Direct Measurement and Forecasting of Software Team Efficiency

Steve Bannerman

April 17, 2007

Abstract

In general, software development managers do not know how efficiently their teams are changing their software products. They may know the ratio of their development cost to the size of the software created, or something similar, but they do not know the ratio of effective changes to all changes, over time. To address this void, we developed a set-theoretic model for efficiency. Further, we developed tools to measure efficiency across time for Java products. With these tools, we measured the changes for six open source projects, over a one year period. Based on these measurements, we proposed a family of curves - a mathematical model for software team efficiency. Next, we performed a forecasting tournament, trying to establish the model's usefulness for predicting the efficiency of particular changes. A hybrid forecasting method, combining our mathematical model and a "random walk", was able to predict efficiency values better than a random walk. Provisionally, we conclude that the model is valid and that software development managers can use it, in addition to the supporting tools, to measure and forecast their team's efficiency. With such an ability, for example, they could evaluate new development activities or estimate how long changes will be effective.

1 Introduction

Teams of people execute many (or most) substantive projects. Project managers are usually responsible for the successful completion of these projects. Presuming these managers wish to succeed, they will want their team to perform and improve [1]. That is, to do the right things (effectiveness) and to do those things right (efficiency) [2]. Common sense, and at least one survey [3], tell us that measuring these aspects of a team's performance is important - measuring them facilitates understanding and improvement. Further, understanding may ultimately facilitate forecasting, which can aid these managers as they allocate and coordinate resources to complete their projects.

A team of software developers, coordinating with a project manager, executes most software projects. Typically, their effectiveness and their efficiency are evaluated indirectly - that is, from information outside of the software product that they're developing. For example, their effectiveness might be determined by some combination of the time and cost for their project (or time and cost variance, as compared to planned values). Similarly, their efficiency might be determined as some ratio of software size to some combination of its time and/or cost.

However, we propose that a software team's effectiveness and efficiency can also be evaluated directly. At a high level, a software team modifies the software product via a sequence of changes, resulting in a sequence of software product versions. Those changes that persist from one version to another are effective changes - a measure of the team's effectiveness. And the ratio of those effective changes, to all of the changes necessary to bring them about, is a measure of the team's efficiency.

1.1 Effectiveness Literature

Hackman provided a popular "normative" model of group effectiveness [4]. At a high level, he structured that model such that a team is put together in the context of an organization, they expend effort as they execute a process together, utilizing their collective skills, knowledge, and available resources, to solve a problem or generate a product.

He suggested that we assess team effectiveness based on a team's actual output (the solution or product) in addition to the "state" of the team and the individuals in the team after the project. He recommended that the actual output, or the productive output, be evaluated by organization members or clients. Although such a "perceptual" evaluation is subjective, he recommended it (in part) because an exclusively objective evaluation would limit the applicability of his model to those projects which can and do collect reliable, objective data.

Researchers have used "perceptual" evaluations to determine the effectiveness of a variety of teams. Table 1 summarizes how some researchers have used these evaluation methods to evaluate the effectiveness of: engineering and science faculty teams [5], multi-disciplinary research teams at the NASA Langley Research Center [6], manufacturing teams [7], health care teams within the National Health Service of the United Kingdom [8], a committee serving children and families [9], and aid projects [10].

Although we can also use "perceptual" evaluations to determine the effectiveness of software development teams, some researchers have augmented

reference	*researcher*	*evaluation method(s)*
[5]	Adams	surveys
[6]	Waszak et. al	surveys
[7]	Doolen et. al	surveys, assessments
[8]	Borrill et. al	surveys, interviews, ratings
[9]	Grandinetti	meetings, interviews
[10]	Crawford, Bryce	ecological, social, and economic indicators

Table 1: Perceptual Effectiveness Evaluations

their evaluations with "more" objective measures such as: budget variance, schedule variance, conformance to specifications, quality, or modification requests [11, 12, 13, 14, 15].

Unfortunately, due to a potential multitude of differences between projects, teams, perceptions, and even the "objective" measures, comparing the effectiveness of one team with that of another has proven to be difficult [16].

1.2 Efficiency Literature

In general, we all understand efficiency to be the ratio of effective output to total input [17]. Routinely, we measure automobile efficiency in miles (effective output) per gallon of fuel (total input)[1].

Table 2 summarizes some of the published definitions of efficiency. Berger and Mester discuss the caveats of the various efficiency estimates employed by financial institutions; however, in general financial efficiency is the ratio of the money made (effective output) to the money spent (total input) [18]. Estache et. al. used energy delivered (effective output) and a combination of the number of employees and the network capital (total input) to compare the efficiencies of South American utility providers [19].

reference	*researcher*	*effective output*	*total input*
[18]	Berger, Mester	money made	money spent
[19]	Estache et. al	energy delivered	# employees, network capital
[3]	Sanchez	time, cost	# patents, # publications
[20]	Scharl et. al	customer value	process costs

Table 2: Efficiency Outputs and Inputs

With respect to team efficiency, researchers have used a number of formulations. Sanchez used the number of patents and journal publications (effective output) and the time and cost of development (total input) to compare the efficiencies of some Spanish R&D projects [3]. Scharl et. al. proposed customer value (effective output) and process costs (total input) to evaluate the efficiency and balance the concerns of Web Information Systems [20].

Placed in the context of Hackman's effectiveness model [4], a team's efficiency is the ratio of their effective (actual) output to the effort that they expended, in executing their process, to generate that output. For evaluating software development team efficiencies, researchers have tended to treat the software

[1]Europe uses litres (total input) per 100 kilometers (total output)

itself as the effective output, generated from a total input of developer cost and/or developer time.

In addition to absolute efficiency values, some researchers have used DEA to derive relative efficiency values for a team (within a set of teams). DEA is a linear programming approach to determine the relative efficiencies of "decision making units" where the outputs and inputs are "user defined" [21].

Vitner et. al. investigated using Data Envelopment Analysis (DEA) in order to compare project efficiencies in a multi-project environment [22]. Similarly, Farris et. al used DEA to compare project efficiencies in the Belgian Armed Forces where the projects were completed with different engineering processes [23]. Several researchers have used DEA to compare software project efficiencies - in particular, the effective output (the software itself) has been characterized by lines of code (SLOC) [24, 25], function points (FP) [26], and quality (defects) and combinations of these [27].

Unfortunately, comparing the efficiency of one team with another, either absolutely or relatively, is not easy. As with effectiveness, there are potentially many differences between the definitions of efficiency and with the teams and their work. And although DEA does "level the playing field" somewhat, it requires fairly homogeneous teams and/or projects [28]. In addition, completing a DEA analysis requires someone in your organization to be or become familiar with DEA.

1.3 Research Questions

Referring back to Hackman's model for team effectiveness [4]: he suggested that we assess team effectiveness based on a team's actual output (the solution or product) in addition to the "state" of the team and the individuals in the team after the project. He also recommended that the actual output, or the productive output, be evaluated by organization members or clients (to avoid limiting the domain of the model).

However, we and others observe that software development teams are "unique" because they create a lot of persistent artifacts which facilitate direct, reliable collection of objective data. At least, evaluators can inform themselves with this data when evaluating the team's effectiveness. At most, if the "cost" of the "perceptual" evaluations is too high, they can generate a "standalone" representation of the team's effectiveness with it.

Additionally, we observe that the direct, objective data also provides information regarding the effort to change a software product from one version to another. That is, the objective data provides information about the total input, in addition to the effective output. Hence, it provides information about the team's efficiency.

Therefore, with these observations in mind, we set out to:

- Define a direct measure for efficiency, based solely on the artifacts persisted by a software development team. Although such a measure would convey no "perceptual" information nor "external objective" information (such as budget or schedule variance), we expected (and expect) that it would facilitate direct comparison of team efficiencies. We present our direct measure for efficiency in section 2, using set theory. We apply the theory to an example in section 3.

- Utilize this measurement on a number of projects - we wanted to ensure that the measure could be used in an unobtrusive and inexpensive manner. We present our measurement method and results in sections 4 and 5.
- Identify any patterns, or "efficiency profiles", that could be observed and modeled within and across projects. We discuss our measurement results, with pattern identification as a goal, in section 6.
- Investigate whether those patterns, if there were any, could be used to make forecasts, via an experiment. We develop a mathematical model and describe our experimental method in section 7. We present our experimental results in section 8. We discuss our experimental results in section 9.

Finally, we present our conclusions in section 10.

2 Set Theoretic Model

2.1 Classes in a Build

Software developers usually write "source code" and then use tools to compile it into (virtual) machine language. Most object-oriented developers write their source code in a construct called a Class. Let's define "all of the code" in a given development environment as the set of all Classes.

$$C_{ALL} = \{ x : x \textit{ is a Class} \}$$

Now, each Class in a set of Classes, has a unique "fully qualified name". The actual "fully qualified name", or fqn, for a Class depends on the programming language. For example, in Java, the fqn for a Class is a combination of the Class' package and the Class' name. But, in Smalltalk, the fqn for a Class is just the Class' name. In general, the fqn of a Class is a function which maps from a Class to a Name for that Class.

$$\mathit{fqn} : \mathit{Class} \rightarrow \mathit{Name}$$

Software developers make changes in one of three ways: they create a new class, they edit an existing class, or they delete an existing class. When they create a new class, they create the initial revision of the class. When they edit an existing class, they create another revision of the class.

