A Case Study in Software Project Estimation

Michelle A. Willcox, Devon M. Simmonds, Thomas D. Lovette, Yuli Bonner
University of North Carolina Wilmington
601 South College Road, Wilmington, NC 28403
{ maw3067, simmondsd, tdl6020, myb7721 }@uncw.edu

Abstract

Software estimation is a pivotal activity in the software engineering lifecycle. Indeed, software project planning, including scheduling and resource management are all predicated on the computation of realistic estimates. This paper reports on a case study to evaluate the use of estimation in the model-based development of clipboard management software called ClipBits. Multiple techniques are available for performing software estimation. In this project, three estimation techniques were used: function point estimation, the COCOMO II model and a lines-of-code approach. Our analysis of the results revealed that estimation produced reasonable results. We provide some foresight as to larger variability from actual data for some estimates. In addition, the duration and scheduling predictions proved useful as an aid to help keep the team true to its deadlines. Overall, the results support the fact that software estimation can greatly improve the successful completion of software engineering projects.

Keywords: software engineering, software estimation, planning, scheduling, clipboard.

1. Introduction

Software engineering [1] is a task that requires a lot of preparation and planning. With proper planning, all stakeholders, including investors, project managers, and programmers can have a solid basis for project execution and greater assurance that the project will succeed. While it is hard to accurately predict the software duration, schedule and cost before starting a software project, proper early software project estimation [2, 3] provides goals and milestones for the development team, and enables the allocation and management of human and other resources in a systematic and predictable manner. Estimation is beneficial because it is essential that room be allowed in the development process, for the many unforeseen circumstances that typically accompany complex projects.

Some of the areas involved in software project planning include the schedule the software development team will follow; the project duration, the project size, required human resources, specialized skills and an assessment of risks. Schedule estimation includes such tasks as determining when a requirement of the software should be completed, and how long it is expected to take. Several techniques are available for software sizing including techniques based on the projected lines of code, techniques based on the functionality of the software, and empirical models [3]. Significant project anomalies are typically forecasted and addressed through systematic and continuous risk assessment. While some of the risks might never occur, it is good to plan for them regardless of the outcome. Lastly, the software development team needs to set forth an understanding of what each member's tasks are going to be throughout the project. This may be done using a tool such as a responsibility matrix. All of these areas need to be accounted for during the software project planning. Needless to say, that while software project planning begins at project inception, project planning is an ongoing activity throughout the life of the project.

This paper reports on a case study designed to assess the importance of effective software project planning for successful software development. By analyzing the estimates with actual data, we will be able to see how accurate our estimates were in the end, and what can be done to improve this process. The software that was developed in this case study improved upon the functionality of the system clipboard [4 – 8, 14 – 16] to allow users to increase their work efficiency and productivity [9]. The application, called ClipBits, enables users to store past clipboard data as ClipButtons, and save multiple ClipButtons in ButtonSets locally to be used at any time. This is all done using an unobtrusive user interface that is intuitive and easy to use. Our main

focus was on determining the facets of software project planning that were most helpful for developers, what areas were or were not estimated accurately, and the quality [10, 11] of the resulting application.

The rest of the paper is organized as follows: project planning and estimation is presented in Section 2, software design in Section 3, the results and lessons learned in Section 4, and lastly, Section 5 will present the discussion and conclusion.

