WHITEPAPER

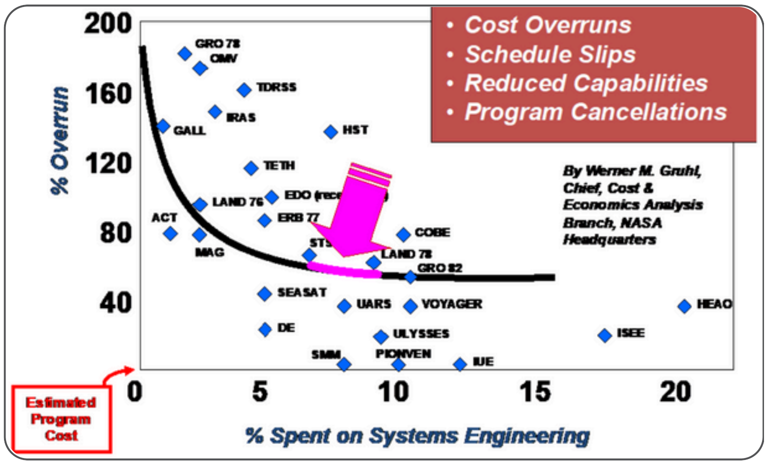
The Return on Investment of MBSE

DR. STEVEN DAM | STEVEN.DAM@SPECINNOVATIONS.COM

Introduction

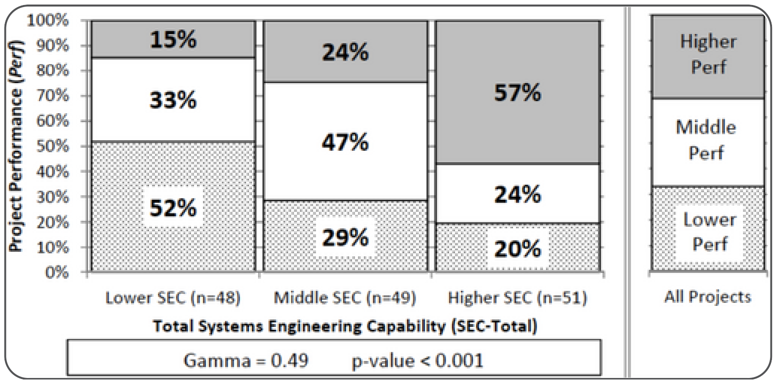
What is the Return on Investment (ROI) of Model-Based Systems Engineering (MBSE)? This question is one that many people ask. In fact, the International Council on Systems Engineering (INCOSE) has that as one of its tasks for their Value Proposition Initiative. A group of systems engineers is trying to find evidence to prove that MBSE has value. However, that becomes very difficult for a concept that has only been around for a dozen years, when the lifecycle of many of the systems of interest is measured in several decades.

We can approach this question by inference. If there is a significant return on investment in systems engineering, then we can infer that there might be one for MBSE. Fortunately, we do have many decades of experience applying systems engineering to projects since the 1950s (at least, depending on how you define systems engineering). One of the best analyses I have come across over the years was a very interesting piece of work by Werner M. Gruhl, who was at the time the Chief of the Cost & Economics Analysis Branch at NASA. His work was published in a NASA technical paper entitled, "Issues in NASA Program and Project Management" (NASA SP-6101 (08), in 1994. In a paper by Ivy Hooks in this publication, she states that "if the program requirements are not well understood, there is not much hope for estimating the cost of a program." She continues: "Werner Gruhl developed a history of NASA programs versus cost overruns" and cited the diagram below (redrawn due to the poor quality of the document found – an old scanned PDF). She interpreted this chart as "if you have not done a good job in Phase A and B in defining and confining your program, you are going to encounter large numbers of changing requirements and the cost will go up accordingly." Note that the figure below indicates the % Spent on Systems Engineering, which is really Phase A and B in NASA terminology.



Thus, it's clear that at least the combination of program management and systems engineering, which is what allows you to properly develop the set of requirements for the program, is required to keep the cost of the project from skyrocketing. Note that program management and systems engineering are flip sides of the same coin. The program manager optimizes cost, schedule, and performance while mitigating risk in each of these areas. The systems engineer is tasked to do the same for the system. That's why these two disciplines have been seen to overlap, which was recognized in a recent book by INCOSE and the Program Management Institute (PMI).

