Analysis of the Gap Between Initial Estimated Size and Final (True) Size of Implemented Software

Erdir Ungan[1], Sylvie Trudel[1], Alain Abran[2]

[1] Université du Québec à Montréal – UQAM (Montréal, Canada)
[2] École de Technologie Supérieure – ETS, University of Quebec (Montréal, Canada)
ungan.erdir@uqam.ca trudel.s@uqam.ca alain.abran@etsmtl.ca

Abstract. In this paper, we investigate the gap between the final true functional size of a piece of software at project closure and the functional size estimated much earlier in the development life cycle when the initial set of requirements is incomplete and often ambiguous. The different components and dimensions of this gap are identified and an approach is proposed to address them. The purpose of this approach is to improve early functional size estimation and, in turn, improve effort estimation. For the purposes of this paper, the ISO 19761 COSMIC standard on functional size measurement is taken as reference for discussion, while the majority of concepts presented are generic to most other similar ISO standards.

Keywords: Functional Size Measurement, COSMIC, size estimation, size accuracy, requirements quality, software estimation, ISO 19761

1 Introduction

Functional Size Measurement (FSM) methods are used for various purposes in software project management, including estimation, activity and resource planning, normalization of quality and productivity indicators, performance management, resource management, etc. FSM stands out among software size measurement methods as it is based on the requirements themselves, which as soon as they become available before any coding is done, can provide size information in the earlier stages of the software development life cycle (SDLC). This makes FSM a tool of choice for planning techniques that require an early view of the software to be developed.

The main input of any FSM method is the set of functional requirements for a piece of software. This is mandatory for an FSM to be recognized as an ISO standard [1]. However, very early in the software development lifecycle it is unrealistic to expect this set of requirements to describe the full scope of functionality of the software as a whole, including all the necessary functional details. There are many functional unknowns at that point in time. Therefore, in the presence of such unknowns it is expected that there can be a large variance, when using any FSM method, with the final size of the software delivered at the end of the project when all the unknowns have been addressed. In other words, there is a gap between the functional size one can estimate

from a set of incomplete and ambiguous functional requirements and the size measured from a very detailed set of functional requirements fully developed and implemented in a piece of software and made available to customers.

Boehm [2] suggests that variation of estimates will progressively decrease as the project progresses and additional more precise information becomes available. Fig. 1 shows a plot of the expected accuracy of software project estimates as a function of the software life cycle phase.

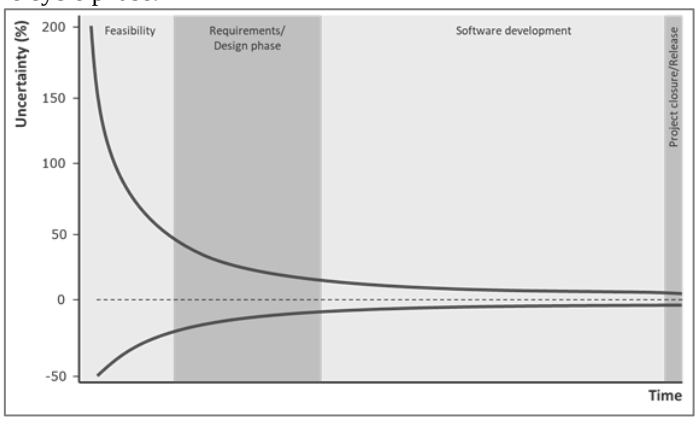

Fig. 1. Software estimation accuracy versus lifecycle phases [3]

Fig. 1 can be used to explain that:

- At time = project closure, everything is known about the software. When a measurer has access to all of the requirements information as implemented in the software code, he can then measure the size with high accuracy.
- At t = requirements, specifications are complete, all necessary levels of functional detail are fully documented. At this stage, all functionality has been verified and approved, measurement can be done accurately on the software to be developed. Will the size measured at this point in time of the lifecycle be the same as the size at t= closure?
  - Yes, provided that no changes will be made to the set of approved requirements.
  - No, if there is scope creep during the development and testing phases.
- At time = feasibility study, the information available on the requirements is typically high level and without much detail. From a functional perspective many unknowns remain. With imprecise and incomplete inputs, there cannot be accurate measurement. Rather, based on the incomplete information available the expected functional size at project closure can only be ‘estimated’ and, as with any type of estimate, this comes with an uncertainty range that will vary depending on the quality, completeness and stability of the set of requirements. [4]

Measurers and software engineers need a clear understanding of the level of information available at the time of measurement and how it impacts the accuracy of any measurement or estimation of size. In this paper, we identify some of the sources leading to the gaps between true size of a piece of software at project closure and the size estimated much early in the development lifecycle. This understanding will provide insight for designing and proposing techniques for estimating functional size early in the development lifecycle.

