AN ESTIMATE CAPTURE AND MONITORING TOOL

A COMPUTER AIDED SOFTWARE ENGINEERING TOOL FOR ESTIMATING AND MONITORING PERSONAL SOFTWARE DEVELOPMENT PROGRESS

by

Dong Shao

A thesis submitted in conformity with the requirements

for the degree of Master of Science

Graduate Department of Computer Science

University of Toronto

Your file Votre référence
ISBN: 0-612-84201-0
Our file Notre référence
ISBN: 0-612-84201-0

Abstract

An Estimate Capture and Monitoring Tool:

A Computer Aided Software Engineering Tool for Estimating and Monitoring Personal Software Development Progress

Dong Shao

Master of Science 2003

Department of Computer Science

University of Toronto

Release Planning is a management framework for commercial software vendors to estimate and track software releases. We developed an Estimate Capture and Monitoring Tool (ECMT), a computer aided software engineering tool for estimating and monitoring personal software development progress, to support the Release Planning management framework at an individual level.

The goal of ECMT is to provide a tool aid for developers applying Release Planning in order to determine if the Release Planning approach is practicable. ECMT's features are divided into three categories. The tracking time feature records time spent on hierarchically organized tasks, and provides graphical views of time spent. The estimation feature logs and analyzes the estimations of tasks as projects proceed. The combination of estimation and time tracking make ECMT distinct from related works by providing up-to-date progress measurement of projects.

Acknowledgements

First of all, I wish to thank my supervisor, Professor David A. Penny, for his valuable guidance. David put much effort into my work, and his helpful suggestions improved it significantly. I would also like to thank my second reader, Professor Dave Wortman. His input has greatly benefited this thesis.

Many thanks to my friends in DCS for their generous support. Discussions with them inspired me in many ways and suggestions from them are highly appreciated. Special thanks go to my wife, Wei Zhang. The work in this thesis would not have been possible without her input. She shared every bit of my prosperous and hard time.

I also deeply appreciate my parents and sisters in China. They have always been there for me and encourage me.

Finally, everyone in our department deserves my thankfulness. They provide the best environment for research and study I have ever met.

Table of Contents

1 Introduction

Penny [Penny 2001; Penny 2002] proposed Release Planning as a management framework for commercial software vendors to estimate and track software releases. A successful commercial software vendor produces new software releases that provide features required by customers. Product management in the organization should be aware of the needs of customers and business, and make release plans that include new features to be delivered in the product by the release date. The software development division in the organization should estimate the schedule, track development and implement all features before the release date.

Inaccurate schedule estimation and schedule slippage are common problems in many software projects [van Genuchten 1991; Ma 2000], and Release Planning also needs to deal with these problems. In part, these problems are due to individual software developers [Brooks 1995; Humphrey 1995]. In Release Planning, a release plan specifies features required in the current release. The development group will break the features down into appropriately sized parts and assign them to developers considering their skills. Developers measure the work effort of each feature. Then development managers collect data from each developer, and summarize it as the overall estimate for the release. However, some software developers are not willing or able to make accurate schedule estimates. Furthermore, even if they make an accurate initial estimate, they often fail to monitor their progress consistently. Without this monitoring information, it is hard for developers to update estimates to reflect their current development progress, thus their estimates and hence the overall project estimate will become inaccurate.

Even when developers are able to make accurate estimates and corresponding plans, sometimes they are behind schedule due to a lack of work efficiency. There are many causes for low work performance. For instance, a bad work environment, low morale, outdated hardware and software tools, lack of domain experience, and so on. A lack of personal time management accounts for some developers' low work performance. We characterize the absence of personal time management as unproductive days, last minute rushes to meet a deadline, and infeasible plans. When a schedule slips, developers should identify the reason. If the reason is a lack of personal time management, they need to improve their time management skills.

Continuous and quick feedback has been considered an effective treatment for schedule slippage [Beck 2000]. By taking measurement of development progress as feedback, developers can determine whether they are on schedule or not. If they find their estimates are not true, they can modify them to be more accurate. If they are behind schedule due to lack of time management, the feedback will motivate them to work more efficiently and improve their time management skills. In this way, feedback helps to avoid personal schedule slippage.

Trying to use continuous and quick feedback to deal with personal inaccurate estimation and schedule slippage in Release Planning, this thesis develops an Estimate Capture and Monitoring Tool (ECMT), a computer aided software engineering tool for estimating and monitoring personal software development progress, to support the Release Planning management framework at an individual level. ECMT supports estimating and monitoring during development by allowing developers to make schedule estimates and record actual time spent on each task as feedback. What makes ECMT unique from existing software tools is that it combines estimation and monitoring. This combination provides up-to-date progress measurement of projects,

hence developers can be aware of their development status better.

1.1 Estimation in ECMT

While working on some intellectual work, for example, reading, writing, designing, developing software, people can get into an ideal work state where they just concentrate on the work totally, work fast and effectively, and the work goes smoothly as if no effort is required. In that way, people are even unaware of the passage of time. Some psychologists call this state "flow". DeMarco named it *uninterrupted time,*[DeMarco and Lister 1987] since this state is easily interrupted, for example, by phone calls, friends dropping by or something similar. The *uninterrupted time* for the software developer is the actual work time. We call it *dedicated time* in this thesis, and use it to measure the work requirement and work capacity (chapter 2).

There are two kinds of estimates in ECMT: work capacity estimates and work requirement estimates. Usually, developers are unlikely to spend eight work hours on one project a day. Because ECMT is project-oriented, we only consider the *dedicated time* for each project, which influences the project's schedule. ECMT requires developers to estimate their *dedicated time* for a project in one day, rather than using eight hours a day to calculate work capacity. We calculate work capacity of a developer in a period by multiplying workdays with the estimate of the dedicated time in one day.

In ECMT, we assume that development managers divide a project into small development tasks, and assign them to developers. ECMT asks developers to estimate the *dedicated time* required for each development task. For example, a developer estimates that a print function in an account application needs 40 hours. After summarizing the *dedicated time* of each task, the developer will get his or her work requirement estimate for the project. By comparing the work requirement estimate and

the work capacity estimate, the developer is able to know whether the current estimate of work requirement is less than work capacity.

We use statistical estimates in ECMT rather than single value estimates. Single value estimates are common in most situations, for example, a developer, Bob, predicts that he needs 40 hours to develop an XML-parsing software component. However, since estimation is predicting or guessing something, no one has 100% confidence that the estimate is accurate. An estimator might intend to say, "I think this task has a 50% probability of being completed in 35 hours, and 50% probability not. However, I have 80% confidence that it will be achieved in 40 hours." Therefore, in ECMT, we use stochastic variables to make estimates, and assume that distributions of variables are Normal distributions (section 4.2.1). This estimation method is more complex than the single value method, but it can answer some practical questions, such as how much confidence a developer has to complete all tasks before the delivery date, how many workdays are required for 80% confidence, and so on.

1.2 Learning time management skills

For software developers, time management skills mean accurate estimates, using time in an efficient way, and achieving the project on schedule. The following are the assumptions for improving developers' time management skills used in the ECMT.

1. People tend to have consistent patterns for using time. As for spending time, one inclines to have the same amount of working time day after day, week after week.
2. By recording the work time spent on tasks, developers can collect historical data to know their work efficiency and how much time a development task will take. When estimating new similar tasks, they can use these data to make better estimates.

3. Based on estimates, developers are able to devise detailed plans for projects. These plans suggest the amount of work they should do in one certain day, and ensure that they can accomplish their projects if they follow the plans strictly. In order to keep estimates accurate and plans feasible, developers need to monitor their progress, which means to track their work time and check time spent records against their plans. Only when developers monitor their progress, can they know whether their estimates are accurate and plans are feasible. Hence, they can update the estimates and plans. On the other side, the time log feedback motivates developers to work hard to follow the plans and estimates, as well as provides confidence in their ability to deliver a project on schedule.

1.3 Logging time and estimates

Keeping a time log on paper without the help of tools is inconvenient. Developers have to write down each step of an activity, and need good skills to organize the data format for easy recording and reviewing. Some books[Humphrey 1997] provide the time log table, but it still takes time to learn how to use them. Moreover, it is hard to enforce their usage. Updating estimates also needs some effort, it requires developers to review the time log and write down new estimates. Updating estimates is important, since if the estimates are not updated for a long period, they become outdated.

After recording time and estimates, analyzing them requires much effort. Manually summarizing the time spent on each task or each day, comparing estimates with the time log to decide whether the project is on schedule or not, and adjusting estimates based on historical data may make many developers abandon the effort.

We developed ECMT to assist developers in solving such problems. The user is

only required to input estimates and record the start time and stop time of an activity. ECMT will track work time and analyze estimates. In addition, it provides graphical views and text summaries for analysis.

1.4 Thesis overview

This thesis is composed of six chapters:

Chapter 1: This first chapter introduces ECMT by providing some background, and by explaining what it is, its motivation.

Chapter 2: This chapter gives an overview of the Release Planning approach.

Chapter 3: This chapter explores research relating to ECMT. In particular, personal software process, software estimation, general time management, and some available time tracking tools are examined.

Chapter 4: This chapter introduces the ECMT concept in details. Tracking time, estimation, and combination of estimation and monitoring features are designed in ECMT for supporting Release Planning approach.

Chapter 5: This chapter provides details on ECMT features and how they are implemented.

Chapter 6: This chapter discusses the author's experience with ECMT and feedback of colleagues who used this tool.

Chapter 7: The final chapter draws some conclusions, and discusses directions for future research.

2 Release Planning

This chapter gives an overview of the Release Planning methodology.

Release Planning is a management framework for commercial software vendors to develop packaged software, proposed by Penny [Penny 2001; Penny 2002]. Aimed at small-size and middle-size fast-paced commercial software vendor organizations where there is little time or management focus to enhance the process, Release Planning provides a simple yet elegant approach to aid vendors in making better project plans and tracking development. Release Planning concentrates on the coding phase in software development. This thesis is based on Release Planning, but it concentrates on the personal level.

A successful commercial software vendor produces new software releases that provide the features required by customers. Product management in the organization should be aware of the needs of customers and make release plans that include new features to be delivered in the product by the release date. The software development division in this organization is supposed to implement the features by the release date. One of the goals of Release Planning is to improve the communication between the product management and software development organizations.

2.1 Effective coder days

By definition, eight *uninterrupted* (section 1.1) hours compose one *effective work day*. If it is for coding new features, we call it an *effective coder day*. Effective coder days are used to measure the work requirement (section 2.3) and work capacity (section 2.2) in Release Planning.

2.2 Work factor

Body-present time is the period of time during which a person is available in the work place. It includes uninterrupted time and interrupted time, such as meeting, chatting, having coffee, phone call time, and so on. In Release Planning, *dedicated time* is the uninterrupted time for coding new features in some release. The *Work factor* shows the ratio of dedicated time for coding new features to body-present time.

$$WorkFactor = \frac{DedicatedTime}{Body\_presentTime}$$

Work factor is a helpful parameter for software developers. By collecting time-spent data, developers can learn their work factor, and thus can predict the available amount of dedicated time in a certain period. We call the amount of dedicated time for a project *work capacity*.

Assume that a developer, Tom, has work factor 0.45, and he is working on only one project. If there are 40 workdays before the delivery date, then his work capacity is $0.45 \times 40 = 18$ effective coder days.

2.3 Estimation

A release plan specifies the features required in the current release. The development group will break them down into appropriate sized parts and assign them to developers considering their skills. Developers measure the work effort of each feature by dedicated time, which is called the *work requirement*. Then development managers collect data from each team member, and summarize it as the overall estimate for the release. Such an estimating process is not very complex, and most people do not need extra training.

This bottom-up estimation approach pays much attention to personal difference.

This difference is so large that one programmer may have ten times the performance that of another [H. Sackman 1968]. It is quite possible to get an inaccurate estimation when ignoring this difference.

2.4 Capacity constraint

After completing a project, work requirement must have been equal to work capacity, since we measure them both by the effective coder days spent on the project. It is a de facto result. However, during the course of the project, we can only make estimates for work requirement and work capacity. If work requirement is less than work capacity, we probably have enough human resources for this release. If not, we are at risk of schedule slippage.

2.5 Stochastic Capacity Constraint

The precise capacity constraint is not suitable for estimating in advance. Release Planning uses stochastic variables to make estimates rather than using single value estimation. We will discuss it in chapter 4 in detail.

2.6 Update release plan regularly

If we make estimates and plans only at the beginning of a project but do not update them, they will no longer reflect development progress after a period. From a single developer's view, updating estimates means re-estimating the remaining work, and the work factor. From a project's view, it needs to summarize data from developers and compare total remaining coding capacity with the total remaining coding requirement. Through updating the estimates, the team is made aware of its development progress, and hence can take appropriate actions to avoid schedule

slippage.

3 Background

This chapter explores research relating to ECMT. In particular, personal software process, software estimation, general time management, and some available time tracking tools are examined.

3.1 Personal Software Process

In determining whether process improvement principles work for individual software professionals [Humphrey 2000], Humphrey developed the Personal Software Process (PSP) [1][Humphrey 1995; Humphrey 1997]. PSP attempts to provide a defined, planned, and measured way for software engineers to improve the quality, predictability, and productivity of their creative engineering work.

3.1.1 Overview of PSP

Software process is the sequence of steps required to develop or maintain software [Humphrey 1995]. Unlike most process improvement methods that address the organizational level, e.g. Capability Maturity Model (CMM) for software, Humphrey's PSP focuses on the personal level. PSP assumes that a software developer can achieve better performance by applying some reasonable software engineering principles. These principles are the measurements of personal performance. Taking these measurements as feedback, software developers will improve their performance. Humphrey introduces these principles into PSP through a gradual process. Figure 3.1 shows the progression of PSP.

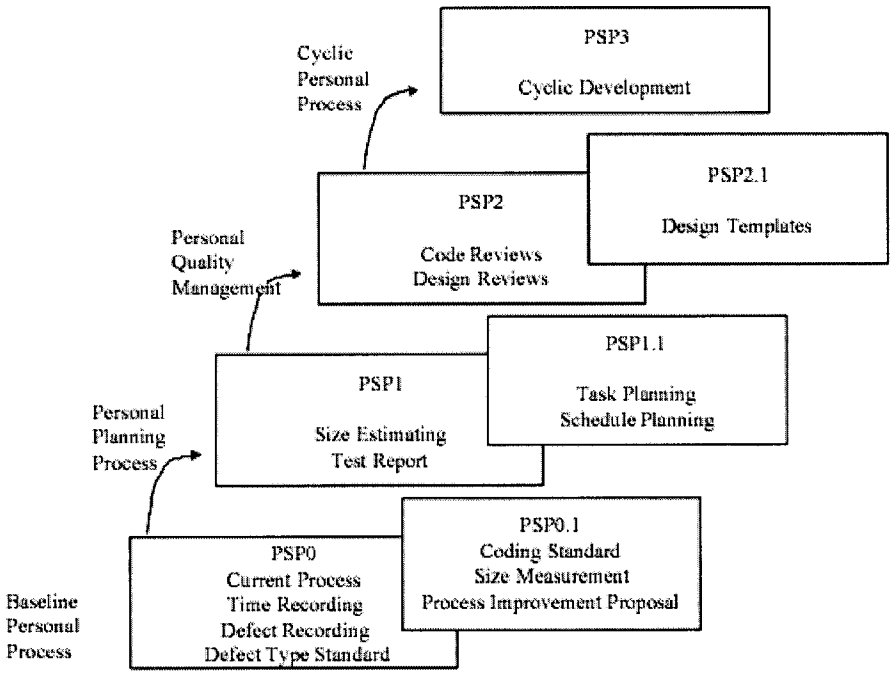

Figure 3.1 The PSP process development

PSP is a practical methodology to help individual software professionals become more effective. Some researchers [El Emam 1996; Ferguson 1997] have found positive results for software defect quality, size estimates, and planning when applying PSP.

