Requirements-based Estimation of Change Costs

LUIGI LAVAZZA
CEFRIEL and Politecnico di Milano

GIUSEPPE VALETTO
CEFRIEL

Abstract. We present a case study that aims at 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 an integrated view of the process and product models. The elements in the process and product models are quantitatively characterised through proper measurement, thus achieving a sound basis for different kinds of sophisticated analysis concerning the impact of requirements changes and their costs. We present the results of the application of modelling and measurement to an industrial project dealing with real-time software development. The ability to predict the impact of changes in requirements and the cost of related activities is shown.

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

1. Introduction

Both software development and maintenance are characterized by frequent changes in user requirements. These changes are normally formalized in Change Requests (CR). In order to keep the software development or maintenance process under control it is extremely important to be able to assess the consequences of CRs, in terms of the amount of artifacts to be modified, the amount of testing to be performed, etc. In particular, it is important to be able to estimate the effort required to carry out the required change.

In the normal practice the consequences of a CR are best assessed by an "expert" who knows the product to be changed and the process employed to carry out the required modifications, since the cost of maintenance depends on the features of the product and of the process.

We propose an approach that:

- Exploits the knowledge of both the software product and the process employed to develop and modify it. This knowledge is expressed in terms of models of the process and artifacts (by artifact we mean any piece of software or document produced, used and/or modified during the development process). The proposed approach applies to development from scratch as well as to maintenance. In the case of maintenance the product already exists, thus it is easier to build a reliable model; nevertheless, it is also possible to devise models of artifacts still to be built.
- Defines the quantitative characteristics of the process and product through proper measurement, so that the aforementioned models are quantitatively characterized.
- Takes as input CRs, i.e., it operates at the requirements level.

The latter point makes our approach quite different from traditional methods, which are based on measures such as lines of code (Boehm, 1981) or function points (Albrecht, 1979). In fact these measures are defined on features of artifacts (the code and the specifications, respectively) that do not yet exist when the CR is issued, and thus have to be estimated. We adopted a more radical approach, trying to use the quantitative characterization of requirements and requirements changes as a base for assessment. In this way, we can base

our estimations on the CRs themselves, without having to guess how many SLOCs or FPs will be changed or added.

Our approach was made possible by the features of the software product being considered in the case study, where requirements are described thoroughly at a quite fine granularity, traceability relations are represented and maintained carefully, and the product is well modularized, so that every functional requirement is implemented by only a few procedures. We simply exploited the requirements structuring and classification practices already used in the considered project. In fact, pieces of requirements were labelled, and labels used for recording traceability relations. We just had to count labels, in order to have a rough (yet effective) measure of requirements. In other contexts (especially where a requirement can be traced down to several artifacts) this practice could prove inapplicable.

The paper describes an experimental application of the approach mentioned above. The rest of the paper is organized as follows: in Section 2 the process and product models of the pilot project are illustrated; Section 3 describes in some detail the metrics that were used, while Section 4 illustrates data analysis and the resulting impact and cost models. Section 5 reports results and lessons learned. Section 6 accounts for related work. Section 7 draws some conclusions and outlines future work.

2. Modelling the process and product

We used as a test-bed an industrial project aiming at the development of real-time, safety-critical software, being carried out at TXT Ingegneria Informatica (http://www.txt.it/). The detailed characteristics of the project are confidential.

2.1 Product model

The product is organized into “units” (there are about ten units in the product). Every unit is itself decomposed into up to four subsystems, each implementing a set of related functionality. This kind of modularization of the product affects the process and the way it is organized. For instance, every unit is assigned to a programmer (or to a small team of programmers). *Figure 1* reports a UML class diagram illustrating the main artifacts involved in the process and their relationships:

- Requirements. The complete set of requirements of every product release is documented. Each requirement is a single functional or non-functional feature of the product, as indicated by the procurer.
- Design Document. It defines the design of every unit. It includes both high-level design and detailed design. Every unit is organized into subsystems: each subsystem is a software deliverable that can be tested against requirements. Each subsystem is further decomposed into modules.
- Source Code files implement the specification of a module. They collect a set of coherent procedures.
- Test Design Document. It describes the testing strategy and procedures for the corresponding unit. It is composed of a high-level description of the testing procedure for the unit it refers to, and the whole set of test cases which must be carried out on the units. The entire set of test cases is subdivided into Module Test Cases and Subsystem Test Cases. Test reports contain the pass/fail state of the tests for the corresponding unit, as well as the number and types of bug found (and fixed) during the testing.

