A PROJECT ENGINEERING TOOL TO ASSIST IN THE DEVELOPMENT AND MAINTENANCE OF PROJECT LIFE CYCLES

S. Chandrashekar, B. Mayfield, and M. Samadzadeh

Abstract

This paper discusses the existing problems in developing and maintaining project life cycles and proposes a tool as a possible solution. This tool is part of an integrated set of tools, called Project Engineer, which has been designed to support the maintenance of software projects from a Project Management perspective. A PC platform is chosen and the user interface is based on Microsoft Windows 3.0. One module is selected for discussion and demonstration of the concept; the Life Cycle Builder. The purpose of this module is to manipulate project life cycles and templates. Project Engineer uses Multiple Document Interface, which is an IBM SAA/CUA (Systems Applications Architecture/ Common User Access) standard provided by Microsoft Windows 3.0. It also uses the Objects, Properties, Roles, and Relationships (OPRR) based data engine. Each module in Project Engineer is internally independent of the other modules. Coordination among the modules will be provided by the OPRR project database engine.

1. Software Development Life Cycle

Software Engineering encompasses a variety of technical methods, a set of management procedures, and a suite of automated tools (often called CASE - Computer Aided Software Engineering) that enhance the ability to build effective computer-based systems [8]. A (software) project goes through several stages, from project initiation to implementation and maintenance; these stages are often collectively referred to as the software development life cycle. A stage is comprised of a group of steps, and a step is comprised of a group of tasks. Tasks can be further divided into actions for further refinement. This partitioning of a larger module into several small modules is referred to as a Work Breakdown Structure (WBS) [12].

2. Statement of the Problem

CASE is a combination of software tools and structured development methodologies [4]. Whereas the tools automate the software process, the methodologies define the process to be automated. There are two aspects to a software project: techniques and planning. There are several tools to automate the techniques part, in particular, the CASE tools which support the Programmer/Analyst/Designer. However, the automation of methodologies along with planning and management aspects usually are not supported in a satisfactory way. This is customarily accomplished manually by project managers, which makes manipulation and maintenance cumbersome. A tool to assist in these management aspects could increase the productivity of the project manager, while providing a proper foundation for the project from the start. There are tools which allow the user to perform project scheduling, drawing of PERT/Gantt charts, etc. Currently, support for the Project Manager is limited to such single-purpose tools.

Proper management of a project life cycle leads to better project planning and easier maintenance. Software projects range from design and development of simple to complex systems. Each project has a separate Work Breakdown Structure. When managers try to plan a project without being able to reference an internal corporate project history file, a methodology checklist, or even a book, they may simply be stymied about where to start [12]. Unless managers have some pre-existing template of project plans, the first step may prove to be very difficult. An existing template can offer much needed help in this first and most important step.

This paper reports on the design, implementation, and testing of a Computer Aided Project Engineering (CAPE) tool for assisting in the development and maintenance of project life cycles from a Project Management perspective. This tool allows the user to load and manipulate project life cycles and templates, view the project from various perspectives, and provide on-line method help (hypermedia based), with an Export link to scheduler packages. Emphasis is given to the design of the user-interface component of this tool, keeping in mind the IBM SAA/CUA standard [2]. The tool is called Project Engineer. Project Engineer uses Multiple Document
Interface (MDI) child windows to display information to the user from different perspectives. One of the advantages of this approach is that multiple tools can be invoked at the same time and the interaction among tools can be established while providing the ability to display several windows (tools) simultaneously. A pictorial representation of the Work Breakdown Structure is also provided for the user to view the project hierarchy. In the rest of this paper, the design and implementation of Project Engineer is discussed further and a detailed explanation of the Life Cycle Builder module is provided.

The Life Cycle Builder (LCB) module is an MDI child window in Project Engineer. It uses the Grid Window class to display project information in a tabular fashion. It enables the user to bring up the Object Editor (another MDI child window) to help the user in editing any selected object. Interaction between the Object Editor window and LCB window is provided by MDI and the life cycle (another MDI child window) to help the user in editing any selected object. Interaction between the Object Editor window and LCB window is provided by MDI and the database. A pictorial view of the hierarchy of the project life cycle (WBS) is displayed in another window.

3. Object Oriented Design

Project Engineer employs an object-oriented approach in the design of its modules. The isolation of the interactive part from the application part in any software design has several advantages [3]. This isolation can be achieved easily using object-oriented design providing abstraction and encapsulation. There is a subtle separation between the front end, which is what the user sees and uses, and the back end or the database.

