Establishing a Requirements Baseline by Functional Size Measurement Patterns

Ina Wentzlaff

paluno - The Ruhr Institute for Software Technology
University of Duisburg-Essen, Germany
ina.wentzlaff@uni-due.de

Abstract. *[Context]* The requirements baseline is an agreed-upon set of desired product features committed to a specific release. It determines intra- and interproject planning and involved performance assessment. *[Problem]* To establish a credible baseline, one which fits into the limits e.g. time box of the project and which satisfies user expectations, a reasonable choice of requirements must be made. This demands commensurable units of measure for requirements that are meaningful to most decision makers in a project team. *[Principle Idea]* This work proposes the use of functional size measurement patterns as a reusable point of reference to classify requirements consistently and to estimate their functional size in a reproducible way. It develops measurable requirements patterns out of problem frames and a counting procedure, that facilitates determining function points for requirements in accordance with ISO/IEC 20926:2009. *[Contribution]* These patterns provide the project team with a bias-free anchor regarding the functional size of requirements with which they can correlate their requirements estimates and anticipated work plan. Establishing the requirements baseline with respect to these patterns eases the comparability of project outcomes and thus contributes to the predictability of project planning.

Keywords: Integrated Requirements Engineering · Agile Project Practice · Planning Poker · Problem Frames · Functional Size Measurement

1 Introduction

Implementing software projects the agile way is all the rage. It is deemed to be the fast track to deliver software features at low cost due to self-organizing project controls. An agile development iteration follows a Plan-Do-Check-Adjust (PDCA) cycle [10, page 88]. Each completed cycle brings an improvement of the software increment i.e. of the product towards desired functionality. Sprint events of the Scrum project process framework [23] can be categorized accordingly, which is illustrated in Figure 1. In case multiple iterations are planned to cumulate in a product, it is generally referred to as release planning [27, page

216]. To this end, 'Requirements First' is still the name of the game. Before any development starts, the team needs to reach consensus on the work plan to be done during the upcoming iteration, which depends on the agreed-upon requirements baseline. In order to decide on it, a gamified decision-making process known as Planning Poker [5, page 56] is played by the team to estimate requirements. It is a modern adaption of the Wideband-Delphi estimation technique [4, Page 335ff] originated in the 1980s.

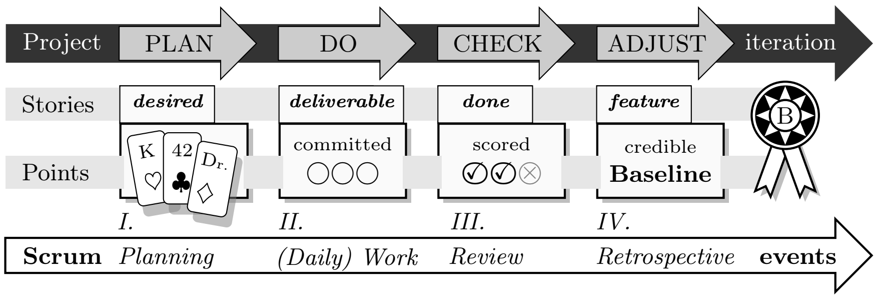


Fig. 1. Illustrating an agile project iteration by the four steps of a PDCA cycle

Estimating by Planning Poker starts with User Stories [5], which are brief requirement statements. In the following, each is to be assigned with a point value by the members of the project team. These Story Points serve to express the size of a user story [5, page 36]. They justify the amount of functional scope to be delivered for a story. The number of points is taken to organize the team's workload for an iteration. As the outcome of Planning Poker, the team commits to a work plan i.e. the requirements baseline by the agreed-upon number of points. Its commensurability manifests after the iteration is completed. Done functionality that delivers desired product features and thus satisfies the work plan scores the respective points for the baseline.

The key benefits of requirements estimating in points are (i.) that they make "the unit of estimation abstract, which makes it easier to commit to, and easier to adjust your commitments to" [27, page 161], and (ii.) their self-correcting nature [27, page 160]. "Even if a team is bad at estimating, as long as they're consistently bad, this makes a team's commitments self-correcting" [27, page 159], which helps "to meet expectations more consistently" [27, page 161].

A common criticism is that this consistency of estimates depends on the team, since (the meaning of) points assigned to requirements or a user story relates to an estimator's "experience and good feel rather than on formal criteria" [9, page 1343]. Team virtualization, membership turnover, silo mentality, etc. impacts the team's capability to establish a common respectively shared understanding on their requirements estimates. In each iteration they suffer from a bootstrapping problem [6]: what is the baseline they will compare to, which accounts for keeping their estimates consistent.

