Enhancing Requirements and Change Management through Process Modelling and Measurement[1]

Luigi Lavazza and Giuseppe Valetto

CEFRIEL and
Politecnico di Milano[2]

Abstract

Effective management of requirements is of vital importance for any software development effort. This activity spans from software procurement to project management, to the technical development activities.

We present a methodology that aims at improving the effectiveness of requirements management in software development and maintenance. In particular, we address quantitative assessment of the impact of requirements changes, and quantitative estimation of costs of the development activities that must be carried out to accomplish those changes.

Our approach is based on enhanced traceability and process measurement. Traceability relations are derived from an integrated view of the process and product models adopted within the software development organization. Hence, traceability spans over all the software life cycle and its products. Moreover, through proper measurement, the elements in the process and product models are quantitatively characterised, thus achieving a sound basis for different kinds of sophisticated analysis concerning the impact of requirements changes, their cost over the development process, and the sensitivity of the product to changes.

The methodology presented here is the basis of the ESPRIT project SACHER. In this project a tool supporting the proposed approach was developed, and both the methodology and the tool were used in different case studies, in order to assess their ability to handle the complexities inherent in the technical, managerial and contractual dimensions of requirements and change management.

Keywords: requirements management, change management, traceability, software process modelling, software measurement, cost estimation.

1 Introduction

Industrial software-intensive systems are affected by unavoidable requirements changes that can occur during any phase of the production cycle and have strong effects on a variety of technical and managerial aspects. Those effects span from the procurement phase, to project management, to the technical development activities. For this reason, requirements management is emerging as a key factor for the success of industrial software projects, impacting the contractual, technical and cognitive dimensions [9].

Both procurers and suppliers of software-intensive systems are beginning to demand for sophisticated requirements management capabilities. Procurers, on the one hand, aim at more effective management of contractual practices: for instance, quantification

---

[1] This work is partly funded by the CEC as ESPRIT-IV project n. 23156 SACHER.

[2] P.za Leonardo da Vinci, 32 - 20133 Milano (Italy). E-mail: lavazza@elet.polimi.it, valetto@cefriel.it. Fax: +39-02-23954254

of the contract clauses and assessment of their accomplishment. Suppliers, on the other hand, perceive requirements management as a competitive advantage and a way to business success. Besides understanding correctly and completely the technical issues and goals related to the system development (which is the purpose of "classic" requirements engineering [6]), they also need to reduce the risk of supply failures and to enhance contracting and customer management; for these purposes, they need the ability to precisely estimate and control supply costs with respect to mutable requirements.

These expectations, however, are not completely met by methodologies and tools for requirements management. The state of the art focuses on the concept of requirements traceability [7] [10], which in principle provides means to keep track of and analyze the consequences of requirement changes over the whole software product. It is recognized that the amount of knowledge captured by means of traceability can be invaluable for a software development organization, but that knowledge remains often implicit and can be scarcely exploited without proper formalization and structuring. Moreover, traceability must be customized according to organization-specific as well as project-specific goals to maximize its usefulness [8].

Nowadays, a number of environments devoted to requirements traceability exist, mostly commercial tools [11]. The current market offer provides facilities to import and group together representations of the various project artifacts into proprietary requirements management repositories, and to decorate those artifacts with traces originating from the requirements specifications. On top of that, querying facilities to carry out different kinds of trace analysis, e.g. impact analysis, and various reporting and presentation facilities are made available.

It is noticeable that most tools provide little support to structuring of the information (both artifact representations and traces) - especially in comparison with the complexity of software development scenarios - and do not mandate nor guide the creation of sophisticated data models. This lack of information structuring makes difficult to capture the richness and complexity underlying the data, and in particular to adapt the functionality of the tool to the peculiarities of a given organization or project. Consequently, the kind of analysis and knowledge extraction that is possible is limited and generic.

Instead, we believe that to fully exploit the potential knowledge embedded in requirements traceability, it is essential that the structure of the information reflects goals and methods of the development organization. A natural and convenient way to achieve this is to explicitly model that information according to the adopted product and process models. In this way a requirements management tool becomes *process and product aware*, with noticeable advantages. Awareness of the product model enables customized traceability and analysis, focusing on the knowledge relevant to certain product facets or certain project goals. Awareness of the process model adds further value, enabling to better investigate and understand the causal relations underlying traces (e.g. an artifact affects another since the former is the input of an activity that produces the latter).

Product and process knowledge together enable more precise qualitative reasoning about the kind and amount of work to be carried out in response to requirement changes, and allow evaluation of consequences on the overall project, (e.g. evaluating

the impact of a requirement change in the various phases of the life cycle and upon ongoing development activities).

