American Journal of Engineering and Technology Management
2019; 4(2): 47-56
http://www.sciencepublishinggroup.com/j/ajetm
doi: 10.11648/j.ajetm.20190402.12
ISSN: 2575-1948 (Print); ISSN: 2575-1441 (Online)

A Real Options Analysis of Spacecraft Software Product Line Architectures

Joseph R. Laracy[1, 2, *], **Thomas Marlowe**[1]

[1]Department of Mathematics and Computer Science, Seton Hall University, South Orange, USA
[2]Department of Systematic Theology, Seton Hall University, South Orange, USA

Email address:
joseph.laracy@shu.edu (J. R. Laracy), thomas.marlowe@shu.edu (T. Marlowe)
[*]Corresponding author

To cite this article:
Joseph R. Laracy, Thomas Marlowe. A Real Options Analysis of Spacecraft Software Product Line Architectures. *American Journal of Engineering and Technology Management.* Vol. 4, No. 2, 2019, pp. 47-56. doi: 10.11648/j.ajetm.20190402.12

Received: April 25, 2019; **Accepted**: May 29, 2019; **Published**: June 12, 2019

Abstract: Software and systems engineering for aerospace platforms presents many unique challenges. The decision if, and how, to employ software product line architectures is one recurring question. Real options analysis—applying option valuation techniques to budgeting decisions—can be a powerful tool for engineering managers, project leaders, and mission directors. In this paper, we demonstrate a real options valuation approach to explore this question.

Keywords: Real Options, Software Product Lines, Software Architecture, Space Systems

1. Introduction

On March 26, 2019 NASA Administrator Jim Bridenstine and Vice President Mike Pence came together for the fifth meeting of the National Space Council. The Vice President presented the Administration's vision for space exploration and spoke to NASA's progress on key elements to accomplish the President's Space Policy Directives. Commenting on the meeting, Administrator Bridenstine stated,

Among the many topics discussed during our meeting at the U.S. Space and Rocket Center in Huntsville, Alabama, was to accelerate our return to the Moon:

i. NASA is charged to get American astronauts to the Moon in the next five years.
ii. We are tasked with landing on the Moon's South Pole by 2024.
iii. Stay on schedule for flying Exploration Mission-1 with Orion on the Space Launch System (SLS) rocket next year, and for sending the first crewed mission to the lunar vicinity by 2022.
iv. NASA will continue to 'use all means necessary' to ensure mission success in moving us forward to the Moon [1].

NASA leadership, as well as most managers and engineers involved in the aerospace industry, are well aware that one of the major challenges for creating safety- and mission-critical space systems is the development of spacecraft software. Despite the high quality of design in the areas of electrical, mechanical, and materials engineering, a failure of control software can lead to a quick and expensive mission failure [2]. Software bugs can manifest in many ways, and trace back to different underlying hazards. However, it is very common for software safety hazards to arise from software process errors. There may be errors in specification, translation to design, coding, inadequate verification and validation. Additionally, problems may arise related to maintenance and modification, often a result of the accumulation of technical debt—the failure to restore code and artifact quality, consistency, and global structure after local, often rushed or kludgy, fixes to and extensions of the system [3]. The necessity of employing up-to-date, reliable software engineering methods is most evident.

In general, American spacecraft have utilized custom software developed "from scratch." Early attempts to employ a product line approach were generally not ultimately pursued. A product line in this case is a fundamental architectural design that may be instantiated in a variety of different applications to suit mission needs. Of course, a *software* product line also implies a *spacecraft* product line because software is primarily a control mechanism. This approach further entails mutually aware evolution of the two product lines. Not only must the software change as the spacecraft design changes, but changes in the design (and construction) of the spacecraft need to take account of both the hardware (e.g., sensor-computing-actuator, I/O, storage, and communication) used by the software and the control logic and other functionality of the software application.

2. Software Product Lines

The Carnegie Mellon Software Engineering Institute (SEI) defines a software product line as "a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way." [4] SEI suggests that some of the benefits of product line architectures include improved productivity, increased quality, decreased cost, decreased labor needs, and decreased time to market. Software product lines may be distinguished from other efforts at software reuse in that the reuse is *predictive* rather than opportunistic. Instead of simply stockpiling generic software components in a database in the hope for future reuse, product lines indicate creation of software components only when reuse is planned in one or more products in a well-specified product line [5]. Moreover, the software design, validation and verification (including testing), and refactoring are focused on "points of protected variation," characterized by one or more of the following: high likelihood of non-trivial change, high criticality, high risk (likelihood or consequences) of failure, possibly varying interaction with external systems, or product-line-specific problems such as tight timing constraints [6, 7]. Given the inherent risks of patching flight software bugs in deployed space systems, the highest standards of software and system quality assurance engineering is necessary.

