MANAGEMENT TOOLS ASSOCIATED WITH THE DEVELOPMENT OF COMPUTER SOFTWARE PRODUCTS

Csaba J. Egyhazy
Computer Science Department
Virginia Polytechnic Institute & State University
Falls Church, VA 22042

CS830015

Management Tools Associated with the Development of Computer Software Products

1.0 INTRODUCTION

During the past decade there has been a growing realization that the many tools designed and created to assist the software development organizations and more lately the software maintenance activities have basically failed. This failure is manifest by a lack in reduction of development costs, no significant increase in programmer productivity and the software's delivered quality remaining at or near the same levels. Upon investigating this failure it is clear that the blame resides with the actual tools developed. Most of these tools were developed, and thus focused their support, along very narrow aspects of the software development process, such as implementation or testing, while ignoring all others. This inability to provide a real and comprehensive support base for the software development effort is the main contributing factor to this failure. In addition, most tools force the user to adapt to the incantations and protocols of the tools and the philosophies of their creators.

Clearly, this situation would improve if tools were configured to be continuously supportive to the user in each phase and role of the software development process [10]. Such a supportive configuration of tools, work place and equipment is referred to as a software development environment. The essence of these software development environments is the synergistic integration of tools in order to provide a strong, effective support for a software job.

This paper addresses some of the tools associated with the management of software development projects/programs.

2.0 PROGRAM/PROJECT DEVELOPMENT TOOLS

The following section describes the tools associated with software development, programmatic tools and automated techniques. A specific requirement for each tool is provided. These requirements relate the software engineering methodologies to good engineering management practices. Secondly, any known examples of tools that partially or completely address the techniques are discussed as to capabilities and availability. Then any problems, limitations, or restrictions on the use or usefulness of the tools are provided. Finally, a discussion of trends and future activities are provided.

A number of software engineering techniques have been found to be highly cost effective on large software projects, such as early project planning, thorough requirements and design specification and validation, early development of user's documentation, use of unit development folders and configuration management techniques, and independent project testing. In general, these techniques require a large amount of documentation and early non-programming activity on large projects. More importantly, to be effective they must be well planned and be capable of being assessed. Too often the knowledge of what to do is available to the project, occasionally an adequate amount of planning effort is expended to provide a good basis for the project, but rarely do the projects measure and track performance to that knowledge and those plans with the same detail.

This is not because it is not recognized as important, but rather that it is hard work, and small perturbations can result in significant rework and replanning. Too often this is neglected or

given a lower priority to the more interesting management and development tasks. Unfortunately, neglect in this area results in problems, often major, downstream. Therefore, automated techniques to assist in the project planning (scheduling, task assignment, critical path analysis, and performance measurement) are critical to a strong development environment.

There are extensive lists of scheduling and performance measurement software packages available. The following identifies a few representative units.

The Critical Path Analysis [3] program assists in the planning and management of a project where the interrelationships between activities and functional areas are numerous and involved; if the responsibilities for various phases are fragmented or widely spread; or if the project is so large or complex it is difficult to visualize as a whole. This program assists in project management by using the systems concept for planning, scheduling, and control of an organization's project objectives and plans.

PERT TIME III is a Program Evaluation and Review Technique system that aids in the monitoring and scheduling of the various activities with a particular project. The system can effectively control the project in the areas of time, cost, and manpower. It points out current and potential problem areas and allows for adjustment, refinement, and rescheduling of effort.

The ability to create this level of planning tool is obviously within the scope of many software firms. What is needed is a consensus on the actual performance capabilities for such a system.

Then capabilities should be developed from a documented software engineering development methodology.

2.1 Software Cost Estimation

It should be noted that even with the most optimistic view concerning scheduling and performance measurement, these techniques will only help a project if the initial estimates of the cost, and realistic cost profiles by task, have been used as the basis of the initial plans [8], [9]. Two examples of this type of project management tools are Software Cost Estimation Program (SCEP) and the Constructive Cost Model (COCOMO).