This paper focuses on the practice of estimating and approximating functional size throughout the SDLC. Any discussion of effort estimation and the relationship between functional size and effort estimation is out of scope here. Any sizing activity performed with a less than detailed set of requirements is either an approximation or estimation of functional size. These terms (e.g., measurement, approximation and estimation) are at times used interchangeably in the literature.

For the purpose of this paper, the following definitions are adopted:

- **Size Measurement:** Recognized functional size measurement methods [5-9] dictate that a piece of software can be measured only when its functional requirements are defined at a certain level of detail including all of the functionality to be implemented and delivered by the software. For example, COSMIC ISO 19761 requires that, for a proper measurement with all corresponding ISO rules, each requirement must be defined at the level of Functional Processes with enough information to identify all of the 'data movements' within them.
- **Size Approximation:** Approximating the functional size of a piece of software using the less than adequate information currently at hand about existing software fully developed and implemented. For instance, high level requirements and/or components are used as an input instead of the full set of detailed requirements implemented in the piece of software.
- **Size Estimation:** The practice of estimating the final functional size of a piece of software that is yet to be implemented based on the information at hand in the initial SDLC stages. When an FSM is used on either high level requirements and/or requirements of low quality this leads to quantitative results based on a set of requirements which does not capture the required level of detail in Data Movements. Neither does it capture the whole scope of functionality delivered, or expected to be delivered, by the finished software product, which then leads to a subset of the final size that will be misleading. Such quantitative results should be taken as estimates of the final size keeping in mind the related constraints and limitations.

To our knowledge, there is no research that has investigated directly this gap between early size estimation and final size measurement, nor any approach to address this issue. To approach this issue, it is first necessary to carry out analytical research to determine the nature and causes of the gaps between earlier sizing and final/delivered size of software, and to identify the sources of these gaps. Only then, can empirical research be carried out to observe the magnitude of these gaps and develop quantitative indicators and relationship models across indicators to bridge the gaps throughout the SDLC. The research reported in this paper is limited to the analytical portion of this research program.

This paper is structured as follows. Section 2 presents related work. Section 3 presents a discussion on the final and true functional size of a software at project closure. Section 4 presents a discussion on the known unknowns and unknown unknowns of the initial requirements at the project initiation phase. Section 5 presents a discussion on the sources of the gaps between the initial visible size in the SDLC and the final size at SDLC closure, including a proposed approach to address these issues. Section 6 presents a summary and suggestions for future work.

2 Related Work

Despite being available earlier than other sizing methods, a precise application of an FSM requires that the functional requirements of software be detailed and the architecture defined [10]. More often than not, this point in the SDLC is late for project estimation needs of the organization. Therefore, several size estimation techniques have been proposed in order to have a size estimate that can be used as an input for these early management activities.

For instance, the COSMIC 'Guideline for Early or Rapid COSMIC Functional Size Measurement by using approximation approaches' [3] proposes the following seven approximation techniques:

1. Average functional process
2. Fixed size classification
3. Equal size bands
4. Average use case
5. Functional size measurement patterns
6. Early and quick COSMIC sizing
7. Easy Function Points

It also mentions some emerging approaches based, for instance, on informal text, fuzzy logic, etc.

This COSMIC guideline document presents the pros and cons of each of these estimation-approximation techniques, their recommended area(s) of application and what is known of their validity (i.e. when available in the corresponding references). Most of these techniques are based on some quantitative analysis of data at project completion and the development of indicators at that point in time in the SDLC. However, they do not look backward into the SDLC to identify the information missing early in the SDLC and the corresponding sources. Most size estimation techniques usually take "known unknowns" into account. That is, when the requirements at hand are not detailed enough at the time of estimation they suggest steps to approximate for the missing size information. However, there is usually an "unknown unknown" component of the functional size to be estimated.