Product and process knowledge also poses the basis for *quantitative* analysis. The prerequisite is a *measurable* process, that is, a process whose activities are quantitatively characterized through measurement. Measuring a process activity involves choosing the characteristics of interest that must be measured (for example, quantitative properties of the activity's input, output and of the employed resources). By taking a significant number of observations (i.e. by carrying out an appropriate *measurement process*), and applying statistic analysis, it is often possible to derive predictive models that can be employed to quantitatively estimate the consequences of a change along various dimensions: for example the cost of implementing the change.

Quantitative impact analysis and cost estimations can be clearly beneficial in a variety of ways, for example:

- In negotiation and contracting, to assess the validity of bids and offers against (changes to) the user requirements specification, something that is equally appealing to both sides of the procurer-supplier relationship.
- In project management, to support estimation of the effort related to a proposed change and to minimize the risk of supply failures; also, to carry out "what if" analysis and allocate work and resources in the most efficient manner.
- In development and maintenance, to assess the flexibility of the product with respect to proposed modifications.

In this paper, we present a methodological approach to quantitative requirements and change management, which is founded upon model-compliant traceability and measurable processes. In Section 2 we discuss the characteristics of models adequate to support the approach. In Section 3 we illustrate how to exploit measurement of the process and product for quantitative requirements and change management. In Section 4 we give an overview of the kinds of impact and cost analysis enabled by the methodology. In Section 5 we provide information on the implementation and validation of the methodology in the SACHER (Sensitivity Analysis and CHange management support for Evolving Requirements) ESPRIT project. In Section 6 we compare and contrast our work with related work in the domains of requirements management and cost estimation. Finally, in Section 7 we wrap up with some concluding remarks.

2 Modelling Traceability According to the Product and Process

The conceptual models of the product and process adopted by a software development organization can be exploited to identify, organize, situate and maintain various kinds of traceability relations. Such situated traces can either originate from semantic inter-artifact relationships explicated by the product model, or from the data flow described by the process model. The former – which we call *product-based traceability* - can span from user (change) requests to user and system requirements (pre-requirements specification traceability), onward to software artifacts that satisfy the requirements (post-requirements specification traceability). The latter – which we call *process-based traceability* - can cover the whole development (and maintenance) process, as well as the management process.

2.1 Product Models and Traceability

The product model can be defined as the conceptual schema capturing the software artifacts, such as requirements, design documents, source code, etc. Among the various types of software artifacts, numerous significant relationships exist, such as aggregation, composition, specialization, dependency, etc. Those relationships are themselves part of the product model, and of crucial importance for any software development effort. Product models can be explicitly represented using the same techniques that have been proposed for conceptual data modeling.

Much as for the product model, also the traceability information for a given product exists inherently, but must be described explicitly to be of any use. Traceability is often described, established and maintained separately, even when the product model is well formalized. Instead, we see product-based traceability as derivative with respect to the relationships in the product model: traces originate from requirement artifacts and traverse the various relationships they are involved into, to include other artifacts, which in turn are connected by other relationships to more artifacts, etc. Thus, the product model is in a sense a way to make traceability relationships explicit.

Potentially, all relationships in a product model can provide valid traceability information. In order to avoid information overkill, only a subset of the very complex network of relationships among the product artifacts is employed in practice to establish essential traceability. For example, dependency relationships are often the most common source of traceability.

In order to support our approach, the product model must have the following properties:

- enable the description of artifacts both at the type and at the instance level;
- enable the definition of attributes characterizing the various artifact types;
- enable the definition of multiple types of relationships among artifact types, with different semantics, some (or all) of which constitute the basis for product-based traceability;
- enable the instantiation of relationships holding among artifact instances;
- allow to vary the level of abstraction and granularity of the product description, according to the amount and precision of available traceability information.

We employed UML [5] class and object diagrams to capture product-based traceability, since they comply with all the requirements set above. For example, they support specialization and decomposition, hence allow product modelling at various levels of abstraction: in requirements management, switching among abstraction levels is important to get an adequate view of the change, either by hiding unnecessary details or by focusing on important ones. A product models expressed in UML can include only a few generic and coarse-grained artifact types representing the major by-products of software development, with simple traceability relationships holding among them; but the same model can be incrementally detailed by including sub-types and sub-components of the fundamental types, thus providing more comprehensive and refined traces.

Moreover, UML class diagrams allow the definition of attributes that characterize the various artifact types and can be used for measurement, as explained in Section 3.

2.2 Process Models and Traceability

The process model captures the way software development is carried out in an organization. It describes the activities that are performed over the software artifacts by means of various kinds of resources, among which human resources have paramount importance. Modelling a process as complex as the software process is not trivial: a large number of diverse formalisms have been conceived [12] in the attempt to capture all of the various facets of possible interest (data flow, co-ordination, synchronization, control, collaboration, dependencies, deviations, monitoring and management, etc.).