2. Project Planning and Estimation

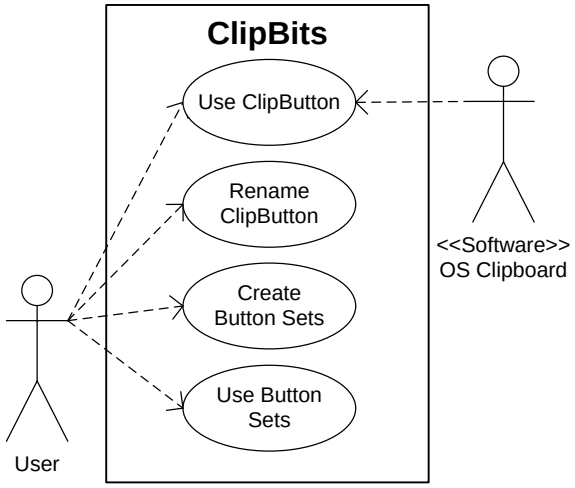

Figure 1. ClipBits Use Case Diagram

The above Use Case Diagram (Figure 1) demonstrates typical actor interaction with the ClipBits software. Actors include the end user and the operating system clipboard, both of which interact with the application. However, the methods through which they interact are slightly different. The end user interacts with ClipBits through the use of clip buttons and UI components, whereas the OS clipboard interacts with the application via Windows messaging. Figure 1 is a context model for the application, and therefore understanding this context model during the software planning phase will allow each of the developers to relate their specific development task to the actor(s) for whom the feature is required.

The many vital roles necessary to the completion of the ClipBits software were assigned as illustrated by the responsibility matrix shown in Table 1. The table shows the lead software engineer for each task. The tasks were well-distributed according to the various areas of expertise present within the group for the purposes of increasing efficiency while mitigating the risk associated with learning curves (see Table 3). Throughout the development process, each member will be able to refer back to this table if there is every any task in question regarding who is responsible.

	Thomas Lovette	Yuli Bonner	Michelle Willcox
Project Management	x		
Requirements Engineering		x	
Software Design	x	x	x
Quality Assurance			x
Data Storage/Retrieval Mechanism	x		
ButtonSet Design/Implementation	x		
Clipboard Management		x	
ClipButton Development		x	
UI Design/Development			x
Cross-Tier Integration	x	x	x
Product Testing			x
Project Proposal	x		
Requirements Engineering Document		x	
Software Design Document	x		
Implementation/ Testing Document			x

Table 1. ClipBits Responsibility Matrix

The overall estimation process for development of the ClipBits application began with the development of the work items in the work breakdown structure shown in Figure 2. Broad task categories include Planning and Research, Modeling and Design, Construction, and Deployment. Each category contains multiple sub-tasks, each with unique requirements and deadlines. However, the work items have been structured in such a way that the main categories can fall one after another, with Planning and Research directly preceding Modeling and Design, etc. Some tasks- such as research and integration- are carried out irrespective of hierarchical patterns, as their success impacts every area of the development process. It is evident that much is to be determined in the Planning and Research phases because these activities greatly influence the Modeling and Design, Construction, and Deployment phases.

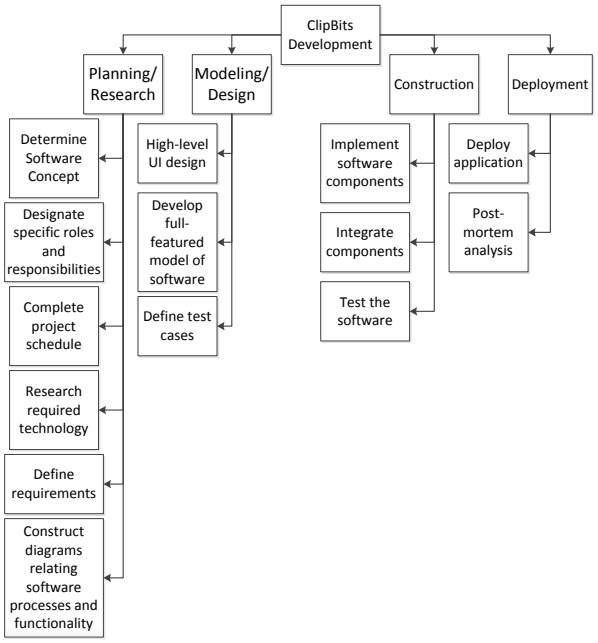

Figure 2. ClipBits Work Breakdown Structure (WBS)

