Towards the Integration of Quality Attributes into a Software Product Line Cost Model

Andy J. Nolan¹, Silvia Abrahão² Paul Clements³, John D. McGregor⁴, Sholom Cohen³

¹Rolls-Royce  
SIN C-3, Rolls-Royce plc, PO Box 31  
Derby DE24 8BJ, England  
Andy.Nolan@Rolls-Royce.com

²Dept of Computer Science  
Universitat Politècnica de València  
Camino de Vera s/n, 46022  
Valencia, Spain  
sabrahao@dsic.upv.es

³Software Engineering Institute  
Carnegie Mellon University  
4500 5th Avenue, Pittsburgh  
PA 15213-2616, USA  
(412) 268-5800  
{clements, sgc}@sei.cmu.edu

⁴Dept of Computer Science  
312 McAdams Hall  
Clemson University  
Clemson, USA  
(864) 656-5859  
JohnMc@cs.clemson.edu

Abstract—A good estimation tool offers a “model” of a project and is usually used to estimate cost and schedule, but it can also be used to help make trade decisions that affect cost and schedule as well as to estimate risks and opportunities. It was evident that Rolls-Royce needed a cost model to underpin decisions when they launched a Software Product Line initiative. The first generation cost model was based on COCOMO II, which represents the software product as a single size measure (Source Lines of Code) but makes limited use of the architecture or any characteristics of the product being developed. The next generation of the cost model, currently under development, is intended to account for the quality attributes of the core assets and the resulting products in order to estimate their impact on cost and net-benefit to the business. The objective of this paper is to describe our current efforts to integrate key quality attributes into the SPL cost model. We describe the quality attributes selected, the reason for their selection and the benefits we expect to obtain after integrating them into the model.

Keywords: Cost Estimation; Software Product Lines; Industrial Experiences; Quality Attributes; Safety-Critical Software.

I. INTRODUCTION

Several benefits are expected from the adoption of a product line approach for related sets of software systems. These include cost reduction, time-to-market improvement, project risk reduction, and quality improvement. Product line engineering is often an economical choice when managers wish to amortize the cost of their core asset base across multiple products or across products with sufficient commonality [8], [4]. However, there is a need for tools that will help managers analyse in which situations and scenarios product line investment pays.

Several cost estimation models for Software Product Lines (SPL) have, therefore, recently been proposed ([19], [20], [6], [4], [22], [9], [14], [12], [16]). A discussion of these models can be found in [18]. However, most of these cost estimation models only represent the product or core assets as the number of source lines of code, and ignore the architecture or characteristics of the product being developed. Very few of these models have, overall, been used in industrial settings, and only one ([12]) considers the quality of products when analysing the costs and benefits of product lines. In [12], a quality-based product line life-cycle estimation model called qCOPLIMO extends the Boehm et al. model [6] to consider the “future costs for correction of defects undetected at product release.” The qCOPLIMO model is based on two existing models which are themselves extensions of the COCOMO II model [5]: COPLIMO [6], which provides a baseline cost estimation model of the SPL life cycle, and COQUALIMO [6], which estimates the number of residual defects. The qCOPLIMO model uses this measure of product quality to enable the cost-benefit analysis of SPL. The model is not granular in terms of the product itself, i.e., it considers the software as a single entity (source lines of code) rather than a collection of discrete features, each of which has different quality attributes.

In this paper, we present an update to a previous experience report concerning the use of cost estimation models at Rolls-Royce [18]. The objective of this paper is to describe how the model is currently being extended with quality attributes in order to analyse the costs and benefits of having core assets with certain qualities, along with the impact of the resulting products (or systems) on quality attributes.

This paper is organized as follows. Section II discusses past experiences in the use of a cost estimation model for SPL at Rolls-Royce, and the need to extend this model with quality attributes. Section III shows the approach used to select critical quality attributes for the Engine Electronic Controllers product area and the benefits that the incorporation of those attributes is expected to bring. Section IV describes an operationalization for one of the attributes selected, in addition to a scenario that helps to illustrate the importance of considering quality attributes when performing product line cost-benefit analysis. Section V presents our conclusions and future work.

II. A COST ESTIMATION MODEL FOR SOFTWARE PRODUCT LINES

The Control Systems department at Rolls-Royce is responsible for the Engine Electronic Controllers (EECs) for a range of small and large gas turbine engines for the aerospace industry. EEC software is developed to DO-178B Level-A standards [10] for safety-critical software.
The company has been developing high integrity software for over 20 years and has extensive data on its processes and productivity.

Rolls-Royce Control Systems has a greater project load than ever before. Customers want faster development, so project timescales continue to decrease, while functionality increases. Meanwhile, shareholders demand lower costs and greater profitability, and a step change in productivity is therefore needed. Software product lines will help to meet these challenges. Since 2008, Rolls-Royce has developed an SPL for the business, which has potential for both the software and hardware aspects of engine design.