Establishing consistency of estimates from one iteration to another is obtainable by finding comparable requirements that require “a similar amount of work” [7]. These ’representative requirements’ serve the team as a point of reference to which they can correlate their project planning activities. In order to be applicable as a ’reference baseline’, they need to comply with mutually accepted best practices or standards.

With the bootstrapping problem in mind, it cannot be assumed that historical data about requirements estimates exist nor that these compare to the team estimates i.e. that the same principles and methods have been followed to establish them. Commensurable units of measure for requirements reusable in any project are needed to keep the baseline consistent. These units must be meaningful to most decision makers in a project team to “bridge any gaps in understanding [. . .] between clients and developers” [9, page 1343].

This work proposes to unify consideration of requirements and their respective amount of work expressed as a point value by means of patterns. Therefore, it joins the use of problem frames as best practices to classify requirements consistently, which are introduced in Section 1.1, and ISO/IEC 20926 as a standard for functional size measurement to determine the requirements’ size in a reproducible way, which is introduced in Section 1.2.

1.1 Requirements Classification by Problem Frames

Problem Frames are patterns, which characterize classes of recurring problem situations [19]. They are used to classify a set of requirements into simple i.e. self-contained subproblems, which allow to derive specifications in a reproducible way. This approach has been developed by Michael A. Jackson since 1995 [18] to deal with problem complexity in Requirements Engineering.

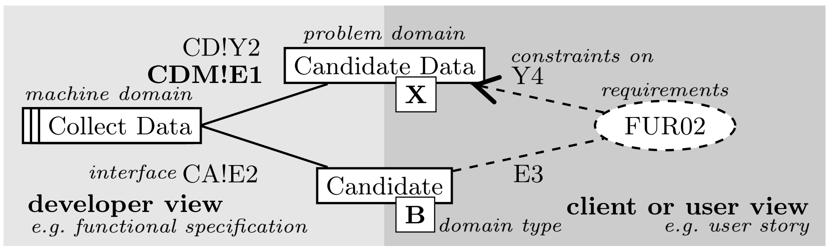
Using Problem Frames to classify requirements helps to identify the kind of functionality that is of relevance to the problem. They allow to find a proper machine behavior, i.e. specification, which can be created by the developer and that will make what is required by the user [19, Page 106]. Therefore, the machine to be built i.e. the software application has to control a specific part of the problem in a way as the requirement demands [19, Page 107]. A problem frame indicates this part by a requirement constraint on the respective problem domain.

In general, each problem frame is a unique combination of problem domains and their respective shared phenomena, which differ in their type, quantity, and correlation. The functionality of a software as describable by problem frames establishes one of the following three kinds of machine control: a constrained

$\boxed{\text{X}}\dashleftarrow$ Le(X)ical domain “is a physical representation of data” [19]. The machine controls reads or writes of these data.

$\boxed{\text{D}}\dashleftarrow$ (D)isplay domain is “an output device for the machine” [8]. On behalf of the machine, it provides “information to other problem domains” [8].

$\boxed{\text{C}}\dashleftarrow$ (C)ausal domain is provided with information controlled by the machine to invoke specific behavior of this problem domain.



Legend for interfaces and their {shared phenomena}:
E1{store40FormData}, E2{fillIn40FormData,FormData1..40}, E3{fillInCandidateData}, Y2{FormData1..40}, Y4{collectCandidateData}

Fig. 2. Subproblem for the requirements set FUR02 of a student recruitment web portal, which is based on the "simple workpieces" problem frame

For example, in Figure 2 a set of requirements named FUR02 builds a subproblem for a Student Recruitment Web Portal [15], that fits the "simple workpieces" [19, page 96] problem frame. These requirements are concerned with collecting a candidate's personal data necessary to prepare the candidate's application to a study program. They belong to a problem class which accounts for functionality that enables the processing i.e. storage of personal data.

1.2 Requirements Size Measurement by ISO/IEC 20926

Function Point Analysis [11] according to the International Function Point Users Group [21] is a functional size measurement (FSM) method, which is standardized in ISO/IEC 20926 [17]. It has become of vital importance in the early phases of software projects, since it is of use as input to effort estimation in project planning. Function Point Analysis was originally created by Allan Albrecht in 1979, who suggested a measure for developers productivity [1].

