WHITE PAPER • NOVEMBER 2017

ca technologies

Understanding Software Capitalization: A Multi-Perspective Approach for a Modern PMO

A commonsense, risk-aware approach that works with any software development method for project-oriented work.

Donna Fitzgerald

Four Perspectives

Some topics never seem to disappear completely. It's been almost two decades since the Accounting Standards Executive Committee (ASec) released SOP 98.1, and yet if you put three PMO leaders in a room and ask them how their company approaches determining the how, when and what to capitalize about internally developed software, you'll probably get three different answers.

Before we go into detail, we will begin by asking the question: Why do companies capitalize internally developed software expenses in the first place? The obvious answer is because they have to be based on accounting standards, but the business driver and underlying reason is to more closely match expenses with revenue or internal operational value at the time the revenue or value is recognized. Furthermore, if a company has competent software development practices in place, **there should be no difference between the capitalized value of identical systems developed by either waterfall or agile methods.**

Why, then, do we hear comments that agile changes how and what we can capitalize? The polite answer is well-meaning enthusiasm. The more direct answer is that software capitalization has become an unnecessary partisan battle, and even a modern PMO can have trouble knowing whom to believe and how to respond when confronted with multiple perspectives. To help cut through the confusion, we'll cover four of the most common perspectives in this white paper.

The Financial Perspective

Because software capitalization is by its very nature a financial activity, we'll begin by reviewing the financial perspective. Why is this issue coming up now? The answer is that in 1998, ASec did not want to get down into the weeds and dictate in detail how companies needed to track the work they intended to capitalize, so they defined what they expected would be a clear but high-level framework for capitalization.

TABLE 1.
Framework for capitalization

Preliminary Project Stage	Application Development Stage	Post-Implementation /Operational Stage
Conceptual formulation of alternatives	Design of chosen path, including software configuration and software interfaces	Training
Evaluation of alternatives	Coding	Application maintenance
Determination of existence of needed technology	Installation of hardware	
Final selection of alternatives	Testing, including parallel processing phase	

As shown in Table 1, everything before the design and development of the software and everything after the software has gone live is intended to be an expense. How the software is developed is irrelevant. All that matters is there is a way to tell when the work on the "software product" enters the state of being eligible for capitalization.

Consider a classic make or buy software decision. A firm can buy commercial software (which is capitalized) or choose to write it internally with the expectation that it would be able to capitalize roughly the same (or less than the purchased software) when the internal project is complete. This expense would then be charged out to the P&L on an annual pro rata basis until the cost was fully amortized (generally three to seven years) or until the software is retired (if retirement occurs earlier).

There's an old joke that the difference between an accountant and a finance person becomes obvious when you ask each of them, "What's two plus two?" An accountant will always answer four. A finance person will answer, "What do you want it to be?" An accounting standard can always be read literally (as above), or (as any finance person will tell you) it can be fully explored to see how much wiggle room generally accepted accounting principles (GAAP) would allow. Historically SOP 98.1 has been interpreted as offering a lot of wiggle room, which has led to a great deal of confusion in how various organizations have accounted for these costs.

Wiggle room or not, there are certain well-understood principles that apply to all accounting rules.

1. The rules of materiality apply. While every organization is free to set its standard with the agreement of its audit firm, the numbers we've seen are generally in the hundreds of thousands (though we know several who have raised the capitalization threshold to a million).
2. Once materiality has been defined, the rules as to when the capitalization period begins must be applied consistently across all situations. This is where the hairsplitting between agile and waterfall methods arises, and we'll cover it in our next section.
3. Treatment of various software development efforts under SOP 98.1 must be transparent (and auditable). The determination of what work qualifies for capitalization must be clear, and proof that it is consistently applied must be freely available.
4. The rules that have been applied to SOP 98.1 must not violate any other accounting rules (e.g., SOX still applies).
5. The simpler the rules used inside the organization about what is capital and what is expense, the better it is (i.e., use the KISS—or keep it simple, stupid—principle).