Requirements are univocally identified by means of labels. A database application specifically developed in TXT maintains the relations of labels with artifacts in the lower phases of the development process, specifically with: procedures, module test cases, and sub-system test cases.

Currently, product traceability information is thus focused on the coding and testing artifacts and does not address directly the by-products of the intermediate phases of the

process (e.g. Design artifacts). Nevertheless, it is still possible to trace a requirement to a design document, with some effort.

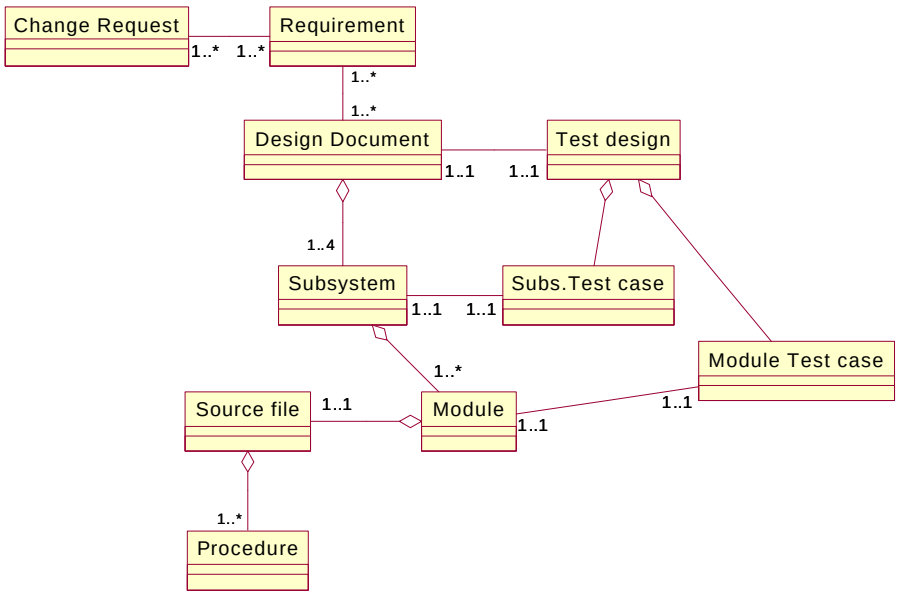

Figure 1: Artifact types involved in the software development process.

2.2 Process model

We rely on a relatively detailed model of the process activities carried out in order to build a new release of the product, according to new or changed requirements. For our case study we had to rely on existing data, i.e., no specific measurement process was carried out in order to support our work; as a consequence, we had to adopt a simplified model, consistent with the available data.

The actual process model of reference is shown in Figure 2. A fundamental feature of our models is that their elements (i.e., activities and artifacts) must be quantitatively characterized. Since we had no data characterizing design documents, we decided to collapse design and coding into a single activity.

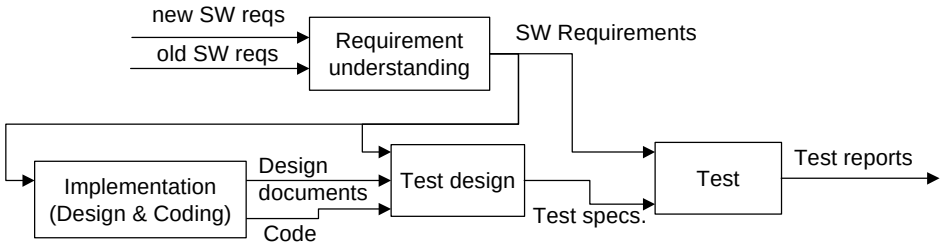

Figure 2. The process model actually used in the case study.

