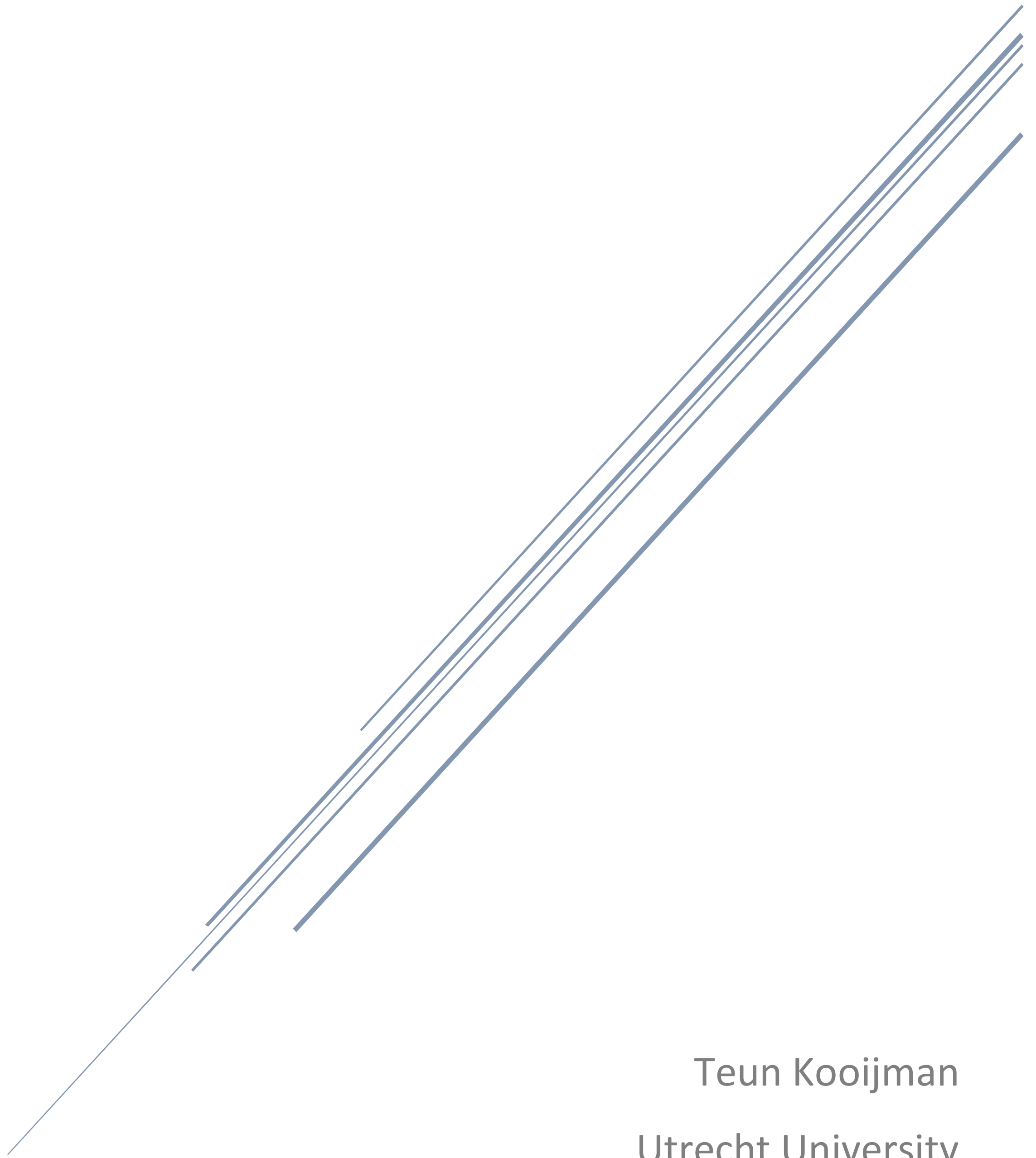
# A MODEL FOR SOFTWARE DEVELOPMENT TEAM PERFORMANCE



Teun Kooijman

Utrecht University

Master Thesis

**Author:**

T.S. Kooijman, B.Sc.

Student ID: 3970183

t.s.kooijman@students.uu.nl

**First Supervisor:**

dr. ir. J.M.E.M. van der Werf

j.m.e.m.vanderwerf@uu.nl

Utrecht University

**Second Supervisor:**

dr. S.J. Overbeek

s.j.overbeek @uu.nl

Utrecht University

**External Supervisor:**

F. Verbruggen, M.Sc.

frank@diamondagile.net

Diamond Agile B.V.

# Contents

Contents 2
1. Introduction 4
2. Research Methods 5
    2.1 Research Questions 5
    2.2 Methods 6
3. Systematic Literature Review 8
    3.1 Methods 8
    3.2 Execution 9
    3.4 Discovered Metrics 17
    3.5 Overview 58
4. Expert Inquiry 59
    4.1 Experts 59
    4.2 Suggested Metrics 60
5. Metric Quality Model 68
    5.1 Qualities 68
6. Systematic Mapping 70
    6.1 Data Structures 70
    6.2 Technology 70
    6.3 Axial Encoding 71
7. Aggregation 79
    7.1 Venues 79
    7.2 Publishers 79
    7.3 Papers 81
    7.4 Keywords 81
    7.5 Authors 82
    7.6 Metric Quality Assessment 84
    7.7 Metric Distributions 84
8. Team Performance Model 90
    8.1 Candidate Metrics 90
    8.2 Perspectives 91

8.3 Input Correlation ... 92
8.4 Tooling ... 94
9. Validation ... 98
10. Discussion ... 107
10.1 Metrics ... 107
10.2 Models ... 109
10.3 Threats to Validity ... 110
11. Conclusion ... 113
11.1 Research Questions ... 113
11.2 Limitations ... 114
11.3 Future Work ... 115
12. Acknowledgements ... 117
13. References ... 118
Appendix A - Google Scholar Candidates ... 133
Appendix B - Scopus Candidates ... 136
Appendix C - Snowballing Results ... 141
Appendix D - Metric Introductions ... 143
Appendix E - Metrics per Paper ... 147
Appendix F - Papers per Metric ... 150
Appendix G - Full Aspect Encoding ... 156
Appendix H - Full Input Encoding ... 162
Appendix I - Keyword Occurrences ... 165
Appendix J - Author Occurrences ... 170
Appendix K - Metric Quality Assessment ... 173
Draft Paper ... 179

# 1. Introduction

According to the yearly State of Agile surveys, an increasing amount of software development companies are embracing Agile in order to increase their performance. It's methodologies boast many positive effects, such as more flexible projects, reductions in project duration, increases in adaptation and satisfaction, fewer deadline-transcending projects, and lower overall project costs. One of the most prominent and oft-quoted advantage of the Agile approach is the claim that it makes your development process *more efficient* (Sutherland, 2014), (Prechelt, 2019), (Leffingwell, 2018).

The methodology provides a number of ways to measure a team's performance, such as *Story Point Velocity* or *Focus Factor* (Padmini, Bandara & Perera, 2015), yet it remains to be seen if these are *strong* measures of software development team performance. If not, this means that management will not be able to accurately determine which teams are performing admirably, or even extremely well, and which teams are not. At the same time, individual team members will not know whether their team is excelling or failing.

The problem with these performance metrics is in their manner of size estimation. All size estimation is done in terms of *Story Points*. *Story Points* are estimated from the expert opinions of the team that is going to perform the work. First, an initial reference story is set to an arbitrary number of *Story Points*. From there, the team members estimate the *Story Points* of the other stories in a *relative* fashion, comparing the size of the work of the new story to that of the reference story and other, already estimated stories. Their expert opinions are based on relative estimates of the effort required for implementing the story, but are likely to be coloured by their experience, expertise, technical aptitude, or even ulterior motives. This means that the size estimation of a story in terms of *Story Points*, may differ quite substantially from team to team. Consequently, a comparison between their performance in terms of their *Story Point Velocity* or *Focus Factor*, is inherently flawed, and easy to game.

Until an accurate assessment of software development performance can be performed, the inability for an organization to accurately determine the benefits that the adoption of Agile has brought them in terms of efficiency, remains (Oszewska, 2016).

**[Problem Statement]** In this thesis, we will attempt to devise or consider a new model, which can be used to accurately measure the performance of a software development team. The following chapters of this thesis describe the execution of this attempt.

# 2. Research Methods

In this chapter, we will devise a primary research question and divide it into several sub-questions. Finally, we will go over the research methods that we will use for our investigation.

## 2.1 Research Questions

This section introduces the main research question. By answering this question, we will have defined or considered a new set of metrics for measuring team performance. The main research question is outlined below, labelled **R1**.

**R1:** *How can we measure the performance of a*

*software development team?*

Subsequently, the main research question is supported by three additional sub-questions. These sub-questions are outlined below, labelled **S1** through **S3.**

Before we can determine how we can accurately measure the performance of a software development team, we need to determine what constitutes an accurate and objective metric in the first place. The following sub-question addresses this need.

**S1:** *What constitutes a strong software development metric?*

In order to determine what constitutes a valid and accurate efficiency metric, we need to determine what metrics already exist today. Additionally, we will need to extrapolate what makes these metrics valid or accurate. The following sub-question addresses this.

**S2:** *Which software development metrics already exist today?*

Finally, we need to determine what set of software development metrics is most suitable for measuring team performance, based on their strength, their domain, and the potential correlations caused by a shared set of input data-points. The following sub-question addresses this need.

**S3:** *What set of software development metrics is most suitable*
*for measuring team performance?*

## 2.2 Methods

In our attempt to answer the various research and sub-questions, we will employ a *Grounded Theory* approach, consisting of a data-collection phase, and a data-structuring phase. Afterwards, two new models are constructed, based on discussions and conversations about the collected data with prevalent experts in the field. Finally, the constructed models are subjected to a preliminary validation, gauging their perceived clarity, relevance and completeness.

### 2.2.1 Data Collection

#### *2.2.1.1 Structured Literature Review*

In order to determine the current state of the literature, we will perform a structured literature review, consisting of two phases. In the first phase, an automated search on Google Scholar and Scopus is performed, followed by the second phase, in which the results are snowballed, based on the snowballing technique outlined in (Wohlin, 2014). The aim of this investigation is to determine what software development metrics exist in literature today.

#### *2.2.1.2 Expert Inquiry*

Additionally, an *expert inquiry* will be held among some prevalent experts in the field, where the collected set of metrics will be presented and discussed. The aim of this *expert inquiry* is to determine additional software development metrics which could potentially be *strong* metrics for team performance, that have not been discovered in literature.

### 2.2.2 Data Structuring

#### *2.2.2.1 Axial Encoding*

The software development metrics that are found, as well as the aspects of the software development process that they target, and their individual input data-points, are then processed using the *Grounded Theory* approach of *axial encoding*. Here, the concepts will be encoded into a final set of aspects and input groups.

#### *2.2.2.2 Systematic Mapping*

The software development metrics that are found, will be systematically mapped in a graphing database, along with the inputs required to calculate them, the papers that mention them, the authors who wrote them, the keywords those papers use, the journals in which they were published, and the publishers who published them. This systematic mapping is subsequently used for the theory building phase that follows.

### 2.2.3 Grounded Theory Building

#### *2.2.3.1 Model for Metric Strength*

The software development metrics that are found, as well as their aspects and inputs, and the systematic mapping, are presented to and discussed with the experts. In these discussions, we will attempt to extrapolate the expert's tacit knowledge about determining which metrics can be considered *strong*, and which metrics can be considered *weak*. This tacit knowledge will then be distilled in a newly devised model for metric strength. All of the encountered metrics can subsequently be assessed on their strength, using the new model.

#### *2.2.3.2 Model for Team Performance*

Subsequently, we construct a model for team performance, based on additional discussions and conversations with the experts. This model is to focus on a set of metrics targeting a broad set of software development process aspects, while sharing a minimum amount of input data-points so that cause-and-effect can be more easily isolated.

#### *2.2.3.3 Validation*

Finally, we probe the perceived clarity, relevance and completeness of the models using a preliminary validation survey among professionals in the field. A thorough validation process is postponed to future work.

### 2.2.4 Deliverables

The primary deliverable of this thesis will thus be the systematic mapping of the available software development metrics in industry and literature, which is embodied in a graph database. Similarly, a newly devised model for metric strength will be constructed, as well as a newly devised model for team performance.

# 3. Systematic Literature Review

In this chapter, we will detail the execution and the results of our systematic literature review, consisting of two phases. The first phase identifies a starting set through an automated search process, whereas the second phase aims to identify missing work, based on Wohlin's snowballing technique (2014), until an iteration no longer results in additional discovered relevant metrics. The aim of this review is to discover as many software development metrics as possible. The process denoted in the remainder of this section was performed on Google Scholar, and duplicated on Scopus.

## 3.1 Methods

### 3.1.1 Inclusion Criteria

The inclusion criteria used for selecting or discarding literature was kept as broad as possible. The selected papers should be written in English, and should be published in a peer-reviewed journal, or presented at a venue which was facilitated by a peer-reviewed journal, such as a conference or workshop. We will not employ inclusion criteria based on year of publication, specific authors or specific journals. The latter two because we want to evade any such bias, and the former because we deem year of publication to be irrelevant to our purpose. The final decision on whether or not to include a piece of literature is done through manual examination of the candidate work. Here, the abstract of the candidate is examined, and if needed, the paper is thoroughly studied. In this examination, we will look for the presence of metrics in the work, that are deemed relevant to the field of software development. In the context of this review, a *relevant* metric is defined as a metric that can be used to measure any aspect of a software development process. This results in the following collection of inclusion and exclusion criteria:

- **[exclusion]** is not written in English;
- **[exclusion]** is not published in a peer-reviewed journal, or not presented at a venue which was facilitated by a peer-reviewed journal;
- **[inclusion]** must mention *relevant* software development metrics that have not yet been mentioned in previous snowballing iterations.

### 3.1.2 Approach

A starting set of literature will be selected using an automated search on four different queries. These queries will be executed on the academic search engines Scopus and Google Scholar. From this starting set, backward and forward snowballing will be performed until an iteration no longer yields any additional included work.

The following sections will detail the execution details of our systematic literature review, and we will end this chapter by providing a brief summary and discussion of the collected work.

## 3.2 Execution

### 3.2.1 Start Set

The search for the starting set of literature was performed on Tuesday the 22nd of January, 2019. In the following section, we will introduce each of the search queries used to generate a part of the starting set. Note that these search queries were devised so as to get an extremely varied start set. This was done by approaching the field from many different angles. The result sets for some of these queries on Google Scholar were so large, that complete analysis was unfeasible for the size and scope of this study. This has caused us to make compromises in terms of validity, for the sake of time. This means that, instead of analysing over 3.500.000 results in order to generate a starting set, only the first ten results were considered for inclusion when performing the automated search on Google Scholar. On Scopus, however, the entire result set was considered, as it was significantly smaller. This consolidation is a severe threat to the validity of our results.

The search queries performed on both Google Scholar and Scopus are listed below in *table 1*, together with the total amount of results that were returned.

| **Search Engine** | **Query** | **Results** |
|---|---|---|
| Google Scholar | Software Development Metrics | 3.590.000 |
| Google Scholar | Agile Efficiency Metrics | 42.200 |
| Google Scholar | Scrum Productivity Metrics | 7.170 |

| Google Scholar | Agile Productivity | 115.000 |
|---|---|---|
| Scopus | Software Development Metrics | 14 |
| Scopus | Agile Efficiency Metrics | 42 |
| Scopus | Scrum Productivity Metrics | 7 |
| Scopus | Agile Productivity | 1 |

*Table 1 - Automated Search Queries and Amount of Results*

***3.2.1.1 Google Scholar***

The results of the automated search on Google Scholar are outlined in *Appendix A*. From this set of candidate literature, we extract a start set of literary work according to the set of inclusion and exclusion criteria introduced in *section 3.1.1*. The first criteria is whether or not the work is written in English. All of the candidates pass this criteria. Subsequently, the next criteria is whether or not the work is published in a peer-reviewed journal. This criteria eliminates candidates **GS.2.03**, **GS.2.08** and **GS.2.09**. Finally, the candidates that were left over were inspected in more detail, to determine whether or not it mentions relevant software development performance metrics, as described in section 3.1.1. The results of the application of the inclusion criteria can be found in *table 2* below. This lead to the inclusion of 12 candidate works.

| Candidate | English | Peer Reviewed | Relevant Metrics | Included |
|---|---|---|---|---|
| | | | | |
| GS.1.01 | Yes | Yes | No | No |
| GS.1.02 | Yes | Yes | Yes | Yes |
| GS.1.03 | Yes | Yes | No | No |
| GS.1.04 | Yes | Yes | Yes | Yes |
| GS.1.05 | Yes | Yes | No | No |
| GS.1.06 | Yes | Yes | No | No |
| GS.1.07 | Yes | Yes | No | No |
| GS.1.08 | Yes | Yes | No | No |
| GS.1.09 | Yes | Yes | No | No |
| GS.1.10 | Yes | Yes | No | No |
| | | | | |
| GS.2.01 | Yes | Yes | No | No |
| GS.2.02 | Yes | Yes | No | No |
| GS.2.03 | Yes | No | **N.A.** | No |
| GS.2.04 | Yes | Yes | No | No |

| GS.2.05 | Yes | Yes | No | No |
|---|---|---|---|---|
| GS.2.06 | Yes | Yes | Yes | Yes |
| GS.2.07 | Yes | Yes | No | No |
| GS.2.08 | Yes | No | **N.A.** | No |
| GS.2.09 | Yes | No | **N.A.** | No |
| GS.2.10 | Yes | Yes | No | No |
| | | | | |
| GS.3.01 | Yes | Yes | Yes | Yes |
| GS.3.02 | Yes | Yes | Yes | Yes |
| GS.3.03 | Yes | Yes | No | No |
| GS.3.04 | Yes | Yes | Yes | Yes |
| GS.3.05 | Yes | Yes | Yes | Yes |
| GS.3.06 | Yes | Yes | Yes | Yes |
| GS.3.07 | Yes | Yes | Yes | Yes |
| GS.3.08 | Yes | Yes | No | No |
| GS.3.09 | Yes | Yes | No | No |
| GS.3.10 | Yes | Yes | Yes | Yes |
| | | | | |
| GS.4.01 | Yes | Yes | No | No |
| GS.4.02 | Yes | Yes | No | No |
| GS.4.03 | Yes | Yes | No | No |
| GS.4.04 | Yes | Yes | Yes | Yes |
| GS.4.05 | Yes | Yes | No | No |
| GS.4.06 | Yes | Yes | Yes | Yes |
| GS.4.07 | Yes | Yes | No | No |
| GS.4.08 | Yes | Yes | No | No |
| GS.4.09 | Yes | Yes | No | No |
| GS.4.10 | Yes | Yes | No | No |

*Table 2 - Applied Inclusion Criteria on Google Scholar (GS) Candidates*

#### *3.2.1.2 Scopus*

The results of the automated search on Scopus are outlined in *Appendix B*. From this set of candidate literature, we extract the second part of the starting set of literary work according to the set of inclusion and exclusion criteria introduced in *section 3.1.1*. All of the candidates are written in English and published in a peer-reviewed journal, or presented at a venue that is facilitated by a peer-reviewed journal. Then, 18 candidates were found to mention relevant software development metrics, which were subsequently included in the starting set. The results of applying these criteria are shown below in *table 3.*

| Candidate | English | Peer Reviewed | Relevant Metrics | Included |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| SC.1.01 | Yes | Yes | Yes | Yes |
| SC.1.02 | Yes | Yes | No | No |
| SC.1.03 | Yes | Yes | Yes | Yes |
| SC.1.04 | Yes | Yes | No | No |
| SC.1.05 | Yes | Yes | Yes | Yes |
| SC.1.06 | Yes | Yes | No | No |
| SC.1.07 | Yes | Yes | No | No |
| SC.1.08 | Yes | Yes | No | No |
| SC.1.09 | Yes | Yes | No | No |
| SC.1.10 | Yes | Yes | No | No |
| SC.1.11 | Yes | Yes | No | No |
| SC.1.12 | Yes | Yes | No | No |
| SC.1.13 | Yes | Yes | Yes | Yes |
| | | | | |
| SC.2.01 | Yes | Yes | No | No |
| SC.2.02 | Yes | Yes | No | No |
| SC.2.03 | Yes | Yes | No | No |
| SC.2.04 | Yes | Yes | No | No |
| SC.2.05 | Yes | Yes | No | No |
| SC.2.06 | Yes | Yes | Yes | Yes |
| SC.2.07 | Yes | Yes | No | No |
| SC.2.08 | Yes | Yes | Yes | Yes |
| SC.2.09 | Yes | Yes | Yes | Yes |
| SC.2.10 | Yes | Yes | No | No |
| SC.2.11 | Yes | Yes | No | No |
| SC.2.12 | Yes | Yes | No | No |
| SC.2.13 | Yes | Yes | No | No |
| SC.2.14 | Yes | Yes | No | No |
| SC.2.15 | Yes | Yes | No | No |
| SC.2.16 | Yes | Yes | No | No |
| SC.2.17 | Yes | Yes | No | No |
| SC.2.18 | Yes | Yes | No | No |
| SC.2.19 | Yes | Yes | Yes | Yes |
| SC.2.20 | Yes | Yes | No | No |
| SC.2.21 | Yes | Yes | Yes | Yes |
| SC.2.22 | Yes | Yes | No | No |
| SC.2.23 | Yes | Yes | No | No |
| SC.2.24 | Yes | Yes | No | No |
| SC.2.25 | Yes | Yes | No | No |
| SC.2.26 | Yes | Yes | No | No |
| SC.2.27 | Yes | Yes | No | No |

| SC.2.28 | Yes | Yes | No | No |
|---|---|---|---|---|
| SC.2.29 | Yes | Yes | No | No |
| SC.2.30 | Yes | Yes | No | No |
| SC.2.31 | Yes | Yes | No | No |
| SC.2.32 | Yes | Yes | No | No |
| SC.2.33 | Yes | Yes | Yes | Yes |
| SC.2.34 | Yes | Yes | No | No |
| SC.2.35 | Yes | Yes | Yes | Yes |
| SC.2.36 | Yes | Yes | No | No |
| SC.2.37 | Yes | Yes | Yes | Yes |
| SC.2.38 | Yes | Yes | No | No |
| SC.2.39 | Yes | Yes | No | No |
| SC.2.40 | Yes | Yes | No | No |
| SC.2.41 | Yes | Yes | No | No |
| SC.2.42 | Yes | Yes | Yes | Yes |
| | | | | |
| SC.3.01 | Yes | Yes | No | No |
| SC.3.02 | Yes | Yes | No | No |
| SC.3.03 | Yes | Yes | No | No |
| SC.3.04 | Yes | Yes | Yes | Yes |
| SC.3.05 | Yes | Yes | Yes | Yes |
| SC.3.06 | Yes | Yes | Yes | Yes |
| SC.3.07 | Yes | Yes | Yes | Yes |
| | | | | |
| SC.4.01 | Yes | Yes | Yes | Yes |

*Table 3 - Applied Inclusion Criteria on Scopus (SC) Candidates*

### 3.2.2 Snowballing

From here on out, no more distinctions will be made between work retrieved using Scopus, and work retrieved using Google Scholar. For all included work, snowballing will be performed until an iteration no longer yields *new* software development metrics. As snowballing has a huge potential for blowing up, and this study is limited in terms of time and resources, references will initially be judged on their title. Only if a title indicates that the work might mention new software development metrics, is the reference inspected more closely. If it is then determined that the work mentions no new relevant software development metrics, it is discarded after all, and the snowballing path for that branch ends there.

