Accounting for Extreme Developers and the Software Mob

Charles Becker, (E-mail: cbecker@cs.nmhu.edu), New Mexico Highlands University[1]
David West, (E-mail: dwest@cs.nmhu.edu), New Mexico Highlands University

Abstract

Whenever a "new and improved" means for software development is adopted, managers are confronted with a new set of evaluation problems. Among these: the determination of costs and benefits - including intangibles. Traditional software development methods emphasized up-front planning and design along with some measures (however unreliable they might have been) that a manager could use to estimate both the costs and benefits of any proposed new system. Two new approaches to software development -Extreme Programming and Mob Software (the latter also known as Open Source Software) - are being adopted very rapidly by a wide range of businesses and software development providers. Both of these approaches radically alter the basic premises of how software is (should be) done and, accordingly, invalidate most, if not all, of the ways managers estimate feasibility and monitor progress. Finding new measures and metrics that can be used by software managers and developers would seem to fit the mission of managerial accounting professionals. This paper will briefly review how costs and benefits were calculated for traditionally managed software projects, summarize two new approaches to software development in order to highlight the problems the pose for managerial accounting, and speculate on some possible approaches to solving those problems.

Introduction

Software development projects are notoriously difficult to manage. Cost overruns - both in terms of dollars and time - are endemic. Benefits (at least in terms of productivity gains) have been non-existent or highly speculative. Intangible benefits are difficult to demonstrate while intangible costs - user annoyance and dehumanization of work, for example - are clearly evident but equally difficult to measure and convert into balance sheet entries.

"Structured Development" and "Software Engineering" are labels for formally defined methods that were intended to make software development projects more tractable and more manageable. Both were responses to the declaration - first made in 1967 - of a "software crisis" (one that appears to continue to this day). Both methods emphasize up-front planning and feasibility determination in order to provide a foundation for project management and accounting.

Critical aspects of the planning and feasibility phase of these classical methods include:

- Categorization of the development workforce - typically into three categories based on skill level (novice, junior, and senior).
- Decomposition of the project into discrete modules with each module assigned a level of complexity.
- Construction of a development timeline (PERT/CPM and Gantt charts, usually) with time being a function of the level of developer assigned and an estimate of how long that category of developer would require to complete a module of specific complexity.
- Assignment of tangible costs - salaries (derived from timeline), equipment and software acquisitions, training costs being the most commonly identified.
- Estimation of tangible benefits - salary savings from displaced employees, reduced inventory costs, and similar savings.

- At least a cursory exploration and estimation of intangible costs and benefits - an idea always noted but seldom developed to any extent.

Developing of a management and accounting model for a project was relatively straightforward given this information. A manager would have a model that showed how much work should be accomplished at each point in time and take action if a variance was detected. Accountants would be able to monitor actual and projected expenditures and correlate those with the same timeline and alter projections when actual figures deviated from planned figures.

[[[An example of a typical accounting model for software development - if any can be found, OR a summary of guidelines for constructing such a model - if any can be found, would be inserted here.]]]

Unfortunately, both these methods were totally reliant on having a complete and accurate specification of the new system available at the very beginning of the process. Equally problematic were the means for determining module complexity, developer skill level, and the derived person-months required to complete each task. (Fred Brooks[2] exposed these problems almost as soon as these classical methods became widely adopted.) Finally, the granularity of accounting and control was at the project level[3] - masking the true sources of project variance and cost which occur at module and task levels.

Classical methods, despite any shortcomings, at least attempted to provide a managerial accounting model - or the foundation for constructing such a model - that development managers and accounting departments could use to evaluate both projects and progress towards achievement of project goals. Contemporary approaches to software development are radically different from classical approaches and may not provide the kind of data necessary for classical managerial accounting models. (Conversely, they may provide a basis for even better accounting models by providing a finer-grained level of control and accounting.) The absence of well-defined (at least potentially) accounting and project management models creates a severe tension for managers deciding whether or not to adopt these new approaches. The needs of potential adopters, and advocates, of new development approaches would be well served if some kind of managerial accounting model could be proposed.