All of the above activities are regularly measured as part of the development process at TXT. The detailed description of the available metrics is reported in Section 3.

2.3 Impact and cost functions

In order to clarify the role of impact and cost functions, let us consider the implementation activity described in *Figure 3*.

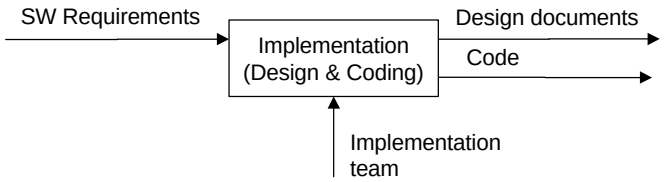

Figure 3. The implementation activity.

Our objective is to estimate the effort required to carry out the activity. It is reasonable to expect that such effort will depend on the characteristics of the inputs, outputs, and resources associated to the activity. Of course, when we start the estimation for this activity we do not know the characteristics of the outputs, while in general the characteristics of the inputs are either known or they have been estimated as the output of previous activities.

Therefore, we organize the estimation in two steps:

- Characteristics of the outputs are estimated on the basis of inputs and resources. For instance, the size and complexity of the code are estimated on the base of the size and complexity of the requirements, and on the skill and experience of the implementation team. The estimation is based on the knowledge of the relations that link inputs to outputs.
- The effort is estimated taking into account all the relevant characteristics the activity, including the expected characteristics of the outputs, as computed at the preceding step. The estimation is based on the knowledge of the relations that link inputs, outputs and resources to effort.

The mentioned relations are derived by means of statistical analysis on previously collected data. We call such quantitative relations impact and cost functions, respectively.

3. Metrics

3.1 Goals and definitions

The main goal of the proposed approach is the early estimation of the overall implementation effort due to new/changed requirements, computed on the basis of the cost of the activities that compose the process (as described in *Figure 2*).

The metrics for the process and product model described above were defined according to the GQM method (Basili *et al.*, 1994). However, TXT management required to employ only existing data, rather than carrying out a full-fledged GQM process. Nevertheless we used the GQM plan (not reported here for space reasons) to provide a rigorous framework for the interpretation of the existing data.

3.2 Available metrics

Available metrics concerned 21 unit releases, characterized by the following ranges of values:

- Total number of requirements: 17-499 per unit;
- Number of updated requirements per unit release: 2-72;
- Unit size in SLOCs: 1158-22184;
- Total effort: 6-99 persondays.

Historic data concerning the activities of the process reported in *Figure 2* were available. In particular, the following attributes of the activities and of the employed resources were represented:

- Effort spent to carry out each activity;
- Number of resources employed to carry out each activity;

- Skill of personnel employed to carry out the activity (subjective measure, given by the manager of the employed people);
- Level of knowledge of the application domain owned by the personnel employed to carry out the activity (partly subjective measure, based on the experience in the field).

Historic data concerning the product artifacts were also available. In particular, every release of every unity of the product was described by the following data :

- Number of updated requirements.
- Total number of requirements before the changes.
- Number of modified/added SLOC.
- Number of deleted SLOC.
- Complexity of each unit (subjectively evaluated by the project leader).
- Number of SLOC for the involved unit before the changes.
- Type of error raised during the internal test (classification of errors).
- Required level of target quality (it refers mainly to documentation).
- Required level of testing coverage.

Given the above list of available metrics, it is clear that some artifacts are not well characterized. For instance, we had no data whatsoever concerning design documents. We managed this situation by considering only activities whose inputs and outputs were well characterized by the available data. The activities satisfying this requirement are those reported in *Figure 2*.

4. Derivation of cost models

In this section we describe the derivation of the impact functions and cost functions. We also describe how these functions are used to compute the impact and cost of changes.

4.1 Feasibility study

Before proceeding to quantitatively characterize (through measurement and analysis) the different activities of the process we performed a little feasibility study. The goal of this study was to understand whether the hypothesis of basing the impact and cost functions on measures of requirements was viable. In fact it is well known that measuring requirements is often very difficult, since requirements can be expressed in a great variety of ways, with different levels of details, precision and abstraction.