This technology-agnostic approach makes use of logical so-called base functional components to classify functional, user-recognizable requirements (FUR) into an "elementary unit of FUR defined by and used by an FSM Method for measurement purposes" [17, page 2, section 3.8]. Their size is determined within the counting process and expressed by a number in function points.

Functional size measurement introduces a means for quantifying requirements specifications. However, it is no Requirements Engineering method, since its prime purpose is not to result in a comprehensive requirements specification. Actually, its quality depends on a good requirements specification [22].

In this connection, identification of the application boundary, which is a "conceptual interface between the software under study and its users" [17, page 3, section 3.9] is crucial. The base functional components are counted with respect to the application boundary as illustrated in Figure 3. In order to determine the functional size of some requirements consistently, they need to be decomposed to fit the base functional components in a reproducible way.

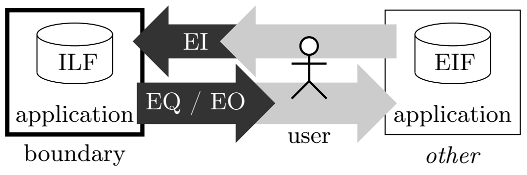


Fig. 3. Overview of base functional components (ILF, EIF, EI, EQ, EO) and their relation to the application boundary as considered by ISO/IEC 20926:2009 [17]

The measurement process according to ISO/IEC 20926 [17, page 8] distinguishes data functions and transactional functions. Data functions such as Internal Logical File (ILF), and External Interface File (EIF) relate to requirements that are concerned with data storage needs. An ILF is some ”information maintained within the boundary of the application being measured” [17, page 6, section 3.39], in contrast to an EIF, which is some ”information, which is referenced by the application being measured, but which is maintained within the boundary of another application” [17, page 5, section 3.29]. Requirements that are concerned with different kinds of data processing are considered by transactional functions. These are synonyms for the following elementary processes:

- $\boxed{\text{EI}}\leftarrow$ External Input (EI) is an “elementary process that processes [. . .] information sent from outside the boundary” [17, page 4, section 3.27], its primary intent is to “maintain an ILF [. . .]” [21].
- $\boxed{\text{EQ}}\leftrightsquigarrow$ External Inquiry (EQ) is an “elementary process that sends [. . .] information outside the boundary” [17, page 5, section 3.28], its primary intent is to “present information to a user. It presents only data that is retrieved [. . .]” [21].
- $\boxed{\text{EO}}\rightarrow$ External Output (EO) is an “elementary process that sends [. . .] information outside the boundary and includes additional processing logic beyond that of an external inquiry” [17, page 5, section 3.30], its primary intent is to “present information to a user. It presents data that is calculated or derived [. . .]” [21].

Given that an elementary process is the “smallest unit of activity that is meaningful to the user” [17, page 4, section 3.21], it specifies what the software shall do in terms of different kinds of processing and functionality, respectively. This drives decomposition of FUR comparable to problem frames.

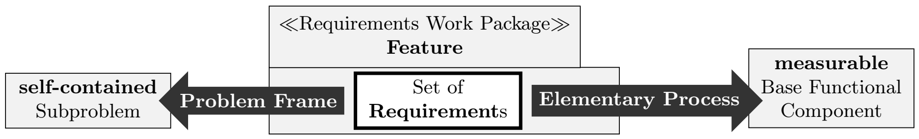
2 Requirements Work Packages

“At the start of each development iteration, [. . .] Groups of user stories are associated with a generic ’feature’. The complexity of each feature is then estimated” [9, page 1343]. Thus, the challenge is to set up commensurable units of requirements, i.e. requirements work packages [2] in the following, that maintain a consistent level of detail and address a comparable kind of functionality. Problem Frames tackle this challenge.

On the one hand, they allow for classifying requirements into self-contained subproblems, which maintain requirements description at a consistent level of detail, as long as their underlying frames belong to the same hierarchy of patterns. Each Problem Frame in Table 1 belongs to the set of Basic FSM Patterns "that apply to a complete yet single functional process" as defined in [26].

On the other hand, each set of requirements grouped by a problem frame is concerned with a particular kind of functionality, which is indicated by the constrained problem domain as discussed in Section 1.1. It can be compared and mapped to the primary intent of an elementary process as discussed in Section 1.2, which indicates similarly the kind of functionality or respective processing to be sized in a function point count. Table 1 provides for this relation by a bullet point.

