A Shortcut to Estimating Non-Functional Requirements?

Architecture Driven Estimation as the Key to Good Cost Predictions

F.W. Vogelezang
METRI IT Benchmarking
the Netherlands
frank.vogelezang@metrigroup.com

E. van der Vliet
CGI
the Netherlands
eric.van.der.vliet@cgi.com

R. Nijland
Capgemini
the Netherlands
rene.nijland@capgemini.com

E.R. Poort
CGI
the Netherlands
eltjo.poort@cgi.com

H.R.J. Mols
Capgemini
the Netherlands
harry.mols@capgemini.com

J. de Vries
Ordina
the Netherlands
jelle.de.vries@ordina.nl

ABSTRACT

Non-Functional Requirements determine a significant amount of the cost and effort that are needed to realize or maintain a software engineering solution. Yet the effect of Non-Functional Requirements on cost and effort estimates is largely underexposed in Software Engineering research.

A few estimating solutions have been proposed but yield unsatisfactory predictive power or lack a theoretical foundation of their mechanisms. From our earlier research on packaged software estimation we have derived that the basic mechanisms that drive the estimation of cost and effort from both Functional and Non-Functional Requirements are more complex than the currently proposed methods.

In this paper we present why in most cases only Architecture Driven Estimation mechanisms can lead to good cost predictions and we explain why current estimating solutions are unsuccessful.

CCS CONCEPTS

• General and reference → Cross-computing tools and techniques → **Estimation**

IWSM/Mensura '17, October 25–27, 2017, Gothenburg, Sweden
© 2017 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-4853-9/17/10…$15.00
https://doi.org/10.1145/3143434.3143440

KEYWORDS

Non-Functional Requirements, Project Requirements, Project Constraints, Cost Estimation, Effort Estimation, Architecture, Architecture Driven Estimation, Cost driver, Size driver, Productivity driver, Size-independent cost

1 INTRODUCTION

Software system estimating approaches that are based on functional size measurement methods like COSMIC give guidance on how to estimate the cost of fulfilling NFRs [1]. COSMIC and other functional size based approaches use a factor to convert functional size to effort (in this paper, we will refer to this factor as "productivity" for ease of reading; other possible terms are "benchmark figure", "conversion factor" or similar). These estimating approaches specify that a single productivity factor should be used for each functional component that has different project characteristics, and that this productivity factor is determined by project characteristics like the NFRs and technology profiles of the project that is being estimated [1] [2]. In practice, the total size of all functional components is commonly used as the functional size of the total system and thus the project characteristics have to apply to the total functional size.

This practice presents a conundrum for estimators of many modern, component-based software systems. The majority of systems we currently see in our daily practice cannot be characterized by a single NFR or technology profile. We call them *heterogeneous* solutions, because they consist of components that each have their own NFR profile. These differences between NFR profiles within a single system can be caused by differences in technology, complexity or any other factor that influences the productivity factor of a single component. Modern architectural styles like micro-services actively promote such diversity of technology in software systems in order to allow local optimization of loosely coupled components [3]. In our experience, heterogeneous solutions are becoming the norm, and we need estimating standards that facilitate this.

The key to analyzing the impact of NFRs on software systems estimation is the system's architecture. The architecture determines substantially whether a system is able to exhibit its desired (or required) quality attributes. Architectural patterns, tactics and styles are applied to enable specific sets of quality attributes [4]. Architectural decisions can add functionality to fulfill NFRs, and architectural decisions set the system's technology profile, which co-determines the productivity factor(s) [5]. In this time of heterogeneous software systems, there is no shortcut around the architecture to estimate the cost of fulfilling NFRs. It is time to make architecture a first class citizen in software estimating standards.