Another more recent attempt at qualifying the value of systems engineering came from the Software Engineering Institute. They conducted several studies to determine the value of systems engineering, including one documented in their November 2012 paper entitled "The Business Case for Systems Engineering Study: Results of the Systems Engineering Effectiveness Survey." The authors "found clear and significant relationships between the application of SE best practices to projects and the performance of those projects, as seen in the figure below.



Project Performance vs. Total SE Capability

In a later presentation by Mr. Joseph P. Elm, one of the authors of the 2012 paper, on "Quantifying the Effectiveness of Systems Engineering," he cites a finding for a General Accountability Office (GAO) report (GAO-09-362T) that states:

"… managers rely heavily on assumptions about system requirements, technology, and design maturity, which are consistently too optimistic. These gaps are largely the result of a lack of a disciplined systems engineering analysis prior to beginning system development …"

So, it is recognized that there is great value in performing at least the "right amount" of systems engineering. If we use the Gruhl graph as a basis, we need to spend around 7-12% of the program's budget on the combination of program management and systems engineering. Since the cost of the program could as much as double on average, according to the chart, the return would be about 10 times the investment. For example, if we spent $100,000 on systems engineering and program management and the overall cost of the program was $1,000,000, then since the cost could have doubled to $2,000,000, we save $1,000,000.

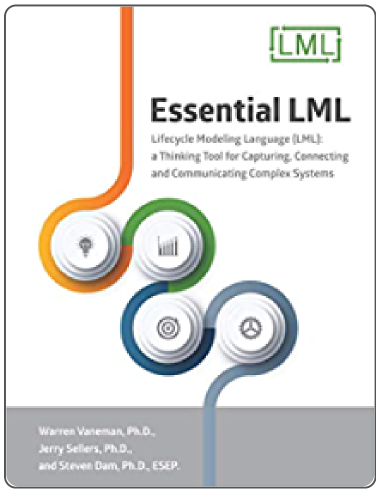
So now that we agree there is a substantial return on investment in systems engineering, let's get back to the question of ROI on MBSE. The question here is, "Does modeling help systems engineering?" Since we have always done modeling in systems engineering, I think that is clearly a part of sound systems engineering. But the flavor of MBSE being pushed by many in the community has equated MBSE to SysML, and many have also equated SysML as implemented by MagicDraw. But do SysML and MagicDraw® do all the things we need to do in systems engineering? In particular, do we obtain a good set of system requirements in a form easily used by all the stakeholders?

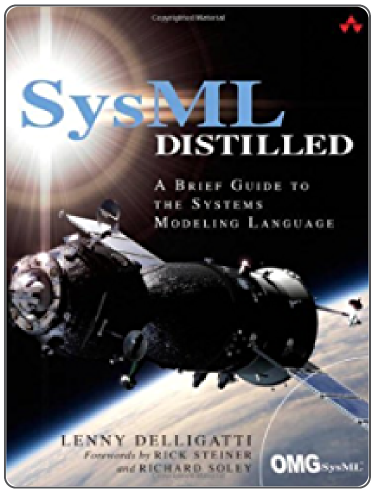
To begin to answer these questions, let's go back to Mr. Elm's paper. He states that the systems engineer must perform the following tasks:

- Requirements Development
- Requirements Management
- Trade Studies
- System Architecture Development
- Interface Management
- Configuration Management
- Program Planning
- Program Monitoring and Control
- Risk Management
- Product Integration Planning and Oversight
- Verification Planning and Oversight
- Validation Planning and Oversight

LML vs. SysML

SysML consists of nine diagram types, most of which were derived from software engineering practices, not systems engineering. Yes, there is overlap between the two, but not as much as the overlap between systems engineering and program management. That becomes obvious from the task list above, many of which include explicitly the word "management."