All revisions of a particular class have the same fqn. Hence, the inverse of the fqn function is not a function, but a relation. It maps a particular Name to a set of Classes - its revisions.

$$\mathit{revisions} : \mathit{Name} \leftrightarrow \mathit{Class}$$

At certain points in a development cycle, a development team takes a specific subset of Classes and includes them in a build - let's call it build "i". Each Class that they include in a build should have a unique fully qualified name - that is, at most one revision of a Class should be included in a build. Otherwise, there may be ambiguity for a (virtual) machine executing the build code.

$$C_i = \{ x : x \in C_{ALL} \wedge \mu\, \mathit{fqn}(x) \}$$

2.2 Methods in a Build

When a developer writes a Class, they declare a set of Methods within it. Within a Class, they differentiate Methods by giving it unique signature. Within Methods, they specify how instances of a Class will behave in response to certain events. This specification is also often called an implementation, because it corresponds to the implementation phase in many software development lifecycles. Unfortunately, this can be confusing.

$$methods : Class \leftrightarrow Method$$
$$methodSig : Method \rightarrow Signature$$
$$methodSpec : Method \rightarrow Specification$$

We choose a methods relation, rather than a methods function, because it conveys that a particular Method could be "reused" by several revisions of a Class (which is often the case). Unfortunately, this allows a particular Method to be related to several Classes. But, in most class based languages, "sharing" of this type is facilitated through inheritance. Therefore, we prefer a constraint which relates a particular Method to a particular Class - its declaring Class.

$$declaringClass : Method \rightarrow Class$$

So, when the development team includes a set of Classes in a build, build i, they include all of the Methods related to those Classes.

$$M_i = \{\, methods(x) : \; x \in C_i \,\}$$

2.3 Changes - Methods

Now, consider two builds: build i and build j, where build j follows build i in time. The Methods in build j that were not in build j are the Methods that the development team has added.

$$MAdded_{ij} = M_j \setminus M_i$$

Similarly, the Methods in build i that are no longer in build j are the Methods that the development team has deleted.

$$MDeleted_{ij} = M_i \setminus M_j$$

Finally, the Methods that are in both build i and build j are the Methods that the development team has retained.

$$MRetained_{ij} = M_i \cap M_j$$

Of the methods retained, some of them will be identical and some will have changed. By definition, those that are identical have the same signature and the same specification. Those that have changed have the same signature and a different specification.

$$MIdentical_{ij} = \{\; x, y : MRetained_{ij} \mid methodSig(x) \equiv methodSig(y) \land methodSpec(x) \equiv methodSpec(y) \,\}$$

$$MChanged_{ij} = \{\; x, y : MRetained_{ij} \mid methodSig(x) \equiv methodSig(y) \land methodSpec(x) \neq methodSpec(y) \,\}$$

Methods are not the only components of a Class. However, they are the primary components. Therefore, as an initial approximation, we define the Change from build i to build j as the set of Method changes between the two.

$$\Delta C_{ij} = MAdded_{ij} \cup MDeleted_{ij} \cup MChanged_{ij} \tag{1}$$

2.4 Changes - Instructions

However, intuitively, the Method seems to be the wrong level of abstraction for changes between two builds. Although it is the main "building block" in class based object-oriented systems, there is a tremendous amount of variability in the size and complexity of Methods.

For example, "getter" and "setter" methods are very simple. And, although it's not recommended, some Methods are large and complex. Therefore, it does not seem reasonable to allocate the same "weight" to Method changes, independent of their contents. Rather, the changes to the Instructions within a Method seems more reasonable.

Fortunately, we can reuse the relationships for changes, at a lower level of abstraction. In the same way that Classes are composed of Methods, Methods are composed of Instructions. And, in the same way that Methods belong to a particular Class, Instructions belong to a particular Method. Similarly, the Instructions in build i are the sum of the Instructions for each Method in build i.

$$instructions : Method \leftrightarrow Instruction$$
$$declaringMethod : Instruction \rightarrow Method$$
$$I_i = \{\, instructions(x) : x \in M_i \,\}$$

So, following the pattern of our previous derivations, we have:

$$IAdded_{ij} = I_j \setminus I_i$$
$$IDeleted_{ij} = I_i \setminus I_j$$
$$IRetained_{ij} = I_i \cap I_j$$
$$IIdentical_{ij} = \{\, x, y : IRetained_{ij} \mid x \equiv y \,\}$$
$$IChanged_{ij} = \{\, x, y : IRetained_{ij} \mid x \neq y \,\}$$

And, instead of ΔC_{ij} being derived from Methods, it is derived from Instructions.

$$\Delta C_{ij} = IAdded_{ij} \cup IDeleted_{ij} \cup IChanged_{ij} \tag{2}$$

2.5 Force

In classical physics, force is defined as a vector quantity that tends to produce an acceleration of a body. In a software context, and in the context of this discussion in particular, we think that a reasonable analog is a software team changing a software product. The software product is the body and the software team's changes are its acceleration.

Consider a software development iteration with builds 1 through N. The software development team's force during the iteration is the sum of their individual forces, or changes to the product, for each build.

$$Force_{0N} = \sum_{0}^{N-1} \Delta C_{i,i+1} \tag{3}$$

2.6 Efficiency

Efficiency is typically defined as the ratio of output over input.

Consider the difference between three software builds, build i, build j, and build k. We define the output as the changes between build i and build k. We define the inputs as the changes between build i and build j and between build j and build k.

And, since efficiency is the ratio of output over input, we have:

$Efficiency_{ik} = \frac{Output_{ik}}{Input_{ij} + Input_{jk}} = \frac{\Delta C_{ik}}{\Delta C_{ij} + \Delta C_{jk}}$

From a historical perspective, developers will either preserve changes made during a sequence of builds, or they will discard them. We define the output as the set of changes between the first build and the last build (in a particular sequence of builds) - that is, the changes that the developers preserve. We define the input as the sum of all changes between the first build and the last build - that is, the sum of all of the changes that the developers make. For convenience, we refer to the changes that developers preserve as effective changes; likewise, we refer to the sum of all changes as cumulative changes.

$$Efficiency_{0N} = \frac{Output_{0N}}{\sum_{0}^{N-1} Input_{i,i+1}} = \frac{\Delta C_{0N}}{\sum_{0}^{N-1} \Delta C_{i,i+1}} = \frac{\Delta C_{0N}}{Force_{0N}} \tag{4}$$

Now, with these definitions, we observe that efficiency will range between 0 and 1 and that the more samples we take, the more accurate our efficiency measurement will be:

- If there are no changes between build 1 and build N, the efficiency will be undefined (0 over 0). This is reasonable since efficiency is meaningless if no changes have been made.
- If the cumulative differences between build 1 and build N are equal to the effective differences between build 1 and N, the efficiency will be 1. This is reasonable since all changes were preserved and no changes were discarded. A typical example for this is a "one build" iteration.
- The effective changes between build 1 and build N cannot be greater than the cumulative changes between build 1 and build N. This is reasonable because each of the effective changes will be included as one of the cumulative changes.
- The more samples of changes we take, the lower and more accurate the efficiency will be. This is reasonable because with each sample, the denominator will either increase or stay the same and the numerator will remain constant.

3 Example

Let's look at a specific example.

3.1 Classes

Consider a contrived software product, written in Java, by a small software development team. Further, consider a single class within the software product - an Amount class. It is used to model the data and behavior for an amount of money. Listing 1 shows what the class looks like initially.

Listing 1: Revision 1

```
public class Amount {
    protected double value = 0.0;

    public Amount(double value) {
        setValue(value);
    }

    public double getValue() {
        return value;
    }

    public void setValue(double value) {
        this.value = value;
    }

    public Amount add(Amount amount) {
        double sum = getValue();
        sum += amount.getValue();
        return new Amount(sum);
    }
}
```

3.2 Changes - Methods

Now, for whatever reasons, the development team modifies the Amount class so that it models an amount in a particular currency. Listing 2 shows what it looks like after the changes.

Listing 2: Revision 2

```
public class Amount {
    protected Currency currency = CurrencyConverter.DEFAULT_CURRENCY;

    protected double value = 0.0;

    public Amount(double value) {
        this(CurrencyConverter.DEFAULT_CURRENCY, value);
    }

    public Amount(Currency currency, double value) {
        setCurrency(currency);
        setValue(value);
    }
```

```
    public Currency getCurrency() {
        return currency;
    }

    public void setCurrency(Currency currency) {
        this.currency = currency;
    }

    public double getValue() {
        return value;
    }

    public void setValue(double value) {
        this.value = value;
    }

    public Amount add(Amount amount, CurrencyConverter converter) {
        double exchangeRate = converter.convert(amount.getCurrency(), getCurrency());
        return new Amount(getValue() + (amount.getValue() * exchangeRate));
    }
}
```

By inspection, we can see that:

- The "Amount(double)" constructor method was changed
- An "Amount(Currency,double)" constructor method was added
- Currency "getter" and "setter" methods were added

What might not be obvious is that the "add(Amount)" method from the first has been deleted and a new "add(Amount,CurrencyConverter)" method has been added - their signatures are different. Because we do not have visibility into how the developers made the change, we treat it as a "method delete" and a "method add" rather than a "method change".

Using (1), we observe that the method changes are:

$$\Delta C_{12} = MAdded_{12} \cup MDeleted_{12} \cup MChanged_{12}$$
$$\Delta C_{12} = 4 + 1 + 1 = 6$$

Now, to make the example more realistic, the development team again changes the class so that a currency converter need not be passed around. Listing 3 shows what it looks like after the changes.

Listing 3: Revision 3

```
public class Amount {
    protected Currency currency = CurrencyConverter.DEFAULT_CURRENCY;

    protected double value = 0.0;

    public Amount(double value) {
        this(CurrencyConverter.DEFAULT_CURRENCY, value);
    }
```

```
    public Amount(Currency currency, double value) {
        setCurrency(currency);
        setValue(value);
    }

    public Currency getCurrency() {
        return currency;
    }

    public void setCurrency(Currency currency) {
        this.currency = currency;
    }

    public double getValue() {
        return value;
    }

    public void setValue(double value) {
        this.value = value;
    }

    public Amount add(Amount amount) {
        return add(amount, getDefaultCurrencyConverter());
    }

    public Amount add(Amount amount, CurrencyConverter converter) {
        double exchangeRate = converter.convert(amount.getCurrency(), getCurrency());
        return new Amount(getValue() + (amount.getValue() * exchangeRate));
    }

    protected CurrencyConverter getDefaultCurrencyConverter() {
        return DefaultCurrencyConverter.getSoleInstance();
    }
}
```

This time, not much has changed.