Since 2004, Control Systems has invested in developing reliable estimation models to predict software development cost and schedule based on COCOMO II [5]. The development of estimation models has led to an improvement in stability and productivity, with an average of around 11% productivity improvement (cost per equivalent functionality) per project. This benefit has occurred because a project has a better understanding of the factors that affect cost and risk. The estimation models encourage a project to place greater emphasis on improving factors that will reduce overall project cost.

When the SPL initiative was launched in 2008, the development of a reliable and comprehensive estimation model was seen as critical to making the right decisions to develop core assets that would bring the greatest business value. The first version of the SPL estimation model [17] was developed by using some additional data from Rolls-Royce (effort data, defect data, process performance and asset size) as a basis. This initial estimation model allows core assets to be modelled according to the product line’s software architecture, thus permitting the cost involved in developing and deploying each asset to be estimated. The model estimates the cost-benefit of each core asset based on the size, complexity, volatility, and difficulty involved in development and deployment. Benefit is calculated for each core asset, along with the benefit it brings to each product and the overall benefit to the business.

That first SPL estimation model used at Rolls-Royce produced useful results, but was insufficient given the importance of some core asset quality attributes. For example, the dominant costs for a safety-critical product line lie in gathering evidence to support safety certification. This evidence is mainly gathered from verification activities (reviews, analysis and testing). New estimation models will need to consider quality attributes such as testability in order to make trade-off decisions about core assets and product development and deployment costs. For example, there is a need to assess the number of variations a core asset can have (variability) while not compromising its testability.

III. SELECTING AND PRIORITISING QUALITY ATTRIBUTES

Gathering and applying quality attribute information to extend existing models can be difficult. A key issue is how to identify the quality attribute requirements and their competing importance for a particular domain or market segment. The goal in building an estimation model is to collect information on the relative importance of specific quality attributes and to form possible hypotheses (to be validated empirically) about their impact both on the cost-benefit of product lines and on each other.

Several methods can be used to choose and define those quality attributes which are critical to a product line in order to include them in a model. These include interviews, surveys, and quality attribute workshops [3]. These methods are complementary and can be used in combination with each other to provide more evidence about the relevance of any specific quality attribute.

We began with a list of quality attributes taken from the new ISO/IEC SQaRE standard [13], in addition to other attributes that are specific for safety-critical embedded systems (e.g., safety, certifiability). The quality attributes for updating the model were selected and prioritized by means of a series of interviews with selected stakeholders. Each attribute was classified by the stakeholders as follows:

- Process in which the quality attribute is measured/demonstrated (e.g., core asset development, product development, product certification)
- Artefacts that are measured (e.g., system architecture, software architecture).
- Relative importance for safety-critical systems: Low, Average, High.
- Measurability: Low, Average, High.

The classification was based on stakeholder intuition and experience in the development/management of safety-critical systems at Rolls-Royce. Four key practitioners (SPL architect, SPL specialist, metrics specialist and SPL manager) were interviewed. Interviewees in different roles were involved in order to collect different viewpoints and perspectives about the relative importance of quality attributes. They were selected in accordance with their roles and responsibilities in defining the software architecture and measurement/process improvement programs at Rolls-Royce. The interviews ranged from 60 to 90 minutes in length.

The results of these interviews indicated that a large proportion of quality attributes in safety-critical systems are not measured but are rather demonstrated through analysis and tests. Moreover, several quality attributes are not tradable with other quality attributes (e.g., safety vs. functional correctness). Some quality attributes for products were classified as being of low importance for this domain (e.g., usability). The results also indicated that only a subset of the quality attributes for core assets are currently measured or demonstrated; the majority are demonstrated for core assets, for the software architecture, or for complete products. Table 1 presents the quality attributes selected for the software product line and domain being considered. They are presented by priority. All attributes listed in the table were classified as being of high importance to the enhancement of the Rolls-Royce product line estimation model. Each of the selected quality attributes is discussed below.
A. Testability

Testability is an important quality attribute for any type of product, and for safety-critical products in particular. This type of product differs from traditional software primarily in the requirement to provide evidence that the software fits its purpose before the system is deployed [11]. Evidence to support the product’s approval is typically gathered through verification plans that define the development processes, analysis, and test activities to be undertaken.

Based on data collected at Rolls-Royce (see Fig. 1), around 52% of the total development effort of an average non-SPL safety-critical product consists of some form of verification and validation. An SPL initiative will help to reduce development effort, but most of that verification effort is made at the time of product integration. For example, component testing may be performed by the core asset team, but the product-producing team may have to re-execute the test cases and introduce new ones that are unique to the new product context.