Many of these early size estimation and approximation techniques in the literature view their input, i.e. the requirements set, as a whole. They tend to suggest methods to "fix" the effect of any inadequately defined and/or missing functionality by adjusting the estimated sizes but do not discuss this issue in detail.

3 The Functional Size of a Software at Closure: True Size and Accuracy of Measurement

At closure, all software functions have been developed and their functional size can be measured with high accuracy by skilled measurers with expertise in the measurement rules of the FSM used for any specific measurement exercise.

But expertise in measurement is not sufficient for measurement accuracy, where measurement accuracy is defined in the International Vocabulary of Metrology as "closeness of agreement between a measured quantity value and a true quantity value of a measurand" [11]. The measurer must also have access to all the software functions being implemented. In practice, not all measurers have access to all such functions and the related information needed for a proper application of the measurement rules:

A. On one hand, software developers who have developed themselves all such functions have the information necessary for measurement. Since they implemented these functions they have firsthand knowledge of all the details necessary for measurement, whether there is documentation of such functions or not, and whether such documentation is complete or not they can rely on their own implementation of these functions within the code itself, at any point in time in the development and testing phases.

B. On the other hand, measurers who have not been involved in the development phases must rely on the documentation made available to them for measurement purposes. Therefore, these measurers are highly dependent on the completeness of such documentation, both in terms of the number of functions and the consistency of such documentation with respect to the functions actually implemented in code. Also, the documentation might not include all of the details of the functions implemented by the software developers. For instance, the written documents made available to measurers may:

- Include only the user-related functions and may not include other software functions developed to implement interfaces with other parts of the system architecture and derived from a number of system non-functional requirements allocated to software functions during late development phases.
- Not include last minute functional changes implemented during the testing stages.

With respect to A above, software developers who have measurement expertise and are measuring software they themselves developed have a much better context to determine a measured size that is close to the 'true size' of the software developed and implemented. Such measurers have, therefore, the best conditions to carry out 'accurate' measurements.

With respect to B above, both the low level of availability and completeness of the documentation leads measurers to measure only whatever number of functions and related functional details are 'visible' to them. At the time of measurement, this leaves a number of invisible functions and related functional details unknown to them and un-

measured. This of course leads to inaccurate functional measurement and the magnitude of such inaccuracy is typically unknown to the measurers.

Two types of corrective actions can be taken to address this type of unknown and related inaccuracy:

A. Software developers with measurement expertise and who have been involved in the development process could measure in parallel to establish the reference value closest to the 'true size'. With these reference results then:

i. The total size level can be used to determine the accuracy of the measurers of group B, and
ii. The detailed comparison of the size differences, and of their sources, can be used to establish both the sources of 'unknown' and their relative contributions to measurement errors.

Of course, such an approach would be expensive. On the one hand, it requires duplicate measurements and the availability of software developers who may have already been re-assigned to other projects with higher priority and deadlines to meet.

B. An alternative approach is to ask software developers to carry out an audit on the documents used by the measurers of type B to identify, based on their practical knowledge, both the functions and functional details missed in the functions already measured. This type of audit can then identify the 'unkowns' (e.g., the functions and functional details that have been 'invisible' to the measurers of type B). The next sub-step would then be to:

i. Carry out the exact measurement if the documentation becomes available and is of high quality and completeness, or
ii. Apply an 'approximation' technique on such unknowns (e.g., unknown at the moment of measurement).

Note that in this context where the software developers themselves have been involved in the measurement process at project closure, there should be no 'unknown unknowns'.

4 Requirements at the Initiation Phase: Initial Requirements, Known Unknowns and Unknown Unknowns

At the beginning of any software development initiative, be it a project, a sprint or a change request, the piece of software to be developed is described by an overview of a set of requirements typically defined at a high level of description without the details of each specific functionality. This high level, and incomplete, set of requirements will be referred to as **Initial Requirements**.

