Towards the semi-automation of the software development process

J.M. Bennett, D. Ho

Department of Computer Science, University of Western Ontario, London, Ontario, Canada N6A 5B7, and IBM SWS Toronto Laboratory, Toronto, Canada

Abstract

This paper describes an approach to improve the Software Development Process (SDP). By integrating current estimation tools and metrics collection under the paradigm of the IEEE Software Project Management Plan, we outline the framework of a new tool to help automate many of the troublesome aspects that currently plague project managers. We describe a prototype which confirms the feasibility of our approach and relate this to current and future management directions for organizations committed to improving their Capability Maturity Model level.

1 Introduction

Quality software development depends on having both a good Software Development Process and an accurate way of estimating the resources needed to complete a particular project. There has been great activity in this area in the last few years, with some notable isolated successes. The work reported tends to fall into one of three approaches: general suggestions for improving the SDP; the use of metrics such as function points (Function points is a language-independent sizing metric that is being increasingly validated as the only way of estimating code sizes. Its estimating applicability extends to all aspects of the software life-cycle. See [6].) [3], KLOCs, defect counts, etc., to quantify both the process and the product; and the use of estimation tools (such as CoCoMo[1], SLIM[5], and Checkpoint[6]) to help the software project manager plan the process. Unfortunately, these three approaches (and in some cases, their associated tools), are isolated, unrelated, and very user-unfriendly. They are accurately seen by SDP managers as unwieldy and hard to use (which is quite correct) and are thus ignored unless the manager is actually coerced by the company to use them. We contend that these three approaches are too intimately interrelated to be separated, and that the field of knowledge is now mature enough to combine them into a single tool that can facilitate the automation of the SDP. Our starting point was the IEEE Standard for Software Project Management Plan (SPMP)[4].

The SPMP specifies the format and contents of software project man-
Software Quality Management

agement plans. Using this standard, organizations can develop and tailor a set of practices and procedures to address the project organization, managerial process, technical process, work packages, schedule, and budget. These practices and procedures are among the first activities to be completed on a software project, and are key to the completion of successful software products. As the project evolves, the SPMP must be updated periodically to reflect reality.

In effect, we have used the SPMP as a template for the SDP, and have attempted to automate its application to the SDP to ensure both that the necessary SPMP steps have been included and that it is metricized appropriately. Taking the manager’s Work Breakdown Structures as input, we then use standard backend estimation tools such as SLIM and CoCoMo, to provide resource estimations, and then schedule the project using a standard package such as Microsoft Project. Thus, we can automatically generate a project schedule and resource allocation. Once this baseline has been established, it can then be used to automatically track project development, Earned Value Analysis, and other measures that the company culture might ordain. Defect metrics can also be inserted, reported, and archived for future use. Certain aspects of the SDP (for example formal code reviews after n Function Points), can be built into the model and used to produce a plan that causes the manager to both hold the reviews and to report on their results. Thus, most of the picky details of managing can be semi-automated, allowing the manager to maintain total project control with a minimum of administrative effort. In other words, our approach simplifies the project management overhead.

We begin with a survey of some of the difficulties in managing software. Next, we present an exposition of the SPMP, pointing out the advantages of using it in managing software projects, and give a few tips on utilizing its mandatory and optional materials for local customization. Then follows a description of current management techniques and tools used in the IBM SWS Toronto Laboratory development environment. We then introduce our new model for project management. We report on the results of a prototype that validates the model by combining parts of the SPMP with Microsoft Project. We conclude with a description of our general approach which provides a framework for assisting the software manager throughout the software development process. The results of this work will remove much of the tedium of the SDP, and make its management much easier and more defect-free. A by-product of this approach is that much of the material can be archived easily for future use in process and metrics refinement (in the SEI - ISO 9000-4 sense).
2 Problems in the Management of the SDP

Software construction is hard. We all know of “horror stories” of projects that have failed, that have been years late, that have had massive cost overruns. The chances are excellent that the project you are currently working on is in one of these three categories. Data of the last few years support our suspicions. More than a decade ago, DeMarco [2] pointed out that 25% of large system projects never finish; more recently, Jones [6] reported that the average MIS project in the USA is one year late and 100% over budget. We are slowly beginning to understand why this is so. Central to the problem is the inability to estimate times and costs correctly; and once estimated, to assess whether or not we are on target or if changes must be made, the effect that these changes will have to the schedule, and eventual costs. We need to be able to estimate accurately at all stages of the process.