In the software product line at Rolls-Royce, data show that up to 72% of a product’s overall effort will be spent on some form of verification and validation (see Fig. 1). The percentage is higher on a product line not because verification has increased but because the development effort has decreased making verification costs higher in relation to the overall effort. The verification effort could, theoretically, increase to over 90% in cases of very low costs in core asset reuse. Testability therefore becomes critical for safety-critical product lines. A core asset architect would benefit from measuring trade-offs between testability and other quality attributes such as variability.

B. Maturity

The concept of maturity as a quality attribute is not new to software product lines. McGregor [15] refers to core assets maturing with the frequency of their use, and Bosch [7] identified three maturity levels for each of the three main artefacts that make up an SPL (product line architecture, core assets, and the products derived from core assets).

The maturity of a core asset has been defined here as the degree to which an asset is free from need for further modification i.e., it meets its intended functionality and is free from defects. A low maturity asset is likely to be exposed to changes and, depending on when these changes occur, this can lead to high levels of effort to fix defects. At Rolls-Royce this is known as “scrap & rework.”

<table>
<thead>
<tr>
<th>Quality Attributes</th>
<th>Definition</th>
<th>Process in which it is measured/demonstrated</th>
<th>Relative Importance</th>
<th>Measurability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testability</td>
<td>The ease with which a product or a core asset can be made to demonstrate its faults through (typically execution-based) testing.</td>
<td>Core asset development, core asset deployment, product development.</td>
<td>[High] Testing makes a significant cost impact on SPL in safety critical domains.</td>
<td>[High] In the short term it is modelled. In the long term it will be measured.</td>
</tr>
<tr>
<td>Maturity</td>
<td>The degree to which a core asset is free from need for further modification.</td>
<td>Core asset deployment, product development.</td>
<td>[High] Allows trade-offs with testability, variability, volatility.</td>
<td>[High] In the short term it is modelled. In the long term it will be measured.</td>
</tr>
<tr>
<td>Variability</td>
<td>The ability of a system, an asset, or a development environment to support the production of a set of artefacts that differ from each other in a pre-planned fashion [1].</td>
<td>Core asset development, core asset deployment, and product evolution.</td>
<td>[High] Mandatory for any software product line.</td>
<td>[Medium] In the short term it is modelled. In the long term it will be measured.</td>
</tr>
<tr>
<td>Maintainability</td>
<td>The ease with which a product or a core asset can be modified to correct faults, improve performance, or adapt to a changing environment.</td>
<td>Core asset development, core asset deployment and product development.</td>
<td>[High] Allows trade-offs with testability, variability, maturity.</td>
<td>[Medium] Maintainability index [1] and other existing measures.</td>
</tr>
<tr>
<td>Reusability</td>
<td>The degree to which a core asset can be used in more than one software system, or in building other core assets.</td>
<td>Core asset development, core asset deployment and product development.</td>
<td>[High] Allows trade-offs with testability, variability, maturity.</td>
<td>[Medium] Existing frameworks for evaluating reusability (e.g., [21]).</td>
</tr>
</tbody>
</table>

TABLE I. QUALITY ATTRIBUTES SELECTED

Figure 1. Percentage of effort spent in testing by the core asset team and then for each project (P1 – P17). The red bar indicates the percentage of testing effort for a non-SPL project.
Maturity has become a sensitive issue for product lines, particularly if a product is using low maturity assets. Fig. 2 shows that for an average engine controller, developed in a traditional manner, approximately 50% of effort can be spent on scrap & rework and 50% on the development of the assets. For product lines, this impact may have wider implications if multiple products are affected as an asset matures.

The original SPL estimation model used by Rolls-Royce was limited in that it assumed that any core asset used in a product was already 100% mature. This is overly simplistic and hides the consequences of iteration and redesign, since assets are redeveloped and used in different product contexts.

C. Variability

Variation mechanisms used in core assets help to control the required adaptations and to support the product developers in their task [2]. The selection of a specific variation mechanism for a core asset may have a significant impact on the final product development cost. This quality attribute was modelled by using the first version of the SPL cost estimation model [17]. The intention was to have a detailed understanding of the process costs of variability and to understand how these costs would change depending on the type of variation mechanism adopted and who was developing the core assets.

The costs involved in developing and deploying a core asset are based on the effort needed for each process (e.g., software requirements, software architecture, component test) and sub-process (e.g., write, review, correct & issue), as well as on the selection of processes that will be used by both the core asset team and the product team. Table II shows the relative cost per process based on an average of non-SPL projects at Rolls-Royce. The first version of the SPL estimation model also considered the effort involved in using each variation mechanism (see Table III for an explanation of the mechanisms used at Rolls-Royce).