Significant advances in software engineering have taken place over the last twenty years. A noteworthy example is the research by Kathryn Anne Weiss. While a member of MIT's Complex Systems Research Laboratory, Weiss developed a novel approach to spacecraft software engineering. Her approach focuses on incorporating a product line approach to software development as well as a software architecture-centric design process to support that approach. Weiss created a systems engineering-based development, evaluation, and selection process for the construction of the software product line architecture: Multi-Attribute Software Architecture Trade Analysis (MASATA). MASATA helps to ensure that engineers achieve their desired quality attributes, e.g., "analyzability with respect to safety, ease of verification and validation, sustainability, affordability, buildability, ability to meet real-time requirements and constraints, and 'monitor'-ability." [8]

It is important to point out three contrasts in software vs hardware development:

i. Software requirements can be refined during design and even later, although there are challenges with some extra-functional requirements including security, temporal constraints, and reliability. In fact, this is a major characteristic of agile software development. On the other hand, changes in the design of spacecraft, or of a plant to manufacture spacecraft, are both expensive and time-consuming, and later changes may be impossible or prohibitive in terms of resources.
ii. Software replication is almost free, and after a change, all instances of a software product can be updated even on-site with minimal difficulty other than coordinating with ongoing processing or differing external interfaces.
iii. The validation and verification of a software artifact, although necessarily ongoing, can occur on a single copy, after which all that needs to be checked for another instance is the fidelity of the copy, the correctness and integrity of its external connections, and the security of its environment.

3. Options

In light of the opportunities offered by MASATA and other software product line engineering developments, in this research we explicate an approach to apply *real options analysis* and compare product line architectures to the more common non-product line development projects. Flexibility can have great value. However, even if a product line architecture is well instantiated for a particular spacecraft and mission, it still may not be preferable to custom software for every spacecraft, or a hybrid in which some components are replaced or extended by specialized custom components, such as additional mission-appropriate sensors. In some scenarios, real options analysis has the potential to identify a winner, or similarly disqualify a loser.

In the field of finance, an *option* is right, but not an obligation, to take an action now, or for a period into the future, for a predetermined price (strike price, K = exercise price). A premium must be paid to acquire the option. Options provide a formal way to define flexibility and yield asymmetric returns, e.g., all gain, no pain. They can also be nested or simultaneous. Option value asymptotically approaches immediate payoff for increasing stock price, S. Option value also increases with volatility as well as increasing time to expiration. There are both *call* and *put* options. A call is right to take advantage of an opportunity, i.e., buy an asset for a set price. The payoff is Max [0, $S^*–K$], and so increases with asset price increases. Conversely, a put is a right to limit losses in a bad situation, i.e., sell an asset for a set price. The payoff is Max [0, $K–S^*$], so it increases with asset price decreases [9].

Engineering projects often contain option-like flexibilities, involving "rights, not obligations" and providing asymmetric returns, i.e., they are exercised only if advantageous. Hence, we say that these flexibilities are "real" options. Traditional net present value (NPV) analysis has ignored these options because it assumes decisions are not possible or are pre-determined, and also disregards the effect of intelligent management. Real options analysis includes valuation of flexibility by applying some form of options analysis that systematically increases value of projects, especially for engineering projects with great uncertainties, e.g., research and development projects, employing new technologies, etc. Richard de Neufville of MIT has done substantial work developing this field [10].

4. Salient Uncertainties

Now that spacecraft software product lines can be well executed, should they be done for a particular project or program? If so, to what degree should they be used? Exclusively? Perhaps real options can provide some answers.

To begin, we identify some salient uncertainties:

4.1. NASA Funding

Federal spending on non-military space operations is a huge uncertainty. The budget history (and therefore mission) of NASA has varied tremendously since its inception. The implications of funding are tremendous. Funding dictates both mission goals and implementation. Both very low and very high funding might even lead policy makers not to pursue software product lines.

4.2. Impact of Advances in Computer and Aerospace Engineering That Will Affect Spacecraft System Capabilities

The effect of technological advances could range from insignificant to non-trivial. Specifically, if software engineering techniques and computational power do not increase significantly, pursuing a product line architecture to be used for the next thirty years might be a good idea. However, if technology makes large strides, the effect of committing to a product line could constrain future designs. "Moore's Law" data, data on the utilization of increased computational power, the pace of software engineering advances, and the pace of aerospace design advances can clarify this uncertainty.

As another consideration, the possible long-term future development of reliable and economically feasible quantum computing would most likely suggest complete re-examination and possibly re-engineering of the entire product line. However, the uncertainties in the effect and the timeframe are large enough that we have not included this possibility in our current analysis.

Complementarily, there have recently been investigations of the applicability of suitably modified and adapted agile techniques in hardware manufacture, in domains including automotive component and aeronautical product design and manufacture [11, 12]. Process developments that promote efficient and effective design, modification, and evolution of aerospace hardware will make development of spacecraft product lines more attractive, especially if these changes also improve, simplify, or facilitate testing of those products. Likewise, further improvements in software processes, especially for control and real-time systems, will tend to advantage such coordinated product lines.