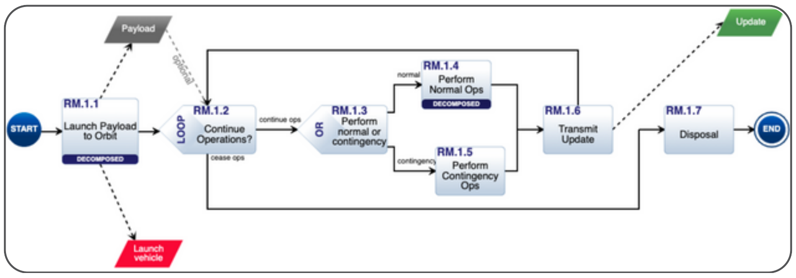

LML
Essential LML
Lifecycle Modeling Language (LML): a Thinking Tool for Capturing, Connecting and Communicating Complex Systems
Warren Vaneman, Ph.D., Jerry Sellers, Ph.D., and Steven Dam, Ph.D., ESEP.


SysML DISTILLED
A Brief Guide to the Systems Modeling Language
LENNY DELLIGATTI
Forewords by RICK STEINER and RICHARD SOLEY
OMG SysML

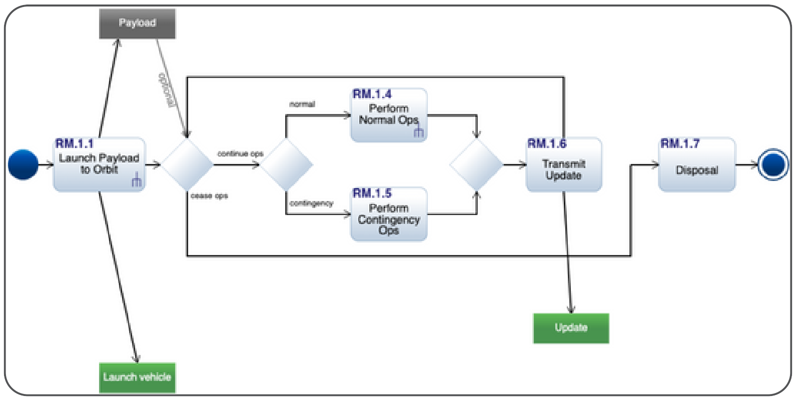
SysML also has proven to be very difficult for most other disciplines to understand, since they speak other languages. It also takes at least two large books, "A Practical Guide to SysML" and "SysML Distilled." In comparison, the Lifecycle Modeling Language provides a whole systems engineering ontology and limited diagramming that completely subsumes SysML. However, LML can still be explained in a very thin book, "Essential LML."

You're probably asking, "How is it possible that LML subsumes SysML?" By using an ontological approach that defines a set of entity classes and their relationships, along with the attributes of both, LML provides all the elements of a real language (nouns, verbs, adjectives, and adverbs). This ontology can be used to capture information easily and efficiently. Then that information can be displayed in many ways, including all nine SysML diagram types.

The tool, Innoslate®, proves this assertion, as it produces all nine SysML diagrams (and many more) from this ontology as extended in Version 1.1 of the LML specification. In addition to the SysML diagrams, Innoslate produces the LML Action Diagram, which represents the same information as the SysML Activity Diagram, but in a significantly more understandable form. We can see this when we compare side by side the two types of diagrams.



LML Action Diagram Example



SysML Activity Diagram

In the SysML diagram, I need to know what the diamond and fork symbols mean. In the Action Diagram, I know exactly what they mean, because the words: OR, LOOP, and Decomposed, make their intent clear. In addition, in SysML, I cannot just allocate the decision points to who or what performs them. I can in LML. Of course, if there were only two symbols I needed to decipher, then I would not care as much, but SysML has over 30 such symbols. You will need a "3-D decoder ring" to fully understand how to use all the symbols, hence the very large books and long training classes needed to try to learn SysML. This learning curve translates into a significant investment in the workforce to get them up to speed on this complex language. Of course, electrical engineers, mechanical engineers, logistics experts, and all the other disciplines have their own languages and have no interest in learning something this complex.