Lines of Code Estimation

Table 1. ClipBits Cost Estimation using Lines of Code Model

Function	Optimistic LOC Estimate	Average LOC Estimate	Pessimistic LOC Estimate	Estimated LOC
User Interface	1,550	3,000	4,500	3,000
Clipboard Monitor	100	500	1,000	516
Clip Buttons	200	800	2,000	900
Data Read/ Write	250	600	1,500	691
Button Set	100	300	500	300
				5,407

Table 1 shows the basic data used to compute the software estimates using the lines of code (LOC) approach. The computation is done by finding the mean of the optimistic, likely, and pessimistic LOC estimates following a beta distribution [12], for the five broad categories of the application. Each item in The "Estimated LOC" column is calculated using the formula:

$$Estimated\ LOC = \frac{Optimistic + (4 \times Average) + Pessimistic}{6}$$

The total estimated Lines of Code for the application is then simply the sum of the entries in the "Estimated LOC" column. If an organizational average productivity rate of 620 LOC/person-month is assumed along with a burdened labor rate of $8,000.00/month [**pressman]**, then the average cost per line of code is approximately $12.90. Given this information, the total estimated cost of building this application is:

$$\$12.90/LOC \times 5{,}407\ LOC = \mathbf{\$69{,}750.30}$$

Likewise, the estimated effort required to build this application is:

$$\$69{,}750.30 \div \$8{,}000\ per\ mmonth = \mathbf{8.72\ person\ months}$$

These lines of code estimations allow the developers to visualize how much work each function will require, and can be used to further predict how long the implementation process will take.

Function Points Estimation

The Function Points estimate [13] for an application can be calculated using data as shown in Table 2. The computation consists of two components: an information domain value and a value adjustment factor. The information domain value (IDV) is computed in three steps. In the first step an estimate for five information domain factors such as "External Input" is computed. This is typically done following a beta distribution scheme as was outlined above, by computing the mean of optimistic, average and pessimistic values. In the second step the mean is multiplied by a weighting factor to get the total estimate for that information domain element. For example, in Table 2 the mean for "External Input", 10, is multiplied by the weighting factor, 4, to get a total of 40. Weighting is typically done based on the estimated complexity of the software as shown in Table 3 [17]. In the third step, a value adjustment factor (VAF) is computed by estimating the importance of fourteen factors to the overall project. These factors include such element as "backup and recovery", "distributed processing" and "code designed for reuse". In step 4, the number of function points for the project is computed using the formula:

$$Function\ Points = IDV * [0.65 + (.01 \times VAF)] = 102 \times [0.65 + (.01 \times 46)] = \mathbf{113.22}$$

Table 2. ClipBits Function Points IDV Computation

Function	Count	Weighting Factor	
External Input	10	4	40
External Output	5	5	25
External Inquiries	0	4	0
Internal Logical Files	3	10	30
External Interface Files	1	7	7
Count total			**102**

Table 3. Weighting Factors in IDV Computation

	Weighting Factor		
	Simple	**Average**	**Complex**
External Input	3	4	6
External Output	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10

The function points analysis has proven to be a reliable estimation method that helps express the software functionality more from the user's prospective than from the developers' perspective.

PERT Chart

The PERT Chart (Figure 3) demonstrates directed, parallel paths from the start of the project to deployment. Many of the preliminary items were simpler, whereas many of the later tasks are predicted to require several weeks. However, most tasks can occur concurrently, implying that, whereas there are many tasks to complete, the overall amount of time spent is well-distributed among various branches, and no one branch dominates the others in terms of effort and manpower. The time (in days) estimates are based upon how complex the developers predicted each task to be in the development phase.