4.3. Involvement of Private Industry and Foreign Competitors in Space Activities

For economic or national security reasons, the presence of private industry, e.g., SpaceX and Blue Origin, or foreign governments, e.g., China and Russia, in space can radically affect US space policy and therefore systems. NASA's mission will certainly change as other players continue to enter the scene. For example, NASA's historic role of launching satellites might be largely outsourced to private industry. However, NASA might also need to develop a new capability to develop a Moon base before the Chinese arrive there and set up their own. These considerations depend on the material, intellectual, and economic requirements to achieve particular space operations. Conversely, the United States may at some point decide to participate in one or more new international ventures, adding interoperability and interface constraints as well as new missions and objectives for both spacecraft and software.

5. Decision Analysis

In this paper, we analyze three different system designs. One is a traditional, inflexible architecture and the other two incorporate various levels of flexibility by using a product line approach. Software product lines provide great opportunity for reuse because they exploit the presence of shared requirements, architectural design, components, modeling and analysis, testing, and process. The principal activity associated with creating a product line architecture is to identify variation points and develop a design that supports them. Identifying variation points is a non-trivial process because the software architects must decide both what features of the architecture will be common to all instances of the product family as well as how much detail should be provided given the inherent tradeoff of rapid instantiation and deployment versus high customizability.

Traditional Software Architecture—This design is the historic approach used by NASA to develop spacecraft software. Great effort goes into making the software safe and reliable, but little to no effort goes into creating a design that could be partially reused in the future for a different mission. With the traditional approach, consideration of a new mission includes a full analysis for its software suite, and committing to each new mission may require near total redesign and fresh implementation of the suite. Thus its long-term cost is affected by the diversity of missions under consideration and the number of mission types that reach planning stage or implementation.

Moderate Product Line Architecture—This design will be the first flexible approach. It is a compromise between flexibility and rigidity. For the purpose of this research, we define the following generic modules:

a) Data manager.
b) Task manager.
c) Telemetry manager.
d) Command manager.
e) Memory manager.
f) File System.
g) Instrument manager.
h) Attitude determination and control (ADCS).
i) Propulsion.
j) System manager.

Extreme Product Line Architecture—This design is the most flexible, but directly supports the least functionality. More work must be done to instantiate it into an executable architecture, but it makes very few assumptions that would provide problems in even radically different future missions. Here we provide the following generic modules:

a) Task manager.
b) Telemetry manager.
c) ADCS.

Part of the intent is to support plug-and-play modules, either in a software and physical component repository, or provided through other sources, particularly for sensors and actuators for exploration of planets and minor bodies or their neighborhoods. It is thus also more compatible with international efforts, or collaborations between NASA and foreign or private space ventures, with the need for different software interfaces for new sorts of new instruments, or for radically different implementations of physical components incompatible with the former interfaces.

In the coming decades, it is less likely (in the absence of major developments such as general-purpose quantum computing) that there will be opportunities for substantial change in the area of task scheduling, telemetry, and attitude determination and control. Scheduling algorithms and operating system design are relatively mature disciplines within software engineering and are not hindrances on current missions; changes may affect design or implementation of those components, but most likely not functionality. The telemetry system in use now will hopefully still exist in thirty years because of the infrastructure that has been deployed on the Earth and in the satellite network. It does not provide excellent bandwidth, but it is usually sufficient. The mathematics currently employed for computing attitude determination and control responses is also mature and will likely be instantiated in a similar way in the coming decades.

Below is a decision analysis tree. It looks at two periods of uncertain funding for the three architectural approaches utilizing subjective probabilities. Like all decision trees, it enjoys the benefit of specificity—each possible scenario is represented by a clear fork and node so that all possible solutions clearly in a single view. A decision tree, as illustrated here, can help decision makers understand better the role of probability in their project as well as the nature of the interaction of chance events and management decisions. On the other hand, the decision tree analysis here suffers from the common draw backs of decision trees such as instability—sensitivity to parameters, complexity—large trees are unwieldy and time-consuming to construct and analyze, and the reality of sometimes displaying too much information. Also, given the lack of data from which to compute "objective probabilities," decision trees are most often dependent on "subjective probabilities," i.e., expert judgment. These probabilities are usually based on the experts' domain knowledge and industry experience [13]. The tree below is primarily developed to show the benefit of the options analysis to follow. The authors' intent here is ultimately to show the utility of real options analysis to the overall engineering management problem and so one should not read too much into the particular subjective probabilities.

Outcomes in Figure 1 are in units of *capability per dollar of investment*. After carefully analyzing the tree, it is clear that the traditional design approach should be avoided. It is also clear that one should generally avoid scrapping the architecture and starting fresh because if subsequent funding is low, disaster will ensue. The moderate product line and extreme product line approach are very comparable with the current values in the tree. A small change in values would likely tip the optimal solution in one direction. Right now, the absolute highest expected value is found in the extreme product line. However, further analysis (expanding the tree or using other intellectual tools) is needed to better differentiate the two product line approaches. It seems that an optimal strategy includes pursuing a product line and making changes to it in the future, rather than abandoning it.