#### *3.2.2.1 Backwards Snowballing*

The backwards snowballing process identified 18 additional literary works to be included in the review. These additional works were identified in four iterations of backwards snowballing. The details of these iterations can be found in *Appendix C*.

#### *3.2.2.2 Forwards Snowballing*

Forwards snowballing was performed by using the Google Scholar search tools to repeat the defined queries on each of the starting set's paper's citations. It is interesting to note here, that this process did not actually yield any additional work to be included. We presume that this is primarily because of the fact that it was performed *after* having performed the backwards snowballing, causing all of the work to be excluded due to the fact that it mentioned no *new* software development metrics. This points at a decent probability of having included a significant portion of the available body of knowledge. While we find that the probability is quite low, there might still exist clusters of literary work that are completely separated from any of the works in the starting set.

#### *3.2.2.3 Visual Representation*

A visual representation of the relationships between works included in the start set, and those identified through the snowballing process, is shown below in *figure 1*.

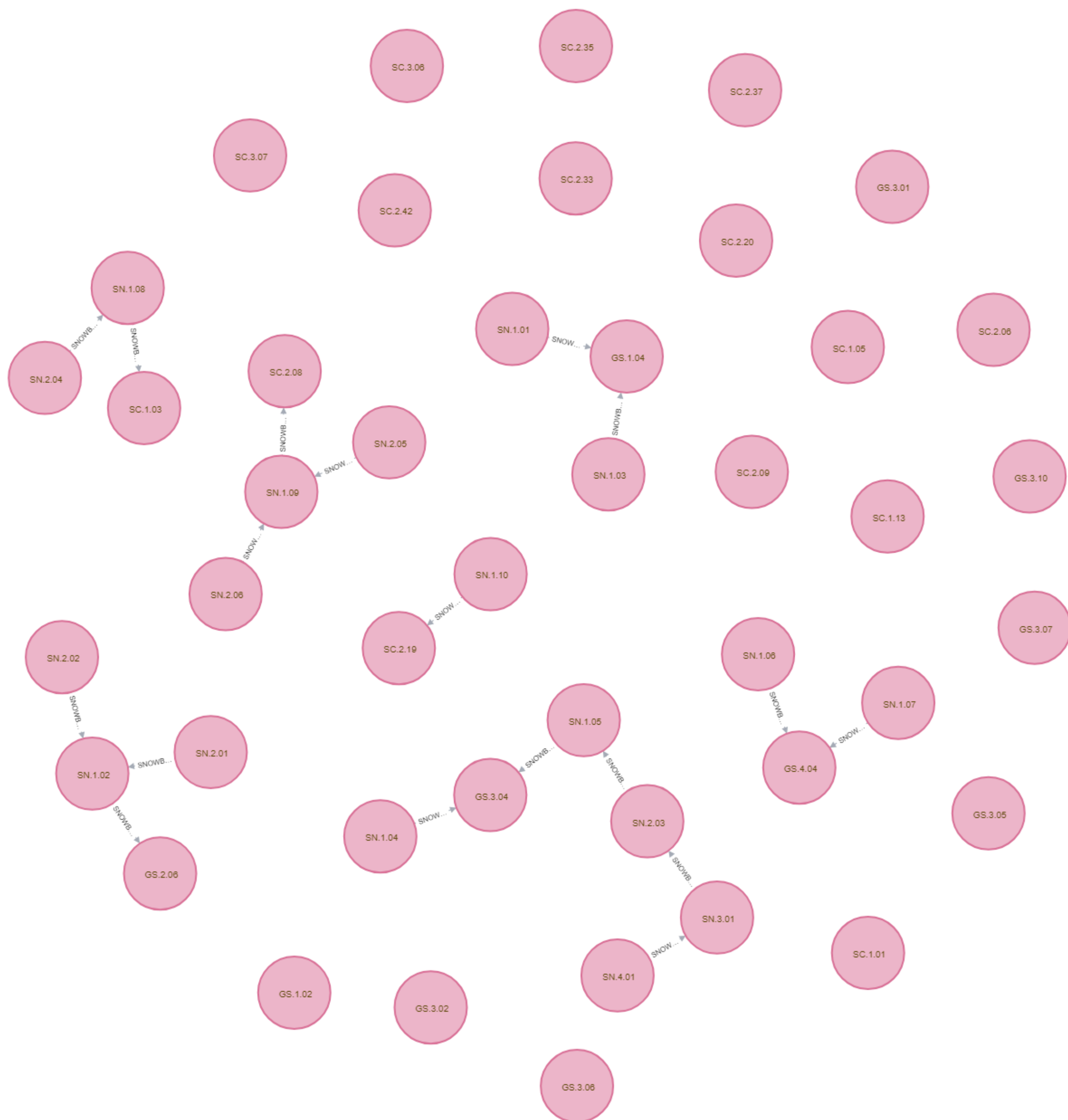



*Figure 1 - Included Work and their Snowballing Relationships*

### 3.2.3 Rebranding

Before we continue, we re-brand the selected work according to the mapping displayed in *table 4*. Note that **GS.4.06** is actually the same work as **GS.3.07**, and has therefore been left out of the table. The same applies to **SC.3.04**, being a duplicate of **SC.2.20, SC.3.05**, being a duplicate of **GS.3.01**, and **SC.4.01**, being a duplicate of **GS.3.10**. In total, the Scopus and Google Scholar candidate sets showed little overlap, with only 6 overlapping candidate works.

| Source | Candidate ID | Inclusion ID | Reference |
|---|---|---|---|
| **Google Scholar** | | | |
| | GS.1.02 | P1.01 | Jefferey, Ruhe & Wieczorek, 2001 |
| | GS.1.04 | P1.02 | Boehm, Abts & Chulani, 2000 |
| | GS.2.06 | P1.03 | Padmini, Bandara & Pererea, 2015 |
| | GS.3.01 | P1.04 | Downey & Sutherland, 2013 |
| | GS.3.02 | P1.05 | Greening, 2010 |
| | GS.3.04 | P1.06 | Agarwal & Majumdar, 2012 |
| | GS.3.05 | P1.07 | Sutherland, Schoonheim & Rijk, 2009 |
| | GS.3.06 | P1.08 | Sutherland, Harrison & Riddle, 2014 |
| | GS.3.07 | P1.09 | Maurer & Martel, 2002 |
| | GS.3.10 | P1.10 | Shah, Papatheocharous & Nyfjord, 2015 |
| | GS.4.04 | P1.11 | Moser, Abrahamsson, pedrycz, Sillitti & Succi, 2008 |
| **Scopus** | | | |
| | SC.1.01 | P1.12 | Bhardwaj & Rana, 2016 |
| | SC.1.03 | P1.13 | Fitzegrald, Musial & Stol, 2014 |
| | SC.1.05 | P1.14 | Calikli, Bener, Aytac & Bozcan, 2013 |
| | SC.1.13 | P1.15 | Moreau & Dominick, 1989 |
| | SC.2.06 | P1.16 | Beer & Felderer, 2018 |
| | SC.2.08 | P1.17 | Alfraihi, Lano Kolahdouz-Rahimi, Sharbaf & Haughton, 2018 |
| | SC.2.09 | P1.18 | Lano, Alfraihi, Kolahdouz-Rahimi, Sharbaf & Haughton, 2018 |
| | SC.2.19 | P1.19 | Rosero, Gómez & Rodríguez, 2016 |
| | SC.2.21 | P1.20 | Grimaldo, Perrotta, Corvello & Verteramo, 2016 |
| | SC.2.33 | P1.21 | Cuatrecasas-Arbos, Fortuny-Santos & Vitro-Sanchez, 2011 |
| | SC.2.35 | P1.22 | Koru & El Emam, 2009 |
| | SC.2.37 | P1.23 | Khadem, Ali & Seifoddini, 2008 |
| | SC.2.42 | P1.24 | Kupiainen, Mäntylä & Itkonen, 2015 |
| | SC.3.06 | P1.25 | Huijgens & Van Solingen, 2013 |
| | SC.3.07 | P1.26 | Sjøberg, Johnsen & Solberg, 2012 |
| **Snowballing Iteration 1** | | | |
| | SN.1.01 | P2.01 | Minkiewicz, 1998 |
| | SN.1.02 | P2.02 | Oza & Korkala, 2012 |
| | SN.1.03 | P2.03 | Kunz, Dumke & Zenker, 2008 |
| | SN.1.04 | P2.04 | Kemerer & Paulk, 2009 |
| | SN.1.05 | P2.05 | Rosenburg & Hyatt, 1997 |
| | SN.1.06 | P2.06 | Demeyer, Ducasse & Nierstrasz |
| | SN.1.07 | P2.07 | Sahraoui, Godin & Miceli, 2000 |
| | SN.1.08 | P2.08 | Petersen & Wohlin, 2011 |
| | SN.1.09 | P2.09 | He, Avgeriou, Liang & li, 2016 |

| | SN.1.10 | P2.10 | Leung & White, 1991 |
|---|---|---|---|
| **Snowballing Iteration 2** | | | |
| | SN.2.01 | P2.11 | Hartmann & Dymond, 2006 |
| | SN.2.02 | P2.12 | Mahnic & Zabkar, 2008 |
| | SN.2.03 | P2.13 | Tegarden, Sheetz & Monarchi, 1992 |
| | SN.2.04 | P2.14 | Damm, Lundberg & Wohlin, 2006 |
| | SN.2.05 | P2.15 | Fontana, Braione & Zanoni, 2012 |
| | SN.2.06 | P2.16 | Li, Liang, Avgeriou & Guelfi, 2014 |
| **Snowballing Iteration 3** | | | |
| | SN.3.01 | P2.17 | Oliveira, Redin, Carro, Da Cuhna Lamb & Wagner, 2008 |
| **Snowballing Iteration 4** | | | |
| | SN.4.01 | P2.18 | Aggarwal, Singh, Kaur & Malhotra, 2006 |

*Table 4 - All Included Work*

# 3.4 Discovered Metrics

The following section details the results of the analysis of the included work.

## 3.4.1 Scrum Metrics

The first category of discovered metrics are those that are closely related to Scrum. Being the most prevalent Agile framework in the world, Scrum advertises itself as *a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value*. In its attempt to deliver, it is lightweight, highly iterative, and extremely empirical.

### *3.4.1.1 Story Points and Velocity*

The most prevalent metric we encountered for size estimation is *Story Points*. This metric was mentioned by **P1.03**, **P1.04**, **P1.05**, **P1.06**, **P1.07**, **P1.12**, **P1.24**, **P2.02**, **P2.11**, and **P 2.12**. Most of these also mentioned the efficiency metric of *Velocity*. These metrics are widely known and used throughout the agile community, as they are a standard part of Scrum. As explained in the introduction, story point estimation employs a Delphi approach towards size estimation, and thus leads to a velocity that is heavily coupled to the professionals who performed the estimation. This in turn results in a metric that is difficult to compare across different teams.

#### *3.4.1.2 Accuracy of Forecast and Accuracy of Estimation*

Study **P1.03**, as well as **P1.04**, mention the *Accuracy of Forecast* and *Accuracy of Estimation* metrics. While both of these metrics are closely related to Scrum, they are not an integrated part of the standard Scrum methodology. They are also closely related to the metrics of *Percentage of Found Work* and *Percentage of Adopted Work*. The *Accuracy of Estimation* refers to the team's ability to accurately estimate their *story backlog items*. Measuring the *Accuracy of Estimation* involves adjusting the story point estimate of user stories post-sprint, to reflect the actual effort that was involved in implementing the story. The *Accuracy of Estimation* is then defined by the sum of the changes in story point estimates, divided by the total initial story point commitment of the sprint. This definition is outlined below in *equation 1*.

$$Accuracy\ of\ Estimation = 1 - \frac{\sum(Estimate\ Deltas)}{\sum(Original\ Estimates)}$$

*Equation 1 – Accuracy of Estimation*

The author of these metrics, Jeff Sutherland, implies that the ideal value for this metric is somewhere between 72% and 88%. A value higher than that indicates that the team is spending an inordinate amount of time researching and digesting information on what the story backlog item entails. Lower than 72% indicates that the story backlog items are too poorly understood upon estimation, and signals that an investigation on outside pressures on the team is necessary by the Scrum Master.

The *Accuracy of Forecast* instead measures the team's ability to accurately estimate the work they can accomplish in *their sprints*. The metric looks at the sum of the original estimates of a sprint's story backlog items, and compares it to the sum of the actual effort that was involved in implementing it. The latter is defined by the sum of the original estimates of a sprint's story backlog items, as well as the sum of their additional found work, and the sum of the adopted work. The latter two metrics are described in *section 3.4.1.3*. It's official definition is outlined below in *equation 2.*

$$Accuracy\ of\ Forecast = \frac{\sum(Original\ Estimates)}{\sum(Original\ Estimates) + \sum(Found\ Work) + \sum(Adopted\ Work)}$$

*Equation 2 - Accuracy of Forecast*

Here, Jeff Sutherland implies that the ideal value for this metric is somewhere between 75% and 90%. Any higher than that, and the Scum Master will need to evaluate the environment of the team to make sure that they feel safe making a good faith effort at more work. Lower than 75% usually indicates that the team is not sufficiently protected by the Scrum Master from outside forces, leading to story backlog items inside the sprint that aren't sufficiently worked out, have no clear definition of done, or are still waiting for dependencies to be resolved.

In the end, study **P1.03** found that 54% of the teams they investigated used this metric, while 58% of the teams used the *Accuracy of Estimation* metric. This lead them to be the $16^{th}$ and shared $17^{th}$/$18^{th}$/$19^{th}$/$20^{th}$ most used metric they encountered.

#### *3.4.1.3 Percentage of Adopted Work and Percentage of Found Work*

Both **P1.03** and **P1.04** mention the metrics of P*ercentage of Adopted Work* and P*ercentage of Found Work*. The first study found 81% of the teams they interviewed used the first of the two metrics, making it the fourth most commonly used metric of the 22 that they encountered. Adopted work is defined as new work pulled into the sprint, because the team has completed its forecast. The percentage of adopted work is then defined as the sum of the original estimates of the adopted work, divided by the sum of the original estimates of the items that were initially included in sprint.

Similarly, found work is defined as the unexpected extra work that is necessary to complete a backlog item. The percentage of found work is then defined as the sum of the increase in estimates, divided by the sum of the original estimates of the items that were initially included in the sprint. These metrics can be useful post-sprint to determine how accurate the initial estimates were.

Closely related to these metrics are the *Average Number of Stories Added to an Iteration* and the *Average Number of Stories Removed from an Iteration*. These metrics were mentioned by **P2.02**, and essentially measure the same concepts, yet are not normalized on story point estimates.

#### *3.4.1.4 Effort Burndown*

Additionally, studies **P1.03**, **P1.04**, **P1.06** and **P1.24** also mention the *Sprint Level Effort Burndown*. This metric is also widely adopted as a standard part of the Scrum process, but will most likely require some form of automated tooling to measure. The metric determines the relationship between the remaining work capacity and the remaining estimated effort of a sprint over the duration of that

sprint. Ideally, this relationship should have a negative linear direction from 100% to 0% work capacity and remaining effort. This metric gives the team a post-sprint indication of how accurate their estimation of the backlog items in that sprint was. Similarly, the *Release Level Effort Burndown* is also mentioned by study **P1.24**, shifting the scope of the metric from an individual sprint to an entire release.

#### *3.4.1.5 Enterprise Velocity*

Study **P1.05** mentions scaling the *Story Point Velocity* metric to an entire enterprise, dubbing the resulting metric the *Enterprise Story Points* and *Enterprise Velocity.* The paper states that an *Enterprise Story Point* is roughly equivalent to "estimated *team months times 100"*. This seemingly arbitrary measure is then used to determine the velocity of the entire organization, as opposed to a single team.

#### *3.4.1.6 Focus Factor*

**P1.03** and **P1.04** mention the *Focus Factor* metric. The metric is defined as the story point velocity of a team, divided by the total amount of hours (or *Work Capacity*) that the team has spent on the project during that sprint. It essentially measures the story point velocity per working-hour, instead of per sprint, and can thus be used to normalize velocity over the amount of team members.

#### *3.4.1.7 Stories per Iteration*

While most studies mention using *Story Point Velocity* when trying to gauge efficiency, **P2.02** also mentions using a raw count of the amount of stories delivered per iteration as a means of measuring efficiency, but it does not take in to account the apparent size differences between stories at all.

#### *3.4.1.8 Stories per Day per Developer*

Similarly, **P2.02** mentions using a raw count of the amount of S*tories Delivered per Day per Developer* as a means of measuring efficiency. Again, this metric does not take into account the apparent size differences between stories at all, and can be expected to fluctuate more wildly than its *per iteration* cousin.

#### *3.4.1.9 Targeted Value Increase*

Studies **P1.03** and **P1.04** both mention the *Targeted Value Increase* metric. It denotes by how much the team has overshot (or undershot) their expected story point velocity for a particular sprint. In essence, it is a ratio between the sprint's actual velocity and the velocity that was forecast based on historical data. The official definition it outlined below in *equation 3.*

$$Targeted\ Value\ Increase = \frac{Sprint's\ Actual\ Velocity}{Expected\ Velocity}$$

*Equation 3 - Targeted Value Increase*

In **P1.03**, the metric was the least encountered metric, with only 35% of the investigated teams indicating that they used the metric.

#### *3.4.1.10 Success at Scale*

Studies **P1.03** and **P1.04** introduce the *Success at Scale* metric. This metric can actually be calculated for each value along the *Fibonacci sequence*, which are used to estimate story backlog items in terms of story points. The metric works by looking at all of the stories that have been estimated with a specific *Fibonacci* value (e.g. all story backlog items that have been estimated as 8 story points worth of effort). Then, the *Success at Scale* metric for *Fibonacci* value 8 is defined as the total amount of story backlog items that were estimated as 8 story points worth of effort, that have ever been successfully implemented in a single sprint, divided by total amount of attempts to do so. This official definition is outlined below in *equation 4.*

$$Success\ at\ Scale(i) = \frac{Succesful\ single\ sprint\ implementations\ of\ stories\ with\ estimate\ i}{Attempts\ to\ implement\ stories\ with\ estimate\ i\ in\ a\ single\ sprint}$$

*Equation 4 - Success at Scale*

The metric can help a team to determine whether or not it is wise to include a story backlog item of a particular effort estimation size into the sprint, even if it appears to fit within their projected velocity. The author does, however, stress that a team should never be denied the opportunity to try, but including a 13 point story backlog item when it has been unsuccessful 19 out of 20 times might be unwise, and the team might be better off splitting up the functionality into multiple, smaller stories.

#### *3.4.1.11 Win/Loss record*

The authors of **P1.03** and **P1.04** also introduce the concept of keeping a win/loss record. In their definition, a sprint can only be deemed a win if (a) a minimum of 80% of the work is accepted into production, and (b) the sum of the found and adopted work during the sprint remains at 20% or less of the original sprint forecast. The evolution of wins versus losses can then be tracked over time in order to determine whether or not the team is improving. This is essentially a measure of how capable the team is in estimating the required effort of a work-item.

#### *3.4.1.12 Yesterday's Weather*

In **P1.08**, the author introduces the *Yesterday's Weather* metric. It denotes the unabridged, absolute amount of story points that was delivered into production in the last sprint. Here, Jeff Sutherland, Neil Harrison and Joel Riddle argue that "*Yesterday's Weather is, in most cases, the most reliable predictor of how many story points the team will complete in the next sprint".*

#### *3.4.1.13 Summary*

While some of the discovered Scrum metrics, such as Story Points and Velocity, were introduced directly at the genesis of Scrum, many have been introduced more recently. Virtually all of these metrics tell the user something about their capabilities in terms of estimating required effort and projecting commitment. Yet, they tell very little about actual effort, productivity, performance or speed.

### 3.4.2 Lean Metrics

The *Lean Manufacturing* philosophy, originating from the industrial manufacturing industry, has had considerable impact on software development in the form of *Lean Software Development*. The philosophy promotes eliminating waste from the development process, and limiting the concurrent amount of work-in-progress, in order to minimize the lead-time of individual stories.

#### *3.4.2.1 Lead Time*

The metric of *Lead Time* was mentioned in studies **P1.03**, **P1.21**, **P1.23**, **P1.24** and **P1.26**, and measures the total time it takes for a particular component to go from conception to being delivered to its user. In the world of industrial manufacturing, this is the time it takes to create one additional product from start to finish (e.g. from when a new car is ordered to when it is delivered to the customer). In the world of software development, this is often the time between getting a request for a particular functionality from a customer, to having that functionality available to that customer in its production environment. This metric is also sometimes referred to as *Total Time*.

