Analogy Based Prediction of Work Item Flow in Software Projects: a Case Study

Audris Mockus
Avaya Labs Research
Department of Software Technology Research
233 Mt Airy Rd., Basking Ridge, NJ 07920

Abstract

A software development project coordinates work by using work items that represent customer, tester, and developer found defects, enhancements, and new features. We set out to facilitate software project planning by modeling the flow of such work items and using information on historic projects to predict the work flow of an ongoing project. The history of the work items is extracted from problem tracking or configuration management databases. The web-based prediction tool allows project managers to select relevant past projects and adjust the prediction based on staffing, type, and schedule of the ongoing project. We present the workflow model, and briefly describe project prediction of a large software project for Customer Relationship Management (CRM).

Key Words and Phrases: software changes, project schedule, analogy based prediction

1. Introduction

Despite considerable research and practical experience it is still a formidable challenge to understand and accurately predict what will happen in a large software project. We focus on a problem of planning and managing development and testing resource allocation in large software projects. Our goal is to obtain a detailed model and predictions of how work flows in a large software project.

Specific questions that we wanted to answer were:

1. Will the release quality goals be achieved by set date?
2. Is the current software work on track with respect to the situation in past projects.

We model inflow and outflow of work items or Modification Requests (MRs) into and out of the project as a set of queues of various priority MRs. The MRs may arrive from developers adding enhancements or new features, testers or developers discovering problems during development, or customer support team and customers reporting problems found in the field. MRs can get into the queue if they are reassigned from a different project or change their priority. In such case they are removed from the corresponding queue they came from. MRs are dequeued if they are fixed, determined to be duplicates, change priority, or are assigned to a different project.

The history of a project is reconstructed from MR history files that record all changes to attributes of an MR. The MR history indicate the date and time, the person, and the modification to MR attributes including priority, release, ownership, and resolution status. The data collection and processing in the considered project is conducted weekly (implemented as a cron job[1]) before project team meetings.

The availability of project history allows calculation and presentation of vital project characteristics, including work inflow (new MRs) and outflow (resolved MRs) as well as queue length or backlog of MRs. In the considered project many of the critical process steps including project readiness for customer deployment were determined by the size of backlog in various priority queues.

We employed analogy based prediction method to calculate the anticipated work items and the anticipated state of the project at the proposed release date. The past project(s) were suitably transformed and used as a predictor for the ongoing project. This approach allowed us to perform very detailed predictions because complete evolution of the project is reproduced via the analogy transformation. The challenge was to construct a suitable transformation that would lead to accurate predictions.

We start from our motivating questions in Section 2. Section 3 describes ways to obtain data on work items. The model of MR flow for a software project is introduced in Section 4; Section 5 describes the result of applying such models to actual projects; Section 6 considers ways to validate these empirical results, and Section 7 outlines steps needed to model other software projects. We conclude with literature review in Section 8 and discussion.

[1] A cron job is a unix shell command executed according to the specified schedule.

2. Motivation

We set out with the desire to answer the two basic questions stated in Section 1. While similar issues were observed in many projects, we present our findings for one project. The principal concern was whether or not the release date would be met. Would there be sufficient time and effort to complete the features scheduled for the release, including repairing any problems that arose during the development, and fixing existing outstanding problems. To be "ready" the project had to satisfy a number of release criteria expressed in maximal numbers of critical, high, and low priority MRs that had to be satisfied before the product was delivered to customers. These quantities had to be predicted for the anticipated release date. In addition, it was important to track progress in achieving the desired goals at weekly and monthly intervals.

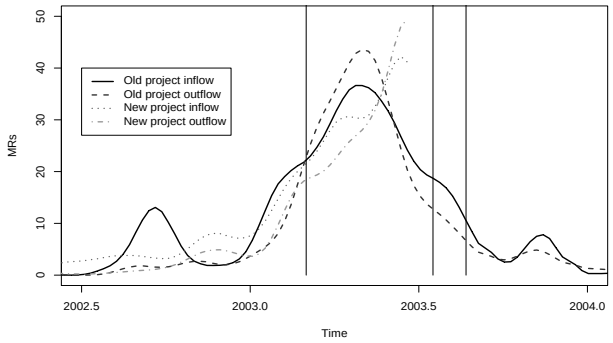

Figure 1. Analogy based predicted project workflow.

Figure 1 shows weekly numbers of new and resolved MRs for two projects. Dotted and dash-dot lines show arrival and resolution of MRs for the ongoing project, while solid an dashed lines show the appropriately transformed past project. Three vertical lines indicate dates at which the initial prediction was done, the proposed release date at the time of the initial prediction, and the current plan for the release date. The proposed release date has slipped by about five weeks since the time of the initial prediction. The purpose of the analogy is to predict the workflow using the transformed past release.

The next section describes background information on using change management and version control repositories to obtain information on software changes that is used to obtain and predict the workflow in a software project.