3.1.2 Time tracking in PSP

Humphrey supports time tracking in PSP as well as providing detailed steps to create the time log in his book, "*Introduction to the personal software process*"[Humphrey 1997].

The book proposes a standard time recording log, as shown in Table 3.1.

[1] PSP and Personal Software Process are trademarks of Carnegie Mellon University

Table 3.1 TIME RECORDING LOG

Student Date
Instructor Class

Date	Start	Stop	Interruption Time	Delta Time	Activity	Comments	C

The form's head includes name, recording date, instructor, and class (this form is for students). Each time period is entered on one line of the form as follows:

- Date. The date for a certain activity (for example, a task).
- Start. The starting time of the activity.
- Stop. The end time.
- Interruption. Any time lost due to interruptions.
- Delta time. The time spent on the activity, in minutes, from the starting time to the end time, subtracting the interruption time.
- Activity. A descriptive name for the activity.
- Comments. A more complete note on the activity.
- C (Completed). Check this column if the activity is complete.

It is a well-defined recording form for logging activities. However, there are some suggestions in the book for using the form: keep the engineering notebook (including the forms) with you at all times; use a stopwatch to track interruptions; summarize the time promptly (in another form). Maybe some developers like this way, but not everyone can stand so much effort, especially over a long period.

3.1.3 PSP and ECMT

ECMT uses time tracking technique in PSP to monitor the development

progress. However, since ECMT is a computer aided software engineering tool, users can avoid writing down all time tracking records and summarize them manually. If only considering the time tracking feature in ECMT, we can take ECMT as a PSP tool.

3.2 Estimation

Brooks believes that software estimation is one of the "three great challenges for half-century-old computer science". Even though there have been many advances on theory and practice, there is still no method to make software engineering as predictable as civil or electrical engineering [Brooks 2003].

In most estimation models, there is a relation between cost and effort (person-months needed), for example, one person-month is taken as $5000; thus, given effort estimates in person-months, it is possible to translate it to cost estimate. In this thesis, we consider effort and cost estimation synonymous.

The following are some popular estimation approaches.

3.2.1 Function Point Analysis

Albrecht developed Function Point Analysis (FPA) in the 1970s [Albrecht 1983]. Instead of using "lines of code" as a measure of software size, it measures systems from a functional perspective. FPA assumes that the number of different data structures is an accurate indicator of software size. Therefore, it is more suitable for business applications, where data structures play the dominant role in development, as opposed to applications emphasiz algorithms (e.g., compilers).

There are five major components considered in FPA: the number of input types (I), the number of output types (O), the number of inquiry types (E), the number of logical internal files (L), and the number of interfaces (F). After we classify each

component in one of the above five major component catalogs, we assign it a rank of low, average, or high with some predefined rules (refer to the IFPUG [2]*Function Point Counting Practices Manual*). Then FPA assigns a weight to each component with the rank shown in the following table:

Table 3.2 Counting rules for function points

Complexity Rating	Input (I)	Output (O)	Inquiry (E)	Logical internal (L)	Interfaces (F)
Low	3	4	3	7	5
Average	4	5	4	10	7
High	6	7	6	15	10

From the data in the above table, the number of (unadjusted) function points, *UFP*, can be calculated. It is a weighted sum:

$$UFP = 3 \times Low\ (I) + 4 \times Average\ (I) + 6 \times High\ (I)$$
$$+ 4 \times Low\ (O) + 5 \times Average\ (O) + 7 \times High\ (O)$$
$$+ 3 \times Low\ (E) + 4 \times Average\ (E) + 6 \times High\ (E)$$
$$+ 7 \times Low\ (L) + 10 \times Average\ (L) + 15 \times High\ (L)$$
$$+ 5 \times Low\ (F) + 7 \times Average\ (F) + 10 \times High\ (F)$$

The final Function Point Count is obtained by multiplying this *UFP* by an adjustment factor referred to a *VAF* (value adjustment factor):

$$FP = VAF \times UFP$$

The value adjustment factor reflects 14 general system characteristics (GSCs, details of how to evaluate them are available in *IFPUG*[3] *Function Point Counting Practices Manual*) that influence development effort of the application counted. The degree of influence for each of these characteristics ranges on a scale of zero (no influence) to five (strong influence). The total degree of influence (DI) is the sum of

[2] The International Function Point Users' Group (IFPUG), http://www.ifpug.org
[3] The International Function Point Users' Group (IFPUG), http://www.ifpug.org

scores for all characteristics. Then, the IFPUG Value Adjustment Factor (VAF) will be defined as:

$$VAF = 0.65 + DI / 100$$

The Function Point method addresses software size estimate. Some researchers have analyzed much data to get information how many function points can be coded in an average person-month. Thus the FPA can be converted to estimate effort and cost. The relation between function points and lines of code in a certain programming language is also available. This makes it possible to use FPA with other estimation methods.

ECMT does not use FPA directly. ECMT measures work requirement by *dedicated time*. However, in practice, users can convert function points estimates to dedicated time estimates, hence FPA can work together with ECMT

3.2.2 COCOMO

Bohem proposed the COCOMO (COnstructive COst MOdel) cost estimation model in 1981 [Boehm 1981]. In basic COCOMO, the following effort equation shows the relation between effort and software size:

$$E = bKLOC^c$$

Where b and c are constants that depend on the kind of project considered, *KLOC* (thousands of source lines of code) represents software size, and E is effort measured in person-month.

In COCOMO, a project is classified into 3 categories:

- Organic. The project is developed using stable techniques, and developers have much experience on similar projects. Usually, the product is not very

large, and needs little innovation. Example: An accounting system.

- Embedded. There are tight and inflexible constraints for this kind of project. A great deal of innovation is required. Example: Air traffic control, embedded weapon systems.
- Semidetached. The project's characteristics are intermediate between organic and embedded.

The required effort (measured in person-month) is obtained from the following formulas:

Organic: $E = 2.4 \times (KLOC)^{1.05}$

Semidetached: $E = 3.0 \times (KLOC)^{1.12}$

Embedded: $E = 3.6 \times (KLOC)^{1.20}$

Basic COCOMO is suitable for early, rough estimates, since it is based on a simple and crude classification of projects into only three types. In the book *Software Engineering Economics*[Boehm 1981], there are two more complex models described: *Intermediate Model* and *Detailed Model*. The intermediate model uses an *Effort Adjustment Factor* (EAF) and different coefficients for the effort equation:

Organic: $E = EAF \times 3.2 \times (KLOC)^{1.05}$

Semidetached: $E = EAF \times 3.0 \times (KLOC)^{1.12}$

Embedded: $E = EAF \times 2.8 \times (KLOC)^{1.20}$

The EAF is determined by 15 Cost Drivers, such as Product Complexity, Programmer Capacity, Applications Experience and "Use of Software Tools", all of which affect productivity. Therefore, since the intermediate model describes projects more accurately, it is supposed to produce better estimations.

The detailed model differs from the intermediate model for it is phase

dependent. It defines six phases: Requirements, Project Design, Detailed Design, Code and Unit Test, Integrate and Test, and Maintenance. For each phase, there is a different effort multiplier for each cost driver. By considering situations in different development phases, Boehm argues that the total estimate will be more accurate.

COCOMO 2 is a revision of the early COCOMO model [Boehm 2000]. There are two models in COCOMO 2: the Early Design model and the Post-Architecture model.

A software project uses Early Design model at early stages where little information is known. The Post-Architecture model is a more detailed estimation model that is used after a software architecture has been developed. Both have the following basic effort equation:

$$E = 2.45 \times EAF \times (KLOC)^c$$

Where, EAF is the effort adjustment factor. EAF is a product of seven effort multipliers in the early design model, while it is the product of seventeen effort multipliers in the post-architecture model (details in [Boehm 2000]).

COCOMO provides the relationship between work effort and software size. If we know the software size in LOC, we can use COCOMO to make a rough estimate in dedicated time. It is especially helpful when we have no ECMT historical data, but have such data in LOC. However, it is not clear if COCOMO applies to individual feature estimates or to entire project only.

3.2.3 Delphi

Although major research work in software cost estimation field has been devoted to algorithmic models, expert judgment [Hughes 1996] is a commonly used

estimation method in practice. A Dutch study carried out by [Heemstra 1992] revealed that 62% of estimators and organizations use the intuition technique. The wideband Delphi approach for software estimation [Boehm 1981] is a structured technique of expert judgment. It is a better approach than many algorithmic models when there is no historical data. Experienced experts with a good understanding of the project are crucial in applying the Delphi approach. This approach has the following steps:

1. Experts are issued the specification and an estimation form by the coordinator.
2. A group meeting is held to discuss the project and estimation issues.
3. Each expert independently completes the estimation form.
4. Estimates are returned indicating the median estimate and the expert's personal estimate.
5. Another group meeting is held to discuss the results.
6. Experts prepare a revised independent estimate.
7. Steps 3-6 are repeated until a consensus is reached.

Delphi can make accurate estimation in many situations, but often it will require a long time to achieve consensus amongst all the experts.

ECMT is a personal tool. If a developer takes a crucial task that has a wide impact on the progress of the overall project, the team leader should consider using the Delphi approach to help the developer to make an accurate estimate. Future version of ECMT could provide direct support for this approach.

3.3 Personal time management

Lack of good personal time management will result in unproductive days, last minute rushes to meet a deadline, infeasible plans, ignoring long-term objectives, and so on. Personal time management is a group of common sense strategies, some

reasonable suggestions for managing time, for instance, planning tasks with priority, using to-do lists, and making good use of an agenda [Covey 1990; Covey 1996; Davidson 1999]. There is little theory or experiment research on these approaches, however we can learn something by studying them.

3.3.1 The 80-20 rule

Around 1900, Italian economist Vilfredo Pareto, in his study of the patterns of wealth and income, observed that the distribution of wealth was predictably unbalanced. About 80% of the wealth in most countries was controlled by a consistent minority, about 20% of the population. This pattern is summarized as "80:20 rule" or "Pareto's Principle". In 1998, Richard Koch [Koch 1999] observed that the imbalance applies to many aspects in modern life, such as stocks, company sales, and even web site performance. For time management, this rule argues that, for common situations, 20 percent of the total time gives out 80 percent of results and the remaining 80 percent of time only generates 20 percent of the output. 20 or 80 percent is not extremely accurate when used in this way. The principle gives a hint for time management by showing that most of the results come from a minority of time. If we can manage time appropriately, and make as much of our time "20 percent" time, it is possible to achieve greater results with the same amount of time. That means we can be more productive if we know how to use time efficiently.

3.3.2 Prioritize tasks

Prioritizing tasks is an important aspect in personal time management. Each person has his or her goals, long term or short term. Usually, we can decompose a goal

into several tasks. The task list is a list of all the tasks to be carried in a day or a period. However simply writing down such a list and following tasks one by one is not enough. Since in that way, tasks at the top of the list often have the best chance of being done first, no matter whether they are urgent or not.

That is why we need prioritize tasks. By giving important tasks higher priority and sorting tasks by their priorities, we can put more concentration on important tasks.

3.3.3 Planning, monitoring and reviewing regularly

"Since personal time management is a management process just like any other, it must be planned, monitored and regularly reviewed." [Blair 1992] Blair summarized time management skills from a broader view.

Making a prioritized task list helps one to concentrate on important tasks, therefore maximizing the use of time. Monitoring time by recording all activities (not only tasks in the list) will reveal where time is spent. Reviewing a time log is important. This is where we compare the time log with the original plan. If it agrees with the plan, we will have confidence that everything is under control. Otherwise, it will motivate us to follow the plan more strictly or modify the infeasible plan.

3.3.4 Personal time management in ECMT

ECMT attempts to improve developers' work performance by applying some personal time management skills. A to-do list based on estimates and a time log helps developers to make daily plans. This to-do list can be seen as the tasks with high priority. Moreover, ECMT allows developers to monitor each development task, as well as provides text and graphical summaries for users to review their work easily.

3.4 Time tracking tools available

There are many time tracking tools available to fit different requirements. According to their purposes and usage, time tracking tools can be categorized into four classes:

- Time tracking tools: mainly for tracking time.
- Tracking time and billing: mainly for billing.
- Monitoring employees: working as a spy program to monitor employees' use of computers.
- Tracking software usage: recording work time by collecting time spent on certain software applications.

3.4.1 Time tracking tools

Tools in this category provide general time tracking features. Users manage their tasks and track time as a stopwatch (click on an icon to start and stop one timer). However, these tools are not intended for any particular purpose. ECMT provides similar tracking time features.

3.4.1.1 Time Tiger

(http://www.indigo1.com/timetiger/learnabout.asp)

Time Tiger takes simplifying the entry process as a design objective. Figure 3.2 shows its interface.

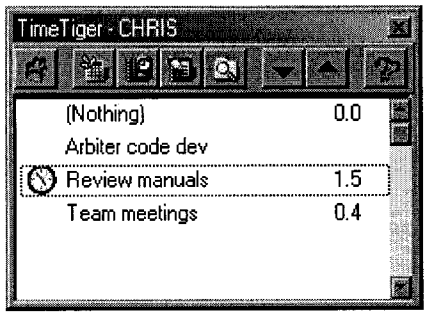

Figure 3.2 Time Tiger

Time Tiger provides a small task list that includes user-defined tasks in a day. The user clicks on the current activity item, and Time Tiger logs the time. Time Tiger provides a report wizard that assists the user in analyzing the time logs.

3.4.1.2 Time Track

(http://timetrack.sourceforge.net/)

Time Track is a simple Personal Software Process tool to help users track their time on projects and activities. Time is input through a Java Swing GUI and recorded in an XML file. Users can also use some other tools to manipulate the XML file to generate other desired report.

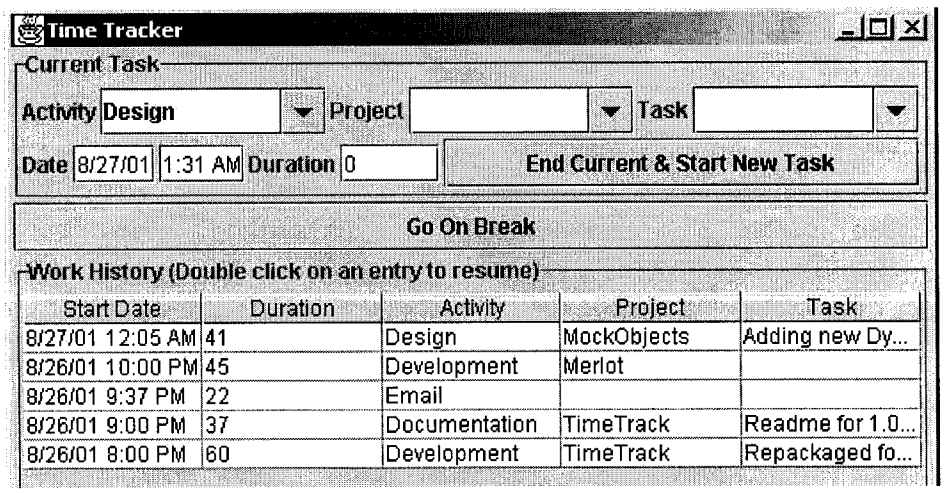

Figure 3.3 Time Tracker

3.4.1.3 GnoTime (formerly known as GTT) - The Gnome Time Tracker

(http://gttr.sourceforge.net/)

GnoTime provides many features more than a simple stopwatch. The project webpage describes the tool as such: "The Gnome Time Tracker is a desktop utility for tracking the amount of time spent on projects, and generating configurable reports and invoices based on that time." By organizing projects into a tree structure, some projects can be the sub-projects of others. The tree can be expanded or collapsed to simplify viewing (figure 3.4). Besides the basic features, GnoTime allows users to do simple project planning, such as making to-do lists and estimating the amount of time needed to complete the project, but it does not support continuously updating estimate information. Moreover, this application provides basic billing support.

