Similarity in Task Contexts

Adam Tait
University of Waterloo
atait at uwaterloo.ca

April 5th 2007

Abstract

I would like to acknowledge and thank Professor Gail Murphy (UBC) for her thoughts and guidance.

1 Introduction

In his famous essay, *No Silver Bullet*[1], Frederick Brooks states that software complexity is an essential property of all software entities. Complexity is so fundamental that many of the other classical problems in developing software products can be derived from it.

> Not only technical problems but management problems as well come from the complexity. This complexity makes overview hard, thus impeding conceptual integrity. It makes it hard to find and control all the loose ends. It creates the tremendous learning and understanding burden [1]

> software can be changed more easily - it is pure thought-stuff, infinitely malleable [1]

In *Using Task Context To Improve Programmer Productivity*[2], the concept of task context was introduced to reduce the complexity presented to a developer. The intent is to reduce the number of decisions required by a software developer to complete a given task, and thus improve their productivity. Task context is a powerful tool in itself, but it also opens the door for an entirely new field in software evolution research. Task context is a completely different approach to understanding software evolution; describing changes to the system in terms of the developers actions and effects.
Until recently, the study of software evolution was limited to analyzing change in the code base over time. The practice of building software systems can generally be reduced to the practice of building conceptual models. Software evolution can study how code changes with age, but the more important lessons are how the *design* changes. How can we understand the design, or semantic model, of a given system without understanding the conceptual model? How can we understand the evolution of the conceptual model without using the knowledge and understanding of those persons performing the evolution?

2 Task Context

This section will give a brief overview of a task context, specifically highlighting focuses of this study. For more details on task contexts, please refer to *Focusing Knowledge Work with Task Context* [3].
We will define a task to be *a usually assigned piece of work often to be finished within a certain time* [2]. A task is an atomic unit of action in a plan. In terms of software development, a task could be refactoring a module, or creating interaction between class A and class B, or fixing a software bug.
A task context is *the information, a graph of elements and relationships of program artifacts, that a programmer needs to know to complete that task* [2]. The information used to build a task context

comes from *Interaction History*, a time ordered series of interactions. *Interaction events* describe the interaction a developer has with the program artifacts. An example interaction event would be a developer editing a method, foo, in a class, bar.
The real power of the task context is in its application. The operation of *projecting* a task context onto a developers view of the system, is effectively reducing the software system entities to those required to complete the task. A projection is done by presenting only system artifacts of high interest to the current task, to the developer. In order to decide which artifacts are of high interest, the task context contains a *Degree Of Interest* algorithm. The degree of interest of a given program artifact is a real number based on the recency of developers interaction and the types of interactions with that artifact. Task context *slicing* can be used to build a subset of a set of interaction events, within a particular constraint. A slice could be used to focus on a particular type of event or events within a particular time, or events related to a specific artifact.
The *prediction* operation is used to find other entities within the system that could be considered of interest, but do not occur in the interaction history. A prediction could use a task context slice to find artifacts related by a constraint.
Task contexts support a structure called the *task activity context.* A task activity context is a meta task context that references the developers interactions with different tasks.

3 Research Goal

Previous work on task context has shown that it can be a very powerful productivity tool. Missing from their study is the captured information on developers behaviour. Implicitly, a task context uses information that the developer already has, or had previously, to help them in the future. Capturing information on developer behaviour begs the question, *how can we use this behavioural information?* Providing access of specific and detailed information about human thought to the research community should result in a multitude of different areas of study based on this information. To this day, there has been research towards abstracting meta information from software. The study of software modelling, for example, attempts to build a conceptual model of a software system. However, most of these approaches for extracting *semantic* information from software have been based entirely on analyzing or slicing the code for information.
In fact, the most current research on task contexts gives a subtle hint of building a semantic model of the given system, in order to apply it to the specific slice related to a task. The operation of prediction tries to anticipate what other software artifacts are required to the complete the task. At any given moment in the developers interaction with the software system, the task context should contain what artifacts or subset of the software is relevant to your last selection or edit. This implies that the task context is presenting a subset of the semantic model of the system.
The problem of building a semantic or conceptual model of a software system is a very large and broad topic. Instead, this study will try to lay groundwork for doing research in this area, including making some suggestions of future research, in the final section. This study will focus on a more specific question; *how can we use the developer behaviour information to improve upon the current idea of the task context?*. Even more specific, *how can we use this information to augment an existing task context or provide history to a new context?*