2 NON-FUNCTIONAL REQUIREMENTS IN THE COSMIC METHOD

In 2015 the Common Software Measurement International Consortium (COSMIC) published a document on how to consider non-functional and project requirements in software project performance measurement, benchmarking and estimating [2]. This document defines the various types of requirements and gives guidance on how these can be translated into cost:

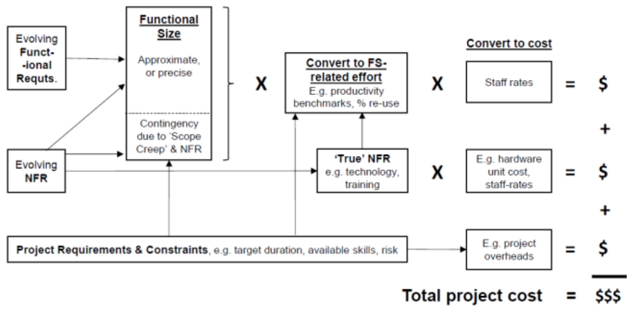
Evolving Funct- -ional Requts.
Functional Size
Approximate, or precise
Contingency due to 'Scope Creep' & NFR
Evolving NFR
X
Convert to FS-related effort
E.g. productivity benchmarks, % re-use
X
Convert to cost
Staff rates
= $
+
'True' NFR
e.g. technology, training
X
E.g. hardware unit cost, staff-rates
= $
+
Project Requirements & Constraints, e.g. target duration, available skills, risk
E.g. project overheads
= $
Total project cost = $$$

Figure 1: Transformation of requirements to cost in [2]

As can be seen from figure 1, the NFRs impact the cost estimate in three ways: by affecting (increasing) the functional size, by affecting the productivity factor for each functional component and by adding costs that are not related to software production, like hardware and training. All three of these impacts are determined by architecture:

- NFR-related increase of functional size is caused by architectural decisions to fulfill these NFRs by adding functionality. NFRs are fulfilled by applying architectural patterns, styles and tactics that introduce new functionality to the system. Examples are fulfilling a security NFR by adding authentication functionality, or improving response times by adding caching functionality. An extensive list of these tactics and how they are related to NFRs can be found in [4].
- NFR-related impact on productivity is caused by architectural decisions to use certain technologies (such as programming languages) or architectural styles that do not add functionality, but impact productivity, e.g. by enforcing a certain way of working. Examples are the choice to implement functionality in Python, or the decision to use only RESTful interfaces to improve modifiability.
- Other NFR-related costs are also determined largely by architectural decisions, such as a choice of vendor, the decision to use specialized hardware appliances (e.g. to increase performance), or selection of a deployment platform with its associated costs (e.g. to fulfill a scalability NFR).

In short, architecture is an implicit part of this mechanism to account for NFRs.

The calculation depicted in Figure 1 can be formulated as follows:

$$C = \sum_{x \in S} F_x \cdot P_x \cdot R_x + \sum_{y \in T} n_y \cdot r_y + O$$

where

C is the total project cost
S is the set of functional components in the project to deliver the Functional User Requirements
F_x is the functional size of functional component x
P_x is the productivity factor of functional component x
R_x is the (average) staff rate of functional component x
T is the set of deliverable elements added to the project to fulfill "true" NFRs
n_y is the size or count of deliverable element y
r_y is the unit cost per size element of type y
O is the project overhead

The project overhead in the COSMIC method is defined by the Project Requirements & Constraints (PRC). The PRC are defined as: "Requirements that define how a software system project should be managed and resourced or constraints that affect its performance" [2].

PRC can affect an estimate in various ways, e.g.:

- a time constraint may result in a de-scoping exercise or by adding more staff to work in parallel, at the expense of lower productivity.
- low staff experience in a new technology may increase effort.
- an uplift on staff rates, or an uplift to cover the cost of support activities like a project management office or database management, in which case it is a multiplying factor, not an addition.

From the COSMIC perspective there are not likely to be direct effects of architecture on PRC factors. Architecture factors affect the product, PRC factors affect the project.

3 NON-FUNCTIONAL REQUIREMENTS IN THE EPA METHOD

In 2012 three of the authors proposed a framework to estimate the implementation cost of packaged applications [6]. In 2016 the full version was published as a Nesma guideline on Estimating Packaged Applications (EPA) [7]. This framework defines the various types of cost drivers in the different life-cycle stages of the implementation of packaged software and gives guidance on how these can be translated into cost:

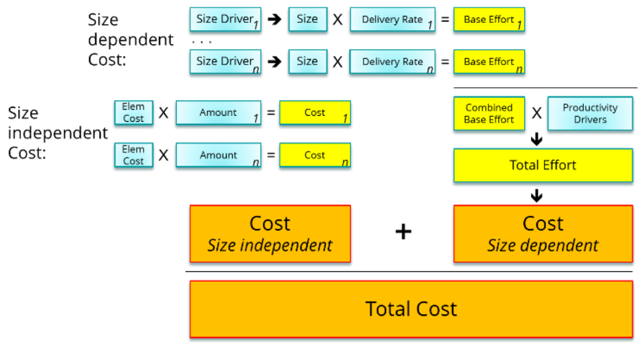

Figure 2: EPA Cost Estimating Model

As can be seen from figure 2, all three ways in which NFRs can impact the cost estimate as described in the previous section are present in a more or less similar way.
The way in which size is defined in the EPA framework means that size can be related to either functional or non-functional requirements. Examples of size drivers in the EPA framework include: workshops, key-users, data conversions and modules. Non-functional requirements also play a role in the productivity drivers that are defined in the model.
The calculation depicted in Figure 2 can be formulated as follows:

$$C = \left(\sum_{x \in Q} P_x\right) \cdot \sum_{y \in S} F_y \cdot D_y \cdot R_y + \sum_{z \in T} n_z \cdot r_z + O$$

where

C is the total project cost
Q is the set of productivity drivers of the project
P_x is the productivity driver of activity x
S is the set of functional components in the project to deliver the Functional User Requirements
F_y is the functional size of functional component y
D_y is the delivery rate of functional component y
R_y is the (average) staff rate of functional component y
T is the set of deliverable elements added to the project to fulfill "true" NFRs
n_z is the size or count of size-independent element z
r_z is the unit cost per element of type z
O is the size independent cost

4 NON-FUNCTIONAL REQUIREMENTS IN THE SNAP METHOD

In 2010 the International Function Point User Group (IFPUG) published the first version of the Software Non-functional Assessment Process (SNAP) on how to consider non-functional requirements in software estimating [6].

While function point analysis (FPA) measures the functional requirements by sizing the data flow through a software application, SNAP measures the non-functional requirements. The SNAP model consists of four categories and fourteen subcategories to measure the non-functional requirements. Each sub-category is sized, and the size of a requirement is the sum of the sizes of its sub-categories.
SNAP is complementary to the standard functional size approach and both measures have to be translated to effort and cost separately [9]:

$$C = F \cdot P_f \cdot R_f + N \cdot P_n \cdot R_n$$

where

C is the total project cost
F is the size of the functional requirements
P_f is the productivity factor for the functional reqs
R_f is the (average) staff rate for the functional reqs
N is the size of the non-functional requirements
P_n is the productivity factor for the non-functional requirements
R_n is the (average) staff rate for the non-functional requirements

As can be derived from the equation, this is a different approach to measuring the impact of NFRs than is described in the previous sections. Functional size is considered to be fixed and all NFRs are measured in a single size number with its own productivity factor. Project Requirements & Constraints are not measured by both FPA and SNAP [9].
A 2013 field test showed a weak correlation (R^2=.41) between effort and the functional and non-functional size. By excluding applications with extensive help functionality and recalibrating the subcategory Data Configuration the correlation was improved drastically (R^2=.89) [8]. This statistics-based improvement of the method had large repercussions on individual contracts that used the SNAP method as a contract base [9].
As mentioned in the introduction, the key issue with this formula is the fact that there is only one productivity factor to be applied to all functionality, and one to all non-functional requirements, whereas most modern solutions are heterogeneous and non-functional requirements are often satisfied by functionality.
Solutions nowadays consist of multiple types of components, which can differ substantially in terms of technology and NFRs, requiring us to apply different productivity factors. Keeping track of which effort is related to which type of requirement (functional or non-functional) when one type is evolving into the other as a project progresses is another challenge.

5 SOLUTION-BASED ESTIMATING

In 2014, two of the authors introduced the Solution Based Estimation approach to estimate the cost of delivering heterogeneous solutions [9]. Solution Based Estimation explicitly makes the architecture part of the estimating process by way of an architectural model called the Solution Breakdown Structure (SBS).