- The "add(Amount)" method has been (re)added
- A "getDefaultCurrencyConverter()" method has been added

Using (1), we observe that the method changes are:

$$\Delta C_{23} = MAdded_{23} \cup MDeleted_{23} \cup MChanged_{23}$$
$$\Delta C_{23} = 2 + 0 + 0 = 2$$

3.3 Force

As derived in (3), the force applied by the development team is:

$$Force_{13} = \sum_1^2 \Delta C_{i,i+1}$$
$$Force_{13} = 6 + 2 = 8$$

3.4 Efficiency

As derived in (4), the efficiency applied by the development team is:

$$Efficiency_{1N} = \frac{\Delta C_{1N}}{Force_{1N}}$$

So, we first need to determine the method changes between revision 1 and revision N. We observe that:

- The "Amount(double)" constructor method has changed
- An "Amount(Currency,double)" constructor method has been added
- Currency "getter" and "setter" methods have been added
- The "add(Amount)" method has changed
- An "add(Amount,CurrencyConverter)" method has been added
- A "getDefaultCurrencyConverter()" method has been added

So, we have:

$$\Delta C_{13} = MAdded_{13} \cup MDeleted_{13} \cup MChanged_{13}$$
$$\Delta C_{13} = 5 + 0 + 2 = 7$$

Therefore:

$$Efficiency_{1N} = 7/8 = 0.875$$

4 Measurement Method

In this section, we describe how we measured the efficiency of several open source projects, using the measurement defined in section 2. The sample product URLs are listed in the Appendix, section 13.1.

4.1 Revision Comparison

The first thing we required was a tool set to measure the changes between two sets of Java classes. We built such a tool set, leveraging the Byte Code Engineering Library (BCEL) [29]. Its name is "jeanda" - Java Efficiency AND Agility. Its implementation can be found here: http://jeanda.tigris.org.

As specified in sections 2.3 and 2.4, this tool determines the methods that have been added, changed, and deleted. Additionally, it determines the instructions that have been added and deleted. Finally, it determines the number of bytes (within those instructions) that have been added and deleted.

Section 13.2 in the Appendix describes the detailed steps required to compare sets of Java classes, as well as a sample comparison result.

4.2 Revision Reconstruction

Our next step was to reconstruct the revisions of a product, according to its configuration management log. At a high level, this involved identifying the revisions for the product, assembling the files for each particular revision, and building and storing each revision of the product. Section 13.3 in the Appendix describes the detailed steps required to reconstruct product revisions.

Once we had reconstructed all of the product revisions that we could, we recorded the timestamp for each. We did this so that we'd be able to determine the time between one product revision and another.

4.3 Sample Identification

Once we had reconstructed the product revisions for a project, we were ready to begin comparing revisions, in order to form an "efficiency profile" - that is, a profile of the team's efficiency with respect to time. However, because real-life projects often have tens or hundreds (or more) revisions per week, an exhaustive comparison of all permutations was not realistic - we did not (and do not) have the computational resources. Therefore, we needed to identify a reasonable set of "comparison samples".

We decided to sample a product's efficiency as follows:

- We treated all of the changes for a month as a "monthly change set".
- We decided to sample the efficiency for each revision within the "monthly change set".
- Outside of the monthly change set, we decided to sample the efficiency, on a weekly basis, as far as we could into the future.

We decided to sample exhaustively within a "monthly change set" in case there were short-term patterns. We decided to sample on a weekly basis, outside of the "monthly change set", due to resource limitations.

4.4 Sample Measurement

Once we had identified our "comparison samples", we were ready to measure the team's efficiency, with respect to time, for each sample. To measure the efficiency for a particular "comparison sample", from a particular revision (N) to a future revision (M), we determined the following values:

- The total input (i.e. the sum of all of the effective forces from N to M).
- The effective output (i.e. the effective force between N and M).
- The time (t) between N and M.

With these values, we had our efficiency sample: the efficiency (effective output divided by total input) as a function of time.

5 Measurement Results

In this section, we present the results of our efficiency measurements. Figure 1 presents the absolute efficiency measurements for the ehcache project, according to the day of the year they were measured. For example, the measurements for the 2006-06 change set began around the end of May, which is approximately day 151.

The rest of the figures present relative efficiency measurements, where the absolute curves have been shifted to a common origin - this shifting makes it easier to visualize any patterns in a curve group.