The concept of reusability is emphasized in this design. Encapsulation facilitates the construction of entities that can be depended upon to behave in certain ways, and this is done by grouping both data and operations that affect that data [11]. Such entities can be reused in every application that make use of this behavior and knowledge. Microsoft Windows allows the creation of Windows Classes, which, once coded, can be used for several different purposes. The Grid Window Class described below is developed as a Windows Class. Another window, the Object Editor, is also developed as a Windows Class. Each instance of the Object Editor has its own in-memory storage area.

4. Templates and Projects

Different methodologies use different strategies for breaking up a project into several parts. A project can be divided into several stages depending upon its complexity. Thus a small three-month software project may have just three stages and another two-year project may have eight stages. Hence, several "templates" are be provided in Project Engineer for the user to cater to the different profiles of project life cycles. The need for a template application occurs when certain groups of tasks must be used over and over again within the same project or across several projects [12]. These templates present a starting point for the user to build the required life cycle.

Two types of files exist in Project Engineer: Life Cycle Templates and Project Databases. There is no internal difference between the two types (except in the DOS file extension) and both types are based on the same OPRR Meta database as described in section 7 (both are instance databases).

A Life Cycle Template is used as a basis for a Project Database. It is modified only when changes that affect all future uses of the template are being made. The user is also allowed to make changes to a Template and save it. This facility is useful while handling projects which follow a pattern but differ slightly. To create a new project, a template is loaded, modified, and then saved as a project.

5. Project ViewPoints and Task Hiding

ViewPoints is a feature than allows the user to restrict the view of the project database to that of a single project team member. This is the basis for the report that can be handed to a project team member as his/her personal project plan including action items, estimates, resources, etc. The ViewPoints facility gives precisely the information required by hiding redundant or extraneous data.

The objects in Project Engineer are never physically deleted. Instead they are marked as deleted and 'hidden' from view. There are two reasons for this, the primary reason is that certain other modules in Project Engineer like the Life Cycle Advisor and Validator may need to know what objects are 'missing' as well as those that are present. Further, the user can easily 'undelete' a task at any point in time, without having to recreate all of the associated information. A Hide/Unhide function will allow the user to toggle between views that include or exclude 'deleted' objects.

6. User Interface Design

Recent studies have shown that the user interface forms a significant part of any application [6]. It is also arguably the most difficult part to develop, since it is necessary for the designer to understand the problem technically considering the human factors issues involved. The user interface of any software package is that part which accepts input, interacts with the user, and presents him/her with a friendly environment. It has to be designed in such a way that it makes the interaction between the use and the system easy and intelligent; with good response time and efficient use of devices.
The design of the user-interface for this project is generally inspired by the IBM SAA/CUA standards. The design adopts all the applicable features listed in the SAA Advanced Interface Design Guide. Direct manipulation interface design along with form fill-in approach (for certain parts) is used here [9], [5].

Project Engineer uses Multiple Document Interface (MDI) windows to display information from different perspectives. MDI Child windows are windows which are controlled by and appear within a Parent or the main window [7]. Each MDI Child window performs a function and can be 'iconized' and brought back to full size whenever needed. This interface is used by many other Windows products (e.g., Microsoft Excel and Word) and should seem familiar to experienced Windows users.

7. The Project Database

A repository is a mechanism for storing and organizing all information concerning a software system [4]. The purpose of a repository is to store system information at a central place, keep the data uniform, and be accessible to all users. The repository must cater to the needs of large software projects and must be scalable.

Project Engineer will use an Object, Property, Role, Relationship (OPRR) Meta database model. A Meta model is the "database schema" for a CASE repository [10]. It describes the structure and meaning of information that can be stored in the repository and shared among CASE tools. The OPRR data engine actually uses two kinds of databases in its operation: the Meta database and the Instance database. The Meta database defines the objects stored in the database, i.e., Activities, Description of tasks, Status and Schedule information, etc. An Instance database contains the actual data about a project. This closely follows the object-oriented paradigm. For example, the Meta database can be considered as a class, and the Instance database an instance of that class.

Thus the Meta database approach imparts flexibility by supporting multiple methodologies and lets the user customize his/her own methodology [10]. Also it fully supports future evolution of the project and lets the user add more analysis and reporting functions.

8. The Grid Window Class

One of the most frustrating aspects of GUI applications is that the control mechanisms of the user interface are not always consistent. While window, menu, and dialog box controls are fairly standardized, other user interface controls are not. Designers must always balance the unique user interface requirements of their application against the benefits of adopting a familiar control mechanism. Several control designs were evaluated/prototyped including a list-box based control and an edit-control based control. Finally, the decision was made to develop a Grid Window class because it is more user friendly and has been generally accepted by the users of other products (as in Microsoft Excel and Wingz spreadsheets).