3. Background

The basic premise of analyzing software changes is that software is created incrementally through a series of work items, each potentially resulting in a change to the source code or documentation. Each incremental change is recorded by a version control and, possibly, problem tracking system. The data typically contains a set of attributes such as the following:

- The identity of the person making the change.
- A short comment written by the author of the change.
- The file(s) changed and the lines changed or the file(s) contents before and after the change.

3.1. Work items in software projects

The purpose of the typical work item in a software organization is to make a change to a software entity. Work items range in size from very large work items, such as releases, to very small changes, such as a single delta (modification) to a file. A hierarchy of changes with associated attributes is shown in Figure 2.

The source code of large software products is typically organized into subsystems according to major functionality (e.g., database, user interface, etc.). Each subsystem contains a number of source code files and documentation.

The versions of the source code and documentation are maintained using a version control system (VCS) such as Concurrent Versioning System [4] commonly used for open source software projects, or a popular commercial system, such as ClearCase. We frequently deal with Source Code Control System (SCCS) [17] and its descendants. Version control systems operate over a set of source code files. An *atomic* change, or *delta*, to the program text consists of the lines that were deleted and those that were added in order to make the change. Deltas are usually computed by a file differencing algorithm (such as Unix diff), invoked by the VCS, which compares an older version of a file with the current version. Included with every delta is information such as the time the change was made, the person making the change, and a short comment describing the change.

In addition to VCS, most projects employ a change request management system (CMS) or Problem Tracking system that keeps track of individual requests for changes, which we call Modification Requests (MRs). Whereas a delta is intended to keep track of lines of code that are changed, an MR is intended to be a change made for a single purpose. Each MR may have many deltas associated with it. Some commonly used problem tracking systems include ClearDDTS from Rational, and the Extended Change Management System (ECMS) [10]. Usually such systems associate a list of deltas with each MR.

Modifications are typically made for one of the following reasons.

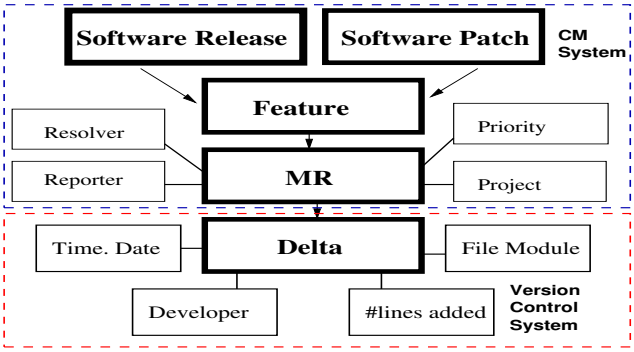

Figure 2. Hierarchy of changes and associated data sources. Boxes with dashed lines define data sources (VCS and CMS), boxes with thick lines define changes, and boxes with thin lines define properties of changes. The arrows define an "is a part of" relationship among changes, e.g., each MR is a part of a feature.

- Repairing previous changes that caused a failure during testing or in the field.
- Introducing new features to the existing system.
- Restructuring the code to make it easier to understand and maintain. (An activity more common in heavily modified code, such as in legacy systems.)

To understand the activities occurring in a software development project, it is critical to know which MRs belong to each of these categories. Fortunately, this information is often recorded in CM systems as a field identifying whether the MR represents a new feature or repairs a problem. Unfortunately, the quality of this classification varies by project. In cases when it is unacceptably low for the purpose of a particular analysis we use an automatic classification technique that uses words in the MR abstract to determine the purpose of the MR [11]. The automatic classification of MR abstracts tends to produce medium to high quality classification as was shown in [11]. For the projects reported in Section 5 the CM systems had high quality attributes identifying class of an MR.

Based on informal interviews in a number of software development organizations within AT&T, Lucent, and Avaya we obtained the following guidelines that are used to divide work into MRs:

1. Work assignments that affect several subsystems (the largest building blocks of functionality) are split into distinct MRs so that each MR affects one subsystem;
2. A work assignment in a subsystem that is too much for one person is further organized into several MRs so that each one could be completed by a single person.

For practical reasons these guidelines are not strictly enforced, so that some MRs cross subsystem boundaries and some have several people working on them.

A group of MRs associated with new software functionality is called a feature. A set of features and repairs constitute a customer delivery, also known as a release. Put another way, each release can be characterized as a base system modified and extended by a set of MRs.

3.2. Organization Specific Work Item Process

In the considered organization the MRs were tracked via customized CRM system that was also its product, i.e., the resulting software of the project was used to track work items. The system involved all aspects of software production and support from items related to customer reports, patches and software modification requests to requirements changes. We investigate the part related to MRs because that is more or less similar to other configuration management systems used in most large software projects. The source code version control system used was ClearCase. The ClearCase comment for software changes included relevant MR numbers.