SCEP applies to projects (greater than five man-years) producing deliverable software. It calculates development effort in man-months beginning from a baselined set of software requirements established at a Software Requirements Review (SRR), and ending with the completion of software acceptance testing.

To use SCEP, a software system must first be partitioned into subsystems, and the subsystems into units. Then each subsystem and unit must be described in terms of a set of cost driver ratings. These cost drivers include the source code language, execution time constraints, storage constraints, data storage constraints, computer access, required development on a target machine, computer experience, applications experience, required quality, security environment, requirements volatility, use of software tools and a basis of estimate. Once these data have been input, the program develops cost estimates for the system, subsystems, and units by each phase and for the total project.

The Constructive Cost Model (COCOMO) [8] is a partly analytic, partly table-driven model. It accepts descriptions of software components in terms of their size and their ratings with respect to 16 cost driver attributes (e.g., hardware constraints, data base size, required fault-freedom, personnel experience, use of tools, and required programming practices). It uses these to calculate the amount of effort (and resulting dollar cost) required to develop each component and the overall system, and provides a breakdown of the effort and cost into four major development plans.

The use of these tools, and others like them will depend on a more formal description of the metrics that are to be applied and how to better estimate their weights. This should result in better cost estimation by reducing the subjectiveness of the inputs.

2.2 Configuration Management

Software configuration management [11], [2], is designed to ensure an orderly control of software products produced in the software development process and to provide an effective mechanism for incorporating software changes, both during development and during program operational use. These objectives are addressed through the following activities:

(1) Establishing approved baseline configurations for the computer programs and supporting documentation;

(2) Maintaining configuration change control over all units under baseline control;

(3) Providing traceability for all modifications and reported problems to all units under baseline control.

These activities, if performed manually, are labor intensive, dull, and thus prone to human error. They also are well matched to automation.

The following example is the closest commercially available system. It is an inventory control package used to implement an inventory management and material planning system for manufacturers. It is divided into four phases. The inventory planning phase provides for the analysis of inventory items based on usage and cost. The projection phase analyzes historical demand data and establishes estimates of demand for future time intervals. The requirements planning phase determines requirements for component parts, subassemblies, and raw materials. An output report provides detailed requirements information, and an exception report highlights problem areas. The execution phase processes input transactions to update the item master record.

2.3 Documentation

Concurrent documentation is required for each task in the software engineering methodology framework [5]. The documentation walks are in step with the development process and for the first part of the project is the only product.

Documentation preparation and processing, word processing, and document control software abounds [4]. Presently multiple volumes of references to this type of code exist [1], [4]. Specifics will not be provided on any standalone system; however, the following description relates to a document generator for Digital Equipment Corp. (DEC) computers and thus is provided as a representative class

of software that many machine vendors and other software houses are currently marketing.

Runoff [11] provides an automated approach to preparing, maintaining, and producing narrative documentation. The package provides many capabilities of a text-editing system but does not require the learning of complex editing rules or commands. Output is directed to the computer printer in a format suitable for distribution or reproduction.

The Runoff package consists of two programs; the documentor program and the concordance program. Input can be recorded on cards, tape, or disk. In addition, input from the source statement library can be specified. The input consists of the original user test and control information. Through simple parameter control, the user specifies the number of copies, lines per page, data insertion, revision number indicators, headers, fill characters, spacing, and the like. From this input, the Runoff program builds (in upper- and lower-case characters, if desired) a table of contents, finished text, and an index. The concordance program accepts any input that is acceptable to Runoff and provides an alphabetical list of all user-supplied words with card sequence numbers of each word's occurence. The primary function of this output is to help the user prepare tables of contents and index entries and detect spelling errors.

The problems with this type of software is closely related to that of scheduling and performance measurement. In particular, the majority of these systems, especially those with significant capa-

bilities, require separate systems of hardware and software or require a significant amount of the mainframe resources. In addition, traceability from document to document are not generally available. Other similar documentation support systems are SOLID [16] and DOCUMENT [12].