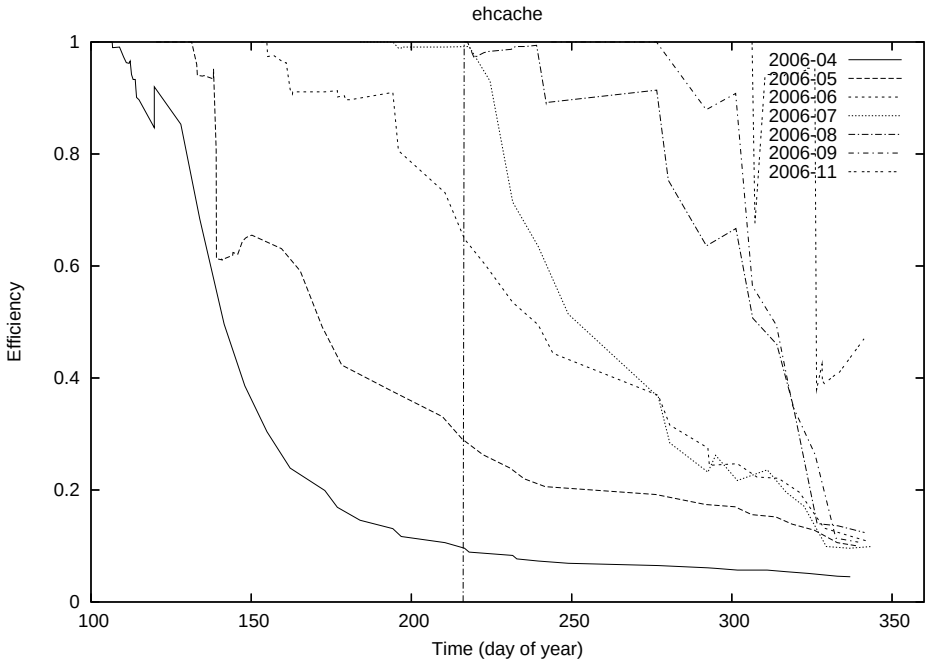

Figure 1: ehcache - Efficiency over Time

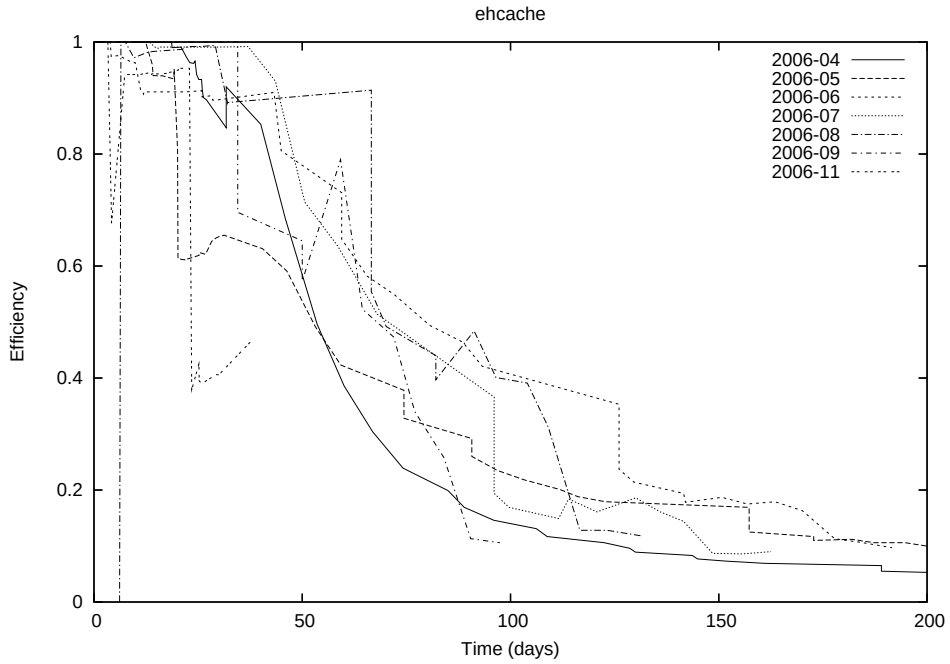

Figure 2: ehcache - Efficiency over Time

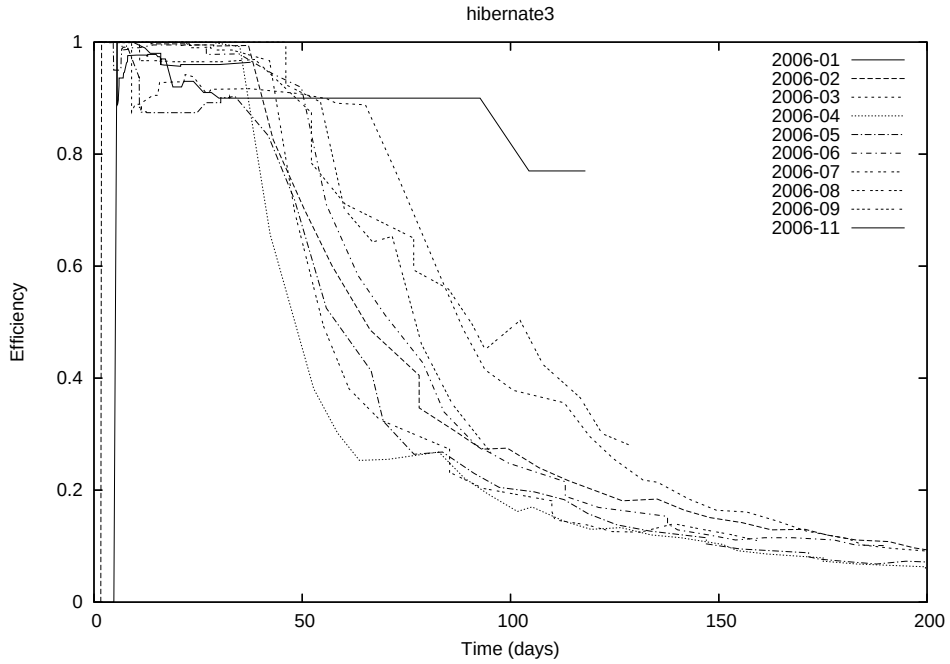

Figure 3: hibernate3 - Efficiency over Time

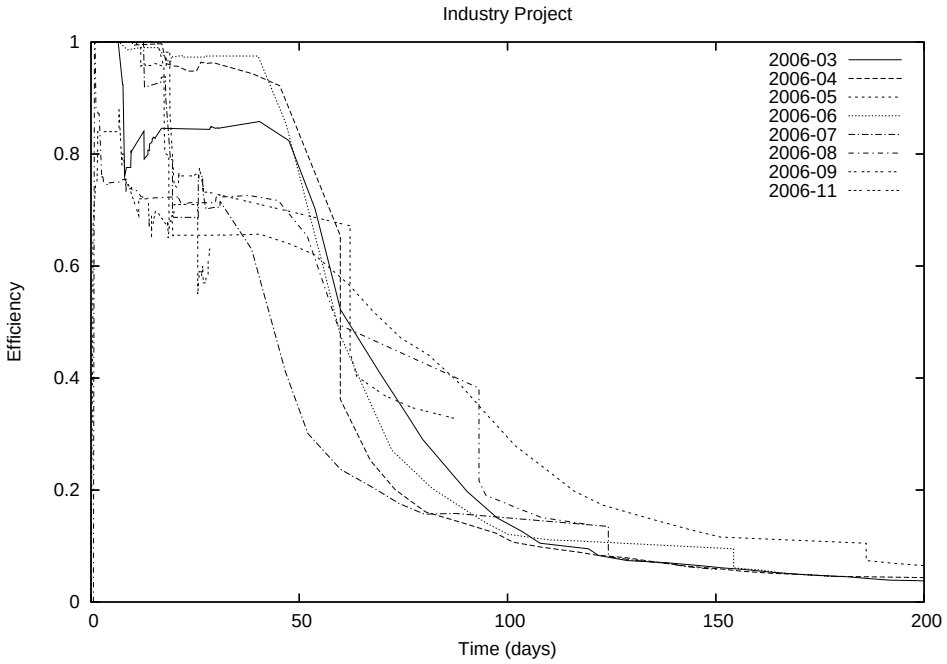

Figure 4: Industry Project - Efficiency over Time

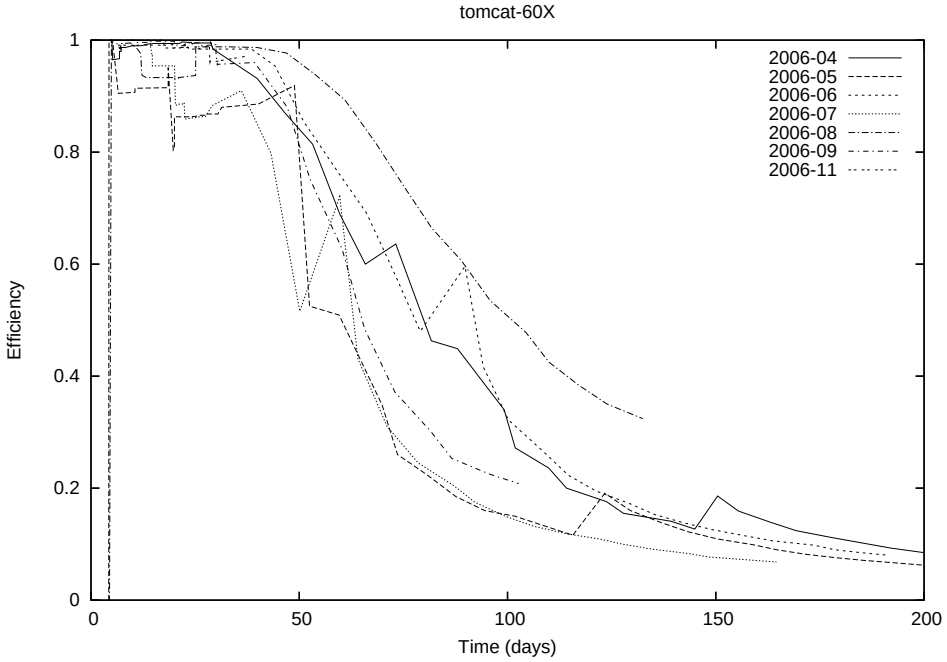

Figure 5: tomcat-60X - Efficiency over Time

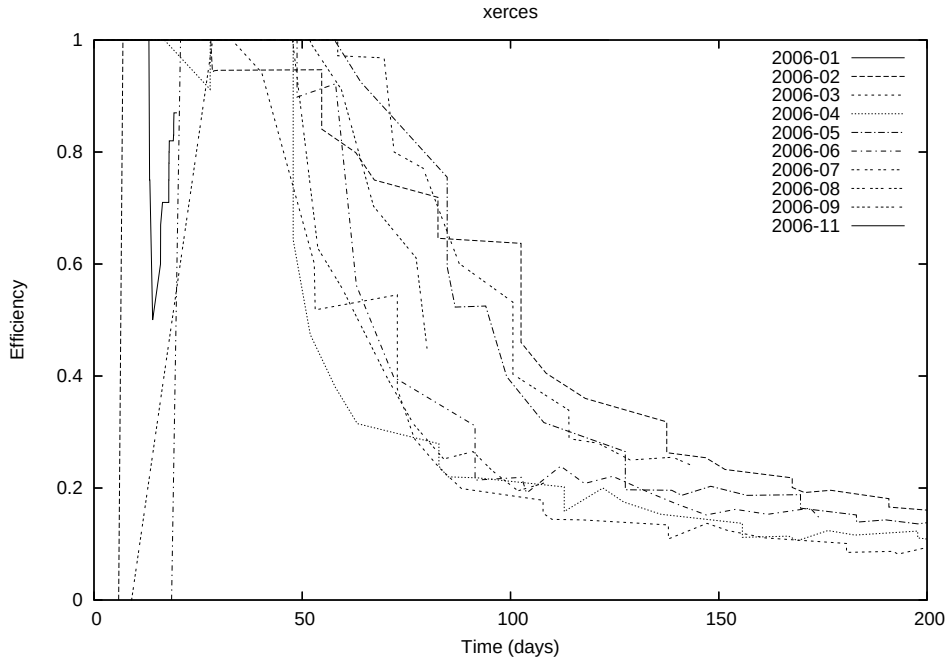

Figure 6: xerces - Efficiency over Time

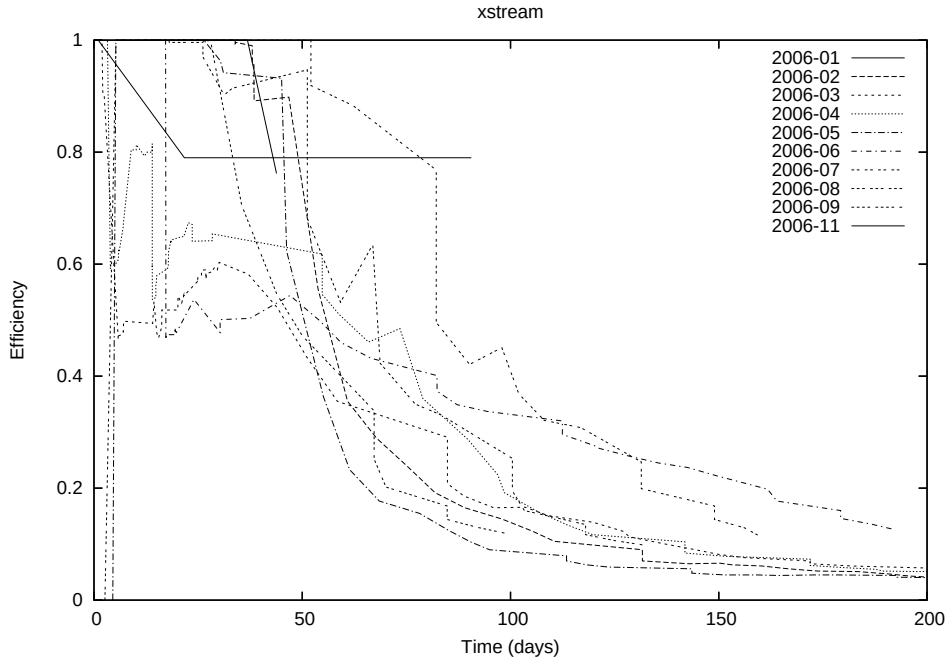

Figure 7: xstream - Efficiency over Time

6 Measurement Discussion

In this section, we discuss the measurement results in section 5. One of our research questions was: are there any patterns within a project and/or between products with respect to efficiency per time? Based on our measurements, we think there are a few patterns, at least within and amongst this small set of projects.

6.1 Stabilization Period

Our first observation is that there appears to be a "stabilization period" for the efficiency of a "monthly change set". For all of the projects we measured, this period seems to be about 40-50 days and is characterized by "erratic" or random fluctuations in efficiency.

At first, this period was a little confusing. Since we sampled all of the revisions within a "monthly change set", we expected this period to be characterized by a smooth decay from an efficiency of 1. We did not expect to see increases in efficiency!

However, upon closer inspection, it made more sense. During this period, a "critical mass" of changes is "built up". Since our starting point is arbitrary (the start of the month), changes can come at any time and in any size. If a relatively small number of inefficient changes is followed by a large number of more efficient changes, then we see a "local" increase in efficiency - the larger changes "overpower" the smaller ones.

6.2 Decay Period

Our second observation is that there is a "decay period". For all of the projects we measured, this period seems to start about 40-50 days and end about 150-200 days. During this period, the efficiency decays from between about 75-95% to about 10-20%.

We also note that the decay is not linear. That is, the efficiency does not drop off at a constant rate. Rather, the decay is exponential - most of the decay happens rather quickly. This would seem to indicate that that changes which will eventually be discarded, will more likely be discarded sooner rather than later.

Finally, we note that the decay curve is not identical for each of the projects. Since we know nothing else about the projects, we cannot speculate as to why this might be.