Since we know from COCOMO and other studies that several characteristics of the software process and products (such as effort to develop, faultiness, etc.) are related to the size of the products, we investigated whether the size of our software units was somehow related to the “amount” of the corresponding requirements.

TXT made available measures concerning the overall number of requirements implemented in every release of every unit of the system[1]. It was therefore possible to verify whether there is a logical dependency of the size of the product on the number of requirements. The constraint to use existing data prevented us from adopting more precise metrics (such as the size –in text lines– of each requirement).

We discovered a good (exponential) correlation between the number of requirements and the size of the units. In order to further improve the correlation we took into account the complexity evaluated at the unity level. Again, the constraint to use only existing data prevented us from evaluating the complexity of each requirement. The result was a function that estimates the size of a unit given the number of requirements and their overall complexity with an error[2] less than 19% (the estimation errors being less than 30% for 70% of the unit releases).

On one side this is a good result, since it proved that we were reasoning on a sound basis. On the other side, it is clear that the imprecision of the relation that links the number of requirements with the size of code will be present (possibly amplified) also in all the other estimates, which partly rely on this fundamental underlying relation. In order to get better results we would need to collect more metrics concerning the nature of requirements (e.g., individual complexity of requirements). This is an objective of future work.

4.2 Impact and cost functions

In this section we describe the feature of the impact and cost functions. Since the data we used is confidential, we do not report the coefficients of the functions we derived. We believe that this is not a big limit, since the actual values of the coefficients is meaningful in the environment where measures were collected, i.e., TXT, while would not be easily usable outside this environment. In fact our approach is not meant to provide generally applicable results; instead it is tailored on a single organization or process.

The size of the development team was not considered, even though this is known to be an important factor for determining productivity, because of the communication overhead. In our particular case requirement changes were performed by "teams" composed of one or two persons, thus making this variable not much relevant. In the derivation of the following cost functions, also the complexity of code was not taken into account, because the available data refer to whole units, while we would have been interested in the complexity of the portion of code actually affected by a requirement change (one to four procedures, out of the several tens that compose a unit).

Cost function for the requirements understanding phase

We found a good linear correlation of effort employed to understand the requirements of a CR with respect to a set of independent variables including:

- Total requirements;
- Number of updated requirements;
- Size of the code;
- Level of knowledge of the application domain.

The average error was less than 32% (less than 40% in 66% of cases). This is an acceptable prediction error, also considering that the effort dedicated to this phase is small with respect to the overall effort required by the project.

An interesting fact is that the contribution of the total number of requirements to the effort is negligible, while the contribution of the code characteristics is relevant. This means that the mental process required to understand new requirements takes into account *the implementation* of the previous requirements instead than the requirements themselves.

Impact function for the implementation phase

The implementation phase (represented by activity "Implementation (Design & Coding)" in *Figure 2*) yields two types of results: code and design documents. No data were available concerning design documents, thus we decided to adopt measures concerning code alone as representative of the whole activity results.

Therefore, impact function for the implementation phase correlates the size of the change (in terms of lines of code added, modified or deleted) to the number of change requirements.

We found that there exists a good exponential correlation between the number of updated/added/deleted lines of code and a set of data including:

- the number of requirements,
- the number of updated requirements,
- the size of code,

- the skill and experience of the development team.

The average error is about 28%. The linear regression yields a slightly greater error, probably because of the small size of changes –less than 10% of the total LOCs of a unit in 80% of the cases.

Cost function for the implementation phase

We found a relatively good correlation between the implementation effort (encompassing design and coding) and a set of variables including:

- the number of requirements,
- the number of updated requirements,
- the size of code,
- the number of modified (i.e., updated/added/deleted) LOCs,
- the skill and experience of the implementation team.

Linear regression can be used to estimate effort with an average error less than 27%.