In discussing an integrated development environment, Newman [7], expects the documentation system, among other things, to be consistent from one level to the next and able to pass documentation among levels. Thus, from this novel perspective, the collection of tools and methodologies can be thought of as a documentation system.

2.4 Project Management

The technological and managerial aspect of software engineering are both recognized and supported. However, improvements and developments in management have not kept pace with advances in the technology of software development. Although the methodology of software engineering as a discipline is relatively new, software engineers have progressed to the point where many major issues relevant to software production have been identified. Practical working tools to support improved software production are commonly available.

Software engineering project management has not enjoyed the same program. Where it might be argued that approaches to project management have been defined, they are far from consistent and do not provide a recognized discipline. Software developers who have demonstrated competence as developers and programmers have been elevated to project managers without the benefit of education or train-

ing. The major issues and problems of project management have not even been agreed on by the computing community as a whole, and, consequently, priorities for addressing them have varied widely among the organizations and institutions that wish to see them established [14]. The following examples address a few of the project management efforts that have been automated.

The Scheduling and Resource Management System (SRMS) [5] is an interactive decision-support system for planning and controlling projects and resources. Its uses range from project management and manpower loading to requirements analysis studies and zero-based budgeting systems. The system offers an interactive command language and three modes of operation. The system includes project monitoring, resource constraints, support of precedence networks, and CPM analysis. In addition, the modeling and simulation component accommodates 'what if' analysis and comparison reporting.

A Systems Dynamics (SD) approach, presented in [1], is used to analyze several key dynamic software project scheduling issues. SD, based on a number of computer simulation techniques, is the application of feedback control systems principles. The SD approach begins with an effort to understand the system of forces that has created a problem and continues to sustain it. Relevant data are gathered usually from a variety of sources (literature, informed people). As soon as a basic understanding has been gained, the model is developed. The model starts off as a set of logical diagrams showing cause-and-effect relationships. As soon as is feasible, the visual model is translated into a mathematical version. The model is

refined by a process of iteration, while a final version is agreed upon.

The past efforts [9] and future activities [1] [6] are directed toward the large project management efforts. Neglecting those that pertain to the needs of medium and small size projects.

2.5 Automated Office

The essential components of any information-handling system are Input, Process, Storage, Distribution, and Output. These functions are easy to identify in a standard software development. The input invariably consists of words that are captured initially in some vague medium such as handwritten drafts, memoranda, requirements and statements of work, or technical direction. The processing function can be more aptly described as a conversion function because its main purpose is to convert these unstructured inputs, which are not universally understandable, into formatted, documented, universally understandable outputs. The outputs are generally typewritten original documents with some number of copies. One or more of the copies is usually filed centrally while the remaining copies of the output must distributed among the project and user personnel.

The literature contains volumes of information concerning text processing, word processing, and the automated office [13]. Therefore, rather than provide one or more examples of these systems it seems that a list of commonly incorporated features is of more interest. Some of the more useful features for a software development facility and environment are the following: electronic mail, optical character reader, automated calendar, telecommunications,

printing and duplicating, automated libraries, automated filing and retrieval, and training.

Software engineers, systems analysts, project managers and programmers alike work in relatively unstructured situations; they produce and communicate ideas. Therefore, not every office automation system satisfies the needs of software development environments; an example of a system that does is Xerox's Star Professional Workstation [16]. An approach to supporting work in the office using and extending ideas from the field of artificial intelligence (AI) is being implemented at MIT [17]. In the software development view of problem solving, the designer or programmer has a well defined initial state, for example the configuration of an inventory control subroutine, a well defined state, for example to add a feature to calculate the cost of unused storage space, and a finite collection of actions or state transformers. The characterization of problem solving is thus being defined in terms of a problem solving process that can be aided by a knowledge based system. It consists of a database with information about similar situations reported by experts in designing/implementing inventory control systems. The knowledge based system then screens the information and provides the user with glimses of alternate ways of solving the stated problem.

Research in this direction will, if tailored to the needs of software development environments, provide the foundation for the creation of new and valuable tools for managing software products.