6.3 Constant Period

Our third observation is that there is a "constant period". For all of the projects we measured, this period seems to start at about 150-200 days. During this period, the efficiency approaches a constant (somewhere between 10 and 20%).

Once the efficiency reaches a constant, for the set of changes in question, pretty much all of the discarding has taken place and the remaining changes are preserved. That is, the resilient changes remain within the product into this constant period - the rest are discarded during either the stabilization period or the decay period.

7 Experiment Method

In order to perform an experiment, we needed a basis for making forecasts - we used statistical nonlinear regression to derive a mathematical model capable of doing so. In the following sections, we describe this model's derivation and then how we used it to make efficiency forecasts.

7.1 Mathematical Model

According to Winkler and Hays, when a statistician wishes to predict the value of one variable, given another variable, he or she uses regression. If there is a high level of correlation between the variables, then linear regression is possible. However, not all interesting relationships are linear - when they are not, the statistician relies on nonlinear regression techniques [30].

If they know the exact joint distribution, then they can compute the exact form of the regression curve. However, this is usually not the case. Therefore, they normally estimate a regression curve on the basis of sample data. First, they specify a mathematical model for the data, using "scatter plots" to aid in their selection. Second, they fit that mathematical model to the data by estimating its unknown parameters [30].

Efficiency Model

In our case, we wished to predict the efficiency of a software team, given a particular time in their development cycle. Since we did not know the exact joint distribution, and since we had sample data, we decided to specify a mathematical model using that data.

However, we deferred estimating the model's unknown parameters for a few reasons: (1) it was obvious from our measurements that the projects did not conform to the same specific model; and (2) it was questionable that the curves in the same project would conform to the same specific model. But, it looked like all of the curves in all of the projects belonged to the same family of curves. Therefore, at this point, we concentrated on specifying a family of curves.

As we discussed in section 6, our "scatter plots" exhibited a common pattern: stabilization, decay, constant. Because the stabilization periods had no consistent pattern, we focused on specifying a model for the decay and constant periods (from 50 to 200 days). Using a trial and error process, in addition to the gnuplot tool [31], we arrived at this family of "exponential decay" curves - efficiency as a function of time with three variables (a, b, c).

$$E(t) = \frac{1}{e^{\frac{t-a}{b}}} + c \tag{5}$$

We used an "a" variable in order to allow for a shift of the start of the decay (Figure 8) - that is, where the decay curve has an efficiency value of 1. We used a "b" variable in order to allow for a shift in the slope of the decay (Figure 9) - that is, how quickly the decay curve decays. Finally, we used a "c" variable in order to allow for a shift of the constant value (Figure 10).

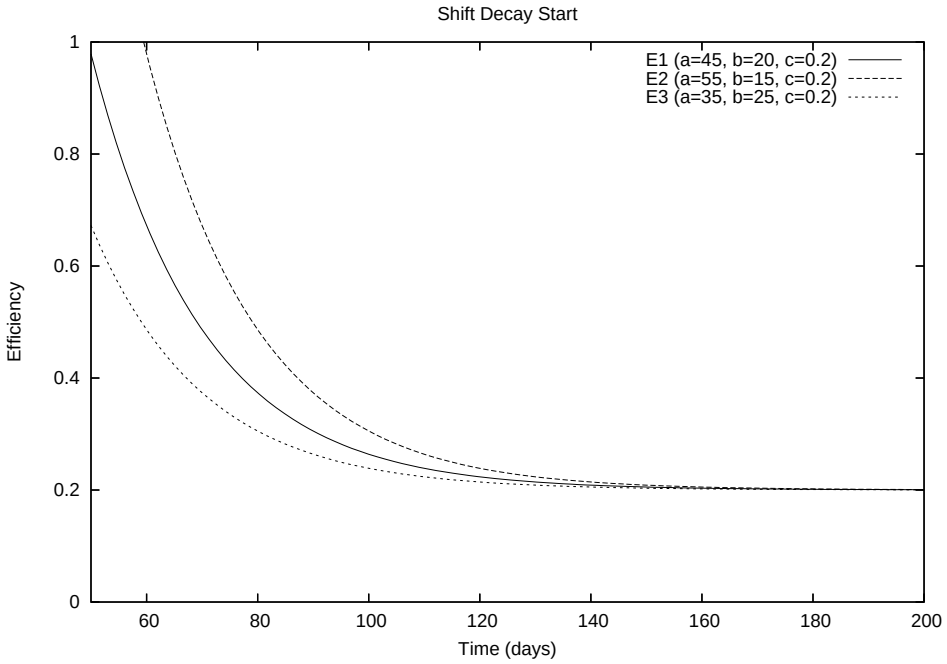

Figure 8: Shift Decay Start

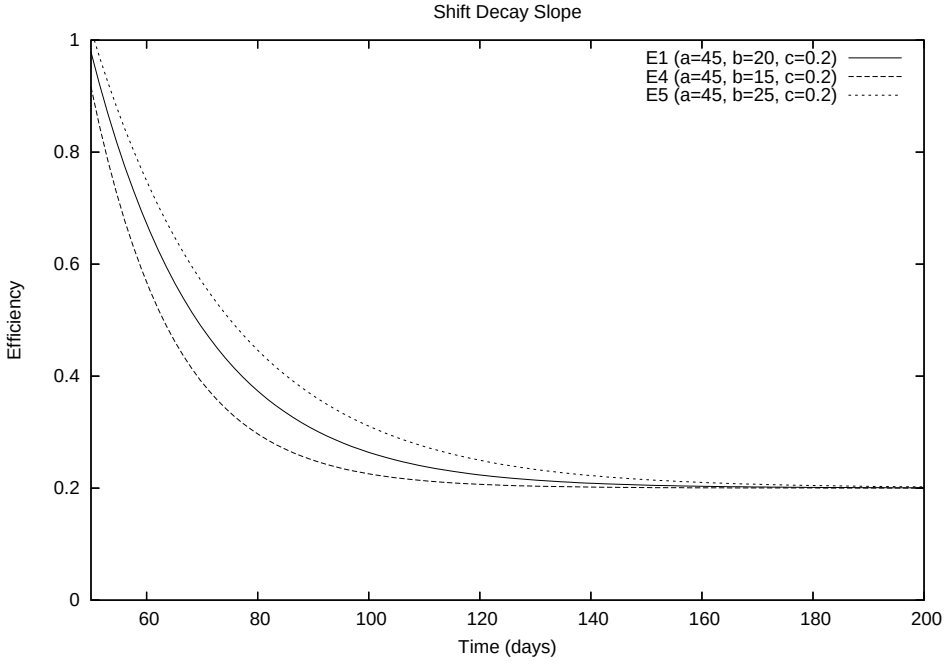

Figure 9: Shift Decay Slope

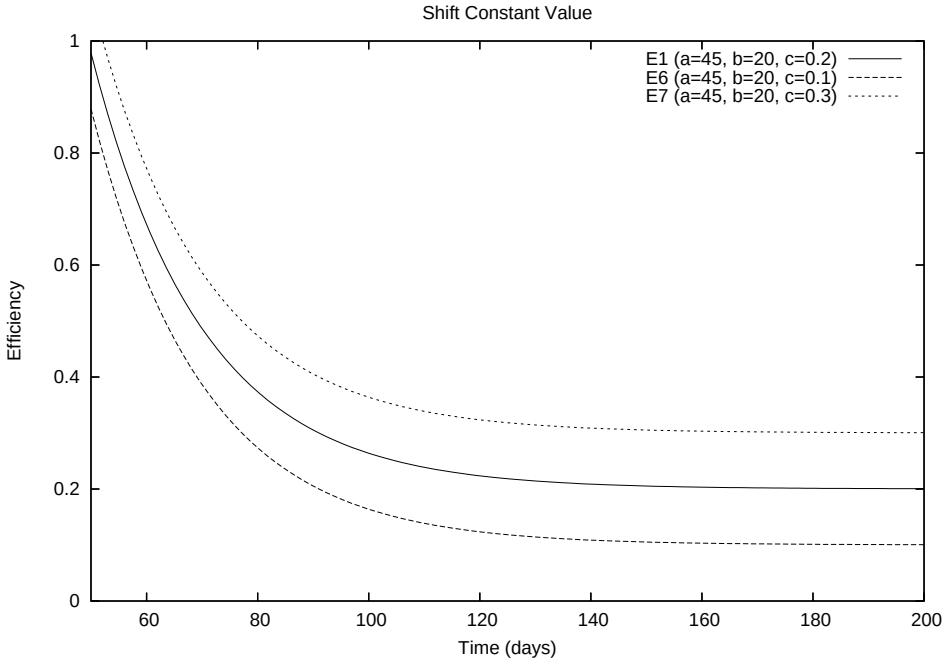

Figure 10: Shift Constant Value

7.2 Efficiency Forecasting

Since our data was time-series data, as opposed to cross-section data, it was important that we only use ex-ante (past) efficiency values to predict future efficiency values [32]. In addition, since we could not find clear guidance on which forecasting method to use, we set up a forecast "tournament", as recommended by Armstrong [33]. We defined forecast methods, competitors in the tournament, along two dimensions: (1) the past data used to forecast; and (2) the mechanism used to forecast . We evaluated the forecast methods with two standard measures: mean absolute error (MAE) and mean absolute percentage error (MAPE), also as recommended by Armstrong [33]. We also compared the forecast methods with the results of a "random walk" forecast.

Forecast Data

Within the first dimension, we used two different types of past data. The first type was limited to the past efficiency data for a particular change set, within a specific project - we referred to this as "local" data. The other type included the past efficiency data for all of the change sets within that specific project - we referred to this as "global" data.

Forecast Mechanism

For the second dimension, we used two different mechanisms to forecast.

The first mechanism was fitting our general model (equation 5) to a data set, with gnuplot [31], and then using the solved parameters to forecast a value

- we referred to this as a "custom" mechanism. The gnuplot tool fits a data set with a nonlinear least-squares (NLLS) Marquardt-Levenberg algorithm. More detail on this method is in the Appendix (section 13.4).