#### *3.4.2.2 Queue Time*

Studies **P1.03**, **P1.21**, **P1.24** and **P1.26** also mention the metric of *Queue Time*, which is most often defined in the world of software development as the time in which a particular story is defined, but is not yet picked up by a developer (e.g. sitting idle on a backlog). In industrial manufacturing, it is defined as the time in which a particular component "sits around waiting for someone to work on it". The shorter a team can keep its *Queue Time*, the shorter its *Lead Time* will be as a result.

#### *3.4.2.3 Cycle Time*

Studies **P1.23**, **P1.24** and **P2.02** also mention the metric of *Cycle Time*, which is defined as the *total amount of time that elapsed from the moment the work on that task is started, until its completion***.** In software development, this means the time from when a particular story is first being worked on, until its functionality is available in the production environment(s).

#### *3.4.2.4 Interrupted Time*

Study **P1.20** mentions the *Interrupted Time*, indicating the amount of time that was spent on a particular component, but did not produce any tangible outcome. An example of interrupted time could be a co-worker who comes and asks you a question for ten minutes, while you were working on a particular component. The *Interrupted Time* is closely related to the *Value-Added Time* introduced in the next section, as they are each other's inverse.

#### *3.4.2.5 Value Added Time*

Additionally, studies **P1.23** and **P2.08** also mention the Lean metric of *Value-Added Time*. This is defined as the amount of time that was spent on a particular component, that *did* produce tangible outcome. Study **P1.24** mentions the *Actual Development Time* metric, which is not clearly defined in the paper, but presumed to be the same as *Value Added Time*.

#### *3.4.2.6 Work in Progress*

Studies **P1.03**, **P1.13**, **P1.21**, **P1.23**, P1**.24** and **P1.26** all mentioned the *Work in Progress* metric. This metric denotes the amount of components that is being worked on concurrently at a particular time. Additionally, **P1.21** and **P1.23** also mention the *Average Work in Progress*, while **P1.23** and **P1.26** mention the *Maximum Work in Progress*. The idea behind this metric is that superfluous context switching is harmful to productivity, and should be kept to a feasible minimum.

#### *3.4.2.7 Common Tempo Time or Task Time*

**P1.24** and **P2.08** mention the *Common Tempo Time* metric. This metric is fairly similar to the *Cycle Time* metric, and is defined as the net working hours available divided by the number of work items required. In traditional industrial manufacturing, it provides an estimated cycle time needed in order to accomplish the required work items, given the available working hours. E.g. if a particular factory needs to produce 50.000 paperclips, and they can produce 25.000 paperclips per 8 hour day, the *Common Tempo Time* would be 50.000 / (16 * 60 * 60), and the factory would thus need a *Cycle Time* of 0.87 seconds per paperclip. Applied to software development, this gives the team an indication of how much time, on average, they have left per work item (or normalized on story points) that needs to be delivered. **P1.21** refers to this metric as *Task Time*.

#### *3.4.2.8 First Time Yield*

Study **P1.23** mentions the *First Time Yield* metric. This metric is defined as the percentage of units produced without a defect, and can be calculated by the formula denoted below *in equation 5.* In classical industrial manufacturing the production of a defective unit can yield quite devastating effects in the sense that an entire product might need to be discarded. In software development, however, defective units can often be repaired with a patch or update. However, the financial consequences of

a defective unit might be far larger in a software development context. The *First Time Yield* metric can thus be a relevant indicator in both contexts.

$$First\ Time\ Yield = \frac{Units\ Produced - Defective\ Units\ Produced}{Units\ Produced}$$

*Equation 5 - First Time Yield*

#### *3.4.2.9 Flow Efficiency*

Study **P1.24** mentions the concept of *Flow Efficiency*. This Lean metric measures the percentage of time spent adding value to a particular component, and can be calculated by the formula denoted below in *equation 6*. In the domain of software development, this is more commonly known as a Kanban metric, but its roots lie in Lean Manufacturing. This measure directly translates Lean's focus on eliminating waste into an easy to grasp and easy to optimize metric for efficiency.

$$Flow\ Efficiency = \frac{Value\ Added\ Time}{Lead\ Time}$$

*Equation 6 - Flow Efficiency*

#### *3.4.2.10 Throughput*

Study **P1.24** also mentions the *Throughput* metric, which is defined as *the number of units processed by a given phase or activity, per unit of time*. While traditionally more of an industrial manufacturing metric, *Throughput* can be thought of in terms of *Story Point Velocity*, or even a simple count of work-items delivered in the context of software development.

#### *3.4.2.11 Summary*

The majority of *Lean Manufacturing* metrics are generic enough to be applicable to any phased process, and are thus implemented almost exactly as-is in the *Lean Software Development* movement. The optimum values, however, may lay at wildly different numbers for each of these practices. In software development, for example, where faulty components can be easily fixed, the consequences of producing a faulty component are much less severe than in industrial manufacturing, where the entire component might need to be discarded. A metric such as *First Time Yield* will thus have to be much closer to 100% in industrial manufacturing than in software development. Similarly, the adverse

effects of context switching might be less severe for industrial manufacturers than for software developers, resulting in a different optimal value for *Work in Progress*.

### 3.4.3 Function Points

Another oft-recurring metric is the *Function Point* metric, which was introduced by Albrecht in 1979 as *a unit of measurement to express the amount of business functionality an information system provides to a user*. In the process of function point size estimation, requirements are categorized into *inputs*, *outputs*, *inquiries*, *internal files* and *external interfaces*. For each of these classified requirements, an estimation of their complexity is made using function points. This approach leads to requirements that are very user-oriented, and allows for an easy mapping of functions to end-user-functionality. Function point analysis requires individuals with loads of expertise to assess the requirements, which can cause problems in communication with other stakeholders. Additionally, the measure is not truly objective, even after thorough assessment by experts. The biggest drawback of function point analysis, however, is the fact that it tends to overlook internal functions, such as algorithms, which also require resources to implement. Over the years, many different forms of function point analysis have surfaced, each attempting to fix the perceived weaknesses of the original proposal, which is also known as *IFPUG*, and is specified in the ISO 20926 standard. There are several officially recognized standards and specifications for these other functional size measurement methods. These are *COSMIC* (ISO 19761), *FiSMA* (ISO 29881), *Mark-II* (ISO 20968), and Nesma (ISO 24570). Other forms of function point analysis, not officially encapsulated in ISO standards, include ESTIMACS, AFP and SPQR/20. Function point analysis, in any of these forms, were mentioned in **P1.01**, **P1.02**, **P1.07**, **P1.10**, **P1.17**, **P1.24**, **P2.01**, **P2.02**, **P2.03**, **P2.11**, and **P2.18**.

### 3.4.4 Code Analysis Metrics

The following section introduces the *code analysis* metrics that were found. It introduces metrics that range from targeting quality and complexity, to size and efficiency. The big benefit of code analysis metrics is the fact that their calculation can be easily integrated in automated build, test or deployment tools. Their biggest downside, however, is the fact that they are often not very simple to understand or interpret. Additionally, they are often very prone to be used to infer about aspects of the software development process for which they were not designed. *Lines of Code*, for example, can

be an excellent measure of complexity or quality in the context of the average size of a method. However, it might seem very intuitive to start measuring efficiency in terms of *Lines of Code per Unit of Time*. In the last few decades, however, the latter form has been widely critiqued by the industry as *invalid*, *inaccurate* and *misinformed*.

#### ***3.4.4.1 Lines of Code***

Unsurprisingly, a lot of work mentions *lines of code* as a form of post-process size estimation or a measure of efficiency when related to other metrics such as time. The line of code metric looks at either the absolute number of lines of code written, or at executable lines of code, in order to determine the size of an application, or the efficiency per unit of time of team members. The metric is part of a larger family of metrics, which we'll call *code analysis metrics*, that attempt to provide size, complexity, quality and efficiency estimates through similar code analysis techniques. *Lines of code* are mentioned in **P1.07**, **P1.09**, **P1.10**, **P1.11**, **P2.04**, **P2.05**, **P2.13**, **P2.15**, and **P2.17**. An advantage of this approach is that it can easily be automated, and will deliver the same, objective answer every time. However, the metric has been widely critiqued by the industry as misinformed, inaccurate and invalid, and have lost most of its popularity over the past one or two decades. It's most prevalent downside is the fact that its benchmarks are not stack-, framework- or even language-agnostic. Additionally, the metric can only be used for size estimations after the work has already been done, and refactoring can even cause a negative production or efficiency value. The following famous, slightly paraphrased *Antoine de Saint-Exupery* quote is often used to expose the inherent flaw in determining efficiency at the hands of lines of code or other code analysis metrics:

> "*Perfection is achieved, not when there is nothing more to add,*
> *but when there is nothing left to take away*."
>
> \- *Antoine de Saint-Exupery*

#### ***3.4.4.2 Code Generation***

**P2.09** mentions using the *Percentage of Modified Generated Lines of Code* in order to measure the complexity of a project. This metric relates the *Amount of Generated Lines of Code* metric to the *Amount of Manually Modified Generated Lines of*, and indicates a perceived project complexity. Minimizing this metric limits the amount of manual maintenance required on generated code. They thus also make an explicit distinction between *Amount of Lines of Code* and *Amount of Manually Created Lines of Code*, which no other study has done.

Similarly, **P2.09** introduces the *Percentage of Modified Generated Files* metric, which takes the metric one level of granularity higher, relating the *Amount of Generated Files* to the *Amount of Modified Generated Files*.

#### *3.4.4.3 Component Counts*

Many studies mention component counts in source code. The following counts are mentioned in the following studies: *Number of Classes* (**P2.17**), *Number of Generated Files* (**P2.09**), *Number of Manually Created* Files (**P2.09**), *Number of Inherited Methods per Class* (**P2.06**, **P2.07**), *Number of Interfaces* (**P2.17**), *Number of Methods Added per Class* (**P2.07**), *Number of Modified Generated Files* (**P2.09**), *Number of Overridden Methods per Class* (**P2.06**, **P2.07**, **P2.17**, **P2.18**), *Number of Packages* (**P2.17**), *Lines of Code per Method* (**P2.01**, **P2.06**, **P2.15**, **P2.16**), *Number of Static Methods per Class* (**P2.17**), *Parameters per Method* (**P2.03**, **P2.05**, **P2.17**) and *Number of Static Variables per Class* (**P.2.17**). These component counts can easily be automated in build, test and deployment pipelines, and allow its user to, for example, place limits on certain components that have been shown to limit complexity and increase quality, such as the amount of lines of code per method.

#### *3.4.4.4 Churn*

Study **P1.26** introduces the metric of *Churn*, which is defined as *the number of lines of code added, deleted or modified*. This metric is closely related to *Lines of Code*, and exhibits the same drawbacks. While it might thus be unsuitable for measuring efficiency, it can yield interesting results in terms of how often particular components are touched, and how significant those changes are.

#### *3.4.4.5 Access to Foreign Data*

Study **P2.15** mentions the code quality metric of *Access to Foreign Data*, which is defined as *the number of external classes from which a given class accesses attributes, directly or via accessor methods*. This can be used as a measure of coupling, and is indicative of code quality and complexity.

#### *3.4.4.6 Foreign Data Providers*

Study **P2.15** also mentions the code quality metric of *Foreign Data Providers*. While the paper fails to clearly define the metric, it presumably measures the inverse of the coupling relationship of *Access to*

*Foreign Data*. Here, presumably, the coupling relationship of *the number of external class that access the class's attributes, directly or via accessor methods* is measured.

#### *3.4.4.7 Efferent and Afferent Coupling*

Studies **P2.13** and **P2.17** define *Efferent Coupling* as *a measure of the total number of external classes coupled with classes of a package, as a result of outgoing coupling***.** It can be more simply stated that *Efferent Coupling* measures the number of classes in other packages that the classes in the current package depend upon. Similarly, these studies mention *Afferent Coupling*, defined as *a measure of the total number of external classes coupled to classes of a package, as a result of incoming coupling*. Again, simplified, this metric measures the number of classes in other packages that depend upon classes within the current package.

#### *3.4.4.8 Attribute Import Coupling*

**P2.07** then mentions the *Attribute Import Coupling*, but subsequently fails to properly introduce its definition. Presumably, it measures the same kind of coupling as *Afferent Coupling*.

#### *3.4.4.9 Coupling Between Objects*

Studies **P2.01**, **P2.05**, **P2.07**, **P2.12** and **P2.18** all mention the first widely known *Chidamber and Kemerer* code quality metric of *Coupling Between Objects*, which is defined as *the number of other classes whose methods, field or properties are used*. This metric appears to measure the same kind of coupling (outgoing coupling) as the *Efferent Coupling* metric from **P2.13** and **P2.17**, and the *Access to Foreign Data* metric from **P2.15**.

#### *3.4.4.10 Response for a Class*

Studies **P1.11**, **P2.05**, **P2.13** and **P2.17** mention a second widely known *Chidamber and Kemerer* code quality metric called *Response for a Class* (RFC). It is defined as the count of (public) methods in a class and method directly called by these, and is used as a measure of complexity and coupling.

#### ***3.4.4.11 Weighted Methods per Class***

Studies **P1.02**, **P1.11**, **P2.02**, **P2.05**, **P2.06**, **P2.13**, **P2.15**, **P2.17** and **P2.18** all mention the third widely known *Chidamber and Kemerer* code quality metric of *Weighted Methods per Class*. It is defined as *the sum of all cyclomatic complexities of all of the methods of a particular class*, and can be used to indicate how much effort is required to develop and maintain a particular class. The *Cyclomatic Complexity* metric is introduced in *section 3.4.4.32*.

#### ***3.4.4.12 Depth of Inheritance Tree***

The studies **P1.22**, **P2.01**, **P2.05**, **P2.06**, **P2.13**, **P2.17** and **P2.18** all mention the fourth *Chidamber and Kemerer* metric of *Depth of Inheritance Tree*. This metric measures the longest inheritance chain in a given program's source code. Study **P.2.07** also mentions averaging this metric for all inheritance chains in a given program's source code, but calls it *Average Class-to-Leaf Depth* instead.

#### ***3.4.4.13 Number of Children***

The studies **P2.01**, **P2.05**, **P2.06**, **P2.07**, **P2.13** and **P2.18** all mention the fifth *Chidamber and Kemerer* metric of *Number of Children*. This metric measures how many subclasses are going to inherit the method of a parent class. The value of this metric approximately represents the level of reuse in an application, and a higher value thus represents a higher level of reuse. It also states that, as the value increases, the amount of tests are also likely to increase, because more children indicate more responsibility.

#### ***3.4.4.14 Lack of Cohesion of Methods***

Many studies mention the final *Chidamber and* Kemerer metric called *Lack of Cohesion of Methods (LOCM).* The metric was mentioned in studies **P1.11**, **P1.15**, **P1.17**, **P1.18**, **P1.22**, **P2.05**, **P2.13**, **P2.15**, **P2.17** and **P2.18**, and is widely adopted in automated tooling such as linters and CI/CD pipeline facilitators. It measures the number of connected components in a class, where a connected component is a set of related methods and class-level variables. This measurement is done by looking at how interconnected these components are in terms of how many of the other components each particular component references directly. If more than one separate clusters of connected

components exist, this is often a strong indication that the object does not adhere to the *Single Responsibility* principle of Robert (Bob) C. Martin's SOLID principles.

#### *3.4.4.15 Coupling Concentration Index*

**P1.22** mentions the *Coupling Concentration Index*. The paper states that the metric is a measure of inequality, and is defined as twice the area between the concentration curve and the equality line, resulting from relating *Coupling Between Objects* to module size.

#### *3.4.4.16 Coupling Factor*

Study **P2.18** introduces the *Coupling Factor* metric. The study fails to clearly introduce the metric, instead providing just a complicated formula for its calculation.  It appears to measure *Total Coupling* in terms of *Dynamic Coupling* and *Static Coupling*, and ignored coupling that is caused by inheritance relationships. Furthermore, the study states that *Coupling Factor* is 0% if no classes are coupled, and 100% if all classes are coupled.

#### *3.4.4.17 Message Passing Coupling*

Study **P2.18** introduces the concept of *Message Passing Coupling*, defined as *the number of send statements defined in a class*. While it does not explicitly defined what is considered *send statement*, the context of the introduction suggests that the access of a method or attribute of a different class, indicates a single send statement. This would make it nearly identical to other incoming coupling metrics discussed in this section.

#### *3.4.4.18 Information Based Coupling*

Study **P2.07** introduces the concept of *Information-Based Coupling*, but subsequently fails to introduce the metric, or what it is supposed to measure.

#### *3.4.4.19 Data Abstraction Coupling*

The *Data Abstraction Coupling* metric is mentioned by **P2.07** and **P2.18.** It measures the coupling complexity caused by Abstract Data Types (ADTs), and is thus a coupling metric that is limited to type references, as opposed to object references.

#### *3.4.4.20 (Descendant) Method-to-Method Export Coupling*

Study **P2.07** introduces the metrics of *Method-to-Method Export Coupling* and *Descendant Method-to-Method Export Coupling*. It subsequently fails to properly introduce or explain them.

#### *3.4.4.21 Specialization Index*

Study **P2.07** mentions the *Specialization Index*, which it defines as *the amount of refined instance methods in all classes, times the total amount of super classes, divided by all instance methods of all classes*. Here, a refined instance method is one that has been defined in a superclass, but adapted in a sub-class. It essentially measures how much functionality is refined by subclasses.

#### *3.4.4.22 Specialization Ratio*

In turn, study **P2.18** mentions the *Specialization Ratio*, which it defines as the *number of subclasses divided by the number of super classes*. Where *Specialization Index* measures refinement at the level of instance methods, *Specialization Ratio* does so at the level of classes.

#### *3.4.4.23 (Normalized) Code Smell Occurrences*

Study **P2.09** mentions the metric of *Amount of Code Smell Occurrences*. In their work, they aim to automatically detect these code smells based on other code analysis metrics. Similarly, they introduce the *Normalized Amount of Code Smell Occurrences* metric, where the former is normalized over the amount of files *or* the amount of lines of code.

#### *3.4.4.24 Cycles in Dependency Graph*

Study **P2.03** mentions the *Amount of Cycles in the Dependency Graph* as a measure of complexity and quality for a software project. They posit that more cycles in the dependency graph increase complexity, and are thus a sign of lower code-base-quality.

#### *3.4.4.25 Method and Attribute Hiding Factor*

Study **P2.18** mentions the metric *Method Hiding Factor*. This metric is defined as a percentage of the methods in a particular program that cannot be called by other classes. This metric is 0% if, for instance, all metrics are declared public, and 100% if all method are declared private. Similarly, *Attribute Hiding Factor* measures the same concept for attributes. The study, however, is not clear on whether this metric should include static methods or attributes, or exclude them from consideration. Similarly, it is unclear if protected access modifiers are considered as hidden or not.

#### *3.4.4.26 Method and Attribute Inheritance Factor*

Similarly, study **P2.18** defined the *Attribute Inheritance Factor* as *the sum of the number of attributes declared in all classes, divided by the sum of the number of attributes inherited in all classes*. It denotes a percentage of attributes that are declared by the class itself, as opposed to inherited from a base class. *Method Inheritance Factor* measures the same concept for methods, as opposed to attributes.

#### *3.4.4.27 (Code) Abstractness*

Studies **P2.13** and **P2.17** introduce the concept of *Abstractness*. It defines the metric as *a comparison of the number of abstract classes and interfaces, to the total number of classes in the evaluated package.* The metric thus has a range of 0% to 100%, where the former signifies an absolute concrete package, and the latter an absolute abstract package.

#### *3.4.3.28 (Code) Instability*

Similarly, studies **P2.13** and **P2.17** introduce the concept of *Instability*, which they define as *the ratio of Efferent Coupling to Total Coupling*. Here, they define *Total Coupling* as the sum of *Efferent Coupling* and *Afferent Coupling*. Again, this metric has a domain of 0% to 100%, where the former signifies an

absolutely stable package, and the latter signifies an absolutely instable package. Here, stability refers to how much impact changing a particular component would have.

#### *3.4.4.29 Normalized Distance from Main Sequence*

Studies **P2.13** and **P2.17** also introduce the *Normalized Distance from Main Sequence* metric. The metric is defined as *the perpendicular distance of a package from the idealized line of Abstractness + Instability = 1*. The authors state that a package is ideally absolutely abstract and stable, or absolutely concrete and unstable. If the package is somewhere in between, the sum of both should ideally equal 1. Thus the normalized distance from the main sequence should ideally be 0.

#### *3.4.4.30 Comment Percentage*

Studies **P2.05** and **P2.13** mention the metric of *Comment Percentage*, which is defined as the percentage of *Lines of Code* that are comments. The former posits that they have found a perceived optimum at 30%, while the latter does not make any such claim. It does, however, mention that a code-base that is adequately named and built, should require less comments than one that does not exhibit those qualities, indicating that a lower percentage of comment can, in some cases, be better than a higher percentage.

#### *3.4.4.31 Percentage of Dead Code*

Study **P2.02** mentions the *Percentage of Dead Code* as a metric for evaluating the quality of a code base. Similar to most other metrics that this study introduces, the metric is not properly defined, and no further reference to it has been made in the remainder of the text. Dead code, however, is presumed to mean code that is no longer in use, in the sense that no execution path through the application's expression tree can ever reach the declared code. Technically, *code reflection* will sometimes be able to reach such code at run-time, and is thus often not included in such definitions. Examples could be code that is written after a return statement, or written inside an if statement that will always evaluate to false, but it might also include code in classes or methods that are simply not used anymore and thus not referenced anywhere else in the code-base.

##### *3.4.4.32 Cyclomatic Complexity*

The *Cyclomatic Complexity* metric is mentioned by **P1.03**, **P1.14**, **P1.15**, **P1.16**, **P1.17**, **P1.18, P2.02**, **P2.05**, **P2.15**, and **P2.17**. The metric measures the *number of linearly independent paths within a section of source code*, and can be easily computed using a program's *Control Flow Graph*. A lower value is regarded as less complex and thus easier to grasp.

##### *3.4.4.33 Halstead Complexity Measures*

Studies **P1.03, P1.14** and **P2.15** mention the *Halstead Complexity Measures*, which is a set of software metrics introduced by Maurice Howard Halstead in 1977 (Halstead, 1977). The set includes measures for program vocabulary, length, volume, difficulty, effort and estimations of required time and expected defects.