The MRs may be created by developers adding enhancements or new features, testers or developers discovering problems, or customer support team reporting problems in the field. When or after an MR is created it is assigned, among other things, a priority, release, resolver, and resolution status. As time goes by, these attributes may be changed if, among other things, MRs are fixed, determined to be duplicates, change priority, or are assigned to a different project (we use release and project terms interchangeably).

For our model we needed to capture changes in priority, reassignments to different releases, and various modes of MR resolution. This information had to be obtained by processing MR history: a text field capturing log of MR attribute changes, including MR creation. The log included date, individual making the change, and names and values of the attributes changed. The data collection and processing was implemented as a weekly cron job run before project team meetings.

3.3. The value of analyzing changes

The analysis of software changes has a number of distinct benefits that may not be immediately obvious.

- The data collection is nonintrusive, using only existing data and making analysis possible in commercial projects that are usually under intense schedule pressure and do not have time or resources to collect additional data.

- Long history on past projects is available, enabling comparison to what happened in the past and customization and calibration of the methods to the existing environment. Nonetheless, one must be mindful of changes to the environment and application that make comparisons problematic.
- The information is fine grained, at the MR/delta level. Such fine level data collection on a large scale would not be possible otherwise.
- The information is complete, all parts of software, documentation, test cases that are under version control are recorded.
- The way the version control system is used rarely changes, making data uniform over time.
- Even small projects generate large volumes of changes making it possible to detect even small effects statistically.
- The version control system is used as a standard part of the project, so the development project is unaffected by experimenter intrusion eliminating observer effects.

We believe that MRs are a very rich source of information about software development and that their analysis can evoke rewarding insights. Unfortunately drawing conclusions about characteristics of a project is fraught with challenges. We describe some of these challenges in the following sections. Basic to all of them is that special care must always be taken to obtain information on how version control and change management are used in the project so as not to misinterpret the MR classifications or misunderstand the process used to create, make progress on, and record information about MRs.

The next section describes the project model we used, or the way MRs are injected and resolved in a software project.

4. Work Item Flow Model

We model inflow and outflow of work items or Modification Requests (MRs) into and out of the project. A project can be thought of as a set of queues of various priority MRs. The MRs are placed in these queues when they are created or reassigned from a different project or priority. The MR leaves a queue if it is resolved or reassigned to a different project or priority.

The MRs are created by developers adding enhancements or new features, testers or developers discovering problems, or customer support team reporting problems in the field. Therefore, suitably chosen MRs can represent work on new features, defects detected during development, and defects detected by customers. Only very early stages of the software project, like planning, that require little or no involvement from developers, have no traces in a typical CM and version control systems. It is worth noting, that in the experience of authors, all projects continue to perform to some extent all stages of a project including planning, architecture, enhancements, testing, and repair activities over entire duration of the project. Obviously, the effort for different activities is unevenly distributed over time: for example, testing related activities constitute a larger fraction of total effort spent toward the end of the project.

MRs are resolved if they are fixed, determined to be duplicates, or determined not to represent an issue that needs or can be fixed by changing software.

To simplify later discussion we introduce following notation. Let $in_p(t) = \sum_i source_i(t)$ be inflow, $out_p(t) = \sum_j sink_j(t)$ be outflow, and $l_p(t) = \sum_{s \le t} in_p(s) - \sum_{s \le t} out_p(s)$ to be queue length at time period t. Here parameter p denotes the queue identified by project id and priority, $source_i(t)$ denotes the number of MRs arriving from source i at time interval t, $sink_j(t)$ denotes the number of MRs departing to sink j at time interval t, and $s \le t$ denotes all non-overlapping time intervals s up to and including t.

All the considered quantities are directly observed (by processing MR history data as described in Section 3.1) for past and ongoing project up to the current time (there are two notions of current time: when the initial prediction was done and when this description was revised four months later). It is worth noting, that simply displaying MR inflow, outflow, and queue lengths can help project management see immediately the "big picture" of the project, including the current and historic rate at which new problems arrive and are solved, as well as the size of the work backlog and whether or not it is decreasing over time. Such information allows timely decision regarding staffing allocation and corrections to project schedule.

Our main objective, though, is to provide the same form of transparency going forward in time. More specifically, we want to estimate the size of various MR queues at the proposed release date at least few months or more in advance. Furthermore, we want to predict the development, i.e., inflow and outflow of MRs, up to the release date, so that the project could track its progress by following the predicted path.

While it is possible to model inflow and outflow of MRs parametrically, see, e.g., [14], we chose to design an analogy based method, i.e., we assume that a suitable transformation of a past project can yield accurate predictions for the ongoing project.