You might say, "But of course, MagicDraw overcomes these limitations in SysML?" The answer to that is not as well. In particular, if we go back to the list of systems engineering tasks, MagicDraw only does one "well:" System Architecture Development. Although MagicDraw has some limited requirements capability, almost everyone uses another requirements tool in conjunction with it. Innoslate by comparison has a robust Requirements View that includes automated requirements quality checking using the artificial intelligence technique of Natural Language Parsing (NLP). The requirements then can be directly traced to the diagram entities within the same tool, resulting in one database and no configuration management problems that you encounter having two databases.

Innoslate also has a built-in Test Center for the V&V activities. In addition, Innoslate provides Discrete Event and Monte Carlo simulators to verify that the Action (or Activity) Diagrams have been correctly done. We use that same approach to support Program Planning, Monitoring, and Control.

So back to the ROI discussion. Can MBSE provide a healthy ROI? Only if we do all the things we need to do in systems engineering. In addition, if we can use modern technology to help automate these difficult tasks, we can provide even higher ROI than systems engineering by itself. Innoslate and LML provide a means to provide this higher ROI, while MagicDraw and SysML actually cost much, much more to implement, and you end up with a poorer result. So, if you want ROI from your MBSE investment, use Innoslate and LML.

So far, we have identified the return on investment (ROI) for systems engineering (and program management) and discussed MBSE, noting that the type of modeling techniques and tools will determine the amount of ROI. Now, in the next section, we will try to quantify the differences.

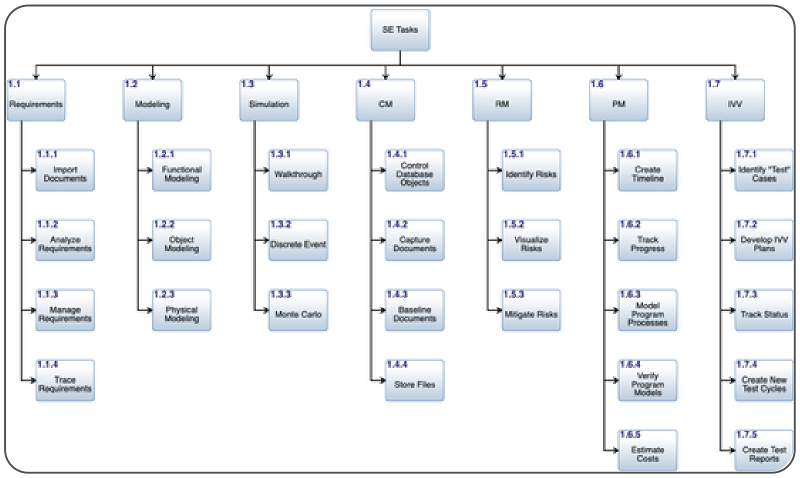
In general, today in systems engineering we have three types of techniques: ad hoc (the most prevalent); Systems Modeling Language (SysML); and Lifecycle Modeling Language (LML). The ad hoc requires you to follow system engineering processes to the letter manually, using nothing but MS Office (MS Word, MS Excel, MS PowerPoint/Visio, MS Access, MS Project). The SysML technique requires that you follow the systems engineering processes, which include rule checking for SysML diagrams, and supplement the "modeling" with requirements, testing, simulation, risk, and several other tools. For this analysis, I will look at a suite of tools to perform the systems engineering, choosing the most common "MBSE" tools being used (DOORS, MagicDraw, Python/Cameo Simulation Toolkit, Risk Register, MS Project, and MS SharePoint). The LML technique requires following systems engineering processes with extensive rule checking using Natural Language Parsing (NLP) technology, and Innoslate®. Note that several other popular tools could fit into this latter category since they are also ontology-based, but they do not have the complete feature set, and I will let the owners of those tools defend their ROI.

To assess the ROI, we will assume to use the systems engineering tasks of Mr. Elm's paper mentioned earlier.