3.0 TOOLS FOR TYPICAL SOFTWARE DEVELOPMENT ENVIORNMENTS

The approach to program development through the use of a stong standardized software development methodology can be applied to all levels of software development. The difference between the small 5 man project and the large 500 man project is only in the formality and structure applied to the process. Therefore, the support environment conceived here is based upon a single methodology with automated tools replacing manual methods only when it is cost effective to do so. The integration of these tools and their application must provide a single and uniform user interface that is the same in a manual approach to an automated one. The programmer then deals with his project uniformly in terms of software engineering methodology.

Modularity of the methodology and the application of tools according to that methodology provides the needed consistency. Modularity also restricts the scope for each development verification stage making it much more simple and efficient. The fact that a tool resides in a well defined environment, i.e., specific tools exist and they are applied at appropriate times, can also be used to simplify the design and implementation of that tool. To enhance this cooperation all tools share a common program representation -- the syntax tree representation and a common database. This avoids the usual necessity of constructing one representation in terms of another or of having to develop interface code.

The major impacts of this approach to a software development environment is its integrated approach and the fact that it provides an interactive environment (either manual or automated) based on

a single software engineering methodology. The development environment is now viewed as a single system rather than as a set of independent tools. There is a smooth transition between the tools and the phases of the project. In addition, the project control and management is an integral part of the initial planning and continues throughout the development. Lastly, the tools are generally not visible to the developer or manager as they require the same inputs as do the manual processes and thus their existence or non-existence on a project has minimal impact.

Four classes of software development environments are being proposed. Each can now be built using state-of-the-art technology. The environments were designed by analyzing the life-cycle products generated during a software development project. The environments contain tools and techniques that can be used to construct those products. Four classes of software projects were also considered to match the environments, a small-sized data processing project, a medium-sized signal processing project, a large-sized data processing project, and a large-sized embedded real-time system.

The tools and techniques included in the four environments are summarized in Table 1. Tools are identified as either manual (M), automated (A), or both (M/A). The environments are consistent not only in their use of the individual tools and techniques listed in the tables but also in the software engineering database. The database is the basic unifying, integrating component in each of the environments. Environment 1, which is the most austere from the view of automated tools and techniques, is still expected to develop

and maintain a database of all the information from the project. Environment 4 with its full complement of tools demands that the database exist from the very start of the project.

As may be noted, two classes of tools and techniques were purposely exclused from the proposed environments:

- o those beyond the current state-of-the-art -- such as transformational programming systems; and
- o those whose range of applicability is highly restricted or whose cost-effectiveness is open to question -- such as symbolic evaluation and program mutation testing.

4.0 CONCLUSIONS

The tools identified relate to the chosen software engineering development methodology and are designed to indicate typical examples and are certainly not complete. However, the generic individual tools exist as presented in this paper and the software engineering development methodology that was selected is one that is generaly recognized as a standard approach [11].

Automated tools to support software development environments should, in most cases, be understood as facilitators of activities within a well defined software engineering methodology. Furthermore, they must be integrated with the program/project management practices and the technical areas associated with the methodology.

The need for a structured methodology for the development of software systems is widely recognized. Part of a current effort of a research group at Virginia Polytechnic Institute and State University (VPI & SU) is directed toward the creation of a system development methodology that integrates methods of human factors analysis with software engineering for improved management of the software development process.