One missing and possibly unknowable set of parameters is the likelihood of collaboration, the extent of sharing of knowledge and of components, and the possible opportunity/risk tradeoffs entailed. However, the first two of these parameters largely derive from policy decisions, and can be incorporated if desired; the third is likely to become clearer in the near future. In general, the higher these parameters, the clearer are the benefits of the extreme approach.

Note: In Figure 1, modifying a product line architecture in both low and high funding environments leads to the same capability/$ because the additional money in the high funding situation is unnecessary to attain the maximum capability.

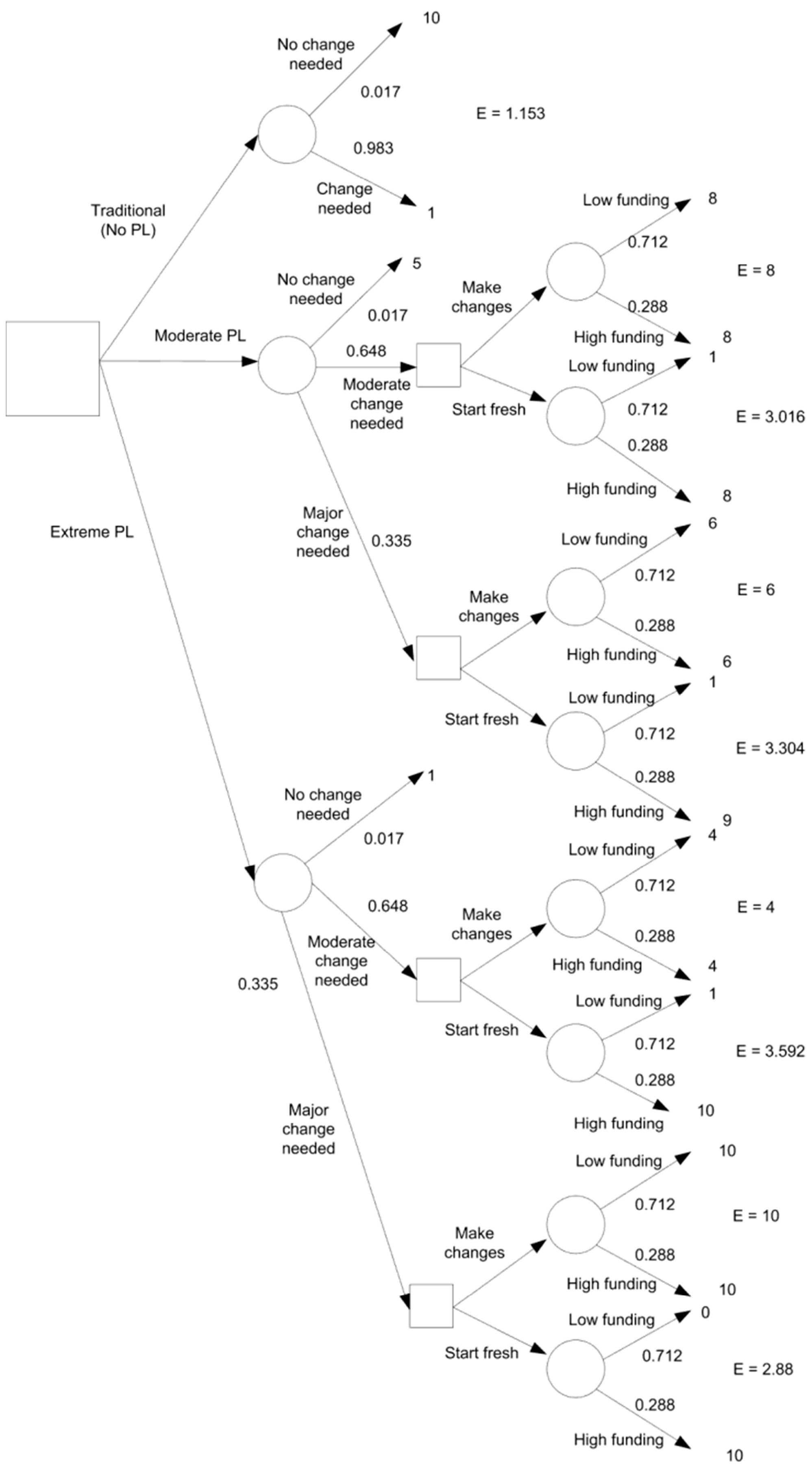

Figure 1. *Decision Analysis Tree.*

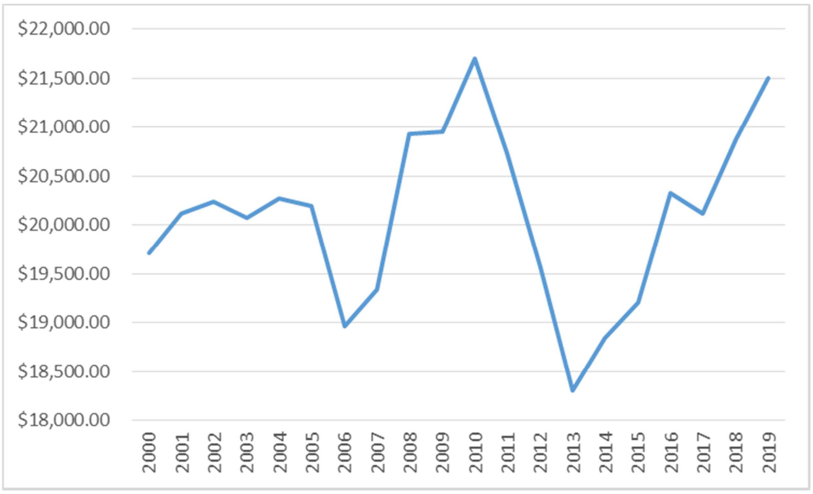

Figure 2. *NASA Budget in 2019 Millions of Dollars.*

6. Lattice Analysis

For the following analysis, we assume:

i. No major technological developments that would require complete redesign, and;

ii. Collaboration between space ventures largely limited to knowledge sharing and use of compatible hardware (and possibly software) interfaces.

One of the salient uncertainties identified in earlier analysis was the NASA budget. Software product line architectures are particularly valuable when requirement changes necessitate a new design, but the funding is not sufficient to build a new system from scratch. They provide a way to economically and safely instantiate a new spacecraft software system.

Figure 2 above plots the NASA budget from FY 2000 to 2019, according to the US Office of Management and Budget. Although there is likely diverse opinion on the suitability of selecting any particular time period, the authors believe that the volatility in budget since the turn of the century will likely continue in the coming years [14].

We now employ a binomial lattice analysis to explore the budget uncertainty. Binomial lattice analysis makes three fundamental assumptions. First, the evolution process is the same over time, i.e., it is stationary. Second, each state in the model leads to only two other states after one time period. Third, each later state is a multiple of an earlier state. State S evolves to $u \times S$ and $d \times S$ (by convention up > down).

A benefit of the lattice model is that it assumes path independence—states coincide. Paths that lead "up then down" arrive at the same state as paths that go "down then up."

$$d\,(uS) = udS \tag{1}$$

$$u\,(dS) = udS \tag{2}$$

