Dealing with Cost Estimation in Software Product Lines: Experiences and Future Directions

Andy J. Nolan[1] and Silvia Abrahão[2]

[1] Rolls-Royce
SIN C-3, Rolls-Royce plc, PO Box 31
Derby DE24 8BJ, England
Andy.Nolan@Rolls-Royce.com
[2] ISSI Research Group, Department of Computer Science
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022, Valencia, Spain
sabrahao@dsic.upv.es

Abstract. After 5 years invested in developing accurate cost estimation tools, Rolls-Royce has learnt about the larger potential of the tools to shape many aspects of the business. A good estimation tool is a "model" of a project and is usually used to estimate cost and schedule, but it can also estimate and validate risks and opportunities. Estimation tools have unified engineering, project and business needs. The presence of good estimation tools has driven higher performance and stability in the business. It was evident we needed this capability to underpin decisions in our new Software Product Line strategy. The objective of this paper is twofold. First, we report the experiences gained in the past on the use of estimation tools. Second, we describe the current efforts and future directions on the development of an estimation tool for Software Product Lines. At the heart of the Product Line estimation tool is a simple representation of the product – represented as the number of Lines Of Code (LOC). The next generation of tool, will need to consider wider aspects of product quality in order to create more accurate estimates and support better decisions about our products.

Keywords: Cost Estimation, Software Product Lines, Industrial Experiences.

1 Introduction

The production of quality software, on time, and within budget, remains an open problem of Software Engineering that has been addressed from different approaches. An industrial approach to this problem is to use Software Product Lines (SPL). Several benefits are associated to the introduction of product lines in software development organizations such as cost reduction, time-to-market improvement, project risk reduction, and quality improvement.

However, the associated costs and the quality of the software products may greatly differ due to systematic reuse. In addition, product line engineering is often the more economical choice in the long-term run. It might not be the best choice when project managers want to amortize their core asset base across only a few products or across

J. Bosch and J. Lee (Eds.): SPLC 2010, LNCS 6287, pp. 121–135, 2010.

products with little commonality [1]. Therefore, there is a need for tools to help project managers to analyze in which situations and scenarios product line investment pays. To address this issue, several cost estimation models for Software Product Lines (SPL) have recently been proposed in the literature. However, to understand their benefits and weaknesses, it is important to analyze the experiences gathered in applying these models in industrial or organizational settings.

In this paper, we present an experience report about the use of cost estimation tools at Rolls-Royce. The objective of this paper is (i) to report the experiences gained in the past on the use of a cost estimation tool based on COCOMO (Constructive Cost Model) [3] (ii) to describe how this tool was extended for its use with software product lines as well as the lessons learned (iii) to describe future extensions for this tool based on the preliminary results obtained within the MULTIPLE (Multimodeling Approach for Quality-Aware Software Product Lines) project conducted at the Universidad Politécnica de Valencia in Spain with close collaboration of Rolls-Royce.

This paper is organized as follows. Section 2 discusses existing models and tools for cost estimation in SPL. Section 3 discusses past experiences on the use of a Cost Estimation tool at Rolls-Royce. Section 4 presents an overview of the SPL initiative launched in 2008 as well as the development of an estimation tool which was built for assessing the benefits of SPL. Section 5 describes the lessons learned and the future extensions of the tool. Section 6 presents our conclusions and further work.

2 Related Works

In the last few years several cost estimation models for software product lines have been proposed. Some representative proposals are: [16], [1], [19], [4], [7], [10], [9] and [13]. Poulin [16] proposed one of the first models for analyzing the effects of employing a systematic reuse approach. The model is based on two parameters: the relative cost of reuse (RCR) and the relative cost of writing for reuse (RCWR). The first parameter can be used for comparing the effort needed to reuse software without modification to the costs associated with developing the same software for a single use. The second parameter relates the costs of creating reusable software to the cost of creating one-time use software. These parameters can also be applied in the context of software product lines. The Poulin model uses the RCR and RCWR to calculate two other indicators (i.e., reuse cost avoidance and additional development cost) that predict savings for developing a specific project.

Böckle *et al.* [1] proposed a software product line cost model to calculate the costs and benefits that we can expect to have from various product line development situations. In particular seven reuse scenarios were identified. The cost model proposed involves the following four costs: (1) the cost to an organization of adopting the product line approach for its products; (2) the cost to develop a core asset base suited to support the product line being built; (3) the cost to develop unique software that is not based on a PL platform; (4) the cost to reuse core assets in a core asset base. The authors then analyze the cost savings of converting products to a software product line as they evolve over time.