GnoTime is the most similar tool to ECMT. However, their purposes are different. ECMT combines the estimation and tracking time functions to assist developers to make better estimates and avoid schedule slippage. GnoTime provides similar functions, but does not combine them.

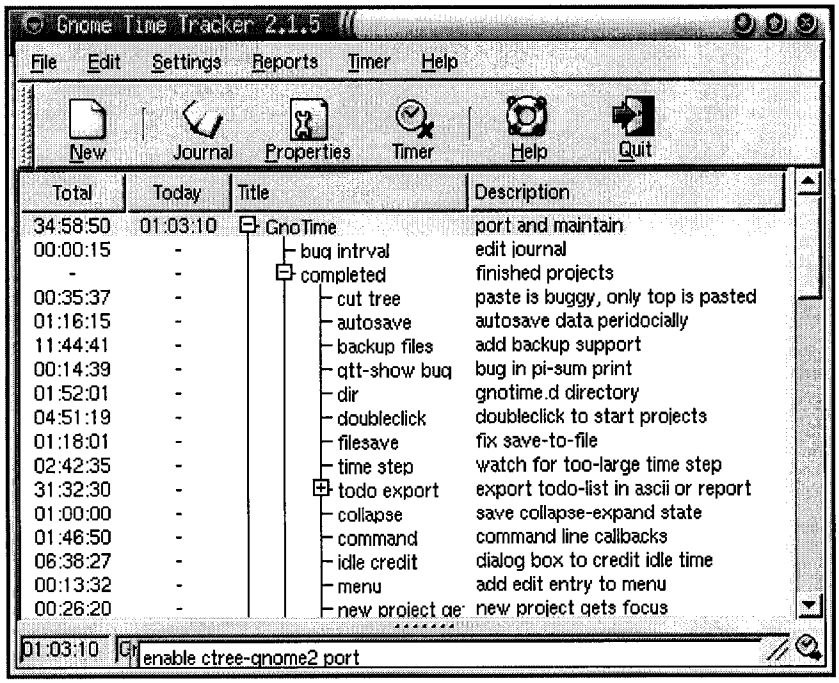

Figure 3.4 Gno Time

3.4.2 Tracking time and billing

Tools in this category are mainly for billing. For example, a consulting company uses such tools to track the time spent on its client, and calculates the fee according to the time log.

3.4.2.1 Time Track

(http://www.trinfinitysoftware.com/timetrack.shtml)

Time Track is a time logging application originally designed "for consultants and freelancers to minimize time spent logging hours and to enhance billing accuracy." This application provides billing functions besides common time tracking features. The following figure shows the interface for calculating the charge based on the time recorded.

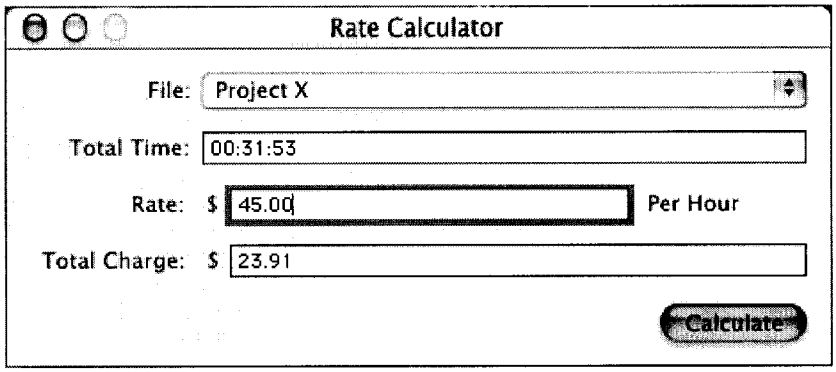

Figure 3.5 Time Track

3.4.3 Monitoring employees

Tools in this category are intended to monitor employees' work by tracking their time spent on computers.

3.4.3.1 TimeFix - Employee Time Tracking Software

(http://timefix.aklabs.com/)

"TimeFix is an effective multi-user software solution for automatic tracking and logging of time you and your colleagues spend working with various software applications." TimeFix can work as a personal time tracking software, but it is mainly for monitoring what application and its time that employees have used. After installing a small monitoring program on each employee's PC, the manager can see time reports for software usage on all computers.

3.4.3.2 Time Hunter

(http://www.structurise.com/TimeHunter/index.htm)

Time hunter helps business managers to keep track of how their employees spend their time on computers. It logs applications, documents, and internet sites employees have used or visited. At the end of the workday, it sends a log to the manager. Time Hunter also provides a graphical view of time data for the manager.

3.4.4 Tracking software usage

When most of a developers' work can be done using computer applications, the running time for each application is an indication of work time. Some applications use this indication to track users' work time.

3.4.4.1 Smart Work Time Tracker

(http://tracker.aklabs.com/)

Some people have to use special software applications to do their main work, for example, using Visual C++ to code, Microsoft Word to write documents and so on. Smart Work Time Tracker (SWTT) provides the ability to log the amount of time spent on different software automatically. In such way, SWTT collects the data concerning working time consumption and distribution. SWTT is sophisticated enough to judge whether a user is working or not. Suppose that a user runs an application, and then goes to lunch. After a while, SWTT on longer detects any activity on the keyboard or mouse, it considers the user is away and thus will stop tracking time. However, if some work is done without using any software, SWTT will not log that work time nor add to SWTT time log.

It is better than other tools if users' work is totally based on software running on one computer, since users even do not need to act on a stopwatch. However, some software development requires developers to think, work with paper, or on different systems, and then SWTT is not able to count this work time in. Moreover, SWTT cannot distinguish tasks using the same application (e.g., coding versus debugging when using Visual C++).

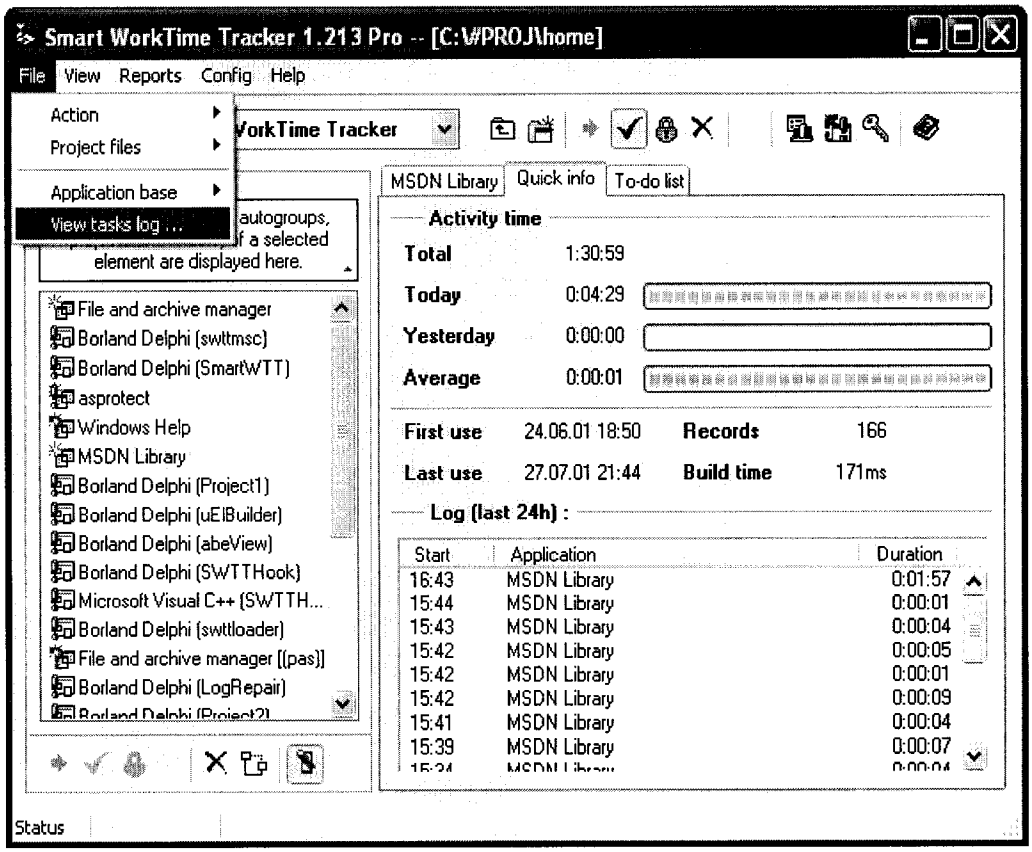

Figure 3.6 Smart WorkTime Tracker

4 Personal Software Development Estimate Capture and Monitor

This chapter describes the key features required for supporting Release Planning at the individual level. ECMT's features are divided into three categories. The tracking time feature records time spent on hierarchical organized tasks, and provides graphical views of the time spent. The estimation feature logs and analyzes the estimations of tasks as projects proceed. The combination of estimation and tracking time provides up-to-date progress measurement of projects.

Before we describe ECMT, some terms are explained here.

Task: A task is a piece of work that must be done. As for software development, we often take a feature as a task. We can estimate the work time for a task and monitor the progress of it, which means tracking the work time on it. Developers should estimate all development tasks. For non-work-related tasks, for example phone time and chat time, developers can ignore the estimation, but still track their time.

Project: A project is a combination of tasks and/or some other projects, and it can be planned over a period of time. A project in ECMT can be considered to include all the tasks assigned to one developer in a software release (which is often developed by a development team.). The work requirement estimate of a project is the sum of all tasks in the project. As a reflection of the actual project, the project in this system has a start date and a deadline. We can estimate the development work capacity (section 2.2) between them.

4.1 Tracking time

Tracking time records where developers' time goes. These records assist developers to understand their time spending habits and are used to track the accuracy of estimations.

4.1.1 Organization of projects and tasks

A tree structure is convenient to manage hierarchically structured information. Since developers may have many tasks which are accessed often, a tree structure is selected to organize all the tasks in ECMT. Every task is represented by a leaf node in the tree, which is called a *task node*. Related tasks are grouped under a non-leaf node that is given a name to describe the reason why they are in the same group. Such non-leaf nodes are *project nodes*, and a project node can contain some other project nodes besides the task nodes. With this structure, the user can easily access any project or task for tracking or some other operations, for example, renaming, changing description, deleting, adding new projects or tasks, and so on.

4.1.2 Tracking time

4.1.2.1 Stopwatch to record dedicated time

Dedicated time is the actual work time (section 1.1), and has an effect on the schedule. ECMT records dedicated time rather than body-present time (section 2.2).

A design goal of ECMT is to make it easy for users to track time. A stopwatch is the common way to record a period of time spent on an activity. Each task in ECMT has a stopwatch to record the time for it. At the start time and stop time, users trigger the stopwatch, and ECMT will log all records for each task (operation descriptions in

section 5.4).

4.1.2.2 Interrupted time

The "uninterrupted time" (section 1.1) is easy to be interrupted by phone calls or friends dropping by. Although users can track phone time or chatting time as tasks, many people may not like to use a stopwatch before picking up the phone, especially if such time is too short or the user does not want to collect such non-work-related data. However, if such time is recorded in the work time, the time log will be inaccurate. In ECMT, the interrupted time is used to reflect such time. The user can add "interrupted time" for a time record to show there is a period that is not productive time. ECMT will subtract it from the total work time.

4.1.3 Reviewing time records

Reviewing time records is a way we get development feedback. Different views are provided to help developers understand the tracking time data. Graphical views provide an easy-to-understand way to know where developers spend time. There are two kinds of graphical views in ECMT. The chronological day graphical view describes the work done in a workday. The time distribution graphical view shows the ratios among the tasks. Text views give most detail information. Moreover, we can modify the time record in the chronological day graphical view (operation description in section 5.5).

4.2 Estimation

ECMT attempts to provide an efficient way for developers to create, track, and

analyze estimates. ECMT's estimation functions are based on the estimation approach used in Release Planning. There are two kinds of estimates in ECMT: work capacity estimates and work requirement estimates. Usually, developers are unlikely to spend eight work hours on one project a day. ECMT requires developers to estimate their *dedicated time* for a project in one day, rather than using eight hours a day to calculate work capacity. We calculate work capacity of a developer in a period by multiplying workdays with the estimate of the dedicated time in one day. In ECMT, we assume that development managers divide a project into small development tasks, and assign them to developers. ECMT asks developers to estimate the *dedicated time* required for each development task, which is the work requirement estimate.

4.2.1 Statistical estimation

In general estimation models, the estimator gives a single value as the estimate. For instance, one estimator estimates a feature needs 10 work days. However, that does not reflect how much confidence the estimator has. Usually, an estimator intends something more: "I have 90 percent confidence that the feature will be done in 10 work days, and 50 percent for 8 work days" to express the estimate. That is the stochastic way of estimation.

However, it is unrealistic to gather the probability distribution of estimates from each estimator, since the real distribution of a stochastic variable is difficult to get in practice. Instead, it is better to assume the distribution type for estimates, and then ask for some parameters from the estimator to figure out an approximate density function.

The work effort for a development task is associated with such factors as the complexity of algorithm, input or output interface, data structure, and even work environment, and each effort tends to add to others to contribute to the estimate. Thus,

the estimate can be considered as the sum of a number of random variables. According to the *Central Limit Theorem*[4], the distribution of work effort estimate is approximately normal. The estimate for the work factor is similar. So, they are assumed normally distributed in ECMT. These assumptions should be tested in future work.

Since two values are enough to fully describe a normal distribution, developers are asked for two guesses with different confidence intervals in ECMT for each estimation. The estimator will give two values of effective work days estimate for a task, the first value is work days required that the estimator predicts with 50% confidence. This value means that it has 50% probability that the estimator will achieve the task before that time, and 50% probability after that time. That is also the mean of the distribution. The second value is the estimate that the estimator predicts with 90% confidence that the feature will be achieved before that time (we select 90% only because it is a high confidence, others are also reasonable). It is considered the 90% worst case. For the work factor, there are also two values required. The first is the work factor that the estimator has 50% confidence, which is the average case. The other one is the value for which 90% of the time the work factor would not be less than that. Work factor estimation is related with a project. Since each project has a start date and deadline, we can calculate the work capacity estimate by multiplying the work factor estimate and the workdays between the project's start date and deadline.

4.2.2 Updating estimates while monitoring progress

When a project is ongoing, work capacity and requirement estimation need to be updated regularly to remain meaningful.

---

[4] The sum of the sample measurements, $\sum_{i=1}^{n} x_i$, as n becomes large, would tend to possess a Normal distribution.