To simplify the analysis a bit, the above tasks are grouped into the following 7 categories:

- **Requirements:** For both Development and Management
- **Modeling:** To include Interface Management and System Architecture Development
- **Simulation:** For Trade Studies
- **Configuration Management (CM)**
- **Program Management (PM):** Program Planning, Monitoring, and Control
- **Risk Management (RM)**
- **Integration, Verification, and Validation (IVV):** Product Integration, Verification, and Validation Planning and Oversight

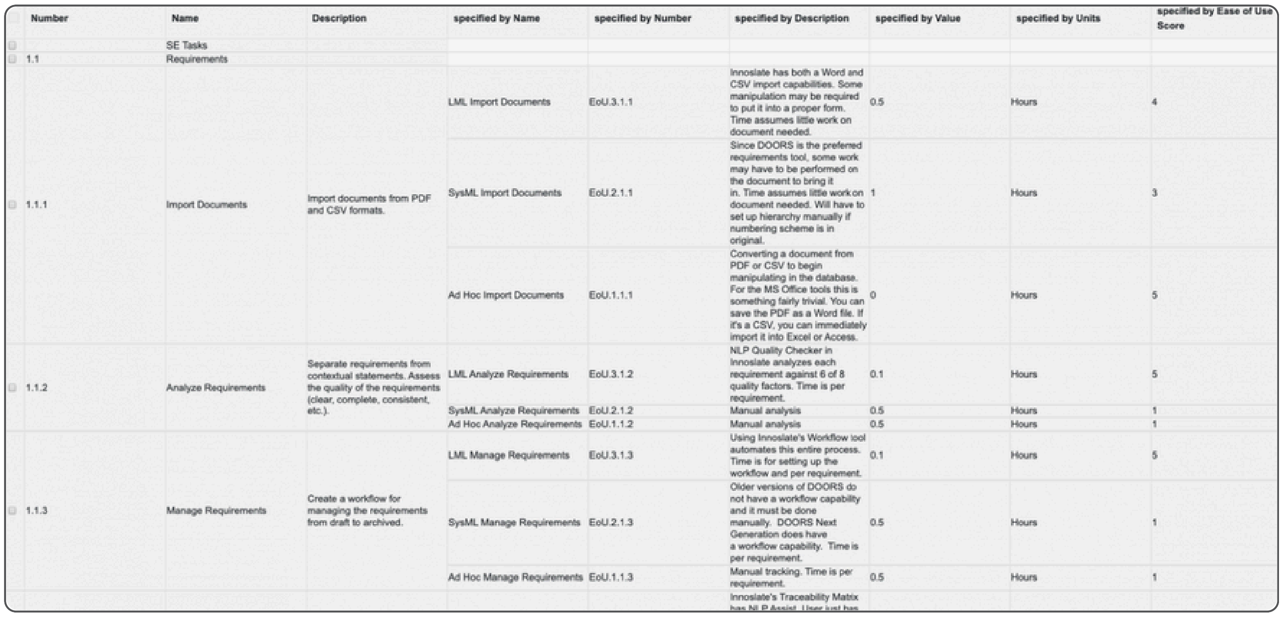
Now we want to get down a level in each area to identify specific tasks so that we can estimate the time it takes to accomplish these sub-tasks for each MBSE approach. The figure on the right shows the individual sub-tasks required. This list is not exhaustive, but should be sufficient for this analysis.



These tasks represent the "operations and maintenance" phase of systems engineering, while the initial startup costs are reflected in the price of the tools and the time to learn them. An additional source of costs comes from intangibles, such as collaboration and reuse. Clearly, there is a benefit in those intangibles, but it would be difficult to determine a value for them, so I won't try in this paper. Since most of the cost/benefit should be in the operational use of systems engineering, I will address that first.

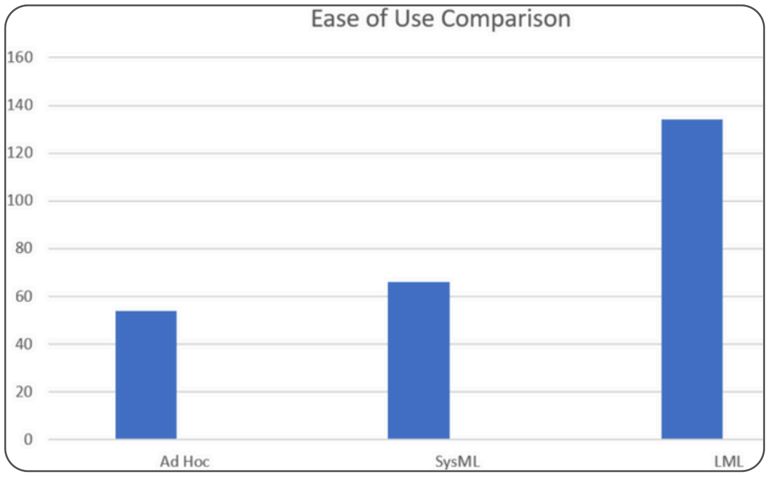
ROI Analysis of Performing Systems Engineering