More specifically, we consider following transformation for the incoming MRs:

$$in_o((t - Center + Lag) * Scale + Center) = in_p(t) * C + error(t), \quad (1)$$

where in_o represents predictor of incoming MRs for ongoing project and in_p are incoming MRs for the past project used in the analogy, Lag represents the time lag between onsets of the projects, $Scale$ represents the difference in time duration between projects, $Center$ is the time around which the project is rescaled (we typically choose the start of system test), and C is MR inflow adjustment based on the size and type of the project and on the quality of past releases already in the field that generate MRs that need to be resolved in the ongoing project. MR resolution is predicted using a similar transformation:

$$out_o((t - Center + Lag) * Scale + Center) = in_p(t) * Prod_o/Prod_p + error(t) \quad (2)$$

where $Prod_p$ is team productivity in project p. It may be approximated by staffing levels or could also be estimated from existing data.

The transformation parameters Lag, $Scale$, $Center$, $Prod$, and C can be estimated or overridden by the analyst (project manager). In our implementation we left for project manager to determine following parameters because that was information they could accurately provide.

1. $Prod$ is the productivity of the team. We used staffing levels of the project as a substitute for team productivity. While it is a rough approximation, we did not need to obtain more precise productivity assessments, mainly because the same people were involved in past and current projects.

2. $Scale$ is the duration of the project. In our case it was easier for an expert to anticipate the duration of the project than to determine its start date. We left an option for this parameter to be estimated from existing data.

3. $Center$ is the time around which to rescale the analogous project. We used start of system test because at that time MR activity peaked and because project managers were reasonably confident about the anticipated system test start date.

After the parameters have been estimated or set, the transformed past project can be used to calculate the size of queues at the release date and to track progress of the ongoing project.

4.1. Estimating transformation parameters

The basis for finding optimal parameters are equations 2 and 3. One can simply look for the parameter values that minimize sum of squared errors $\sum_s error^2(t)$. We minimized the error of predicting the backlog of MRs at the release date. The error is based on cumulative inflow and outflow of MRs, where cumulative inflow is simply $cin_p(t) = sum_{s \le t} in_p(s)$. More specifically, the minimized function was:

$$\sum_{s<=t_0} (cin_o((s - Center + Lag) * Scale + Center) - cin_p(s) * Prod_o/Prod_p * Scale)^2, \quad (3)$$

where $(t_0 - Center + Lag) * Scale + Center$ is equal to current time at which the prediction is performed. There are no observed data beyond that point for the ongoing project. The cumulative counts of the past project $cin_p(s) * Prod_o/Prod_p$ have to be multiplied by $Scale$ because the time intervals are stretched (this is a discrete version of the Jacobian when integration variable, in this case time, is transformed).

Some of the parameters can be estimated more directly: for example, the parameter $Prod$ may be estimated by calculating the number of MRs solved by the project team, and the members of the team can be obtained from MR data for the ongoing project or from project management for the projects that have not started.

Another parameter C can be broken down into parts according to the sources MRs are coming from. Lets recall that $in_p(t) = \sum_i source_i(t)$. The three principal MR sources are:

1. Field or customer reported MRs ($source_f$). Such MRs have a relatively constant flow punctuated by introduction of a new major release causing a flood of new MRs in first few months. The timing of introduction is known and can be included in the model.

2. Developer generated MRs working on the code ($source_d$). The intensity of such MRs depends on the number of developers working on the release. More developers tend to create more MRs per unit time, so this source can be adjusted by the ratio of developer staffing between past and current project. In fact, all the developers are known for the past and current projects, so we can directly calculate the ratio of team MR generation productivities for past and present projects.

3. Testers or verifiers generating MRs in the course of testing ($source_v$). More testers generate more MRs per unit time, so this source can be adjusted by the ratio of tester staffing between past and current project. As with developers, all testers are known in the past and current project, so we can directly calculate the ratio of team MR generation productivity for the past and present projects.

Separating the parameter into three parts we can rewrite Equation 2 as

$$in_o((t - Center + Lag) * Scale + Center) = \quad (4)$$
$$source_{f,p}(t) * C_f + (source_{d,p}(t) +$$
$$source_{v,p}(t)) * C_d + error(t),$$

The parameters $Prod$ can be estimated similarly to estimating coefficient C_d, except developer productivity when solving MRs (rather than creating MRs) has to be used here.

4.2. Assumptions

To perform accurate prediction the model has to learn (incorporate information) from completed projects. Basically, we assume that the organization has completed some projects that are similar to the project to be predicted, e.g., similar development team, process, and software functionality. This is to ensure that it is possible to transform a previous project so it would resemble the current project.

While the project can be predicted before it's start, all transformation parameters have to be chosen by experts. After the project has started, some or all of the parameters can be estimated from the data of the ongoing project (see Section 4.1.) The prediction can become more accurate in more advanced stages, because more accurate estimates of the transformation parameters are available. As the results in Section 5 indicate, the prediction was quite accurate performed when less than 20 percent of work on the project has been completed.

Obviously, we assume that the development work is recorded via MRs, which may be untrue in smaller or more collocated projects.

An interesting issue is the impact of the monitoring on the course of the project. If the monitoring causes project management to add staff or to drop functionality in some release, the staffing and functionality changes must be accounted for when performing predictions based on that release.