The cost of variability in a core asset is calculated by multiplying the cost of deploying the asset (in a specific process) by the cost of using the different variation mechanisms each time a product is derived from the core assets. Table VIII (see Appendix A) shows how the effort for asset development and deployment was calculated for each variation mechanism. The table multiplies the effort needed to follow a process (as shown in Table II) by the percentage of that process that must be performed for each variation mechanism and for both core asset development and deployment. The product of these two values gives the estimated % of the effort required. The estimated effort (hours) to develop and deploy an asset is then calculated as the effort to develop the core asset multiplied by these % values.

Although variability was included in the first version of the SPL cost estimation model [17], we plan to explore the use of existing measures (e.g., [23]) in order to measure variability in a predictive manner by considering the characteristics of the product line architecture.

### Table II. Relative Effort Per Process

<table>
<thead>
<tr>
<th>Process</th>
<th>% Total Project Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systems Architecture</td>
<td>6%</td>
</tr>
<tr>
<td>Software Requirements</td>
<td>19%</td>
</tr>
<tr>
<td>Software Architecture</td>
<td>8%</td>
</tr>
<tr>
<td>Software Design</td>
<td>11%</td>
</tr>
<tr>
<td>Software Code</td>
<td>6%</td>
</tr>
<tr>
<td>Component Test</td>
<td>10%</td>
</tr>
<tr>
<td>SW/SW Integration</td>
<td>9%</td>
</tr>
<tr>
<td>HW/SW Integration</td>
<td>3%</td>
</tr>
<tr>
<td>Validation</td>
<td>9%</td>
</tr>
<tr>
<td>Overhead</td>
<td>19%</td>
</tr>
</tbody>
</table>

### Table III. Variation Mechanisms Used at Rolls-Royce

<table>
<thead>
<tr>
<th>Variation Mechanism</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calibrated</td>
<td>Component's behaviour is altered by manipulating &quot;constant&quot; data values to fine-tune the performance of the component to match the domain. This is NOT used to switch in/out functionality, only to shape the component's algorithmic response.</td>
</tr>
<tr>
<td>Plug Replaced</td>
<td>Multiple components are produced to the same interface specification to allow large-scale functional replacement.</td>
</tr>
<tr>
<td>Autocoded</td>
<td>Component has a defined interface but its behaviour is specified by the project using a graphical programming language, and the component implementation is auto-generated.</td>
</tr>
<tr>
<td>Composed</td>
<td>Component's behaviour can be varied by the inclusion/exclusion of code fragments and operations. The PL asset contains all permissible variations and mechanisms for a project to select and compose the required behaviour.</td>
</tr>
<tr>
<td>Generated (PL Generator)</td>
<td>The PL assets are a restricted definition language (Domain Specific Language) and a code generator which can produce components for a specific purpose. Projects define the required behaviour in the restricted language and the component is auto-generated. This is a more specific and restricted form of the &quot;Autocode&quot; shown above - here the languages are typically much more specialised and defined by the PL organisation rather than being a COTS autocode product.</td>
</tr>
</tbody>
</table>
D. Maintainability

In a product line, the maintainability of core assets is of great importance. SPL developers no longer maintain each individual product, but rather maintain the core assets used to develop the products. It is also important to understand the degree of maintainability that a core asset should have if it is to be cost-effective to the business. More is not always better, since making a core asset maintainable comes with a cost that, depending on the actual changes that occur, is not always recovered.

Predictive measures thus seem to be more useful than actual measures. Several existing studies deal with maintainability in SPLs. For example, Aldekova et al.[1] used the Maintainability Index measure to quantify maintainability in feature-oriented product lines. However, we did not find any existing work that shows the impact of this quality attribute on SPL cost.

E. Reusability

A core asset is a key element in product line engineering. Software systems are built by instantiating core assets. The reusability of core assets therefore largely determines the success of product line projects. Reusability is defined here as the degree to which a core asset can be used in more than one software system, or in building other core assets. The reusability of core assets can be measured by using existing frameworks for their evaluation (e.g., [21]). A higher reusability of core assets indicates the potential for a higher return-on-investment. However, we are not aware of any existing work that provides empirical evidence with regard to the impact of reusability on product line cost-benefit.

F. Trade-off Among Quality Attributes

The analysis of both the relationships among quality attributes and the impact of quality attributes on cost-benefit, allowed us to extract a number of hypotheses to be validated. Examples include:

- Variability reduces testability
- Testability improves maintainability
- Testability reduces cost
- Reusability reduces cost
- Testability improves maturity
- Variability reduces maturity
- Maturity reduces cost

These hypotheses need to be tested empirically so as to gather empirical evidence about the impact of quality attributes on the cost-benefit of SPL.