Extreme Programming

Extreme Programming[4] (XP) is the best-known example of a category of software development approaches labeled, "agile methods." These approaches are motivated by observation of how software development is actually done - in contrast to preconceived notions of how it should be done. In form, but not necessarily in substance, these approaches resemble some of the rapid-prototyping methods advocated (but seldom adopted) in the seventies and eighties.

It is necessary to summarize some key points that define XP (and by extension all the other agile methods) in order to identify how accounting and managerial models might be constructed.

Kent Beck[5] notes that there are four variables that shape software development: Cost, Time, Quality, and Scope. These variables are opposed. For example, if you want higher quality you must allow for greater time and less scope. If you want to minimize time and cost, be prepared to sacrifice quality and scope. Beck's "extreme" solution for dealing with the four variables - let management select three of the variables to control and let the development team control the fourth.

Let's assume that management is less interested in 'control,' in the sense of specifying and setting hard limits, and more interested in understanding and monitoring so that they can make intelligent decisions. The four variables will then be addressed as follows:

- Quality - will be consistent with our organizational mission and standards (i.e. as high as possible).
- Scope - will be as broad as possible - ultimately we are building a system that supports all aspects of our organization and our business. It is a mistake to think that we can treat portions of our

information system in isolation any more than we can segment and isolate aspects of our organization.

- Time - it will take as much time as required but no more - but, for each unit of development, I must have accurate information about the allocation and expenditure of time. (There is a further caveat here - modules must be defined in such a fashion that modules can be completed in a "timeframe of need" which is defined by the business aspect that the module is intended to support.)
- Cost - it will cost as much as it costs but no more - but, for each unit of development, I must have accurate information about the allocation and expenditure of dollars.

At first glance, this treatment of the four variables seems to diminish the possibility of creating solid accounting and managerial models. However, a brief discussion of the ideas implicit in our treatment of scope and an equally brief review of the practice of XP will expose a basis for the kind of models we want.

Information systems have seldom been treated as systems. Instead they are, at best, a semi-integrated amalgamation of special purpose software artifacts. Even less frequently have the computer-based applications been integrated effectively with human and business systems. All to often the two are at odds, especially when it is noted that business requirements evolve and change rapidly while the software artifacts that ostensibly support them require much longer periods of time to alter and redevelop.

To resolve these difficulties we initially note that there is only one information system and that all software development efforts are merely aspects of that unitary project so each unit of development has the same scope as any other - to support the business in its entirety. This means that whatever products accrue from a unit of development must be compatible and usable in every facet of the business where they might be appropriate. (This statement is simply a restatement of the definition of an object and object-oriented development ideas, especially as developed by David Taylor.[6])

But we want to account for projects at a finer-grained level than traditional development projects, not a larger one. Here, the practices of XP provide the key. XP advocates units of development (stories in the XP vernacular) that can be accomplished in one week (or less). A finished unit of work is capable of production use - it can actually support a business task by Friday afternoon.

Of course, most projects involve the development of multiple units. So it is necessary to have a means of aggregating costs and assessing value as the project proceeds. But it is not necessary to have the kind of monolithic specifications mandated by traditional approaches to software development. This is quite desirable given that those specifications were, at best gross estimates and, at worst, complete works of fiction.

It is desirable (actually it is necessary[7]) to develop an overall object model of your enterprise - again following the ideas of David Taylor - so that you have some notion of the totality of objects that might eventually be necessary and have a mapping of objects-to-business-task. This global model is based on a "responsibility-driven" view of objects - and makes no distinction between software objects, people, machinery, or other kinds of non-software objects. The responsibility-driven part of the model can be stated simply, "here are the things we, as a business, need to do and here is who is responsible for doing them." The 'who' may be a person, a software module or any other object.

We are now ready to redefine what is incorporated in the scope variable discussed above. The 'scope' of a unit of development is the "story." A story is nothing more than the scripted interaction of a small group of objects that collectively must fulfill a specified set of responsibilities. The scope of any individual object, however, encompasses all the things it must do in every context in which it is encountered. The contexts in which an object participates are defined by the global responsibility-driven object model.