Consequently, the model states increase linearly rather than exponentially. For example, after 24 months there are 25 states (parameterized by the total number of d phases), not ~17 million that one would have from traditional decision tree analysis! As a recombinatorial model, the binomial lattice eliminates the so-called "curse of dimensionality."

It should be noted that the binomial model does not allow shifts from positive to negative values. The lowest value is always positive. This is realistic, and in many situations necessary, e.g., the NASA budget will not go negative. With regard to the relationship between states, the relative value between a lower and the next higher is constant = u/d.

$$S \rightarrow uS \text{ and } dS \tag{3}$$

Hence the ratio of:

$$uS/dS = u/d \tag{4}$$

In our model:

i. S is the 2019 NASA budget.

ii. v is estimated annual growth, expressed as a percentage.

iii. σ is the standard deviation of the annual budgets, expressed as a percentage.

There are two conditions that must be met:

i. The average increase over a period is.

$$v\Delta T = p \ln (u) + (1 - p) \ln (\mathrm{d})$$

ii. The variance of the distribution is the sum of weighted squares of the observations.

$$\sigma^2\,\Delta T = p\,[\ln (u)]\,2 + (1 - p)\,[\ln (d)]\,2 - \{p\,[\ln (u)] + (1 - p)\,[\ln (d)]\}^2$$

At this point there are 2 equations and 3 unknowns (u, d, p). If we set $\ln (u) = -\ln (d)$ then $u = 1/d$.

We then solve the previous equations and obtain:

$$u = e^{(\sigma\sqrt{\Delta t})} \tag{5}$$

$$d = e^{(-\sigma\sqrt{\Delta t})} \tag{6}$$

$$p = 0.5 + 0.5\,(\nu/\sigma)\sqrt{\Delta t} \tag{7}$$

The calculated values can then be used directly in the model. The assumption behind these calculations is that actual probability distribution function (PDF) has a Gaussian aspect to it. Recall that the value of an option is the increase in expected value due to flexibility.

We then proceed with our lattice analysis using the following parameters:

Table 1. *Lattice Parameters.*

S =	$21,500.00		u =	1.04219265
v =	2%		d =	0.9595155
σ =	888.53	4%	p =	0.74197369
Δt =	1			

Applying the probability model to the outcome model leads to a probability distribution on outcomes. Figure 3 shows the probability density function resulting from the lattice analysis.

7. Options Valuation

There are many ways to analyze the uncertainty associated with the NASA budget. Static analysis is the simplest, although often a questionable technique. One may wish to assume that because the budget goes up and down, it may be prudent to model it as a constant, e.g., over the next six years.