Table 1. List of 17 problem frames applicable as basic functional size measurement patterns to set up self-contained and measurable requirements work packages



#	Problem Frame No. in [8]	Refer. Domain I	Refer. Domain II	Constr. Domain ←--	Problem Frame Name	Elementary Process EI ←	EQ ⇝	EO →
01	**PF 2.4**		C	X	model building	•		
02	**PF 2.7**		B	X	simple workpieces	•		
03	**PF 3.2**	C	X	X	triggered transformation	•		
04	**PF 3.3**	B	X	X	commanded transformation	•		
05	**PF 3.11**	B	C	X	commanded model building	•		
06	**PF 2.3**		X	D	model display		•	
07	**PF 2.6**		C	D	information display		•	
08	**PF 2.9**		B	D	commanded display		•	
09	**PF 3.8**	B	X	D	query		•	
10	**PF 3.15**	B	C	D	commanded information		•	
11	**PF 3.24**	C	X	D	triggered information		•	
12	**PF 2.2**		X	C	data-based control			•
13	**PF 2.5**		C	C	required behavior (var.)			•
14	**PF 2.8**		B	C	commanded behavior			•
15	**PF 3.7**	B	X	C	commanded data-based control			•
16	**PF 3.12**	X	C	C	triggered data-based control			•
17	**PF 3.14**	B	C	C	commanded behavior (variant)			•

Legend:
Problem domain type: (B)iddable, (C)ausal, (D)isplay, Le(X)ical
Elementary process: External Input (EI), External Inquiry (EQ), External Output (EO)
Problem Frame relates to Elementary Process: • both address a comparable kind of functionality

As a result, Table 1 lists 17 Basic FSM patterns, which allow to set up requirements work packages. These patterns provide for a grouping of requirements, which relate to measurable base functional components. They can be sized according to the rules of functional size measurement. A respective requirements sizing method is presented in Section 3.

Table 1 makes use of a short cut representation for problem frames introduced in prior work [8]. In each row, it simply enumerates relevant elements of a frame without changing its semantics. For instance, the "simple workpieces" problem frame as applied in Figure 2 is given by row #02 in this short cut, tabular form.

3 Requirements Sizing Method

This work proposes to use a customized requirements sizing method to provide for consistent and reproducible requirements estimates. It is applicable within planning of a project iteration as supplementary method to Planning Poker and given in detail in the Frame Counting Agenda [28]. This counting procedure adopts the steps of the ISO/IEC 20926 [17, section 5, pages 8–19, and 21] functional size measurement process to become effective for determining function points based on requirements work packages, such as developed in Section 2, and by means of the complexity and size metrics for data and transactional functions as defined by Tables A.1–A.5 in ISO/IEC 20926:2009 [17, page 23]. With regard to its process description, it follows the agenda concept [14].

Figure 4 depicts the requirements work package "Collect Candidate Data" in a stereotype notation as applied in the UML4PF [13] eclipse plugin for modeling problem frames. Domains that take the role of base functional components e.g. Candidate Data as ILF, Candidate as EIF, and Collect Data as an EI, can be easily identified and jointly considered in the counting process due to this uniform problem representation. That way, the size of a requirements work package is determined with special attention to the steps *Determine Data Functions* and *Determine Transactional Functions* of the counting procedure, which are the most critical and error-prone steps at once [12, page 205] as has been observed in industrial practice [16].

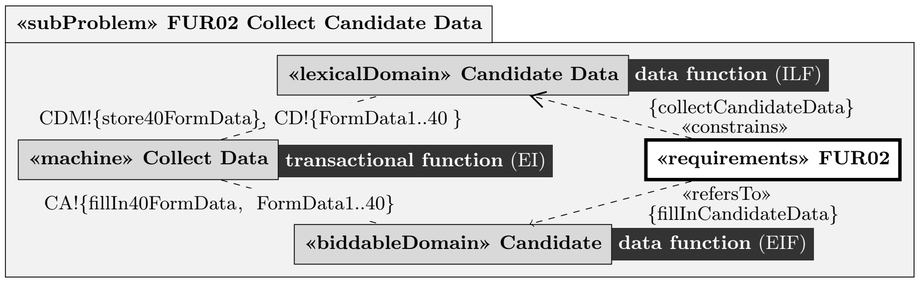

«subProblem» FUR02 Collect Candidate Data
«lexicalDomain» Candidate Data
data function (ILF)
CDM!{store40FormData}, CD!{FormData1..40 }
{collectCandidateData}
«constrains»
«machine» Collect Data
transactional function (EI)
«requirements» FUR02
CA!{fillIn40FormData, FormData1..40}
«refersTo»
{fillInCandidateData}
«biddableDomain» Candidate
data function (EIF)