The practices of XP ensure that each unit of development incorporates all of the cost incurring work necessary to complete that unit. All planning, design, testing, coding, user evaluation, refactoring[8], and integration tasks for the entire story are completed within the bounds of the story-defined unit of development.

The accounting model for each unit of development (be it called a story or an object) is fairly straightforward: one week of salaries for the development team plus some proportional (very small) amount of cumulative overhead (space, computer and networks systems, development tools) distributed among all of the objects that were created or changed in the process of realizing the story. Each object can accumulate a record of the development costs incurred over its lifetime as a collection of cost entries that include a date, a story id, and its share of the distributed costs incurred while developing that story.

Because the global responsibility-driven model (discussed previously) matches tasks to objects - to business goals - the cost of each responsibility can be obtained using traditional accounting methods. When a development project results in a software object (or collection of objects) assuming responsibility for a task previously done by a non-software object the differences in cost can be directly observed and measured. If the accumulating savings exceed the total expenditures required to create the new object then the project was worthwhile.

Not only can this kind of comparison be done at a fine-grained level[9], it can be done immediately and with a fair degree of accuracy. Within a weeks time a manager can have reasonably accurate information to decide all critical questions about specific units of development. This dramatically reduces management risk.

In the next few paragraphs we will propose an accounting model consistent with XP practices as outlined above before turning our attention to the accounting problems presented by Mob Software (open source).

[[[over to you Chuck]]]

Mob Software

The term 'mob software' was coined by Richard Gabriel in his invited address to OOPSLA 2000 in Minneapolis, Minnesota. It is a colorful way of describing the world of open source software. Open source is a radical revision of the economic model behind software development and, because of this, presents some interesting accounting challenges for organizations electing to participate in open source development.

Traditionally, software artifacts are considered proprietary intellectual property. A company invests money in the creation of the software and some measure of value is assigned to the resulting artifact. Even if the software artifact is intended solely for internal use, it still has a balance sheet value. If the software artifact is intended for market, it has a market value just like any other product. Traditional accounting accommodates this model very well - excepting perhaps, the interesting notion that marginal cost for software production is no longer a salient issue.

Linux is the best-known example of open source software. It is possible to find almost any kind of application software, operating system, development tool, game, etc. as an open source product. The most obvious differentiating feature of open source software is the fact that it is free. You cannot license or charge for any open source product[10], nor can anyone charge you for its use. Open source software can be modified, extended, and improved by anyone since everyone receiving the software also receives the source code (hence the term open source). Anyone modifying the original is restricted from selling their modifications. All modifications must are subject to the same open source license and must be distributed free. (Or not distributed at all - used exclusively by the modifying individual or organization.)

Open source software, if it catches the imagination of software developers, attracts hundreds if not thousands of contributors (the software mob). Each contributor adds some feature to the software, detects

and corrects some flaw, or enables the software in some new environment (called 'porting' the software to a different hardware/operating system platform). This can happen with great rapidity. When Squeak (the latest incarnation of the Smalltalk programming language) was released as open source it tool less than a month to have versions running on almost any conceivable platform - from mainframes to handheld digital assistants. No single development team could provide this kind of rapid development - it takes a mob, working in parallel.

Management of open source projects is almost an oxymoron. By definition, open source is provided to the world and any individuals in that world that want to contribute do so, on their own time and without compensation (except reputation enhancement). Most open source projects do, however, maintain some kind of "gate-keeping" control. Linux, for example, has a committee that reviews all changes and enhancements to the Linux kernel (the most basic part of the operating system) and decides which contributions to include in any official release of that kernel.