5. Empirical Results

The project under consideration involves a large Customer Relationship Management (CRM) system that was developed over 10 years, consisting of more than 5 million lines of source code. About half was C++ code, about quarter was Java code, and the remaining code involved C and other languages. More than 100 developers were involved in this project over time and there were more than 30K MRs. The effort spent on predicted and used for prediction projects ranged from 1K to 2K person months. Durations of the projects are visible from the presented figures.

The development team was based in four primary locations in the United States and a site in Australia. This distribution of teams was due to history of acquired companies that were consolidated to work on this CRM product.

The considered project divides MRs into critical, high, and low priorities. Critical MRs indicate issues that prevent product from functioning, i.e., blocks major functionality, or involves data loss. High MRs are related to less severe issues that are likely to be observed by a customer, e.g., blocking minor functionality, incorrect documentation, noncritical memory and resource leaks. Low MRs are enhancement and convenience/aesthetic or spelling and appearance issues and inconsistencies found by code analysis tools (e.g. Purify). More details on the process used are in Section 3.2.

For our analysis we present results related to low priority MRs because the backlog of these MRs posed challenges to the project at the time of analysis and because the results regarding other types of MRs are similar and would make this report repetitive.

5.1. Estimating transformation parameters

We chose to discretize project time into calendar weeks because that was the desired resolution for the progress reports and backlog predictions. Consequently, the time periods t and s in Equations 2,5,3 represent calendar weeks.

One of the critical choices is the selection of past projects that are reasonably similar to the predicted project. For example, to predict a major release it makes sense to use past major releases, and, similarly, minor releases are likely to mirror past minor releases. Some releases are done mostly for internationalization, some to add an new computing platform or to integrate with a different external software provider. In our case we were predicting a feature (major) release and we chose to use another major release and a platform release to perform predictions based on recommendations of the project manager. We will present the results based only on the major release, mostly due to space constraints.

Consistently about 90 percent of all MRs in four inspected releases were raised by developers and testers and the remainder came from the field in steady streams similar in both projects. This information provides us with estimates for Equation 5: $C_f = 1$ corresponding to the coefficient for inflow of field MRs. We chose to use ratio of project staffing of 80 percent for ongoing project as an estimate for C_d. We also chose as 80 percent the MR resolution ratio $Prod_o/Prod_p$ in Equation 3. The parameter $Center$ was chosen to correspond to system test start date of the release. These parameter estimates were obtained in discussion with project managers. The uncertainty about the values of these parameters was harder to elicit but ranges of at least plus or minus 10 percent represent an "educated

guess" of their variability.

The remaining parameters $Scale$ and Lag were estimated by minimizing Equation 4. Optimal values were the $Scale = 1$ indicating that both projects follow similar time line, and $Lag = 50$ indicating that ongoing project schedule is shifted by 50 work weeks. The same values were suitable for MR outflow as well. As was mentioned earlier, the prediction was performed at the time indicated by the first vertical line in Figure 1. Optimal parameters at the time of writing were $Scale = 1.1$ and $Lag = 51$ indicating slightly longer duration of the predicted project. The results are reported only using the estimates of the initial prediction.

Figure 3 illustrates cumulative MR arrivals and closures for the ongoing and past projects. The total number of MRs is normalized to add to 1000 for confidentiality reasons.

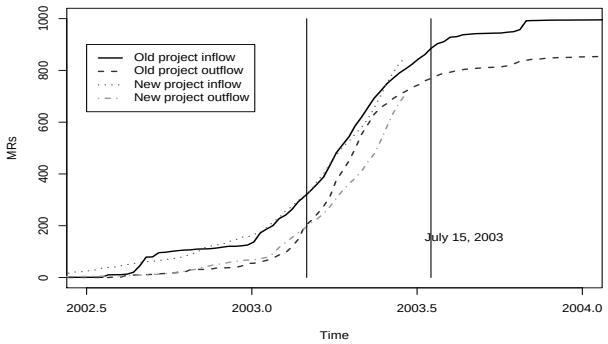

Figure 3. Cumulative numbers of MR inflow and outflow.

The solid line shows cumulative MR inflow for the old release that has been transformed as described above. The dashed line shows outflow for the same release. Dotted and dash-dot lines show inflow and outflow for the ongoing release. The planned release date is marked on the plot. At the time of writing the planned release date has been changed as shown in Figure 1.

While the new project is evolving more gradually, the two projects behave similarly up to the time of prediction. While new MR generation has been as predicted up to the present time, the MR resolution lagged considerably since the time of prediction as can be seen by comparing dash and dash-dot lines.

5.2. Predictions of backlog

Currently the prediction is implemented as a web based tool where an expert (project manager) fills a form by selecting relevant past release and suitable transformation parameters including an option to optimize some or all of these parameters. Upon submittal of the form the results are calculated on the server within a short time (depending on the amount of optimization but not exceeding one minute).

