Capitalizing on Systems Engineering

Jason J. Sherey
ICTT, Inc.
100 East Campus Drive, Terre Haute, IN 47802
(812) 232-5968   sherey@ictt.com

Abstract. Project managers often find it difficult to justify allocating significant resources and schedule to systems engineering tasks when "real" engineering has to be done. With ever-decreasing time to market demands, systems engineering continually loses out to design, integration, and test. A new method of systems engineering called Pattern-Based Systems Engineering (PBSE) enables companies to transfer portions of systems engineering costs out of project specific budgets and into company capital asset accounts. Such a change in accounting provides a series of benefits that include improved documentation and management of core corporate intellectual property, best practices, and standards as well as not having to reserve as much precious project money on tasks that need constant re-justification to product development managers with typically constrained budgets. This paper reviews PBSE, relevant accounting standards, and how much of systems engineering can be performed as a company-wide capital asset development program instead of as project overhead.

The Current Situation

Justifying Systems Engineering in Project Budgets. Systems engineering is sometimes described as a way of reducing risk to product development and support areas such as:

- Project Schedule,
- Life Cycle Cost,
- Integration and Test, and
- Product Performance (ICTT 2005b).

Most of these cost impacts, however, occur after product development phases and during support phases whose costs are often reported in completely different organizational accounts (Blanchard et al. 1998). The effects of not performing enough systems engineering are therefore not very visible to the product manager planning development budgets. This creates an "iceberg" effect in which the project manager figures a small allocation to systems engineering is all that is needed while the hidden yet much larger costs of maintenance, distribution, market share losses due to poor performance, and other life cycle support costs are ready to cripple the company (Blanchard 1998).

Arguments that explain that while program costs are spent mostly towards the end of the life-cycle the costs are actually committed based on the work very early in the program when most of the systems engineering is performed can be used to clarify the need for systems engineering (Blanchard 1998). However, systems engineering costs are usually accounted for in an overhead expense account in typical Work Breakdown Structures (WBS) that also includes costs for project management (DOD SMC 2001). When looking at the rest of the WBS, it is very difficult not to reallocate hours from overhead activities like project management and systems engineering to the large number of accounts that apply to specific technologies and component work or typical trouble accounts like integration and test.

No matter how much numerical proof is gathered to validate an explicit and direct link between systems engineering and the reduction of life cycle cost and/or improved product performance and market acceptance, it still will be very difficult to overcome the human tendency of project managers and business executives to fight fires and allocate money needed for systems engineering to other WBS accounts.

Product Line Pressures. While developing one successful product at a time is hard enough, today's economy dictates that a company must be able to introduce a set of similar products into various markets at the same time. In order to be successful, a company needs to strive towards commonality and configurability in their products and development processes (Lehnerd et. al. 1997).

Most companies usually try to solve the need for product commonality by institutionalizing "Best Practices" or company-wide standards. Often written and followed blindly, these standards don't allow for the intricacies of each project (Reinertsen 1997). Typically, each market introduction has its own product development teams. Additionally, each market has its own needs and pressures for different features. Given intense schedule and cost pressures, each project will characteristically compromise the standards they are required to follow. Even with good intentions to follow up with a more complying product, the development team for the next version will face the same pressures, and will eventually yield, to improve on that market's unique solutions and features. Therefore, even if the "Best Practices" had correct advice and design patterns, the set of product configurations in a given product line or portfolio tend to fragment into completely different designs and therefore completely different products (ICTT 2005a).