In the context of requirements and change management, the process formalism must primarily relate process activities to the artifacts they produce: in other words, the focus is essentially on the *data flow*. Notice that the product and process models must be mutually consistent: every artifact type described in the product model must appear in some activity model as an input and/or output, and vice versa.

We are also interested in quantitative factors related to the activities, such as cost factors. For this reason, we consider the *model of process resources* as part of the process model itself (resources can be easily modelled in the same way as artifacts).

One of the process formalisms that suit our needs is IDEF0 [1]. In IDEF0, activities are related through the flow of their input and output, and resources employed by the various activities are explicitly described with their properties; moreover, activities can have properties of their own. IDEF0 also allows modelling of the process control flow, but this kind of information can be omitted for our purposes. An activity model according to IDEF0 is shown in Figure 1.

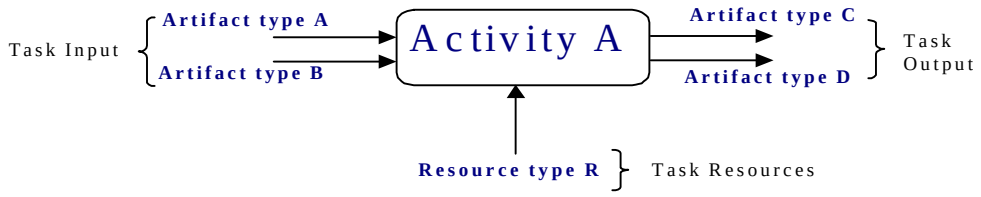

Figure 1: Example of a process activity in IDEF0.

Describing the data flow provides process-based traceability, which enriches more traditional product-based traceability in two ways. First of all, it offers a more comprehensive view over traces connecting artifacts, emphasizing their underlying causality: traceability relationships often derive from the fact that the connected artifacts are involved in the same process activity. This situation is exemplified in Figure 2, in which the presence of the “affects” traceability relationships is motivated by the fact that the various artifacts are input and output to the process activities. Furthermore, the dependencies among artifacts due to the data flow may in some cases produce traceability relationships that were not per se identified in the product model.

In these cases, product-based traceability complements and completes product-based traceability.

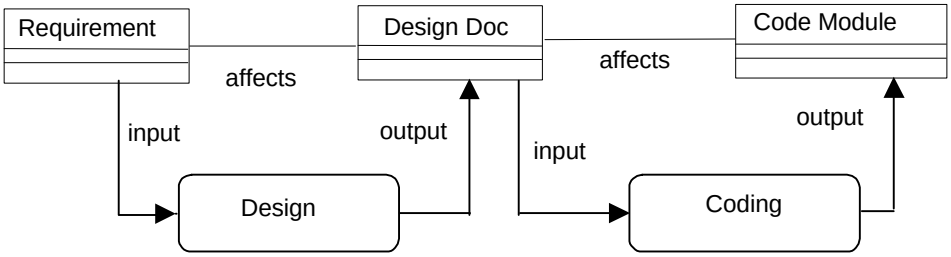

Figure 2: Traceability established on the basis of the product and process model.

In all cases, process-based traceability provides added value, since it allows to take into account process knowledge for the analysis of requirements change: it relates a change to the corresponding work through its impact. For instance, in the simplistic example of Figure 2, it is evident from the integrated product and process models not only that a change to Requirements has consequences onto Design Documents and Code Modules, but also that to implement that change some work must be performed in the Design and Coding phases of the software life cycle.

The amount of necessary work depends on a number of factors, typically characteristics of the artifacts (e.g., size and complexity) and of the activities (e.g., how the work is done, what kind of resources are employed, …). Proper identification and measurement of these characteristics leads to a quantitative perception of software change (see Section 3).

Much as for the product model, abstraction and granularity issues apply also to the process model: each activity can be refined in more detailed sub-steps, or activities can be aggregated to hide unnecessary details. Increasing detail and compliance to the actual development practices leads to refined requirements management capabilities. However, there is a clear trade-off with respect to the effort necessary to represent and maintain such richer process information, including traceability, characteristics of the activities, and measurements.

3 Measuring the Development Process for Quantitative Requirements Management

The goal of measurement is to characterize the elements of the models in order to support quantitative analysis. We must consider each of the process activities and determine what attributes satisfactorily describe it; i.e., we have to identify a set of attributes for:

- every input type;
- every output type;
- every type of resource used;
- the activity itself[3].

[3] Attributes concerning the activity are typically used to take into account variations in the way the activity can be performed (e.g. development techniques, tools, etc.).