3.1 Augmenting a given task context with information from other contexts

The effect of augmenting a task context with other interaction events is part of the current task context model. The current model inserts indirect events into the interaction history, during operations such as prediction. These interaction events change the way the degree of interest algorithm is applied, resulting a different context. This model fails to take into account information already accumulated in other tasks. For good reason, as we will see it is difficult to select appropriate information based on any criteria.
Task context information from a task other than current task may still apply, even though the current task is unqiue. In order to apply existing contextual information, we must first determine how to objectively judge which tasks or information are similar enough that such information can be used. This study will focus on trying to decipher *how similar is similar* when it comes to a task context.

3.2 Similarity Meaning

What does it mean for a task to be similar? The first clear method of defining similar tasks is to classify tasks into different types. Tasks based on the phase in the development cycle; such as prototyping, design, development, integration, release, short term and long term maintenance. We can classfy tasks by their intended effect on the system; add functionality, remove functionality, refactor, redesign, fix a bug, introduce a bug. These sets of classifications may be distinct in their descriptions, but in practice, it would be very difficult to unambiguously classify a task. Luckily, we are looking for a very specific description of similarity between tasks.
When looking for similar tasks, or tasks that might produce similar contexts, it is logical to look at tasks that focus on similar entities within the system. Tasks that work with similar entities must share some semantic meaning because of the relationship those entities have with the rest of the system. Since it is the semantic meaning that we wish to present to the developer in the task context, this is exactly what we hope to extract and re-apply.
This approach makes the assumption that repetition in a developers behaviour implies a high correlation to entities of mutual interest, sharing some semantic relationship. I recognize that semantic relationships are not the only factor that influences a developer to refer to an entity repeatedly, however it is very likely of significance. As we will discuss in the final section, research in this field could better prove (or disprove) this theory.

3.3 Granularity of Similarity

This study will mostly look at similar on the scale of a full and completed task context. I hope to find that trends in an entire task contain some relevant and valuable semantic information. However, separating the interactions into tasks is not the only level of granularity that might contain semantic information. There might be interesting results by looking at interactions within a certain unit of time, such as a minute, an hour or a day. I believe that semantic information might be available on the granularity of a sequence of interactions.

3.4 How Similar is Similar?

The problem of deciding whether a result is correct or incorrect is a difficult one. If I was going to augment a context with a similar context, what is the most accurate augmentation? Logically, the most accurate augmentation would be the identical context. However, the identical context is not available. Therefore, the most accurate augmentation would be the common set of events with the highest and lowest degree of interest. The occurrance of the artifact in the common set is of more importance than having a similar degree of interest, or a different artifact with a similar degree of interest. Therefore, the degree of interest is not a highly correlated attribute. However, selecting artifacts of high dis-interest (or negative interest) in the original context probably have no correlation to a similar context. Thus, artifacts of high dis-interest should be removed from the common set.
Since the degree of interest of a given event decays with time, using common artifacts with relative degree of interests is not a good indicator. The degree of interest applies to a given task, at a given point in time, and does not translate to a larger, more general, context.