Tomer *et al.* [19] proposed a model that enables software developers to systematically evaluate and compare alternative reuse scenarios. The model supports the clear

identification of the basic operations involved and associates a cost to each basic operation (e.g., adaptation for reuse, new for reuse, new development, cataloged asset acquisition). In 2004, Boehm *et al.* [4] proposed a software product line life cycle economics model called Constructive Product Line Investment Model (COPLIMO). The model facilitates the determination of the effects of various product line domain factors on the resulting PL returns on investment.

Since the previous cost estimation models do not properly consider the software quality cost, In *et al.* [9] proposed a quality-based product line life cycle estimation model called qCOPLIMO as an extension of the Boehm et al. model [4]. This model is based on the top of two existing models proposed as an extension of the COCOMO II model: COPLIMO, which provides a baseline cost estimation model of the SPL life cycle, and COQUALIMO which estimates the number of residual defects. The model provides a tool to estimate the effects of software quality cost for enabling cost-benefit analysis of SPL. However, quality is measured only as the cost per defect found after product release and the tool is not granular in terms of the product itself.

Clements *et al.* [7] proposed the Structured Intuitive Model for Product Line Economics (SIMPLE) model. Its purpose is to support the estimation of the costs and benefits in a product line development organization. The model suggest four basic cost functions to calculate (1) how much it costs an organization to adopt the PL approach for its products; (2) how much it costs to develop the core asset base to satisfy a given scope: (4) how much it costs to develop the unique parts of a product that are not based on assets in the core asset base; (4) how much it costs to build a product reusing core assets from a core asset base.

In [10] Lamine et *al.* introduce a new software cost estimation model for SPL called SoCoEMo-PLE. This model is based on two previous models: the integrated cost estimation model for reuse [11] and the Poulin's model [16]. The authors claim that when compared to the two costs models used, the proposed new model gives different results and presents more details because it takes into account more features of PLE development life cycle. However, no evidence for this claim was found.

Finally, other authors suggest that a decision analysis model could be integrated into the cost model to provide an interpretation to the values obtained by the cost functions. This is the case of the Nóbrega *et al.* [13] proposal where an Integrated Cost Model for Product Line Engineering (InCoME) is presented. The aim of this model is to perform investment analysis for a set of reuse scenarios to help an organization to decide if an investment in a product line is worthwhile. The model was applied in a small product line with 9 products, 10 core assets and two reuse scenarios. Although the results seem promising, the model should be applied to other organizations with larger PLs in order to test the generalizability of the results obtained.

An analysis of these cost estimation models revealed that the majority of them estimate costs and/or benefits of using SPL through several reuse scenarios (e.g., [1] [19] [13]). Other models identify a clear separation of cost estimation and investment analysis [13]. Some models consider variations in the cost of reuse [4] [9]. In general, the proposed models suggest several parametric values that must be accurately calibrated. Finally, the majority of the proposed models often not considered other factors such as quality and time-to-market.

3 Past Experiences on the Use of a Cost Estimation Tool

This section gives a brief overview of the software development and cost estimation practices at Rolls-Royce. This is important to understand as it was the foundation for building the Product Line Estimation tool shown in Section 4.

3.1 Rolls-Royce Control Systems

Rolls-Royce provides power systems and services for use on land, at sea and in the air, and operates in four global markets - civil aerospace, defense aerospace, marine and energy. In all the business sectors in which Rolls-Royce operates, there are demands for improved capability and effectiveness of the power systems, more economic and faster product development, better transition to operation (minimum post-delivery changes) and better in-service cost and availability, with commensurate reduction in cost of purchase and/or cost of ownership.

The Control Systems department of Rolls-Royce's Aerospace business is responsible for the Engine Electronic Controllers (EECs) for a range of small and large gas turbine engines, for the aerospace industry. The EEC contains a significant amount of software that is designed to 'control' the engine, as directed by the pilot, in a way that is safe for the engine, safe for the aircraft, fuel-efficient, component life efficient and environmentally efficient. We have been developing high integrity software for over 20 years and have extensive data on our processes and productivity. We have had some level of success with clone-and-own- reuse but this tended to be opportunistic from existing projects. Since 2008, we are developing our SPL for the business which has potential for both the software and hardware aspects of our engine design.