The Accounting Perspective

We've noticed some unconscious confusion on the part of many PMO leaders around the subject of what is capital and what is an expense. As a simple rule of thumb, some IT software projects can be entirely expense, but no software project under SOP 98.1 is ever entirely capital. All that can be capitalized is the effort to create a working software product once everyone has agreed the design will work.

Many in the agile community have begun to conflate the idea of capitalization with the idea of value. The contention is that by counting the beans differently using agile approaches, more value has been created and capitalized. While we understand all the nuances for this claim (anything on the balance sheet equals value), in our opinion the argument falls apart when looked at from the perspective that software is an intangible asset that is relatively short-lived. Given this fact, our sole objective with SOP 98.1 is to more closely match costs to the period in which these costs were consumed. In a nutshell, the agile community and its methodology call into question how to be consistent and track capitalization, but that does not change the overall impact to the bottom line—just how it's done.

It's important to continue to stress that within the context of GAAP, accounting capitalization is simply a "pay me later" approach. If $1 million is placed on the balance sheet with a five-year life, then in the first year the software is in use, $200,000 will be expensed. Assuming the software remains useful, the same charge will be made every year for the next four years. If another $1 million is spent on another project in year two, the pattern will be repeated until, at year four, the capitalization rate of $1 million a year equals the amortization rate.

A Fortune 500 CFO observed that "today's balance sheet is tomorrow's P&L," which we felt put much of this discussion into perspective. The goal is not to create a short-term positive impact on the P&L because over time it will even out; the goal is to more clearly represent true operating costs.

Up until this point, we have deliberately confined our discussion to GAAP accounting. In the last several years, Wall Street has adopted a new non-GAAP term, EBITDA, which has had an outsized impact on how many people are starting to view the concept of capitalization. EBITDA stands for earnings before interest, tax, depreciation, and amortization and it is a way for a firm to showcase "higher than GAAP" income from operations. Investors often try new ways to compare dissimilar companies, and EBITDA was designed to do just that. The problem is, ideas and concepts have a habit of morphing into something they were never intended to be, and the concept of EBITDA has started to influence thinking around software capitalization.

The most common agile benefit we've heard is that with agile it becomes possible to more clearly define, in real time, what is new or "value add." That means that if the developer is working on something that has been classified as routine maintenance, if you figure out how to turn it into an enhancement, it can be capitalized. From an accounting standpoint, this doesn't work. The developer doesn't have the authority to "spend" capital. There are clearly delineated accounting rules requiring that spending authority be documented. If the violation of signature authority isn't sufficient, anything the developer does on the fly falls below the materiality threshold and should simply be expensed.

We could continue debating the line of reasoning and invoke the Sarbanes–Oxley Act of 2002 (SOX) as well, but bottom line accounting controls are based on clear black-and-white rules. The rules are not intended to be onerous; they are intended to protect the integrity of the P&L.

To Summarize:

1. SOP 98.1 requires the capitalization of internal use software.
2. Under SOP 98.1, the only costs that can be capitalized are the actual software development costs of internally developed software applications. All other project costs are expense.
3. All costs that a project might incur prior to starting the actual development effort are expense, and all costs after the software is complete and in production are expensive. (See Table one.)
4. The useful life of capitalized software should be reassessed periodically, and the non-useful portion should be written off immediately (we'll discuss this later).
5. Aggressive capitalization at best has a short-term advantage and at worst (if the project fails) an immediate negative impact on the P&L.

Advice: Take a commonsense, risk-aware approach. The worst thing that can happen, from a financial performance perspective, is an unplanned write-off of a capitalized expense. Given this, being aggressive on validating that the software can be developed and will be fit for purpose upon implementation is where the modern PMO should be focusing.

The Modern PMO Perspective

Why does the PMO exist? Like any other organization, there are many reasons, but we would contend that the primary reason is to provide fiduciary assurance that the millions (or billions) of dollars that are being spent on projects or programs are being spent responsibly, in order to deliver the value the organization expects to receive.