We found that the effort can be predicted with almost no loss of precision even not considering the deleted LOCs. This is reasonable, since in the considered application requirements are seldom cancelled, more often they are replaced. As a consequence, deleted lines are generally replaced by others, therefore the required effort is naturally proportional to the number of new or modified lines.

Cost function for test design

The effort to be employed for designing the test cases was assumed to depend on the following quantities:

- total number of requirements,
- number of changed requirements,
- size of code,
- number of modified/added/deleted LOCs,
- skill and experience of the team.

We found a good linear correlation among the aforementioned quantities and the effort for test design. The prediction error is less than 17% (less than 30% in 80% of the cases).

Cost function for the test phase

The effort employed for testing (including both the unit-level and the system-level testing activities) was assumed to depend on the following quantities:

- total number of requirements;
- number of changed requirements;
- size of code;
- number of modified/added/deleted LOCs;
- skill and experience of the team.

We found a good linear correlation among the aforementioned quantities and the effort for system test. The prediction error is about 29% (less than 40% in 75% of the cases).

We were not able to find a correlation between the effort for unit-level testing and the set of variables reported above. This was due to the low number and quality of available data concerning effort employed for unit test.

Effectiveness of the test phase

The number of errors found was assumed to depend on the following quantities:

- effort employed for testing (including both the unit-level and the system-level testing activities);

- total number of requirements;
- number of changed requirements;
- size of code;
- number of modified/added/deleted LOCs;
- skill and experience of the team.

We found a good linear correlation among the aforementioned quantities and the number of errors found. The prediction error is about 24% (less than 30% in more than 60% of the cases).

This correlation, together with the function estimating the effectiveness of the test phase, allows to put in relation the number of requirements changes to the number of errors introduced while implementing those changes. Of course, this correlation would be fully significant only if we could consider also residual errors left over in the code at release time. Unfortunately those metrics were not available to us.

We plan to collect data (concerning the residual errors) that allow to estimate the all-important relationship between requirements changes and the total number of errors introduced during maintenance.

4.3 Combining input and cost functions

As mentioned in Section 2.3, cost functions often have arguments whose values are estimated by means of impact functions. For instance, the cost function of the implementation activity reported in *Figure 3* has among its arguments the modified/added LOCs. We applied the impact function described in Section 4.2 in order to estimate the number of modified/added LOCs, then we used the resulting values as arguments of the cost function (also described in Section 4.2). In this way we obtained an estimation of the required implementation effort having an average error around 32% (less than 40% in 60% of the cases).

5. Achievements

The results observed in the TXT case study allowed us to predict with reasonably good precision the effort/cost of major software maintenance activities. Since we were able to quantitatively correlate such effort to requirement artifacts (either new or modified), we have achieved *early* predictive capabilities that can be fully exploited for contracting as well as for project management purposes.

It is particularly noticeable that in our approach the estimation of effort is performed at the level of the single process activity. This approach features additional predictive capabilities, such as:

- Relating software size at the unit level to characteristics of requirements, as discussed in Section 4.1 (currently, only the number of relevant requirements, but in the future we plan to extract and exploit other characteristics, to attain even better precision).
- Relating the size of code changes at the unit level to the number of requirements changes, and in parallel to the point above.
- Relating effort spent in the implementation phase to the number of requirements changes. Further valuable and more precise results could derive from splitting the implementation phase in the design and coding activities, and being able to evaluate their effort in isolation.
- Relating effort spent in system testing to requirements changes.
- Relating the number of errors introduced to requirements changes. This has been only indirectly and partially achieved, and will be the object of future investigation.

6. Related work

In this Section, we relate our work to various frameworks for the estimation of software development effort/cost. We also position our work with respect to other research efforts that use observable properties of requirements to obtain a forecast of (some characteristics of) a software projects.

6.1 COCOMO

COCOMO II models (COCOMO Research Group, 1997; Boehm *et al.*, 1995) correlate effort mainly to code size. The COCOMO II Early Design model and Post-Architecture model both address issues, such as re-use and maintenance, that are tightly linked to changing requirements. COCOMO II accommodates the effect of software re-use, by adjusting code size according to the percentage of modification (which is not generally known in advance). Moreover, COCOMO II provides a way to compute a "maintenance size" that correlates to maintenance effort. The maintenance size is expressed in terms of factors, such as the size of additions and modifications to the pre-existing code base, but again the model assumes those figures are somehow known.