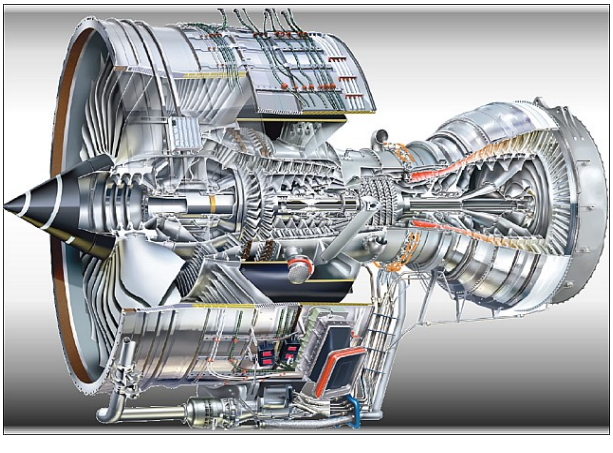

Fig. 1. Rolls-Royce Trent 900 engine used to power the A380 – the control software is in excess of 200,000 lines of code

3.2 The Adoption of COCOMO II

Since 2004, Control Systems has invested in developing reliable estimation tools to predict software development cost and schedule. The work was undertaken as part of a six-sigma Black Belt project to understand the factors that influenced good estimates. One of the outputs from that study was a calibrated estimation tool based on COCOMO II [2]. COCOMO is an algorithmic software cost estimation model developed by Boehm. The model uses a basic regression formula, with parameters that are derived from historical project data. COCOMO was first published in 1981 as a model for estimating effort, cost, and schedule for software projects.

COCOMO II is the latest extension to the original COCOMO and was developed by the combined efforts of USC-CSSE, ISR at UC Irvine, and the COCOMO II Project Affiliate Organizations. The revised cost estimation model reflects the changes in professional software development practice that have come about since the 1970s.

The process of evaluation and tool development took around 1 month of effort. The objective was to find a simple, accurate and believable estimation tool that would allow managers to express and defend the critical project assumptions in a way that the business could understand. Believable and dependable estimates were key requirements as well as having a tool that anyone could use.

It was necessary to find accurate data for historic projects and then to estimate their cost as if they were future projects. A "blind" estimate was generated and then validated against the actual project results. There was a "common cause" discrepancy which was down to tool calibration. In other cases, there were special cause exceptions which had to be investigated. At the end of the analysis, we had both a calibrated estimation tool as well as a thorough understanding of the COCOMO factors and how to drive them. This knowledge became part of the user guide and training program.

The COCOMO II model is a very simple equation relating factors to final cost and schedule. We added "front end" tools to help derive estimates for size (Lines of Code) as well as "back end" tools to help unwrap the results into resource profiles, phases, plans and even error predictions. The COCOMO model sat at the heart of an otherwise comprehensive resource/project planning tool.

We built many versions of the tool to meet the needs of different domains – including hardware development. In each case, we identified the "questions" we needed to ask about the project/business, selected the factors from COCOMO II that would address these questions, then built this into a tool. In those cases where COCOMO II could not provide the factor to address a question, we would develop our own factors, gather data from the business and perform a regression analysis to understand the sensitivity and range for the new factors. Examples of new factors included requirements volatility (from our customers) and Scrap & Rework generated from our evolutionary development approach to engine, hardware and electronic development.

3.3 What an Estimation Tool Teaches You

An important breakthrough occurred when we relished that the estimation tools, like COCOMO II, were not only useful for estimating costs, but were actually teaching us what was important about a project or product. Estimation tools are actually models of a project and like all models they are there to help you make good decisions. An estimation

tool defines a formal and objective representation of a project or business and is by definition a simplification of reality.

An estimation tool need only be as precise and accurate as required to make a meaningful and accurate decision. They are there to tell you something that you would not (or could not) know without them. They are central to good project management and control, estimation, improvement and risk management.

The estimation tools are also there to remove the subjectivity from the decision making process. It's tempting for a manager, fuelled by ego and heroics, eager to prove themselves, to exaggerate the truth or guess at key decisions. You also need a good estimation tool to help with reasoning, collaboration and negotiation in order to persuade the business to invest in the right things e.g., in a product line. We have found that a well-constructed estimate makes persuasion a whole lot easier then relying on good intentions and opinions.

3.4 The Business Benefits

Through the development and deployment of estimation tools across the business, we have seen an improvement in stability and productivity – on average around 11% cost saving per project. This is primarily because, estimation tools, like COCOMO II, are informing you of what is important. This information has shaped what we measure, how we identify and validate improvements, how we identify and mitigate risks and how we manage and estimate projects. They help us optimize and refine the business around objective reasoning rather the subjective guesswork i.e., we make better decisions.