As an example, the maturity/variability relationship leads to trade-offs in testability (see Fig. 3). This is because the desire to manage variation across products (variability) may lead to an increase in complexity, size, number of variation points, etc. Each of these factors adds to the test burden, thus reducing the testability of the core asset.

We believe that maturity has an influence on (or is influenced by) other quality attributes such as variability, reusability, testability and maintainability. If the domain is unstable with a large range of possible requirements, it may be tempting to add more variability to a core asset. However, as the variability (or the number of variations) of a core asset increases, it will take longer to achieve maturity from product exposure (defined in terms of using most of the variants in actual products). Moreover, as we add more variability, the number of test cases needed to test the core asset may increase and the core asset may become larger and more complex. This decreases the testability of a core asset and drives up testing effort per project. In addition, if a core asset is maintainable, then its maturity may have decreased.

A core asset that has been widely reused should, in theory, have an increased maturity. In cases in which the number of variants is small (e.g., a library function), this trade-off is not difficult or as critical. As the number of variants increases, the architect will need to make more complex trade-offs between variability and reusability, which will in turn affect testability (total testing effort) and maturity (total cost to the business).

![Figure 3. Relationships between quality attributes where “+” implies a positive influence (i.e., improvement) and “-” is a negative influence.](image)

IV. OPERATIONALIZING MATURITY AND ANALYSING ITS IMPACT ON PRODUCT LINE COST-BENEFIT

According to the ISO/IEC 25010 [13], more than one measure can be used to quantify a quality attribute, suggesting that a quality attribute can be operationalized in different ways. This section describes an operationalization for the maturity quality attribute described in Section III (B), and how maturity can be quantified by considering the characteristics of the processes employed at Rolls-Royce. It also considers trade-offs with testability and variability, as shown in Fig. 3.

We propose that the maturity of a core asset should be measured in terms of the risk associated with its use, where risk represents any likely reason that the core asset will change. This relationship is expressed as follows:

\[
Risk = (1 - \text{maturity})
\]

(1)

Maturity can be quantified in terms of three factors, as follows:

\[
\text{Maturity} = \text{f} (\text{process exposure, product exposure, uncertainty})
\]

(2)

where \text{process exposure} indicates the maturity gained from the application of processes used to identify and remove defects from a core asset; \text{product exposure} indicates
the degree to which the range of variants has been exposed to real products; and uncertainty indicates any uncertainty that the variation built into the core asset will not accommodate future needs.

We assume that it is possible to have a core asset that has 100% process exposure but may still be immature with regard to product exposure. That is, a core asset may be developed to a high standard and yet may not be functionally complete. On the other hand, it is possible to have a core asset that is functionally complete but has not had sufficient process exposure. The exact contribution from each of these three factors has yet to be determined.

Rolls-Royce’s experience in measuring project maturity can be used to model the maturity of core assets throughout their life. Fig. 4 shows a maturity curve for an average of five products. According to data collected at Rolls-Royce, a product may take up to 10 years to fully mature. Engine integration will occur around years 1 and 2, and aircraft integration around years 2-3. Aircraft trials take place from year 3 onwards, and so on. Process exposure is complete at around the time of engine certification, i.e., year 2.

![Figure 4. The average percentage of software change requests closed over the first 10 years of a “non-SPL” engine development. The chart represents an average of several projects.](image)

Fig. 5 shows a maturity curve for core assets modelled according the maturity curve for products shown in Fig. 4. The figure accounts for the three factors of asset maturity: process exposure, product exposure and uncertainty. The maturity from process exposure will occur very early on i.e., in years 1 - 3, and the product exposure and the elimination of uncertainty will take place from year 2 onwards.

A measure of maturity can be used to predict the scrap & rework that a product development project should expect (i.e., amount of effort required to fix defects). Scrap & rework is the result of unmitigated risk. Strictly speaking, some scrap & rework is also caused by other factors; for example, a decision to add new functionality causes some rework but this new functionality is not driven by risk. Therefore, risk causes scrap & rework but not all scrap & rework is caused by risk.

![Figure 5. Maturity curve for core assets modelled according to the product maturity curve](image)

The details below are a summary of the data gathered to date, and the approach to be used to validate how the maturity of core assets affects the product line cost-benefit. Our hypothesis is the following: The use of more mature assets will reduce the effort spent on scrap & rework, and reusing more mature core assets will therefore reduce the cost involved in building a specific product. Greater maturity leads to lower costs.

A. Process Exposure

Each process used during development, deployment and verification brings a different level of value to a core asset in terms of developing its maturity. Fig. 6 shows data from a range of Rolls-Royce projects, showing where errors are, on average, detected. Process exposure becomes interesting when considering the processes that can be used by the core asset development team and those by the product team. Table IV shows this in principle. By combining the information in Table IV and Fig. 6, we can see that the core asset team adds 91% of the process maturity and that the remaining 9% is added by the product teams. 91% is gained from the actual development of the core asset (50% as described in Section III-B) and the remaining 41% is gained through defect detection and correction from the review, modelling, and some form of testing.