Because the majority of PMOs in existence today are IT PMOs, this clear mandate has gotten muddied. The PMO does not exist because it oversees capital spending. The fact that some of the IT dollars fund software development (which is capital) is just a color-of-money issue. **The PMO exists because the total spending on non-operational (i.e., projects) is materially significant, and without a project-based organization to oversee this form of spending, there might be insufficient oversight.**

The formation of IT project delivery organizations should have made it easier to develop an expertise in doing capitalized software projects, but like everything else that goes on in IT, things got complicated. Most IT departments call relatively small units of repeatable work "projects." Essentially, if someone has to be assigned to do the work, by today's IT standards some organizations will call it a project (we've seen the definition as low as 60 hours in divisions of Fortune 50 companies).

These small pieces of work have made their way into the portfolio, and too many organizations have begun to conflate any project with capitalized software. This enormous focus on small units of work has made the agile contention that small units should be dynamically capitalized seem less far-fetched than it should be.

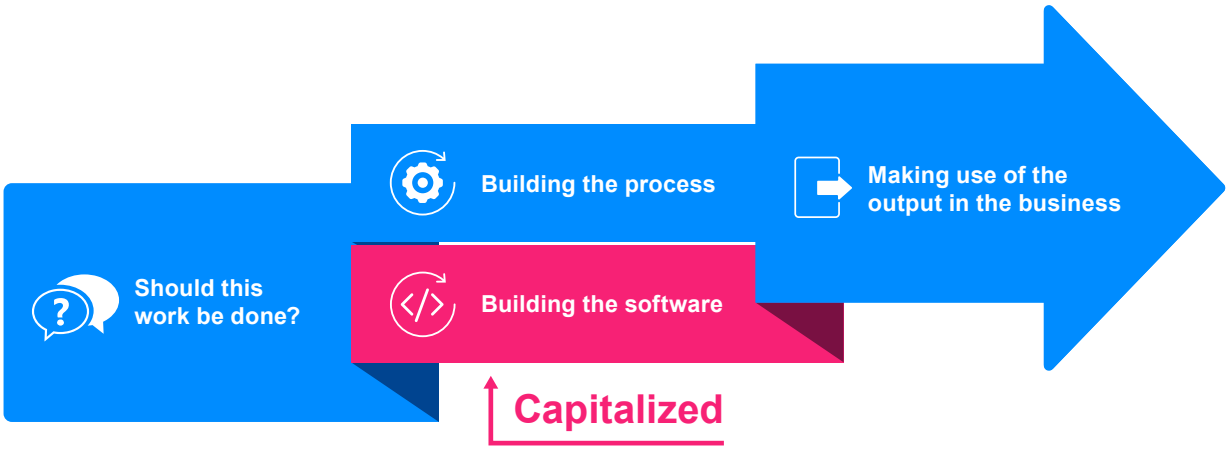
Years ago, we coined the term "messy middle" to make clear what was happening in IT. The top five enterprise programs generally receive appropriate attention and staffing. Priority-one production outages get the staff they need to fix the problem. **Everything else is in constant competition for resources because once a project starts, everything has the same effective priority.**

Because of the problem with resource constraints, many PMOs in IT have ended up with a merged responsibility for operations and project work. We believe it's this mixed responsibility that has been making the capitalization issues more confusing than they should be.

The push toward digital business may have a positive impact on this situation. Executive management in most companies is pushing to force the typical IT "run" expense be reduced from an average of 70 percent to 40% percent. PMOs will need to become much more sophisticated about how they use the tools available to them.

Given this situation, how can the modern PMO ensure that the money budgeted for specific software development project is actually going toward work on a defined business outcome? We would suggest that all capitalized software efforts (be they agile or waterfall) have a cost breakdown structure. We understand that almost no one does cost breakdown structures anymore because years ago project software drove people to focus on tasks, but with the advent of tools like CA Project & Portfolio Management 15.3, software has come a long way.