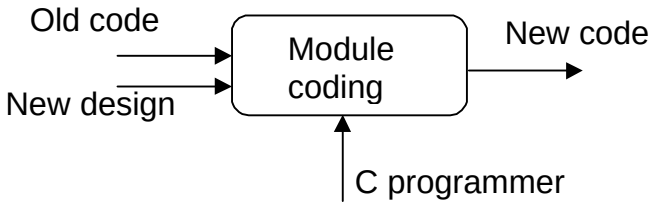

Figure 3: An activity of the process model.

Considering the example activity of Figure 3, relevant measurable properties can be:

- code features (size, complexity, …);
- resource profiling (experience, knowledge of domain, …);
- design properties (size, complexity, clarity, …);
- activity features (programming environment, coding conventions used, …).

The definitions above are still generic, and have to be properly detailed/customized. For instance, if the design notation in a project is UML, the design size could be given by the number of classes, states, etc. contained in the UML diagrams. Our methodology promotes the definition, collection and exploitation of project-specific metrics, customized and tuned on the peculiarities of the project. On one hand this increases the adequacy of the measures; on the other hand it limits the generality of metrics, the corresponding metrication procedures and their results: their reusability depends on the degree of analogy with other projects and different development practices.

In order to identify suitable sets of measurable attributes, we propose the GQM approach [2]. However, our methodology does not impose any specific measurement approach, since it could represent a serious barrier to the adoption of quantitative requirements and change management in organizations having their measurement practices already set up: any approach that supports the definition of suitably detailed measurement plans is equally valid.

For each occurrence of every activity type (in our example, for every occurrence of a "Module coding" task) the identified properties are measured. Then, collected data is employing standard statistical methods – to derive relevant correlations. For instance, in the case of the "Module coding" activity, we could be interested in knowing how the new code size depends on the old code size and the design complexity, or how the coding effort depends on the new code size and the resource properties, and so on.

Notice that the definition of measurement plans has to be consistent with models of the process and product it refers to [14]. Since the models can be more or less sophisticated, depending for example on the maturity of the organization [16], the measurement plans can also be more or less articulated. As a consequence, the metrics campaigns can vary greatly in duration and cost. In defining the measurement plans a trade-off must always be sought between the cost and duration of the operations and the precision and reliability of the results. In order to make the adoption of our approach as fast and smooth as possible, it might be advisable for example to use at first a coarse-grained model of the process, where the activities are essentially the phases of the life cycle and only a few artifacts' properties are considered. In this way,

it is also very likely that generic pre-existing measurements collected within the organization can be re-used.

4 Quantitative Estimation of Impact and Costs

Although several kinds of analysis can be supported by our approach, we concentrate here on impact evaluation and cost estimation.

4.1 Impact evaluation

With the term "impact" we mean the consequences of a change. Such consequences involve activities to be carried out to accomplish the change, and the artifacts to be manipulated by those activities. In order to evaluate the impact we make use of the integrated process and product models (as exemplified in Figure 2).

For each impacted process activity, the analysis of collected measures allows the synthesis of "impact functions". These functions have the following structure:

$O = f(I, R, A)$

Where O, I, R and A are vectors, containing values for the relevant properties of outputs, inputs, resources and the activity respectively. For instance, for the activity shown in Figure 3, an hypotethical impact function could be:

$<Size_{nc}, Cplx_{nc}> = f(<Size_{oc}, Cplx_{oc}>, <Exp_{cPr}, DomKn_{cPr}>, <Size_{Des}, Cplx_{Des}, Clrty_{Des}>, <PrgEnv_{Mcod}, CodConv_{Mcod}>)$

where the function parameters are as follows:

- $Size_{nc}$, $Size_{oc}$, $Size_{Des}$ represent the size of the new code, old code and design, respectively;
- $Cplx_{nc}$, $Cplx_{oc}$, $Cplx_{Des}$ represent the complexity of the new code, old code and design, respectively;
- Exp_{cPr}, $DomKn_{cPr}$ represent the experience and domain knowledge of the C programmer, respectively;
- $Clrty_{Des}$ represents the clarity of design;
- $PrgEnv_{Mcod}$ and $CodConv_{Mcod}$ identify the programming environment and the coding conventions employed.

An impact function like the one above computes the properties of the output of the activity, given the characteristics of input and employed resources. Since the output of an activity is the input of another activity, impact calculations are propagated throughout the process, in order to compute the overall impact of any given change.

Note: in practice the tool developed in the SACHER project allows the user to override automatic estimations before they are used as input in other estimations (e.g. cost estimations, see Section 4.2). This also allows to employ expert judgement whenever an impact function is unavailable (e.g., because the underlying metrics do not provide reliable correlations).

Actually the tool allows the user to complement/correct the evaluations of the tool at every step of the computation. In fact, especially when the model is of relatively coarse granularity, it may happen that some instances of the product or some instances of the activities behave differently with respect to the general abstract model. Their

peculiarity is generally known by the user, that can thus correct the evaluations made by the tool before they are propagated (i.e., used as input of other evaluations).