![Figure 6. Percentage of defects detected per process (non-SPL development). The data shows the percentage of functional changes and combines both system defects and software defects.](image)
TABLE IV. AN APPROXIMATION OF THE PROCESSES DEPLOYED BY THE CORE ASSET AND PRODUCT TEAMS

<table>
<thead>
<tr>
<th>Process used at Rolls-Royce</th>
<th>Core Asset Team</th>
<th>Product Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Development &amp; Review</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Requirements Modelling</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Design &amp; Design Review</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Coding &amp; Coding Review</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Component Test</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SW/SW Integration</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>HW/SW Integration</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Systems Integration</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Engine Test</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Iron Bird</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Flight</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Service</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Since the majority of the integration effort is spent by product teams, this aspect of maturity cannot be gained by the core asset team. The value of 91% is an approximation and varies according to the variation mechanism adopted. Table V shows the respective effort spent by the core asset team and product team for each variation mechanism.

TABLE V. THE EFFORT EXPENDED BY THE CORE ASSET TEAM AND THE PROJECT TEAM FOR EACH VARIATION MECHANISM

<table>
<thead>
<tr>
<th>Variation Mechanism</th>
<th>Core Asset Team</th>
<th>Product Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calibrated (Constant Data)</td>
<td>60.7%</td>
<td>39.3%</td>
</tr>
<tr>
<td>Plug Replaced</td>
<td>60.7%</td>
<td>39.3%</td>
</tr>
<tr>
<td>Autocoded (e.g. SCADE)</td>
<td>57.1%</td>
<td>42.9%</td>
</tr>
<tr>
<td>Composed</td>
<td>59.5%</td>
<td>40.5%</td>
</tr>
<tr>
<td>Generated (PL Generator)</td>
<td>21.3%</td>
<td>78.7%</td>
</tr>
</tbody>
</table>

However, the situation is more complicated when factoring for the cost of a change. Fig. 7 shows the average relative cost to make the same change at different points in a non-SPL project life-cycle. Errors detected and removed at review will cost, on average, one unit of work. If allowed to escape to integration (SW/SW and HW/SW) the cost grows to five units of work, and so on. What remains to be shown is whether this cost escalation still holds true for SPL.

When adjusting maturity by the cost of change, we obtain a measure of maturity in terms of cost to the business. Based on this analysis, an average asset will have gained a process maturity of 61% through asset development. Low maturity can be improved through exposure to simulated products, demonstration projects and from the first product exposure.

**B. Product Exposure**

This is defined as the exposure of an asset with a range of variations in real products. For example, if an asset has 10 possible configurations that will ever be used (it might actually have 1000 potential variations but only 10 will be used), then an asset can only be said to be mature once all 10 configurations have been exposed in relevant product contexts. It is tempting to believe that an asset that has been deployed and used on 10 products is likely to be equally mature. However, if the same configuration is used in each of the 10 products, then it does not gain any new maturity. If all 10 configurations have been used in 10 different product contexts, then we can say that the asset is mature in terms of product exposure. Product exposure can be quantified as follows:

\[
\text{Product exposure} = \frac{\# \text{ of configurations deployed on products}}{\text{Total} \# \text{ of configurations that will ever be used}}
\]

**Product exposure can be related to test coverage. This means that unless an asset has actually been used in a real situation, it is not possible to assume that its functionality is mature. Emergent needs or emergent behaviour may cause unforeseen modifications.**

**C. Uncertainty**

We consider two types of uncertainty here: foreseen and unforeseen. Foreseen uncertainty is characterized by identifiable and understandable influences that might occur within a given situation. For example, during core asset development, when developing a domain model, there may be some scenarios (variants) that may be seen as less likely to occur and would be too expensive to invest in, but to not factor for these variants poses a risk to the core asset. The analysis can be performed on an asset by asset basis to reveal where the risk may arise. Studies of this form have been accurate to 80% (in non-SPL single projects) when performed on engine controllers, thus suggesting that uncertainty is relatively predictable. The foreseen risk can therefore be estimated (quantified) by using the standard risk
that it would take 100 sure that the core asset used to build a specific product (in terms of its and so on. Although unforeseen, a level of change should be expected scrap 

(75% of uncertainty remaining). In year 2 it is 0.13 * 1-0.55 (45% of uncertainty remaining), and so on.

Fig. 8 shows a maturity curve for a single product, along with the relative core asset maturity over the same time period. This illustrates that if a product is built using core assets that have been used over the last 5 years and are 90% mature (i.e., 90% defect free), the amount of expected scrap & rework is much less than if the same product is built using core assets that are 72% mature.