There are three parts on the presentation page:

1. The table of predictions for the number of unsolved MRs at the release date that includes currently open MRs, MRs that will arrive minus MRs that will be resolved before the release date. These numbers can be directly obtained from Figure 3. The vertical gap between the inflow and outflow curves for a release shows the backlog of MRs at that particular moment in time. The number of MRs that will arrive is the difference in hight of the solid line between present and the top of the curve, the number of solved MRs is the difference in hight of the dashed line between present and the GA date. Yet another important number is the number of new MRs after the release date, because it describes the number of problems a customer may see if the product is deployed at the proposed date.

2. The second item on the presentation page is Figure 3 for the chosen releases and transformation parameters. In case the parameters are chosen by an expert this figure provides visual feedback to determine if the transformation parameters are properly selected.

3. The last item presents the progress of the project with respect to past prediction in order to make decisions about schedule, staffing, or recalibrating the prediction with more recent information. While the cumulative counts in Figure 3 clearly show the backlog and the total number of remaining and past MRs, it is not very helpful illustrating ongoing activities. Figure 1 can better illustrate weekly activities by showing how many MRs were created and resolved during a week.

Because the project is not finished at the time of the writing, it is too early to judge the accuracy of the predictions, however understanding of the workflow has proved to be very helpful to project managers by quantifying the capabilities of the development process, by providing comparisons to past projects, and by presenting detailed predictions of remaining work and, in the later project stages, the remaining defects for various choices of release dates. This allowed project managers make informed decisions about realistic release dates when dealing with unanticipated contingencies related to staffing changes and to make trade-offs between earlier release dates and lower number of defects remaining after release.

6. Validation

There are several threats to validity. First, is the fact that our assumption that a past project can be transformed to resemble current projects may be flawed. Only ongoing investigations of this and other projects can bring a clear answer to this question. Second assumption is that the actual activity in the project is reflected in the flow of work items captured by the configuration management system. We found

evidence of that in many of the projects we studied, and, in particular in the project described here. However, it is important to ascertain this assumption if a similar method is to be applied in a new project. In fact, it is not essential that all activity would be captured in MRs, as long as the activity leaving MR traces represents a reasonably stable proportion of all activity in the project.

Other set of validity threats relates to the way the proposed analogy-based method is executed. In particular, the choice of unsuitable past projects, inappropriate choice of transformation parameters, and unanticipated events, including changes to the development process and the way MRs are created, prioritized, or assigned to a project could throw the predictions off target.

Another issue is seemingly arbitrary precision (many significant digits) of the prediction because after the transformation one gets a complete history of the past project that now describes the future of an ongoing project. Such detail and precision does not necessarily lead to higher accuracy. We try to counteract this issue in two ways. The first approach is to decrease precision by smoothing the history of the past project, so that only a general trend remains as illustrated in Figure 5. The second approach recommends project management to consider predictions obtained with a range of plausible parameter values and using several past projects. Figure 4 illustrates the impact of increase and decrease of the estimated reporting productivity coefficient (represented by staffing levels) by ten percent. The illustration shows relatively large bounds increasing over time, and the actual inflow (dotted line) has stayed within bounds and close to the prediction (dashed line) for more than four months after the date (indicated by the first vertical line) the prediction was performed. The second vertical line shows planned release date at the time of prediction.

It should be noted, that the Lag and $Scale$ coefficients were not reoptimized for these changes unfairly increasing the bounds. However other parameters are uncertain as well, potentially increasing the bounds. We are working on ways to obtain statistical 90 percent probability error bounds.

In the future, we expect such informal consideration to be automated using, perhaps, a Bayesian model.

Of course, from the project planning perspective the ultimate validity is the accuracy with which the backlog and entire project evolution are predicted.

7. Application to other projects

The utility of the presented model in other projects can be shown if the model is used in project planning or project understanding. In this section we outline the steps needed to apply the model in a software project. There are four basic stages: change data extraction, change data validation, and

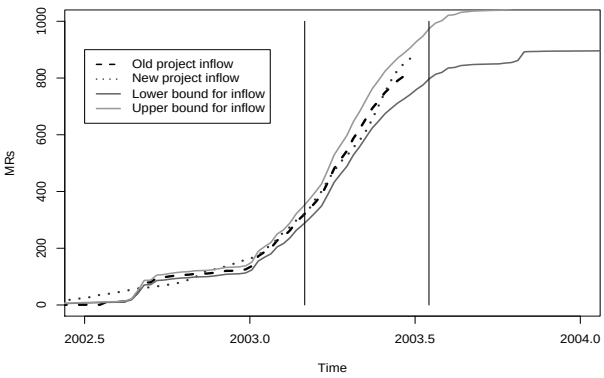

Figure 4. The inflow prediction bounds obtained by varying the C_d coefficient by ten percent.