4.2 Cost estimation

Cost estimation works much like impact evaluation. At the activity level, the analysis on collected measures - including effort - allows the synthesis of "cost functions". These functions have the following structure :

C = f(I, R, A, O)

where C is a vector defining and expressing the cost of carrying out the activity[4]. Note that the vector O regarding the properties of the output, whose values may have been derived via an impact function, is employed here as a parameter to the cost function.

Cost functions provide an estimation of the cost of each activity instance that must be carried out. In general, to respond to a proposed change multiple instances of the various process activities must be performed, each one working on some portion of the software product (for example, multiple Module coding tasks for different impacted modules). In order to assess the overall cost of the change, it is sufficient to compose the cost of the individual activities involved. Partial roll-up of costs according to the various activities also allows to identify the weight of each software develoment phase with respect to the total cost of the change, which is relevant for project management.

5 Implementation and Validation

The methodology presented above has been embraced and experimented in the SACHER project, whose objective is providing a requirements management environment that enables quantitative impact and cost estimation, analysis of software change sensitivity, and enhanced support to software development and maintenance contracting.

5.1 A SACHER Scenario

A scenario for employing the SACHER environment and the underlying methodology is informally represented in Figure 4, in which thin arrows represent the flow of information and thick arrows represent logical correspondences.

The SACHER environment requires some set-up:

1. The process and product models for a given project are defined.
2. Impact and cost functions are included in the models. This implies that measurement data consistent with the models are available: a measurement campaign is required if data are not available. The measurement process can be carried out employing the GQM technique and tools.
3. The product model is populated with instances (i.e. model-compliant representations of the project artifacts).

[4] Cost can be composed of different elements, depending on the specific model. However, effort is usually the main component.

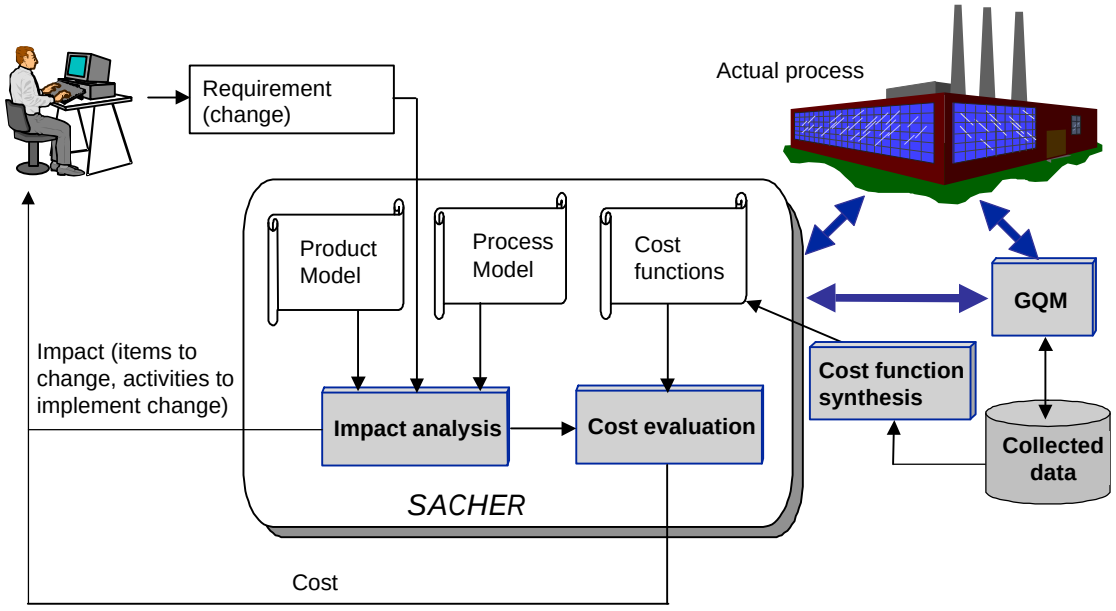

Figure 4: The SACHER environment at work

The environment can then be used for different purposes:

- Impact evaluation:
  1. Whenever the procurer asks for changes in requirements, these are formalized by the supplier and injected into the system; this involves establishing pre-requirements specification traceability from the proposed changes to the stored requirements.
  2. The system exploits the model-compliant traceability information to compute the impact of the change, in terms of artifacts to be modified, and activities to be carried out.
  3. The estimated impact is quantified through the available impact functions, whose parameters are filled with values collected from the involved artifacts, or supplied by the user.
- Cost estimation:
  1. The quantitative characteristics of the impact are used as parameters of the cost functions of the various process activities involved in the change, obtaining an estimation of the cost of carrying out the corresponding work. Costs of the individual process activities are rolled up to obtain the overall estimated cost for the work package implementing the proposed change.
  2. all estimation results are provided to the user, and also recorded in the measurement base.
- Estimation tuning. Recorded estimates and actual results are compared a posteriori: the impact and cost functions are then refined in order to take into account possible differences. For every change an estimation tuning cycle is performed, and impact and cost functions are progressively improved.