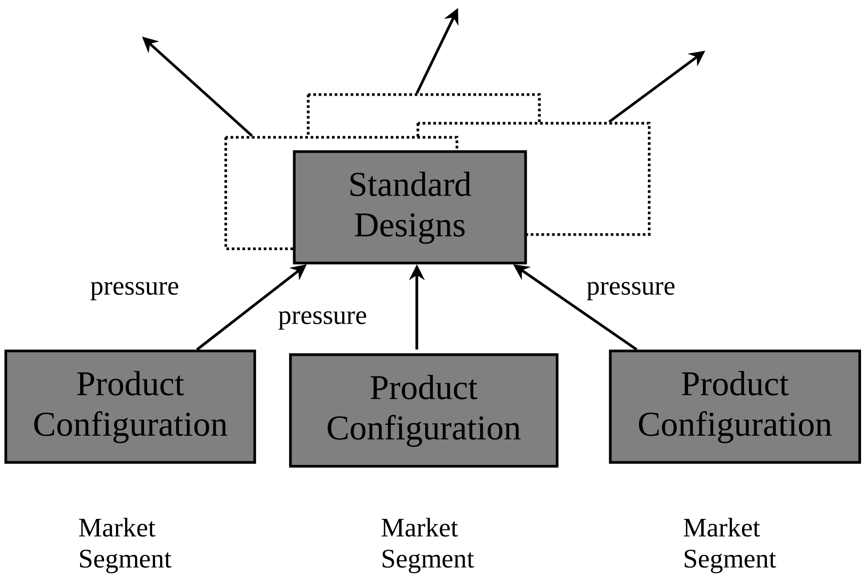

Figure 1: Market Segments Fragment Design Standards (ICTT 2005a)

The "Best Practices" and standards are typically inflexible due to the nature of the work that produced them in the first place. Once a company realizes the need for commonality, they usually commission work to develop "best practices" and standards from the experiences of successful engineers and project managers. Unfortunately, this effort is often a one-time task to capture current knowledge and practice. By definition, a new product will require some change to previous work, which instantly applies the market pressures previously mentioned. Therefore, the “Best Practices” and standards become out of date and difficult to manage entire product lines towards commonality.

A seemingly simple solution to this problem is to constantly manage and improve the standards, which requires a team of “Best Practices” consultants to actively maintain the standards. These staff members are often considered secondary team members while a project is on the line to develop a new product before an upcoming deadline (Babcock et. al. 2002). Project managers find it increasingly difficult to even work with the consultants to improve the standards, as they wonder why they should spend their project’s money on non-project discussions and issues. This opinion eventually manifests itself into the reduction of non-project specific company overhead to provide each individual project more resources. After all, the product development projects are the ones directly providing profit.

The effects. The end result, then, is that companies have a difficult time performing the necessary planning and analysis for each product by not allocating enough resources to systems engineering tasks. Nor are they very successful at enforcing or at least influencing commonality through “Best Practices” or design standards. Given the typical high variance of individual project outcomes (Reinertsen 1997), is it any wonder why it is so common to see products in similar portfolios or product lines have such a lack of commonality that they use different parts for the same functions (Lehnerd 1997)?

Pattern-Based Systems Engineering (PBSE)

The Solution. Systematica™, the systems engineering methodology this paper references, provides a process and meta-model that helps to both reduce systems engineering effort on a project basis as well as provide a better strategy on managing common corporate knowledge such as “Best Practices,” design standards, and “lessons learned.” The meta-model, summarized at a high level in Figure 2, captures the information relevant to any system or product’s needs, requirements, and designs (ICTT 2005c).

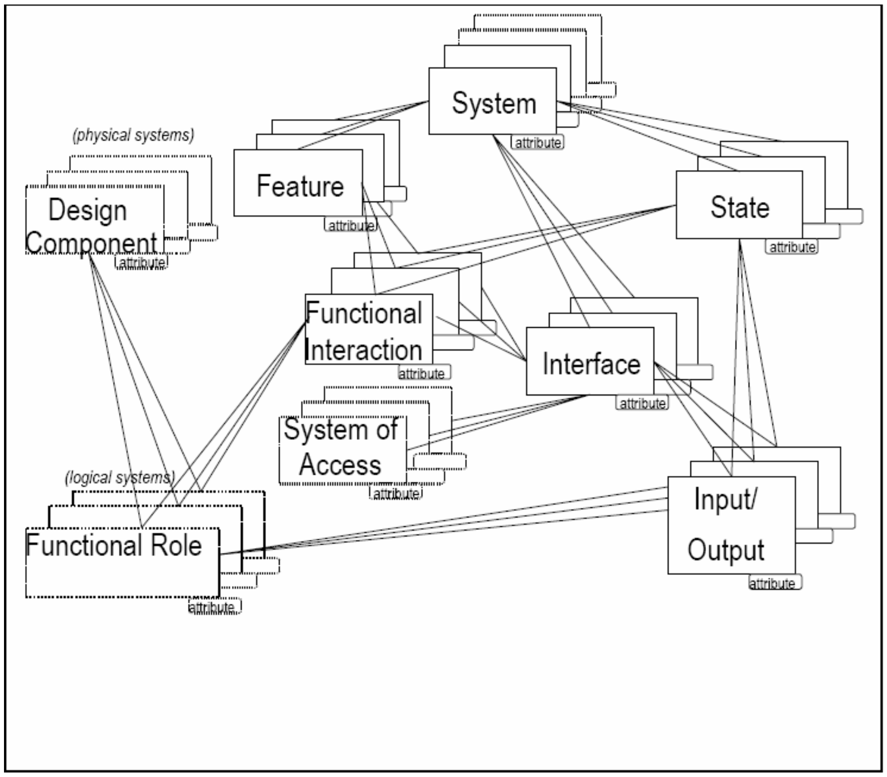

Figure 2: Systematica™ Informal Meta-Model (ICTT 2005c)

