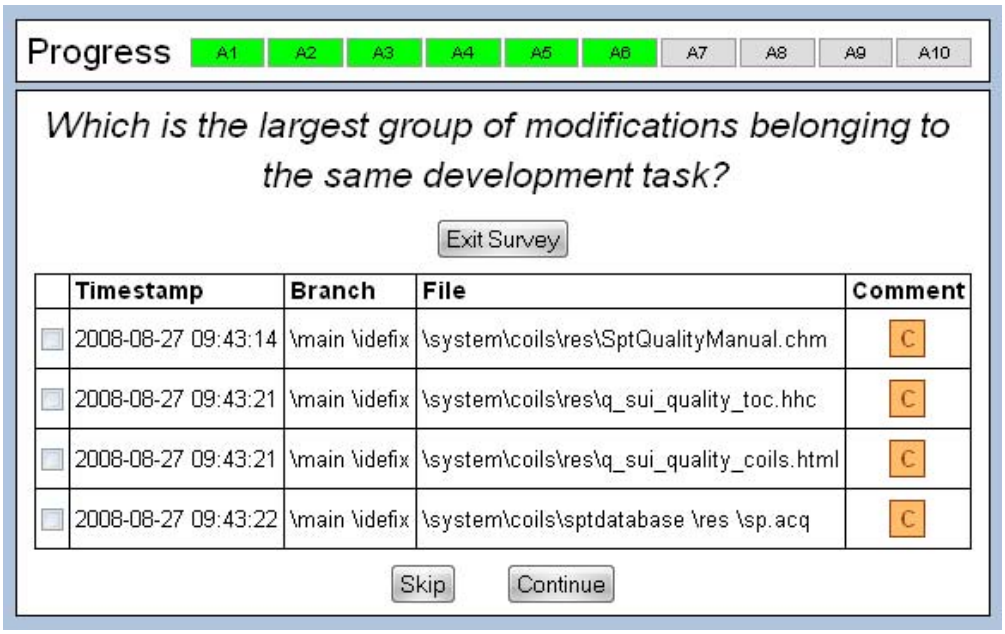

Figure 2.3: Screenshot of the on-line survey to evaluate change sets

recall which files were indeed modified for the same development task and mark them using the check-box on the left most column for each row. In order to provide some assistance to the developers in recalling the change sets, we provided the timestamps for when the files were checked-in to the system and a commit message (comment column) if available that summarized the rationale behind the change. Recall that in our study environment, such comments were extremely rare and hence were seldom provided in the survey. We also ensured that the change sets presented to the developers were from the recent past such that developers could more reliably draw from their memory to identify the change sets.

At the beginning of the survey, the survey participants were provided with a description of the purpose of the survey and explicit instructions including a demonstration on how to use the on-line form. In addition, we also made clear to the participants our definition of a change set and how the approximated change sets were collected. However, in order to reduce chances of bias on part of the respondents, we did not indicate the time interval used to approximate the presents change set for evaluation. The participants were then presented a series of approximated change sets one after another and they were required to mark the largest set of files that they recall being changed together for a specific task. The presented change sets were randomly selected from a single pool that contained change sets committed by the respective participant and approximated using the different time intervals: 200 seconds, 1 hour, 1 day, 1 week, and 1 month. The change sets from the pool were randomly presented one after

Table 2.2: Precision estimated with help of developers

Time interval	Precision (in %)			#ACS*	
(δ)	Max.	Min.	Avg.	Analyzed	Skipped
200 seconds	100	50	91	19	3
1 hour	100	33	91	15	4
1 day	100	40	78	21	4
1 week	100	6	66	14	7
1 month	100	2	36	6	8

*ACS stands for Approximated Change Sets

another until 10 approximated change sets were evaluated. Respondents were made aware of the option to skip questions pertaining to change sets that they were not certain about and also that they could exit the survey before answering 10 questions. The progress bar on top of the survey indicated the number of answered questions (Figure 2.3).

Considering that the data in the version repository spanned across nine years, the main criterion to select developers and invite them to participate in our survey was that they are currently working in the company. We narrowed the list of developers to invite by looking at their level of activity, which was measured as the number of check-ins made in the year 2009 (from January to May). We identified 25 developers with the highest number of check-ins ranging from 7,300 to 441 during the selected time period (see Figure 2.4). In the figure, the *x-axis* represents the 25 anonymized developers and the *y-axis* represents the number of check-ins made by them. The top-4 developers immediately stand-out in the figure owing to the marked high number of check-ins made by them. An investigation into the role of these four developers revealed that they were often given the task to merge one version branch to another that resulted in many check-ins. Since such merges are of little interest to our goal of approximating change sets, we chose to approach the next ten most active developers indicated in the figure.