Notice that the scenario above is only an example. For instance, it does not cover the benefits of SACHER related to the procurer-supplier negotiation, or to project management.

Another consideration is that different kinds of analysis can be defined on top of the models and for some of them the cost evaluation part is optional: for instance, to evaluate the sensitivity of the system with respect to change, it is sufficient to evaluate impact from various technical points of view.

Finally, we believe that our approach can be applied not only to modifications of existing products, but also to new developments. In this case it is required that the user supplies the models of artifacts that will be developed. Experimenting with this way of using the proposed technique will be the object of future work.

5.2 Implementation of the SACHER environment

The SACHER environment for quantitative requirements and change management is built upon the DOORS [15] system by QSS Inc.[5], one of the market leaders among the requirement management tools.

On top of DOORS generic data modelling facilities (named *formal* and *link modules*) and programming facilities (the proprietary DXL language) SACHER provides a well-organized structure to define and accommodate product and process models, put in place and maintain product-based and process-based traceability, record impact and cost functions with their parameters. All of this information is stored in specific DOORS modules, which are reserved for modelling purposes and have precise semantics for SACHER[6]. Other DOORS modules are used to populate models with instances. Upon this semantically rich data structure, SACHER implements interactive mechanisms to define and inject requirement changes, retrieve, explore, and quantify the corresponding impact over the product and the process, and perform, report and analyze cost estimations.

SACHER also exploits DOORS import facilities (for instance the possibility of extracting information which is relevant for traceability management from UML diagrams) for populating automatically the product models.

5.3 Validation

In order to validate the methodology as well as the SACHER environment, we have launched three independent initiatives, pertaining to industrial software development efforts. All case studies are conducted within real production environments. Hereby, we present some preliminary but nevertheless indicative results.

5.3.1 An case study in real-time software development

This case study is related to a complex medium-sized software produced by a Small-Medium Enterprise[7] (SME) entertaining a long-lasting procurer-supplier relationship with its customer, who requires regular updates to the software. The application

---

[5] QSS Inc. is a partner of the SACHER consortium.

[6] Basic DOORS modules have a very limited semantics: link modules represent relationships, while formal modules represent collections of objects that can be arranged in aggregation hierarchies. It is the user that has to define the meaning of the objects and relations. Instead, SACHER provides the user with ready-to-use structures and operations.

[7] The name of the enterprise is not reported because of confidentiality issues.

domain is very stringent in terms of quality requirements and the supplier must abide to a fairly traditional software process standard. The major concern of the supplier is to correctly evaluate the cost of carrying out each proposed update, in order to optimize and motivate its offer in the contracting phase.

At the moment we have completed a first round of the estimation cycle. We have identified through GQM plans measures relevant to cost estimation and we have collected the corresponding data concerning two different baselines of 11 major components of the software product throughout the development life cycle. According to the SME management the effort for collecting such data amounts to around 2% of the overall effort, which is considered acceptable in project management terms.

Our metrics campaign has produced a number of measurement points that is sufficient for synthesising cost functions with sufficient statistical reliability for 4 of the 7 high-level process phases whose contribution to costs the SME management is keen to measure. Further data are being collected, in order to provide sufficient statistic reliability for the cost functions of the remaining activities.

The average estimation error of the four cost functions currently in place varies between 25% and 30%. The variance of the estimation error is also acceptable: error is reasonable in around 70% of cases.

The synthesised cost functions are being employed for estimating the costs of upcoming developments and the experiment is ongoing. We consider these preliminary results encouraging for various reasons:

- The accuracy of our methodology seems comparable to that of established cost estimation techniques and practices; for example the estimation error of the first versions of COCOMO was less than 25% in 70% of cases.
- The data we collected concerns high-level phases (e.g., design and coding) and large subsystems. This is due to the need of exploiting existing data, that were traditionally collected at the level of the life-cycle phases. According to the SME management subsystems contain rather etherogeneous elements (e.g., of different complexities) and phases are made of quite different activities. Our models represent the average behaviour of phases and subsystems. Individual changes to a subsystem can involve a subset of the components more complex than the average of the subsystem. Of course in such cases our model underestimates the effort needed to accomplish the required changes. We are confident that by collecting more detailed measures we will be able to improve the precision of the estimations.
- Our approach is incremental, and supports the convergence of estimations and actuals over time. We are confident that the second and further rounds of estimation tuning will carry increasingly better performance.
- Once the estimation tuning cycle is in place, the development organization completely owns the methodological approach and its results; that is, it is able not simply to obtain quantitative estimations, but also to motivate its findings, evaluate the levels of accuracy attained, identify shortcomings, perform correcting actions, enhance the cognitive dimension of its requirements practices.