The use of the meta-model provides a company with the benefits of Model-Based Systems Engineering (MBSE) as opposed to normal prose methods. MBSE helps to more clearly understand, catalog, model, analyze, and even simulate a system's requirements and designs (INCOSE 2005a). While this paper does not intend on reviewing the meta-model in detail, it includes standard systems engineering concepts such as Systems, Features, Interfaces, States, Input/Outputs, and Design Components, as defined below (ICTT 2005c):

- Architectural Relationships: A relationship that summarizes the architectural significance of a set of interactions between systems.
- Class Hierarchy: A General-Special hierarchy, in which each progressive layer is more specialized case of the layer above it.
- Containment Hierarchy: A Whole-Part hierarchy, in which each progressive layer is a part of the layer above it.
- Design Component: A Physical System that is within a Subject System's Physical System Containment Hierarchy and to which is allocated Functional Roles.
- Feature: A collection of Functional Interactions that has marketable value or provides a valuable service.
- Functional Interaction: An interaction of Systems, expressed as an external (outcome) relationship in which at least one system affects the state of another system.
- Functional Role: The behavioral description (and therefore a Logical System) of a part played by a System in a Functional Interaction's relationship.
- Input/Output: That which is externally exchanged between interacting systems.
- Interface: The association of a System with a set of its Functional Interaction(s), Input/Output(s), Architectural Relationship(s), and System(s) of Access.

- Logical System: A system identified solely by its externally viewable behavior or responsibility.
- Physical System: A system identified solely by its physical identity or make-up.
- State: The condition of a system that determines its interactive behavior, viewed externally from the system.
- System: A collection of interacting components.
- System of Access: A System providing the means of access for interactions between other Systems.

While MBSE provides certain benefits, Systematica™ combines it with a Class Hierarchy concept to create a methodology called Pattern-Based Systems Engineering (PBSE). Using the very same meta-model, an abstract model can be generated by "uncovering" the common requirements and design models of two or more similar systems. This abstract, reusable model is called a pattern, and is described with exactly the same concepts defined above. An inheritance model can then be constructed and audited such that any modeled item in a specialized class can be examined if it and its related concepts have an abstract origin in the relevant pattern. This paper does not intend to describe the referenced PBSE process, depicted in Figure 3, in detail, but a brief summary follows (ICTT 2005a).

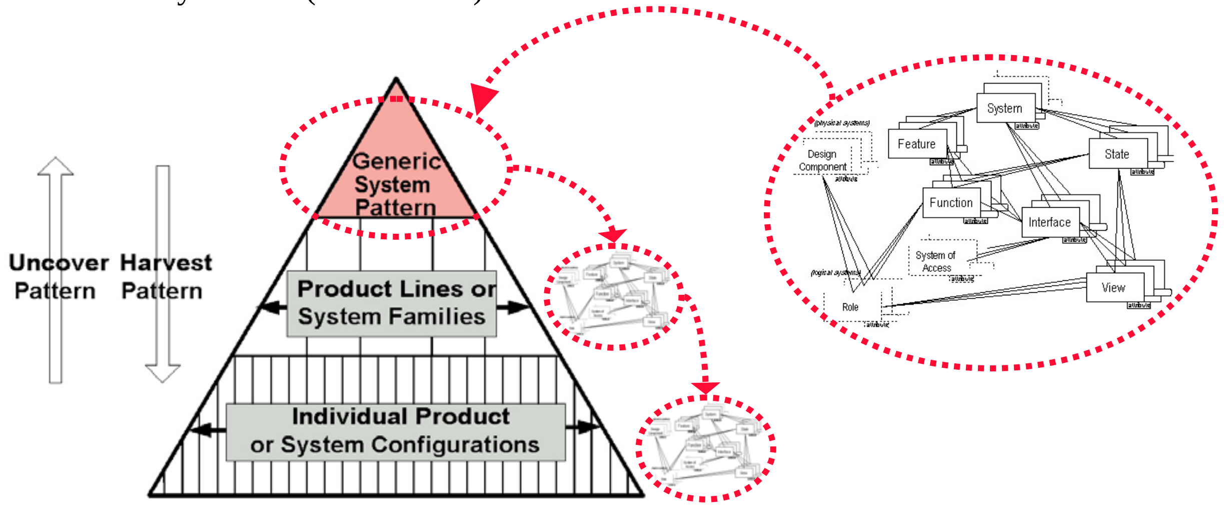

Figure 3: PBSE Pattern Pyramid (ICTT 2005a)