Tools

	Environments			
Size: / Type:	Small Data Processing	Medium Signal Processing	Large Data Processing	Large Embedded Real-time System
I. REQUIREMENTS ANALYSIS				
Charts and Diagrams	M	A	A	A
Prose Specification	A	A	A	A
Test Plans	M	M	A	A
Machine Readable Specification	M	M	A	A
Specification Archive	M	M	A	A
Specification Cross-reference	M	A	A	A
Specification Traceability Matrix	M	A	A	A
Specification Review	M	M	M	A
II. DESIGN				
Functional Design Specification	M	M	A	A
Design test Plans	M	M	A	A
Data Dictionary	M	A	A	A
Program Design Language	A	A	A	A
Interface Verification	M	M	M	A
Module Cross-reference	A	A	A	A
Design Specification Archive	M	M	A	A
Design Reviews	M	M	M	A
Design Build Development	M	M	A	A
III. PROGRAMMING				
Text Manager	A	A	A	A
Source Code Manager	A	A	A	A
Program Cross-reference	A	A	A	A
Program Test Plans	M	M	M	A
Configuration Management	M	M	A	A
Source Code Format	M	A	A	A
Source Code Debug	M	M	A	A
Program Archive	M	M	A	A
Code Walk-throughs	M	M	M	A/M
IV. Verification				
File Comparator	A	A	A	A
Test Criteria Analysis	M	M	A	A
Functional Verification	M	A	A	A
Unit Verification	M	M	M/A	A
Test Data Generation	M	A	A	A
Performance Monitor	M	M	A	A
Control Flow Analysis	M	M	A	A
Data Flow Analysis	M	M	M	A
Test Archive	M	M	A	A
Re-test Determination	M	M	A	A
Test Conduct	M	M	A	A
V. MANAGEMENT				
Scheduling	M	M/A	A	A
Performance Measurement	M	M/A	A	A
Project Plan	M	M	A/M	A
Project Status Report	M	A	A	A
Project Financial Report	A	A	A	A
Configuration Management	M	M	M/A	A
Documentation	A	M/A	A	A
Office Function	M	M/A	A	A

Table 1

REFERENCES

[1] Abdel-Hamid, T. and Madvick, S. "The Dynamics of Software Project Scheduling" Communications of ACM, Vol. 26 No. 5 (May, 1983).

[2] Bersoff, E. H., Henderson, V. D., and Seigel, S. G. Software configuration management: A tutorial. Computer, 12, 1 (January, 1979), 6-14.

[3] Moder, J. and Phillips, C. Project Management with CPM and PERT, Van Nostrand Reinhold Co. (1970).

[4] U. S. Department of Defense, "Automated Data Systems Documentation Standards," DODI 7935.1, (September, 1977).

[5] Interactive Logic Sales Brochure, "SRMS - The Scheduling and Resource Management System."

[6] Kernighan, B. W., and Mashey, J. R. The Unix programming environment. Computer, 14, 4 (April, 1981)

[7] Newman, P. S. "Towards an Integrated Development Enviornment," IBM Sy. J. 21, 1 (1982).

[8] Davis, C. G., and Vick, C. R. The software development system, IEEE Transactions on Software Engineering, SE-1, 1 (January, 1977).

[9] Boehm, B. W. "Software Engineering," IEEE Trans. Comput., Vol. C-25, (December, 1976).

[10] Reynolds, C. H. What's wrong with computer programming management? in On the Management of Computer Programming, G. F. Weinwurm, Ed. Philadelphia, Pa. Auerback, pg. 35-36, (1971).

[11] Jensen, R. and Tonies, C. Software Engineering Prentice-Hall (1979).

[12] Girill, T. R., and Luck, C. H. Document: An interactive online solution to four documentation problems Commmunications of ACM, Vol. 26, No. 5 (May, 1983).

[13] Egyhazy, C. J. "Microcomputer based DBMS in Support of Office Automation," VPI & SU Computer Science Department, Technical Report CS830005 (May, 1983).

[14] Davis, C. G. and Vick, C. R. "The Software Development System," IEEE TOSE Vol. SE-3, No 1, (January, 1977).

[15] Brooks, Jr., F. P. The Mythical Man-Month Reading, MA: Addison Wesley, (1975).

[16] Purvy, R., Farrell, J. and Klose, P. "The Design of Star's Records Processing: Data Processing for the Noncomputer Professional" ACM Transactions on Office Information Systems, Vol. 1, No. 1, pgs. 3-24 (January, 1983).

[17] Barber, G. "Supporting Organizational Problem Solving with a Work Station" ACM Transactions on Office Information Systems, Vol. 1, No. 1, pgs. 45-67 (January, 1983).