5.3.2 Other ongoing case studies

Other case studies, selected with the purpose of maximizing the coverage of the various aspects addressed by the methodology, are currently taking place.

One case study is related to the same application domain as the one presented in Section 5.3.1 but on a different scale. Moreover, it concerns multiple derivative developments within a product family, required to the supplier by a variety of procurers. The major concern of the supplier is evaluating the requirements of a new product with respect to the various existing elements in the product family, in order to gain a competitive advantage through maximized software reuse and optimized bids.

Another case study is rooted in a different application domain, related to the provision of advanced information systems within a multinational ICT company. The procurer-supplier chain is internal to the organization, which implies a peculiar way to negotiate and define requirements, and encourages the adoption of a prototypical process model. Since the evolution of the product is particularly fast-paced, the major concern of the supplier in this case is the ability to estimate costs in conjunction with the time-to-market of new or modified functionality.

In both cases, the approach was successfully used to defined models and to perform traceability-based impact evaluation. Nevertheless, since the organizations and projects involved are larger and little reusable data was available, the collection of data took a longer time, and we are right now entering the phase in which the collected data and the SACHER prototype are used to provide estimations about upcoming developments.

6 Related work

Our approach and the derived tool set compare to two areas of research and development, which are traditionally disjoint.

6.1 Requirements management

Commercial requirements management tools [11] generally provide very simple schemas for requirement data types (see for instance the formal modules and link modules of DOORS). The only built-in structuring mechanism is often a decomposition hierarchy, which is given no precise semantics and constraints (e.g., a requirement can “contain” the source files that implement it). The querying and navigation facilities are consequently limited: since the semantics of relationships, artifacts and attributes is not known by the system, substantial programming of the tool is required in order to perform non-trivial analysis.

Research environments like TOOR [17] support the definition of requirements, other artifacts and the relationships linking them in an object-oriented style, and allow browsing and querying requirements-related data.

PRO-ART [19] also supports sophisticated structuring of traceability information according to different models (e.g., hypertext, entity/relationship, structured analisis).

Our approach is similar to the mentioned ones as far as modeling and representing requirements-related information is concerned. In fact it allows to establish accurate, customized models before populating the environment with data, so that data

consistency is inherently guaranteed and sophisticated queries are possible. Moreover, libraries of models are made available to users, who can derive from them their customized models. However, our approach also allows to perform different kinds of analysis concerning quantitative aspects of requirements changing, while the mentioned tools concentrate on representing, maintaining and consulting traceability information or (as PRO-ART) on automatic collection of traceability information.

6.2 Cost estimation

Our approach differs from other well-known cost estimation techniques essentially because of the underlying process model. For instance, both the COCOMO [4] and Function Points methods derive their estimation models from "black-box" process models. I.e., they derive correlation between characteristics of the box that can be observed from the outside (e.g., lines of code or function points as parameters and effort or duration as results). This implies that - since very different processes can be hidden within the black box - there is the risk of mixing oranges and apples. It is therefore necessary to classify the black boxes according to the type of process they contain and derive different estimation models for every process type. This is exactly the approach that the COCOMO research group is currently pursuing: they are producing variants of COCOMO, which apply to specific "families" of development processes [13]. For instance, COCOTS deals with developments exploiting commercial off-the-shelf (COTS) components, while CORADMO focuses on the cost of developing software using rapid application development techniques.

We overcome this problem by adopting a "transparent box" approach. It is the users who describe their own process model. The model can be more or less detailed (i.e., it can represent activities at different granularity levels) and extended (e.g., it can include just technical activities or management activities as well). Of course, there is a price to pay for achieving a customized estimation model, i.e. the effort required to build models, to collect metrics and to derive cost functions.

Independently from the nature of the development process, the presented approach allows to compute different kinds of impacts. For instance, it is possible to assess the expected amount of bugs generated as a consequence of a change in design: this computation only requires that bugs are represented as an output of the activities impacted by design changes, and that the corresponding metrics are collected. Notice that the COCOMO research group proposes a specific model (COQUALMO) exactly for predicting the density of residual defects in a software product.

A relevant advantage of our approach is that it allows evaluating the consequences of any change (e.g. occurring in design, in requirements specification, etc.). This means that one can compute the consequences of changes in requirements specifications, and later, e.g. when the design has been updated, can estimate the cost of implementing the updated design, and so on.