The pyramid in Figure 3 represents a Class Hierarchy of system or product models. A system or product in any of the layers of the pyramid is represented by the meta-model referenced earlier. The bottom layer of the pyramid includes the requirements and design models that are specific enough that the resulting specifications can be followed to build a product or system configuration. The second layer in the pyramid as shown in Figure 3 represents complete models, described with the referenced meta-model, that capture the common aspects of the configurations in the bottom layer similar enough to be part of the same product line or system family. The top layer includes complete models, again described with the very same meta-model, that capture the common aspects, both requirements and design, across all of the product lines or system families of a company (ICTT 2005a).

The process by which product or system configurations on the bottom level are abstracted into product line or system family patterns or by which product line or system family models are abstracted into core system patterns is called Uncover the Pattern™ (UTP). This process requires active pattern management to ensure the pattern includes the most up-to-date commonalities of its configurations.

The process by which a new product line or system family model is created from a core system pattern or by which a new product or system configuration is created from a product line or system family pattern is called Harvest the Pattern™ (HTP). This process involves creating a configuration by starting with a pattern and explicitly managing the consistencies between the two.

Organizationally, the PBSE process can be depicted as in Figure 4.

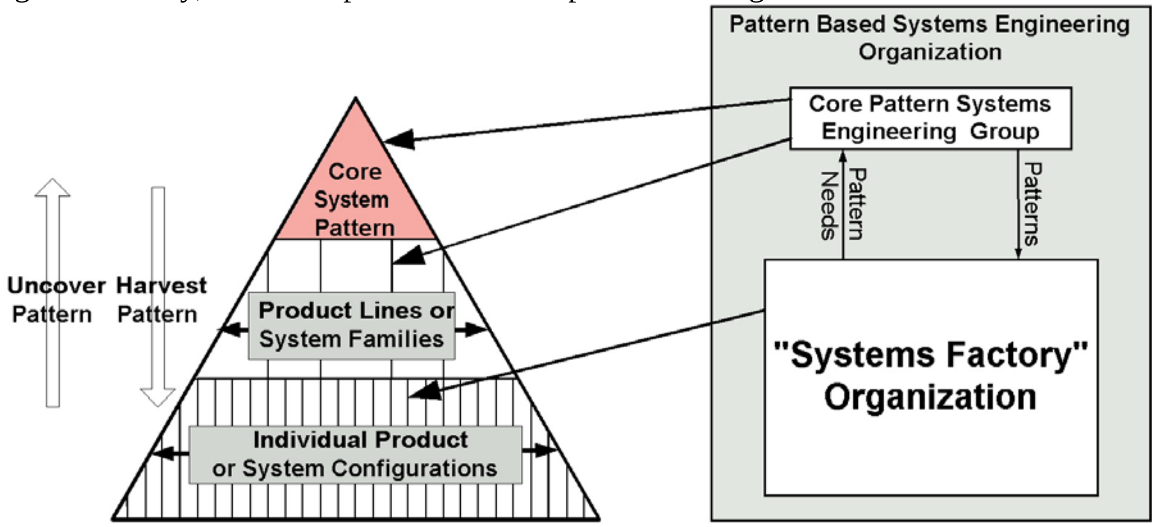
Pattern Based Systems Engineering Organization
Core Pattern Systems Engineering Group
Pattern Needs
Patterns
"Systems Factory" Organization
Core System Pattern
Product Lines or System Families
Individual Product or System Configurations
Uncover Pattern
Harvest Pattern

Figure 4: PBSE Organization Summary (ICTT 2005a)

A small group of experts, labeled the "Core Pattern Systems Engineering Group" in Figure 4 own, update, and manage the patterns in all but the bottom level of the pyramid. The larger group of systems engineers become the '"Systems Factory" Organization.' This group quickly configures the patterns into models that represent solutions meeting the needs of a new product and/or market segment (ICTT 2005a).