For example, if a project is taking on novel features, or there are concerns over the aircraft maturity, we would expect a high level of requirements volatility and scrap & rework. The estimation tool would quantify the impact (increased cost and a longer program) and this would then be used to drive for changes in the development approach, risk mitigations, negotiations with the customer and so on. If this was a critical factor for success, then this attribute of the project would be carefully monitored and reported. Similarly, if in order to achieve a low cost project, you assume a high performance team, then this aspect of the project will need to be carefully monitored and reported. The output from an estimate is a measurement plan of critical factors that need to be monitored, controlled and where possible, improved.

4 The Estimation Tool for Software Product Lines

This section starts describing the software product line initiative launched at Rolls-Royce followed by the description of new estimation factors considered in the development of a cost estimation tool for SPL. The section ends by discussing lessons learned from practical experiences using the tool.

4.1 The Software Product Line Initiative

The challenges facing Rolls-Royce Control Systems are not unique; we have a program load greater than ever before. Our customers want faster development, so program timescales continue to decrease, while functionality increases and our shareholders demand lower costs and greater profitability. Each new project development

represents a significant engineering challenge, moving engineering from research to product development at a rapid pace. Even though the approach used today is competitive, our future order book growth means that we cannot sustain our current engineering approach. We need a step change in productivity. SPL will help us step up to these challenges.

In addition we have an extensive legacy portfolio that will require refresh as electronic equipment becomes obsolete. This will be an ongoing concern for any business, like Rolls-Royce, involved in long-life products, that includes electronics, and inevitably that use software solutions. It is this long-life that also has the potential for Product Line solutions to cost effectively refresh our legacy portfolio – but we have to accept that they are part of the 'future' in our initial Product Line market scope.

4.2 The Need for New Estimation Factors

When the SPL initiative was launched in 2008, the development of a reliable and comprehensive estimation tool was seen as critical to ensure we were making the right decisions. Based on some additional data from within Rolls-Royce, we developed the first version of the SPL Estimation tool [14] in under a day. Several versions later and about 4 weeks of prototypes and demos the first model was released and populated.

The original estimation tool was calibrated and developed around our existing processes and approach to software development. With the development of the product line, there were new questions we needed to ask, which led to the need for additional factors in the estimation tools.

- It needed to guide the business – the tool needs to help us persuade "the business" – it needs to be able to estimate business level cash flow and benefits and communicate trades and decisions to people who did not understand software.
- It needed to guide each project – each project should be able to use the estimation tool to make trade decisions between bespoke and Product Line assets.
- It needed to guide the architects – we need to pick the right features and the right variation mechanisms that add the greatest benefit.

In addition to these requirements, the tool needed to be able to answer the following:

- When to develop a Product line asset and when to rely on clone-and-own-reuse and bespoke development.
- The costs of development and deployment of assets for a range of variation mechanisms.
- To factor for organizational overheads and new roles not normally associated with traditional project development.
- The effect the Product Line Strategy has on the organization i.e., disruption and risk as well as the benefits from alignment of processes and objectives.
- The costs for redeveloping the Product Line architecture i.e. product line refresh.

We did not adopt COPLMO directly because it did not contain the granularity we needed or the "decision points" described above. The first estimation tool was developed and used in early 2009. The structure of the tool is shown below (see Fig. 3) and contains the following additional decision points:

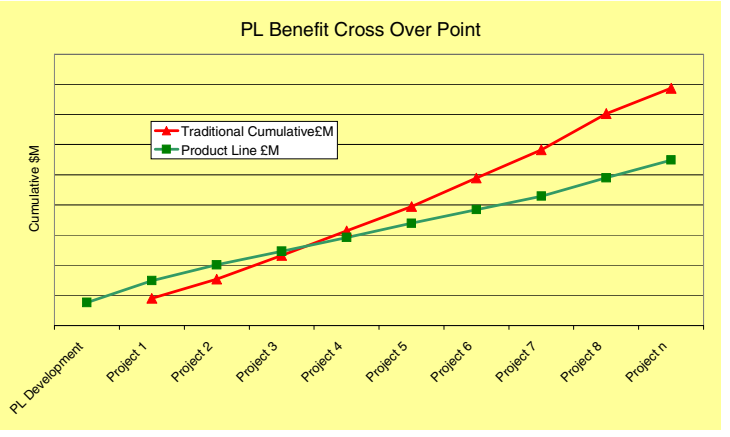

Fig. 2. The output from the Software Product Line estimation tool showing the break even point of the initiative. This information was used to persuade the business to invest in a strategic initiative rather then to focus on a project-by-project return on investment.