While the measurement rules of the various ISO standards for FSM can be applied to these initial requirements, it is unreasonable to expect the size obtained in these conditions to be similar to the true size of the software at project closure. In such a state of

incompleteness of the requirements (both in breadth and depth), there are too many '*known unknowns*' and '*unknown unknowns*'.

The terms 'known unknown' and 'unknown unknown' were originally used to categorize risks in risk management [12]. The same concepts can be applied to estimation:

- Known unknowns refer to functionality (size) that we know will be added over time (e.g., based on historical data, experience, etc.) but do not know the exact impact on size.
- Unknown unknowns typically refer to unforeseeable changes in the scope of functionality, usually stemming from stakeholders or factors outside the organization.

In these contexts, the measurers need to use size approximation techniques to offset some of these unknowns. Size measurement can be performed at any time in the SDLC and represents the requirements available at the time of measurement. We call this the **Visible Size** of that piece of software at that point in time. As a special case, visible size obtained using the Initial Requirements is called the **Initial Size.**

In summary, there is a gap between the set of Initial Requirements and the functionality delivered by the software at closure. Similarly, any **Visible Size** for a piece of software before its completion will be a subset of its **Final Size**.

5 Analysis of the Sources of the Gap Between the Initial Visible Size and the Final (True) Size

This section presents an analysis of the sources of the gap between the initial visible size of the functionality documented at the initiation of a project and the final and true size at project closure.

5.1 Sources of the size gaps

Several factors may lead the final functionality delivered by the software at the end of a project to be different from that defined at the time the initial requirements were created:

- There may be some functionality added to the software during implementation not documented in the set of initial software requirements.
- There may be new requirements added after the initial set of requirements was defined.

These factors will increase the amount of functionality in the software resulting in a gap between the initial and final size. In addition, there will be a gap due to requirements quality as mentioned in previous sections. When the documentation of any requirement is of lower quality, even if it is possible to name the functionality (i.e. a functional process), it will not be possible to identify all relevant Data Movements.

Therefore, its measurement at that point in time would not adequately correspond to

its true functional size. This will introduce another gap between the Visible Size at that point in time in the lifecycle and the True Final Size. Combining these factors, we can identify the components of the gap between the initial and final true size as:

- Lower lever functionality that is implemented as a part of the higher level requirements but not detailed in the initial requirements (**Hidden Functionality)**.
- Functionality that has been implemented but not fully documented in the initial set of requirements **(Undocumented Functionality)**.
- Functionality that has been added after the initial analysis as the project progresses **(Added Functionality)**.

All these causes need to be considered in order to understand the variations in size between the **Initial Size** at project initiation and the **Final Size**.

We can visualize the **Initial Size** in comparison to **Final Size** in Fig. 2 where the solid lined boxes represent available size components, and the dotted lined boxes represent the components of final size that are not (yet) available at the time of measurement.

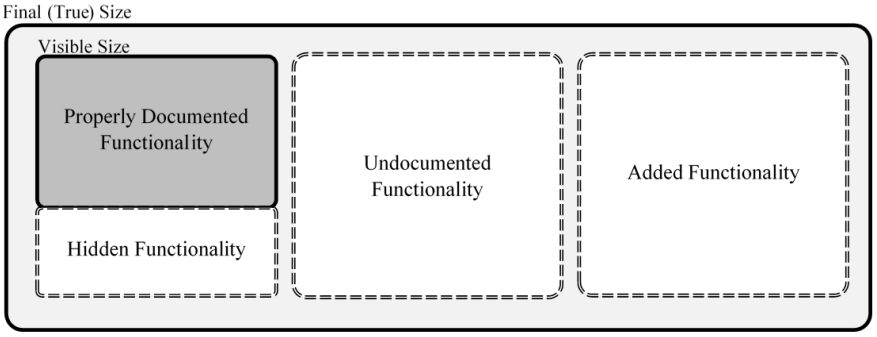

Fig. 2 Initial visible size vs. final true size

As the project progresses it is expected that:

- Initial requirements will be detailed progressively (added functional details to a functionality already identified).
- Some initially undocumented functionality will be approved and documented.
- Additional functionality will be captured as additional requirements.
- Some functionality will be removed and/or changed.