PBSE and Project Budgets. Given fully managed patterns of product line systems, system engineers in product development projects are freed to focus solely on the valuable new ideas, requirements, architectural considerations, trade space decisions, and designs that are specific to the product introduction and market segment. One could call the resultant process "Systems Engineering by Exception" due to the parallels of the management philosophy called "Management by Exception (Bazley et. al. 2004)." The supplied patterns act as the planned budget, and the differences between it and the needed product description are the areas needing systems engineering attention by the product development team.

The PBSE process does not suggest blindly following a pattern and perform no systems engineering at all during projects. However, the initial needs analysis and concept discovery processes can be done relative to the patterns provided. The pattern can act as a filter by identifying those needs and concepts of the new system that do not fit the company's core technology and product understandings. The remaining needs and concepts should be considered the major risk areas of the project and mitigated by focusing the project systems engineering effort on them.

PBSE impacts the budgets of projects by playing to the previously discussed tendencies of project managers. Patterns and systems engineering are more clearly understood as risk management tools and thus are more justifiable. In addition, PBSE allows much of the work of systems engineering to be done by pattern managers and accounts not directly tied to the project. Only resources needed to analyze the actual and value-added differences between the core product line system and the potential product are required to be allocated in the project's budget.

PBSE and "Best Practices." Just like "Best Practices," PBSE captures core technology and product knowledge and intends to distribute them for use in future product development projects. There are, however, two major differences between patterns and "Best Practices." First, patterns are abstractions, and second, they are actively used to manage risk in individual projects.

The standard method of implementing "Best Practices" is to develop some cataloging system with searching capabilities. Then, trade studies, white papers, successful design standards and patterns, and "lessons learned" are faithfully added to the company-wide database. Development processes are also updated such that project managers and engineers are required to review this database at the beginning of a project to mitigate risks of repeating previous mistakes and to attempt to manage towards commonality between products. The long lists of "Best Practices," standard designs, and "lessons learned" become very difficult to sift through even with advanced searching mechanisms. Additionally, even if information is actually uncovered that is relevant to the new project, it is often written in a manner that is difficult to understand how it directly impacts the tasks at hand (ICTT 2005a).

The UTP™ process alters this strategy by **<u>abstracting</u>** the "Best Practices," design standards, and "lessons learned" into an actual model of requirements, designs, and configuration rules that should be common across similar products or systems. The resulting pattern is modeled and viewed in the exact same language and manner as the models of specific product or system configurations with the major differences being some features and components being optional and required performance and design parameters having allowed value ranges instead of specific numbers. A valuable and useful intellectual property asset arises from the process as the pattern is a model-based specification of both common requirements and designs of the essence of core product ideas in a product line or product lines across an entire company. For instance, a Generic Lawn Mower Pattern would be a complete requirements and design model of what the Acme Lawn Mower Company considers all lawn mowers should be. The UTP™ process reviews each product development project, abstracts the lessons, practices, and models, and makes any updates to the patterns that may be relevant and helpful to future projects. The difference between PBSE and typical "Best Practice" implementations is that a new project would use the updated patterns as the starting point of its modeled-based requirements and designs of new product in the product line or family. There would be no sifting through and interpreting old "lessons learned," white papers, design standards, or "Best Practices," as they are already folded into the pattern by the Core Pattern Systems Engineering Group. Performing the UTP™ and HTP™ processes catapults a company into the often sought yet rarely achieved "learning organization" category (ICTT 2005a).

The second major differentiating trait of PBSE is the use of patterns to manage risk in specific product development projects. As the model-based requirements and designs of a specific product or system configuration are developed, they are checked to see how they conform to the appropriate patterns. PBSE employs a simple set of auditing rules called the Gestalt Rules™ using Class Hierarchy and inheritance concepts:

1. Every component class in the candidate model must be a subclass of a parent superclass in the pattern—no "orphan classes".
2. Every relationship between component classes must be a subclass of a parent relationship in the pattern, and which must relate parent superclasses of those same

component classes—no "orphan relationships".
3. Refining the pattern superclasses and their relationships is a permissible way to achieve conformance to (1) and (2) (ICTT 2005a).

These three simple rules lead to a valuable concept known as Return on Variation ™ (ROV). The Gestalt Rules™ determines how the requirements and design of a new or potential product differ from those of the core pattern. The incremental revenue and expenses can be determined to derive ROV™ as seen in Figure 5.