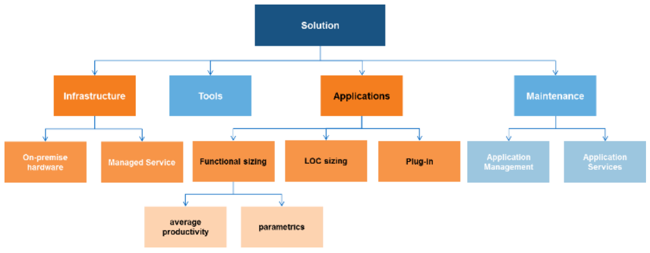

Figure 3: Solution Breakdown Structure (example)

The Solution Breakdown Structure is the representation of the solution that serves as a basis for cost estimation. It decomposes a heterogeneous solution into homogeneous deliverable elements, based on the architecture. These deliverable elements are the tangible result of applying the architectural patterns, styles and tactics applied to address NFRs; e.g. applying a hub-and-spoke architectural style results in a deliverable element to fulfill the 'hub' function, and applying a caching tactic to address a response time NFR results in a deliverable element to fulfill the 'cache' function. Thus, the impact of NFRs on the structure of the solution is made explicit. Cost calculation of each branch *b* in the SBS tree is done by recursively adding the cost of each sub-branch or leaf, and adding the integration overhead for that branch, or:

$$c(b) = \sum_{x \in b} c(x) + O_b$$

where

$c(b)$ is the cost of delivering branch b
$c(x)$ is the cost of delivering sub-branch or leaf x of b
O_b is the integration overhead to integrate b

In a good SBS the leaves in the SBS tree are homogeneous (this is an important condition of a good SBS), allowing the use of sizing techniques for estimating homogeneous elements, such as functional size for software elements, square footage and power for hardware hosting, number of FTEs for organizational entities, or NFRs like bandwidth, storage and computing capacity for infrastructure [10]:

$$c(l) = S_l \cdot R_l$$

Where S_l is some measure of the size of leaf l, and R_l is the unit cost.

In software systems, some of the leaves will be pieces of software, with functionality that is either an implementation of direct Functional Requirements, or the result of an architectural decision to address an NFR. When the leaf is a homogeneous piece of software, we can use functional size to estimate its cost, substituting S for $F \cdot P$:

$$c(l) = F_l \cdot P_l \cdot R_l$$

We can now see that the COSMIC formula from [2] is actually a special case of our Solution Based Estimating formula:

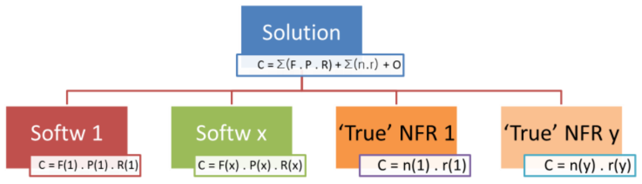

Figure 4: Special case of Solution Based Estimating

Figure 4 shows that the method proposed in [2] is equivalent to Solution Based Estimating for a software system where all software can be estimated as a number of homogeneous deliverable elements.
The same analysis of the EPA formula shows that this method is equivalent to Solution Based Estimating for a software system where the productivity driver is equally applicable to all sizeable elements.
The same analysis of the FPA+SNAP formula shows that this is equivalent to a unique case of the Solution Based Estimating formula, where the Solution Breakdown Structure consists of one homogeneous software element, one homogeneous NFR element and an overhead equal to zero.

6 DISCUSSION

The role of architecture in addressing NFRs has been firmly established for decades; for example, in the 90s Lawrence Chung's NFR Framework [11] provided a 'Goal-driven, process-oriented architectural design' method based on NFRs. More recently, Raymond Slot's PhD research into 49 software development projects found significant positive correlations between the application of architecture practices and the accuracy of project budget calculations. Slot specifically found that both the presence of an architect and the presence of a high-quality project architecture during the calculation of the technical price of solution are significantly correlated with a lower variance of the actual project budget [12]. Slot's finding is fully in line with our reasoning that budget estimates that take into account the architectural structure of a software system tend to be more accurate than those that don't.
Architectural decisions to address NFRs often represent choices between alternatives that each carry their own costs. These discrete choices cause discontinuities in the relationship between quantified NFR values and the cost of realizing them, leading to a very critical relationship between NFRs and costs that can only be understood by looking at the architecture [16].

7 CONCLUSION

Sophisticated architecture based software and its associated NFR in a system cannot be estimated using the same productivity factor for all architecture components. Therefore we need a more advanced model to account for the cost of NFRs. Solution-Based Estimating [9] is an approach that was proven in practice, which provides a candidate for such a more advanced model. For the software part of Architecture Driven Estimation, the COSMIC method offers a compatible approach. For heterogeneous solutions, Architecture Driven Estimation facilitates accounting for the impact of NFRs by explicitly including the architecture in the calculation.