In our method, impact and cost functions are tailored upon a given organization or even a given project. On the contrary, COCOMO II models are derived from data originated by different organizations, and are intended to be generally valid (although the calibration of the model with respect to a specific environment can be difficult and lead to unsatisfactory results). Our method instead demands for precise product and process knowledge, together with corresponding quantitative data sufficient to statistically derive reliable correlations. The effort and time for the collection and analysis of those metrics may not be negligible, although in mature organizations the required information can be obtained at little cost. In any case, the resulting models are reliably applicable in the originating environment.

COCOMO II bases its effort estimates on the size of the code, either physical (Lines of Code) or functional (Function Points). In both cases the required information is generally not available at the time designated for the estimation, thus it has to be estimated itself. Since COCOMO II does not indicate how to obtain LOC or FP estimates, an independent and reliable estimation of size is an implicit pre-requisite. Our method employs impact functions for exactly this purpose, and can be used earlier than COCOMO II models (see Figure 4), since it is based mainly on quantitative characteristics of requirements, which are observable upfront in the development lifecycle.

COCOMO II provides a fixed set of estimates (namely effort and duration) for a fixed set of development phases. The extension of the capabilities of COCOMO II requires to extend the model, as is being done with COQUALMO, the model that estimates the number of introduced and removed errors (Chulani, 1998). On the contrary, our approach considers, estimates and combines the contributions of every single development activity in a given process. This finer granularity represents a way to account and adjust intrinsically for idiosyncrasies found in a project and its process; hence it can capture precisely the peculiar software development economics of a given organization. Furthermore, the corresponding model and formulas remain at all times completely in the ownership of that organization.

6.2 Function points count and estimation

Function Points (Albrecht, 1979; IFPUG, 1994; Behrens, 1983) based techniques can be used relying on productivity values that have been either derived outside the target organization - e.g. for a whole application domain - or computed on the basis of historical data collected within the target organization. The latter case is comparable to our approach, and requires similar set-up effort and time.

It is known that the precise count of the Function Points of a software system can be carried out only after the complete functional specifications become available (as shown in *Figure 4*), when a relevant amount of effort has already been expended. In order to better guide

contracting and project management, various approaches for early estimation of Function Points have been proposed (Santillo and Meli, 1998; Bundschuh, 1998; Tichenor, 1997). All of them exploit early/incomplete specifications (e.g., feasibility studies) for forecasting the Function Point count. This is represented in *Figure 4* by the curved arrow originating from the Requirement Analysis task. It is immediately noticeable that our method offers the advantage of earlier applicability. Moreover, estimated FP provide a more indirect path to the estimation with respect to our method.

FP-based techniques take into account the entire development as a whole. The idea of considering the characteristics of intermediate artifacts and activities, which is applied in our approach, is completely out of the principles of the FP-based techniques. In principle it is possible to correlate several qualities (such as reliability) to FPs, but this generally requires measurement campaigns that take into account the driving factors affecting the considered qualities (e.g., design methods, testing tools, features of the target platform, etc.). In summary, as for tailorability FP-based techniques are similar to our approach while their granularity is definitely coarser (and fixed).