##### *3.4.4.34 Duplicate Expressions*

Studies **P1.17**, **P1.18** and **P2.02** mention the *Duplicate Expressions* metric. This metric calculates the percentage of a code base that contains duplicated code. None of them, however, explain how this is measured, or at what granularity. When asked, one of the experts introduced in chapter 4 indicated that code duplication is often measured at the 'two lines-of-code' granularity.

##### *3.4.4.35 Index of Inter-Package Extending*

Study **P2.16** introduces the *Index of Inter-Package Extending*. This metric is defined as *the ratio of the number of 'extend' dependencies between classes within a local package, against the total number of 'extend' dependencies between all classes of the software system*. It explicitly states that the '*extend'* dependency can both be an inheritance relationship between two classes, or an implementation relationship between a class and an interface.

##### *3.4.4.36 Index of Inter-Package Extending Diversion*

Study **P2.16** also introduces the *Index of Inter-Package Extending Diversion*. It continues to define the metric as *the average extent of how diverse the classes extended by a specific package, distribute in different packages*.

#### *3.4.4.37 Index of Inter-Package Usage*

Study **P2.16** introduces the *Index of Inter-Package Usage*. The metric is defined as *the ratio of the number of 'use' dependencies between classes within a local package, against the total number of 'use' dependencies between all classes of the software system*.

#### *3.4.4.38 Index of Inter-Package Usage Diversion*

Similarly, study **P2.16** introduces the *Index of Inter-Package Usage Diversion*, and continues to define the metric as *the average extent of how diverse the classes used by a specific package distribute in different packages*.

#### *3.4.4.39 Index of Package Changing*

Study **P2.16** also introduces the *Index of Package Changing.* The study defines the metric as *the percentage of the number of the non-dependency package pairs against the total number of all possible package pairs.* It claims that this metric measures the strength of the independency of packages.

#### *3.4.4.40 Index of Package Goal Focus*

Study **P2.16** also introduces the *Index of Package Goal Focus*, which is defined as the *average extent of the overlap between the different service sets provided by the same component to other different components in a software system*. The study claims that the metric indicates the average extent to which the services of a specific package serve the same goal.

#### *3.4.4.41 Information Based Cohesion*

Study **P2.18** introduces the concept of *Information-Based Cohesion*. This metric is defined as *the number of invocations of other methods of the same class, weighted by the number of parameters of the invoked method*.

#### *3.4.4.42 Locality of Attribute Access*

Study **P2.15** mentions *Locality of Attribute Access* as a tool for identifying classes or methods that use too many attributes from classes other than their own. The study subsequently fails to introduce how this metric is measured.

#### *3.4.4.43 Maximum Nested Block Depth*

Study **P2.17** mentions the *Maximum Nested Block Depth*, defined as the *maximum depth of nested blocks of code*. It posits that more nested blocks lead to worse readability and more complex solutions, and thus advocate for a low maximum and average nested block depth.

#### *3.4.4.44 Release Deltas*

Study **P1.09** introduces *New Classes per Release, New Methods per Release* and *New Lines of Code per Release* as a means of gauging the size of a software project over time. Similarly, the study introduces the *New Features per Release* as a means of gauging the development speed of a particular project over time.

#### *3.4.4.45 Polymorphism Factor*

**P2.18** mentions the *Polymorphism Factor* metric. The study states that the metric measures *the degree of method overriding in the class inheritance tree*, and formally defines it as *the sum of all newly introduced methods in all classes, divided by the sum of all overridden methods in all classes*, times something they call the *descendant count*, which is presumably the *depth of the inheritance tree* or *class-to-leaf distance* for that particular class.

#### *3.4.4.46 Reuse Ratio*

Study **P2.18** also mentions the *Reuse Ratio* metric, defined as *the number of super classes divided by the total number of classes*. The *Reuse Ratio* indicates how much reuse is apparent in the code-base specifically due to inheritance relationships.

#### *3.4.4.47 Summary*

The class of *Code Analysis Metrics* has by far yielded the most software development metrics. Their applications range from quality and complexity, to size and efficiency. It is interesting to note that, while you would expect most of these metrics to from the latter part of the previous century, new code analysis metrics are still being devised today. Additionally, it appears that concepts such as *technical debt* and *code smells* are universally accepted as having a detrimental effect on the health of a code base, but remain somewhat elusive for automated code analysis tools to detect, leading to more and more complex and complicated concepts of *cohesion* and *coupling*.

### 3.4.5 Complex Mathematical Metrics

The *code analysis metrics* mentioned in the previous section, are often accompanied by studies that employ *complex mathematical models* to analyse them even further. Some examples include regression models, the COCOMO/COCOMO2 method, and SLIM. For the purpose of this study, we have decided to omit them from the review.

### 3.4.6 Testing Metrics

The next section introduces metrics for testing a product or code-base. Just as with *code analysis metrics*, most of the discovered metrics can be easily automated and incorporated in deployment pipelines.

#### *3.4.6.1 Unit Test Coverage*

Studies **P1.03**, **P1.07, P1.16, P1.24** and **P2.02** mention the metric of *Unit Test Coverage*. This metric was used by 88% of teams that were examined in **P1.03**. Nowadays, unit test coverage is a metric that is embedded in many Continuous Integration/Delivery (CI/CD) pipelines, Command Line Interface (CLI) tools and Integrated Development Environments (IDE). The metric is a measure that is used to describe the degree to which the source code of a program is executed when the test suite runs. The underlying idea is that a program that has more of its code executed by its test suite, should have a lower chance of containing undetected bugs.

#### *3.4.6.2 Amount of Tests*

Additionally, studies **P1.06**, **P1.16, P1.19,** and **P1.24** mention the *Amount of Tests* as a valid metric for assessing code quality, while also stressing the importance of automation when it comes to executing them.

#### *3.4.6.3 Review Rates*

Study **P2.04** mentions using the *Average Code Review Rate* and *Average Design Review Rate* to denote the average amount of time between code or design reviews, as a means of indicating how well these review practices are embodied in their software development processes.

#### *3.4.6.4 Test Suite Rates*

Study **P1.16** mentions the *Regression Test Cycle Time* and the *Smoke Test Cycle Time* as a means to gauge how often regression and smoke test suites are ran, and how much overhead there is in doing so.

#### *3.4.6.5 Running Tested Features*

Study **P1.24** and **P2.02** mention the *Running Tested Features* metric. While neither explicitly defines the metric, it is presumed that it measures the amount of features that currently pass their automated test suite.

#### *3.4.6.6 Test Failure Rate*

The *Test Failure Rate* metric is introduced in **P1.24**, but subsequently not explained or referenced. Presumably, the metric measures what percentage of a test suite fails their criteria.

#### *3.4.6.7 Test Pass Rate*

The *Test Pass Rate* metric is introduced in **P1.24**, but subsequently not explained or referenced. Presumably, the metric measures what percentage of a test suite passes their criteria.

#### *3.4.6.8 Test Growth Ratio*

Similarly, the *Test Growth Ratio* metric is introduced in **P1.24**. The metric relates the increase of lines of code of the test suite(s) to the increase in lines of code of the code base under test. In some investigations, it is found that the test growth is disproportionate from the source-code growth, which may in turn require practices to remove unit tests that have not failed for an extended amount of time, as proposed by James O. Coplien. This remains a highly debated view, however, to this day.

#### *3.4.6.9 Test Runtime*

Study **P2.02** introduces the *Test Runtime* metric as a means of gauging its perceived overhead on the code-build-test cycle. If the runtime of a test suite becomes too large, developers might be tempted not to run them after every cycle, or be severely encumbered by the overhead.

#### *3.4.6.10 Tests per Story*

Study **P2.02** also introduces the *Tests per Story* metric, but fails to define or reference it, simply introducing it as *a well-known agile metric*. It can be presumed that it's a simple division of the number of test cases written by the number of user stories currently implemented.

#### *3.4.6.11 System Analysis Cost*

Study **P2.10** introduces four cost aspects to testing. The first of these is the *System Analysis Cost*. It states that, before a test can be selected, the test analyst must become familiar with the system specification, design and possibly the program. Time must be spent studying the various requirements and design documents. The costs associated with this process, are introduced as the *System Analysis Costs* of a tester.

#### *3.4.6.12 Test Selection Cost*

**P2.10** then introduces the *Test Selection Cost* metric, stating that*, after gaining some knowledge about the system, the test analyst can finally select the test cases for testing the actual behaviour of the system.* The costs uncured include working out the test input, and identifying the correct output or system behaviour.

#### *3.4.6.13 Test Execution Costs*

Similarly, **P2.10** states that additional testing costs are incurred by having the tester setting up the environment for testing (such as loading and computing required modules, and entering the proper data tables), and the actual time spent executing the tests. It continues to state that this cost can be quite high for some applications, and brings forward the example of the telecommunication industry, in which the costs of setting up a testing lab to simulate an actual communications network, can be as high as several million dollars. It calls this metric the *Test Execution Cost* metric.

#### *3.4.6.14 Test Result Analysis Costs*

Finally, **P2.10** introduces the *Test Result Analysis Cost* as *the last cost associated with the testing process*. It identifies the tester’s time spent collecting the test outputs, comparing those outputs to the system specifications, and computing resources required for recording the system behaviour under test.

#### *3.4.6.15 Summary*

Most of the testing metrics that were discovered, focus on the frequency at which tests are performed, how much of the product is being tested, or the costs of performing those tests. They also make surprisingly little distinction between unit-, regression-, integration- or smoke-tests.

### 3.4.7 Team Composition Metrics

#### *3.4.7.1 Amount of Team Members*

Several studies mention team composition metrics, such as the *Amount of Team Members* in a given team, or the *Maximum* or *Average Amount of Team Members* in a given team, in a particular time period. These kinds of metrics are mentioned in **P1.01**, **P1.02**, **P1.12**, **P1.20** and **P1.25**.

#### *3.4.7.2 Average Projects per Employee*

Additionally, **P2.12** mentions the *Average Projects per Employee* metric, which measures how many projects, on average, an individual employee is working on concurrently. The idea behind this metric is that excessive context switching has a detrimental effect on productivity, and should thus be kept to a feasible minimum.

#### *3.4.7.3 Scrum Teams per Project*

Study **P1.20** mentions the *Scrum Teams per Project* metric, which it uses in its subsequent calculations of *Technical Efficiency.*

#### *3.4.7.4 Personnel Turnover*

Finally, **P2.12** mentions the *Personnel Turnover* metric, as a means of gauging the stability of a team or the entire workforce of a company. It measures how many team members leave or join an organization or team during a particular time-span.

### 3.4.8 Build Metrics

#### *3.4.8.1 Build Runtime*

Study **P2.02** mentions the *Build Runtime* metric (introduced here as *time taken per build)*. Sadly, the study does not properly introduce the metric, nor quotes an appropriate reference, simply introducing it as a well-known agile metric. Presumably, this metric can be used to give some sort of an indication as to whether or not the iterative code-build-test cycle is sufficiently quick. Additionally, the study also mentions the *Builds per Day* metric, but similarly fails to properly introduce, reference or describe the metric.

#### *3.4.8.2 Percentage of Successful Builds*

Study **P2.02** also introduces the *Percentage of Successful Builds.* Just as with *Build Runtime*, the study makes no further reference to the metric, simply introducing them all as *well-known agile metrics*. We

can thus not make any deductions about why the metric is important, and whether a high or low value is preferable and why that might me the case.

### 3.4.9 Time Based Metrics

While Scrum explicitly attempts to stay away from time based metrics, there are still plenty such metrics to be found in literature. Note that, while they were initially introduced in section 3.4.2, most *Lean Manufacturing* metrics, such as *Lead Time, Processing Time, Value Added Time* and *Queue Time*, may also be considered *time based metrics*. Where most Scrum metrics are meant to estimate future capabilities, most time-based metrics can often be used to retroactively assess performance.

#### *3.4.9.1 Hours*

These range from simple estimations in terms of hours or days, to somewhat more complex estimation techniques. Studies **P1.02**, **P1.05**, **P1.07**, **P1.09**, **P1.11**, **P1.12,** and **P1.20** mention using *Person Hours* as a means to estimate work effort, while **P1.07, P1.12,** and **P1.16** mention *Person Months* instead. Additionally, **P1.11** mentions measuring the average time in seconds spent coding a specific method. Furthermore, **P1.24** mentions the *Actual Development Time*, defined as the time which was spent actually developing a story or product. As an extension, studies **P1.01** and **P1.10** mention the metric of *Hours per Function Point*, while study **P1.07** mentions the metric of *Hours per Story Point* instead.

#### *3.4.9.2 Ideal Days*

Alternatively, **P1.06** and **P2.02** mention working with *Ideal Days*, which is defined as *a unit for estimating the size of product backlog items based on how long an item would take to complete if it were the only work being performed, there were no interruptions, and all resources necessary to complete the work were immediately available*. Ideal days are often easier to grasp than function points or story points, for team members as well as managers and outside stakeholders. This can lead to a higher standard of communication amongst stakeholders, which may in turn translate into a higher success rate. Additionally, the metric requires less training and expertise than function- and story points, because of its intuitiveness. This may lead to more confident estimates than would be achieved by means of other point-based estimation techniques. The metric, however, also has significant downsides. For example, the estimation technique does not really allow for team members to collaborate, which fosters a sense of individuality instead of teamwork. Additionally, a task that

might take five ideal days at the start of the project, might only take three later on down the line, once the team members are more comfortable with the project. This results in the estimate becoming less accurate over time. Outside stakeholders might also have a hard time grasping why a project that is estimated as 50 ideal days will take the team closer to 100 real days to complete.

#### ***3.4.9.3 Load Factor***

Study **P1.24** refers to the *Load Factor* metric, which denotes the amount of *real days in an engineering day*. This metric appears to be closely related and inverse to *Ideal Days, w*here *Ideal Days* denotes *how many engineering days are in a real day*.

#### ***3.4.9.4 Work Capacity***

The metric of *Work Capacity* denotes the total amount of hours that an entire team can spend on a given sprint or iteration. This metric was mentioned in studies **P1.03** and **P1.04** and is sometimes used to estimate future capacity, and sometimes used to indicate factual historical capacity. Study **P1.20** also mentions this metric, but calls it *Ideal Capacity* instead.

#### ***3.4.9.5 Overtime***

Study **P2.12** mentions the *Average Overtime per Day* and *Average Overtime per Sprint* metrics as an indication of healthy company culture.

#### ***3.4.9.6 Time to Market (per Function Point)***

Study **P1.25** mentions the *Time to Market in Days per Function Point* as a means of measuring how long it takes a particular team to deliver a single *Function Point* to the production environment(s).

### 3.4.10 Defect Metrics

#### ***3.4.10.1 Mean Time to Recovery***

Studies **P1.03**, **P1.08**, **P1.24**, **P2.02**, **P2.04** and **P2.12** mention a *Mean Time to Recovery* metric. This metric determines the average time it takes for a backlog item of the type *bug* or *defect* to be closed.

For most use-cases, the time starts ticking as soon as the item enters the backlog, but some variants opt to start from the moment the bug is estimated, or even from the moment the bug enters a sprint. The latter variants allow the team to more easily game the metric, and a good average time thus fails to prove that additional value was provided to the end-user in a sufficiently quick fashion. Not all occurrences use the same term for this metric, where terms like *Bug Correction Time, Average Defect Correction Time* or *Defect Removal Efficiency* have also been used. Study **P1.09** also mentions using *Defects Fixed per Release* as a measure of gauging defect removal efficiency.

#### *3.4.10.2 Fault Latency*

Study **P2.14** introduces the metric of *Fault Latency*, which is defined as the difference in time between when a particular piece of code was written, and when a particular defect in that piece of code was identified. While in traditional waterfall development, the former may be easy to identify, in iterative development this may not be so easy.

#### *3.4.10.3 (Open) Defect Severity Index*

According to **P1.03**, the *Defect Severity Index* can be used to measure the quality of the delivered work. The underlying metric of *Defect Severity* denotes a measure of impact a particular defect has, expressed as an integral value. In this denotation, defects with a higher impact have a higher integral value assigned to them. The *Defect Severity Index*, then, aggregates the *Defect Severity* of an entire backlog. The most common approach to calculate the index appears to be to take the sum of all defect's severity, and divide it by the total amount of stories on the backlog. Finally, the *Open Defect Severity Index* only takes into account the defects that are still unresolved, as well as the total amount of stories on the backlog that are still unresolved. The metric can then be used to determine the quality of the product after each iteration, and tracked over time to see whether the team is improving. A downside of the *Defect Severity Index* is that it is extremely easy to game, as the defects can simply be estimated at a lower severity in order to "increase the team's performance".

#### *3.4.10.4 Average Fault Cost*

Study **P2.14** mentions measuring the *Average Fault Cost*. The cost of a fault or defect sums the cost of the resources spent fixing the bug, the cost of the damages caused by the bug, and the revenue missed by the effects of the bug.

#### *3.4.10.5 Defects Carried Over*

Study **P2.02** introduces the metric of *Average Amount of Defects Carried Over to Next Iteration*. This metric essentially measures the amount of defects that still exist at the end of each iteration. Optimizing this metric may for instance lead to a lower *Mean Time to Recovery* or a lower *Defect Severity Index*.

#### *3.4.10.6 Defect Slippage Rate*

Additionally, studies **P1.03**, **P1.24** and **P2.02** mention the *Defect Slippage Rate*. This metric indicates the amount of defects that are not caught in the development, test or acceptance processes, and are subsequently discovered in production environments. While again the study does not outline exactly how the metric is calculated, *Defect Slippage Rate* is usually defined as the ratio between the amount of bugs discovered in production and the total amount of bugs discovered in the product. This definition is outlined below in *equation 7* (Padmini et al. , 2018).

$$Defect\ Slippage\ Rate = \frac{Amount\ of\ bugs\ discovered\ in\ production}{Total\ amount\ of\ bugs\ discovered\ in\ the\ product}$$

*Equation 7 - Defect Slippage Rate*

This metric, as opposed to the *Defect Severity Index*, is less easy to game. At the same time, however, the metric is less telling about the impact of its outcome. A single, million dollar defect in production would still yield a low *Defect Slippage Rate*, but would be very undesirable none the less.

#### *3.4.10.7 Fault Slip Through*

Study **P1.03** puts forward the concept of *Fault Slip-Through*. The studies define this metric as *a measure which determines the amount of faults that would have been more cost-effective to find and fix in an earlier phase*. The latter goes on to state that the metric simply calculates the amount of

defects or faults that were identified outside of the phase in which they should have been detected. Study **P2.14** actually refers to this metric as *Improvement Potential*.

#### *3.4.10.8 Number of Bounce Backs*

Study **P1.24** introduces the *Number of Bounce Backs* as the amount of defects that should not have occurred anymore if a root cause would have been fixed earlier. This makes it extremely similar to metrics such as *Defect Slippage Rate* and *Fault Slip Through*.

#### *3.4.10.9 Defect Density*

Additionally, studies **P1.03**, **P2.04** and **P2.12** also mention the *Defect Density* metric. This metric relates the amount of defects to specific constraints. For example, *Defect Density* in terms of a time-based constraint could be *Average* D*efects per Sprint* or A*verage Defects per Day.* Additionally, a *Defect Density* constrained on throughput could be the *Average Defects per User Story* or A*verage Defects per Story Point*. The metric, in any shape or form, was used by 52% of the teams investigated in **P1.03**. Those who did not use it marked that they did not care for the value, as long as all defects were closed by the end of the sprint. The metric can be used as a rudimentary measure of competence, as one would assume less defects would be introduced per constraint by a more competent software developer, but this assumption is daring at best. Studies **P1.03**, **P1.12**, **P1.22** and **P1.124** also mention using simple *Defect Counts* for assessment of quality.

### 3.4.11 Source Control Metrics

#### *3.4.11.1 Modified Components*

Study **P2.16** mentions the *Average Number of Modified Components per Commit*. While the name suggests what it measures, it is not clearly defined what is considered a *component* and what is not. Additionally, it is not clearly stated what the benefits of such a measure could be.

#### *3.4.11.2 Check-Ins per Day*

Additionally, **P1.24** mentions the *Check-Ins per Day*, which was defined as *the number of source control commits to the main trunk of the version control repository*. They state that *the metric can be used to manage risk, provide timely progress monitoring*, and *to communicate progress to upper-management*.

### 3.4.12 Finance Metrics

#### *3.4.12.1 Cost of Quality*

At the same time, **P1.03** mentions the *Cost of Quality* metric. This metric refers to the total cost of all materials and efforts required to make sure the resulting product conforms to the standard of quality that has been set by the organization. It is usually split up into several distinct parts. The first of these is the *Cost of Control*, which refers to the cost of preventing defects before they occur (prevention costs), as well as the cost associated with the inspection that occurred in order to find out whether defects exist at all (appraisal costs). The second constituent is the *Cost of Failure to Control*, which refers to the costs related to fixing defects after they have occurred, which is often split up once again into whether they have occurred internally (internal failure costs) or externally (external failure costs). The final formula for *Cost of Quality* is outlined below in *equation 8 through 10*.

$$Cost\ of\ Quality = Cost\ of\ Control + Cost\ of\ Failure\ to\ Control$$

*Equation 8 - Cost of Quality*

where

$$Cost\ of\ Control = Prevention\ Costs + Appraisal\ Costs$$

*Equation 9 - Cost of Control*

and

$$Cost\ of\ Failure\ to\ Control = Internal\ Failure\ Costs + External\ Failure\ Costs$$

*Equation 10 - Cost of Failure to Control*

You will notice, however, that these equations do not dictate a unit of measurement for its definition of *costs*. While it may seem most obvious to take *money* as a primary unit of measurement for costs, some situations might instead call for measurements in terms of *time*, *defects*, or any combination of

these. The metric originates from physical manufacturing processes, but, according to **P1.03**, has been applied to software development processes as well. In total, they found that 69% of the teams they interviewed used to metric in one form or the other.

### *3.4.12.2 Cost Efficiency*

Study **P2.08** mentions the *Cost Efficiency* metric. This metric is defined as the units delivered (in terms of lines-of-code-added) divided by the amount of hours invested, and thus measures the cost of adding a single line of code to the code base of a particular product.