There is no shortcut to estimating non-functional requirements by using homogeneous productivity factors. Approaches that bypass the heterogeneous nature of today's software systems will not lead to accurate cost predictions. Architecture Driven Estimation makes architecture a first class citizen, allowing more accurate cost predictions for modern software systems.

8 FUTURE WORK

The finding that only by taking the architecture into account we can make accurate cost predictions, means that there is still a lot of work to do.

First we have to reconsider homogeneous approaches that are widely used, like using one productivity factor for all types of software or for all types of non-functional requirements.

Then we need to standardize as much as possible the 'size' measures for NFRs and their associated productivity factors. Some exploratory work, like [2] and [7], has already been done, but these are only frameworks that need to become more concrete to be applicable by practitioners. This would be a logical extension of the much used publicly available ISBSG dataset [17].

We also need to establish how to calculate integration cost. This is still a Greenfield situation with respect to academic research.

REFERENCES

[1] C. Symons, "Accounting for Non-Functional Requirements in Productivity Measurement, Benchmarking & Estimating," in *UKSMA/COSMIC International Conference on Software Metrics & Estimating*, London, 2011.

[2] C. Symons, "Guideline on Non-Functional & Project Requirements," COSMIC, 2015.

[3] C. Richardson, "Pattern: Microservice Architecture," 2016. [Online]. Available: http://microservices.io/patterns/microservices.html. [Accessed 25 June 2017].

[4] L. Bass, P. Clements and R. Kazman, Software Architecture in Practice (3rd Edition), Addison-Wesley Professional, 2012.

[5] J. Tyree and A. Akerman, "Architecture decisions: Demystifying architecture.," *IEEE Software,* pp. 19-27, 22(2) 2005.

[6] F. Vogelezang, R. Nijland, E. van der Vliet, J. Hommes, H. Smit, K. van Straaten, D. Vandendaele and P. Bellen, "Estimating Packaged Software Implementations - The first part of a framework," in *International Workshop on Software Measurement*, Assisi, 2012.

[7] E. v. d. Vliet, F. Vogelezang and R. Nijland, "Estimating Packaged Software – a framework," in *NESMA*, 2016.

[8] IFPUG, "SNAP (Software Non-Functional Assessment Process) APM v2.2," IFPUG, 2014.

[9] T. Ben-Cnaan and C. Symons, "Accounting for Non-- Functional and Project Requirements in Software Project Performance Measurement, Benchmarking and Estimating: COSMIC and IFPUG Developments," in *International Workshop on Software Measurement*, Cracow, 2015.

[10] C. Tichenor, "A New Metric to Complement Function Points The Software Non-functional Assessment Process (SNAP)," *CrossTalk - The Journal of Defense Software Engineering, July/August 2013,*, no. July/August, pp. 21-26, 2013.

[11] M. Krzetowski, "Metoda SNAP jako próba przezwyciężenia problemów z wymiarowaniem wymagań niefunkcjonalnych," in *PSMO Conference on Project Estimation based on Functional Size*, Warsaw, 2017.

[12] E. Poort and E. van der Vliet, "Estimating the Cost of Heterogeneous Solutions," in *Softw. Measurement and Intl Conf. on Softw. Process and Product Measurement (IWSM-MENSURA), 2014 Joint Conf. of Intnl Workshop on*, Rotterdam, 2014.

[13] E. Poort and E. van der Vliet, "Architecting in a Solution Costing Context: Early Experiences with Solution-Based Estimating," in *Software Architecture (WICSA), 12th Working IEEE/IFIP Conference on*, 2015.

[14] L. Chung, B. A. Nixon, E. Yu and J. Mylopoulos, Non-Functional Requirements in Software Engineering, Springer, 1999.

[15] R. Slot, A method for valuing architecture-based business transformation and measuring the value of solutions architecture (PhD Thesis), Amsterdam: University of Amsterdam, 2010.

[16] B. Regnell, R. Berntsson Svensson and T. Olsson, "Supporting roadmapping of quality requirements," *IEEE Software*, vol. 25, pp. 42-47, 2008.

[17] ISBSG, "The International Software Benchmarking Standards Group," [Online]. Available: http://www.isbsg.org. [Accessed 1 July 2017].