So, for any measurement performed at a further point in the SDLC, the visible size will approach the final and true size. If the measurement is repeated at a further point in the SDLC the difference between the visible size and final size can be visualized as in Fig. 3. Note that many missing functionalities are now documented and measurable. However, there will still be size components missing due to low requirements quality and some remaining undocumented functionality. The true size can be measured when

all functionality delivered by the software is documented and each requirement sufficiently detailed. The difference between the visible size further in the SDLC and final true size can be visualized in Fig. 3 where:

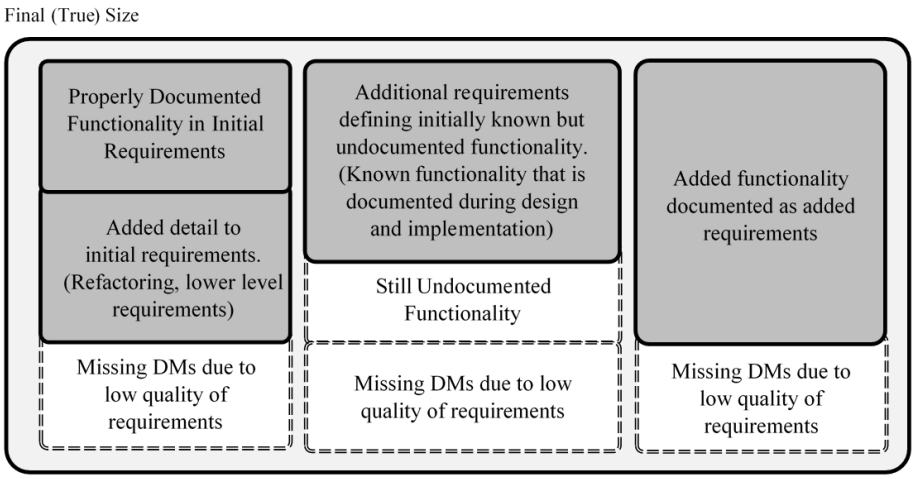

Fig. 3. Visible size at any time in SDLC vs. final true size

- **Properly Documented Functionality**

At least some of the initial requirements may be well defined and detailed at the level that FSM methods require. This will result in some properly measured functionality. Data movements are identified and recorded. Note that having a proper measurement also means the software architecture has been wholly defined.

- **Hidden Functionality**

Within the initial set of requirements most functionality will only be defined at a high level, not allowing the identification of all the specific Data Movements within those functions and that would go undetected until those functions are detailed enough. This can be due to vague and incomplete requirements as well as requirements that mostly define relationships between high level architecture components. Such requirements can seem to be complete and detailed enough in terms of functional steps, whereas the components involved are at a higher level than the actual components implemented in the software. Most of the time, system architecture dictates a portion of these hidden requirements where intermediary data movements between components can become completely known only after the architecture phase is complete. Some of the missing functionality in this category can also stem from the non-functional requirements (NFRs) within the initial requirements that are be converted into FURs to be allocated to software further in the SDLC (i.e. Quasi NFRs [13]).

Example: Initial requirements and preliminary architecture definitions for a software system describe a certain functional process “FP1”. When it is measured with COSMIC FSM, we identify a series of steps corresponding to Data Movements between components A, B, C and D as seen in **Error! Reference source not found.**. This leads to the identification of 7 data movements – e.g., 7 COSMIC Function Points = 7 CFP.

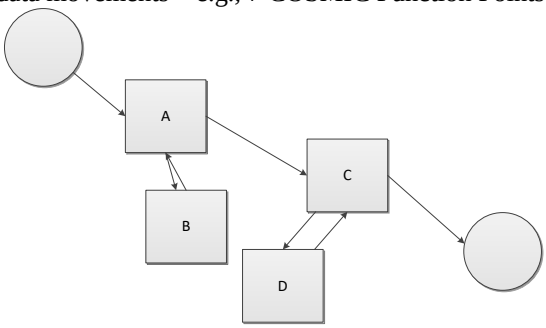

Fig. 4. A set of high level components and Data Movements among them in an FP

However, later during implementation, it can be observed that there are sub components to the components previously defined in the architecture and there exist intermediary date movements among them. See Fig. 5 where *An* represents a subcomponent of component A. When these intermediary data movements are incorporated into the COSMIC measurement, this will lead to a higher functional size of 11 Data Movements (CFPs). The data movements that are invisible in the first graph will contribute to the final size of the function while being invisible in the initial requirements. There can be cases in which some part of a FUR is detailed enough to identify Data Movements and some are left with less detail.