FIGURE 1.
The Complete Cost of the Program



What would a cost breakdown structure look like? It is deliverable- or outcome-focused and includes what was formally known as top-level work packages. In agile speak, it could be at the epic, or in some cases a feature if the feature is large enough. Twenty years ago, software tools simply didn't support this way of thinking, but now there are tools that can capture cost at any level (work package, deliverable or feature) that is appropriate. Actualization needs to happen at a definable activity level to clearly denote to the auditors whether it is capital or expense. For most organizations, we recommend tracking time at the activity level.

Figure 1 shows a complete program and highlights that it is only the activity of building software that can be capitalized. But even there, all expenses aren't capitalized. Administrative time isn't included, and some activities around planning are not included.

Drilling down a level of detail, do meetings count? What about the daily scrum? The first answer is to use common sense. If the software can't be successfully completed without a particular activity, then time gets charged to the project. We chose to use the word "project" here only as representing the top-level "build the software effort." Organizations may choose instead to say, "build the product" or "build the application." As long as we all agree that we are talking about the lower middle box shown in Figure 1, it's all good.

If on the other hand, the answer to the question is no, this activity is not required to "build the software," then the time is charged to somewhere else. Defining that somewhere else is a place where the PMO should exert a great deal of influence. The PMO needs to be involved in understanding where time is going without turning into an operationally oriented organization.

For years, project performance has been stuck at around a 65 to 67 percent general satisfaction rating. A huge reason for that is that most projects under-deliver in the eyes of the business. Why? We strongly contend it's because they are effectively under-resourced. Even if a good developer charges all his planned hours to the project, those hours might have been spent as an hour here or two hours there for three weeks instead of a concerted effort for one week. What we know from personal observation is that with this level of interruption, all creativity stops. To get work done on time, even the best developer will have to settle for getting the minimum done, and a mediocre developer will do what appears to be the minimum in the easiest way possible.

This dedication to one work effort at a time, and doing so with team consistency, has been proven to lead to more outcomes delivered with higher quality. But the real financial gain is predictability. Planning capital spend becomes easier if agile maturity is present. This team consistency might allow an organization to consider no longer collecting time, but only after significant predictability has been achieved in the agile development practices and the auditors have agreed.

For actualizing capital expenses, the PMO that is overseeing capital software projects must know how software is developed. The manager of an enterprise program management office probably won't know how to provide real oversight to a software project, and that is why there will always be some form of PMO in IT.

Capitalization ends when the software is complete. Large rollouts of software that require specific customizations and new code to work as intended make this distinction a little fuzzy and often lead organizations to over-capitalize (with the whole rollout team rather than just the developers coding the new features). Again, **SOP 98.1 does not capitalize implementation costs. It is designed solely to create parity between purchased software and internally developed software.**

To summarize:

- The only thing that can be capitalized under SOP 98.1 is the actual time it took to "build the software"—all implementation cost is not capital.
- The PMO is responsible for oversight on what is being delivered as capitalizable software. Is the effort (however it's being developed) on track? Are there potential risks that would cause the project to be canceled? Are there rumors that quality is so poor it will have to be "rewritten in production"?
- The PMO does not need to have project managers who report to it. The PMO needs enough staff itself to check on what is happening (and by that we mean phoning around and walking around—not just reading status reports).
- The PMO needs to understand where the time is going—too many interruptions can put an entire project (or product or application) in jeopardy. People don't have to be assigned full time; they do have to be assigned smartly. If you don't have a tool that can help,buy one.
- When planning what initiatives to capitalize, the criteria needs to cover value, feasibility, and sustainability
- When actualizing capitalization, the granularity of the time tracking should be at the right level to ensure people will easily fill them out and that it's possible to know where their time is going. Tracking at the activity level is too detailed to support effective cost tracking.