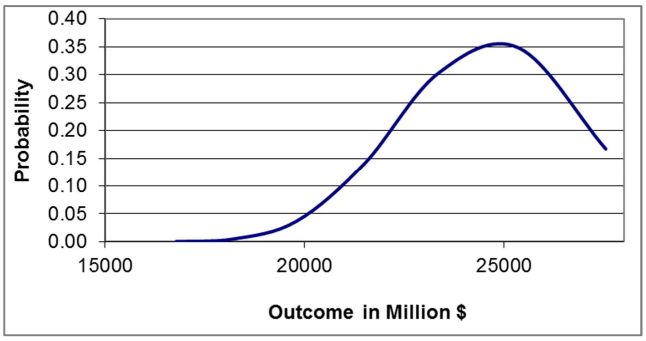

Figure 3. *Probability Density Function for the Binomial Lattice.*

Table 2. *Constant Budget Chart.*

Year	0	1	2	3	4	5	6
Budget (Mil $)	21,500.00	21,500.00	21,500.00	21,500.00	21,500.00	21,500.00	21,500.00
PV (Mil $)	21,500.00	19,196.43	17,139.67	15,303.28	13,663.64	12,199.68	10,892.57
NPV (Mil $)	109,895.26						

NB: Discount Rate = 12%.

Table 3. *1% Annual Increase Budget Chart.*

Year	0	1	2	3	4	5	6
Budget (Mil $)	21,500.00	21,715.00	21,932.15	22,151.47	22,372.99	22,596.72	22,822.68
PV (Mil $)	21,500.00	19,388.39	17,484.18	15,766.98	14,218.44	12,821.98	11,562.68
NPV (Mil $)	112,742.65						

Table 4. *Binomial Lattice Chart Acknowledging Uncertainty.*

Year	0	1	2	3	4	5	6
Budget (Mil $)	21,500.00	21,948.49	22,406.33	22,873.72	23,350.86	23,837.95	24,335.20
PV (Mil $)	21,500.00	19,596.86	17,862.19	16,281.06	14,839.89	13,526.29	12,328.97
NPV (Mil $)	115,935.26						

A better approach may be to assume a 1% annual increase because that is the average annual change from 2000 until 2019. Due to the visual constraints of the page, we apply the analysis for six years in the future. The result is shown in Table 3. Finally, one may wish to use a binomial lattice to take into account the range of uncertainty; results for this computation are shown in Table 4.

It is important to note the significant difference in total budget over the seven years. The budget recognizing uncertainty projects an additional $6,040,000,000 compared to the constant funding budget! Additionally, the budget assuming a 1% annual increase is $3,192,610,000 less than the budget recognizing uncertainty. A dynamic decision approach which permits the intelligent management opportunities provided by a real option analysis is superior because it better models the real world.

The budget lattice is presented in Table 5. As the table indicates, the budget in year six can range from $16.7 billion to $27.6 billion dollars. As time goes by, the probability that a change will be needed increases. The following numerical model accounts for this reality. We model this with an exponential decay equation. By year seven, there is almost a 50% chance that a change will be needed, as shown in Figure 4.

Table 5. *Outcome Lattice Chart in Millions of Dollars over Six Years.*

0	1	2	3	4	5	6
21,500.00	22,407.14	2,3352.56	24,337.86	25,364.74	26,434.95	27,550.31
	20,629.58	21,500.00	22,407.14	23,352.56	24,337.86	25,364.74
		19,794.41	20,629.58	21,500.00	22,407.14	23,352.56
			18,993.04	19,794.41	20,629.58	21,500.00
				18,224.11	18,993.04	19,794.41
					17,486.32	18,224.11
						16,778.40