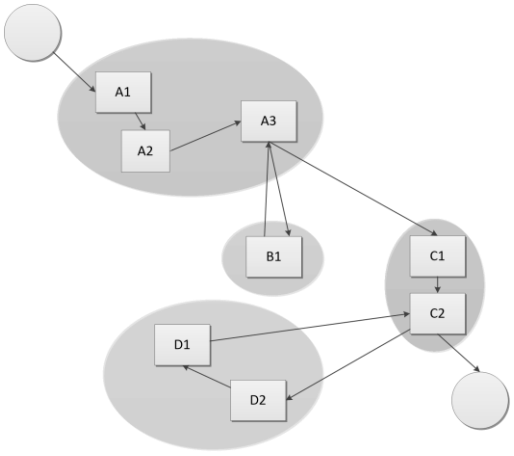

Fig. 5. Subcomponents of software components and additional intermediary Data Movements

In the case study in [14] the COSMIC measurements of the Rice Cooker case study [15] were compared with different levels of requirement details. There it was observed that there can be drastic differences between measurements performed for the same system using requirements of different levels of detail. In order to minimize this size gap, organizations can improve the quality of the requirements through implementing requirement standards, training analysts in requirements engineering best practices and verifying and validating the requirements.

Organizations may decide to have a high level architecture and corresponding high level requirements for management purposes. However, they need to be aware of the level of the requirements and have quantitative data on how much of the actual size is missing due to high level requirements.

- **Undocumented Functionality**

In many cases, even fairly early in the lifecycle, some implied functionality will be implemented in the software that is not mentioned in the requirement documents (e.g. known unknown). For example:

- Certain functions may be "usual" for an organization or application domain and be omitted in order that the requirement texts not be repetitive.
- Or there can be undocumented requirements that apply to every piece of software that is being developed under certain conditions, either as standard practice within a specific organization, or as standards across an industry or software domain.
- They can be some unknowns to the analysts but implemented by developers. Certain functionality may be deemed initially as "technical" for some stakeholders at the customer or management levels, and/or low level and left out of the scope of requirements documents. However, these functions will be implemented and constitute a part of the final size of the software.
- There exist many interfaces between systems/components undefined by the requirement documents for the subsystems at the project level.

For example, when the actual requirements define business functionality for a banking application, other functionality stemming from other quality concerns such as security, authentication, architecture or technology may not be mentioned in the requirements document. Similar to 'Hidden Functionality' defined above, some of the missing functionality in this category can also stem from the non-functional requirements (NFRs) within the initial requirements that are be converted into FURs further in the SDLC.

A previous study [16] illustrated how implied and/or hidden functionality can cause a big gap between the initial and final functional size of a piece of software. The set of requirements used as the input for the initial COSMIC measurement was one of the standard case studies published by COSMIC. Requirements were of high quality and did not include vague or incomplete requirements. For the selected set of requirements in the example, the final size turned out to be 236% greater than the size as measured in the initiation phase. This size difference was due to security functionality that was documented as high-level system non-functional requirements at the time of the initial

measurement and that later was allocated to the software and implemented as additional functionality.

In similar cases, apart from being documented as NFRs, such additional functionality may stem from standard practices of an organization, business domain or required security standards. In order to minimize this size gap, organizations need to be aware that not all of the functionality delivered is captured in their requirement documents. They should know the typical additional functionality added to software in implementation and use relevant approximation methods to incorporate this size increase in their early size measurements. Interface definitions/requirements should be documented.

- **Added/Modified Functionality**

At any time in the SDLC, the functionality scope of a piece of software may be altered. Some functions might be added while some might be removed or modified. Such changes typically occur due to customer change requests and/or scope changes. These additional and/or modified functionalities cannot be measured at the initial stages of a project, or at any stage until the changes actually happen. However, they will be incorporated in the final size of the software developed.