The data recorded from the survey were the number of files presented to the developer as an approximated change set and the number of files selected by the developer. Eight of the ten invited developers responded to our survey. All but one developer who took the survey analyzed ten approximated change sets. The only exception analyzed five sets before exiting the survey. Altogether, developers evaluated 75 approximated change sets and skipped 26.

Results from the survey are presented in Table 2.2. We compute the accuracy of our approximated change sets as the ratio of files checked by developers in a presented change set and the total number of files in the change set. In terms of information retrieval, this is equivalent to measuring the precision. For each time interval, we note the minimum, maximum, and average precision and report them in Table 2.2. Additionally, we also note the number of analyzes and skipped change sets for each time interval. The following observations can be made from the table:

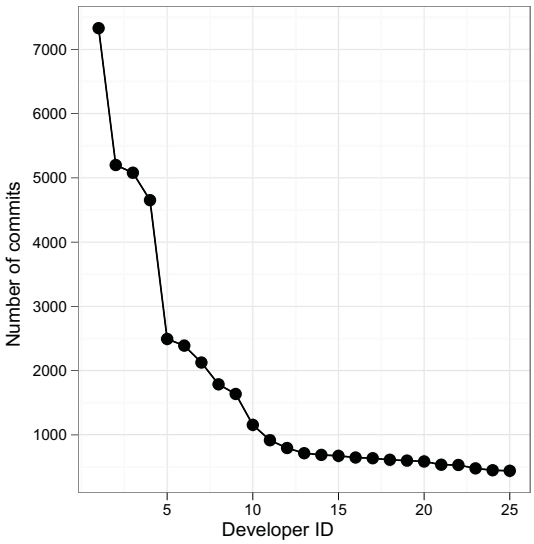

Figure 2.4: Number of check-ins made by most active developers at Philips Healthcare. We invited Developers 5–14 to evaluate the approximated change sets.

- The change set approximations inferred from time intervals 200 seconds and 1 hour have high precision. Average precision decreases with increasing intervals.

- Given that the precision for both time intervals 200 seconds and 1 hour are the same, change sets inferred using the hour interval are more useful because they capture a larger set of files relevant to the task.

- It is noteworthy that despite the low average precision for the 1 month time interval, in selected cases change sets are still correctly approximated (max. precision observed was 100%). This suggests that the life cycles of some of the development tasks are indeed very long.

At this stage, it is important to note the purpose of solely focussing on precision values for this evaluation. Recall that architects at Philips Healthcare are interested in reducing cases where changes to accomplish a task are made to files across different sub-systems. So it is more important that precision of the approximated change sets is high in comparison to higher values of recall. Approximated change sets with higher precision allows architects to reliably identify cross sub-system changes that can then be examined. Increasing recall will pollute the change sets with changes made for several different tasks and be detrimental to any further analysis. To add to this, to reliably compute recall values, it is important to know the complete

```
            Unique ID | <POSTLIST_NAME>nly95872_RFAmpSim_20060103T150114
     Stream information | <DEVSTREAM>FEMAIN
              Developer | <USER>Anna
                   Date | <DATE_DD_MMM_YYYY>3 JAN 2006
Is it a problem report? | <PR_SECTION>Y
  Problem report number | <SOLVED_PR>MR00035599
Rationale behind change | <REASON_TEXT>Improve simulation of the RF-Amplifier
                        | <CODING_STANDARD>N
                        | <PNS_SAR>N
     Developers playing | <BBLOCK_OWNER>James
          a role in the | <REVIEWER>Robert
         review process | <TEAMLEADER>David
                        | <DOC_SECTION>N
Changed files submitted | <POSTLIST_FILE>\path.to.file.one\BDRFAmplifierNorf.h@@\main\femain\5
             for review | <POSTLIST_FILE>\path.to.file.two\bdtransmittercsinterfaces.hcf@@\main\3
                        | <POSTLIST_FILE>\path.to.file.three\BDRFAmplifierNorf.cpp@@\main\femain\5
                        | <TEST_SECTION>Y
                        | <TEST_DONE>@OTM6
 Last versions of files | <PREVIOUSLY_CONSOLIDATED_FILE>\path.to.file.one\BDRFAmplifierNorf.h@@\main\femain\2
   used to build system | <PREVIOUSLY_CONSOLIDATED_FILE>\path.to.file.two\bdtransmittercsinterfaces.hcf@@\main\1
                        | <PREVIOUSLY_CONSOLIDATED_FILE>\path.to.file.three\BDRFAmplifierNorf.cpp@@\main\femain\4
                        | <ACTION_POSTLIST>20060103T150117
                        | <ACTION_POST>20060103T150901
                        | <ACTION_TAKE>20060103T210839
                        | <ACTION_CONSOLIDATE>20060109T161257
                        | <SWID>29
```