The following terms are defined:

actual time spent (T_{ats}): The number of effective work days spent on a task.

time still required ($\tilde{T}_{tsr}$): The number of effective work days needed to finish the task after having spent the *actual time spent*. This is a random variable estimated by developers based on their best knowledge at that moment.

total estimate time ($\tilde{T}_{tet}$): An estimate of the number of the effective work days needed for a task. It satisfies the following formula,

$$\tilde{T}_{tet} = T_{ats} + \tilde{T}_{tsr}$$

Thus, it equals *time still required* before any work on the task is started, and the *actual time spent* after the feature is finished (*time still required* is 0). Because the *time still required* is a random variable, the *total estimate time* is also a stochastic variable. In this thesis, if not mentioned, the estimate for a task means its t*otal estimate time.*

Updating the work requirement estimate for a task in ECMT means to update the *time still required* estimate. This application will prompt the estimator to input two values: 50% confidence estimate and 90% worst-case estimate. The first time estimate is similar. The only difference is that the actual time spent is 0.

Besides updating the estimate, ECMT also records the modification reason for updating for later reference. We predefine four reasons for modification.

1. Update estimation. The estimation is accurate, we only want to update it to show our awareness of the development progress.
2. Misunderstanding requirement. After developers have done some work, they find they are not solving the required problem. It is often due to inaccurate requirement specification or weak communication between users and developers.
3. Requirement change. "Customers can never tell you exactly what

they want." [Beck 2000] In common business software development, users often cannot tell what they really want.

4. Inaccurate estimation. Inexperienced developers (or short of domain knowledge) do not have the estimation skills to make accurate estimation at first. With the progress of development, they understand the task better, and can make a better estimate.

If developers need to update an estimate by more than one reason, we suggest updating the estimate more than once (one reason at a time).

In addition, developers will comment each modification in detail to explain their estimation evolution.

Estimation modification reasons are useful to analyze developers' estimation skills. As feedback of previous estimation, developers can learn what the common reason for personal inaccurate estimation is; what kinds of tasks are error-prone for estimation. Hence, they can improve estimation skills.

ECMT does not track estimate changes of work factor (section 2.2).

4.2.3 Convergence of the series of estimate

A series of estimates for each task can be obtained by updating the estimate for one task continually. By the end of development, the actual work time is available if developers track their work time. Developers will find that the series of estimates tend to converge to the actual work time as the project proceeds, in other words, the difference between the estimate mean and the actual work time tends to converge to 0. In addition, the difference between the 90% worst-case estimate and the mean of estimate will converge at 0, too. The following figure illustrate the convergence:

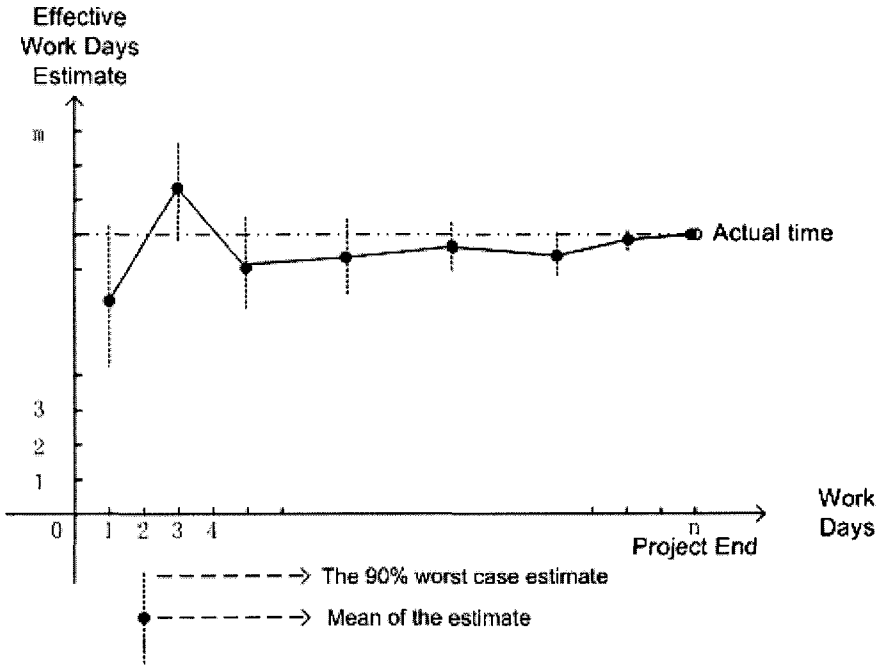

Figure 4.1 Estimates convergence

In figure 4.1, each solid dot is the mean of an estimate. The dotted vertical line reflects the 90% worst-case estimate. A dotted horizontal line end with the last solid dot is the actual work time line. All above data are measured in effective work days.

This conclusion is reasonable due to two reasons.

Firstly, the estimate becomes more accurate with time spent working on the task, since developers know more and more about the task. At the start of the project, developers have to guess some issues about the task. However, as the project proceeds, the uncertainties become smaller and thus the estimates will become more accurate. Correspondingly, the estimator will develop more confidence in the estimate, so the difference between the 90% worst case and the mean becomes smaller. At the end, everything settles down, and the estimate is the same as the actual work time. Thus, the estimate will converge to the actual time and the difference will converge to 0.

Secondly,

$$\tilde{T}_{tet} = T_{ats} + \tilde{T}_{tsr} \quad \text{(section 4.2.2)}$$

The actual work time spent on a task is certain, and the proportion of the estimates (*time still required*) will become smaller with the progress of the project. The estimate error only exists in the *time still required* part, thus the *total estimate time* error will become smaller.

This conclusion manifests the advantage of regular estimation: the ability to realize the potential schedule slippage as early as possible. Each time developers update the estimate, they use their best knowledge at that moment, so it is the most accurate estimate they can make.

4.3 Combination of estimation and monitoring

There are some applications available to deal with estimating and monitoring software development, but no one attempts to combine these two (the author can not find one). ECMT attempts to use estimating and monitoring information assist personal software development.

4.3.1 Estimating with historical data

With using ECMT, developers will accumulate some historical data, that is how much work time is spent on a task. While making new estimates, it is helpful for developers to refer to similar tasks they have completed in the past. The actual work time for a former task (if not finished, the latest total estimate time, in this section, we do not distinguish them) in historical data is a good hint for the total estimate time for the current task. However, if a developer has to browse all the old similar tasks, and compute the average work time manually, most developers will be reluctant to take advantage of historical data. ECMT provides a feature to assist developers in choosing

similar tasks and computing the average work time. Assuming four tasks recorded are similar to the current one and selected, and their actual work times are 4.5, 4.3, 4.0 and 4.7 effective work days. In addition, the developer has spent 1.4 effective work days on the current task. Then this developer will get a suggestion that there should need (4.5 + 4.3 + 4 +4.7) / 4 -1.4 = 2.975 effective work days for the remaining work of the task, if there is no special reason to change the estimate. (operation description in section 5.6)

4.3.2 Estimating with historical estimate series

In section 4.3.1, we take the actual work time of each task as reference for the current estimation. However, we have the estimate series for each task, which includes more information than the actual work time. One kind of important information is our estimate error at a special development stage. For example, the estimate error (mean of the estimate) of a task is 1.1 effective work days at the 54% development progress. Such data can give some hints for the current estimation. If many similar tasks' estimate series show that a developer often make estimates which are about 3 effective work days less than actual work time while these tasks are 60% finished, the developer might consider increasing the current estimate. However, the developer is not forced to adjust the estimate, since there are many possible reasons for estimation error, and the current task is different from others. Moreover, it is possible that the developer has adjusted the current estimate. Therefore, adjusting the estimate based on such information will cause over-adjusting. Figure 4.2 illustrate this usage.

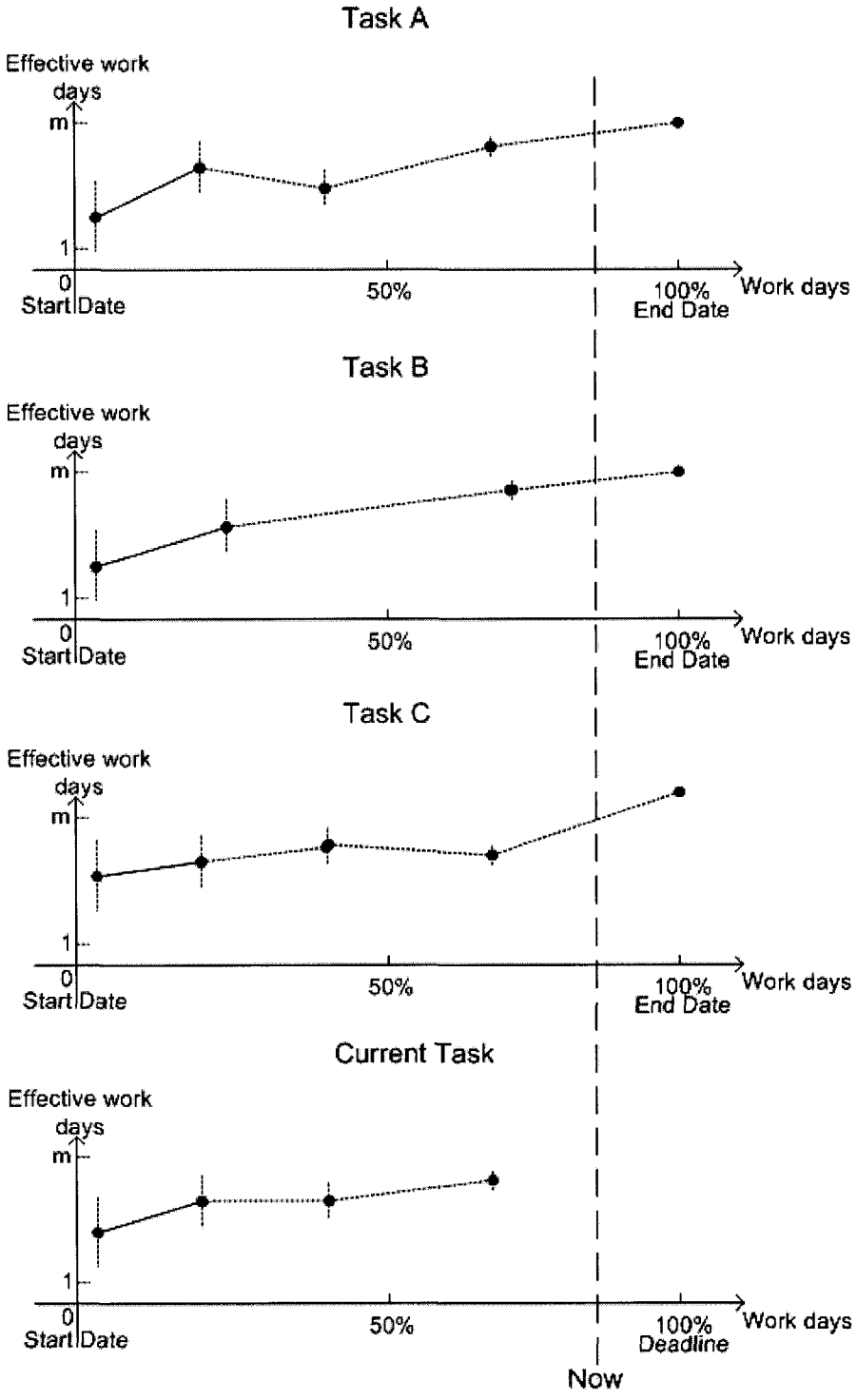

Figure 4.2 Estimating with historical estimate series

In figure 4.2, task A, B, C are similar tasks to the current one. Each Cartesian coordinate system is an estimate series of a task. The x-coordinate is represented with a

scope form 0 to 100%, because the length of each task's work time is different, and we need to convert the estimation date into that scope to compare all these estimates. The long vertical line is the current development stage.

4.3.3 To-do list

Making a plan is a beneficial habit. The project plan for a developer should include all the tasks to be done for the project, which can help to avoid forgetting something important. The plan needs some effort. Since it does not contribute to the work directly, many developers are reluctant to make plans. Based on the mean of the estimate for tasks and their time logs, ECMT is able to generate a reasonable to-do list for a project automatically.

For each task, it should have a start time and deadline. By defining N as *work days* for the task, $D_{deadline}$ as the date of deadline, D_{today} as today's date, $N_{holidays}$ and $N_{weekends}$ as the days of weekends from now to deadline, we can calculate the *work days* (not including weekends):

$$N = D_{deadline} - D_{today} - N_{holidays} - N_{weekends}$$

The remaining work time (T_r) required for the task can be computed (the mean of the stochastic variables is used here) by using time still required (from latest estimate) to subtract the actual time spent after the latest estimate (obtained from the time log). Then the amount of the work time for the task a day (T_{day}) can be obtained by:

$$T_{day} = \frac{T_r}{N}$$

The *time for task a day* means that if a developer can spend such time on the task every work day, the task will be achieved on schedule (With different confidence for remaining work time T_r, developers get deferent confidence).

4.3.4 Project summary

To be aware of a project's progress, the remaining work capacity and remaining work requirement are essential. As a personal tool, ECMT only concentrates on one developer's tasks and summarize one developer's assigned tasks.

4.3.4.1 Time spent summary

By reviewing the work that has been done on the project, developers can be aware of how much work has been done. Because ECMT records the uninterrupted time spent on tasks, it is easy to make such analysis. The *total work time* and the *actual work factor* are selected to indicate the effort spent. The *total work time* reflects how much work the developer has done. The *actual work factor* indicate how much time is really spent on this project against the body present time.

The *total work time* (T) is the sum of all uninterrupted time spent on the task of this project measured by effective work days during a period of work days (D), and the *actual work factor (W)* is:

$$W = \frac{T}{D}$$

4.3.4.2 Capacity constraint

To know if development is on schedule, it is important to know how much work remains besides the work that has been done. Following the definition in 2.3, the remaining work in a project is called *remaining working requirement (f)*. For each task, we can calculate *f* by using its latest *time still required* estimate to subtract the *actual time spent* after the latest estimation.

To get the *total remaining coding requirement* (F), all tasks' *remaining coding*

requirement estimate should be combined, $F = \sum_{i=1}^{n} f_i$. Since F is a line combination of normal distributions, F is also normal distributed, where the mean is $\sum_{i=1}^{n} \mu_{f_i}$, and the variance is $\sum_{i=1}^{n} \sigma_{f_i}^2$.

Remaining coding capacity (N) reflects the available total uninterrupted work time from now to the expected deadline. It is an estimate based on the work factor estimate. *Remaining coding capacity* is the product of *work days remaining (T)* and *work factor (w)*. The *work days remaining* are counted from now to the deadline excluding weekends, holidays, vacation days. Since the *work factor* is estimated as a random variable that is normally distributed, and $N = T \times w$, thus N is also normally distributed. The mean and variance are $T \times \mu_w$ and $T^2 \sigma_w{}^2$.

To reflect the development progress, we shall define a new quantity D(T), "delta", as follows,

$$D\ (T) = N - F$$

D (T) depends on the time T, the number of workdays in the coding phase.

The distribution of D(T) is also a normal distribution because N and F are normal distributed. Considering the distribution of *Remaining coding capacity* and *Total remaining coding requirement,* the mean for D(T) is $T \times \mu_w - \sum_{i=1}^{n} \mu_{f_i}$, and the variance is $T^2 \sigma_w{}^2 + \sum_{i=1}^{n} \sigma_{f_i}^2$.

With the distribution of D(T), we can obtain some interesting results.

The most interesting result is how much confidence the developer has that the project can be done before the deadline. That is the probability that $D(T) > 0$, which can

be calculated from the distribution of D(T). If there is 80% probability that D(T)>0, that is a good news. If there is 25% probability that D(T)>0, it is probable that the project will be late, and something should be done. The developer can increase the work factor estimate so to add more time on remaining work capacity. Basically, work factor is an estimate based on the old data, but the developer can adjust it. If this task has a high priority and there only remains a short period, the developer can make a higher estimate for the work factor. Many people work more time before a deadline, so the higher work factor estimate is reasonable. In a common situation, the work factor for a full time developer is about 0.6 [Penny 2002]. It is rational if the work factor is 0.9 or even 1.2 considering working extra hours a day. If the scheduled release date can be negotiated, postponing the deadline is also a choice.