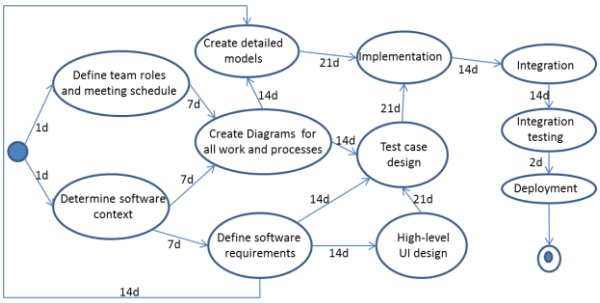

Figure 3. PERT Chart

Risk Plan

Table 3. ClipBits Risk Assessment and Mitigation Plan

Risks	Cat	Prob	Imp	RMMM
Unable to meet established deadlines	BU	40%	1	Set up meetings multiple times per week in order to monitor progress. Allow team member input in order to make realistic deadline decisions.
Compartmentalization overlaps and slows momentum down	PS	20%	2	Equally divide tasks among team members and increase the amount of communication.
Slower performance with larger data	TE	60%	3	Create a file size limit in order to prevent users from slowing down the software's execution time.
Performance impact of using a database versus serialization	TE	50%	3	Research each method in depth in order to decide which method is more efficient.
Standard library is not sufficient for our needs	TE	75%	2	Examine other libraries that have the functionality that will benefit our requirements.
Staff is unable to learn necessary skills/ technology	ST	30%	1	Assist team members in the learning process by pointing them to useful resources that will allow them to acquire the necessary skillset.

Key (Cat=Category, Imp=impact, Prob=probability)

Category	Impact
BU- Business Risk	1- Catastrophic
PS- Project Risk	2- Critical
TE- Technical Risk	3- Marginal
ST- Staff Risk	4- Negligible

There are numerous possible risks associated with the development of ClipBits. The table above (Table 3) lists several categories of risks, along with the development team's planned methods of mitigation. Through wise delegation of responsibility, proper research methods, and effective advance prototyping, many of the listed risks were effectively eliminated.

3. Software Design & Implementation

Figure 4 shows the state diagram for the ClipBits software. The application responds to the system clipboard events by creating ClipButtons and storing them in a stack. Of note are the starting and ending states: the application starts with an initialization of the ClipButton Stack and ends with its disposal. Since the application is largely UI-driven, it is to be expected that many of the states consist of windows opening and closing, with processing occurring within.

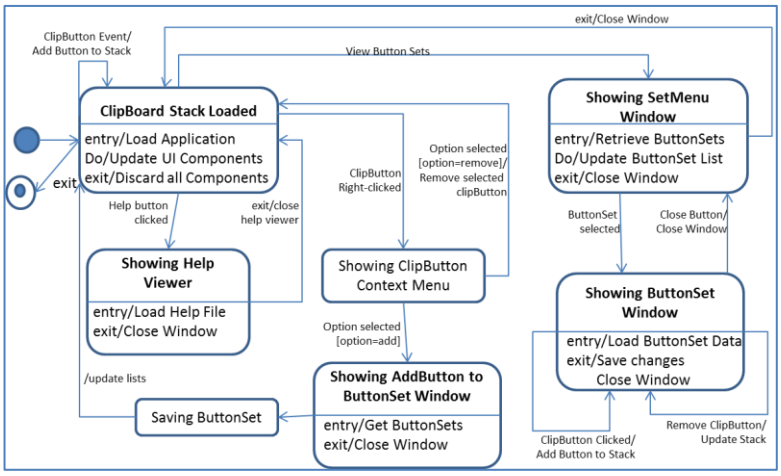

Figure 4. ClipBits State Diagram

A screen shot of the resulting UI for the ClipBits software is shown in Figure 5. The figure shows the Main Window, the Add-All-Buttons-t-ButtonSet Window, the Button Set Menu Window, and finally the individual Button Set window displaying the ClipButtons contained in the Button Set. This design went through multiple alterations in order to satisfy the requirements that were set forth in the planning phase.