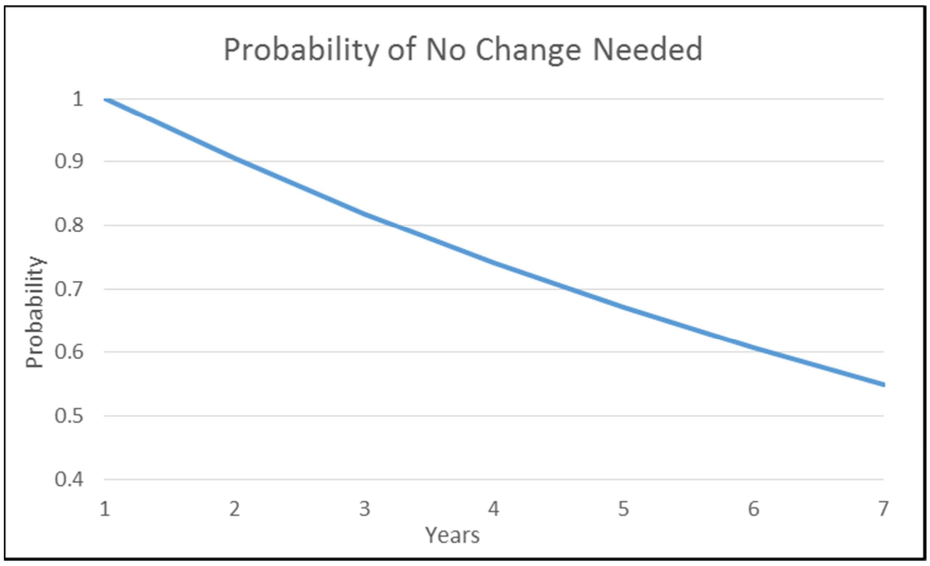

Figure 4. *Probability of No Change Needed Graph.*

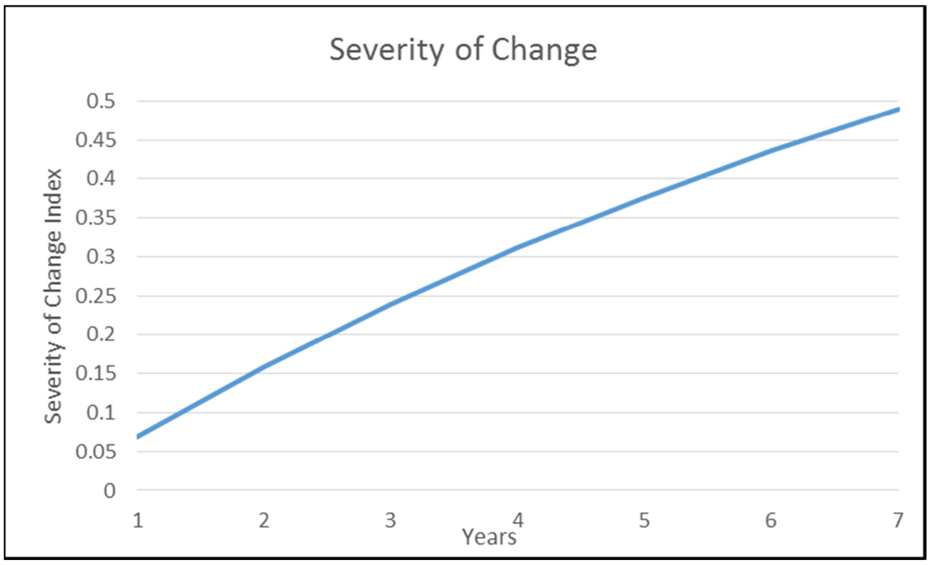

Figure 5. *Severity of Change Graph.*

In addition, as time goes by, the necessary changes will become more dramatic, as indicated in Figure 5. A non-product line architecture will need to be abandoned when the severity of the changes exceeds 0.5. The model assumes that "no change" takes a value of zero and a completely new system (great change) takes a value of 1.

After six years, the mission requirements may have changed substantially so that after a few more years, the original software architecture will likely need to be completely abandoned.

Utilizing a better approach, if the system designers invest the additional money to start with a product line architecture, they will have the right, but not the obligation, to change the functionality of the spacecraft software at comparatively low cost.

Table 6 is the lattice for an estimated future budget for spacecraft software only. It assumes that a constant portion, 3%, of each year's budget is devoted to flight software.

Table 6. *Spacecraft Software Budget Lattice Chart in Millions of Dollars over Six Years.*

0	1	2	3	4	5	6
645.00	672.21	700.58	730.14	760.94	793.05	826.51
	618.89	645.00	672.21	700.58	730.14	760.94
		593.83	618.89	645.00	672.21	700.58
			569.79	593.83	618.89	645.00
				546.72	569.79	593.83
					524.59	546.72
						503.35

Table 7. *Moderate Software Product Line Modified Lattice Chart in Millions of Dollars over Six Years.*

0	1	2	3	4	5	6
1161	1142.764	154.1269	109.5204	68.48481	39.65242	22.31575
	618.8875	141.9000	100.8321	63.05191	36.5068	20.54544
		130.6431	92.83312	58.05000	33.61071	18.91557
			85.46867	53.44489	30.94437	17.41500
				49.20511	28.48956	16.03347
					26.22948	14.76153
						13.59050

By computing the expected value at each stage and summing the results, we find that without a product line, the seven years of flight software development will cost NASA $ 4,807,576,098. This model assumes that the original architecture cannot be changed. Every year, modules are replaced, with every change destabilizing the system. It is only a matter of time for the architecture is inadequate and all the modules must be discarded.

In contrast, Table 7 above is not a "pure" lattice. The first node is 1.8 times greater than the original first node because it has been empirically observed that costs at year zero for such a product line approach would be that much higher. Similarly, the node in year one directly to its right is 1.7 times greater than the original. By year two the option is exercised (the mission requirements and spacecraft hardware have changed before the system is completed and the developers get the updated requirements document – this is often the case). Instead of paying full cost for the modification, earlier code is reused, and the cost is only 22% of the original. The assumption is made to exercise the option at this time because the probability of change model predicts approximately a 20% chance of a serious change and the severity model predicts a severity of about 0.20. An informal decision rule states that this scenario (two 0.20s) calls for a software change. This change is realized by utilizing the opportunities provided in the product line. Continuing the analysis, in year three, only 15% of the predicted amount is necessary, and so on. Instead of constantly building and throwing away software modules with the strategy defined by the table above, with the eventuality of throwing out everything in year six, the product line offers equivalent capability for significantly less development cost. The "moderate product line" is employed here.