Figure 2.5: A sample postlist presenting a change set.

change sets in advance and accurately, which is exactly the purpose of our case study in the first place.

2.5.2 Postlists

The second option available to us to evaluate the accuracy of the approximated change sets are postlists. They are messages manually created by developers to notify the system integrator that a certain set of files have been added or modified and are ready to be integrated into the system (see Figure 2.1). After adequate testing, the changes are approved and consolidated into the latest version of the system.

A sample postlist used at Philips Healthcare is presented in Figure 2.5. Lines that add no value in explaining the postlist are grayed out in the figure. At the top, the postlist contains basic meta-data such as a unique identifier, the name of the developer, date of the message, and whether there is problem report (akin to a bug report) filed that links the changes to the problem. In the meta-data, the developer are also encouraged to submit an explanation of the changes. It then lists the people in the team who will be involved in reviewing the changes. Next, the files that have been submitted for review are listed along with the complete absolute path, branch information, and the new version numbers of the files. Lastly, the postlist also

presents the existing version numbers of the files that are already consolidated in the system.

We would ideally expect these postlists to be the best resource to use to approximate change sets given their content. However, on inspection we found that these postlists were not change set specific, noisy and unreliable for the purpose. They often contained changes made to files for more than one task and provided no means to map the changed files to the respective tasks. Further, change sets often spread across several postlists similar to the culture of committing code iteratively to the version repository. Many postlists also contained generic reasons which could not be reliably mapped to any development task. Lastly, postlists do not capture the range of changes made to files but only the snapshots of the versions that are ready to be integrated. Hence, the change history in the postlists is likely to be incomplete. For instance, in Figure 2.5, version 5 of file `BDRFAmplifierNorf.h` has been submitted in the postlist, but the latest consolidated version of the same file is 2. These reasons ruled out the possibility of an extensive evaluation of the accuracy of approximated change sets by leveraging postlists.

However, we made an attempt to use some carefully selected postlists to evaluate the change sets. Keeping in mind the above cited challenges with postlists, we made sure that the selected postlists were indeed specific to one change set only. In order to select these change-set specific postlists, we collected a random sample of 100 postlists and discarded those that met one of the following criteria and replaced it with another postlist deemed fit (to keep the total number of postlists 100 for the evaluation):

- The reason field that the files in the postlist were modified because of multiple development tasks For instance: *"Fixed two problems: [...]"* and *"Add [...], Remove [...], Add [...]"*)
- The reason field indicated that a postlist came from a division of the company where they almost always group more change sets into a single postlist to reduce interaction costs. For instance: *"Cleveland delivery"*
- The name of the postlist or the reason field pointed out that the postlist was created only because of a merge activity. For instance: *"Merge of [...] to branch [...]"*
- The description in the reason field was too abstract. For instance: *"Update"*. Abstract reasons may be an indicator that a developer worked on multiple tasks and therefore he could not provide a single concrete reason.
- The postlist was created to submit only a partial solution of a development task. We knew that this was the case if the reason field contained descriptions like: *"First part for [...]"* and *"part 2"*.

While we meticulously examined the postlists and arrived at the final set of 100 for our evaluation, we realize that in a handful of cases postlists may refer to more than one change set. But this risk is minimal given our familiarity and experience with the system.