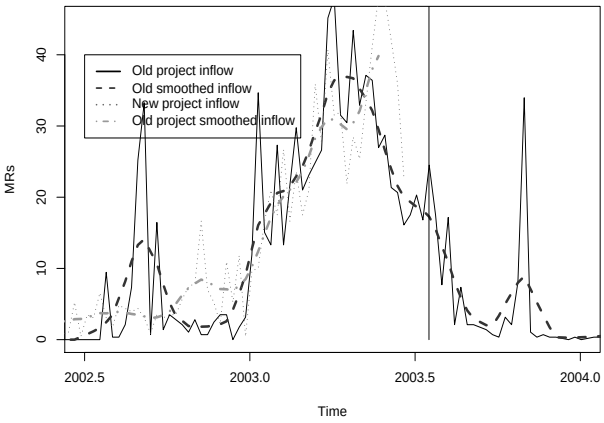

Figure 5. Illustration of observed weekly variabilities of MR inflow.

change and project modeling. In the data extraction stage access to the project systems is obtained and raw change data is extracted. In case of home-grown tools, it may be necessary to interview a person responsible for tool support to understand the structure and functionality of such systems.

The most important is data extraction and validation step. It leads to history of attribute changes for each MR. Some attributes may need to be aggregated if they are equivalent from the project model point of view. For example, the project recorded five priority levels, but decisions were made based on three levels described above where two of recorded levels map into "critical" and another two into "low" priority MRs. The quality of attributes is then assessed and un- or auto-populated attributes and remaining system generated artifacts are eliminated. When engaging a new project it is important to interview a sample of devel-

opers and testers. The interview involves review of recent changes done by the interviewee to to illustrate the actual development process and to understand/validate the meaning various attribute values.

The change data may then need augmented to improve the quality of important attributes. Work in [1] describes how to estimate MR change effort, estimation of change purpose is described in [11] or of change risk in [12]. In the studied project we had to improve the quality of a number of attributes. For example, the history file identified people in three ways: full name, login, or database key. We had automatically to recognize the format in each case and produce a unified identification for modeling purposes. Many other fields had dual formats: textual string or database key.

Once the relevant information on software changes is obtained, a work item model should be inspected for historic releases. Investigating these different projects it is important to note the variability and any patterns of MR inflows, outflows, and queue sizes.

Finally, the predictions for ongoing or planned release and progress reports for ongoing release can be produced as described in Section 5.2.

8. Related Work

Previous work [1, 11, 13] has identified version control and problem tracking databases as a promising repository of information about a software project. We have created methods and tools to retrieve, process, and model such data at the fine level of Modification Requests (individual changes to software) in order to understand the relationships among process/product factors and key outcomes, such as, quality, effort, and interval.

The work in [14] proposes a model based on the concept that each modification to software may cause repairs at some later time, investigates the model's theoretical properties, and reports its application to several projects in Avaya. Model's basic premise is that people involved in a project leave traces of their work in the form of modifications to the artifacts on which they work.

Analogy based estimation methods (see, for example, [18, 19, 9]) rely on collecting a variety of metrics about past projects and then using that data to identify the most analogous project(s) and construct the predictor (typically project effort) from one or several candidates. It may be possible to consider the workflow as being one of such high-dimensional metrics and then use techniques for identifying analogous projects available in the literature. We, however, focused on more detailed prediction of workflow, including MR backlogs at certain dates, and on incorporating current information from an ongoing project.

Typically size, type, and other parameters for the past projects are used in analogy based estimation. In our case we have entire workflow history rather than a few summary parameters. However, it is essential to transform the workflow history of the past projects to predict future project. Furthermore, the choice of most similar projects for the analogy is often treated as a formal problem. We chose a simple expert-based estimation due to complex nature of the parameter space and because we had history of past projects accomplished by the same team. More generally, our hypothesis is that a single past project by the same team on the same product may be sufficient to perform a useful prediction. However only more extensive application of the approach can confirm that conjecture. It remains to be seen if the approach may be suitable for prediction across teams or product types.

The investigation of prediction in the final stages of a software project was considered in [16], where open defects, code churn, test pass rate, and defect find rate metrics from past projects are compared to the metrics of the current project. Some quantitative models for managing software development are discussed in [7]. In our case, we focused on prediction of MR backlogs at some future date because they were a part of release quality criteria.

More generally, this work relates to two areas in software engineering: cost and schedule estimation and risk assessment. The software cost estimation may be roughly organized into expert and algorithmic techniques to estimate software cost and schedule.

The expert based techniques are typically best suited for projects that are not too different from projects completed in the past and where the estimator has extensive experience of estimation with these past projects (a good review of expert estimation techniques may be found in [8]). The main drawback is the subjective and non-transparent nature of the estimation process that makes it harder to justify the estimates. In analogy based estimation we use expert judgment on key factors that are well know to project managers, like staffing, release dates, and most similar past projects. Other parameters may be estimated from an ongoing project data or overridden by an expert.