- Historically, size was measured in terms of the total (macro) number of lines of code. Step 1 of the SPL estimation tool provided the size of each individual asset based on a library of features from past projects.
- From historic analysis we have shown that approximately 70% of the software costs arise from 30% of assets i.e., not all assets are equal. This was not an issue for our historic estimation tools because we considered size at the macro level only. With the introduction of the SPL, we added Step 2 to understand the cost and value of each asset.
- A new step "P-Process Model" was added to model our safety-critical development process and to understand which processes are required to develop an asset and which processes are required to deploy an asset. We also recognized that the process used to develop and deploy an asset varies depending on the asset variation mechanism.
- Traditionally, features would only be considered at the time of project launch and clone-and-own approaches used to acquire assets from pat projects. Step 3 was used to map out the life of features across the future engine programs.
- Step 4 was introduced so that the SPL team could understand the costs to develop a project using the traditional methods and then compare and contrast these costs with the SPL development costs (see step 10).
- Step 5 was introduced to model the additional costs to develop a SPL asset. This information was taken from COCOMO II – the RUSE (developing for reuse) and DOCU (additional documentation) factors.
- Step 6 was used to map the deployment of SPL assets into projects to understand if there was a net benefit, per asset, when considering the development, deployment and maintenance of assets. The step also revealed those features that would be bespoke to each project.
- Step 7 was a mechanical process and generated all the information together to understand the new costs for developing a project based on SPL assets.

- Step 8 was added to model the organization costs. This was derived from the SIMPLE model [7] and consists of costs above-and-beyond traditional project development. The model contains the costing for, new organization roles and governance activities, new management activities, increased configuration management and change control, reference architecture developers, process & tool development, organizational training & orientation, consultancy costs and business level interface roles and activities.
- Step 9 was added to model the effect of introducing the SPL initiative into the business. With any change comes an initial "shock" followed by a settling in period. Also, the organization would look and behave differently. COCOMO II was again used to model the business environment.
- Step 10 performed the final analysis and compared and contrasted the benefits of the SPL with a more traditional project centric organization.

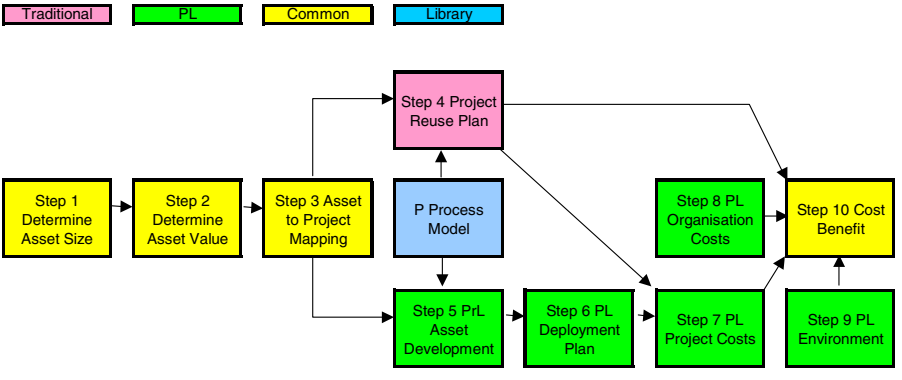

Fig. 3. The structure of the Software Product Line estimation tool

4.3 How We Use the Estimation Tool

The tool is used to perform the following activities:

- Communicate with business leaders in a language they understand. It can model the costs of investment and the point when we have return on our investments.
- Model decisions at the business, project and asset level. For example, through sequencing our projects, we can smooth out the SPL asset development program. At the asset level, the architect can make trades between variation mechanisms and net return on investment. At the project level, they can make trades between PL assets and bespoke features and use this information to either push back or negotiate with the customer.
- Perform risk analysis by understanding variation and sensitivities of decision points (modeled as factors in the estimation tool). For example, what is the effect of not having the capable development team, what is the impact of refactoring the architecture, and so on.

- Assess the return on investment of improvements. The team can perform improvement scenarios with the tool to understand which improvements give the greatest return on investment.

In terms of accuracy, after 5 years of using COCOMO II, we have an R^2 correlation of 0.98 between the predicted and actual costs i.e. at the business level, we have a accurate and normalized estimation tool. Although in reality, at the per project basis, estimates can be up to 20% in error (or higher if assumptions are wrong). We are gathering data on each asset to validate the estimation tool. As the tool was based on historic projects, adjusted for a change of process, we expect an equivalent level of accuracy.