### *3.4.12.3 Net Present Value*

Studies **P1.05**, **P2.02** and **P2.11** mention the *Net Present Value* metric, which is defined as *the difference between the present value of cash inflows, and the present value of cash outflows, over a period of* time. The metric is used in capital budgeting and investment planning, in order to analyse the profitability of a projected investment or project.

### *3.4.12.4 Return on Investment*

Studies **P2.02** and **P2.11** mention the finance metric of *Return on Investment*. This metric is defined as a ratio between the net profit and cost of a project. While it is a strong finance metric, it does not necessarily say much about agility.

### *3.4.12.5 Internal Rate of Return*

Study **P2.11** mentions the finance metric of *Internal Rate of Return*. The metric is very similar to the *Net Present Value* metric, and both are used in the evaluation process for capital expenditure. Where *Net Present Value* denotes the cash surplus or loss for a project, *Internal Rate of Return* calculates the percentage rate of return at which those same cash flows will result in a *Net Present Value* of zero. While it is a strong finance metric, it does not inherently say anything about agility.

#### *3.4.12.6 Business Value Delivered*

Both **P1.24** and **P2.11** mention the *Business Value Delivered* metric. While the former fails to properly introduce the metric, the latter defines it as a metric that *measures the rate of return on investment*. It can be used to answer the questions of when a project or product will begin generating a return on investment, when it will break even, and what the projected earnings are. It states that the metric is expressed in *Net Cashflow per Iteration*.

#### *3.4.12.7 Cost Performance Index*

**P1.24** mentions the *Cost Performance Index*. This metric is not defined in the source study, but presumably measures the ratio of work performed for which there was a defined budget, to the cost of work performed for which there was not a defined budget. The study claims that it can be used to monitor for deviances in the progress of a project, and can provide early signs that something is going wrong.

#### *3.4.12.8 Cost per Size Unit*

Study **P1.08** directly relates the cost of a project or product to individual size units. It mentions the metrics of *Cost per Function Point* and *Cost per Story Point*.

#### *3.4.12.9 Cashflow per Iteration*

Study **P2.11** introduces the *Cashflow per Iteration* as a metric to gauge the profitability of a project over time.

#### *3.4.12.10 Revenue per Customer (per Feature)*

Study **P1.24** mentions the *Revenue per Customer*. This metric, unlike what the name suggests, estimates the (projected) revenue per customer <u>that a particular feature will generate</u>, and can be used to prioritize higher value features in a backlog.

### 3.4.13 Requirements Metrics

#### *3.4.13.1 Requirements Clarity Index*

While study **P1.03** provides no references to any source literature, it does mention the *Requirements Clarity Index*. At the same time, further investigation on academic search engines does not yield any source material either. This is telling of a metric that was introduced not in the academic community, but in the unofficial spheres of the industry. Even though the metric has no clear origin in literature, the study found that 54% of the teams they investigated used the metric. Due to a lack of an official academic definition, what follows is a makeshift definition, defined by combining various blog posts and presentations.

The *Requirements Clarity Index* is an indication of how well specific requirements are understood by the various team members. It can be used as a threshold to determine when work on a story backlog item can actually begin. While some advocates insist on withholding any work until all team members perfectly understand the requirements, this threshold can differ from team to team, depending on what level of uncertainty the team is comfortable with. The metric can be calculated by defining a scale depicting levels of acceptance for individual story backlog items. An arbitrary example of such a scale can be as follows:

1. *Denied*
2. *Need Further Elicitation*
3. *Accepted*

Each individual team member then inspects the requirements and determines where they are at. An average can then be taken to determine the final value for the *Requirements Clarity Index* metric, which can in turn be compared to the threshold to determine if the story backlog item is ready to be included in an upcoming sprint.

#### *3.4.13.2 Requirements Inventory Size*

Study **P1.24** advocates tracking the *Requirements Inventory Size* over time. It states that the metric can be used to identify large handovers of requirements that might cause overloading situations to employees. Additionally, it had been used to identify problems in the development process.

#### *3.4.13.3 Change Requests per Requirement*

Study **P1.24** introduces the concept of *Change Requests per Requirement*, which it states can be used as an indicator of overall customer satisfaction, or an overall tool for understanding and improving quality.

#### *3.4.13.4 Critical Defects Sent by Customers*

**P1.24** introduces the *Critical Defects Sent by Customers* metric. The study simply states that these defects were *tracked and fixed in order to prevent losing customers*, but fails to mention why the count is important or what the metric can be used for.

#### *3.4.13.5 Number of Requests from Customers*

Study **P1.24** mentions the *Number of Requests from Customers*. However, the study does not explicitly define the metric, nor does it state how to use it or what its advantages could be.

#### *3.4.13.6 Implemented Versus Wasted Requirements*

Study **P1.24** posits that not all requirements are always completely implemented, but sometimes some effort may still have been put in to them, for instance in the form of technical specification or prototyping. The study mentions the ratio metric of *Implemented versus Wasted Requirements* as a means of gauging the amount of wasted requirements work.

#### *3.4.13.7 Requirements per Feature/Work-Item*

Study **P1.24** also mentions the *Requirements per Feature/Work-Item* metric, referred to collectively as the *Requirements per Phase*. The study posits that the metric can be used to reveal peaks in the workload, but fails to state how or why.

### 3.4.14 Other Metrics

The following section introduces metrics that were not wholly assignable to a separate classification of metrics. These metrics cover a wide variety of aspects of software development processes, from quality and size, to efficiency and customer satisfaction.

#### *3.4.14.1 Work Effectiveness*

**P2.12** introduces the concept of *Work Effectiveness*, defined as *a ratio between the work spent and the decrement of work remaining.* This definition closely relates it to the concept of *Effort Burndown* in Scrum, and can be used to determine whether or not a particular process is on track.

#### *3.4.14.2 Due Date Performance*

Study **P1.24** introduces the *Due Date Performance* metric. While it fails to properly introduce the concept, the metric presumably calculates the average percentage of work-items that are delivered on time.

#### *3.4.14.3 Predictive Object Points*

Studies **P1.02** and **P2.01** mention *Predictive Object Points* as *a new way of sizing object oriented development projects*. It was introduced in 1998 by Minkiewics, which was in turn based on previous work by Chidamber et al. and Henderson-Sellers in 1994 and 1996 respectively. It was originally brought forward as an alternative for line of code and function point estimation metrics, because they were deemed unsatisfactory in Object Oriented (OO) contexts. In reality, it is an additional *code analysis metric* that has been designed specifically for the Object Oriented paradigm. It employs the code analysis metric of *Weighted Methods per Class*, which looks at each top level class and assigns a weight to the behaviours of that class that are seen by the world. That weight is determined by evaluating the effects that the behaviour has on the objects in the system (by counting the properties that this behaviour impacts), and the amount of control the objects in the system have over this behaviour (by counting the parameters of the method or the pieces of information that get passed to it) (Minkiewics, 1998). Subsequently, the weighted methods per class metric is combined with information about the grouping of objects into classes and the relationships between these classes of objects. According to Minkiewics, this value appeared to correlate to the effort associated with

implementing a solution. While there is some literature that mentions predictive object points and its application, there is little evidence that this is widely used in practice today.

#### *3.4.14.4 Technical Efficiency*

In **P1.20**, *Technical Efficiency* is introduced as *a ratio between actual hours (AH) and ideal capacity (C), where C is the ideal number of hours which a team can deliver, depending on team size, number of teams, non-working days and days spent on principal ceremonies*. The study states that the metric can be used as an indication of how well a team is being utilized, compared to the maximum capacity.

#### *3.4.14.5 Internal Efficiency*

Similarly, **P1.20** introduces the *Internal Efficiency* metric. It state that the *Internal Efficiency* is defined *as a ratio between actual hours (AH) and planned hours (PH),* and that it *measures how good the planning process is.* It continues to state that a value of 1 means there is complete alignment between estimated effort and actual outcome.

#### *3.4.14.6 Delivery on Time*

In **P1.03** and **P2.08**, several more specific metrics are introduced for determining the performance of individual teams. The first of these is the percentage of backlog items that *Delivered on Time*. In **P1.03**, it was identified as the most prevalent Agile Software Development (ASD) metric within the 26 companies they interviewed. Of these 26 companies, 23 of them used the metric for determining whether or not the team was performing admirably. On time is defined here as *within the sprint that they were initially included in*. The thought behind this metric states that a team who delivers a larger percentage of its backlog items *on time*, performs better than a team with a lower percentage. While intuitively this might seem true, the metric can easily be gamed by teams who understand how the metric works, leading to an inaccurate and untrustworthy metric.

#### *3.4.14.7 Impediments per Work-Item*

Study **P2.12** mentions the metric of *Impediments per Work-Item* (referred to as the *average number of impediments per task/sprint/team).* The study states that the metric can help reach the goal of

efficiency impediment resolution. Here, an impediment is not defined as an interruption of the work, but an issues that is preventing work from happening efficiently.

#### *3.4.14.8 Value Delivered over Time*

**P2.08** introduces the *Value Delivered over Time* metric, stating that it is more appropriate for software development teams than *Cost Efficiency*, because in software development, there is not necessarily a linear relationship between input and output (i.e. more lines of code does not mean more value delivered). The metric is defined as *the difference of the value of output and the value of input within a particular time window*, where *the input represents the investment to be made to obtain the unit of input to be transformed in the development process* and *the output represents the value of the transformed input* (i.e. the final product).

#### *3.4.14.9 Schedule Performance Index*

Studies **P1.24** and **P2.12** mention the *Schedule Performance Index*. While the former does not explicitly define the metric, the latter states that it is *the ratio between the earned value (i.e. the value of all tasks completed) and the planned value (i.e. the initial estimate of value of all tasks to be completed until a certain point in the project).* It states that the metric can be used to attain the goal of generating timely information on project performance, but does not explicitly state how.

#### *3.4.14.10 Fulfilment of Scope*

Study **P2.12** mentions the *Fulfilment of Scope* metric, which was defined as *the ratio between the number of tasks completed in a Sprint, and the total number of tasks in the sprint backlog.* The study also posits that the metric can be used at the scope of a release, instead of the scope of a Sprint.

#### *3.4.14.11 Net Promoter Score*

Studies **P1.03** and **P1.24** also mention the *Net Promoter Score* (NPS). This famous customer loyalty metric was introduced by Frederick F. Reichheld in 2003 and is currently said to be used by two-thirds of Fortune 1000 companies. In the original study, the NPS metric was found to correlate with profit and growth in all of the 13 industries in which it was measured. It was introduced because the classical

customer satisfaction surveys were extremely costly, lengthy, easy to game by downstream distributors, and showed little to no correlation with profit or growth. Additionally, these classical surveys often employed complicated, black box scaling functions that made it difficult to apply universally, or to encapsulate it in industry-wide standards. Instead of lengthy, complicated surveys, the NPS metric employs just a single question:

*"On a scale of 0 through 10, how likely are you to recommend this product to a friend or family member?"*

Subsequently, customers are grouped into one of three groups, depending on their answer. *Promoters* are those who are extremely likely to recommend the product, with a score of 9 or 10. *Passively Satisfied* are those who score a 7 or 8, and *Detractors* are those who score a 0 through 6. The *Net Promoter Score*, then, equals the percentage of *Promoters* minus the percentage of *Detractors*. The NPS metric thus represents the ratio of *Promoters* to *Detractors,* and has a domain of -100, where everyone is a D*etractor*, to +100, where everyone is a *Promoter*. In software development processes, the NPS of end-users can be a telling metric of the performance of a particular product in terms of customer satisfaction.

#### *3.4.14.12 Technical Debt*

**P1.03** also mentions the aspect of *Technical Debt*. This is a term used to imply additional rework that is required by the development team at a later stage, due to having opted for an easier, sloppier solution at an earlier stage. While the terms *technical debt* and *code-rot* are often used interchangeably in the industry, the latter is more formally indicative of source code that is inconsistent, misleading or faulty due to the act of refactoring, instead of making poor initial design choices. The study has shown that 65% of the investigated teams used some form of *technical debt* indicator. However, it sadly does not mention which specific *technical debt* metrics were encountered. Most of the *technical debt* metrics in literature employ source code analysis techniques that yield metrics such as application size, number of rule branches, number of helper methods, cyclomatic complexity, and expression duplication (Alfraihi et al, 2018). Others exclusively use code quality metrics such as coupling, cohesion, and complexity metrics such as depth of decomposition (Seaman & Guo, 2011).

##### *3.4.14.13 Percentage of Completed Stories*

**P1.24** mentions the *Percentage of Completed Stories* as a means of gauging the progress of a particular software development process. While it does not explicitly define what *complete* means in this context, it is presumed that the functionality should be available in the production environment(s).

##### *3.4.14.14 Standard Violation*

Studies **P1.06** and **P1.24** introduce the *Standard Violation* metric. This code quality metric is used to track the number of coding standards that are violated per sprint or iteration. It can be used to direct the team towards the behaviour which has been agreed upon and/or reasonably assumed to lead to a higher quality codebase. This metric thus tells us something about the quality of the delivered code, as well as the level of discipline exhibited by the team members.

##### *3.4.14.15 Interruptions*

While **P1.08** does not technically introduce metrics for keeping track of introductions or their effects on productivity or efficiency, it does mention the practice of allocating specific timeslots for interruptions to occur in. Additionally, they advocate setting aside a specific portion of a sprint's capacity for interruptions in the form of unexpected tasks, which is estimated based on historical data. The larger issue at hand, of course, is that the impact of interruptions on productivity has been shown to be far greater than they might at first appear (Coraggio, 1990). Given this premise, one might conclude that metrics for measuring the impact of interruptions might yield interesting insights, even if as simple as *interruptions per day* or *mode interruption duration*. **P1.20** does mention measuring *Impediments*, which comes down to the amount of hours spent working, that did not produce tangible outcome, and can be seen as an inverse of the *Value Added Time* metric.

##### *3.4.14.16 Capacity Utilization*

**P2.08** introduces the *Capacity Utilization* metric, which is defined as the work-in-progress, divided by the output capacity of the process. It states that a value smaller than 1 indicates a workload that is too low, while a value greater than 1 indicates a workload that is too high, with a perceived optimum at 1. It is not clearly defined what the *output capacity* of a software development process should be. This metric thus appears to be primarily introduced for use in industrial manufacturing, as opposed to software development processes. However, the output capacity of a software development team can

simply be equated to the amount of team members in the team under examination. Optimizing this metric could then lead to a severe reduction in concurrent contexts, and thus in excessive concurrency switching.

#### *3.4.14.17 Employee Happiness*

Study **P.08** introduces the *Employee Happiness* metric. The authors claim that a self-assigned happiness score is unconsciously projected out into the future and onto the organization and their role in it. If the team member feels like the company is in trouble, they are doing the wrong thing, or employing the wrong processes, they will feel less happy. Additionally, the employee will probably feel less happy if they are experiencing a major roadblock, or have to implement a frustrating module that does not have a proper definition of done. Thus, the authors propose asking the team member about how happy they are with the company, and how happy they are with their role within it. The team member can then use a 1 to 5 Likert scale to indicate how they feel. The metric is then tracked over time, and the authors claim that significant differences from one measurement to another might need additional investigation.

## 3.5 Overview

An overview of which papers mentioned which metrics can be found in A*ppendix E*, as well as an overview of which metrics were mentioned in which papers in A*ppendix F*.

# 4. Expert Inquiry

In this chapter, we will detail the setup, execution and results of an informal inquiry of prevalent experts in the field, in the form of an in-depth discussion and conversation about the encountered metrics. The aim of this inquiry is to discover additional software development metrics, that have not yet been discovered in the literature review of *chapter 3*. In this inquiry, we have not made any distinction as to why they were not discovered in the literature review. This could be, for example, because no prior research has been performed on this metric, no peer-reviewed work has been published on the subject, or because the literature review missed it due to not encompassing the entire body of knowledge available in literature today. In this inquiry, the experts will be asked about their view on the current state of efficiency and productivity metrics. Additionally, they will be asked to think about possible efficiency metrics that we have not encountered yet, for which they would be very interested in seeing measurement results from the industry.

## 4.1 Experts

In this section, we will introduce the people whose expertise we have requested for the *expert inquiry*. In total, four experts have been consulted during the execution, all of whom have exceptional track-records in the world of agile, and most of which are considered as an authority in the field of software development.

### 4.1.1 Jeffrey Saltz

The first expert is Professor Jeffrey Saltz, who describes himself as *an accomplished technology executive, working at the intersection of innovation, data science and business strategy*. He is currently seated as a professor at Syracuse University, as well as the Chief Executive Officer at Sage Hill Consulting. Here, his primary focus is on transforming his client's data into knowledge, knowledge into insights, and insights into business decisions. He started his career as a programmer, rose to project leader and consulting engineer, to end up as the Chief Technology Officer at Goldman Sachs, and later the Technology Director of JP Morgan Chase. Currently, he is also the head of the Agile/LEAN track at the Hawaiian International Conference on System Sciences.

### 4.1.2 Jeff Sutherland

The second expert is Jeff Sutherland Ph.D., well-known for being the co-inventor of Scum, alongside Ken Schwaber. Currently, he is a senior advisor at OpenView Venture Partners, as well as the chairman of Scrum, Inc. In 2006, Jeff Sutherland founded Scrum, Inc. in order to continue and extend thought leadership on Scrum by coaching, training, and transforming companies. Here, his primary focus is on moving Scrum beyond its initial IT focus to cover all business domains.

### 4.1.3 Frank Verbruggen

The third expert is Frank Verbruggen, who started his career as a programmer, but quickly rose to software architect within Ordina. Since then, he has been an IT architect at the Dutch Chamber of Commerce, and quickly rose to chapter lead at the Dutch ING bank. He is currently the founder and owner of Hi Efficiency and Diamond Agile, where he aims to transform organizations beyond their initial Agile scope. In the process of this thesis, Frank Verbruggen has also functioned as my external supervisor.

### 4.1.4 Kyle Aretae

The fourth expert is Kyle Aretae, who is the founder of Tech Edge, and author of *Ceremony*. He has spent his 35 year IT career shuffling between the roles of software developer, technical agile coach and trainer. He has taught over 10.000 students on over 100 topics, for over 25.000 hours in the last 25 years. He is a huge evangelist for Extreme Programming and Agile practices, with a broad skill-set and loads of experience.

## 4.2 Suggested Metrics

The following section details the metrics that have been suggested by the experts in the *expert inquiry*. In order to make sure that these metrics are defined clearly and wholly transparent, we will attempt to provide unambiguous definitions of their data points and calculations when applicable.

### 4.2.1 Priority Focus

The first additional metric, brought forward by *Jeffrey Saltz*, is the *Priority Focus*, which measures the time that an individual team member has spent adding value to the highest priority story backlog item, as a percentage of the total time spent working. The metric can be calculated for each individual team member, by taking the time that the team member has spent working on the highest priority story backlog item on the previous day, and dividing it by the total time that he or she *could* have spent on it. This metric can be calculated on multiple granularities, e.g. per day or per sprint. At the same time, the metric can easily be calculated for entire teams or companies by aggregating the individual measurements into weighted arithmetic means.

This metric can be used to determine a team's capability to *do the most important things first*. Additionally, the metric can yield interesting insights into how well the team is *swarming* on the highest priority story backlog items. The act of *swarming* has been shown to lead to a reduction of waste in software development processes (Verbruggen, Sutherland, van der Werf, Brinkkemper & Sutherland, 2019). The following sections detail the calculation of this metric for an individual team member, and aggregated into an arithmetic mean for an entire team.

#### *4.2.1.1 Member Priority Focus*

The *Member Priority Focus* for sprint $s$ and member $m$, represented by $pf_{sm}$, is given by

$$pf_{sm} = \frac{\sum_{x=1}^{|E_{sm}|} \begin{cases} p_{e_x} == true & |d_{em}| \\ otherwise & 0 \end{cases}}{wc_{sm}}$$

where $E_{sm}$ is the set of the events that occurred in sprint $s$ for member $m$, $wc_{sm}$ is the *Work Capacity* in sprint $s$ for member $m$, as outlined in *section 4.2.7.1*, $p_{e_x}$ is a Boolean value denoting whether the $x_{th}$ event $e_x$ was marked as targeting the highest current priority, and $d_{em}$ is the set of timestamps included in the duration of event $e$ and the *Work Schedule* of member $m$, as out lined in *section 4.2.7.2.*

#### *4.2.1.2 Mean Team Priority Focus*

The *Mean Team Priority Focus* for sprint $s$ and team $t$, represented by $\mu_{pf_{st}}$, is given by

$$\mu_{pf_{st}} = \frac{\sum_{m=1}^{|M_{ts}|} pf_{sm}}{|M_{ts}|}$$

where $M_{ts}$ is the set of the members of team $t$ who have participated in sprint $s$, and $pf_{sm}$ is the *Member Priority Focus* for sprint $s$ and member $m$, as outlined in *section 4.2.1.1.*

## 4.2.2 Context Concurrency

The second additional metric, brought forward by *Frank Verbruggen*, is the *Context Concurrency* metric. This metric determines the maximum amount of story backlog items that the team has had to work on concurrently throughout a day, sprint or project. Superfluous context switching can hurt productivity, and keeping the amount of concurrent contexts to switch between to a feasible minimum will help minimize its impact. The metric denotes the maximum number of stories that were *in progress* at any given time, during a particular period of time.

#### *4.2.2.1 Context Concurrency*

The *Context Concurrency* of sprint $s$ at timestamp $t$, represented by $cc_{st}$, is given by

$$cc_{st} = |S_t| - |F_t|$$

where $S_t$ is the set of all stories that were started at timestamp $t$, and $F_t$ is the set of all stories that were finished at timestamp $t$.

#### *4.2.2.2 Maximum Context Concurrency*

The *Maximum Context Concurrency* of sprint $s$, represented by $mcc_s$, is given by

$$mcc_s = \bigvee_{t = s_s}^{f_s} cc_{st}$$

where $f_s$ is the timestamp at which sprint $s$ was finished, $s_s$ is the timestamp at which sprint $s$ was started, and $cc_{st}$ is the *Context Concurrency* of sprint $s$ at timestamp $t$, as outlined in *section 4.2.2.1*.