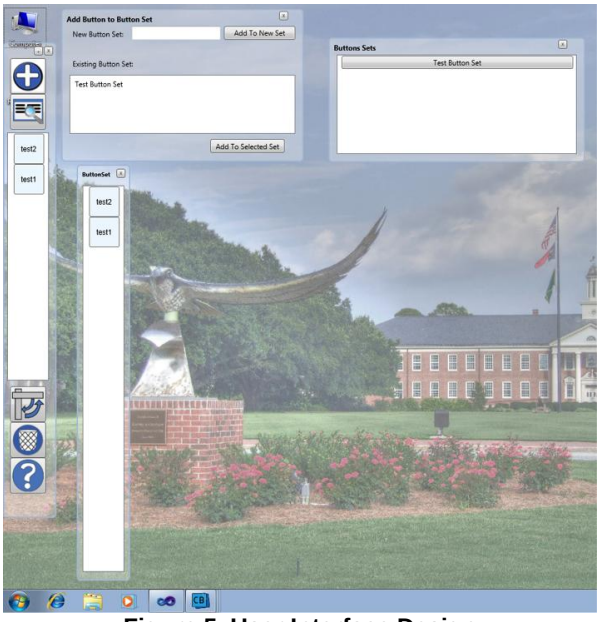

Figure 5. User Interface Design

4. Results and Lessons Learned

The original cost and effort estimates for ClipBits were misjudged. Original estimates called for 5,407 lines of code, but the final product contained far less: approximately 2,000. This is due largely to uncertainty about the extent to which Windows Presentation Foundation (WPF) would be used in development. With WPF's declarative syntax, it ended up being ideal for data-binding scenarios. Thus, operations that would have taken many lines of code to implement (such as UI updates) were relegated to the framework, drastically reducing the amount of necessary coding. In the future, more research on WPF's syntax, specifically for the UI, would allow us to predict our lines of code estimation better.

In the requirements phase of our work, we established a list of non-functional requirements that we strived to meet:

- Users have easy access to *Button Sets* in the current session and in later sessions because a Button Sets folder is stored locally on the user's hard drive.
- UI is user friendly and unobtrusive.
- UI docks to both the top and left sides of the screen.

- User documentation such as a manual or other help feature should are included when the user clicks the help button.
- The interface includes a significant level of customization options in terms of size, shape, and/or orientation.
- The program is lightweight in terms of performance so that it does not impede the performance of any other applications with which it is being used.

Overall, we drafted these requirements with the user's experience in mind, we wanted to minimize the learning curve for ClipBits so that anyone could easily use it for their benefit. The serialization of Button Sets is a key feature that enables users to refer back to data that they created or used in the past. This was made possible by serializing the ButtonSet data into a file that was automatically stored in the user's Documents folder. As such, any future time when a user browses their Button Sets, the file is read and the Button Set data is accessible and may be copied right back to the system clipboard.

The realization of non-functional requirements was a significant element of the user interface design. By creating simple icons for each button, and making the windows semi-transparent we were able to make the application easy to use and unobtrusive. In addition, the main window is easily positioned by the click of a button to either the top of the screen horizontally or to the left side vertically. Therefore, the windows would stay out of the way from other applications. The user also has the choice to hide the windows all together, and move it manually to another position on the screen.

Not only does the application not visually other applications, but performance-wise, the application doesn't slow down the computer's other processes because it is a lightweight design. The only thing the application is constantly monitoring is the system clipboard, and when it recognizes that something has been copied, it generates Clip Button data and displays it in the window.

Lastly, if the user is ever confused or in need of help while using the application, then they can easily access the user manual from the help button on the main window. The user manual features a description of each UI element and explains how each element is utilized.

In the end, our group was able to stay very close to our estimated schedule and designated tasks, even though we ran into multiple small issues. What really helped us stay on that schedule was our meetings twice a week where we would go over what we had been working on during our own time, reviewing what deadlines were coming up, discussing what needed to be done, and designating tasks. We also made it a point to communicate constantly via email or at meetings so that even if we were not designated a specific a task, we were involved to some degree that would allow us to understand where the project stood in relation to our overall schedule.