Software estimation can be defined as the estimate of a system in terms of the duration and cost to build, staffing, and product quality. Project estimation and management are very important to software development because they are cross life-cycle disciplines that apply to all phases of the development life-cycle. The typical steps involved in estimation (and thus in project management) that contribute to the software development process are as follows:

1. estimate product size
2. estimate product schedule, cost, staffing, and reliability
3. develop product schedule
4. track product actuals during development, and revise estimate and schedule

The problems of software estimation are unique. Many project parameters are unknown at the initial planning phase of development. Others must be developed in a stepwise refinement basis; the final parameters being fixed only after being well into the development life-cycle. In addition, projects are influenced by management policies, user involvement, and the development organization. Because these factors change during development, the concerns of management, users, and developers often conflict with one another.

Moreover, we are normally unable to estimate the effect of requirements changes that will inevitably occur if the project lasts more than a couple of months. The lack of a disciplined approach to software estimation and project management further increases the risk.

What can be done? The enemy is being attacked on several different fronts. First, companies are insisting on formal, documented, repeatable
260 Software Quality Management

approaches in preparing the Software Management Plan for each project. Second, metrics can be defined and recorded that help in assessing quality control, the results of testing, the rate of defects, and so on. Third, estimating tools are emerging that are providing better and better results. Fourth, we are beginning to understand some of the underlying structural reasons that explain partly, why we cannot produce good software. We claim that the time has come to blend all of these approaches into a unified approach.

3 The IEEE SPMP

In the mid‘Eighties, as a formal response to the first of these four problems, the Institute for Electrical and Electronics Engineers (IEEE) formed a committee to recommend a standard to address the problems of software project management. This committee produced an IEEE standard 1058 called the Software Project Management Plan, or SPMP [4]. Not a standard in the communications sense, it is rather a guide to structuring the Software Project Plan to cover off the essential steps. In following the SPMP, the project manager can be sure that all of the important steps have been included when constructing the particular plan to be used. We have guided industrial projects using this approach with good success.

3.1 A Quick Tour of the SPMP

The SPMP is divided into five major sections, each with further refining subdivisions. Since we are concerned with the last subsection, we shall only list its subsections.

1. Introduction

2. Project Organization

3. Managerial Process

4. Technical Process

5. Work Packages, Schedule, and Budget

i Work Packages
ii Dependencies
iii Resource Requirements
iv Budget and Resource Allocation
v Schedule

We have concentrated on Section 5.
3.2 Some Comments on the SPMP

A strong advantage of the SPMP is that certain corporate cultural standards can be integrated into it. For example, once the Configuration Management standards have been set, that text can be bolted onto every SPMP the company produces. Second, the project manager is forced to either include each step of the plan, or justify why a step can be omitted. Third, if followed correctly, the plans can become part of the organizational planning of all projects. Thus it becomes feasible to construct a project management database for the SDP. Fourth, project managers can both be tracked in a consistent way and be second-sourced, so that replacement or rotation of managers is easy.

3.3 Experience with the SPMP

The core of preparing the SPMP is of course in doing section 5, the Work Breakdown Structure (WBS). Here is the meat of any project. This is where and when the project is modularized with the estimations, the staffing, the costing and the WBS interrelationships worked out. Like a book where the introduction is the last portion written, the rest of the SPMP is best worked out once the WBS has been finalized. Note that many subplans (such as the Risk Plan, the Documentation Plan, and the Quality Plan) are subsets of the SPMP.

3.4 Problems with the SPMP

Typically, once the WBS is complete the project manager will use a project management tool (such as Microsoft Project) to lay out the project schedule. The tool will typically calculate relevant Gantt and PERT charts, and the cost estimates, and even help in tracking the progress of the work. The problem here, and with other tools mentioned in the next section, is that there is no relationship between the SPMP and the tool used. Thus, when changes are made, the recalculations must be done separately from the updating of the SPMP. Because of the inconvenience of this and the constant schedule pressures that the project manager is under, these are often done at project completion time (if indeed the project completes!). Note also that the tool is only as good as the estimates that the project manager feeds into it, and estimating is very hard. Thus, many of our plans are flawed right from the beginning of planning.
Experience with Software Estimation Models and Commercially Available Tools

What is the current state of the practice? We shall take, as a typical example of a mature software enterprise committed to evolutionary improvement based on the Capability Maturity Model paradigm, the IBM Toronto Laboratory. The Software Engineering Institute (SEI) [9] self-assessment in 1991 revealed a critical need for software estimation techniques and tools in the Toronto Laboratory. In the absence of an internal model based on historical data from the laboratory, the search for one or more commercially available estimation tools was necessary. The SLIM and Costar tools were adopted, and have been used in the laboratory for the past three years on many projects with satisfactory results. No estimation model is perfect, and the use of more than one will make up for deficiencies inherent in the others. The following sections provide an overview of the CoCoMo and SLIM Models, and the tools that implement these models.