Algorithmic techniques such as COCOMO [2, 3] may be used if the key predictors, such as size of the project, can be reliably estimated in advance and calibrated with past projects. The main drawback is that the size of the project (an input to the algorithm) may be more difficult to estimate than the cost (the output of the algorithm).

The risk assessment literature covers a number of issues, but the part most related to our work predicts the number of defects remaining in software during testing [15, 5, 6]. There are two key differences with our work: we do not predict defects based on observed defect counts in an ongoing project, but rather predict defect (and other MR) arrival based on observations in similar projects. The second difference is that testing models assume that the software does

not change during testing, which is not an accurate assumption.

9. Discussion

While our focus was fairly narrow: to predict work backlog at the release date and provide benchmarking for the ongoing release, the workflow model can be applied in more general settings. For example, it could be used to predict the number of defects after the release and used to decide when to stop testing by predicting the number of defects remaining in the code base after a specified date.

Since the model uses data available in many large software projects it is appealing for commercial software settings where all process overhead, including time-consuming data collection, are treated with skepticism.

In addition to providing a framework for answering a number of important project planning questions, the workflow model may help gain better insights into completed projects and may help a software organization better understand its strengths and shortcomings. One significant benefit that we expected and observed, was better understanding of a software projects constraints even by people who directly involved in development.

The separation of incoming MRs by source into development, testing, and customer MRs can be used to check if novel development technologies and practices including code inspections and product line engineering lead to detection of defects in earlier phases of the development process.

While it remains to be seen how widely the work flow model may be applicable, the reported early results show initial promise. Authors' belief is that each project is typically actively managed and the actual workflow is the result of the intense effort by various parties to keep things going. Workflow simply captures a simple summary of this complexity of decision making. Additional information gained from workflow monitoring and comparison to past projects may help make some project management decisions easier and more quickly and makes it easier to learn from past projects, but is unlikely to eliminate the need to make the decisions nor substitute for intelligence needed to make them.

Acknowledgments

We would like to thank all the people in Avaya, Lucent, and AT&T who provided information directly (via interviews) or indirectly (by working on the products under study.) In particular we thank J. Maranzano, D. Sokoler, and others for providing insight on development process, development tasks, project management and other aspects of the studied software projects.

References

[1] D. Atkins, T. Ball, T. Graves, 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*, 28(7):625–637, July 2002.

[2] B. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[3] B. W. Boehm, B. Clark, E. Horowitz, and et al. Cost models for future software life cycle processes: Cocomo 2.0. *Annals of Software Engineering*, 1(1):1–24, November 1995.

[4] P. Cedeqvist and et al. *CVS Manual*. May be fond on: `http://www.cvshome.org/CVS/`.

[5] S. R. Dalal and C. L. Mallows. When should one stop testing software? *Journal of American Statist. Assoc*, 83:872–879, 1988.

[6] A. L. Goel. Software reliability models: Assumptions, limitations and applicability. *IEEE Trans. Software Engineering*, SE-11(12), 1985.

[7] K. Huff, J. Sroka, and D. Struble. Quantitative models for managing software development processes. *Software Engineering Journal*, 1(1):17–24, 1986.

[8] M. Jørgensen. A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, 2002. submitted.

[9] A. Lefteris and I. Stamelos. A simulation tool for efficient analogy based cost estimation. *Empirical Software Engineering*, 5(1):35–68, 2000.

[10] A. K. Midha. Software configuration management for the 21st century. *Bell Labs Technical Journal*, 2(1), Winter 1997.

[11] A. Mockus and L. G. Votta. Identifying reasons for software change using historic databases. In *International Conference on Software Maintenance*, pages 120–130, San Jose, California, October 11-14 2000.

[12] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, April–June 2000.

[13] A. Mockus and D. M. Weiss. Globalization by chunking: a quantitative approach. *IEEE Software*, 18(2):30–37, March 2001.

[14] A. Mockus, D. M. Weiss, and P. Zhang. Understanding and predicting effort in software projects. In *2003 International Conference on Software Engineering*, pages 274–284, Portland, Oregon, May 3-10 2003. ACM Press.

[15] J. Musa, A. Iannino, and K. Okumoto. *Software Reliability: Measurement, Prediction, Application*. McGrawHill, New York, 1987.

[16] T. Pearse, T. Freeman, and P. Oman. Using metrics to manage the end game of a software project. In *Software Metrics Symposium*, pages 207–215, 1999.

[17] M. Rochkind. The source code control system. *IEEE Trans. on Software Engineering*, 1(4):364–370, 1975.

[18] M. Shepperd, C. Schofield, and B. Kitchenham. Effort estimation using analogy. In *18th International Conference on Software Engineering*, page 170, Berlin, GERMANY, March 25 – 29 1996.

[19] F. Walkerden and R. Jeffery. An empirical study of analogy-based software effort estimation. *Empirical Software Engineering*, 4(2):135–158, 1999.