Fig. 4. Requirements work package for FUR02 fits the "simple workpieces" problem frame and is measured according to Table 1 by the rules of an external input (EI)

Establishing requirements work packages based on functional size measurement patterns also prevents a malpositioned application boundary [25], which is a major source of difficulties in identifying base functional components and is therefore a root cause of wrong counts. In order to limit this risk of wrong counts, the proposed requirements sizing method provides validation conditions, which safeguard the application boundary and its involved base functional components. These validation conditions care for the associated counting rules in compliance with ISO/IEC 20926.

4 Discussion

To give an example of the outcome produced by the requirements sizing method as introduced in Section 3, it is applied to a Student Recruitment Web Portal [15].

Table 2. Overview of requirements work packages for a student recruitment web portal ranked by their functional size given in function points (FP)

Rank	**Requirements Work Package**	Functional Size Measurement Pattern according to Table 1		FP
1	**FUR04: Download Candidate Data**	#15 Commanded Data-Based Control	$\boxed{\text{EO}}\rightarrow$	17
2	**FUR06: Compile Candidate Résumé**	#09 Query	$\boxed{\text{EQ}}\leftrightsquigarrow$	16
3	**FUR01: Grant Access Authorization**	#14 Commanded Behaviour	$\boxed{\text{EO}}\rightarrow$	16
4	**FUR03: Review Candidate Data**	#09 Query	$\boxed{\text{EQ}}\leftrightsquigarrow$	13
5	**FUR02: Collect Candidate Data**	#02 Simple Workpieces	$\boxed{\text{EI}}\leftarrow$	13
6	**FUR05: Upload Candidate Files**	#05 Commanded Model Building	$\boxed{\text{EI}}\leftarrow$	11

Table 2 lists several 'features' of this web portal together with their respective point values, which have been established by use of the patterns and method proposed in this work. Details on how to obtain function points for a feature are given in the Frame Counting Agenda [28].

The list in Table 2 can be used by the project team to set up the requirements baseline for a project iteration. These requirement estimates allow to organize the Sprint Backlog, i.e. to commit to a set of requirements work packages, which are then to be done in the time box of a project iteration. A requirement work package, for which the team successfully delivers desired product features after completing the iteration, can be scored for the requirements baseline respectively.

Statistics on these baseline scores can be used to adjust iteration planning [27, page 170]. A credible baseline is established depending on measurable project outcome (scored points), which demonstrably fits into the time box of a project iteration, e.g. $credibility_{baseline} = \frac{project\ outcome\ (points\ scored)}{requirements\ baseline\ (points\ planned)}$. Thus, a requirements baseline, which is established by means of functional size measurement patterns, relates to fulfilled requirements, i.e. satisfied user expectations, rather than to individual team estimates.

5 Related Work

In [20] an “initial investigation” on the general applicability of problem frames to functional size measurement is outlined to be promising. That work is resumed in [3], where problem frames are used to set up UML sequence diagrams for user requirements as a means that is “fairly easy to measure”. In contrast to the aforementioned, this work proposes to use problem frames i.e. functional size measurement patterns as the first class means for requirements priorization and enactment by a project team.

6 Conclusion

The contribution of this work is twofold: As discussed in Section 2, it takes the problem frames approach to obtain recognizable classes of requirements, which are capable of relating user requirements with more measurable, technical specifications. They allow to transfer requirements into self-contained subproblems i.e. work packages, that are meaningful as units of work that the team can commit to and score for the baseline of a project iteration, and as units of measure for requirement’s functional size. As discussed in Section 3, this work proposes to use a counting procedure for the functional size measurement patterns in Table 1, that is based on a match of the problem frame concept with that of base functional components. These contributions serve the estimator to obtain reproducible results, i.e. function point estimates for requirements, that comply with a standard for functional size measurement, i.e. ISO/IEC 20926 [17]. This provides for estimates consistency, which is fundamental to establish a comparative requirements and involved performance baseline [2].