Organizations contemplating the use of open source models for their software development and organizations adopting open source products for internal use face some interesting accounting questions. For example:

- If I purchase a proprietary software product it is usually treated as a depreciable asset. But an open source product is not an asset because there is no ownership. Whatever I do buy when acquiring an open source product is, by definition, a service (akin to consulting or training) or an expense (like shipping costs).
- If an open source product is modified to better suit the purposes of our organization, how are the costs associated with the modification effort accounted for? We do not 'own' the modifications any more than we 'own' the original software.
- If, in the process of modifying the software for our own use we invent a unique and uniquely valuable new algorithm - one that, in other circumstances, we could patent and license and use to generate revenue. Under the terms of open source we cannot do that, we must either be the sole users of that innovation or we must give it away under the open source license. Can I take a business loss in those circumstances?
- If I need a modification, extension or bug fix in an open source product that I am using - how can I account the comparative costs and benefits of waiting for another member of the 'mob' to provide the solution and having my own staff provide the work?
- If we adopt the open source model for internal development (something that is implicit in the XP notion of collective ownership of code) and a computer literate accountant - not part of the software development organization, per se - provides some unit of development how do I account for that contribution?

Some possible approaches for dealing with this type of question [[[back to you again]]]

Conclusions

Clearly the adoption of any new method for software development raises questions and poses problems for managers and accountants supplying information for management decision making. When the adoption involves a method as radically different from standard practice as XP and Mob Software the questions and problems can become significant challenges.

Too often managers have been promised huge returns for investing in new technologies and new development approaches only to be left holding the, empty, bag. It is difficult if not impossible to discern which "new thing" is a real foundation for change and which is merely the "fad *du jour*." One reason it is difficult is that few innovations provide solid accounting models - or bases for constructing such models - that would allow managers to test and evaluate from real data.

In this paper we have discussed some examples of the kinds of problems raised and proposed some ways to address those problems. Of the two approaches examined, XP and Mob Software, XP seems to provide the more solid approach - primarily because we can see how we might construct a model that would allow us to obtain real data and measure real results, right from the beginning of the first project using the XP approach.

Advocates of new technologies and especially somewhat intangible technologies like a new development method should, as a matter of course, develop robust accounting models that will allow managers to evaluate and make appropriate decisions about adopting that new technology. It should be noted that these models are quite different from the, supposedly empirical but very difficult to verify, claims of productivity gains and payback benefits that advocates do supply. The authors of this paper will be exploring the development of precisely this kind of robust accounting model as future work.

Notes and References

---

[1] Charles Becker is a Ph.D. student at Nova Southeastern University in Florida as well as an Associate Professor of Accounting at Highlands. This paper fulfills one of the requirements of his doctoral program.

[2] See the 1995 collection of papers and articles published in earlier years collected in, Brooks, Fred. *The Mythical Man Month*. Addison-Wesley. 1995. ISBN 0201835959.

[3] Theoretically it is possible to track the allocation of developer time, computer usage, etc. to individual software modules or development tasks. Possible, but extremely impractical. Also some costs - meeting time to discuss a variety of related modules, computer time for integration testing, software costs when the software is used throughout the project, etc. - cannot be allocated, only estimated. Hence, in practice, accounting is done only at the project level.

[4] Formulated by Kent Beck and Ward Cunningham and popularized by them and Ron Jeffries, Extreme Programming is detailed in a series of books published by Addison-Wesley, beginning with: Beck, Kent. Extreme Programming Explained: embrace change. Addison-Wesley. 2000. ISBN 201616416.

[5] Ibid, page 15.

[6] Taylor, David A. *Business Engineering with Object Technology*. John Wiley and Sons, 1995.

[7] Global models of an enterprise have always been necessary for truly effective management of information system development. Unlike the global models proposed in the past (e.g. Repositories) the object-based global model envisioned here (and detailed by David Taylor) is quite simple to build and maintain.

[8] Refactoring is the process of analyzing an object and the code that implements that object in order to generalize or simplify that code. By generalization it is generally meant - making it usable in a greater number of contexts. Refactoring may result in breaking a complicated object into two or more simpler parts with the assembly collectively assuming the responsibilities of the original object.

[9] Accounting can be done at the object level - but it should be remembered that details can be aggregated and reported, just as they are with any other accounting model. Managers do not have to look at line item details - but, with an object and XP approach they can if necessary. This is something that was not possible with earlier approaches to software development.

[10] People can charge for the costs of distribution and support (e.g. burning a CD, writing a user manual, providing consulting services).