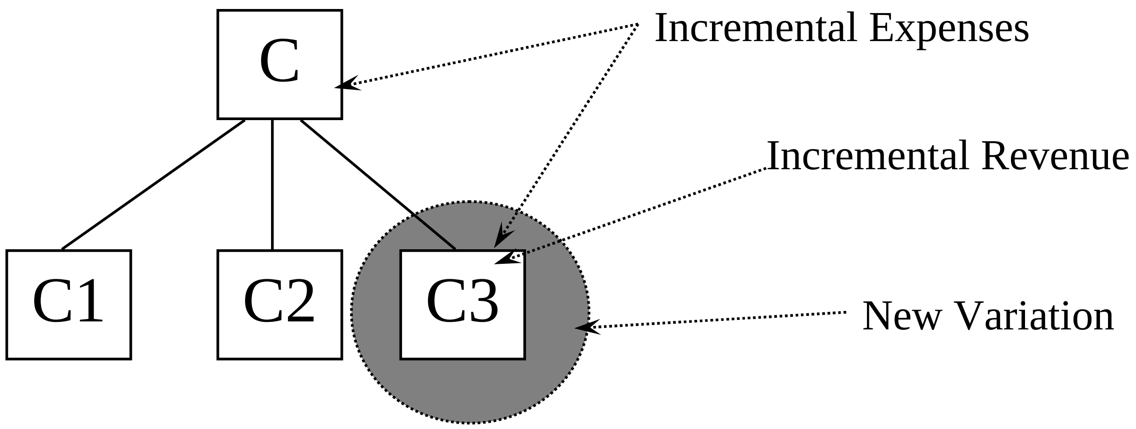

Figure 5: Calculating Return on Variation™ (ICTT 2005a)

The combination of the application of the Gestalt Rules™ and the Return on Variation™ analysis helps project managers explicitly manage whether each configuration variation with respect to its pattern:

1. Is truly value-added and should be kept and integrated into the pattern for future projects,
2. Is value-added for this project only but shouldn't be part of the pattern for future projects, or
3. Violates company standards, adding unnecessary risk without adding value, and therefore the model should be brought back into compliance.

During HTP™, only the configuration of the pattern and fixing any disapproved variations are applicable to project specific accounts, while the update to the pattern can be billed to non-project accounts.

Accounting For Pattern Management Costs

Patterns as Capital Assets. While it has been shown that PBSE can solve some important systems engineering justification, product line commonality, and intellectual property issues, it is understandable that companies would be hesitant to spend the significant resources required to create and manage the patterns. Company-wide overhead expense accounts are usually already considered too large and are frequent first-line targets for cost-cutting measures. However, consider these aspects of PBSE:

- Pattern management costs are significant,
- Patterns are reusable across more than one product/project and therefore are an asset used throughout an enterprise,
- With the right process implementations, patterns maintain their use over time, and
- Patterns can be used to directly aid and improve revenue producing product development efforts.

With these traits in mind, it would seem reasonable to regard patterns as capital assets. This would allow the expenses for pattern creation and management to be depreciated, spreading the investment across several years. In addition, including general patterns as capital assets could be an excellent way of showing more corporate value in the company's financial statements. Of course, there are some accounting standards and precedents that need to be considered in order to validate the idea that patterns can be regarded as capital assets.

Relevant FASB Statements. Unfortunately, the Financial Accounting Standards Board (FASB) have not provided standards statements and guidance on how to book pattern management expenses. The intent of this paper, then, is to review past FASB statements to suggest how to handle pattern management expenses.

The first relevant FASB statement is FASB Statement #2, which describes how to account for research and development (R&D) costs. R&D projects are so prone to failure, that it is considered unlikely that any investment in R&D can be directly associated with any revenue. Therefore, FASB suggests booking all R&D costs into overhead expenses until the costs are directly associated with revenue producing efforts (FASB 1974).

FASB Statement #86 seems a little more useful since it is regarding software development costs. Patterns, of course, typically are created, managed, and stored electronically, which may help a company apply this statement to pattern management costs. Again, FASB is aware of the significant risks of software development, and so it also suggests recording those costs as R&D expenses as well. However, it says once a working model and detail design of the software can prove technical feasibility, then development costs afterwards can be capitalized. Maintenance and customer support should still be considered overhead expenses (FASB 1985).

FASB Statement #142 applies to intangible assets, which seems applicable to the intellectual property aspects of PBSE. Some example Intangible Assets that can be capitalized are customer contact lists, patents, and copyrights. Unfortunately, it is difficult to prove expected use, specific fair value, level of maintenance required to obtain the expected cash flows, and the effects of obsolescence and technological advances on the overall value of the intangible asset. Most companies are wary of these problems and choose to take the conservative approach in not listing such assets in their financial statements so as to prevent the possibility of presenting an inflated corporate value (FASB 2001).