For the second mechanism, we used a third party tool called "OpenForecast" [34]. We chose the "Regression" forecast model, "fed" it the past data, and asked it to forecast a value - we referred to this as a "thirdparty" mechanism. The Regression model was the only forecast model that "behaved" with our "unevenly" distributed time data (which cannot be helped).

Forecast Evaluation

We evaluated the predictions of the competing forecast methods with two measures. The first measure we used was mean absolute error (MAE). We calculated this by keeping track of the absolute error for each prediction, as compared to the actual value, and then determining the mean of those errors. The second measure we used was similar - mean absolute percentage error (MAPE). We calculated this by keeping track of the absolute percentage error for each prediction, as compared to the actual value, and then determining the mean of those errors.

Random Walk

In addition to our four forecast methods, local/custom, local/thirdparty, global/custom, and global/thirdparty, we decided to use the "random walk without drift" method [35]. This method predicts that the next value will be the same as the last. While simplistic, it provides a boundary of sorts for the other methods - in order to be useful, a forecast method needs to be better than this method [33].

Forecast Example

Consider, for example, a fictional project.

Last month's change set data consists of (50, 0.75), (60, 0.7), (70, 0.6), and (80, 0.48), where (t, e) represents an efficiency measurement (e), at a certain time (t). This month's change set data consists of (50, 0.85), (60, 0.78), and (70, 0.62).

In order to forecast the efficiency value at t=80 for this month, using the local/custom method, we will use only this month's change set data (3 points) and have gnuplot fit this data to our general model. Using the global/custom method, we will use all of the data (7 points) and again use gnuplot to fit the data. Similarly, in order to forecast with the thirdparty mechanism, we would either provide 3 points (local/thirdparty) or 7 points (global/thirdparty) to our thirdparty forecaster. Finally, our random walk method would predict a value of (80, 0.62), based on the latest relevant data point - (70, 0.62).

If the actual value is (80, 0.5), then the Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) "components", for our random walk forecast, would be:

$$MAE_{component} = \mid 0.5 - 0.62 \mid = 0.12$$

$$MAPE_{component} = \frac{|0.5-0.62|}{0.5} = 0.24$$

8 Experiment Results

In this section, we present our experimental results. We present our forecast evaluations, Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE), in both Tables (3, 4) and Figures (11, 12). We discuss these results in section 9. Note that we reduced the y-scale on the MAPE figure so that a better distinction can be made for the better methods - this has the effect of hiding some of the results from the "poorer" forecast methods.

project	*local/custom*	*local/thirdparty*	*global/custom*	*global/thirdparty*	*random*
ehcache	.033	.089	.072	.161	.056
hibernate3	.021	.087	.047	.157	.052
industry-X	.016	.124	.068	.155	.046
tomcat-60X	.034	.103	.066	.135	.061
xerces	.041	.212	.083	.138	.061
xstream	.024	.087	.067	.184	.045

Table 3: MAE Evaluations

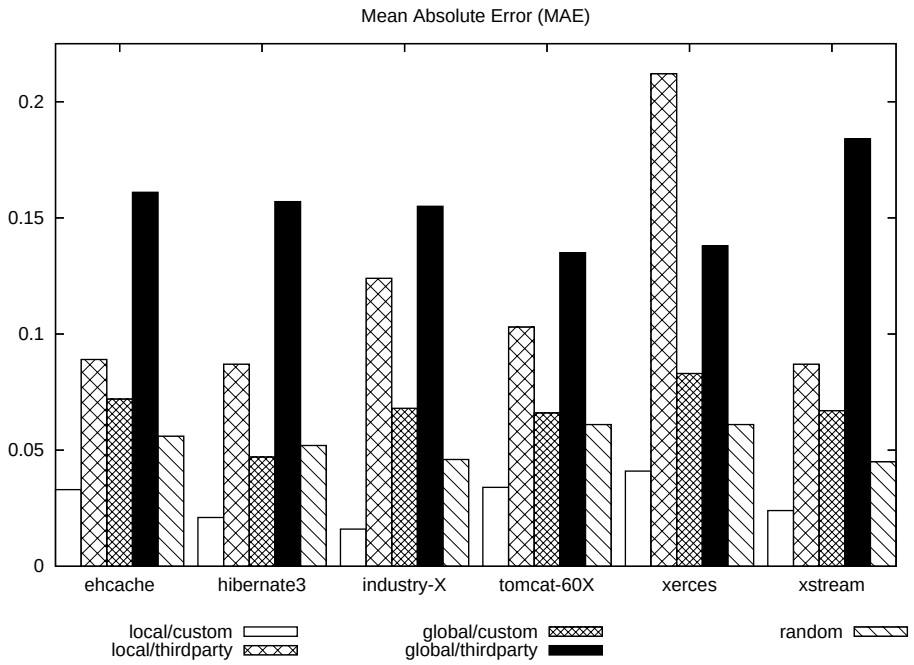

Figure 11: MAE Evaluations

9 Experiment Discussion

In this section, we discuss the experimental results (section 8). One of our research questions was: can we use the "efficiency profiles" (patterns) to predict future efficiency values? Our preliminary result is that at least one forecast

project	*local/custom*	*local/thirdparty*	*global/custom*	*global/thirdparty*	*random*
ehcache	.155	.474	.308	1.715	.179
hibernate3	.089	.519	.204	1.129	.141
industry-X	.149	1.336	.653	1.118	.165
tomcat-60X	.188	.749	.329	1.176	.164
xerces	.158	.758	.248	1.126	.148
xstream	.148	.767	.494	1.329	.152

Table 4: MAPE Evaluations

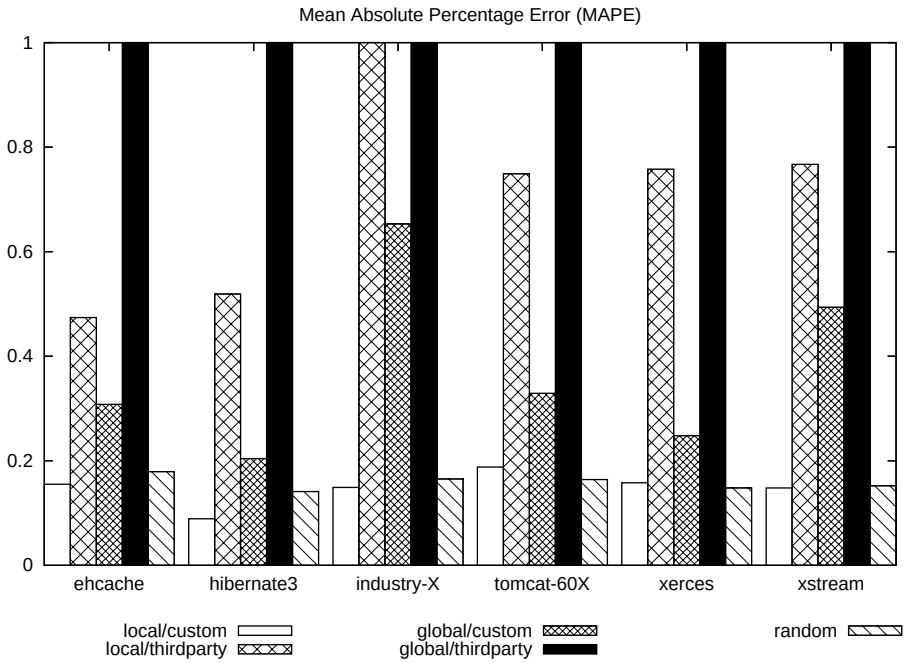

Figure 12: MAPE Evaluations

method is capable of doing so. Finally, we present a "hybrid" forecast method which improves upon that method.

9.1 Observations

Our first observation was that only the local/custom forecast method was better than the random method. It beat the random method by between 1.5-3x for MAE. It was either slightly better or slightly worse than the random method for MAPE.

The global/custom method was about 2-4x as bad, with respect to both MAE and MAPE. We attribute its forecast errors to the fact that each change set starts its "decay curve" at a different point in time. Although the times may be similar for a given project, trying to merge the data from all of the change sets results in poorer forecasts for each particular change set.

Worse still were the local/thirdparty and global/thirdparty methods - they were 3-10x as bad. The "Regression" model was the only forecast model from the toolset that worked with our data - therefore, the thirdparty forecast method fit the data to a straight line rather than to an exponential decay. That is why these forecast methods had such high error values.

Finally, while looking at the individual forecasts from both the local/custom method and the random method, we noticed a pattern. The local/custom method did a better job of forecasting during the "decay" stage; the random method did a better job during the "constant" stage. Therefore, we formed a "hybrid" forecast method.

9.2 Hybrid

We formed the "hybrid" forecast method by weighting the local/custom method and the random method. As the samples moved across the forecast horizon (from 50 days to 200 days), the weight of the local/custom method decreased from 1.0 to 0.0 while the weight of the random method increased from 0.0 to 1.0. For example, at 100 days (or 50 of 150 days), the local/custom forecast was multiplied by 0.67 and the random forecast was multiplied by 0.33.