4.1 The CoCoMo Model and the Costar Tool

Costar [7] is a software estimation tool that implements the CoCoMo[1] mathematical model using refinements of certain preset factors. The model makes use of the effort equation as its fundamental calculation, using lines-of-code as the basic input.

\[ \text{effort} = k_1 \times (KDSI)^{k_2} \text{ staff-months} \]

- where \( k_1 \) and \( k_2 \) are constants whose values are dependent upon the mode of development
- \( KDSI \) is thousands of delivered source instructions

The effort equation is refined by multipliers from product, computer, personnel, and project parameters. The calculated effort also forms the basis for estimating the project duration and staffing.

Costar automates the calculation of estimates based on certain influencing factors. Costar 3.0 is currently used in the Toronto Laboratory. Costar 4.0 (the new release) also supports incremental development, which is typical for object-oriented software development. Our users’ response concerning the use of the Costar tool has been positive. Users are satisfied with the single-panel input with rich online help, the automatic recalculation capability, the support of subcomponents for estimation in a structured manner, and the availability of a wide variety of reports.
4.2 The SLIM Model and Tool

SLIM[5] is a metrics-based estimation tool developed using validated data from over 3000 projects from industry. This database represents over 200 million lines-of-code, 2 million function points, and over 200 languages, spanning over 15 years and containing over 55000 person-years of development effort. The projects are stratified into nine application categories, ranging from microcode to business systems. The products of the Toronto Laboratory fall mostly into the system software category.

The following gives the key equation for the SLIM Model:

\[ ESLOC = PP \times (PY/b)^{(1/3)} \times (Y)^{(4/3)} \]

- where ESLOC is executable source lines-of-code
- PY is effort in person-years
- Y is duration in years
- b is a special skills factor, which is a function of system size
- PP is a productivity parameter, which translates into the productivity index, PI.

The productivity index (PI) is a key concept of the SLIM Model. It is an overall measure of the total development environment such as management practices, skills and motivation of developers, tools and methods, user involvement, and project complexity. Each application category typically has a different PI average. Projects within the same application category may also possess a slight deviation in PI. It is important to establish a baseline for PI within an organization. Based on an internal translation table, SLIM then maps the PI onto the PP. The corresponding PP value can then be applied to the above formula for establishing a cost and time schedule for developing a system of that size.

The SLIM tool embodies the above model. It can be customized to a specific organization through calibration using historical data, and automates the calculation of the optimum solution based on all project assumptions and constraints. A rich set of what-if capabilities is provided to explore alternative solutions in changing time, effort, and cost. Users have been particularly impressed with the tool’s strong what-if capability and sensitivity analysis feature.

4.3 The IBM Toronto Laboratory Experience

The CoCoMo and SLIM Models have been in use for over three years in the Toronto Laboratory. Injection of the technology and tools is facilitated in the following ways:
1. calibration of the tools to the Toronto Laboratory development environment

2. deployment of tools on the Toronto Laboratory Common LAN for ease of access

3. availability of tool information in the Laboratory Experience Warehouse, a central repository of information

4. provision of consultation on the estimation models, assistance in tool usage, and advice on project estimates

5. publications of experience in technical reports and presentations in public fora to increase awareness

6. development of a laboratory-wide customized course teaching the theories and tools of estimation, with intensive hands-on real-project exercises.

At the time this paper was written, over 70 laboratory personnel have been educated on software estimation.

4.4 Problems Encountered and Foreseen

Although the CoCoMo and SLIM Models have been successfully deployed within the Toronto Laboratory, much work still remains. The following gives a summary of some problems encountered during the deployment process and foreseen in the future.

1. Faithful Input to the Models

To come up with accurate estimates, software engineers must faithfully input the product, computer, personnel and project parameters to the CoCoMo Model, and the PI value to the SLIM Model. In practice, they are reluctant to provide details, especially those related to productivity and management policies. It should be noted that these inputs are key for identifying strong points and bottlenecks, for measuring process maturity, and for setting goals for process improvement.

2. Data Collection from Projects

There is a crucial need to continuously collect historical data and compare it against the estimate given by the models, in order to improve the accuracy of subsequent estimations. There seems to be much resistance to capturing this data, because people are afraid of how the numbers will be used or misused by management.
3. Bridging Software Estimation and Project Management

Software estimation is a prerequisite for and an input to project management. Although some estimation tools have interfaces to project management tools, the gaps in standardization, disciplined approaches, and processes are still unbridged. Software estimation and project management should be more tightly integrated. (Tools integration alone is insufficient!)