Patterns can be easily construed as intangible assets, making FASB Statement #142 the most applicable standard. However, due to the reluctance of most companies to use the statement, it is best to take an indirect approach. Because patterns are complex models that include needs, requirements, and designs, special software is required to automate the PBSE process. In fact, there are several modelling tools available that help automate modelling in standards such as Unified Modelling Language™ (UML) (INCOSE 2005b). Patterns can be thought of as a way to customize these tools so that they better reflect the PBSE process and enforce the use of the patterns throughout the company. Training for the customized tools becomes less focused on how to generate UML™ models, but how to view and configure the already provided pattern models. By considering pattern management as a software customization task, FASB Statement #86 can be applied. A very useful side effect of this interpretation is that it greatly improves the probability of proving the generic UML™ tool's technical feasibility. For, typical product engineers are not expert modellers. Customizing these tools with the generic pattern allows a company to transfer strategies from "teaching modelling to teaching the model (ICTT 2005a)."

Pattern Management and Revenue Producing Efforts. The most important threshold to be able to regard patterns as capital assets is to prove a direct link to revenue producing efforts. A helpful strategy in proving such a link is to commission a separate project whose objective is to continually customize the software modeling tools by maintaining patterns that reflect current core corporate product and technology knowledge. Creating a separate project or program for this effort is important in order to use current process infrastructures to advertise, deploy, and support the patterns internally. The project manager for the pattern management project would be held to the same standards as any typical product development project manager: the customization of the software must satisfy and be used by the project's targeted customers. The customers and stakeholders for the pattern management project could include the following organizations or departments:

- Marketing,
- Business development,
- Business management, and
- Individual product development project teams.

Each of these customers and stakeholders can directly use the patterns in the customized software to help with the following revenue producing tasks:

- Review of product offerings across market segments,
- Selling product features and capabilities,
- Identification of new markets to enter,
- Reducing market entry redundancy,
- Parsing, understanding, and responding to RFP's,
- Communicating with Business Development and Engineering,
- Integrating of lessons learned and best practices,
- Identifying project risks,
- Analysing technology trade space and design alternatives, and
- Jump starting the modelling of a product or system's requirements and design.

Standard project reviews would be held, as with any normal product development project, to ensure that the patterns and software customizations are being managed towards feasible use for revenue producing tasks by the project's stakeholders. A company's deployment and support mechanism could be employed to actually implement and verify the impacts to revenue production.

Valuing Pattern Assets. In addition to proving a link to revenue production, capital assets must be able to be valued. Having a separate project commissioned to manage and deploy patterns as software customizations helps in this aspect of accounting as well. Project managers and accountants could use the following questions to determine the value of the patterns and software customizations:

- How much would each of the project's customers be willing to pay for use of the pattern models and views?
- How long would each of the project's customers find the pattern models and views useful in their revenue production tasks?
- How many times would each customer use each pattern "product" or "service" during that time?
- How many product or systems could be developed using the patterns and software customizations?
- How much pattern "maintenance" would be needed to continue the pattern's value as determined in the previous questions.

Given the pattern management project scenario, each of the above questions are legitimate, which helps underscore the idea that patterns can be capitalized.

Pattern Life Spans. All capital assets have life spans that indicate how long costs can be depreciated. Patterns and their associated software customizations are appropriate subjects of life span determining questions. For example, consider the following questions an accountant could reasonably find answers to regarding patterns:

- How often do new initiatives, technological advances, or market shifts change an understanding of the company's products, thus rendering the patterns obsolete?
- Given the company's culture and managerial and executive support, how long will the patterns be used?

Depreciating Patterns. It also appears reasonable to apply depreciation rules to patterns, given that capital asset valuation and life span properties apply as stated above. For example, with the pattern management project scenario in mind, the project could actually keep track of revenue resulting from the usage of the patterns and software customizations based on the impacts on its customers' revenue production. The standard Current/Total Gross Revenue ratio could be then used to depreciate a pattern's value. If this method is too complex to measure for a given company, the simple economic life straight line approach also would be appropriate.

Similar Capital Assets. In addition to meeting the above capital asset properties, patterns are used in the PBSE process in similar fashions as accepted capital assets such as building blueprints and jigs, tools, molds, and dies of proven technologies. The "Systems Factory" Organization in Figure 4 uses the patterns and software customizations to quickly configure value-added product or system configurations with managed commonality much like a factory uses molds and dies to repeatedly form products with the same shape.