6.3 Integration of the two domains

Finally, an interesting remark is that our approach is quite original in merging requirements and traceability management on one side and cost estimation on the other side. In principle, this enables users to estimate the cost of maintenance actions starting directly from the description of the proposed changes. Of course, in practice this is

made difficult by the need to quantitatively characterize the requirements, which is a tough task. However, experienced users who know well their application domain can detect relevant measurable attributes of requirements, which can provide a sound basis for the estimation.

7 Conclusions

The methodology presented provides quantitative estimation of software requirements changes, including impact and cost analysis. We propose an approach that is founded upon traceability in conjunction with knowledge of and compliance with measurable models of the software product and process.

Our approach extends requirements engineering/management and cost estimation techniques and integrates them with respect to each other. It greatly emphasizes the importance of capturing and taking into account the peculiarities of software development organizations and projects.

On one hand our work is oriented to the "high-end" organizations described by Ramesh [18], as it relies on a view of traceability practices as a way to improve the development process, and supports "full coverage of life-cycle, capturing traces across product and process dimensions, …", and finally it helps in developing the "ability to develop realistic cost proposals and gaining competitive advantage". On the other hand, the benefits provided by our approach could motivate "low-end" organizations to improve their traceability information management practices.

The SACHER project has embraced our methodology and aims to comprehensively support it, with a tool set built as an extension of a leading requirements management system, namely DOORS. An advanced prototype is already available to the SACHER consortium and is the object of intense validation initiatives. Preliminary results indicate that the proposed approach can provide added value in the technical domain, as well as in managing a project and in development and maintenance contracting.

Acknowledgments

We thank our colleagues at CEFRIEL and Politecnico, especially Alfonso Fuggetta and Carlo Ghezzi for their suggestions.

Thanks also to the SACHER project members, and to Luciano Aprile who built several product and process models.

8 References

[1] Integration Definition for Function Modeling, *IDEF0*. Federal Information Processing Standards Publications, December 1993.

[2] V. Basili, H.D. Rombach, The TAME Project: Towards Improvement-Oriented Software Environments, *IEEE Transactions on Software Engineering*, 14 (1988), p.758-773

[3] Software Engineering Institute, *The Capability Maturity Model*. Guidelines for Improving the Software Process, Addison-Wesley, 1995

[4] B. W. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.

[5] J. Rumbaugh, I. Jacobson, G. Booch, "Unified Modeling Language User Guide", Addison-Wesley, 1998.

[6] P. Hsia, A.M. Davis, D.C. Kung “Status Report: Requirements Engineering”, IEEE Software, 10(6), November 1993.

[7] O. Gotel, A. Finkelstein “An Analysis of the Requirements Traceability Problem”, in Proceeding of the First International Conference on Requirements Engineering, Colorado Springs, CO, 1994, IEEE Computer Society Press, pp. 94-102.

[8] R. Domges, K. Pohl “Adapting Traceability Environments to Project-Specific Needs”, in Communications of the ACM, Vol. 41, No. 12, December 1998, pp. 54-62.

[9] M. Jarke “Requirements Tracing”, in Communications of the ACM, Vol. 41, No. 12, December 1998, pp. 32-36.

[10] B. Ramesh, C. Stubbs, T. Powers, M. Edwards “Requirements Traceability: Theory and Practice”, in Annals of Software Engineering No. 3 (1997), pp. 397-415.

[11] The International Council on Systems Engineering (INCOSE), “Tools Survey: Requirements Management Tools”, http://www.incose.org/tools/tooltax.html, April 1999.

[12] P. Armenise, S. Bandinelli, C. Ghezzi, A. Morzenti, "A Survey and Assessment of Software Process Representation Formalisms", International Journal of Software Engineering and Knowledge Engineering, Vol.3, no.3, 1993.

[13] Cocomo research group, the COCOMO project home page, http://sunset.usc.edu/COCOMOII/cocomo.html

[14] L. Lavazza, “An integrated view of process modelling, improvement and *ICSE 97 workshop on Process Modelling and Empirical Studies of Software Evolution*, Boston, May 18, 1997.

[15] R. Stevens, "The DOORS Dynamic Object Oriented Requirements System", http://www.qss.co.uk/library/doors.html

[16] S. L. Pfleeger, “Lessons learned in Building a corporate metrics program,” IEEE Software, vol. 10, no. 3, pp. 67-74, 1993.

[17] F. A. C. Pinheiro and J. A. Goguen, “An Object-Oriented Tool for Tracing Requirements” IEEE Software, vol. 13 n. 2, March 1996, p. 52-64.

[18] B. Ramesh, “Factors Influencing Requirements Traceability Practice”, in Communications of the ACM, Vol. 41, No. 12, December 1998, pp. 37-44.

[19] K. Pohl, R. Dögmes, M. Jarke, “Towards selective, model-driven trace Proc. Int. Conf. On Advanced Information Systems Engineering, Barcellona, 1997.