4. The Next Step in the Capability Maturity Model (CMM)

There are three distinct ladders to realize the five levels of maturity in the Capability Maturity Model (CMM):

(a) product/project-specific
(b) process-specific
(c) process evolution

Each ladder has three steps: intuitive, measured, and fact-based management. They are represented in Figure 1.

![Diagram of Capability Maturity Model levels]

Figure 1: Realization of the Capability Maturity Model

The work accomplished by the Toronto Laboratory in the area of software estimation throughout the past few years has been the advancement within the product/project-specific ladder from the intuitive to the measured level. What needs to be done to achieve a higher score? It is not necessary for an organization to climb every step in each ladder prior to
reaching the next ladder. What is the optimum path for the Toronto Laboratory to follow to attain higher maturity in less time, with minimal investment, and at a low cost? The next section will attempt to address some of these problems.

5 A New Proposal

We propose to combine, in a seamless way, the SPMP with one or more tools to support “backend” estimations. Once the life-cycle model has been selected, we can extend the SPMP to insert the WBS steps that derive from that model. For example, suppose that a typical Waterfall model is selected, with inspections to be performed every 100 function points. Then the project manager starts the SPMP by defining the first-level WBSs. Once defined, the augmented SPMP tool will prompt the project manager to supply estimates for the size of each module. Based on the function point estimate and the historical company data, all of the derivative estimates (documentation, design, coding, testing, etc.) will be generated, along with the WBS fine-structure modules to support this. The project manager will be asked for resource approvals and the like. These figures can now be fed into a standard tool for generation of budgets and schedules. Inspections, walkthroughs, and so on can be generated automatically according to the company’s SDP.

Each phase of the project will ask for defects and other phase-dependent metrics, again depending on the template for measurement. The project manager will be prompted to insert these, and they will be automatically added to the historical database.

Now when the inevitable requirements changes come, the project manager will be able to insert these changes and get a revised estimate for downstream effects on budget and time extensions, and have automatic progress reports generated. If the company requires Earned Value Analysis and the corresponding “S” curves, these can be automatically generated and red-flagged for management if necessary.

Regarding the problems addressed in section 4.4, all of these difficulties can be overcome by our approach. In particular, the groundwork for climbing the CMM ladders can be easily laid and supported with our tool. The number and function of those rungs now need to be addressed in a timely manner if the ascent is to occur.

6 A Simple Prototype

To test some of these ideas, a simple prototype was constructed at the University of Western Ontario[8]. The prototype consisted of seamlessly combining an existing tool (Microsoft Project) with a program written in
Visual Basic. Using the DDE (dynamic data link) facility of Windows 3.1, the investigators were able to combine the two modules so that the Visual Basic program used the project management tool as a backend driver. Thus the user could insert project changes such as a project manager would do on a real-life project, and have Microsoft Project automatically use the changes to generate new Gantt and PERT charts plus new project estimates. They could also generate “S” curves, which Microsoft Project cannot presently do. Some of the limitations of Microsoft Project that made it difficult to use were sidestepped with a friendly user interface using Visual Basic (see [8]). Additionally, we were able to correct actual errors. When resources are assigned to a WBS, Microsoft Project only uses the first resource to calculate durations. Thus, adding additional resources has no effect on the time! We changed that to operate correctly. Moreover, the entire interface was much easier to use. One of the constant complaints about Microsoft Project was that it took too long to insert data, and many operations, if even possible, were arcane and not easily accessible. With the new interface (which includes a constant help bar to help users navigate), users can insert and change data in an intuitive manner. Thus we think that we have retained the excellent calculation aspects of Microsoft Project while avoiding one of its major difficulties; namely that since it is so clunky, project managers will not use it unless coerced.

It is useful to remember that Microsoft Project sits in the background. As changes are made by the project manager, they are reflected immediately in both modules. Split screens can show both visual panels at the same time if desired.

The thesis showed that such a scheme is quite practical and can be built rather easily. Thus we can customize Microsoft Project in any manner appropriate to the organization. We can also escape into Access, Excel, or any other product supporting DDE. We are currently extending the work as mentioned above to implement the SPMP paradigm, to incorporate other tools such as the Lab uses, and finally, to construct a company-wide database for project metrics. As the new interface is easy to use we feel that we can provide a painless way for managers to be motivated into entering relevant data as the SPMP dictates into the metrics database for later analyses.

7 Conclusion and Future Directions

We believe that this seamless integration is the way of the future for the SDP. In addition to normal estimation metrics, new results on quality control can easily be included. Much work is needed to capture more historical data and to sift through the several tools to find the one best suited to the type of development work at the Lab. But we believe that this holds a
significant key in solving the problems of cost and schedule overruns that plague our industry.

References


Please note that the actual names of products mentioned are trademarked.