3.4.1 Measure of Similarity

This study is going to judge the strength of similarity between two task contexts based on the number of events in common. Similarity of task contexts can be defined as a real number between 0 and 1, representing the number of artifacts that the contexts share. Since the interaction history is a series of events indexed by time, developers may interact with an artifact then interact with it again at a later time, generating multiple events for the same artifact. However, the frequency of events is very developer specific and should have a lower correlation to a similar task than the unique occurrance of an artifact. This concept might be counter-intuitive because we earlier stated that relationships which occur frequently should be a highly significant source of semantic information. However, high

frequency within an individual task context has a high probability of being an outlying, or false positive, result. This would occur because of an abnormality in that particular task or developer working on the task. Therefore, diversity in similarity is required and an important reason to compare across task contexts.
To summarize, the measure of similarity used by this study will be the number of unique artifacts shared by two similar task contexts, and by association two interaction histories, represented as a percentage.

3.5 Previous Work

There has been some landmark research leading up to this study. Clearly, Gail Murphy and Mik Kersten's work on task contexts [2][3] is the basis for most of this paper.
Hipikat [5], another tool developed in part by Professor Gail Murphy, tries to build relationships between correlated information in a software development project. Based on the context of a query, Hipikat can give information from bug reports, source repository comments, documentation, new-groups, or other such related sources of project information. Hipikat is intended to help developers build a conceptual model of the system, though the results are only as good as the available information.
In *Who Should Fix This Bug?* [6], a machine learning approach is applied to the assignment of new bug requests entering the project. This solution is based on a text categorization of bug summaries. Once trained, this algorithm assigns new bugs in a system to the most appropriate resolver, possibly saving a significant burden on project management.
Strathcona [7] is another very powerful tool. Strathcona performs structural matching on source code to extra information that would relevant to a developer using the API provided by that system. This technology is used to algorithmically produce examples for uses of the API.
Clone detection is another very active area of research in software evolution. Clics [8], the clone interpretation and classification system, uses suffix trees to find code clones in a software system.

4 Process

This study compares two approaches to the decision problem of similarity between task contexts. The first approach is to look at the proportion of co-occurring software artifacts. The second approach compares the co-occurrence of sequences of interaction events.

4.1 Direct Approach

This approach directly implements the measure of similarity, described in the previous section. This approach measures the proportion of co-occurring artifacts by building a list of artifacts in each context. The number of similar artifacts between contexts is calculated by string comparison of artifact names in the artifact list. This approach makes no account for artifacts that occur in many events in the interaction history.

4.2 Creative Approach

This approach is not a direct implementation of the defined measure of similarity. The results from this approach will be compared against the results of the direct approach.
The idea for this approach was born of the concept that an interaction history could contain semantic information at a smaller granularity than a task context. This approach employs techniques from software cloning research to find clones of sequences of artifacts in the interaction history. The hypothesis is that interaction histories have a large number of similar sequences of events should be similar tasks. This approach is clearly similar to the direct approach; if tasks contain a high number of co-occurring artifacts, then it is likely that it also contains a significant number of similar sequences of events. However, this approach is clearly indicating that these similar sequences exist for a reason; the developer has a conceptual relationship between these artifacts.

4.3 Data

The results in this study are based on data from 87 task contexts completed while resolving bugs filed against the open source project, Mylar. Mylar is the tool that was built by the authors of *Using Task Context to Improve Programmer Productivity*[2] to release the power of task context upon the world software development community. Mylar is a plugin for the Eclipse IDE [4].
Prior to the first major release of Mylar, its developers used Mylar plugin while working on successive releases. They were able to store their completed task context data, and store it along with the Bugzilla bug reports. The data used in this study comes directly from the open source Bugzilla bug repository. Specifically, the 87 contexts used came from random bug tasks starting with Mylar version 0.6 until the 2.0 release.
There were several bugs that had been fixed and re-opened, producing multiple distinct contexts for the same bug. By definition, these task contexts must be similar and are used as a control. These were Mylar bugs numbered 150339, 153496, 165169, 165852, 166088, 168776, 173575 and 178496. Of particular interest, bug 166088 was resolved twice by two different developers. Bug 173575 was resolved four times.

5 Results

We will present the results from the direct approach, then employ those results to test the hypothesis of the creative approach.