### 4.2.3 Degree of Swarming

The third additional metric, brought forward by *Jeff Sutherland*, is the *Degree of Swarming*. This metric determines the degree of collaboration and teamwork within the team. It indicates whether team members tend to work on story backlog items individually or in association with other members of the team. It is defined here as the percentage of the team that has performed work on a specific story during a particular day, whether this was two minutes or eight hours.

#### *4.2.3.1 Story Degree of Swarming*

The *Story Degree of Swarming* on story backlog item $i$ on day $d$, represented by $dos_{id}$, is given by

$$dos_{id} = \frac{|M_{id}|}{|M_d|}$$

where $M_{id}$ is the set of all members who participated in work performed on story $i$ on day $d$, and $M_d$ is the set of all members who were working on day $d$.

#### *4.2.3.2 Mean Day Degree of Swarming*

The *Mean Day Degree of Swarming* on day $d$, represented by $\mu_{dos\,d}$, is given by

$$\mu_{dos\,d} = \frac{\sum_{x=1}^{|I_d|} dos_{i_x d}}{|I_d|}$$

where $I_d$ is the set of all story backlog items that were in progress at any time during day $d$, and $dos_{i_x d}$ is the *Story Degree of Swarming* on the $x_{th}$ story backlog item $i_x$ on day $d$, as outlined in *section 4.2.3.1.*

#### *4.2.3.3 Mean Sprint Degree of Swarming*

The *Mean Sprint Degree of Swarming* on sprint $s$, represented by $\mu_{dos\,s}$, is given by

$$\mu_{dos\,s} = \frac{\sum_{x\,=\,1}^{|D_s|} \mu_{dos\,d_x}}{|D_s|}$$

where $D_s$ is the set of days in sprint $s$, and $\mu_{dos\,d}$ is the *Mean Day Degree of Swarming* on the $x_{th}$ day $d_x$, as outlined in *section 4.2.3.2*.

## 4.2.4 Small Correct Change Into Production

The fourth additional metric, brought forward by *Kyle Aretae*, is the *Small Correct Change Into Production* (SCCIP). This metric looks at the overhead of the act of deploying the product into production. It is defined as the time it takes for a single, extremely simple change to the code base, to be available in the production environment(s). If the target team works with deployment windows, it is assumed that the last deployment window has *just* closed. Kyle has seen this metric range from under 5 minutes in some of the truly high-performance teams, to over a year in some of the worst.

The *Simple Correct Change Into Production* for project $p$ , represented by $sccip_p$, is given by

$$sccip_p = t_d - t_c$$

Where $t_d$ is the timestamp at which the change is available in production, and $t_c$ is the timestamp at which the change was committed.

## 4.2.5 Process Efficiency

The fifth proposed metric, brought forward by *Jeff Sutherland* and *Frank Verbruggen*, is the *Process Efficiency* metric. This metric determines the efficiency of a software development team from the perspective of their work, instead of the individual team members. It is defined as the value-added-time divided by the total time spent working. Here, excellency measures a low throughput time, but could also lead to a low capacity utilization.

#### *4.2.5.1 Story Process Efficiency*

The *Story Process Efficiency* for story backlog item $i$, in sprint $s$, represented by $pe_{is}$, is given by

$$pe_{is} = \frac{\sum_{x=1}^{|E_{smi}|} f_{e_x} - s_{e_x}}{ct_i}$$

where $E_{smi}$ is the set of the events that occurred in sprint $s$ for member $m$, targeting story backlog item $i$, $ct_i$ is the *Story Cycle Time* of story backlog item $i$, as outlined in *section 4.2.7.4*, $f_{e_x}$ is the timestamp at which the $x_{th}$ event $e_x$ has finished, and $s_{e_x}$ is the timestamp at which the $x_{th}$ event $e_x$ has started.

#### *4.2.5.2 Mean Team Process Efficiency*

The *Mean Team Process Efficiency* for sprint $s$, represented by $\mu_{pe_s}$, is given by

$$\mu_{pe_s} = \frac{\sum_{x=1}^{|I_s|} pe_{i_x s}}{|I_s|}$$

where $I_s$ is the set of all story backlog items in sprint $s$, and $pe_{i_x s}$ is the *Story Process Efficiency* of the $x_{th}$ story $i_x$ in sprint $s$, as outlined in *section 4.2.7.2*.

### 4.2.6 Innovation Income

The final proposed metric, brought forward by *Frank Verbruggen* and *Kyle Aretae*, is the *Innovation Income* metric. This metric determines the percentage of an organization's income that's coming from innovations. It posits that if a significant part of the value delivered by an organization comes from recent innovation, the organization has the ability to innovate, and dares to move. Such an organization has the ability to change the way they operate on their markets, and can quickly react to changing circumstances.

The *Innovation Income* $ii$ for organization $o$, denoted by $ii_o$, is given by

$$ii_o = \frac{r_{<2}}{r}$$

Where $r_{<2}$ is the amount of yearly revenue obtained through projects that were released within the last two years, while $r$ is the total amount of yearly revenue for the organization. While the initial cut-off is set at two years, empirical validation might show more optimal values for this threshold.

### 4.2.7 Intermediate Metrics

While the following metrics are not part of the set of metrics suggested by the experts, their values are needed for the calculation of some of the metrics that were. Their definitions are stated below in order to provide an accurate and unambiguous account of how their calculations are done.

#### *4.2.7.1 Member Work Capacity*

The *Work Capacity* in sprint $s$ for member $m$, represented by $wc_{sm}$ is given by

$$wc_{sm} = \sum_{x=1}^{|D_{sm}|} f_{md_x} - s_{md_x}$$

where $D_{sm}$ is the set of days during sprint $s$ on which member $m$ worked on the project, $f_{md_x}$ is the time at which member $m$ stopped working on the $x_{th}$ day $d_x$, and $s_{md_x}$ is the time at which member $m$ started working on the $x_{th}$ day $d_x$.

#### *4.2.7.2 Work Schedule*

The *Work Schedule* of member $m$ in sprint $s$, represented by $U_{ms}$, is the union of the intervals of the times that member $m$ worked during sprint $s$, and is given by

$$U_{ms} = \bigcup_{x=1}^{|D_{ms}|} [s_{md_x}, f_{md_x}]$$

where $D_{ms}$ is the set of days that member $m$ worked during sprint $s$, $s_{md_x}$is the time at which member $m$ started working on the $x_{th}$ day $d_x$, and $f_{md_\mathrm{x}}$ is the time at which member $m$ stopped working on the $x_{th}$ day $d_x$.

#### *4.2.7.3 Event Duration*

The *Event Duration* for event $e$ of member $m$, represented by $d_{em}$ is given by

$$d_{em} = \{x \mid x \in U_{ms}, x \in [s_e, f_e]\}$$

where $U_{ms}$ is the *Work Schedule* of member $m$ in sprint $s$, as defined in *section 4.2.7.2*, $s_e$ is the time at which event $e$ has started, and $f_e$ is the time at which event $e$ has finished.

#### *4.2.7.4 Story Cycle Time*

The *Story Cycle Time* of story backlog item $i$, represented by $ct_i$, is given by

$$ct_i = f_i - s_i$$

where $f_i$ is the timestamp at which story backlog item $i$ is finished, and $s_i$ is the timestamp at which story backlog item $i$ is started.

#### *4.2.7.5 Story Cycle Interval*

Similarly, the *Story Cycle Interval* of story backlog item $i$, represented by $ci_i$, is given by

$$ci_i = \{[s_i, f_i]\}$$

where $f_i$ is the timestamp at which story backlog item $i$ is finished, and $s_i$ is the timestamp at which story backlog item $i$ is started.

#### *4.2.7.6 Mean Team Interruption Count*

The *Mean Team Interruption Count* for sprint $s$, represented by $\mu_{ic_s}$, is given by

$$\mu_{ic_s} = \frac{|I_s|}{|M_s|}$$

where $I_s$ is the set of the interruptions that occurred in sprint $s$, and $M_s$ is the set of the team members who participated in sprint $s$.

#### *4.2.7.7 Mean Team Interruption Duration*

The *Mean Team Interruption Duration* for sprint $s$ and team $t$, represented by $\mu_{id_{st}}$, is given by

$$\mu_{id_{st}} = \frac{\sum_{x=1}^{|I_s|} f_{i_x} - s_{i_x}}{c_{i_s}}$$

where $I_s$ is the set of the interruptions that occurred in sprint $s$, $f_{i_x}$ is the time at which the $x_{th}$ interruption $i_x$ was finished, and $s_{i_x}$ is the time at which the $x_{th}$ interruption $i_x$ started.

# 5. Metric Quality Model

In this chapter, we will introduce a model for metric strength, that can be used to determine whether a specific metric can be deemed *strong*. This model consists of five qualities that a metric should have in order to be considered a *strong* metric. This model for metric strength was developed through in-depth discussion of metric strength with the experts introduced in section 4.1, in which tacit knowledge about *what makes a metric good or bad*, was extrapolated and distilled into explicit knowledge.

These qualities state that a *strong* metric should (a) be simple to explain and simple to measure, (b) be difficult to optimize without increasing business value (c) correlate strongly with increased business value when optimized, (d) be useable in multiple contexts, without confusing edge-cases, and (e) have an unambiguous and transparent definition of its data points, as well as how those data points are used in its calculations. In the remainder of this study, we will refer to these qualities as **simple**, **hard-to-game, outcome-oriented**, **universal**, and **transparent** respectively. Together, these criteria spell the acronym **SHOUT**.

The rest of this chapter discusses these qualities in more detail, and ends with an assessment of the discovered metrics in terms of the SHOUT qualities.

## 5.1 Qualities

### 5.1.1 Simple

The first quality criteria is *simplicity*. This addresses the need for a metric to be simple to explain, measure and interpret. It also takes into account how much effort, in terms of time and energy, is required to take the required measurements. Finally, it takes into account the perceived impact on the productivity of the team under investigation. If taking the required measurements takes only a second, but has to be done many times a day, the overall effort required is low, but the impact on overall team productivity might be too high, because of the numerous interruptions that it would cause.

### 5.1.2 Hard to Game

Then, the metric is judged on whether or not its value is hard to game. In the context of this study, *hard to game* is defined as *being difficult to optimize without increasing business value*. This means that we do not truly care whether or not a metric is easy to game or not, as long as the act of gaming still results in the intended increase in business value. An excellent of example of a metric that is hard to game in this sense, is *Work in Progress*. The emergence of the *hard-to-game* quality is not all that surprising, as E.M. Goldratt's '*tell me how you measure me, and I'll tell you how I'll behave'* comes to mind.

### 5.1.3 Outcome Oriented

*Strong* metrics should also *show a strong correlation with increased business value when optimized*. This means that the metric should give a clear indication of where that optimum might be, and can reasonably be assumed to increase business value when a process gets closer to that optimum.

### 5.1.4 Universal

For a metric to be *universal*, it must be applicable to many different contexts, and not just software development or industrial manufacturing. Similarly, it should not have any confusing edge-cases for specific circumstances, resulting in invalid measurements or values.

### 5.1.5 Transparent

Finally, metrics should be transparent, meaning that they should have an explicit and unambiguous definition of their data points. Additionally, all of the metrics should be transparent in the sense that they should unambiguously define how those data points are used to calculate the final metric value(s).

# 6. Systematic Mapping

## 6.1 Data Structures

The systematic mapping uses eight data structures in order to capture the information that was ascertained in the structured literature review. In this section, we will introduce each data structure, and demonstrate their underlying relationships. These are shown below in *figure 2*.

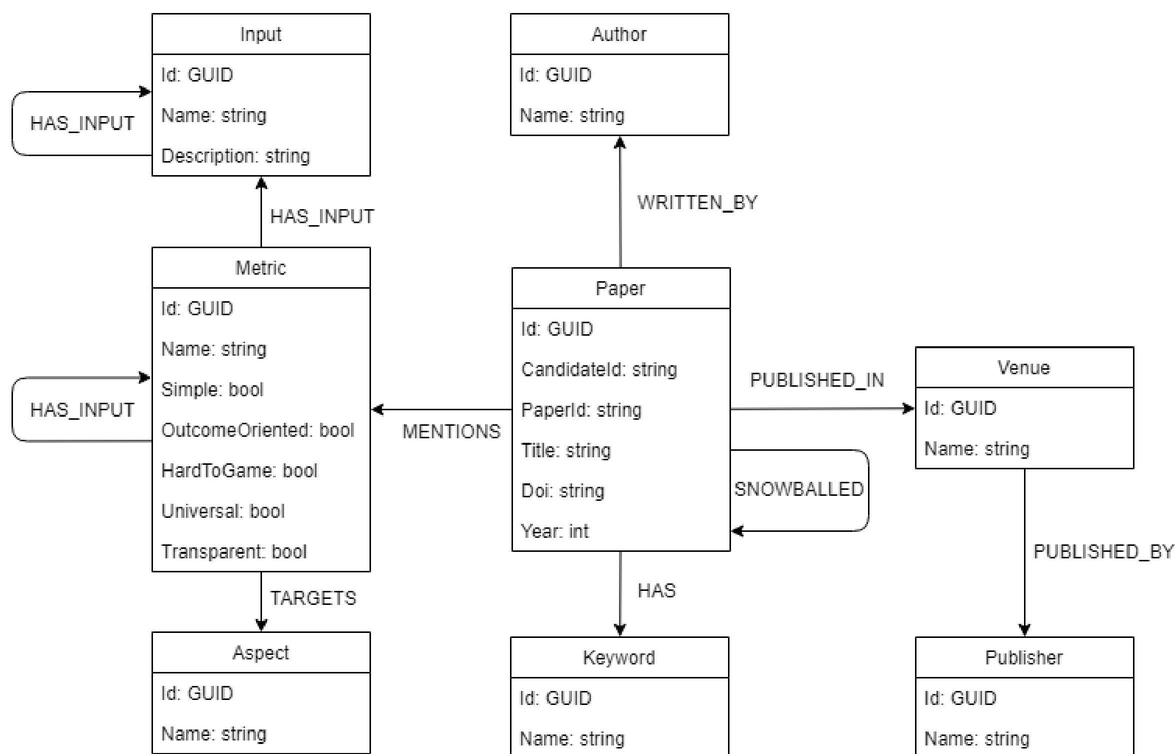



*Figure 2 - Data Structures*

## 6.2 Technology

We implemented the data structures introduced in the previous section in a NEO4J graphing database. Subsequently, an ASP.NET Core 2.1 application was created in order to enter and manipulate the entries and their relationships. Finally, the data set was coupled to a graph interpreter, allowing the inspection and analysis of the data based on the Cypher querying language.

The data structures and their relationships were inserted into the graph database. This resulted in a database schema as shown below in *figure 3*.

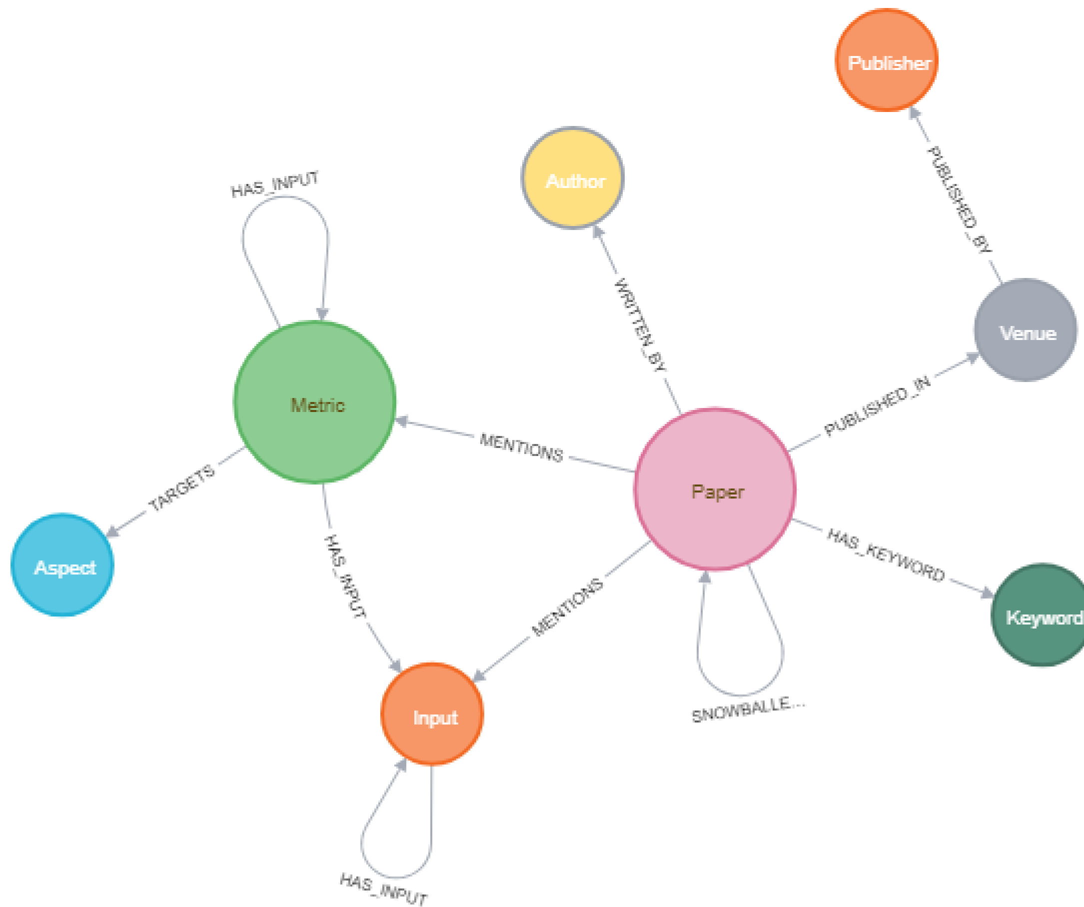



*Figure 3 - Database Schema*

The resulting systematic mapping became too large to coherently visualize with NEO4J's visualization capabilities. The visualization of the resulting systematic mapping, was thus done using a small, custom-made vis.js app, resulting in a set of visual aggregates that highlight particular aspects of the systematic mapping. These aggregations are shown in chapter 7.

# 6.3 Axial Encoding

In this section we will apply the Grounded Theory technique of *axial encoding*, in order to encode every metric so that it *targets* at least one *aspect* of the software development process, as well as to encode every input so that it *belongs* to at least one category of inputs.

### 6.3.1 Aspects

The following table shows the application of o*pen encoding* as well as *axial encoding* to the discovered software development metrics. Note that this table only shows the encodings, as well as the amount of metrics included in the encoding. *Appendix G* shows a complete overview of which metric was assigned to which encodings.

| Aspect Encoding | | Metrics | |
|---|---|---|---|
| **Axial Encoding** | **Open Encoding** | | |
| Efficiency | Time | 54 | 15 |
| | Rework | | 3 |
| | Cycle Times | | 9 |
| | Delivery | | 8 |
| | Flow | | 2 |
| | Impediments | | 2 |
| | Burndown | | 3 |
| | Presumed | | 4 |
| | Effort | | 8 |
| Complexity | Cohesion | 61 | 4 |
| | Coupling | | 22 |
| | Dependencies | | 1 |
| | Code Generation | | 8 |
| | Encapsulation | | 3 |
| | Inheritance | | 16 |
| | Cyclomatic Complexity | | 2 |
| | Expression Tree | | 5 |
| Risk | Clarity | 2 | 2 |
| Size | Effort | 22 | 2 |
| | Components | | 7 |
| | Estimation | | 5 |
| | Code Churn | | 6 |
| | Fulfilment | | 2 |
| Quality | Anti-Patterns | 33 | 8 |
| | Defects | | 17 |
| | Documentation | | 1 |
| | Tests | | 7 |
| Composition | Team Composition | 5 | 4 |
| | Project Composition | | 1 |
| Cost | Cost of Performed Work | 20 | 6 |
| | Cost of Performed Rework | | 3 |
| | Cost of Quality | | 5 |

| | Financial | | 6 |
|---|---|---|---|
| Design | Requirements | 6 | 4 |
| | Reviews | | 2 |
| Process | Story | 11 | 3 |
| | Iteration | | 2 |
| | Team Member | | 2 |
| | Project | | 3 |
| | Requirements | | 1 |
| Satisfaction | Satisfaction | 2 | 2 |

*Table 9 - Axial Encoding of Software Development Aspects*

The encoding has resulted in a set of 10 software development process aspects, which have been listed below in *table 5*.

| Axial Encoding | Open Encodings | Metrics |
|---|---|---|
| Efficiency | 9 | 54 |
| Complexity | 8 | 61 |
| Risk | 1 | 2 |
| Size | 5 | 22 |
| Quality | 4 | 33 |
| Composition | 2 | 5 |
| Cost | 4 | 20 |
| Design | 2 | 6 |
| Process | 5 | 11 |
| Satisfaction | 1 | 2 |

*Table 5 - Resulting Aspects of Axial Encodin*

In total, there are 217 encodings over 197 metrics, with the 20 aspects shown below in *table 6* being assigned to multiple encodings.

| Metric | Amount | Encodings |
|---|---|---|
| Interrupted Time | 2 | Efficiency - Time |
| | | Efficiency - Impediments |
| Descendant Method to Method Export Coupling | 2 | Complexity - Coupling |
| | | Complexity - Inheritance |
| Information-Flow Based Inheritance Coupling | 2 | Complexity - Coupling |
| | | Complexity - Inheritance |
| Lines of Code (per Unit of Time) | 2 | Efficiency - Effort |
| | | Size - Components |
| Lines of Code per Method | 2 | Complexity - Expression Tree |
| | | Size - Components |
| Number of Interfaces | 2 | Complexity - Encapsulation |
| | | Size - Components |
| Percentage of Adopted Work | 2 | Size - Effort |

| | | |
|---|---|---|
| | | Size - Estimation |
| Percentage of Found Work | 2 | Size - Effort |
| | | Size - Estimation |
| New Classes Per Release | 2 | Efficiency - Effort |
| | | Size - Code Churn |
| New Features Per Release | 2 | Efficiency - Effort |
| | | Size - Code Churn |
| New Lines of Code Per Release | 2 | Efficiency - Effort |
| | | Size - Code Churn |
| New Methods Per Release | 2 | Efficiency - Effort |
| | | Size - Code Churn |
| Percentage of Completed Stories | 2 | Efficiency - Effort |
| | | Size - Fulfilment |
| Halstead Complexity Metric | 2 | Efficiency - Effort |
| | | Size - Code Churn |
| Normalized Distance from Main Sequence | 2 | Complexity - Coupling |
| | | Quality - Anti Patterns |
| Parameters per Method | 2 | Complexity - Cohesion |
| | | Quality - Anti Patterns |
| Number of Defects Found by Tests | 2 | Quality - Defects |
| | | Quality - Tests |
| Average Fault Cost | 2 | Quality - Defects |
| | | Cost - Cost of Performed Rework |
| Faults Slip Through | 2 | Quality - Defects |
| | | Cost - Cost of Performed Rework |
| Improvement Potential | 2 | Quality - Defects |
| | | Cost - Cost of Performed Rework |