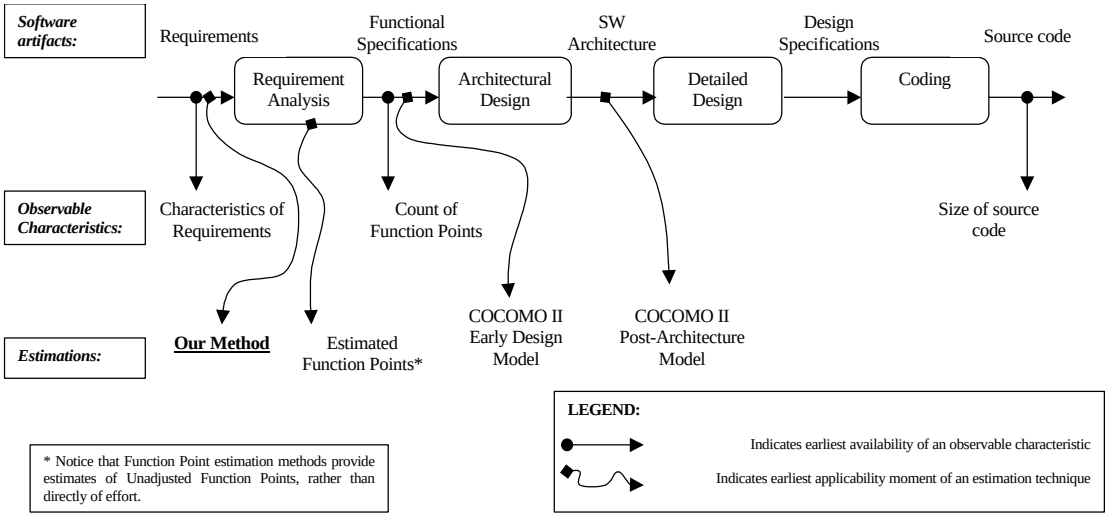

Figure 4. A comparison of various estimation approaces with respect to their earliest application moments.

6.3 MERMAID

MERMAID (Kitchenham and Kirakowski, 1990) is a method for software sizing, effort estimation, risk analysis, and other management-oriented activities, developed at the beginning of the 90's. The MERMAID approach is equipped with toolsets for effort estimation. Just like our approach, MERMAID generates local statistical models (i.e., organization or even project-specific models) based on the analysis of phase-based measurement data.

With respect to MERMAID, our approach deals more specifically with requirements (changes), while considering the characteristics of the whole development process. Moreover, our approach advocates the usage of explicit process and product models, which on one hand can describe development activities at a finer granularity than MERMAID, on the other allow a rich description of the properties of software artifacts and inter-artifact relationships.

6.4 SQUID

SQUID (Bøegh *et al.*, 1999) is a methodology for quality modelling and controlling. Both SQUID and our approach require product and process modelling throughout the development process. Both rely heavily on measures, for which they provide a rigorous definition, collection and interpretation framework. Both advocate the collection of

organisation or project specific data. However our approach is definitely oriented to cost estimation, and emphasizes the role of requirements (changes), while SQUID focuses on quality issues, and addresses quality planning and control rather than effort estimation.

6.5 Other work

Other research efforts aim at exploiting the availability of requirements early on in the software development lifecycle, in order to forecast some facets of a software project. Many of them build upon results of Requirements Engineering and Management, such as the concept of traceability (Ramesh *et al.*, 1997).

A good example is the definition of metrics for measuring characteristics of requirements, with the objective of estimating project *risks* (Hammer *et al.*, 1996). Such metrics include not only the number, size and traceability of requirements, but also requirements quality indicators such as ambiguity, (in)completeness, understandability, etc. This approach can be seen as complementary with respect to ours: those metrics could be applied in our case study, aiming to improve the reliability of estimates.

In (Meli, 199) software requirements are bridged - via traceability – to the Early Function Point Analysis (Boehm, 1981) estimation technique, in order to be able to revise the relevant risks as well as the budget of a project, whenever its requirements change. Once again, our approach is different principally due to the explicit inclusion of process and product information in the estimation process, which provides tailorability as well as finer granularity of the estimations, leading to more flexibility.

7. Conclusions

We report an experimental application of a method aiming at the estimation of the impact and cost of changes in user requirements. The method exploits models of the product being changed and of the change implementation process. These models are quantitatively characterized through proper measurement. Estimations are based on the knowledge of the change requests and of their characteristics: in this way the cost of a CR can be estimated as soon as CRs are known, thus earlier than with other methods (like those relying on lines of code or function points), which exploit information that becomes available at later stages.