Advice: It's time for the IT PMO to grow up and realize that it is responsible for the investments it manages. We realize some cultures can make this difficult, if not impossible, but we see the tide turning with the CIOs we speak with. To do this, it is imperative that the PMO understands directly what the projects are supposed to deliver and why. Documentation isn't the answer; human conversation is. Specifically, conversations with the business, conversations with the development teams, conversations with the product manager. Minute compliance is not important—appropriate rightness is. What we love about SOP 98.1 is that it gives the PMO a soapbox to stand on if used correctly. It requires the right level of detail and focuses attention on the right aspect—producing valuable software. So thank the agilists for creating this crisis. It's an opportunity you shouldn't pass up.

The Product Manager Perspective

Having covered the issues around how to have a waterfall/agile unified approach to SOP 98.1 for software development projects, we now come to the issue of what to do with software "products." It is critical to understand that the designation of what constitutes a product in the IT world has multiple definitions, and this requires being scrupulous in clarifying the terms.

1. **A software product that is sold or is embedded in another product that is sold to external customers. The capitalization rules for this type of product are governed by FASB 86.**

   The distinction between software that services the end customer directly and software developed to support internal operations is not arbitrary hairsplitting. It turns out it's a distinction that facilitates good investment management, and it makes the job of a product manager much easier.

2. **A software system that is designed to service end customers directly but does not generate revenue itself—an easy shorthand approach is to label this system "revenue enablement." SOP 98.1 applies.**

   To explore this in more depth, we'll take an example of a platform that allows a customer to buy insurance over the Internet without involving a salesperson. We'll assume that the platform was originally justified as a project-based investment and was capitalized as we've discussed under SOP 98.1. In year one of operations, as would be true of any other system, there is a constant stream of

requests from customers to add features or change the way the system operates. Pure maintenance, what might be considered a bug, is always expensed. A single feature request would also be expensed, since it would fall under the capitalization limit (again, typically over $100,000). A group of features could be packaged into a release, which could be capitalized. And here is where we hit all the shades of gray.

Issues:

- **Capital expenses must have appropriate governance.** If $100,000 is the line for capitalization, only someone with a spending authorization of more than $100,000 can approve the release. By definition, this means a product manager rarely has the authority to create a capitalizable release on the fly by dynamically grooming the backlog. The product manager does have the authority to optimize what features are delivered in an agreed-upon release; that is their job. They just don't have the authority to spend $100,000 on the fly with no one outside of their area deciding whether or not the money would be better spent somewhere else.
- **The capital budget can include any amount of additions to an existing asset,** but for any new additions, an analysis should be made of how much is completely new and how much is a replacement. At the 50,000-foot level, there is an assumption that in the first year all the features that are no longer useful have been written off (that's what the amortization is intended to do), and the new features, even if they are replacements, are in fact new and valuable features. With end-customer-facing software, determining the real useful life of a code base is important. In our example of a Web-based form to buy insurance, we can assume that a three-year life might be appropriate. In the case of a mobile app, we are aware of organizations that have chosen to expense everything on the assumption that the code base itself turns over too quickly to justify capitalization.
- **Much of what has been written about capitalizing software is premised upon the fact that organizations want to capitalize as much as possible because it will improve short-term earnings.** This concept makes senior financial managers shudder (see Donna Fitzgerald's blog post "Software Capitalization for PMOs: a Not-So-Quick Primer")[1]. No one in IT should ever believe they are helping their company through excessive capitalization. They are just creating more work for the finance people that have to track down the bloat and write it off.

3. A suite of internal software applications (possibly from multiple vendors or internally developed), all of which serve the needs of a specific business function. SOP 98.1 applies.

The primary reason to differentiate between customer-facing (products) and internal-operations-facing (applications) can be best understood by exploring Gartner's concept of pace layering. See Table 2.

TABLE 2
Pace layering

	Economic advantage	Investment risk	Rate of change impacting the system	Investment strategy
System of innovation	Very High	High	High	Small bets
System of differentiation	High	Moderate	Medium	Sized based on estimated value
System of record	None	Low	Low	Minimize as much as possible

Innovation is the lifeblood of an organization, especially in the new digital economy. Since not everything that an organization will try will pay off, there needs to be money set aside to fund a continuous stream of small bets. These investments will be an expense.