Recall that one would often see a difference greater than one between version numbers of a changed file and its previously consolidated version. We accounted for these missing file

Table 2.3: Precision estimated using postlists

Time interval	Precision (in %)		
(δ)	Max.	Min.	Avg.
200 seconds	100	50	93
1 hour	100	20	89
1 day	100	1	69
1 week	100	<1	31
1 month	95	<1	8

Table 2.4: Recall estimated using postlists

Time interval	Recall (in %)		
(δ)	Max.	Min.	Avg.
200 seconds	100	20	74
1 hour	100	20	84
1 day	100	22	92
1 week	100	24	94
1 month	100	25	94

versions by interpolating them in the postlists assuming that all intermediate changes to the file were related to the same development task.

Using each selected postlist, we examined the set of added or changed files and their version numbers. For each such version of a file, we queried which approximated change set contains that version. We considered the approximated change set with the largest overlap of file versions as the dominant change set and evaluated it.

By comparing the (dominant) approximated change set and the list of file versions from the postlist, we measured precision and recall values as measures of accuracy (presented in Tables 2.3 and 2.4 respectively). Precision was computed as the ratio of the number of file versions common to the two change sets and the number of file versions in the approximated change set. Recall was computed as the ratio of the number of file versions common to the two change sets and the number of file versions in the postlist. The following observations can be made from the tables:

- The precision values suggest that on average the approximated change sets with time intervals 200 seconds and 1 hour are precise (with nearly 90% or more).
- The 1 day time interval gives an acceptable level of precision. However, increasing time intervals seem to result in a substantial decrease in precision values.
- From the recall values in Table 2.4, we see that the 200 second time interval on average

missed one out of four files belong to a change set. Given the time needed to execute development tasks at the company, such high recall is a surprising result.

- The value of recall increased further with increasing time intervals, but this occurred at a cost of falling precision.

2.6 Discussion

In the previous section, we evaluated the approximated change sets using two different ways: cross-checking with developers and using postlists. Both evaluation measures have provided similar results.

We used five discreet time intervals for the change set approximation method which yielded five sets of approximated change sets. During the evaluation of the approximated change sets, it turns out that accuracy is optimal when the time interval is set to 1 hour. We expect, however, that using more granular time intervals to approximate change sets may result in more fine tuned results, but this remains to be explored in our study environment. Our motivation to use only five discreet values was to limit the effort spent on the evaluations. We would like to point out that the 1 hour interval is not a recommended universal time interval to approximate changes. Instead, we encourage researchers working in environments similar to ours to conduct a similar exercise to establish an optimal time interval for use.

2.7 Threats to Validity

When evaluating the accuracy of the approximated change sets, we assumed that developers have a good recall of their own change sets and that the postlists selected represent actual change sets. The results from our evaluation could have been affected if developers remembered their change sets incorrectly or the selected postlist did not represent single and complete change sets. We have done our best to mitigate such effects. We presented all meta-data available on the change sets approximated to the developers so that they could recall what belonged to the same development task. Postlists were carefully analyzed to select those that represent complete change sets for our use. The similarity in our results from both evaluation measures suggest that they are valid.

Our method approximated change sets with an implicit assumption that a single developer completed a development task. However, in cases where more developers worked on the task, we could recover only parts of the change set. In our study environment, however, we found no means to relate the modifications of developers who worked on the same development task. Such cases may effect the recall of our approximations. Architects at our study environment were fully aware of this limitation and were supportive of our approach to approximate change sets because precision of the approximations was far more important to them for further analysis than approximating complete change sets.

2.8 Conclusion and consequences

It is important to identify change sets for many evolution-related studies. Change sets (that are different from transactions) are used, for instance, to assess the evolvability of the architecture. Change sets need to be approximated if they are not available directly from the version control system applied. To do so properly, one needs to carefully take into account the development practices used.

We have conducted a case study at Philips Healthcare to approximate change sets from their version repository. Approximations had to be made amongst several constraints at the company, most severe one of them being lack of meta-data. Our approach comprised using only developer information and check-in timestamps of files to recover change sets. Five time intervals were experimented with to approximate the change sets, whose evaluations using developers and postlists resulted in reasonably high accuracy. The optimal time interval found in the study environment was 1 hour. Our evaluation has helped us choose the right set of change sets approximated from the version archives of the company, which has already performed successfully in previous evolution-related studies [Vanya et al., 2010].

Our research has shown that it is possible to approximate change sets even in environments where meta-data is limited. Furthermore, differences in accuracy of the approximations across different time intervals emphasizes the importance of evaluation. As mentioned earlier, decisions based on these change sets can be costly and have far-fetching consequences. Hence, an evaluation of the data is vital for further reliable results.