5.1 How To Read The Results

The results are presented in the form of scatter plots for each threshold value tested. Each point on the graph represents a pair of contexts that were deemed to be similar. Each bug was assigned a unique integer, based on its order in the series of bugs used in the study. This integer is the same across all tests in the experiment, so bug 80 on all the scatter plots refers to the same bug.

5.2 Direct Approach

The threshold for each test presented was calculated by the percentage of artifacts, *in the smaller of the two contexts*, that were in common. This calculation was chosen because the smaller context was most logical reference between any two contexts. A context that is a subset of another context, satisfy our previous definition of task context similarity. This reference point resulted in an anomoly in the results. Those bug contexts that contained a relatively very small number of events had a higher impact than the those that did not. For example, the first graph representing 100 percent similarity has a significant number of bug contexts chosen as similar. Specifically, there are a number of contexts that are similar to bug 48 (refers to bug 165176). This context is relatively small, contain only about 9 unique artifacts in its events.

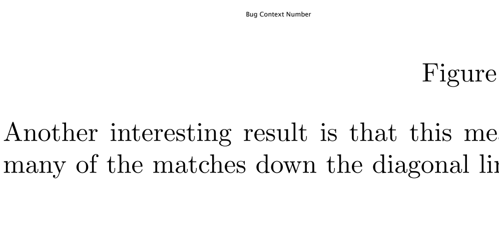

Figure 1: Co-occurrance of 1

Another interesting result is that this measure did catch our list of control contexts, as shown by many of the matches down the diagonal line. However, there is no variablility in this information, it

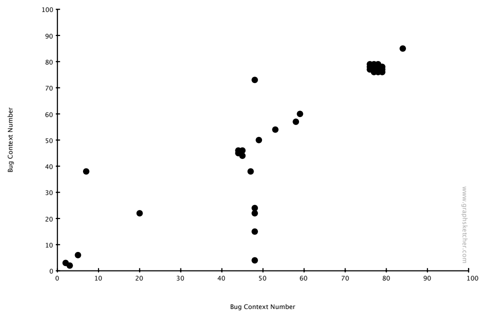

Figure 2: Co-occurrance of 0.9

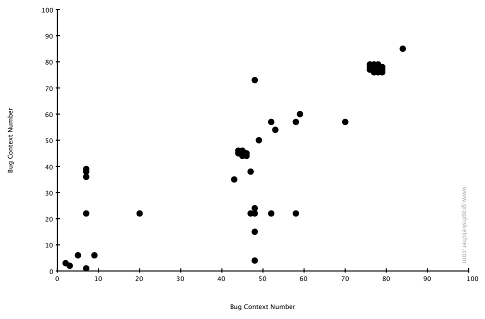

Figure 3: Co-occurrance of 0.8

only represents the data which is identical.
Also interesting to note that there were relatively few additional contexts found to be similar by lowering in the percentage of similar artifacts to 90 percent. However, lowering the threshold percent to 80 introduced nearly half as many new results; a significant increase. The results from the 80 percent threshold have enough variability that we will use it as a reference in measuring the quality of the creative approach.

5.3 Creative Approach

In the creative approach, we tested several string length thresholds and found that similar substrings of length 8 and 9 produced the greatest number of relevant hits.
It's clear that the creative approach produced different results from the direct approach. Using the direct approach with 80 percent threshold as the base, the precision of the both results was calculated. The precision of the first test (substrings of length 8) is close to 60 percent. The precision of the second test (substrings of length 9) is slightly higher, closer to 80 percent. The recall of the first test (substrings of length 8) was nearly the same as its precision, around 60. The recall of the second test (substrings of length 9) was significantly lower, around 60.
The precision and recall tell us that increasing the similar substring minimim length gives more accurate results based on the defined measure. As indicated by the high precision and reasonable recall of the second test, these results are significant.