The Grid Window Class is designed to provide an interface for displaying and manipulating tabular data. Since the data associated with projects will have to be displayed from various perspectives with each row of objects/tasks representing a Work Breakdown Structure object, a grid structure would be appropriate. Columns indicate the Name Status, Estimation information, etc. Columns can be customized by the user to view what the user wants at any point in time. The columns are resizeable with the use of a mouse. The resizing of columns is achieved by direct manipulation. Scroll bars are provided to scroll the grid vertically or horizontally.

Rows and columns can be 'highlighted' as in Microsoft Excel. A highlighted rectangle appears whenever the user selects a particular grid 'cell'; this action will remove the highlighted rectangle from the previously selected area. Simultaneously, the contents of the selected cell will be displayed in an 'edit' window where it can be modified.

Since this grid structure design allows the display of information in a matrix fashion and is designed and implemented as a Windows Class, it can be reused in other modules by just creating another instance of this class.

9. Features of Life Cycle Builder (LCB)

This module provides core services for Project Engineer. It is the foundation for all project Engineer activities and provides underlying services for other modules (OPRR database engine and Traffic Controller). The purpose of LCB is to allow users to load and manipulate the project life cycle and templates. The following are some of the features of LCB:

- Load a project database or a Life Cycle template
- Save/Save-As projects and templates
- Display various views of the Life Cycle
- Print the Life Cycle plan at various levels of detail
- Insert, Delete, Modify, Copy, and Paste Activities
- Promote and Demote Activities
- Collapse and Expansion Activities
- Expand and Collapse Activities
- Explore an Activity to a detailed description
- Hide/Unhide 'deleted' tasks
- Export to other scheduler packages
- Customize the appearance of a Life Cycle.
- Provide on-line Method (hypermedia based help) integration and synchronization [1]
10. The LCB List Window

This child window displays the Life Cycle in list form in a scrollable window. Entries (Work Breakdown Structure objects) in the window can be assigned different fonts and sizes based on styles, where styles are assigned by level in the Work Breakdown Structure. The list window also provides column configurability. Allowing choices of column formats (i.e., different combinations of columns customized for a particular purpose). Further, the list window supports character attributes determined by the activity level. Each activity level (Stage, Step, Task, etc.) can have its own font, style, and size. The List Window also supports an outline format (with activities indented for appropriate levels) and a straight-list format. Users can highlight rows/lines, columns, cells or rectangular areas, and perform editing of the selected items. Column widths can also be resized dynamically, and the list window will automatically clip text outside a boundary.

11. The LCB Object Editor

The purpose of this MDI child window is to display all the information for one Activity, and allow the user to edit any of the information fields. After a modification, the user must either accept or cancel the changes made. This requirement ensures a synchronization among all MDI child windows, i.e., the modified object will be updated accordingly in the List Window.

The object editor 'pops' up when the user 'double-clicks' the mouse button on any selected Activity in the List window of the Life Cycle Builder module. This editor also validates the modified data for any error in formatting. For example, if the user enters an alphabetical string for a date field, an Error Message Box is displayed requesting the user to input the data in the proper format.

The object editor displays the data associated with the selected Activity in the List window and the user can modify any field. The same object editor 'pops' up when the user tries to insert a 'new' activity in the List window; in his case the fields are empty except for the default Activity type. Each insertion is checked/validated to make sure that the Activity level hierarchy is maintained.

12. The LCB Work Breakdown Structure (WBS) Diagram Window

The WBS Diagram Window is another MDI child window, and can be activated from the tools palette. This provides a graphic display of the WBS structure. Each Activity is displayed as an icon with some text filled within to indicate the Activity code. At this stage, the users are not able to edit the structure from this window, but they are able to navigate the structure by clicking and double-clicking icons to display the Object Editor.

All operations within the diagram (highlighting, collapse/expand, etc.) synchronize with the list mode window (if visible) and the Object Editor (if visible). The application may require other MDI child windows to be added for clarity and better user interface.

13. Summary

A tool is provided as a possible solution to the problem existing in the development and maintenance of project development life cycles. An object-oriented approach has been followed in the design of this tool. The user-interface component of this tool is based on Microsoft Windows 3.0 and is inspired by the IBM SAA/CUA Advanced Interface Design Guide. To provide multiple relationships between objects/tasks, an Object, Property, Role, and Relationship model database has been used as the database engine. Another reason for using this engine is that the same engine is also used in some other tools such as the CASE tool and the repository, thus facilitating internal connectivity and interoperability.

References