Increase in size due to such changes in functionality is usually referred to as **scope creep**. Frequency and size of requirement changes are dependent on the development processes, customers and development environment. At the time of measurement, it may not be possible to foresee and prevent size change stemming from scope creep. In some cases, this gap can be estimated using statistical methods on historical data.

In order to minimize this gap, organizations need to keep statistical data regarding customers, the SDLC and processes used, domain etc. in order to be able to make adjustments to their early size estimates to reflect the potential impact of changes. In addition, organizations should improve requirement elicitation processes through activities such as early prototyping, incremental development etc. to minimize the impact of requirement changes on size approximation.

5.2 A proposed Approach for Identifying the Sources of Invisible Functions and for Approximating the Expected Final Size

The proposed approach is based on the development of an organization repository to record size data information related to the initial estimated size (e.g. t = project initiation phase) and size at the time of delivery (e.g. t = project closure). Such an organization repository will allow:

- For each piece of software, to compare the initial requirements and the functionality at the time of delivery.
- Classify additional size (Hidden Functionality, Undocumented Functionality, Additional/Modified Functionality).
- Identify the reasons for the size change in each dimension.
- Use customized techniques for different dimensions of size change to adjust initial measurement results and better estimate the final size of software.

6 Summary and Future Work

In this paper, some of the factors that cause the gap between any early measurement and the final size of a piece of software have been identified and discussed to make it possible to fine tune and customize size estimation techniques for organizations and ultimately more useful size estimation techniques.

We have defined different components and dimensions of the gap between the initial visible size and final true size of a piece of software. That is, the missing components of the size obtained by measuring a piece of software before it is completed and thoroughly documented.

We also pointed out the importance of identifying the knowns, known unknowns and unknown unknowns for estimation or approximation techniques at any given time in the SDLC. This would allow approximation techniques to identify which dimension(s) of the size gap they are using as input and which dimension(s) they are estimating.

Instead of following a 'one size fits all' approach for estimation techniques, future work with statistical early estimation techniques may be updated to incorporate different dimensions of the size gap and combine different approaches to calculate these dimensions separately.

As a number of factors in the software development context affect the composition of the size gap, related to the characteristics of a development environment, certain dimensions may turn out to be more significant than others. Having estimation techniques incorporating different dimensions of size gap will help organizations select, customize and utilize the estimation techniques that are best suited to the characteristics of their development context including business domain, standard processes, teams and even customers.

With different dimensions of the gap between initial size estimation and final size measurement identified, empirical research will be conducted on real life cases. Possible quantitative indicators for different dimensions and relationship models for those indicators will be investigated.

Empirical research will be conducted to compare the estimation accuracies of various estimation methods. Estimation methods will be applied to whole sets of initial requirements and their size accuracies compared. Moreover, composite approaches including applying combinations of methods to different dimensions of the gap between initial size and final size will be demonstrated using real life cases and their accuracies compared.

References

1. ISO/IEC 14143-1:2007 Information technology -- Software measurement -- Functional size measurement -- Part 1: Definition of concepts
2. Boehm, Barry W., Software Engineering Economics, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981