4.4 Lessons Learned

The lessons learned from the use of the SPL estimation tool is as follows:

- The experiences at Rolls-Royce, both good and bad, could be expressed in simple meaningful terms as defined by the COCOMO II factors.
- The estimation tool could also be used to try "what if" scenarios, to elicit improvement opportunities and to validate improvement proposals.
- The tool taught the business what was important, what to manage, what to monitor, where the risks lay and where opportunities would come from.
- Despite initial reservations, the tool was calibrated and in use in only 1 month. The benefits have been on 10,000 times this effort.
- You need to have an owner who is passionate in estimation, to drive the approach into the business.
- Like any tool development, let it evolve – we had over 20 versions of the tool, each an enhancement to address new questions that came to light during use.
- We had to make a decision between developing a simple tool that anyone could use and a tool for experts – we opted for an experts tool making both development and deployment far easier and allowing greater freedom to add complex decision points into the tool designed around the use base.
- Never underestimate the drag as people would rather rely on subjectivity rather than objective reasoning but never underestimate the power of a well constructed, formulated and reasoned argument.
- The tool has been successful mainly because it can meet the needs of a wide range of users e.g., architects, project leaders and business leaders.

We also observed that future versions of the tool can be adopted to provide different view points for different business needs. At present, we can model cash flow but future tools can more accurately model schedule, asset availability, etc. Other views can be added to represent the need for key resources and resource planning.

5 Future Directions

The SPL estimation tool can answer many questions about the development environment and trades between architectural, project and business decisions, but contains only

an approximation of the product itself. At present, the product is represented only as lines of code, adjusted for complexity or difficulty. The next generation of estimation tools will need to consider, in greater detail, *quality attributes* about the assets we are developing. In this sense, we are working currently on extensions for this tool based on the preliminary results obtained within the MULTIPLE project conducted at the Universidad Politécnica de Valencia, in Spain, with close collaboration of Rolls Royce. In this section, we give a brief overview about this project and discuss how the results obtained on it can be transferred to Rolls-Royce.

5.1 The MULTIPLE Project

MULTIPLE (Multimodeling Approach for Quality-Aware Software Product Lines) is a three-year project (2010-2013) supported by the Spanish Ministry of Science and Innovation. Rolls-Royce participates as an EPO entity – declaring interest for assessing the benefits of the results derived from the project and possibly exploit them.

The objective of this project is to define and implement a technological framework for developing high-quality SPL. This framework is based on the existence of several models or system views (e.g., functionality, features, quality, cost) with relationships among them. This approach implies the parameterization of the software production process by means of a Multimodel which is able to capture the different views of the product and the relationships among them.

As part of this project, we developed a Quality Model for conducting the production plan of a SPL [12]. It is one of the views of the Multimodel and captures the quality attributes relevant for the domain, the quality attributes relevant to certain products of the family, as well as the variability among these attributes. The model allows measuring the properties of several artifacts of a SPL (e.g., core asset base, core assets, SPL architecture, product architecture) by providing quality metrics and trade-off analysis mechanisms in order to help architects to select core assets that meet the business need.

The next section describes our current efforts on using the quality attributes of the quality model as new parameters to the cost estimation tool.

5.2 Extension Mechanisms for the Rolls-Royce Cost Estimation Tool for SPL

Cost is usually an important factor in reuse-based development. However, other factors, such as product quality and time-to-market, are also expected to improve by reusing software assets. Lower error rates, higher maturity products and more complete functionality all contribute to improved customer satisfaction. Currently, we are extending the cost model for SPL with two additional factors: quality and variability.

5.2.1 Quality

To define the relevant quality attributes to be used as new parameters in the cost estimation model for SPL, we took the complete SQuaRE-based Quality Model developed for the MULTIPLE project [12] and discussed the relative importance of each quality attribute to the safety-critical embedded systems domain and SPL. Each quality attribute was classified according to the following scale: High (mandatory for this domain), Average (important but not mandatory) or Low (not important for this domain). An extract of the results is presented in Appendix A. In general, the results

show that the *Operability* quality characteristic and its associated quality attributes are not relevant for safety-critical embedded systems since it assess the degree to which the software product can be understood, learned, used and attractive to the user. The most relevant characteristics and quality attributes is as follows:

- *Functional suitability*: the degree to which the software product provides functions that meet stated and implied needs when the software is used under specified conditions. We need to develop assets that have met the appropriate certification standards, are functionally correct, are accurate and have deterministic behaviour.
- *Reliability*: the degree to which the software product can maintain a specified level of performance when used under specified conditions. Performance, in our case, can include fault tolerance, error recovery, coping with the loss of engine signals, hardware failure conditions and so on.
- *Performance efficiency*: the degree to which the software product provides appropriate performance, relative to the amount of resources used, under stated conditions. Performance is measured in terms as timing and memory utilisation as well as response times to stimulus from the aircraft or engine.
- *Compatibility & transferability*: The ability of two or more software components to exchange information and/or to perform their required functions while sharing the same hardware or software environment. We need to understand the level of compatibility with the electronic hardware standards, engine configurations, and airframe communication.
- *Maintainability*: the degree to which the software product can be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications. This issue is not limited to just safety-critical software but the effects of change can be disproportionately high in the safety-critical domain because of the need to capture the certification evidence.