Conclusions

In review, the following can be concluded:

1. Current strategies for justifying significant systems engineering allocations in project budgets rarely garner enough support to actually allow for proper systems engineering and risk management.
2. "Best Practices" and other current corporate knowledge capture techniques are not effective in managing commonality across product lines and portfolios.
3. Pattern-Based Systems Engineering better incorporates and manages core company intellectual assets into reusable requirements and design models called patterns.
4. These patterns can be used as capital assets to justify moving portions of systems engineering costs from project budgets to corporate capital expense accounts.
5. With the proper interpretations and implementation, pattern management can comply with accounting standards to allow pattern capitalization considerations.

References

Alexander, Christopher, *Notes On The Synthesis Of Form*, Harvard University Press, 1964.

Babcock, D. and Morse, L., *Managing Engineering and Technology, 3rd Edition*, Prentice Hall, 2002.

Bazley, J. et. al., *Accounting: Information For Business Decisions*, Thomson South-Western, 2004.

Blanchard, Benjamin S., *System Engineering Management*, John Wiley & Sons, Inc., 1998.

Blanchard, B. and Fabrycky, W., “Systems Engineering and Analysis”, Prentice Hall, 1998.

Brigham, E. and Ehrhardt, M., *Financial Management: Theory and Practice, 11th Edition*, Thomson South-Western, 2005.

Department of Defense Systems Management College, *Systems Engineering Fundamentals*, Defense Acquisition University Press, January 2001.

Financial Accounting Standards Board, *Statement of Financial Accounting Standards No. 2: Accounting for Research and Development Costs*, October 1974.

Financial Accounting Standards Board, *Statement of Financial Accounting Standards No. 86: Accounting for the Costs of Computer Software to Be Sold, Leased, Otherwise Marketed*, August 1985.

Financial Accounting Standards Board, *Statement of Financial Accounting Standards No. 142: Goodwill and Other Intangible Assets*, June 2001.

ICTT, Inc., “Patterns-SE Class”, February 2005.

ICTT, Inc., “SE Class Intro”, February 2005.

ICTT, Inc., “Systematica Abbreviated Glossary”, version 1.6.2, July 2005.

INCOSE Model Driven System Design Working Group (MDSDWG) web site, http://www.incose.org/practice/techactivities/modelingtools/mdsdwg.aspx, 2005.

INCOSE Tools Database Working Group (TDWG) web site, http://www.incose.org/practice/techactivities/modelingtools/tdwg.aspx, 2005.

Lehnerd, A. and Meyer, M., *The Power of Product Platforms*, The Free Press, 1997.

Reinertsen, Donald G., *Managing the Design Factory*, The Free Press, 1997.

Rogers, G. and Schindel W.D., “Methodologies and Tools Supporting Continuous Improvement” *Journal of Universal Computer Science*, March 2000.

Schindel, W. and Smith, V., “Results of Applying a Families-of-Systems Approach to Systems Engineering of Product Line Families”, SAE International, Technical Report 2002-01-3086, November, 2002.

Schindel, William D., “Does Our SE House Have a Foundation?”, INCOSE Crossroads of America Chapter technical program presentation, Peoria, IL, May 22, 2002.

Schindel, William D., “Requirements Statements Are Transfer Functions: An Insight from Model-Based Systems Engineering”, *Proceedings of INCOSE 2005 Symposium*, July, 2005.

Schindel, William D., “System Engineering: An Overview of Complexity’s Impact”, SAE International, Technical Paper 962177, October 1996.

Sherey, Jason J., “A New Method to Justify Systems Engineering”, INCOSE Crossroads of America Chapter Fall Mini-Conference, Fort Wayne, IN, 2004.

Biography

Jason J. Sherey is a Senior Systems Engineer for ICTT, Inc., a systems engineering company. He practices, documents, teaches, and helps develop ICTT’s Systematica™ Methodology. He has modeled patterns for a variety of systems, including engines, tractors, trucks, software, business processes, manufacturing systems, and guidance systems. Jason has earned an M.S. in Systems Engineering and is finishing an M.S. in Engineering Management from Southern Methodist University. He also has a B.S. in Electrical Engineering from Rose-Hulman Institute of Technology. He is currently the president for the INCOSE Crossroads of America Chapter, which covers much of the Indiana and Illinois area.