Combining the two best methods in this "hybrid" method improved our forecast capability. Tables 5 and 6 and Figures 13 and 14 present the measures for the "hybrid" method, as compared to its component methods. In each case, the "hybrid" method improves upon both of the other methods.

project	*local/custom*	*random*	*hybrid*
ehcache	.033	.056	.030
hibernate3	.021	.052	.018
industry-X	.016	.046	.012
tomcat-60X	.034	.061	.022
xerces	.041	.061	.029
xstream	.024	.045	.018

Table 5: MAE Evaluations

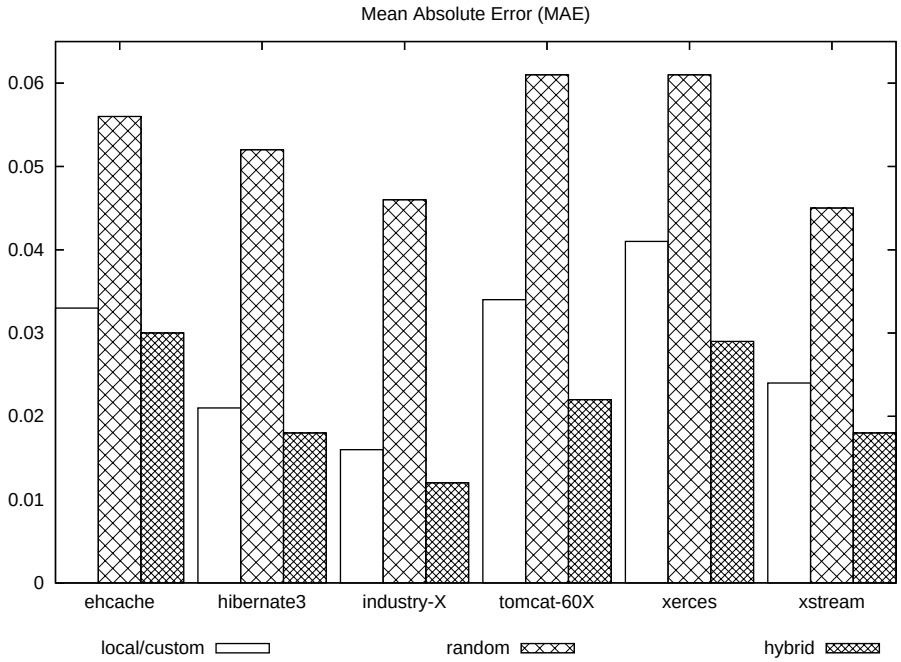

Figure 13: MAE Evaluations

project	*local/custom*	*random*	*hybrid*
ehcache	.155	.179	.139
hibernate3	.089	.141	.076
industry-X	.149	.165	.086
tomcat-60X	.188	.164	.091
xerces	.158	.148	.098
xstream	.148	.152	.089

Table 6: MAPE Evaluations

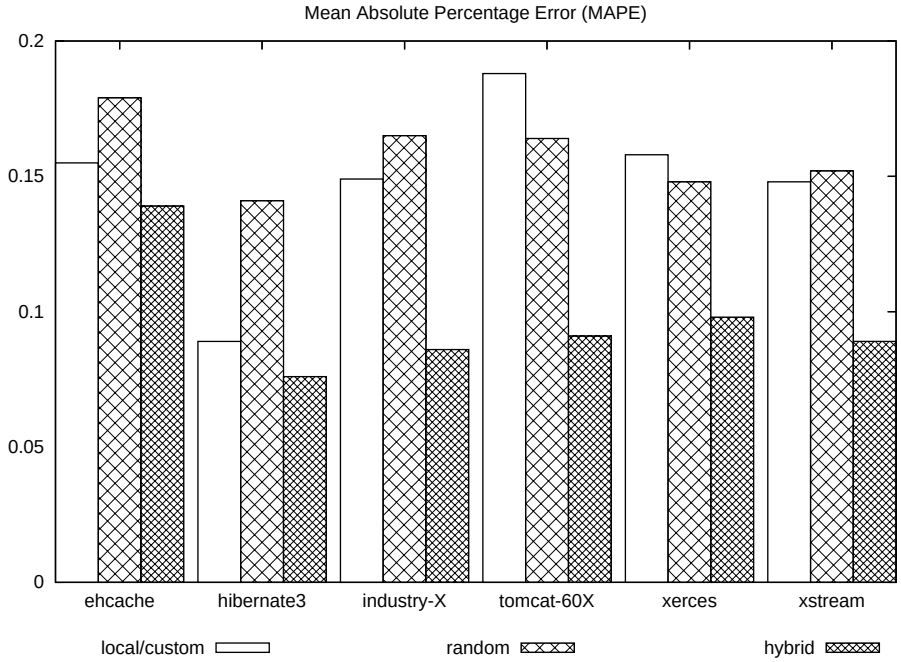

Figure 14: MAPE Evaluations

10 Conclusions

Based on our measurement and forecasting experience, we are able to draw some conclusions.

First, we conclude that the efficiency of a software development team can be measured and compared in an objective and unobtrusive manner. Based on our set-theoretic model and supported by our tools (or similar tools), teams can measure the ratio of their effective changes to their overall changes, over time.

Second, we conclude that the efficiency of a change set follows a particular pattern. Initially, it oscillates as a "critical mass" of changes is assembled. Then, it follows a "decay curve" where newer changes discard some of the older ones. Finally, it smooths out and approaches a constant value where the effective changes that remain are preserved.

Third, we conclude that the efficiency of a change set can be predicted (or forecast), once the "decay curve" has started. The general mathematical model allows a range of specific curves to be fit, after which the fit solutions can be used to forecast future efficiency values. Finally, our "hybrid" forecast method, combining our local/custom and "random walk" methods, won our forecasting tournament by providing the closest forecasts.

11 Future Work

There are a number of natural avenues for further research of these ideas.

First, the sample project set could be expanded in order to widen the model

validity. The current sample size helped us focus our resources and to hone our model and tools. A larger sample size will give us more confidence that the patterns and the forecast methods are generally useful.

Second, the results of this research could be used to evaluate the effect of software development method changes on the efficiency of particular software development teams. If a particular method can be shown to reduce or defer the decay of a team's changes, then that method will have objective data to support its claims (rather than the typical subjective claims).

Third, the model and tools from this research could be used to compare the efficiency of different categories of projects. For example, open source projects could be categorized as small, medium, and large. Alternatively, projects could be categorized as open source, industry, and government. By doing so, meaningful comparisons might be facilitated and benchmarks could be established.

Finally, this work could be extended to include theory, measurements, and forecasts for agility and how it is related to efficiency. If a team can increase its efficiency, will its agility necessarily decrease? Conversely, it a team increases its agility, will its efficiency necessarily decrease?

12 Acknowledgments

I'd like to thank my supervisor, Dr. Andrew Martin, for his substantial guidance on this report. In particular, his help on the set theoretic model was invaluable. He has been both gracious and helpful - a combination I very much appreciate. Although I still have a lot to learn about what it means to research, much of what I have learned has been the result of our interactions. And, of course, any errors in the report are due to me.

Bibliography

[1] E. Schein, *Career dynamics: Matching individual and organizational needs*. Addison-Wesley, June 1978.

[2] P. F. Drucker, *The Effective Executive*. HarperCollins, April 1993.

[3] S. A.M. and P. M.P., "Rd project efficiency management in the spanish industry," *International Journal of Project Management*, vol. 20, no. 7, pp. 545–560, October 2002.

[4] J. R. Hackman, "The design of work teams," *The Handbook of Organizational Behavior*, pp. 315–342, 1986, j. W. Lorsch, Ed. Englewood Cliffs, NJ: Prentice-Hall.

[5] S. G. Adams, "An investigation of the attributes contributing to team effectiveness of engineering and science faculty," IEEE November 10 - 13, 1999 San Juan, Puerto Rico. [Online]. Available: http://citeseer.ist.psu.edu/535643.html

[6] M. R. Waszak, J.-F. Barthelemy, K. M. Jones, R. J. Silcox, and W. A. Silva, "Modeling and analysis of multidiscipline research teams at nasa langley research center: A systems thinking approach," AIAA Paper No. 98-4940. [Online]. Available: http://citeseer.ist.psu.edu/221107.html

[7] T. L. Doolen, M. E. Hacker, and E. M. V. Aken, "The impact of organizational context on work team effectiveness: A study of production team," *IEEE Transactions on Engineering Management*, vol. 50, no. 3, pp. 285–296, August 2003.

[8] C. S. Borrill, J. Carletta, A. J. Carter, J. F. Dawson, S. Garrod, A. Rees, A. Richards, D. Shapiro, and M. A. West, "The effectiveness of health care teams." [Online]. Available: http://citeseer.ist.psu.edu/629677.html

[9] P. H. Grandinetti, "Elementary school child study committee and agencies that serve children and families." [Online]. Available: http://citeseer.ist.psu.edu/635230.html

[10] P. Crawford and P. Bryce, "Project monitoring and evaluation: a method for enhancing the efficiency and effectiveness of aid project implementation," *International Journal of Project Management*, vol. 21, no. 5, pp. 363–373, July 2003.

[11] J. A. Hoffer, J. F. George, and J. S. Valacich, *Modern Systems Analysis Design*, 3rd ed. Prentice Hall, June 2001.

[12] F. M. Hull, "A composite model of product development effectiveness: Application to services," *IEEE Transactions on Engineering Management*, vol. 51, no. 2, pp. 162–172, May 2004.

[13] B. A. and A. J., "Measuring the effect of project management on construction outputs: a new approach," *International Journal of Project Management*, vol. 18, no. 5, pp. 327–335, October 2000.

[14] K. Crowston, H. Annabi, J. Howison, and C. Masango, "Effective work practices for software engineering: Free/libre open source software development." [Online]. Available: http://citeseer.ist.psu.edu/735832.html

[15] J. Herbsleb and A. Mockus, "Using version control data to evaluate the impact of software tools: a case study of the version editor," *IEEE Transactions on Software Engineering*, vol. 29, no. 6, pp. 1–14, June 2003.

[16] P. J. Guinan, J. G. Cooprider, and S. Faraj, "Enabling software development team performance during requirements definition: a behavioral versus technical approach," *Information Systems Research*, vol. 9, no. 2, pp. 101–125, 1998.

[17] Dictionary.com. [Online]. Available: http://www.dictionary.com

[18] A. N. Berger and L. J. Mester, "Inside the black box: What explains differences in the efficiencies of financial institutions?" *Journal of Banking & Finance*, vol. 21, no. 7, pp. 895–947, July 1997.

[19] A. Estache, M. Rossi, and C. A. Ruzzier, "The Case for International Coordination of Electricity Regulation: Evidence from the Measurement of Efficiency in South America," 2002.

[20] A. Scharl, J. Gebauer, and C. Bauer, "Matching process requirements with information technology to assess the efficiency of web information systems," *Information Technology and Management*, vol. 2, no. 2, pp. 193–210, April 2001. [Online]. Available: http://citeseer.ist.psu.edu/scharl00matching.html

[21] A. Charnes, W. Cooper, and E. Rhodes, "Measuring the efficiency of decision making units," *European Journal of Operational Research*, vol. 2, pp. 429–444, 1978.