Table 7 clearly shows that the product line requires a very large initial investment in the first two years (years zero and one). However, starting in year two, using agile methods, modules are able to be changed at very little cost. Even with major changes, the use of techniques such as PPV-aware design and refactoring helps keep the software artifact configuration clean, comprehensible, and easily modifiable, reducing complication and eliminating or at least deferring the need to fully reengineer the suite. As requirements evolve toward the "final" versions, the design is able to constantly change and "converge" on a future, optimal software architecture because the architecture is carefully maintained. The total cost of the moderate product line approach is $ 2,538,919,478. This is a savings of $ 2,268,656.62 compared to the non-product line approach. (This does not include any savings in the spacecraft product line if agile techniques are introduced there and coordinated with software product line development.)

8. Conclusion

In our example, real options analysis supports the use of the moderate product line architecture as the optimal solution to the spacecraft software problem. When carefully parameterized, engineering management decisions between "moderate" or "extreme" product line approaches may be further explored. Options analysis provides the needed flexibility to address the uncertainties associated with NASA funding.

However, there are three factors that would argue for use of the extreme product line architecture:

i. High likelihood of collaboration with private, foreign, or international space ventures;
ii. High likelihood of multiple and radically different sets of missions; or;
iii. The possibility of major technological developments in physical components resulting in radically different interfaces to those components.

We will explore the consequences of these, and the resulting tradeoffs between the moderate and the extreme models, in further research. The introduction of agile techniques in the spacecraft product line, or other coordinated interaction between the two development processes, is also a potentially significant research topic, to be explored in collaboration with others in the future. The flexible approach to design and the valuation of options is without a doubt a very valuable tool in assessing the suitability of spacecraft software designs.

References

[1] Jim Bridenstine, "NASA Administrator Statement on Return to Moon in Next Five Years," Text, NASA, last modified March 26, 2019, accessed March 27, 2019, http://www.nasa.gov/press-release/nasa-administrator-statement-on-return-to-moon-in-next-five-years.

[2] For example, see JPL Special Review Board, Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions, March 22, 2000, accessed March 25, 2019, https://spaceflight.nasa.gov/spacenews/releases/2000/mpl/mpl_report_1.pdf.

[3] Israel Gat and Christof Ebert, "Technical Debt as a Meaningful Metaphor for Code Quality," IEEE Software 29, no. 6 (November 2012): 18–21.

[4] "Software Product Lines Collection," Carnegie Mellon Software Engineering Institute, accessed March 25, 2019, https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=513819.

[5] Charles W. Krueger, "Introduction to the Emerging Practice of Software Product Line Development," Methods and Tools, last modified Fall 2006, accessed March 25, 2019, http://www.methodsandtools.com/archive/archive.php?id=45.

[6] Martin Fowler, with Kurt Beck, Refactoring: Improving the Design of Existing Code, 2nd edition, Addison Wesley Signature Series (Boston, MA: Addison Wesley, 2019).

[7] Craig Larman, Applying UML and Patterns: Introduction to Object-Oriented Analysis & Design & Iterative Development, 3rd edition (Upper Saddle River, NJ: Prentice Hall, 2005).

[8] Kathryn Anne Weiss, "Incorporating Modern Development and Evaluation Techniques into the Creation of Large-Scale, Spacecraft Control Software" (PhD, Massachusetts Institute of Technology, 2006), 3, accessed March 25, 2019, http://dspace.mit.edu/handle/1721.1/35591.

[9] For more information, see Stephen Figlewski, William Silber, and Marti Subrahmanyam, Financial Options: From Theory to Practice (New York: McGraw-Hill, 1992).

[10] See Richard de Neufville, "Real Options: Dealing with Uncertainty in Systems Planning and Design," Integrated Assessment 4, no. 1 (2003): 26–34; Richard de Neufville and Stefan Scholtes, Flexibility in Engineering Design, Engineering Systems (Cambridge, MA: The MIT Press, 2011).

[11] David Socha, Tyler C. Folsom, and Joe Justice, "Applying Agile Software Principles and Practices for Fast Automotive Development," Proceedings of the FISITA 2012 World Automotive Congress, Lecture Notes in Electrical Engineering, vol. 196 (Berlin: Spring): 1033–1045.

[12] Jörgen Furuhjelm, Johan Segertoft, Joe Justice, and J. J. Sutherland, "Owning the Sky with Agile: Building a Jet Fighter Faster, Cheaper, Better with Scrum," Global Scrum Gathering, San Diego, CA 2017.

[13] For more information on decision trees, see Donald Dibra, Project Valuation and Decision Making under Risk and Uncertainty Applying Decision Tree Analysis and Monte Carlo Simulation (Norderstedt, Germany: Books on Demand, 2015).

[14] For context, the costs of the Apollo spacecraft and Saturn rockets came to about $107.43 billion in 2019 dollars. See John Noble Wilford, We Reach the Moon: The New York Times Story of Man's Greatest Adventure (New York: Bantam Books, 1969): 67.