We have achieved good results in the pilot project, i.e., prediction of impact and cost of requirements changes with acceptable precision. The ability to exploit information concerning user requirements was due to good requirement descriptions and good modularization. In fact, requirements were described at a constant and fine granularity, and were each implemented by a small number of procedures: these features made measures of requirements reliable and usable for estimations.

The work described here not only justified the collection of metrics and their usage in project planning, but also suggested the opportunity to collect more metrics. In fact, future work will aim at collecting new types of metrics –such as the cyclomatic complexity (McCabe, 1976) of every procedure– in order to understand whether the precision of the impact and cost functions presented above can be improved by accounting for such variables. Moreover, we adopted a very simple approach to requirements quantification: we just counted the labels placed on pieces of textual requirements for identifying requirements and tracing them to design and code elements. It is our intention to take into account also the size and structure of every requirement, in order to evaluate whether these quantities improve the precision of cost functions.

Acknowledgments

The work presented here was partly funded by the CEC as ESPRIT-IV project n. 23156 SACHER We wish to thank Alberto Salerno and Leandro Vitali of TXT Ingegneria Informatica for their cooperation.

Notes

1. In the rest of the paper the granularity of the evaluations is always the unit release.

2. Throughout the paper by “error” we mean the average absolute error.

References

Boehm B. 1981. Software Engineering Economics. Prentice-Hall.

Santillo L. and Meli R. 1998. Early Function Points: Some practical Experiences of Use. Proc. European Software Control and Metrics Conf., Rome, Italy.

Bundschuh M. 1998. Function Point Prognosis. Proc. First European Software Measurement Conference (FESMA 98), Antwerp, Belgium.

Tichenor C. 1997. The IRS Development and Application of the Internal Logical File Model to Estimate Function Point Counts. Proc. IFPUG 1997 Fall Conference, Scottsdale, USA.

COCOMO Research Group 1997. COCOMO II Model Definition Manual, available at http://sunset.usc.edu/COCOMOII/cocomo.html

Boehm B., Clark B., Horowitz E., Madachy R., Shelby R. and Westland C. 1995. Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. Annals of Software Engineering. Vol. 1 pp. 57-94, J.D. Arthur and S.M. Henry Eds. Amsterdam: Baltzer Science Publishers.

IFPUG 1994. IFPUG Function Point Counting Practices: Manual release 4.0. Westerville: International Function Point Users’ Group.

McCabe T.J. 1976. A complexity measure IEEE Trans. Software Engineering. 4: 308-320.

Behrens C. 1983. Measuring the Productivity of Computer Systems Development Activities with Function Points. IEEE Trans. Software Engineering 9:648-652.

R. Meli R. 1999. Risks, Requirements, and Estimation of a Software Project. Proc. 10th European Software Control and Metrics Conf., Herstmonceux Castle, United Kingdom.

Albrecht A. J. 1979. Measuring Application Development Productivity. . Proc. IBM Applications Develop. Symp. Monterey, USA.

Albrecht A. J. and Gaffney J. E. 1983. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. IEEE Trans. Software Engineering. 9:639-648.

Chulani S. 1998. Incorporating Bayesian Analysis to Improve the Accuracy of COCOMO II and Its Quality Model Extension. Report n. 98056 University of Southern California.

Basili V., Caldiera G. and Rombach D. 1994. Goal/Question/Metric Paradigm. In Encyclopedia of Software Engineering, vol. 1, J. C. Marciniak, Ed. Wiley.

Ramesh B., Stubbs C., Powers T. and Edwards M. 1997. Requirements Traceability: Theory and Practice. Annals of Software Engineering. 3:397-415.

Hammer T., Huffman L., Wilson W., Rosenberg L. and Hyatt L. 1996. Requirement Metrics for Risk Identifiaction. Proc. Ann. Software Eng. Workshop, NASA-GSFC, USA.

Bøegh J., Depanfilis S., Kitchenham B., and Pasquini A.1999. A Method for Software Quality Planning, Control, and Evaluation. IEEE Software. 16(2), 69-77.

Kitchenham B. A., Kirakowski J. 1990. The MERMAID approach to software cost estimation. Proc. Annual ESPRIT Conference, Brussels, Belgium.