These attributes have a major impact on the cost of a SPL for safety-critical embedded systems. In addition, we identified new attributes for *Safety* and *Affordability* that should be incorporated to the Quality Model. The attributes for safety are as follows: Predictability (the degree to which the software behaviour is predictable), Completeness (the degree of test coverage for behaviour), Compliance (the degree of compliance to industry safety standards), and Protection (degree of protection against unsafe conditions). The quality attributes for affordability includes the cost to develop, cost to maintain and cost to deploy in new situation/context.

Future works includes the definition of quality attribute scenarios following an approach similar to the one performed by the SEI for eliciting and representing quality attribute requirements observed in practice. In [15], a distribution of quality attribute concerns according to the SEI-led ATAM evaluations is presented. The top 3 out of 140 attributes are: Modifiability (concern = New/revised functionality/ components; distribution = 6.4%), Usability (concern = Operability; distribution = 4.1%) and Modifiability (concern = Upgrade/add hardware components; distribution = 3.9%).

5.2.2 Variability

Variability may add customer value but allowing too much variability could lead to substantial follow-up costs during the lifecycle. Therefore, variability should be added as a cost driver during the whole SPL lifecycle since each feature must be maintained integrated in subsequent releases, tested and possibly considered during deployment and customer support. In addition, we should also take into account the variability that may exist in the different products of the family with respect to the quality attributes. Therefore, two dimensions of variability should be considered: (1) for the whole SPL lifecycle and (2) with respect to the quality attributes. In the safety-critical domain, it will be necessary to prove the product is safe for all variants and combinations of variability. Too much variability could have an exponential impact on the verification, validation and certification activities.

5.2.3 Other Extension Mechanisms

New mechanisms should be defined and used in conjunction to the cost estimation model for SPL to provide relevant information about cost-benefits to the business. For instance, a mechanism to relate each business goal with the quality attributes. In addition, adding or removing functional features may influence the quality of the product family. Therefore, we need a new mechanism to relate each feature with respect to the quality attributes. There is also a need for other mechanisms to analyze the impacts among quality attributes and to identify potential conflicts among them.

6 Closing Remarks

Along with each generation of the estimation tool came new questions to be answered. Each new generation of the tool then added new factors (or factorization) to help answer those questions. As we refined our understanding and fidelity of management, we needed a higher fidelity of factors in the estimation tool.

The first generation of the estimation tools focused heavily on the development environment, being able to understand, accommodate, monitor, control and improve aspects of the development environment. The first generation estimation tools considered the software product as a single large entity and defined it in terms of lines of code. The second major generation of the tool was designed to answer new questions about the product line and to do this, we needed a refined understanding of the software architecture, its functional breakdown, variation mechanisms and the costs to develop and deploy assets.

The software Product Line estimation tool can answer many questions about the development environment and trades between architectural, project and business decisions, but contains only an approximation of the product itself. At present, the product is represented only as lines of code per feature, adjusted for complexity or difficulty. The next generation of estimation tools will need to consider, in greater detail, quality attributes about the assets we are developing. We also need to further empirically validate the accuracy of the estimates obtained using the SPL estimation tool.

Acknowledgments. This research is supported by the MULTIPLE project (with ref. TIN2009-13838) funded by the “Ministerio de Ciencia e Innovación (Spain)”.