[22] G. Vitner, S. Rozenes, and S. Spraggett, "Using data envelope analysis to compare project efficiency in a multi-project environment," *International Journal of Project Management*, vol. 24, no. ?, pp. 323–329, ? 2006.

[23] J. A. Farris, R. L. Groesbeck, E. M. V. Aken, and G. Letens, "Evaluating the relative performance of engineering design projects: A case study using data envelopment analysis," *IEEE Transactions on Engineering Management*, vol. 53, no. 3, pp. 471–482, August 2006.

[24] R. D. Banker and C. F. Kemerer, "Scale economies in new software development," *IEEE Transactions on Software Engineering*, vol. 15, pp. 1199–1205, October 1989.

[25] Z. Yang and J. Paradi, "Dea evaluation of a y2k software retrofit program," *IEEE Transactions on Engineering Management*, vol. 51, no. 3, pp. 279–287, August 2004.

[26] R. D. Banker, S. M. Datar, and C. F. Kemerer, "A model to evaluate variables impacting the productivity of software maintenance projects," *Management Science*, vol. 37, pp. 1–18, 1991.

[27] J. C. Paradi, D. N. Reese, and D. Rosen, "Applications of dea to measure the efficiency of software production at two large canadians banks," *Annals of Operations Research*, vol. 73, pp. 91–115, October 1997.

[28] R. G. Dyson, R. Allen, A. S. Camanho, V. V. Podinovski, C. S. Sarrico, and E. A. Shale, "Pitfalls and protocols in dea," *European Journal of Operational Research*, vol. 132, no. 2, pp. 245–259, July 2001.

[29] "Byte code engineering library." [Online]. Available: http://jakarta.apache.org/bcel

[30] R. L. Winkler and W. L. Hays, *Statistics: Probability, Inference, and Decision*, 2nd ed. Holt, Rinehart and Winston, 1975.

[31] "Gnuplot plotting utility." [Online]. Available: http://www.gnuplot.info

[32] P. Goodwin, J. K. Ord, L.-E. Oller, J. A. Sniezek, and M. Leonard, *Principles of Forecasting: A Handbook for Researchers and Practitioners*, ser. International Series in Operations Research Management Science. Kluwer Academic Publishers, May 2001, j. Scott Armstrong (Ed.).

[33] "Principles of forecasting." [Online]. Available: http://www.forecastingprinciples.com/

[34] "Openforecast." [Online]. Available: http://openforecast.sourceforge.net/

[35] "Random walk." [Online]. Available: http://en.wikipedia.org/wiki/Random$_w$*alk*

13 Appendix

13.1 Sample Projects

These are the subversion URLs for the sample projects. The "Industry" project URL cannot be shown as permission has not yet been received:

```
URL: https://svn.sourceforge.net/svnroot/ehcache/trunk
URL: http://anonhibernate.labs.jboss.com/trunk/Hibernate3
URL: <permission pending>
URL: http://svn.apache.org/repos/asf/tomcat/tc6.0.x/trunk
URL: http://svn.apache.org/repos/asf/xerces/java/trunk
URL: svn://svn.xstream.codehaus.org/xstream/trunk/xstream
```

13.2 Comparison

This section describes, in detail and with an example, how to compare two sets of Java classes.

Consider the comparison of the classes from our example in section 3. First, we compile the Java files and archive (jar) the two sets of Java classes into .jar files. Second, we specify the two sets of Java classes and the output directory in an XML file, like this:

```
<comparison>
  <jar-source-1 filename="/some/directory/revision-1.jar" />
  <jar-source-2 filename="/some/directory/revision-2.jar" />
  <output-directory dirname="/some/other/directory/1__2" />
</comparison>
```

Third, we execute one of the comparison tools, which compares the two sets of classes and outputs the results (also in XML files).

```
java -cp jeanda.jar ca.simplify.metrics.cmd.JarComparator <comparison file>
```

Below is a partially elided result, which delineates the number of bytes, instructions, and methods that have changed.

```
<class-comparison>
  <type>CHANGED</type>
  <oldSourcePath>/some/directory/revision-0.jar</oldSourcePath>
  <oldProductKey class="string">Amount.class</oldProductKey>
  <newSourcePath>/some/directory/revision-1.jar</newSourcePath>
  <newProductKey class="string">Amount.class</newProductKey>
  <methodBytesAdded>74</methodBytesAdded>
  <methodBytesEqual>20</methodBytesEqual>
  <methodBytesDeleted>27</methodBytesDeleted>
  <methodInstructionsAdded>42</methodInstructionsAdded>
  <methodInstructionsEqual>12</methodInstructionsEqual>
  <methodInstructionsDeleted>17</methodInstructionsDeleted>
  <methodsAdded>4</methodsAdded>
  <methodsChanged>1</methodsChanged>
  <methodsEqual>2</methodsEqual>
  <methodsDeleted>1</methodsDeleted>
</class-comparison>
```

13.3 Reconstruction

This section describes, in detail and with an example, how to reconstruct different revisions of a product. It is divided into subsections which describe: the identification of revisions, the assembling of a particular revision, and the building of that revision.

Revision Identification

Subversion is a "next-generation" software configuration management (SCM) tool. Each time a software developer commits a set of changes to subversion, a "snapshot" is created. Each snapshot is associated with a revision number.

We used the subversion log tool to determine all of the revisions for a project in a relevant time period. Below is an example of the subversion log output, excerpted from the Tomcat 6.0X project. It shows the log output for two changes: revisions 414965 and 415610.

```
------------------------------------------------------------------------
r415610 | funkman | 2006-06-20 07:12:30 -0400 (Tue, 20 Jun 2006) | 3 lines
Changed paths:
   M /tomcat/tc6.0.x/trunk/java/org/apache/catalina/startup/Bootstrap.java

Remove JMX warning message since java5 is required


------------------------------------------------------------------------
r414965 | fhanik | 2006-06-16 20:12:20 -0400 (Fri, 16 Jun 2006) | 3 lines
Changed paths:
   M /tomcat/tc6.0.x/trunk/java/org/apache/coyote/http11/Http11AprProcessor.java
   M /tomcat/tc6.0.x/trunk/java/org/apache/coyote/http11/InternalAprInputBuffer.java

Revert, if Content-Length: <some large value> is an acceptable work around
```

Revision Assembly

We used the subversion update tool to assemble a particular revision. Below is an example of the subversion update output, also for the Tomcat 6.0X project, updating to the 415610 revision.

```
[tomcat-60X]# svn update -r 415610
U    java/org/apache/catalina/startup/Bootstrap.java
Updated to revision 415610.
```

Revision Build

Once we had a snapshot of a project revision, we used ant to compile the source code and to archive the resulting class files in jar files. Ant is a/the standard build tool for Java software projects. Software developers describe ant build targets in an XML file, usually called build.xml.

However, many ant build files differ in the naming, ordering, and content of their build targets. Therefore, we needed to analyze each build file to determine how to compile the Java code and where to locate the output class files. Below is an example, also for the Tomcat 6.0X project.

```
ant clean build-prepare compile
```

Once we had the class files for a particular snapshot of the project, we archived them into two jar files: a product jar file and a product-tests jar file. If the project clearly differentiated between product code and test code, then we archived the tests in the product-tests jar file - otherwise, we archived everything into the product jar file. Then, we packaged the jar files in a zip file, named according to the revision.

```
[tomcat-60X]# unzip -l product.415610.zip
Archive:  product.415610.zip
  Length     Date   Time    Name
 --------    ----   ----    ----
  2641618  06-20-06 16:42   product.415610.jar
      345  06-20-06 16:42   product-tests.415610.jar
 --------                   -------
  2641963                   2 files
```

Build History

Once we had the list of the revisions and a build mechanism in place for a project, we used a binary search of sorts to decide how far back we could go. We started half way down the list and updated our snapshot to that revision. If that revision could be built with our build mechanism, then we moved our "window" to the bottom half of the revisions. If our build mechanism failed, we moved our "window" to the top half of the revisions. We repeated this process until our window closed and we knew the start revision.

Once we knew our revision boundaries, we used command line scripts to iterate over the revisions. For each revision, the script updated the snapshot to a particular revision. Then, it validated that a build should indeed be attempted - if no pertinent Java files had changed then there was no reason to attempt a build. Finally, if validated, it attempted a build - attempted because a build can fail for many reasons, including compilation errors and missing dependencies.

Below is an abbreviated list of the builds that we were able to reconstruct for the Tomcat 6.0X project.

```
[tomcat-60X]# ll *.zip |awk '{print $9}'
product.389946.zip
product.390055.zip
product.390155.zip
...
product.414906.zip
product.414965.zip
product.415610.zip
```

13.4 Custom Forecast

In order to make a custom forecast, using gnuplot, we needed two inputs: (1) a file containing the time/efficiency data; and (2) a gnuplot specification for fitting the data.

Although our time/efficiency files had more information than this, here's an abbreviated version of one:

```
410234  423920  51.119  0.854   0.855   0.807
410234  426537  59.166  0.766   0.770   0.723
410234  428729  66.171  0.691   0.696   0.642
```

The first two columns indicate the revisions that have been compared. The third column indicates how many days have passed since the beginning of the

change set. The fourth, fifth, and sixth columns indicate the byte, instruction, and method efficiency measurements (respectively).

Once we have our time/efficiency data, we could reference it in a gnuplot specification. Below is the one we used - it matches the mathematical model we proposed in section 7 (except that x is the independent variable, instead of t).

```
f(x)=1 / exp((x - a) / b) + c
a=45; b=20; c=0.1
fit f(x) "/some/path/time-efficiency-data.txt" using 3:4 via a,b,c
```

Using this specification, gnuplot uses a "least-squares" algorithm to "fit" the data and solve for a, b, and c. Below is a snippet from the solution output:

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 36.6014          +/- 2.784        (7.608%)
b               = 35.9966          +/- 5.305        (14.74%)
c               = 0.092494         +/- 0.02859      (30.91%)
```