When a great idea works, the organization will need to invest in developing a system of differentiation—a production-grade system that offers a capability the competition doesn't have. Systems of differentiation are where firms should be spending their money. These systems generally have a three- to five-year lifespan. Their competitive advantage may wane more quickly, but once built, they become an operational necessity.

Systems of record are things like ERP and CRM. They are expensive, and organizations often spend the highest percentage of their capital continuously customizing them. The only problem is that about half that spending is money that would be better spent elsewhere (based on Gartner estimates). PMO software has progressed to the point where a modern PMO can be analyzing outcomes against the desired strategy and then mobilizing operations to deliver effectively. This transparency into cost, resource usage and benefits is what the Modern PMO uses to assess the value and to provide input into the decisions around which projects to capitalize and which products to replace, retire and invest in for overall cost reduction.

Many in the agile community say that ERP should be treated as a commercial application, but then they add that it should be funded with a fixed level of investment and a value measure that is limited to a smiley face from the functional area they support. Please look at Table 2 again. There is no competitive advantage to the ERP system if it is not something you sell. The cheapest alternative that does the job is all that is required, and in theory, the pace of change is relatively slow, meaning there should be little need to continuously improve the system. From the perspective of the product manager, their job with regard to systems of record is to ensure that the few changes and enhancements that will be made in any given year are the absolute right ones for the organization as a whole. These will still need to be approved (capital spending must be approved), but what should be in the backlog is as little as possible. Where an organization should be focusing its product managers is on the systems of differentiation, which in today's economy is some flavor of digital business. A recent article in Strategy+Business magazine, "10 Principles for Leading the Next Industrial Revolution,"[2] show clearly the challenges that await a modern company and the demands that will be placed on a product manager.

To summarize:

- Product management will significantly improve the compliance with SOP 98.1 and make the accountants and the PMO happier.
- Product managers need to take responsibility for ensuring that business units get what they need (for systems of record), not necessary what they want. This is a role that even the modern PMO would be unable to perform as well.
- The contents of a capitalized release still need management approval and beyond a certain level should potentially be reviewed in the investment portfolio. Treating the software development process as if it were project-oriented versus product-oriented doesn't change anything. FASB 86 (which covers saleable software) still requires approval and governance.
- The Modern PMO includes managing all outcomes and all spend.

Advice: Creating a distinction between end-user-facing software (products) and internal operating software (applications) will significantly improve the ease with which spending allocations can be made. Support building a strong culture of product management, predictable teams and release plans. We can't stress strongly enough that these people should be your best friends. Their job is to know what is going on, and they have contacts where you don't. Also, having people who deeply understand the quality and robustness of the code base (fragility, technical debt) makes annual capitalization assessments easier.

Conclusions

As stated at the beginning of this white paper, we believe it's time to define a modern approach to capitalization of internal software that is in keeping with the guidance of the ASec and gets us the best software possible with as little overhead as possible. There are places where we have strongly suggested some best practices that are not necessarily common today, but in all cases, these recommendations are designed to ensure that the organization receives the highest possible value for the money invested. Will this be more work than what organizations do today? We don't think so—in the final analysis, we believe it will be different work that will help an organization function better in the new world of digital business.

For more information, CA Project & Portfolio Management supports financial management, please visit **ca.com/ppm**

1 Donna Fitzgerald, "Software Capitalization for PMOs: A Not-So-Quick Primer," September 5, 2017, https://www.ca.com/en/blog-ppm/software-capitalization-pmos-not-quick-primer.html

2 Strategy+Business, Norbert Schwieters and Bob Moritz, "10 Principles for Leading the Next Industrial Revolution," March 23, 2017, https://www.strategy-business.com/article/10-Principles-for-Leading-the-Next-Industrial-Revolution

200-3150512_1117