The second result is that the developer can calculate if he or she needs the 80% confidence, or 50% confidence to achieve the project, then how many workdays are needed. That is to compute the T, given the possibility of D(T)>0, and the distribution of D(T).

5 Prototype implementation

The previous chapter described ECMT concepts in detail. A prototype was implemented to explore and demonstrate the usefulness of ECMT concepts and Release Planning. It also provides a basis for generating ideas for future work. In fact, some concepts in ECMT were updated while this prototype was implemented. This chapter provides details on ECMT features and how they are implemented.

Most features in chapter 4 have been implemented except "Estimating with historical estimate series" (section 4.3.2).

5.1 Overview of ECMT

The main interface (figure 5.1) of ECMT is divided into two parts: the task hierarchy pane at left and the tabbed panes at right. The task hierarchy is a tree structure for organizing tasks and projects. The tabbed pane group at right includes five panes. The summary pane shows the information for current projects. The track pane includes a table that contains tasks that the user wants to track. The estimate for a task is shown in the estimate pane. The detail pane shows the detailed information of a task or a project, and the user can modify its information. The day report pane shows a day's work using graphical reports and a text summary. The panes, except the summary pane, reflect the task or project selected in the task hierarchy.

The menu provides operations for organizing the tree and some controls for the views in the right tabbed panes. We will describe them when introducing related features.

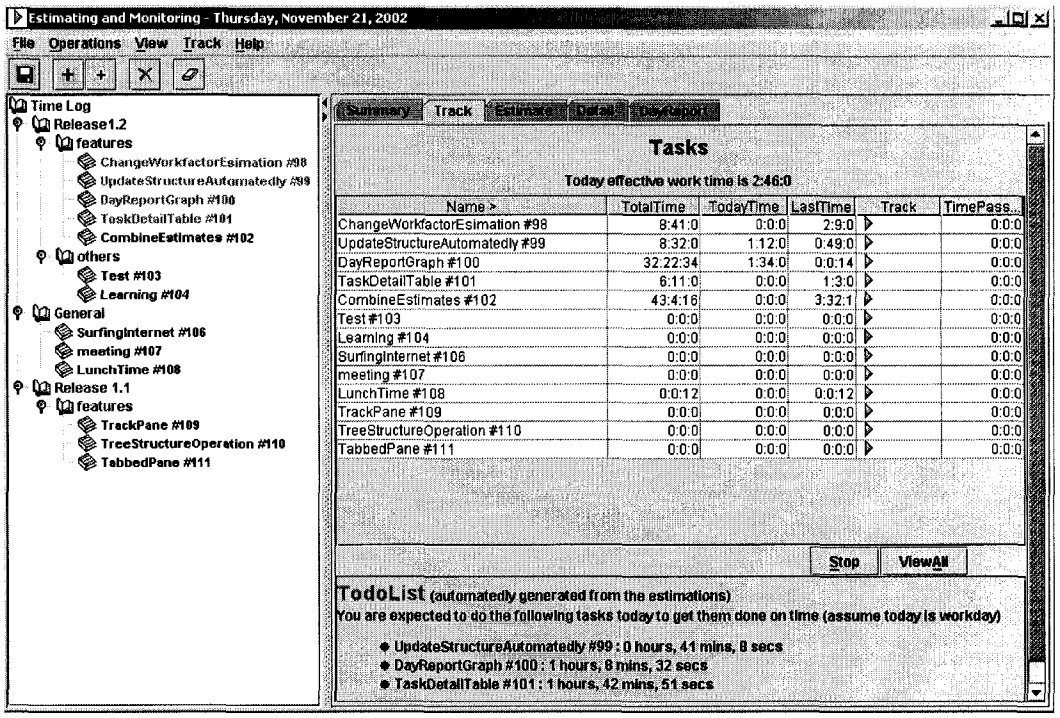

Figure 5.1 Estimate capture and Monitoring Tool

ECMT was implemented using the Java programming language (JDK 1.4) and XML to store the information. Moreover, we used the probability package in the OR-objects[5] library to make normal distribution operations for estimation analysis.

5.2 Tree structure

ECMT organizes projects and tasks in the left pane with a tree structure. The nodes with children represent projects. The leaf nodes are task nodes. Each node has an identification number to avoid name confusion. We suggest users define a project node according to a real release or project, but we do not enforce or check this in ECMT.

[5] http://opsresearch.com

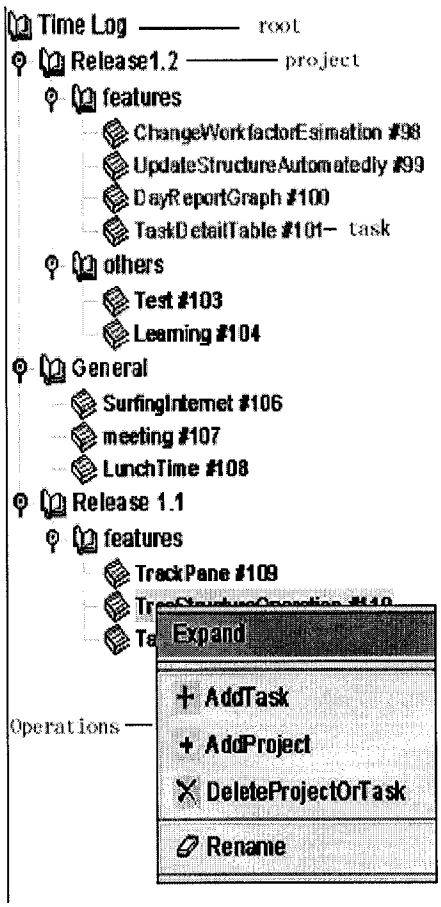

Figure 5.2 Organization of projects and tasks

ECMT provides the following operations to organize projects and tasks. All operations can be accessed from the menu, and some operations can be accessed via popup menu and toolbar.

1. Add project: add a non-leaf node in the tree hierarchy. The user is prompted to input a descriptive name and a detailed description.
2. Add task: add a leaf node in the tree hierarchy. It also requires the user to input a descriptive name and a detailed description.
3. Expand and collapse: if the current node is expanded, the collapse menu item is available; otherwise, the expand menu item is available.
4. Expand Tree and Collapse Tree: only available at the menu for expanding

the whole tree or collapsing it (figure 5.3).

5. Delete project or task: delete the current node.

6. Rename: rename a node.

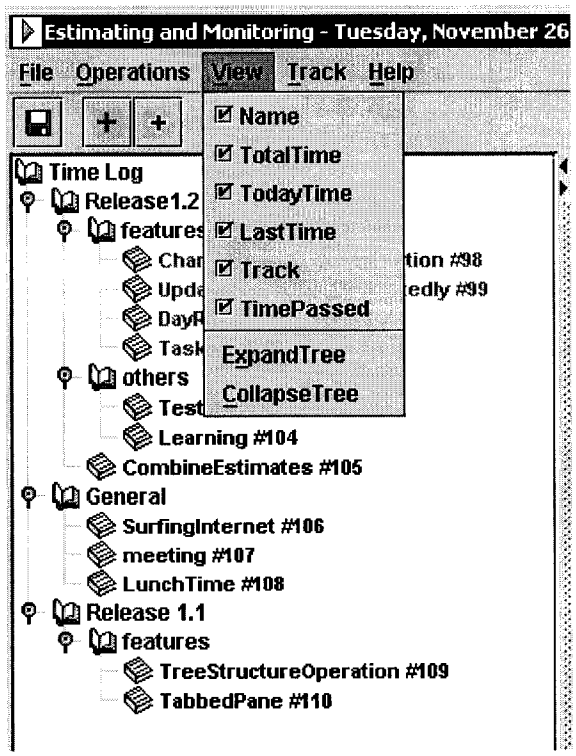

Figure 5.3 Expand or collapse tree

5.3 Details of a project or task

The detail pane is related to the node selected in the tree structure. Since the tree structure includes two kinds of nodes, project nodes and task nodes, there are a project detail pane and a task detail pane.

Figure 5.4 is the detail view of a project node. At top of this pane, a task list shows brief information of all tasks in this project, which includes task name, task description, time spent on the task, time still required, and the remaining workdays. Below is the description for this project, the user can modify it by pressing the "ChangeDescription" button. The "estimatable" check box is used to indicate that the

user needs to estimate a work factor for this project. If users mark the check the checkbox, they should input the start date, deadline for this project, and an estimate for the work factor.

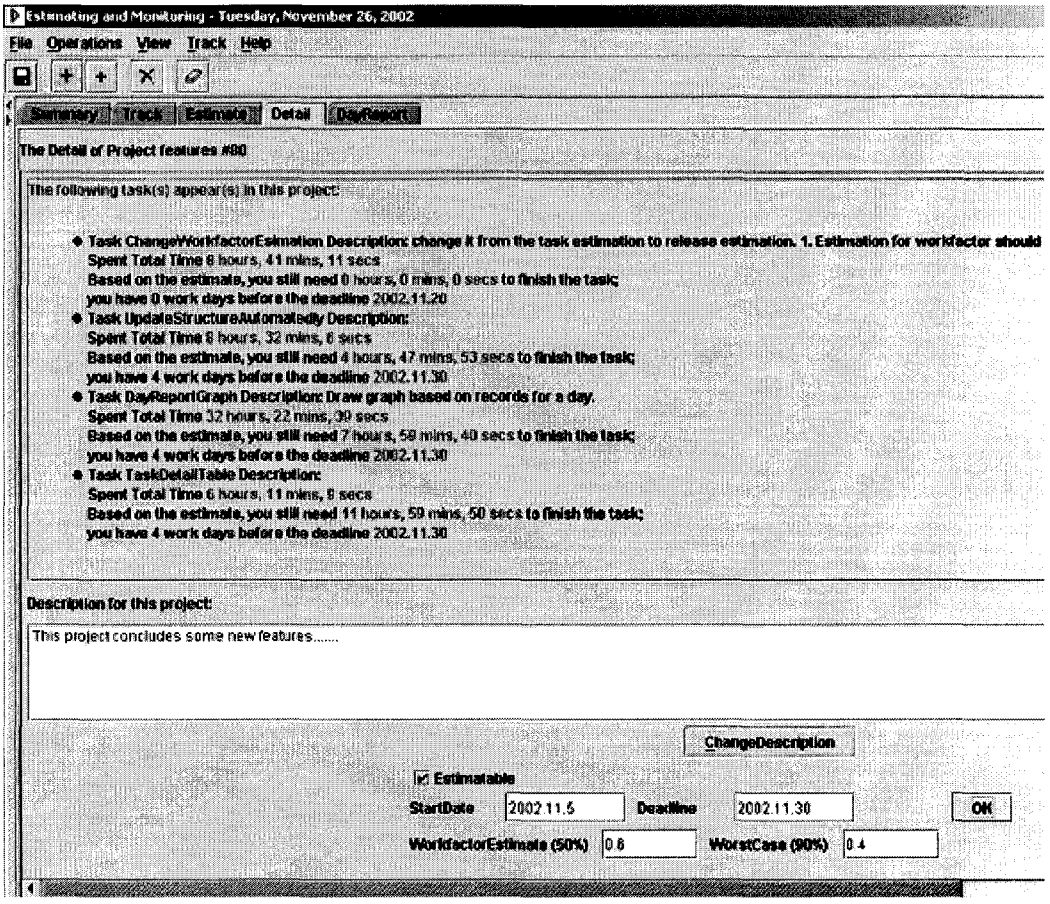

Figure 5.4 Detail of a project node

Figure 5.5 is the detail pane for a task node. The pane includes a table that includes detailed time records when a developer works on the task. Usually, the data in the table is generated by using the stopwatch for each task (section 5.4). By double clicking the time items in the table, the item will become editable and users can modify it. Also, users can insert or delete a record by buttons below the table. ECMT also provides a more convenient way to modify time records with a graphical view (section

5.5). There is an editable pane for users to add a description for the task. At the end of the pane, users can set if the task is "estimatable". All development tasks should be "estimatable", and users are prompted to input the start date and deadline. If the task is non-work-related, we can ignore this property safely.

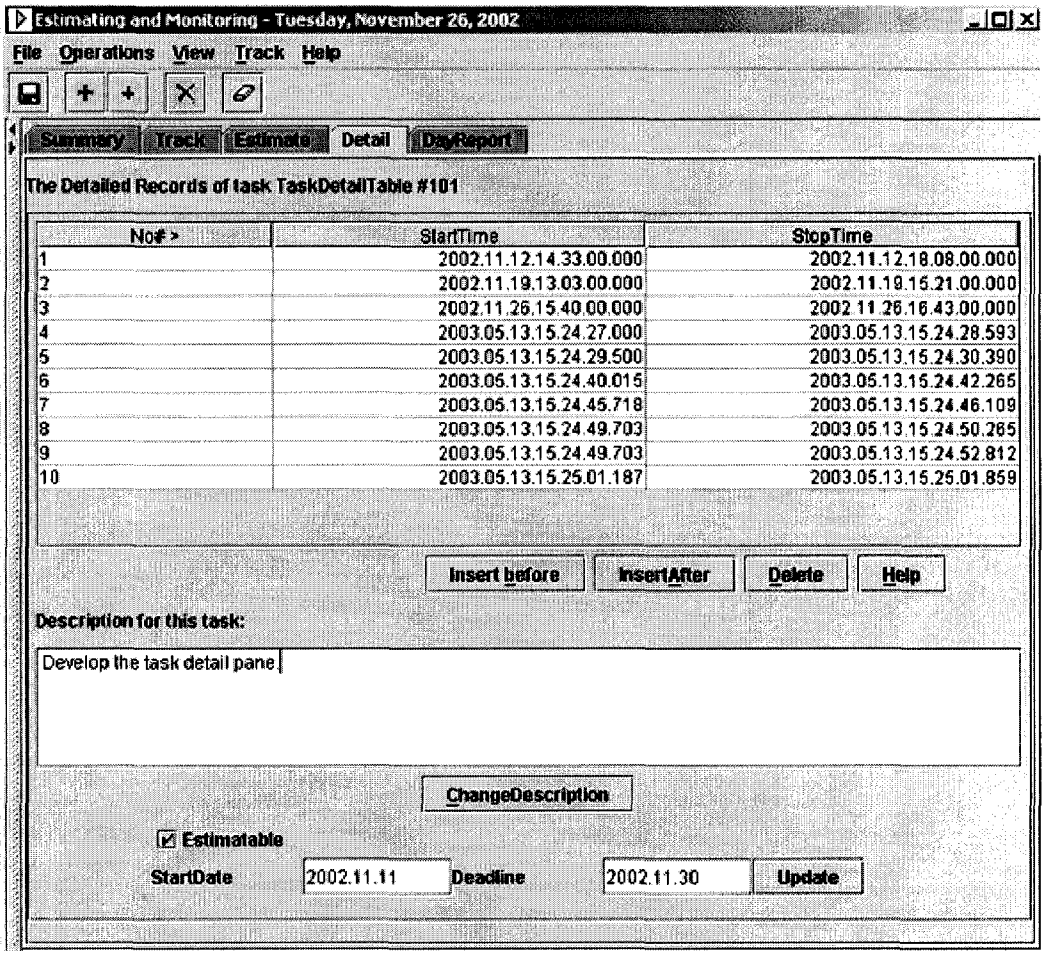

Figure 5.5 Detail of a task node

5.4 Tracking time

A stopwatch is a natural way to record time spent on an activity. ECMT gives each task node a stopwatch in a track table, but only one can run at one time. There is a

"play" button (▶) for each task in the track table. When it is clicked, ECMT will record the start time of the task, and the "play" button will change to a "stop" button (■). Meanwhile, the time passed column in the table starts to show the amount of time elapsed. ECMT will keep tracking time until the "stop" button is clicked. In such a way, time spent on tasks is recorded in detail. The other columns in the table show (figure 5.6, 5.7, 5.8, 5.9): Total Time (overall time spent on the task), Today Time (time spent today on the task), and Last Time (latest time recorded on the task).