![Figure 8. Impact of asset maturity on product maturity for a given time period](image)

The maturity values calculated in Table V have been calibrated to account for the cost of scrap & rework. A project can therefore derive the scrap & rework rate directly from these maturity values. Let us suppose that the core asset has a maturity of 75% (when factoring for both process exposure and product exposure), scrap & rework is calculated as $1 - \text{maturity} = 25\%$. So, over the course of the core asset’s life, there will be an additional cost of scrap & rework:

- Core Asset Team: 100 hours * 25% = 25 hours
- Product Team: 100 hours * 25% = 25 hours

Therefore, the total (statistical) cost for this asset will be:

- Core Asset Team: 100 hours * 25% = 25 hours
- Product Team: 100 hours * 25% = 25 hours

In terms of trade analysis, should the core asset team develop a more variable core asset that takes longer to mature or should it develop multiple simpler core assets that are quicker to develop and mature? The following is an illustration of this trade-off.

Let us suppose that a simpler core asset takes 75 hours to develop and, because of its simplicity, we obtain a higher level of product exposure and therefore this time the maturity is 95%. However, we have to develop two core assets to cover the range of functionality required. The total development costs are now $75 \times 2 = 150$ hours.

- Core Asset Team: 150 hours * 75% = 112.5 hours
- Product Team: 150 hours * 75% = 112.5 hours

8 in Appendix A therefore provides a way in which to split these costs between the core asset development team and the product deployment team, e.g., for the “calibrated” variation mechanism, the costs will be split as follows:

- Core Asset Team: 100 hours * 73% = 73 hours
- Product Team: 100 hours * 31% = 31 hours

Therefore, the total (statistical) cost for this asset will be:

- Core Asset Team: 100 hours * 25% = 25 hours
- Product Team: 100 hours * 25% = 25 hours

In terms of trade analysis, should the core asset team develop a more variable core asset that takes longer to mature or should it develop multiple simpler core assets that are quicker to develop and mature? The following is an illustration of this trade-off.

Let us suppose that a simpler core asset takes 75 hours to develop and, because of its simplicity, we obtain a higher level of product exposure and therefore this time the maturity is 95%. However, we have to develop two core assets to cover the range of functionality required. The total development costs are now $75 \times 2 = 150$ hours.

- Core Asset Team: 150 hours * 75% = 112.5 hours
- Product Team: 150 hours * 75% = 112.5 hours

In this example, it is still more cost-effective to have a single core asset with a lower maturity then to develop two core assets with a higher maturity.

E. Incorporating Quality Attributes into the Cost Model

The SPL cost model [17] described a ten-step estimation model used at Rolls-Royce (Fig. 9). New factors will be added to the model to accommodate each of the quality attributes when
attributes discussed in Section III. A number of quality attributes are already addressed by the existing cost model. For example, in Step 5 of the cost model, the architect identifies the variation mechanisms to be used for each core asset and the resulting cost associated with the development of the core asset. The model then identifies the processes and the effort the core asset team and the product team will spend. The predicted cost for developing the asset is based on COCOMO II [5] and includes the RUSE (design for Reuse) and DOCU (additional documentation) factors [6].

A quality attribute such as maturity is new to the model, and we are now working to extend the model to take maturity into account. As described previously, maturity comprises three elements: process exposure (did we build it right?), product exposure (did we build the right thing?) and uncertainty (is there a risk that the asset will need to change in the future?). At present, Rolls-Royce intends to adopt only process exposure and product exposure in the model. It would appear that in this domain uncertainty strongly correlates to product exposure: as product exposure grows, uncertainty decreases. We do not therefore deem it necessary to incorporate both factors in the model, although modellers for other domains may wish to do so.

Step 6 of the estimation model (Fig. 9) is designed to allow the SPL manager to prioritize the development and deployment of core assets. At present, a core asset is either deployed or not, with no consideration for its maturity being factored into costs and plans. The cost model is now being extended with two new questions related to the maturity of the core asset at the time of adoption. For each asset/product combination, the SPL manager will need to declare the core asset’s maturity by answering two questions:

- Percentage of process exposure (to be determined through analysis and measurement)
- Percentage of product exposure

Tables VI and VII represent an approximate value of the maturity gained through process and product exposure. The values are based on the defects detected by each process as shown in Fig. 6. Process exposure (at present) represents the defects detected up to systems integration. The values for product exposure are based on the defects detected during engine and airframe integration. All values are weighted according to the impact and cost involved in scrap & rework. In the case of product exposure, this value is binary - if a particular variation mechanism has been used on a product, it is assumed to be 100% mature. If this is the first product exposure, then it will, on average, be 87% mature (12% risk per value).