Since every project is different, we often have to create a "basis of estimate" (BOE) based on "engineering judgment." Having performed this analysis for decades, I feel fully qualified to do this for systems engineering. I have been involved in over 100 proposals on the topic, where we had to create the "Tier 3" Work Breakdown Structure (WBS) cost estimate. The hierarchy chart essentially represents WBS. We only need to associate costs with it. Labor cost is usually expressed in the amount of time it would take to perform a specific task. The BOE includes both the time and the rationale for the time. Also, since we are in many ways going to have to compare "apples and oranges" to assess these tasks, we don't want to do what we would do for a proposal, which would be to associate a specific labor category, which had a certain cost per hour then allowing the pricing people to come up with a dollar number. That would only be a point solution. Instead, I created an "Ease of Use" score from 0 to 5, with 5 being easiest, 1 being hardest, and 0 meaning they can't do it at all. So, we will use the times estimated for each method for each task to assign this relative score. The table below shows a few examples of this analysis.



Number	Name	Description	specified by Name	specified by Number	specified by Description	specified by Value	specified by Units	specified by Ease of Use Score
	SE Tasks							
1.1	Requirements							
1.1.1	Import Documents	Import documents from PDF and CSV formats.	LML Import Documents	EoU.3.1.1	Innoslate has both a Word and CSV import capabilities. Some manipulation may be required to put it into a proper form. Time assumes little work on document needed.	0.5	Hours	4
			SysML Import Documents	EoU.2.1.1	Since DOORS is the preferred requirements tool, some work may have to be performed on the document to bring it in. Time assumes little work on document needed. Will have to set up hierarchy manually if numbering scheme is in original.	1	Hours	3
			Ad Hoc Import Documents	EoU.1.1.1	Converting a document from PDF or CSV to begin manipulating in the database. For the MS Office tools this is something fairly trivial. You can save the PDF as a Word file. If it's a CSV, you can immediately import it into Excel or Access.	0	Hours	5
1.1.2	Analyze Requirements	Separate requirements from contextual statements. Assess the quality of the requirements (clear, complete, consistent, etc.).	LML Analyze Requirements	EoU.3.1.2	NLP Quality Checker in Innoslate analyzes each requirement against 6 of 8 quality factors. Time is per requirement.	0.1	Hours	5
			SysML Analyze Requirements	EoU.2.1.2	Manual analysis	0.5	Hours	1
			Ad Hoc Analyze Requirements	EoU.1.1.2	Manual analysis	0.5	Hours	1
1.1.3	Manage Requirements	Create a workflow for managing the requirements from draft to archived.	LML Manage Requirements	EoU.3.1.3	Using Innoslate's Workflow tool automates this entire process. Time is for setting up the workflow and per requirement.	0.1	Hours	5
			SysML Manage Requirements	EoU.2.1.3	Older versions of DOORS do not have a workflow capability and it must be done manually. DOORS Next Generation does have a workflow capability. Time is per requirement.	0.5	Hours	1
			Ad Hoc Manage Requirements	EoU.1.1.3	Manual tracking. Time is per requirement.	0.5	Hours	1
					Innoslate's Traceability Matrix has NLP Assist. User just has			

When completing this analysis, we ended up with the overall results shown in the graph below. This chart shows that the current SysML-based toolset many people use does not enhance the ease of use significantly (54 vs. 66) from just using MS Office. Whereas the LML-based toolset (Innoslate) clearly provides a significant advantage over both. Note I am assuming that ease of use translates to cost savings, which seems reasonable since the easier it is to do a job, it follows that people will be able to do that job quicker. As we all know, in reality, people will spend the time and money they are given. But this means that LML will increase productivity by more than double the current approaches.