Since developers can have many tasks, ECMT usually only shows the tasks in a project in the track table (figure 5.6).

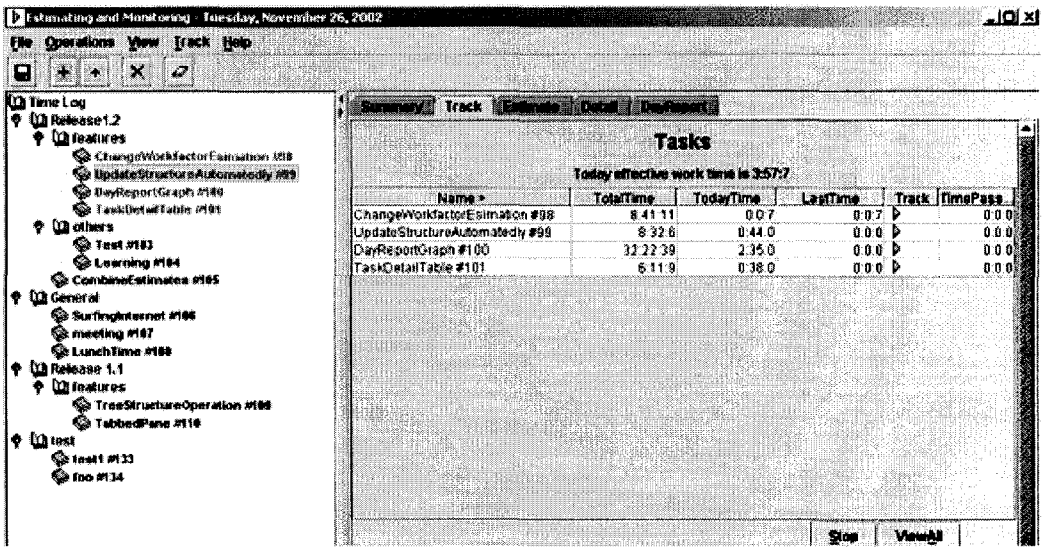

Figure 5.6 Track table for a project

However, users can click the "ViewAll" button below the table to show all tasks (figure 5.7).

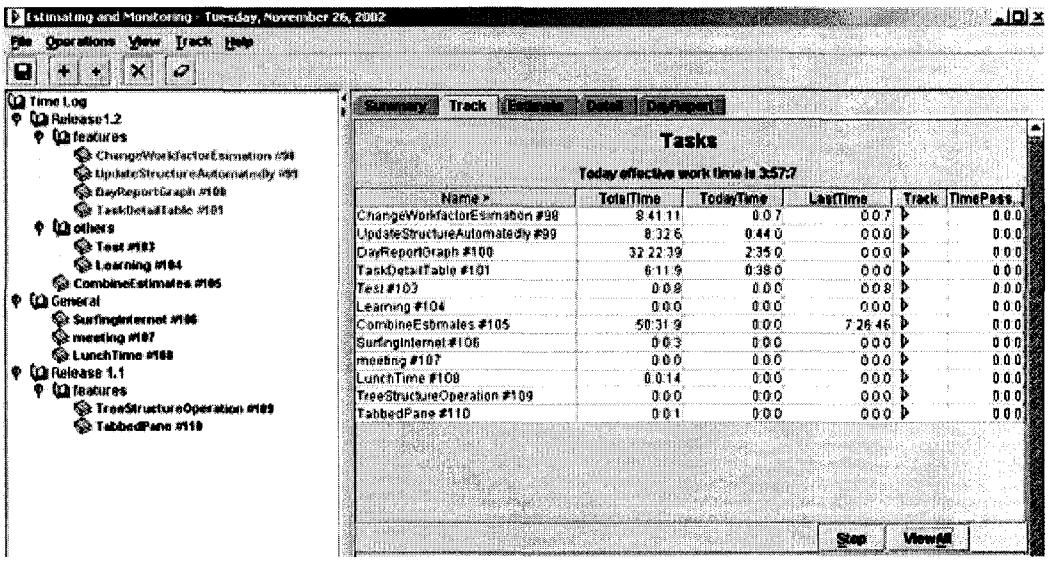

Figure 5.7 Track table for all tasks

Users can configure the table to hide some columns when they do not need all the information. The menu "View" includes the columns items as check boxes, if they are unchecked, they will not show in the table.

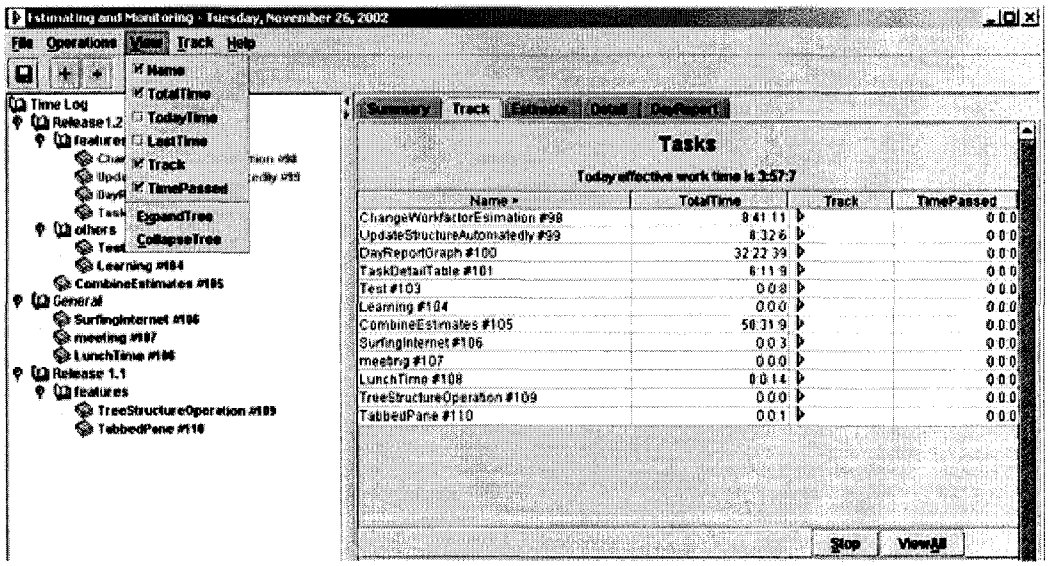

Figure 5.8 Configuration of columns in track table

The items in the track table can be sorted by each column. Users click the column heads to sort the table. There will be a symbol "<" (descending) or ">"

(ascending) to indicate that the table is sorted by this column.

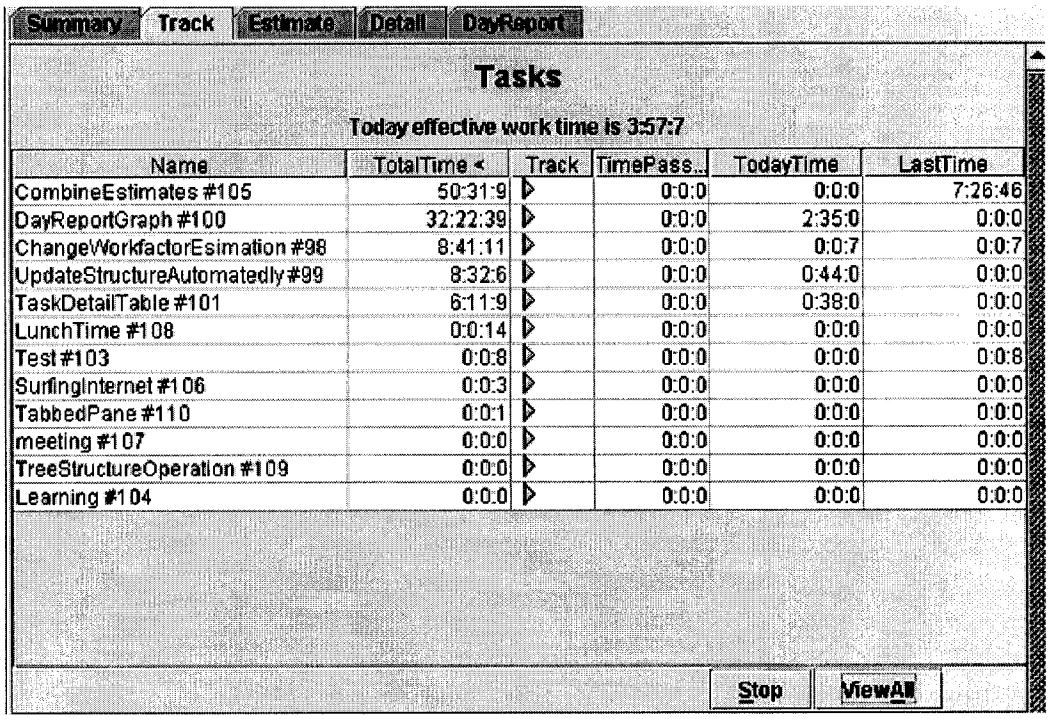

Figure 5.9 Sort track table by "TotalTime"

A "TodoList" below the track table shows the planned work time for each task today (section 4.3.2). ECMT arranges "TodoList" and the track table in the same pane, because developers can easily know what they are supposed to do by viewing the "TodoList" while tracking their activities.

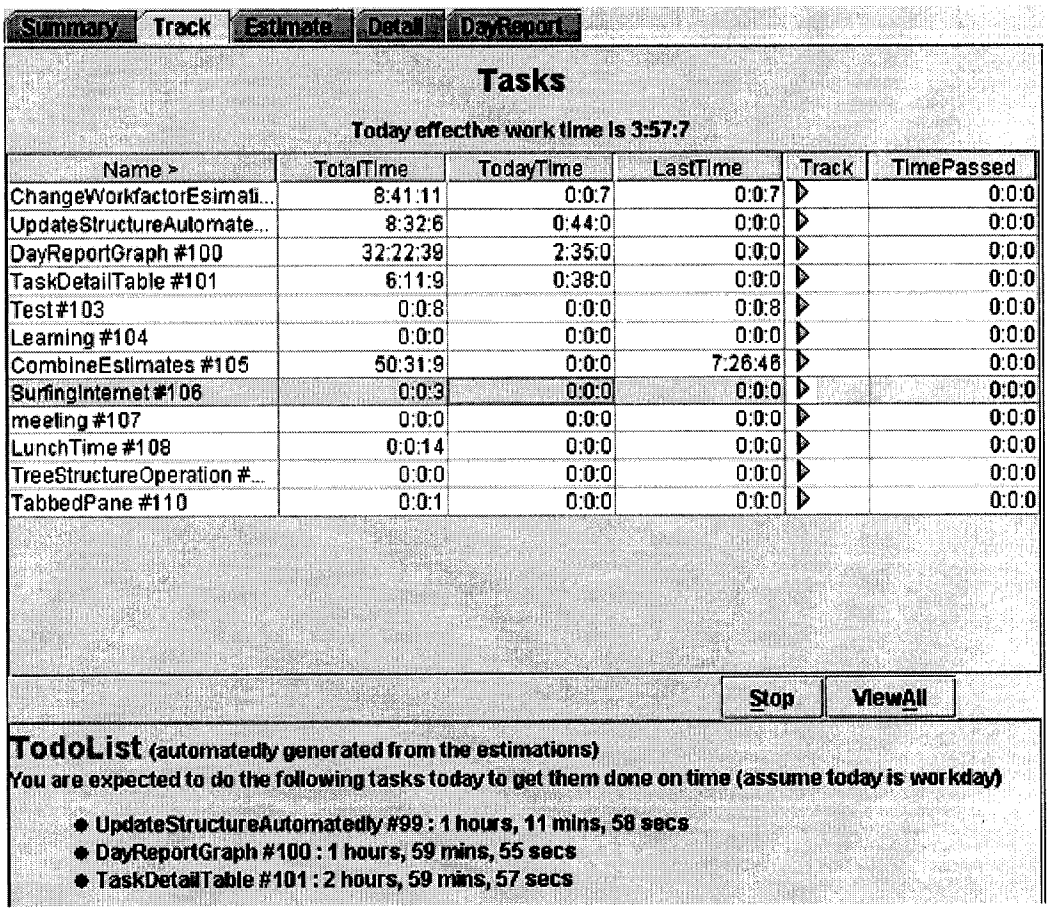

Figure 5.10 "TodoList"

5.5 Review

Graphical view of time records enable developers to understand there their time goes visually. At the end of a workday, we suggest developers review their work in the day and adjust the work time to reflect the real work.

The review pane includes three kinds of reviews of time records in a day. The chronological day graphical view describes the work done in a workday.

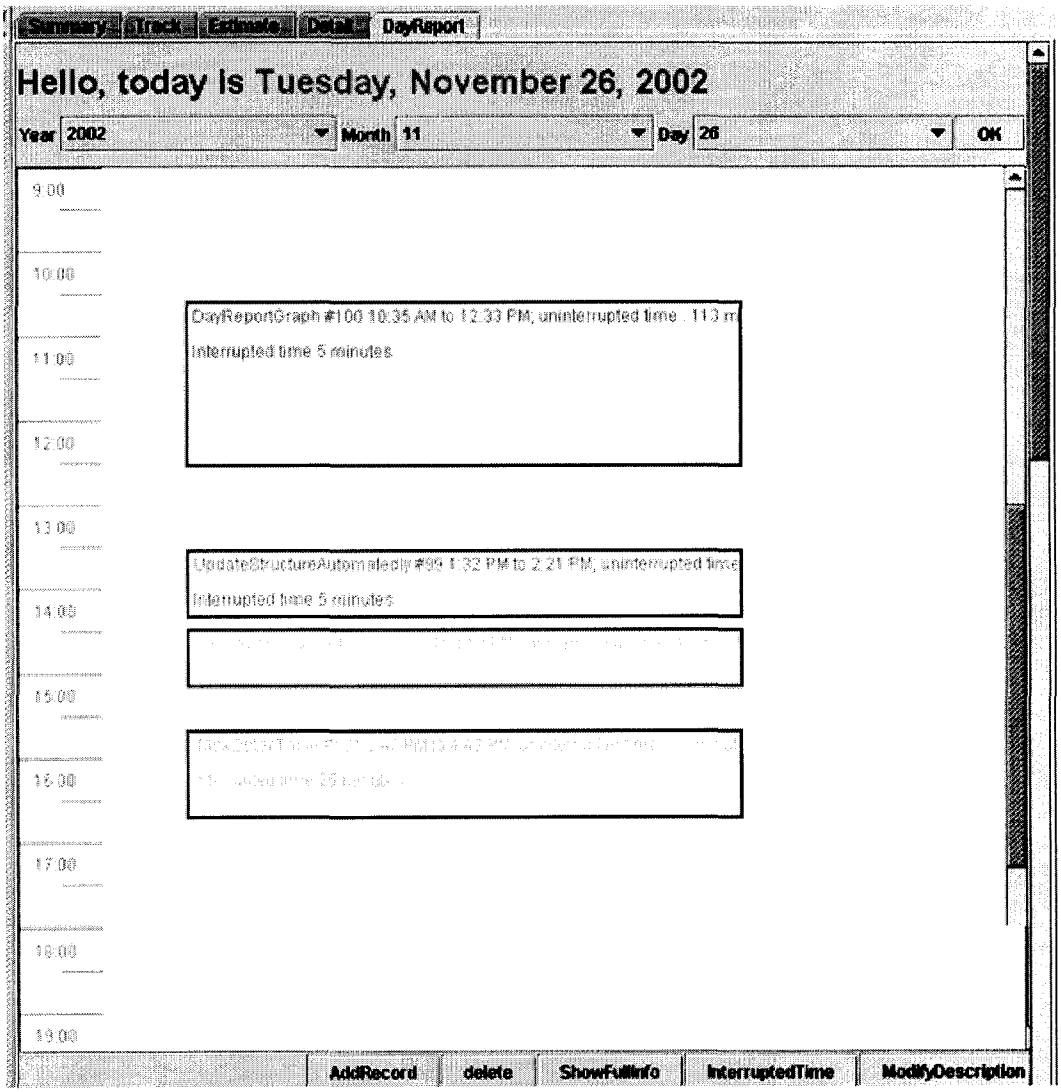

Figure 5.11 Chronological day graphical view

The chronological day graphical view also provides the way to modify the records. The block between its top line and bottom line indicates the time spent on a certain task. Dragging either of the lines will adjust the time log. Moreover, the popup menu and the buttons provide the choice of adding a new record or deleting an existing one.