<table>
<thead>
<tr>
<th>TABLE VI.</th>
<th>PERCENTAGE OF PROCESS EXPOSURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Exposure</td>
<td>% Process exposure</td>
</tr>
<tr>
<td>Developed</td>
<td>50%</td>
</tr>
<tr>
<td>All reviews complete</td>
<td>59%</td>
</tr>
<tr>
<td>Core Asset verification</td>
<td>61%</td>
</tr>
<tr>
<td>Systems Integration</td>
<td>80%</td>
</tr>
<tr>
<td>Engine / Aircraft Integration</td>
<td>90%</td>
</tr>
<tr>
<td>Product in Service</td>
<td>100%</td>
</tr>
</tbody>
</table>

Because risk = 1 – maturity, both the core asset team’s effort and the product teams’ effort can use a core asset’s maturity at the time of adoption to argue for an increase in their allotted resources. When an asset is 100% mature, no additional costs are expected for either team, but when less mature assets are used, both the core asset team and the product teams will have to plan for additional costs for the estimated work required to mature the core asset.

V. CONCLUSIONS AND FUTURE WORK

This paper describes our current efforts to integrate key quality attributes into the Rolls-Royce product line cost estimation model. We have presented a set of prioritised quality attributes and a scenario based on two specific quality attributes (maturity and variability) to analyse the importance of considering quality attributes when performing SPL cost-benefit analysis. We have also explained how maturity is being integrated into the model.

The Rolls-Royce estimation model will need to be updated to take into account the other quality attributes selected (i.e., testability, maintainability, reusability). Correction factors will be added to factors for the effort required by the core asset team and the product teams to consider the iteration of assets with these quality attributes.

The model will need to indicate the quality of a core asset at the time of adoption. For instance, in the case of maturity, if a core asset has never been deployed, the first product team to use it can expect the asset to be 61% mature and can therefore expect to spend a 39% penalty from the process exposure to fund the modifications and retesting. A core asset that has been used many times should have a higher maturity and a lower probability of needing to be modified.

Maturity and scrap & rework are time sensitive issues. If the assets are mature at the beginning of a product development project, the cost to develop a product will be less than if less mature assets are used. Also, maturity has been measured as change requests (they represent the manifestation of process exposure, project exposure and uncertainty) but we need to measure these three elements separately. Future work will address these issues, along with the definition of scenarios for other quality attributes.

ACKNOWLEDGMENT

This research is supported by the MULTIPLE project (with ref. TIN2009-13838) funded by the MICIN, Spain.

REFERENCES


APPENDIX A

TABLE VIII. % EFFORT PER PROCEDURE ACTIVITY AND VARIATION MECHANISM. THE TABLE WAS USED TO MODEL THE COSTS TO DEVELOP AND DEPLOY AN ASSET FROM EACH VARIATION MECHANISM

<table>
<thead>
<tr>
<th>Activity</th>
<th>% Total Project Effort</th>
<th>PL: Calibrated (Constant Data)</th>
<th>PL: Plug Replaced</th>
<th>PL: Autocoded (SCADE)</th>
<th>PL: Composed</th>
<th>PL: Generated (PL Generator)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>% PL</td>
<td>% Proj</td>
<td>% PL</td>
<td>% Proj</td>
<td>% PL</td>
<td>% Proj</td>
</tr>
<tr>
<td>Systems Architecture</td>
<td>6%</td>
<td>100%</td>
<td>8%</td>
<td>100%</td>
<td>8%</td>
<td>100%</td>
</tr>
<tr>
<td>Software Requirements</td>
<td>19%</td>
<td>100%</td>
<td>1%</td>
<td>100%</td>
<td>1%</td>
<td>100%</td>
</tr>
<tr>
<td>Software Architecture</td>
<td>8%</td>
<td>100%</td>
<td>1%</td>
<td>100%</td>
<td>1%</td>
<td>100%</td>
</tr>
<tr>
<td>Software Design</td>
<td>11%</td>
<td>100%</td>
<td>2%</td>
<td>100%</td>
<td>2%</td>
<td>100%</td>
</tr>
<tr>
<td>Software Code</td>
<td>6%</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td>Component Test</td>
<td>10%</td>
<td>100%</td>
<td>26%</td>
<td>100%</td>
<td>26%</td>
<td>100%</td>
</tr>
<tr>
<td>SW/SW Integration</td>
<td>9%</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td>Validation</td>
<td>9%</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td>Overheads</td>
<td>20%</td>
<td>67%</td>
<td>37%</td>
<td>67%</td>
<td>37%</td>
<td>67%</td>
</tr>
<tr>
<td>% of Total Project Effort</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Figure 9. Steps of the Software Product Line Estimation Model.