3. COSMIC Group, Guideline for Early or Rapid COSMIC Functional Size Measurement by using approximation approaches. https://cosmic-sizing.org/publications/guideline-for-early-or-rapid-cosmic-fsm/ . 2015.
4. The Effect of the Quality of Software Requirements Document on the Functional Size Measurement. G. Yılmaz, E. Ungan, O. Demirörs. United Kingdom Software Metrics Association International Conference on Software Metrics and Estimating. London, UK. 2011.
5. ISO/IEC 19761 Software Engineering - COSMIC - A Functional Size Measurement Method, 2011
6. ISO/IEC ISO/IEC 20926:2009. Software and systems engineering - Software measurement - IFPUG functional size measurement method 2009.
7. ISO/IEC ISO/IEC 20968:2002, Software engineering - Mk II Function Point Analysis — Counting Practices Manual, 2002.
8. ISO/IEC ISO/IEC 24570:2005 Software engineering - NESMA functional size measurement method version 2.1 - Definitions and counting guidelines for the application of Function Point Analysis, 2005.
9. ISO/IEC ISO/IEC 29881:2010, Information technology - Systems and software engineering -FiSMA 1.1 functional size measurement method, 2010.
10. Santillo, L. (2011), "Early and Quick COSMIC FFP Overview," pp. 176-191, Abran-Dumke (Eds.), Publisher: COSMIC Function Points Theory and Advanced Practices, CRC Press. ISBN 978-1-4398-4486-1, 2011.
11. ISO: International vocabulary of metrology — Basic and general concepts and associated terms (VIM), 2010
12. Project Management Institute, A Guide to the Project Management Body of Knowledge (PMBOK® Guide), Newtown Square, PA, Project Management Institute, 7th ed. 2017
13. COSMIC Group, 2015. The COSMIC Functional Size Measurement Method Version 4.0.1 Guideline on Non Functional & Project Requirements. https://cosmic-sizing.org/publications/measurement-manual-v4-0-2/.
14. Ungan, Erdir & Onur Demirörs. 2015. A Functional Software Measurement Approach to Bridge the Gap Between Problem and Solution Domains. Selected Papers, 25th International Workshop on Software Measurement and 10th International Conference on Software Process and Product Measurement, (IWSM-Mensura), Kraków, Poland, October 5-7, 2015.
15. COSMIC Group, Rice Cooker – Cosmic Group Case Study. École de technologie supérieure, Université du Québec à Montréal - UQAM, Montréal (2003)
16. Erdir Ungan, Sylvie Trudel, and Luc Poulin. 2017. Using FSM patterns to size security non-functional requirements with COSMIC. 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement (IWSM Mensura '17). ACM, New York, NY, USA, 64-76.
17. Abran, A. (May 2010), Software Metrics and Software Metrology, John Wiley & Sons Intercedence and IEEE-CS Press, New Jersey, p. 328, ISBN:978-0-470-59720-0
18. Abran, A. et al. 2015. The COSMIC Functional Size Measurement Method – Measurement Manual, version 4.0.1, The COSMIC group, April 2015, available from: http://www.cosmic-sizing.org
19. COSMIC Group, Guideline on the Accuracy of COSMIC Function Points. https://cosmic-sizing.org/publications/guideline-on-the-accuracy-of-cosmic-function-points/ . 2018.
20. Gray, A. and S. MacDonell (1997), "A Comparison of Techniques for Developing Predictive Models of Software Metrics," Information and Software Technology, vol. 39, pp. 425-437.
21. J.-M. Desharnais and A. Abran (2003), "Approximation techniques for measuring Function Points," 13th International Workshop on Software Measurement – IWSM 2003, Montréal (Canada), Springer-Verlag, September 23-25, 2003, pp. 270-286.

22. M. Jørgensen and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," IEEE Transactions on Software Engineering, vol. 33, no. 1, January 2007, pp. 33-53.
23. Morgenshtern, O., T. Raz, and D Dovir (2007), "Factors affecting duration and effort estimation errors in software development projects," Information and Software Technology, vol. 49, no. 8, August 2007, pp. 827-837
24. Santillo, L., "Early FP Estimation and the Analytic Hierarchy Process," Proceedings of the ESCOM-SCOPE 2000 Conference., April 18-20, 2000, Munich, Germany.
25. Ungan, E. A Functional Software Measurement Approach to Bridge the Gap Between Problem and Solution Domains, PhD Thesis, Graduate School of Informatics - Middle East Technical University. Ankara, Turkey, 2013.
26. Valdés, F. and A. Abran (2007), "Industry Case Studies of Estimation Models based on Fuzzy Sets," in: IWSM-Mensura 2007, (UIB Universitat de les Illes Baleares, Palma de Mallorca, Spain), Editors: Abran-Dumke-Màs, Publisher: IWSM-Mensura 2007. ISBN 978-84-8384-020-7, November 5-9, pp. 87-101, 2007
27. Vogelezang F. W. and Prins, T. G., "Approximate size measurement with the COSMIC method: Factors of influence," SMEF 2007 Conference, Rome, Italy.