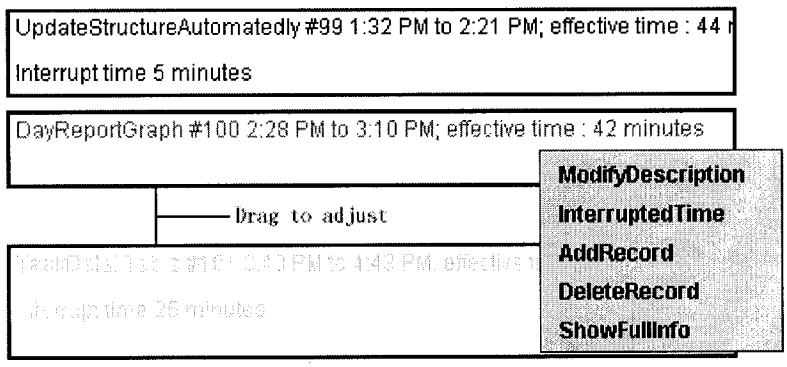

Figure 5.12 Operations in the chronological day graphical view

The time distribution graph (figure 5.12) illustrates the total time spent on each task in a day and each one's percentage. This graph visually displays developer's working time distribution and helps to understand where the time goes.

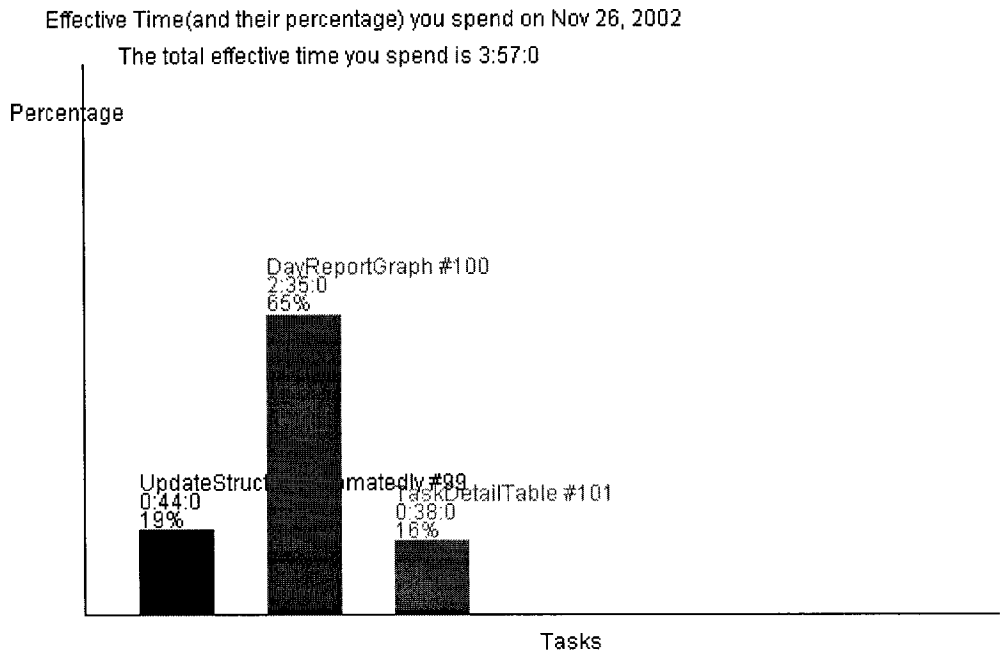

Figure 5.13 Time distribution graphical view

In addition, a text summary gives a text format description for a day's work,

each task in a line.

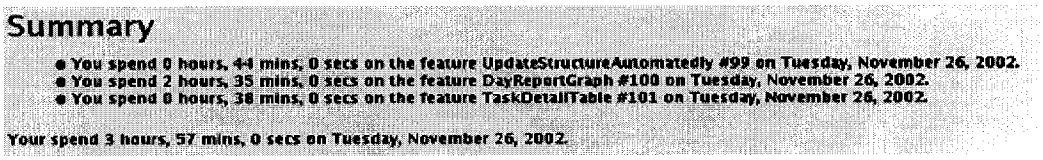

Figure 5.14 Text summary for a day's work

5.6 Estimation

The estimation pane provides features to update estimates and analyze their estimates. Figure 5.15 shows how to update the estimates.

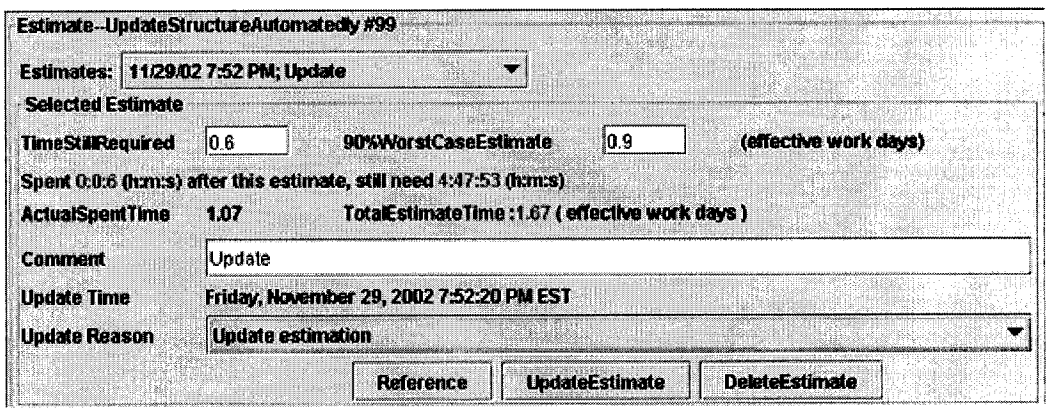

Figure 5.15 Updating the estimate

Users are required to make estimates for the task using units of effective work days. The pane shows the last estimate (first line) and the time spent after last estimate (second line) to assist users in understanding the development status. Users will make new *time still required* estimate for the task. Moreover, users are expected to record the modification reason and add a comment for later reference.

The "reference" button will invoke a reference window to use the historical similar tasks' data (section 4.3.1).

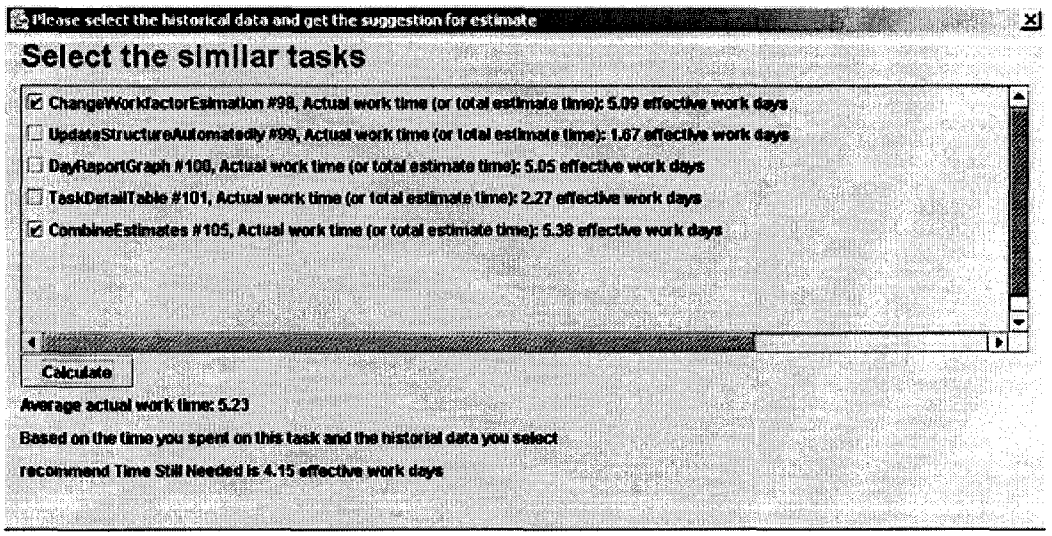

Figure 5.16 Reference window for estimation

A list in the window includes all tasks recorded by ECMT (which may be long in general, however as a prototype tool, it is acceptable.). Each line of the list includes a task name and its work time. After selecting similar features in the list, developers will get a suggestion based on the selected tasks and the current one.

Another graphical view demonstrates the estimate series.

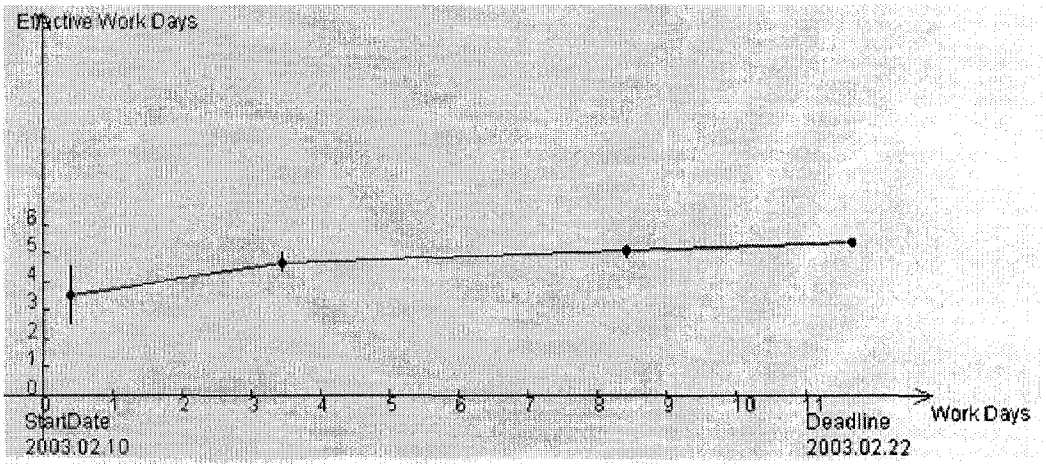

Figure 5.17 Estimate series

This estimate series graphical view illustrates a user's estimate for the task from the start date to the deadline (section 4.2.2).

The red dots in the graphical view are the mean of each estimate. The blue vertical line indicates the 90% to 10% confidence estimate (section 4.2.3).

A text summary for the estimate series provides the most detailed information at the end of the estimate pane.

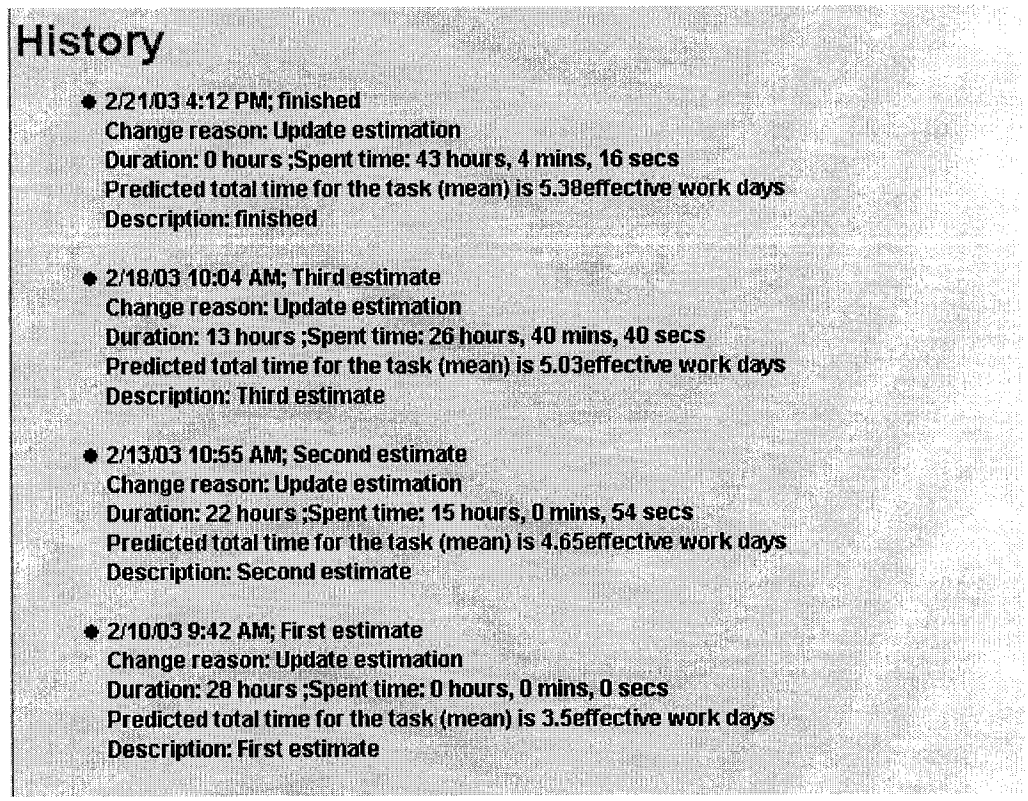

Figure 5.18 Estimation text summary

5.7 Summary pane

The summary pane provides an overall view of the current project. All items in the pane are described in section 4.3.3.