ROI From Startup Costs

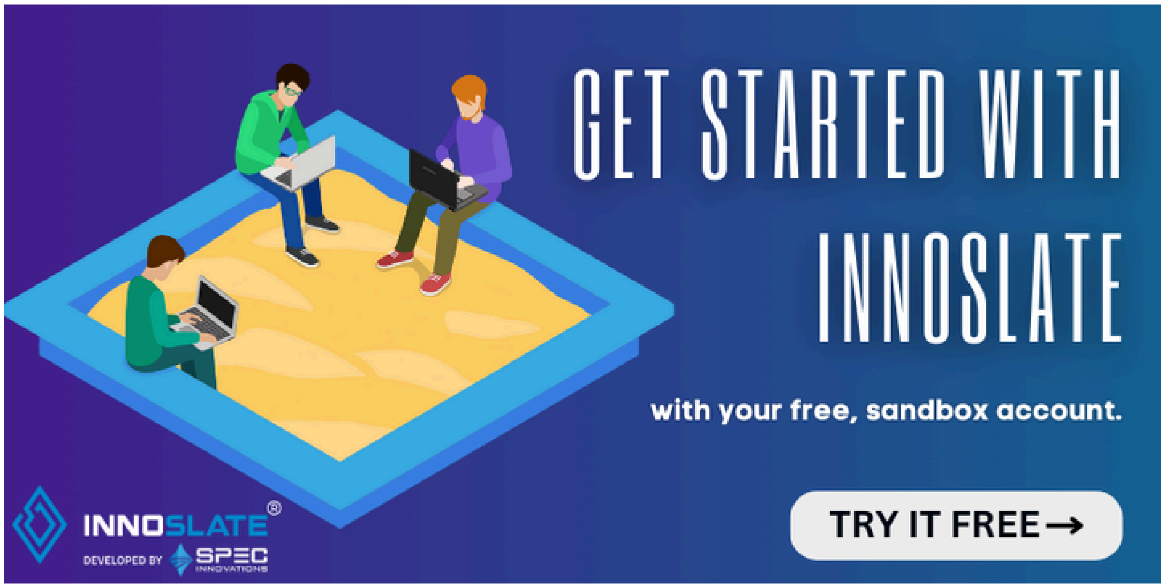
Now let's look at the startup costs. The table below shows the two sources of startup costs: the cost of the tools and the cost of the training. The tool costs shown are rough estimates and assume a single-user buy. In reality, the actual numbers are often less per user because they are procured in larger numbers and over longer periods, but these provide a relative difference. The $30,000 number was provided by a US Government customer. The training "costs" come from discussions with a number of people who have attended the training for SysML and LML. We assigned none to the MS Office since almost everyone knows how to use those tools; although you might want to take a few classes to get more proficient. So again, on a relative scale, the SysML Toolset requires roughly 60 to 100 times MS Office and 15 times more than Innoslate. By the way, the vast majority of Innoslate users never receive formal training. Users have told us that the online resources are sufficient for most of their needs.

Startup Costs Source	**MS Office**	**"SysML" Toolset**	**LML Toolset**
Tool Costs	$500 + upgrade costs	$30K + 18% per year	$2K per year
Training Costs	None	One 5-day training class	One 1-day training class or online resources

Bottom-Line

Now when we assess the ROI for the SysML vs. LML approach, it's clear that SysML is more expensive and thus has a negative ROI over just using the MS Office toolset. We think this analysis also demonstrates for a small additional startup cost, we can provide double the productivity, which means a 100% ROI.

Therefore, there can be value in MBSE if the right methodology is used. Unfortunately, many of the current advocates for MBSE have fixated on using the SysML approach. We think this analysis demonstrates how wrong that assumption is. So, if you want to provide meaningful value to your project you need to use LML and Innoslate.


GET STARTED WITH INNOSLATE
with your free, sandbox account.
TRY IT FREE →
INNOSLATE®
DEVELOPED BY SPEC INNOVATIONS