7 Future Work

This work implements an Integrated Requirements Engineering approach [24]. It establishes uniform requirement components by means of patterns, which are meaningful to clients and developers as the project’s collaborators. To investigate the use of this means to establish a commensurable point of reference for requirements and their involved performance baseline, the proposed requirements patterns and sizing method needs to be evaluated on the basis of more sample applications than the Student Recruitment Web Portal.

With regard to the client, it is intended to investigate how tender preparation and mockup creation relate to requirements work packages. With regard to the developer, it is planed to investigate how an appropriate design or reusable software components can be chosen with respect to a requirements work packages, that fit the project time box.

References

1. A. J. Albrecht. Measuring Application Development Productivity. In *Proc. of SHARE, GUIDE, and IBM App. Dev. Symp.*, pages 83 – 92. IBM, 1979.

2. G. B. Alleman. *Performance-Based Project Management.* AMACOM, 2014.
3. V. d. Bianco and L. Lavazza. Applying the COSMIC Functional Size Measurement Method to Problem Frames. In *Proc. 14th ICECCS*. IEEE, 2009.
4. B. W. Boehm. *Software Engineering Economics.* Prentice Hall, 1981.
5. M. Cohn. *Agile Estimating and Planning.* PHPTR, 2005.
6. M. Cohn. Establishing a Common Baseline for Story Points. http://bit.ly/2dk13Gl, 2008.
7. M. Cohn. The Best Way to Establish a Baseline When Playing Planning Poker. http://bit.ly/2i5g1GF, 2016.
8. I. Côté, D. Hatebur, M. Heisel, H. Schmidt, and I. Wentzlaff. A Systematic Account of Problem Frames. In *Proc. of the EuroPLoP*. UVK, 2008.
9. M. Daneva, E. van der Veen, C. Amrit, S. Ghaisas, K. Sikkel, R. Kumar, N. Ajmen, U. Ramteerthkar, and R. Wieringa. Agile requirements prioritization in large-scale outsourced system projects: An empirical study. *Journal of systems and software*, 86(5):1333–1353, 2013.
10. W. E. Deming. *Out of the Crisis.* MIT-CAES, 2000.
11. D. Garmus and D. Herron. *Function Point Analysis - Measurement Practices for Successful Software Projects.* Addison-Wesley, 2001.
12. C. Hazan. *The 13 Mistakes of Function Point Counting*, chapter 11, pages 197 – 214. In International Function Point Users Group [16], 2012.
13. M. Heisel. UML4PF eclipse plugin. http://www.uml4pf.org/.
14. M. Heisel. Agendas – A Concept to Guide Software Development Activites. In *Proc. Systems Implementation 2000*, pages 19–32. C&H, London, 1998.
15. A. Hunger. Student Recruitment Web Portal. http://bit.ly/2kmU2Ja, 2017.
16. International Function Point Users Group. *The IFPUG Guide to IT and Software Measurement.* Taylor & Francis Group, 2012.
17. ISO/IEC 20926:2009. *Software and systems engineering - Software measurement – IFPUG functional size measurement method 2009.* ISO, 2009.
18. M. A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices.* Addison-Wesley, New York, NY, USA, 1995.
19. M. A. Jackson. *Problem Frames – Analyzing and Structuring Software Development Problems.* Addison-Wesley, 2001.
20. L. Lavazza and V. del Bianco. Functional size measurement based on problem frames: A case study. In *Proc. of the 3rd IWAAPF*. ACM, 2008.
21. IFPUG. *IFPUG CPM 4.3.1 – Function Point Counting Practices Manual (CPM) Version 4.3.1.* International Function Point Users Group, 2010.
22. R. Meli. *Software Measurement in Procurement Contracts*, chapter 29, pages 561 – 583. In International Function Point Users Group [16], 2012.
23. K. Schwaber and J. Sutherland. Scrum Guide™. http://bit.ly/1rulpv2, 2016.
24. I. Sommerville. Integrated Requirements Engineering: A Tutorial. *IEEE Software*, 22(1):16 – 23, 2005.
25. Total Metrics. Function Point FAQs. http://bit.ly/2jtEe9X, 2016.
26. S. Trudel, J.-M. Desharnais, and J. Cloutier. Functional Size Measurement Patterns: A Proposed Approach. In *IWSM Mensura, Berlin*, 2016.
27. K. Waters. *All about Agile.* Createspace, 2012.
28. I. Wentzlaff. Frame Counting Agenda (.pdf). http://bit.ly/2jaDTYY, 2017.