5. Discussion and Conclusion

Now that the project has come to completion, it is evident that an emphasis on comprehensive software estimation techniques can only be a benefit to the software developers throughout the whole project. Even if there are some inaccurate predictions, they will still assist in the goals and visualization of the software's end product. In addition, estimation is a learnt discipline predicated on the availability of historical data. As such this project has taught us not only valuable lessons, but given us some basic data to assist in the next estimation project. A lot of software estimation and planning diagrams and tables were used; and our team was able to supplement that data with frequently collaborations and reacting in an agile manner to issues as they arose. Without that, then our team would not be able to help each other out when these problems surfaced, and our estimated project schedule would have been thrown way off. Luckily, with the help of effective software estimation, we were able to keep on track all of our deadlines relatively well and successfully implement all of our non-functional, and functional requirements.

In the future, our team will most likely be involved in developing other projects. As a whole, we all learned a great deal about the significance of software planning and estimation and we will be able to take this knowledge and apply it to any future software projects we may come across.

References

[1] Pressman, Roger. *Software Engineering: A Practitioner's Approach.* New York: McGraw Hill., 2010. Print.

[2] [30] Roger S. Pressman. Estimating Software Projects, Chapter 26 of Software Engineering: A Practitioner's Approach, McGraw-Hill 2010.

[3] Violeta Bozhikova and Mariana Stoeva. 2010. An approach for software cost estimation. In *Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop*

for PhD Students in Computing on International Conference on Computer Systems and Technologies (CompSysTech '10), Boris Rachev and Angel Smrikarov (Eds.). ACM, New York, NY, USA, 119-124. DOI=10.1145/1839379.1839401 http://0-doi.acm.org.uncclc.coast.uncwil.edu/10.1145/1839379.1839401

[4] Frakes, Dan. "Clipboard Managers." *MacWorld* 7 Nov. 2010: 36-37. Print.

[5] Frakes, Dan. "Clipboard Managers." *Macworld* 27.7 (July 2010): 36-37. *Academic Search Premier*. Web.

[6] Stone, David "The Office Clipboard."*PC Magazine* 21, Dec. 2001: 83-86. Print.

[7] Zardetto, Sharon. "Two Quick Copy and Paste Tricks." *Macworld* 27.4 (April 2010): 53. *Academic Search Premier*. Web.

[8] *xNeat.com*. xNeat. 2011. Web. 5 Nov. 2011.

[9] Champy, James. "Productivity Promise." *Financial Executive.* Oct. 2003: 35. Print.

[10] D. Milicic, Software quality models and philosophies, in *Software quality attributes and trade-offs*, eds L. Lundberg, M. Mattson, C. Wohlin (2008), http://www.bth.se/tek/besq.nsf/%28WebFiles%29/CF1C3230DB425EDCC125706900317C44/$FILE/chapter_1.pdf

[11] B. Zeiss et al, Applying the ISO 9126 quality model to test specifications – exemplified for TTCN-3 Test Specifications (Bonn, 2007), Software Engineering Conference 2007, Lecture Notes in Informatics (LNI) 105.

[12] Roger S. Pressman. Software Engineering: A Practitioner's Approach 7th Ed. Page 700, McGraw-Hill 2010.

[13] Vipin Saxena and Manish Shrivastava. 2009. Performance of function point analysis through UML modeling. *SIGSOFT Softw. Eng. Notes* 34, 2 (February 2009).

[14] Li, Shaobo, Lv, Shulin, Jia, Xiaohui, and Shao, Zhisheng. "Application of Clipboard Monitoring Technology in Graphic and Document Information Security Protection System." *Intelligent Information Technology and Security Informatics* (April 2010):423-6. *IEEE*. Web.

[15] Pomeroy, Bryony and Wiseman, Simon. "Private Desktops and Shared Store." *Computer Security Applications Conference, Annual* (December 1998): 190-200. *IEEE*. Web.

[16] Birss, Edward W.. "The Integrated Software and User Interface of Apple's Lisa." *National Computer Conference* (1984):319-28. *IEEE*. Web.

[17] Roger S. Pressman. Figure 23.1: Computing Function Points *in* Software Engineering: A Practitioner's Approach, McGraw-Hill 2010, page 621.