*Table 6 – Metrics assigned to multiple aspects.*

### 6.3.2 Inputs

The following table shows the application of *open encoding* as well as *axial encoding* to the discovered software development metrics' input parameters. Note that, again, the table only shows the encodings, as well as the amount of inputs belonging to each encoding. For a full overview of what input was assigned to which encoding, see *Appendix H*.

| Input Encoding | | Inputs | |
|---|---|---|---|
| **Axial Encoding** | **Open Encoding** | | |
| Backlog | Backlog | 6 | 6 |
| Company | Company | 1 | 1 |
| Defects | Defect Counts | 6 | 3 |

|  |  |  |  |
|---|---|---|---|
|  | Defect Cost |  | 2 |
|  | Defect Discovery |  | 1 |
| Deployment | Build | 3 | 2 |
|  | Version Control |  | 1 |
| Estimate | Size Estimate | 7 | 5 |
|  | Clarity Estimate |  | 1 |
|  | Commitment Estimate |  | 1 |
| Lifecycle | Day Lifecycle | 20 | 4 |
|  | Interruption Lifecycle |  | 3 |
|  | Iteration Lifecycle |  | 2 |
|  | Product Lifecycle |  | 2 |
|  | Team Lifecycle |  | 2 |
|  | Test Lifecycle |  | 2 |
|  | Work Item Lifecyle |  | 5 |
| Financial | Cost | 14 | 12 |
|  | Revenue |  | 2 |
| Iteration | Commitment | 8 | 2 |
|  | Delivery |  | 4 |
|  | Lifecycle |  | 2 |
| Schedule | Planned Production | 7 | 2 |
|  | Planning |  | 3 |
|  | Unplanned |  | 2 |
| Source Code | Code Churn | 15 | 5 |
|  | Code Complexity |  | 6 |
|  | Components |  | 2 |
|  | Code Coupling |  | 2 |
| Survey | Customer Inquiry | 2 | 1 |
|  | Team Member Inquiry |  | 1 |
| Team | Team Churn | 5 | 2 |
|  | Team Composition |  | 2 |
|  | Team Delivery |  | 1 |
| Test | Test Result | 4 | 1 |
|  | Test Lifecycle |  | 2 |
|  | Test Count |  | 1 |
| Work Day | Day Lifecycle | 3 | 2 |
|  | Planning |  | 1 |
| Work Item | Work Item Count | 17 | 2 |
|  | Work Item Estimate |  | 4 |
|  | Work Item Financials |  | 2 |
|  | Work Item Lifecyle |  | 5 |
|  | Work Item Meta Data |  | 3 |

| | Work Item Requirements | | 1 |
|---|---|---|---|

Table 11 – Axial Encoding of Software Development Input Groups

The encoding has resulted in a set of 15 groups of input parameters to software development metrics, which have been listed below in *table 12*.

| **Axial Encoding** | **Open Encodings** | **Inputs** |
|---|---|---|
| Backlog | 1 | 6 |
| Company | 1 | 1 |
| Defects | 3 | 6 |
| Deployment | 2 | 3 |
| Estimate | 3 | 7 |
| Lifecycle | 7 | 20 |
| Financial | 2 | 14 |
| Iteration | 3 | 8 |
| Schedule | 3 | 7 |
| Source Code | 4 | 15 |
| Survey | 2 | 2 |
| Team | 3 | 5 |
| Test | 3 | 4 |
| Work Day | 2 | 3 |
| Work Item | 6 | 17 |

Table 7 - Resulting Input Groups of Axial Encoding

In total, there are 118 encodings over 84 inputs, with the 34 inputs shown below in *table 8* being assigned to multiple encodings.

| **Input** | **Amount** | **Encodings** |
|---|---|---|
| Amount of Defects | 2 | Backlog - Backlog |
| | | Defects - Defect Counts |
| Amount of Open Defects | 2 | Backlog - Backlog |
| | | Defects - Defect Counts |
| Commit Timestamp | 2 | Deployment - Version Control |
| | | Lifecycle - Product Lifecycle |
| Defect Cost | 2 | Defects - Defect Cost |
| | | Financial - Cost |
| Adjusted Sprint Forecast | 2 | Estimate - Commitment Estimate |
| | | Iteration - Commitment |

| Sprint Story Point Original Forecast | 2 | Estimate - Size Estimate<br>Iteration - Commitment |
|---|---|---|
| Amount of Stories in Iteration | 2 | Backlog - Backlog<br>Iteration - Delivery |
| Sprint End Timestamp | 2 | Lifecycle - Iteration Lifecycle<br>Iteration - Lifecycle |
| Sprint Start Timestamp | 2 | Lifecycle - Iteration Lifecycle<br>Iteration - Lifecycle |
| Process Capacity | 2 | Iteration - Delivery<br>Schedule - Planned Production |
| Planned Workday Start Timestamp | 2 | Lifecycle - Day Lifecycle<br>Schedule - Planning |
| Planned Workday End Timestamp | 2 | Lifecycle - Day Lifecycle<br>Schedule - Planning |
| Interruption End Timestamp | 2 | Lifecycle - Interruption Lifecycle<br>Schedule - Unplanned |
| Interruption Start Timestamp | 2 | Lifecycle - Interruption Lifecycle<br>Schedule - Unplanned |
| Team Members Added | 2 | Lifecycle - Team Lifecycle<br>Team - Team Churn |
| Team Members Removed | 2 | Lifecycle - Team Lifecycle<br>Team - Team Churn |
| Units Produced | 2 | Iteration - Delivery<br>Team - Team Delivery |
| Test Deleted Timestamp | 2 | Lifecycle - Test Lifecycle<br>Test - Test Lifecycle |
| Test Created Timestamp | 2 | Lifecycle - Test Lifecycle<br>Test - Test Lifecycle |
| Workday Start Timestamp | 2 | Schedule - Planning<br>Work Day - Day Lifecyle |
| Workday End Timestamp | 2 | Schedule - Planning<br>Work Day - Day Lifecyle |
| Amount of Available Workdays | 2 | Schedule - Planning<br>Work Day - Planning |
| Amount of Open Work Items | 2 | Backlog - Backlog<br>Work Item - Work Item Count |
| Amount of Work Items | 2 | Backlog - Backlog<br>Work Item - Work Item Count |
| Work Item Function Point Estimate | 2 | Estimate - Size Estimate<br>Work Item - Work Item Estimate |
| Work Item Story Point Estimate | 2 | Estimate - Size Estimate<br>Work Item - Work Item Estimate |

| | | |
|---|---|---|
| Work Item Use Case Point Estimate | 2 | Estimate - Size Estimate<br>Work Item - Work Item Estimate |
| Adjusted Work Item Story Point Estimate | 2 | Estimate - Size Estimate<br>Work Item - Work Item Estimate |
| Work Item Cost | 2 | Financial - Cost<br>Work Item - Work Item Financials |
| Work Item Finished Timestamp | 2 | Lifecycle - Work Item Lifecycle<br>Work Item - Work Item Lifecycle |
| Planned Work Item Finished Time | 2 | Lifecycle - Work Item Lifecycle<br>Work Item - Work Item Lifecycle |
| Work Item Deployed Timestamp | 2 | Lifecycle - Work Item Lifecycle<br>Work Item - Work Item Lifecycle |
| Work Item Start Timestamp | 2 | Lifecycle - Work Item Lifecycle<br>Work Item - Work Item Lifecycle |
| Work Item Created Timestamp | 2 | Lifecycle - Work Item Lifecycle<br>Work Item - Work Item Lifecycle |

*Table 8 – Inputs assigned to multiple aspects.*

# 7. Aggregation

In total, we identified 44 studies mentioning software development metrics. These papers were published during the time period 1989 to 2018. Together, these studies were written by 113 individual authors, using 166 distinct keywords. They were published in 37 different venues, facilitated by 12 different publishers. Collectively, these studies mention a total of 191 software development metrics, targeting 10 different aspects of the software development process. This section shows a thorough investigation of these results.

## 7.1 Venues

*Figure 4*, shown below, illustrates the distribution of venues over publishers.

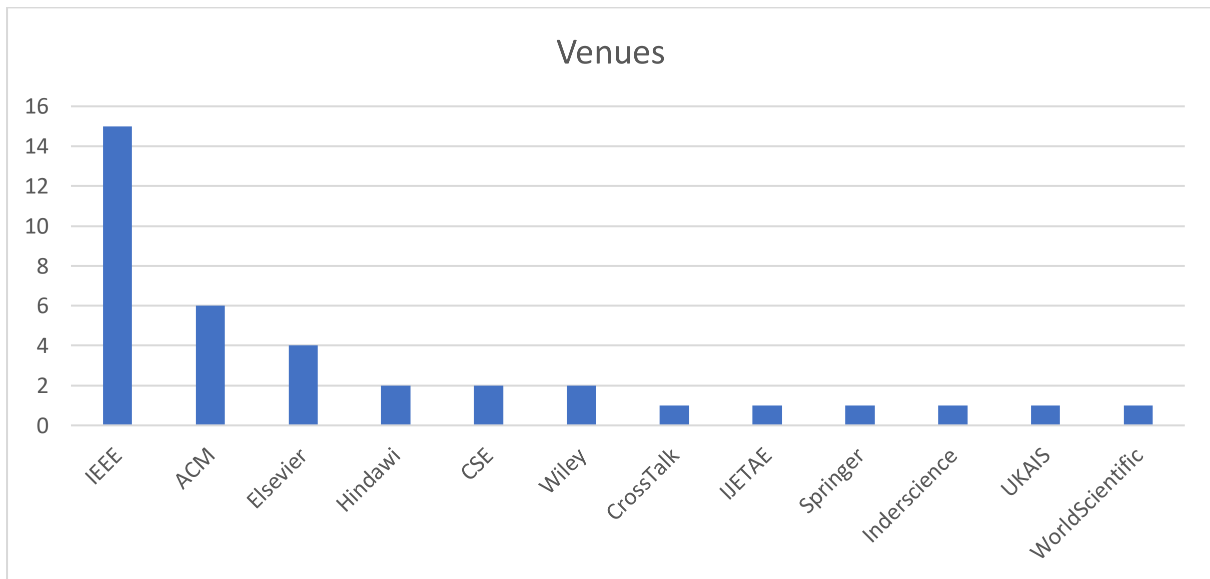



*Figure 4 - Distribution of Venues over Publishers*

## 7.2 Publishers

Collectively, these 12 publishers facilitated 37 venues, shown below in *table 9*, alongside the amount of literary works that were included from that venue.

| Publisher | Venue | Works |
|---|---|---|
| ACM | Proceedings of the International Conference on Software Engineering | 1 |
| ACM | Journal of Software Engineering | 1 |

| | | |
|---|---|---|
| ACM | Transactions on Computers | 1 |
| ACM | Journal of Electrical and Computer Engineering | 1 |
| ACM | Journal of Model-Driven Engineering Lanuages and Systems | 1 |
| ACM | Journal of Object-Oriented Programming, Systems, Languages and Applications | 1 |
| CSE | Proceedings of the International Workshop on Requirements Engineering and Testing | 1 |
| CSE | Journal of Object Technology | 2 |
| CrossTalk | Journal of Defense Software Engineering | 1 |
| Elsevier | Journal of Information and Software Technology | 1 |
| Elsevier | Journal of Systems and Software | 1 |
| Elsevier | Journal of Computers and Industrial Engineering | 1 |
| Elsevier | International Conference on System Analysis and Modeling | 2 |
| Hindawi | International Journal of Industrial Engineering | 1 |
| Hindawi | Journal of Communication and Security | 1 |
| IEEE | Proceedings of Seventh International Software Metrics Symposium | 1 |
| IEEE | Proceedings of the International Workshop on Global Software Development | 1 |
| IEEE | International Symposium on Emperical Software Engineering and Measurement | 1 |
| IEEE | International Workshop on Software Measurement | 1 |
| IEEE | International Conference on Software Process and Product Measurement | 1 |
| IEEE | Conference on the Quality of Software Architectures | 1 |
| IEEE | Journal of Software Maintenance | 2 |
| IEEE | Transactions on Software Engineering | 1 |
| IEEE | Moratuwa Engineering Research Conference | 1 |
| IEEE | Proceedings of the International Conference on Software and System Process | 1 |
| IEEE | Conference on Model-Based Methodologies for Pervasive and Embedded Software | 1 |
| IEEE | Hawaii International Conference on System Sciences | 4 |
| IEEE | IEEE Software | 3 |
| IEEE | Agile Conference | 1 |
| IEEE | Conference on Software Engineering Techniques | 1 |
| IJETAE | International Journal of Emerging Technology and Advanced Engineering | 1 |
| Inderscience | International Journal of Agile Systems and Management | 1 |
| Springer | Annals of Software Engineering | 1 |
| UKAIS | Journal of Information Systems | 1 |
| Wiley | Journal of Software Improvement and Practice | 1 |
| Wiley | Journal of Software Practice and Experience | 1 |
| World Scientific | International Journal of Software Engineering and Knowledge Engineering | 1 |

*Table 9 - Venues and their Publishers*

## 7.3 Papers

*Figure 5*, shown below, illustrates the distribution of the included work over the years. This distribution shows a significant skew towards the later end of the chart, with an arithmetic mean at June of 2008.

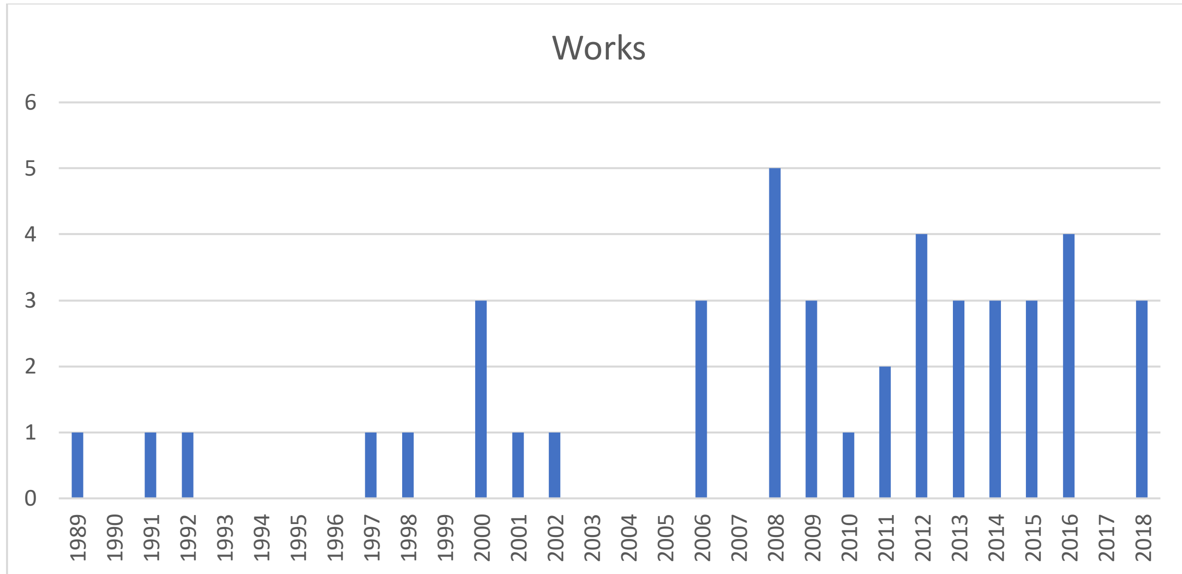



*Figure 5 - Distribution of Included Work over Time*

## 7.4 Keywords

The included work lists 166 distinct keywords, with an average of 4.8 keywords per paper. These keywords, and their occurrences are listed below in *table 10*, which is limited to showing the 25 most often used keywords. It is interesting to note the lack of overlap in the occurrences of these keywords, dropping to 1 after only 25 keywords. The complete list with keywords and their occurrences is listed in *Appendix I*.

| **Keyword** | **Occurrences** |
|---|---|
| Productivity | 7 |
| Software Metrics | 6 |
| Measurement | 5 |
| Software | 5 |
| Software Engineering | 5 |
| Software Measurement | 5 |
| Companies | 4 |
| Costs | 4 |

| | |
|---|---|
| Programming | 4 |
| Testing | 4 |
| Agile | 3 |
| Metrics | 3 |
| Refactoring | 3 |
| Scrum | 3 |
| Agile Development | 2 |
| Agile Software Development Process | 2 |
| Case Study | 2 |
| Coupling | 2 |
| Efficiency | 2 |
| Large Scale Systems | 2 |
| Lean | 2 |
| Lean Manufacturing | 2 |
| Outsourcing | 2 |
| Quality Assurance | 2 |
| Software Quality | 2 |

*Table 10 - Occurrences of 25 most Prevalent Keywords*

# 7.5 Authors

The included work is written by 113 distinct authors. In *table 11*, shown below, the authors of the included work are listed, limited to showing the authors with the most included work. The list drops to 1 included work after 14 authors, and is thus cut off at 14. The complete list with authors and their works is listed in *Appendix J*.

| First Name | Last Name | Works |
|---|---|---|
| Jeff | Sutherland | 3 |
| Zengyang | Li | 2 |
| Turgay | Aytac | 2 |
| Shekoufeh | Kolahdouz-Rahimi | 2 |
| Peng | Liang | 2 |
| Paris | Avgeriou | 2 |
| Ovunc | Bozcan | 2 |
| Mohammadreza | Sharbaf | 2 |
| Kevin | Leno | 2 |
| Howard | Haughton | 2 |
| Hessa | Alfraihi | 2 |
| Gul | Calikli | 2 |

| Claes | Wohlin | 2 |
|---|---|---|
| Ayse | Bener | 2 |

*Table 11 – Authors with the most included work*

*Figure 6*, then, shows the distribution of authors over the included work, as well as their interconnectedness.

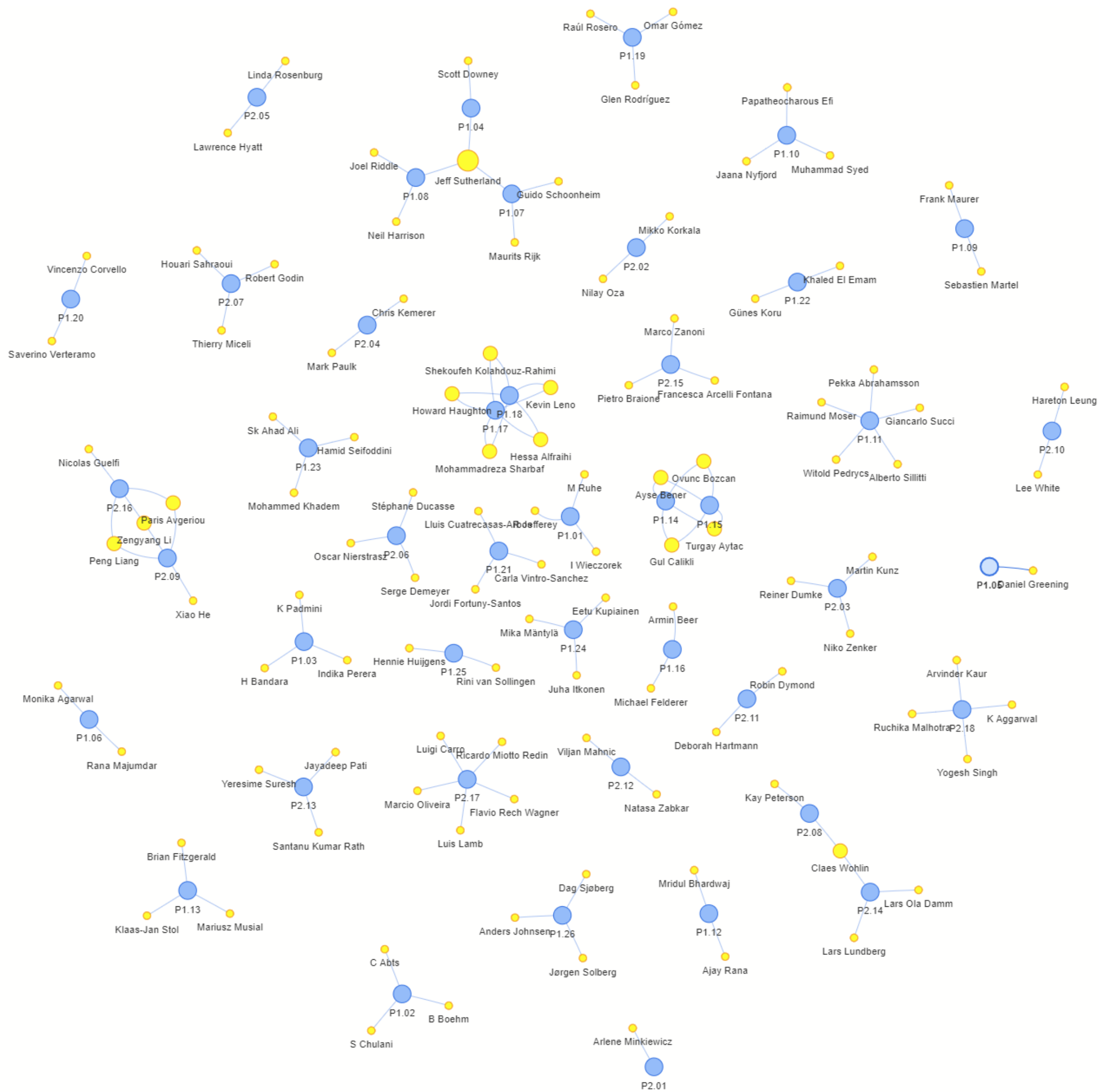



*Figure 6 - Authors per paper*

## 7.6 Metric Quality Assessment

The metrics that were discovered in the literature are listed in *appendix K*, alongside their quality assessments. These quality assessments were performed and discussed with a subset of the experts identified in *chapter 4*.