5.4 Threats

There are several threats to the findings presented in this paper. First of all, the threshold chosen in both approaches seem arbitrary. To a certain extent they are because they were chosen to achieve high accuracy. However, they do present that the approaches used are significant. The data used comes from a very limited number of developers working on Mylar. In practice, most software systems are built and maintained by a consistent and closed set of developers. Thus, the results will be accurate applied across most modern software projects.

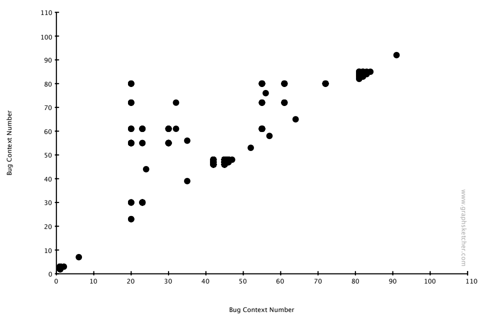

Figure 4: Similar Sequences of length 8 or greater

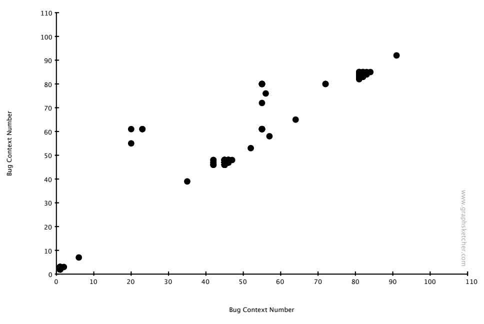

Figure 5: Similar Sequences of length 9 or greater

Mylar performs "task context compression", which compresses and removes unnecessary information from the task context in order to save on space. If this were true, then the data used in the study would be incomplete.
Are results more accurate with both direct and indirect interactions in the interaction history? Should the indirect interactions be removed? Should the data focus on selection and edit events alone?

6 Conclusions

This paper was intended to be just as much an inspiration to research in this field, as much as it presents substantial results. Nonetheless, the significance of the results from the creative approach indicate that it would be effective to augment a task context with information from other, similar contexts. The results also show that looking at information in interaction histories on a smaller granularity does produce a strong correlation to similarity in contained information.

7 Future Work

This paper presents some fundamental information, upon which much research needs to be done. I will highlight a few areas and studies that should be done in order to improve our understanding of this field.
In order for the technique to augmenting task contexts to be practical, we need more information on how similarities in the interaction history relate to descriptions of tasks. Since the english language is an inexact and consistently misinterpreted language, a machine learning approach to achieve an approximation might be the best direction.
Can we relate frequent similarities, of any granularity, in an interaction history to relationships within the code? Can we compare results from other studies of relationships in software to this approach?
Task context contains a history of user interactions with the system. This means that information about a developers behaviour is available. How can we retrieve this information and what value can we discover within it?

Can we construct a semantic model of a system, based on basic principles, to describe a given software system? Can we get accurate or reliable information from a developers interaction information?
This paper states that we should, in theory, be able to reconstruct a semantic model from information about a developers interaction with the code. This is one aspect where the study of software modelling and software evolution might be able to merge, or even collaborate!

8 References

[1] F. Brooks. *No Silver Bullet*

[2] M. Kersten and G. Murphy. *Using Task Context To Improve Programmer Productivity.* 2006

[3] M. Kersten. *Focusing Knowledge Work with Task Context.* 2007

[4] Eclipse Open Source Community and IBM. *Eclipse IDE. http://www.eclipse.org*

[5] D. Cubranic and G. Murphy. *Hipikat: Recommending Pertinent Software Development Artifacts.* 2003

[6] J. Anvik, L. Hiew, G. Murphy. *Who Should Fix This Bug?.* 2006

[7] R. Holmes, R. Walker, G. Murphy. *Approximate Structural Context Matching: An Approach to Recommend Relevant Examples.* 2006

[8] C. Kapser, M. Godfrey. *Improved Tool Support for the Investigation of Duplication in Software.* 2005