References

1. Böckle, G., Clements, P., McGregor, J.D., Muthig, D., Schmid, K.: Calculating ROI for Software Product Lines. IEEE Software (May/June 2004)
2. Boehm, B., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D., Steece, B.: Software Cost Estimation with COCOMO II. Prentice-Hall, Englewood Cliffs (2000)
3. Boehm, B.: Software engineering economics. Prentice-Hall, Englewood Cliffs (1981)
4. Boehm, B., Brown, A.W., Madachy, R., Yang, Y.: A Software Product Line Life Cycle Cost Estimation Model. In: Proceedings of the International Symposium on Empirical Software Engineering (ISESE 2004), pp. 156–164 (2004)
5. Chen, Y., Gannod, G.C., Collofello, J.S.: Software Product Line Process Simulator. In: 6th Int. Workshop on Software Process Simulation and Modeling (May 2005)
6. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Boston (2001)
7. Clements, P., McGregor, J.D., Cohen, S.G.: The Structured Intuitive Model for Product Line Economics (SIMPLE), CMU/SEI-2005-TR-003 (2005)
8. Cohen, S.: Predicting When Product Line Investment Pays, Technical Note, CMU/SEI-2003-TN-017 (2003)
9. In, H.P., Baik, J., Kim, S., Yang, Y., Boehm, B.: A Quality-Based Cost Estimation Model for the Product Line Life Cycle. Communications of the ACM 49(12) (December 2006)
10. Lamine, S.B.A.B., Jilani, L.L., Ghezala, H.H.B.: A Software Cost Estimation Model for a Product Line Engineering Approach: Supporting tool and UML Modeling. In: 3rd ACIS Int. Conf. on Software Engineering Research, Management and Applications (2005)
11. Mili, A., Chmiel, S.F., Gottumukkala, R., Zhang, L.: An integrated cost model for software reuse. In: Proc. of the 22nd International Conference on Software Engineering, Limerick, Ireland, pp. 157–166. ACM Press, New York (2000)
12. Montagud, S.: A SQuaRE-based Quality Evaluation Method for Software Product Lines, MSc. Thesis, PhD Program on Software Engineering, Formal Methods and Information Systems, Dept. of Computer Science, Universidad Politécncia de Valencia, Dic. (2009)
13. Nóbrega, J.P., Almeida, E.S., Meira, S.: InCoME: Integrated Cost Model for Product Line Engineering. In: Proceedings of the 34th Euromicro Conference Software Engineering and Advanced Applications (SEAA 2008), pp. 27–34 (2008)
14. Nolan, A.J.: Building a Comprehensive Software Product Line Cost Model. In: McGregor, J.D., Muthing, D. (eds.) Proceedings of the 13th International Software Product Line Conference (SPLC 2009), San Francisco-CA, USA. IEEE Press, Los Alamitos (2009)
15. Ozkaya, I., Bass, L., Nord, R.L., Sangwan, R.S.: Making Practical Use of Quality Attribute Information. IEEE Software, 25–33 (March/April 2008)
16. Poulin, J.S.: The Economics of Product Line Development (1997), `http://home.stny.rr.com/jeffreypoulin/Papers/IJAST97/ijast97.html`
17. Schackmann, H., Lichter, H.: International Workshop on Software Product Management (IWSPM 2006 - RE 2006 Workshop), A Cost-Based Approach to Software Product Line Management. Minneapolis/St. Paul, Minnesota (2006)
18. Schmid, K.: An Initial Model of Product Line Economics. In: Proceedings of the 4th International Workshop on Product Family Engineering (PFE-4), pp. 38–50 (2001)
19. Tomer, A., Goldin, L., Kuflik, T., Kimchi, E., Schach, S.R.: Evaluating Software Reuse Alternatives: A Model and Its Application to an Industrial Case Study. IEEE Transactions on Software Engineering 30(9) (September 2004)

Appendix A

Table A-1. Excerpt of the Quality Model showing the relative importance for quality attributes

Quality Charac.	Attribute	Description	What is probably today	What it should be for SPL
Functional suitability	The degree to which the product provides functions that meet stated and implied needs		Relative Importance	
	Appropriateness	The degree to which the software product provides an appropriate set of functions for specified tasks and user objectives.	[High] Absolutely Mandatory for Safety and business needs	[High] Absolutely Mandatory for Safety and business needs
	Functional suitability compliance	The degree to which the product adheres to standards, conventions or regulations in laws and similar prescriptions relating to functional suitability.	[High] Mandatory compliance to Do-178B	[High] Mandatory compliance to Do-178B
Reliability	The degree to which the software product can maintain a specified level of performance when used under specified conditions		Relative Importance	
	Availability	The degree to which a software component is operational and available when required for use.	[Medium] Obviously we want this but we can mitigate	[High] Otherwise SPL fails
	Fault tolerance	The degree to which the product can maintain a specified level of performance in cases of software faults or of infringement of its specified interface.	[High] We design this into the architecture and coding/design standards	[High] We design this into the architecture and coding/design standards
Maintainability	The degree to which the software product can be modified		Relative Importance	
	Modularity	The degree to which a system is composed of discrete components such that a change to one component has minimal impact on other components.	[High] Driven through the architecture	[High] Driven through the architecture
	Reusability	The degree to which an asset can be used in more than one software system, or in building other assets.	[Medium] We are not good at this	[High]