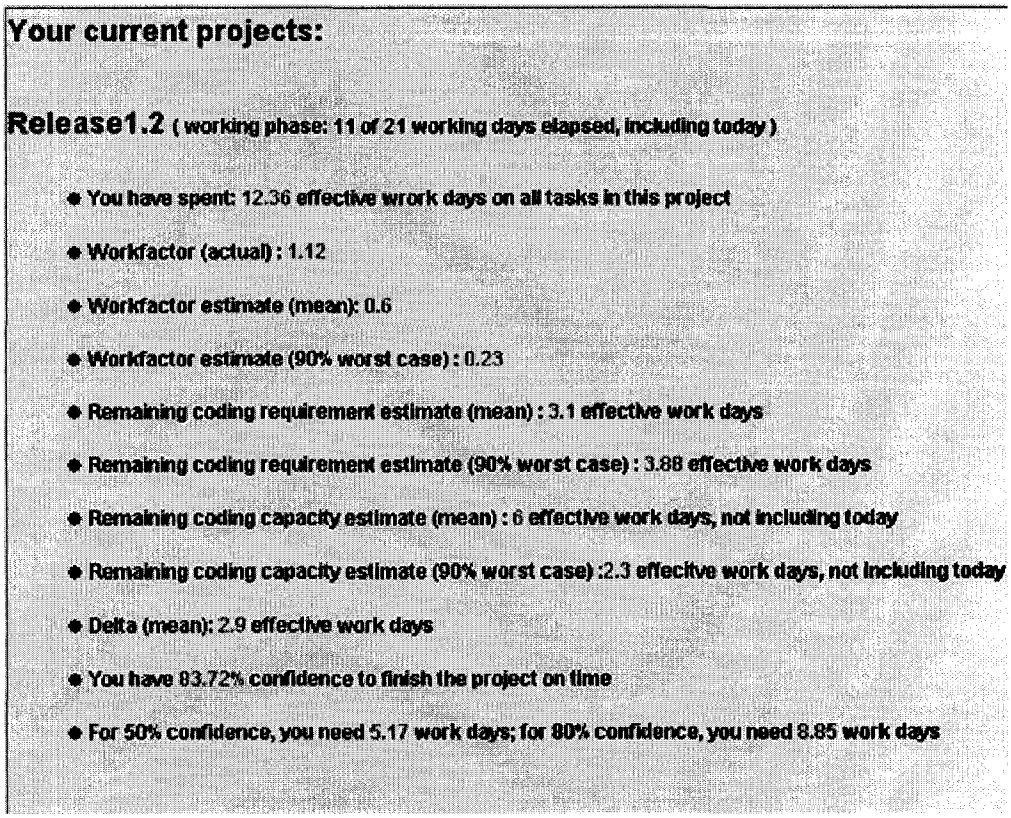

Figure 5.19 Summary pane

6 Evaluation

The author and his colleagues used ECMT to test its usefulness. The experience helps to generate ideas for future work and refine the tool. This chapter discusses the author's experience and feedback of colleagues.

Since many features of ECMT were not available while developing it, and the data format changing, the author only provides the data for the last developed feature.

6.1 Author's experience while implementing a feature

The feature, "combining the remaining work capacity and the remaining work requirement", was developed at the end of this project. ECMT provides an overall view of all the tasks by combining all the estimates as *D (T)* (section 4.3.3):

$$D(T) = N - F$$

This feature calculates the amount of the confidence that all the tasks finish before the deadline and the number of workdays required with 80% or 50% confidence.

The author recorded the work time spent on the task. Figure 6.1 is the detailed time log.

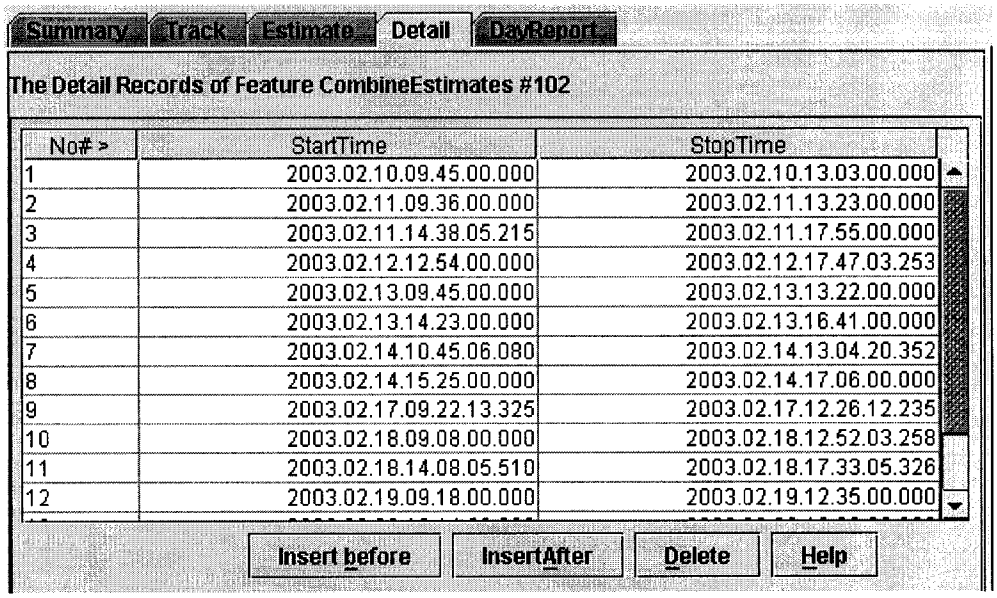

Figure 6.1 Detailed time log

Figure 6.2 is the to-do task generated by ECMT during the development.

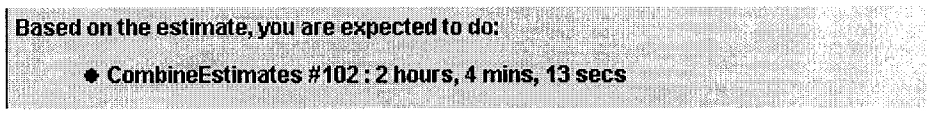

Figure 6.2 To-do task generated

Figure 6.3 is the estimate graphical view for this task. The author made 3 estimates during the development. It is shown that the estimate is more and more accurate as development proceeded.

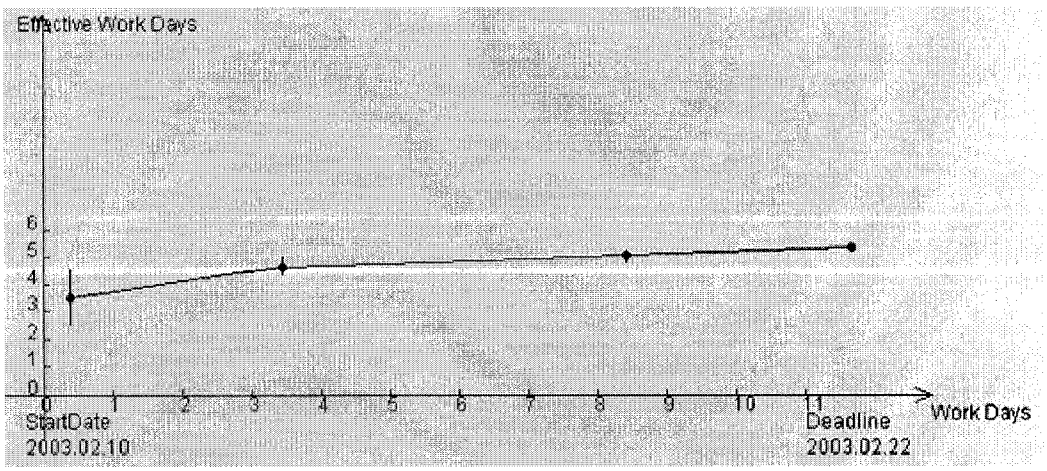

Figure 6.3 Estimate series

It is difficult to estimate work time accurately for a new task at first. From figure 6.3, the first estimate is less than the actual work time by 35%. Moreover, the error between 50% probability and 90% probability is also large, that is 29% in the first estimate. However, the estimates tend to converge to the actual work time with the time going. The differences between the 50% confidence estimate and the 90% worst-case one tend to converge to 0. This agrees with the analysis in section 4.2.3.

6.2 Feedback of ECMT

The author asked his colleagues to try ECMT in a casual way for their development or just as a general-purpose time tracking tool. Most feedback of ECMT is positive. The time tracking feature is not annoying, though it needs some effort to trigger the stopwatch. The continuous estimate updating is useful for understanding the development status, because users make the estimate with the latest monitoring information and the best knowledge of project at that moment. Reviewing the summary pane, time tracking data and to-do list produces some progress pressure. Not every user likes such pressure. Some users think pressure decreases morale when it shows their low work performance is behind schedule. However, the author and some other users feel that bad news motivates developers to higher work performance.

7 Conclusions

7.1 Conclusions

Release Planning is a management framework for commercial software vendors to estimate and track software releases. In the organizations applying Release Planning, the development group will break the features down into appropriately sized parts and assign them to developers. After that, developers measure the work effort of each feature. Then development managers collect data from each developer, and summarize it as the overall estimate for the release. If developers cannot make accurate estimates or cannot follow the estimates, there will be schedule problems. Releasing planning provides approaches at the organizational level to estimate and track software releases, and these approaches are based on personal estimation and tracking work. ECMT provides tool support for Release Planning at individual level. ECMT's features are divided into three categories. The tracking time feature records time spending on hierarchical organized tasks, and provides graphical views of the time spending. The estimation feature logs and analyzes the estimations of tasks as projects proceed. The combination of estimation and tracking time provides up-to-date progress measurement of projects.

ECMT provides features to monitor development progress as follows,

1. Tree hierarchy is appropriate for organizing projects and tasks.
2. Dedicated time is used to measure work effort.
3. Developers track their work time by clicking in track table, which also provides

some overall information, for example, total work time, today's work time, and so on.

4. Developers can review their time log in detailed text-based and graphical reports.

ECMT provides features to assist developers to make better estimate as follows,

1. Dedicated time is used to measure work requirements.
2. Work factor is used to measure work capacity.
3. Developers can review and analyze estimates in detailed text and graphical reports.

ECMT combines monitoring and estimation information:

1. Developers can use historical data as reference to assist current estimation.
2. Summary for the current project reflects its development progress by comparing the remaining work requirement and remaining work capacity.
3. Task to-do list devised by considering remaining work requirement and remaining work days motivates developers to put more effort on important tasks.

7.2 Future directions

7.2.1 Experiments

We implemented a prototype application, ECMT, in this thesis. However, some more experiments are required to validate that ECMT can achieve all the objectives it is designed for. The following provides some concerns in future experiments.

1. One objective of ECMT is to improve developers' estimation skills. Some experiments are required to validate it.
2. The schedule pressure of using ECMT is the key to motivate developers to work more productively. However, we need to investigate whether developers feel the

appropriate pressure when using ECMT, where they get the pressure (time reports, project summary, or to-do list), and how they react to such pressure.

7.2.2 Summary at project-level based on individual records

ECMT is targeted at one developer working alone. However, a software project is usually developed by a team, and the overall view of a project cannot be gained directly by having only the estimate data of each individual developer. Therefore, a new facility targeted at integrating the data from team members is desired.

A client-sever architecture will be appropriate for this feature. The client will implement most features of ECMT, but store all the data in the server. Besides storing data, the server can analyze the data at project-level including making a summary as follows.

By analyzing all developers' time logs and estimates, the new facility should provide a project report to the manager. This report should include two parts. One is the work summary, which is composed by the amount of work done and the average actual work factor from the start date of the project until now. This part reflects the past situation of the project. The other part analyzes the remaining work requirement and remaining work capacity. The remaining work requirement is the combination of the estimates of all developers, and the remaining work capacity is the combination of all developers' work capacities. Each estimate in ECMT is represented by a normal distribution, so their combination is also a normal distribution. With the following normal distribution,

$$D(T) = \textit{Remaining coding capacity} - \textit{Total remaining coding requirement}$$

(Where T is the remaining workdays to delivery date, *remaining coding capacity* is the combination of all developers' work capacities, and *total remaining coding requirement* is the combination of all developers' remaining work requirements), the report will provide the amount of confidence that the project will finish before expected delivery date, as well as the number of workdays required given a confidence. This report is an extension of the personal summary in ECMT. Since it calculates the data from all developers, it is able to reflect progress of whole project.

7.2.3 Using PDA to capture work away from desk

When developers work away from desk, for example, discussing with colleagues, thinking with paper, it is hard to record that time with applications in a desktop or even a laptop. However, a PDA (Personal Digital Assistant) is appropriate for these situations, since a PDA is small enough to be carried anywhere. Usually, a PDA has a small screen and limited computation capability, so it is not very suitable for analyzing complex data or visually displaying the analyzing result. Therefore, it is better to use a PDA to record the time spent data, but synchronize it with the desktop ECMT. In this way, we would not miss any valuable data, and get the same benefits as in ECMT.

7.2.4 Extendable modules to support estimation approaches

There are many estimation approaches available (section 3.2). Some of them can be integrated into ECMT, for example, function points and Delphi approaches. A pluggable module can be designed for each estimation approach, which provides related support for estimation approach and convert its result into *effective work days*.

That is feasible because ECMT only needs the estimation results measured by *effective work days*, but does not care which way users get these estimations by.

Reference

Albrecht, A. J., J.R. Gaffney (1983). "Software function, source lines of code, and development effort prediction: a software science validation." *IEEE Trans. on Softw. Eng.* 9(6): 639-648.

Beck, K. (2000). *eXtreme programming eXplained : embrace change*. Reading, MA ; Harlow, Addison-Wesley.

Blair, G. M. (1992). Personal time management for busy managers. *Engineering Management Journal*. 2: 33-38.

Boehm, B. W. (1981). *Software engineering economics*. Englewood Cliffs, N.J., Prentice-Hall.

Boehm, B. W., Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, A. Winsor Brown, Sunita Chulani, Chris Abts (2000). *Software Cost Estimation with Cocomo II*, Prentice Hall PTR.

Brooks, F. P. (1995). *The mythical man-month : essays on software engineering*. Reading, Mass., Addison-Wesley Pub. Co.

Brooks, F. P. (2003). "Three great challenges for half-century-old computer science." *Journal of the ACM (JACM)* 50(1): 25-26.

Covey, S. R. (1990). *The Seven habits of highly effective people : restoring the character ethic*. New York ; London, Fireside Book.

Covey, S. R., A. Roger Merrill, Rebecca R. Merrill (1996). *First Things First: To Live, to Love, to Learn, to Leave a Legacy*, Fireside.

Davidson, J. P. (1999). *The Complete Idiot's Guide to Managing Your Time*, MacMillan Publishing Company.

DeMarco, T. and T. R. Lister (1987). *Peopleware : productive projects and teams*. New York, NY, Dorset House Pub. Co.

El Emam, K. S., B.; Madhavji, N.H. (1996). *Implementing concepts from the Personal Software Process in an industrial setting VO -*. Software Process, 1996. Proceedings., Fourth International Conference on the.

Ferguson, P. H., W.S.; Khajenoori, S.; Macke, S.; Matvya, A. (1997). "Results of applying the Personal Software Process." *Computer* 30(0018-9162): 24-31.

H. Sackman, W. J. E., E. E. Grant (1968). "Exploratory experimental studies comparing online and offline programming performance." *Communications of the ACM* 11(1): 3-11.

Heemstra, F. J. (1992). "Software cost estimation." *Information and Software Technology* 34(10): 627-639.

Hughes, R. T. (1996). "Expert judgement as an estimating method." *Information and Software Technology* 38(2): 67-75.

Humphrey, W. S. (1995). *A discipline for software engineering*. Reading, Mass., Addison-Wesley.

Humphrey, W. S. (1997). *Introduction to the personal software process*. Reading, Mass., Addison-Wesley Pub.

Humphrey, W. S. (2000). "The personal software process: status and trends." *IEEE Software* 17(0740-7459): 71-75.

Koch, R. (1999). *The 80/20 Principle: The Secret to Success by Achieving More with Less*, Doubleday.

Ma, Z. C., J.S.; Smith-Daniels, D.E. (2000). *Causes and solutions for schedule slippage: a survey of software projects VO -*. Performance, Computing, and Communications Conference, 2000. IPCCC '00. Conference Proceeding of the

IEEE International.

Penny, D. A. (2001). Software Release Planning, Managing at the Boundary between Business Necessities and Software Development Realities. Toronto.

Penny, D. A. (2002). An Estimation-Based Management Framework for Enhancive Maintenance in Commercial Software Products. Toronto.

van Genuchten, M. (1991). "Why is software late? An empirical study of reasons for delay in software development." *Software Engineering, IEEE Transactions on* 17(0098-5589): 582-590.