## 7.7 Metric Distributions

### 7.7.1 Chronological Distribution

When looking at the origins of the discovered metrics, *figure 7*, shown below, shows the distribution of the metric's introductions over the years. A complete table of which authors introduced which metrics in what year, is shown in A*ppendix D*. **This is a good-faith, best-effort attempt to trace each metric back to its original academic introduction, and is bound to have some inaccuracies**. Note that this figure, and its accompanying appendix, does not include all original introductions for all metrics, as not every metric could be traced back to its original academic introduction, or no reasonable deductions could be made as to where it was first introduced. This graph shows the introduction of 121 out of 197 metrics, for which the academic introduction could be deduced.

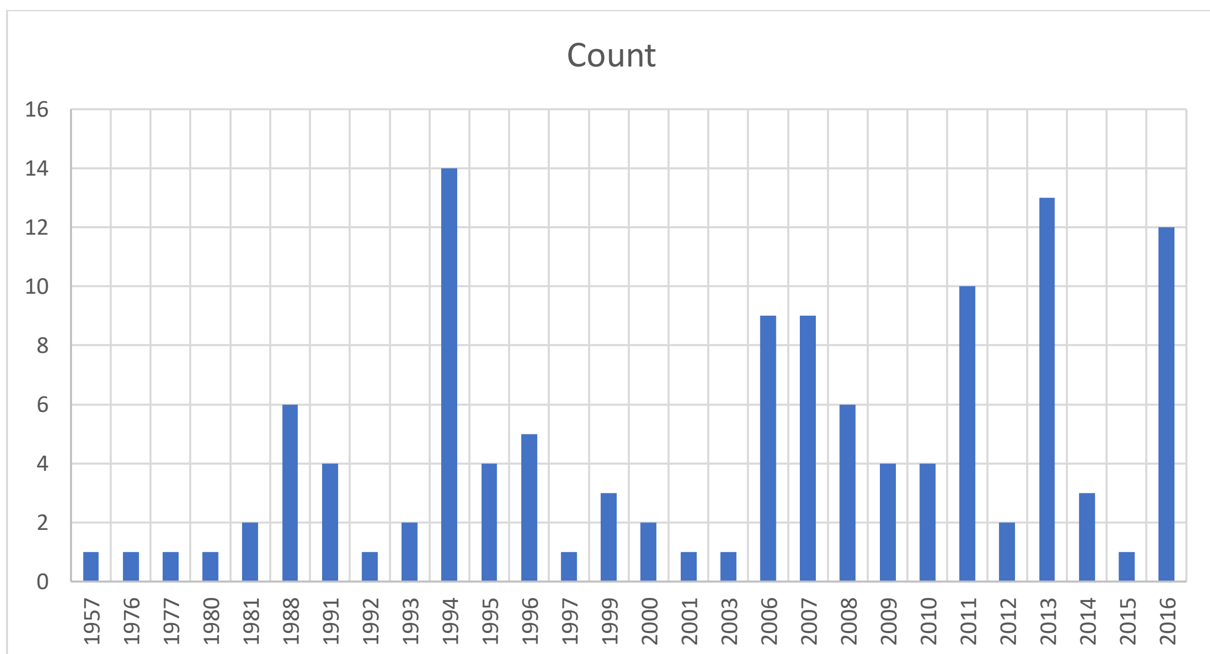



*Figure 7 - Distribution of metric Introductions over the years*

When we relate the year of origin to the aspects those metrics measure, we derive *figure 8* below. **Note that, because every metric can potentially target more than one aspect, the numbers do not necessarily match the ones in the previous figure**.

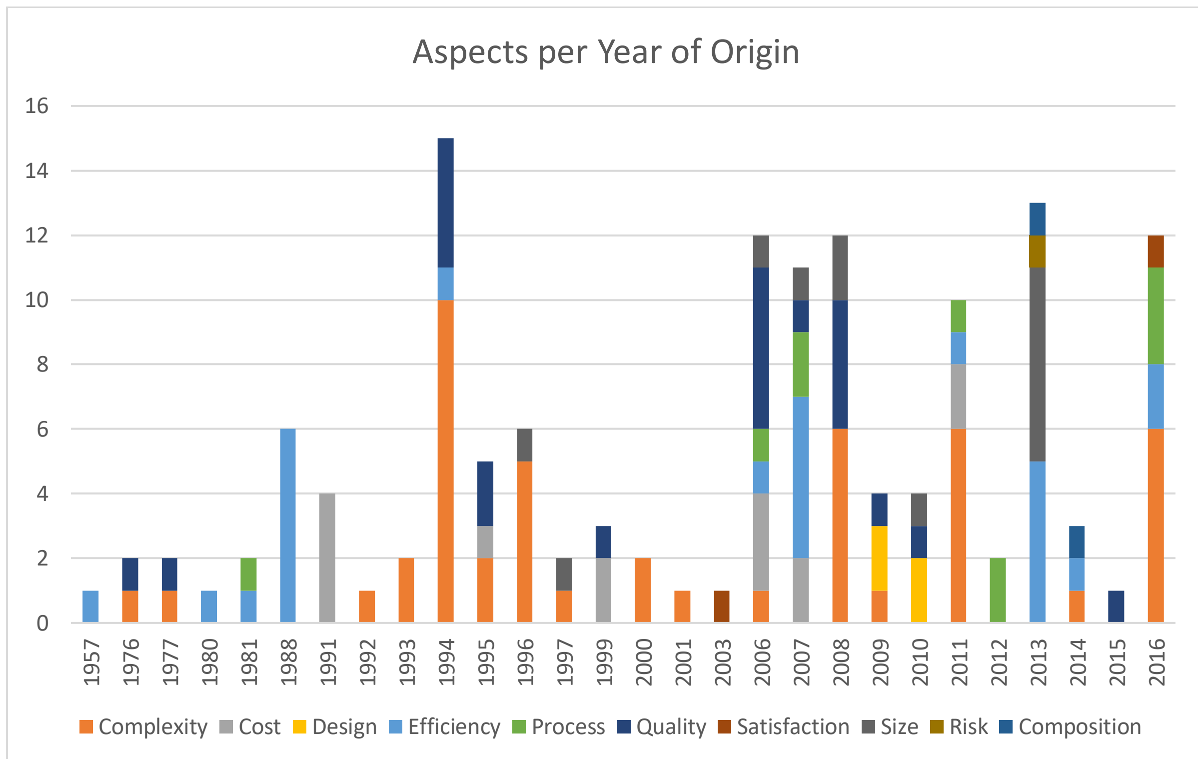



*Figure 8 - Distribution of metric Introductions over the years per aspect*

When relating the year of origin to the input groups at which their metrics look, we derive *figure 9* below. Note, again, that a metric can have multiple inputs, and each input can belong to a different group. **These numbers thus do not necessarily match with the previous figures**.

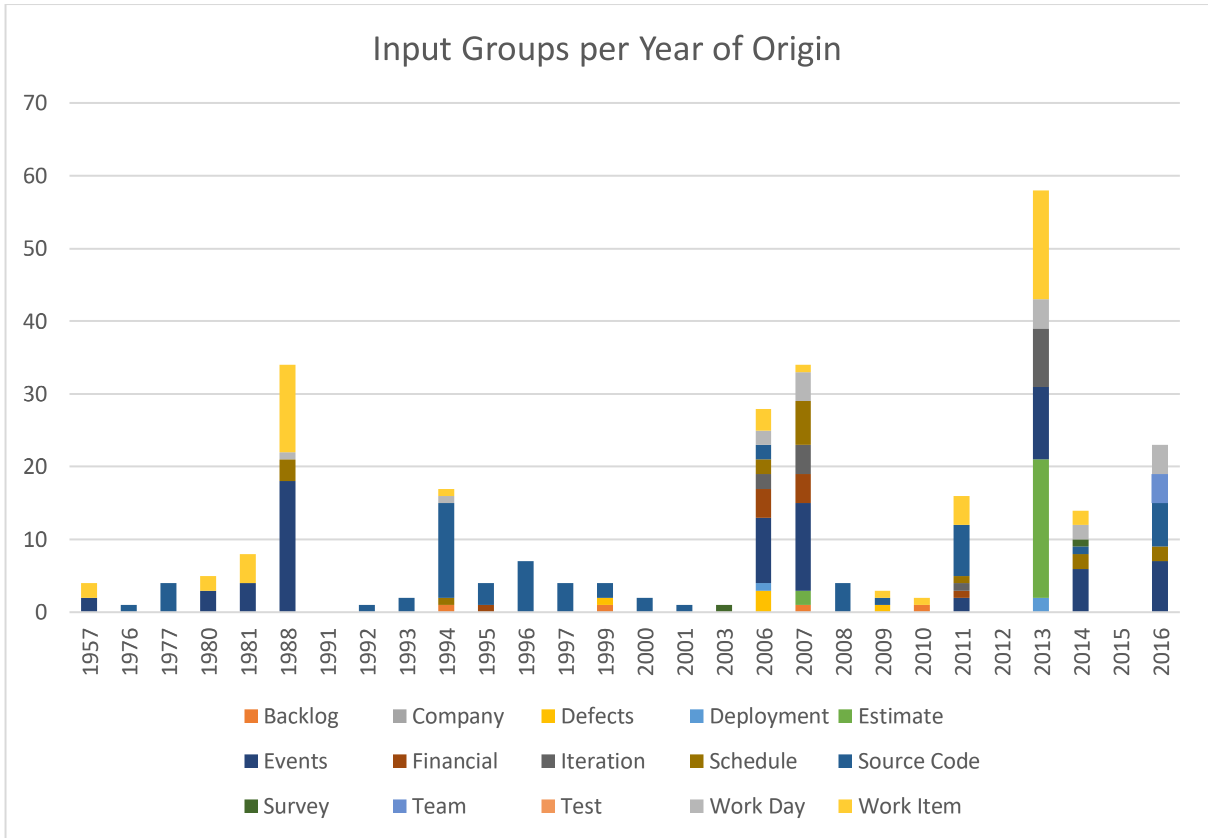



*Figure 9 - Input Groups per Year of Origin*

### 7.7.2 Conceptual Distribution

*Figure 10* below shows the distribution of the encountered metrics over the different aspects of the software development process.

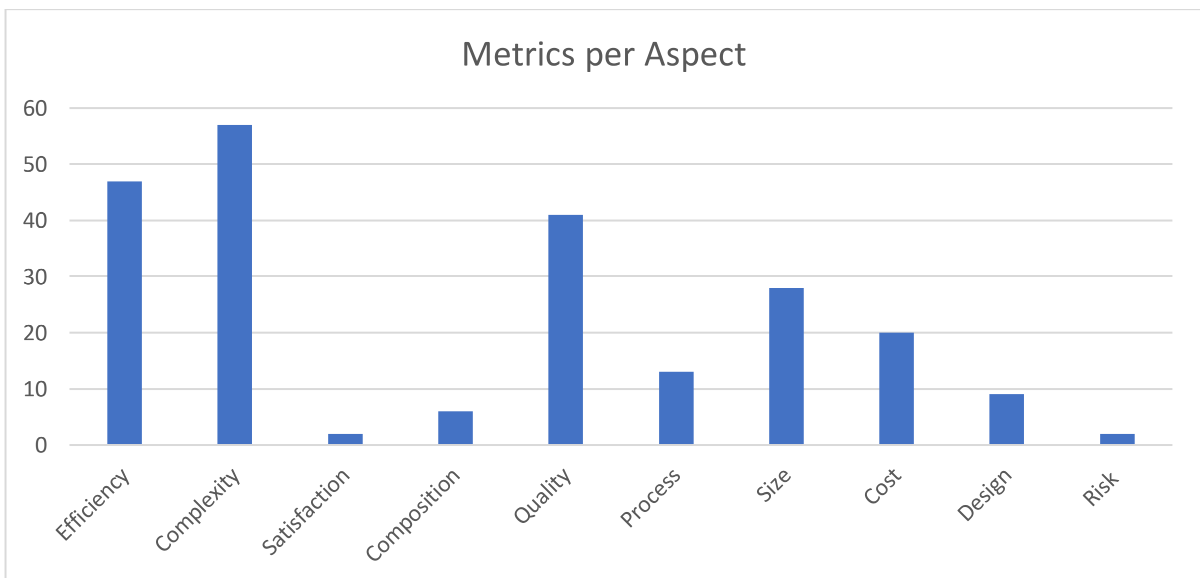


*Figure 10 - Metrics per aspect*

*Figure 11* below shows the distribution of the encountered inputs over the different axial encodings of the input groups.

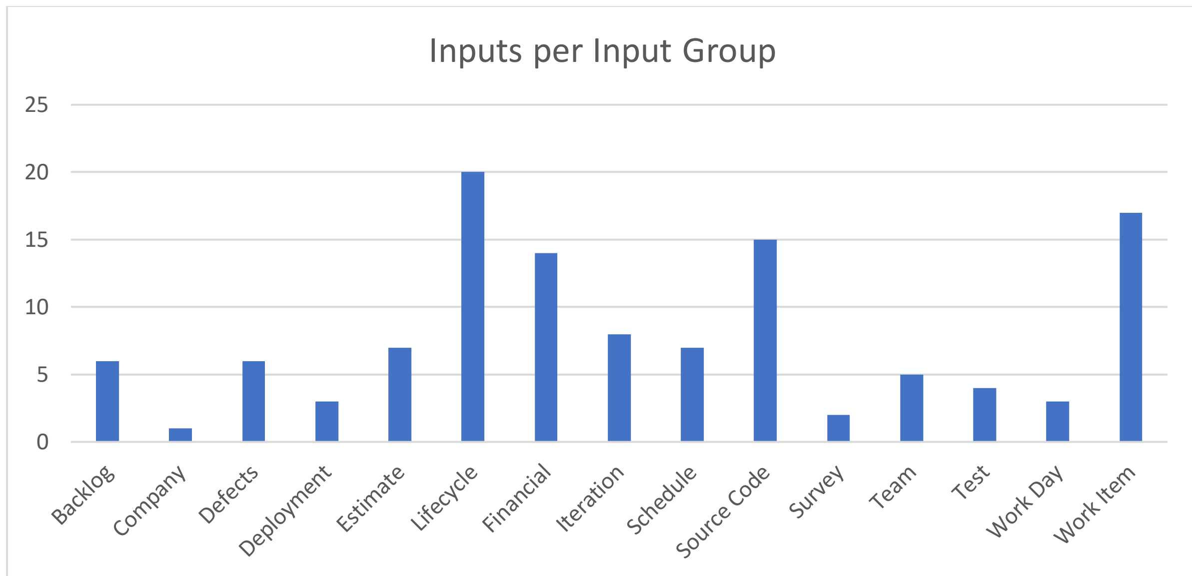



*Figure 11 - Inputs per Input Group*

### 7.7.3 Strength Distribution

*Figure 12* below shows the amount of metrics that were found to have x out of 5 strength qualities. From this figure, we can deduce that only 23 out of 197 metrics could be deemed strong.

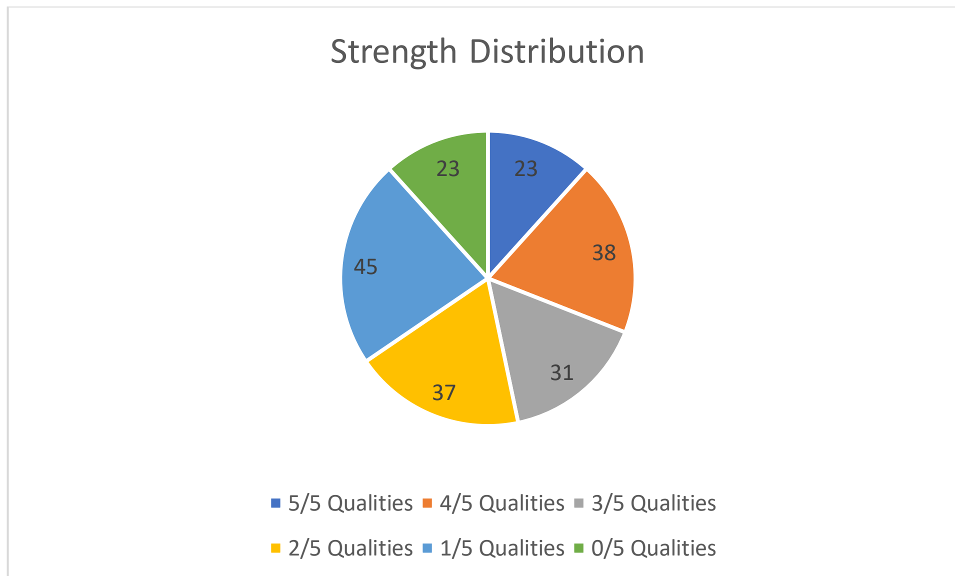



*Figure 12 - Qualities per Metric*

*Figure 13* below, shows the distribution of strong and weak metrics within each aspect

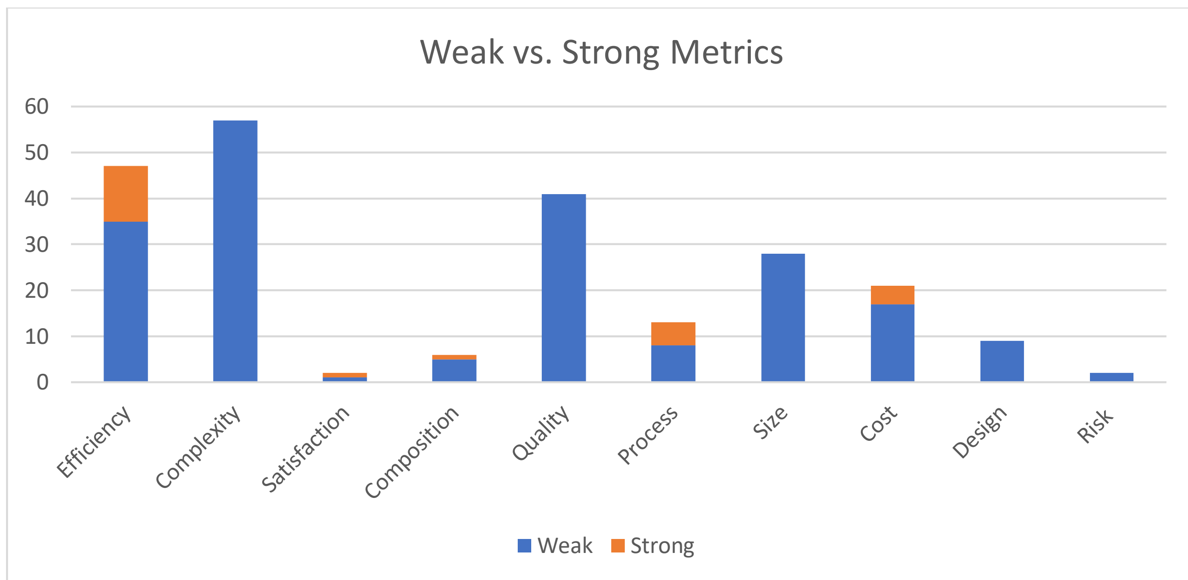



*Figure 13 - Weak versus strong metrics*

### 7.7.4 Quality Distribution

*Figure 14* shows the distribution of metrics within the quality criteria.

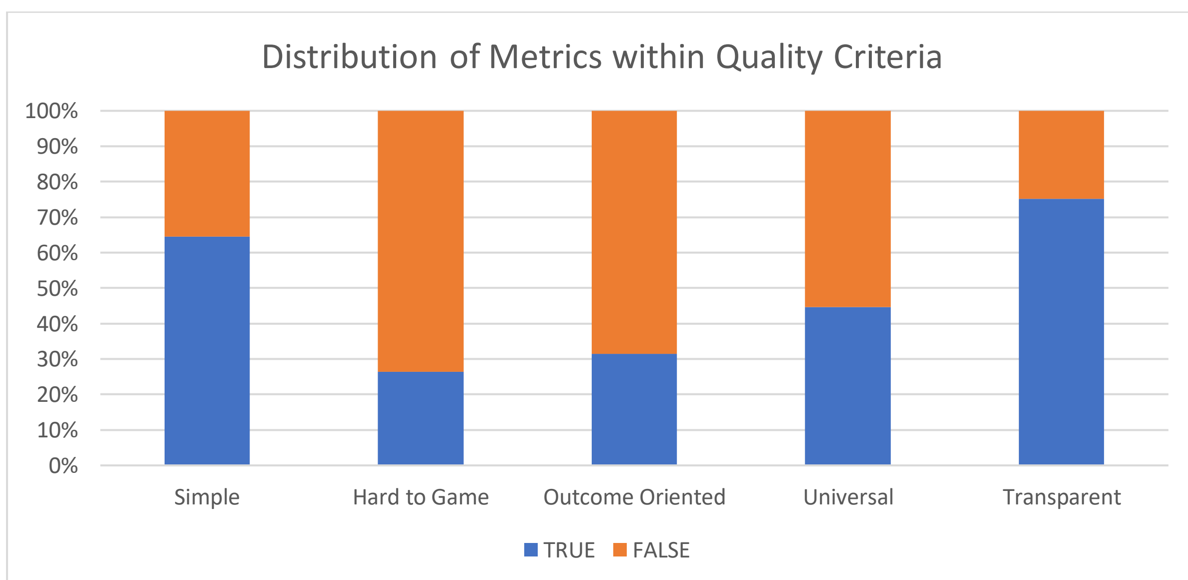



*Figure 14 - Distribution of metrics within quality criteria*

### 7.7.5 Paper Distribution

*Figure 15* then shows the amount of metrics that each paper mentions, regardless of whether they are considered to be weak or strong.

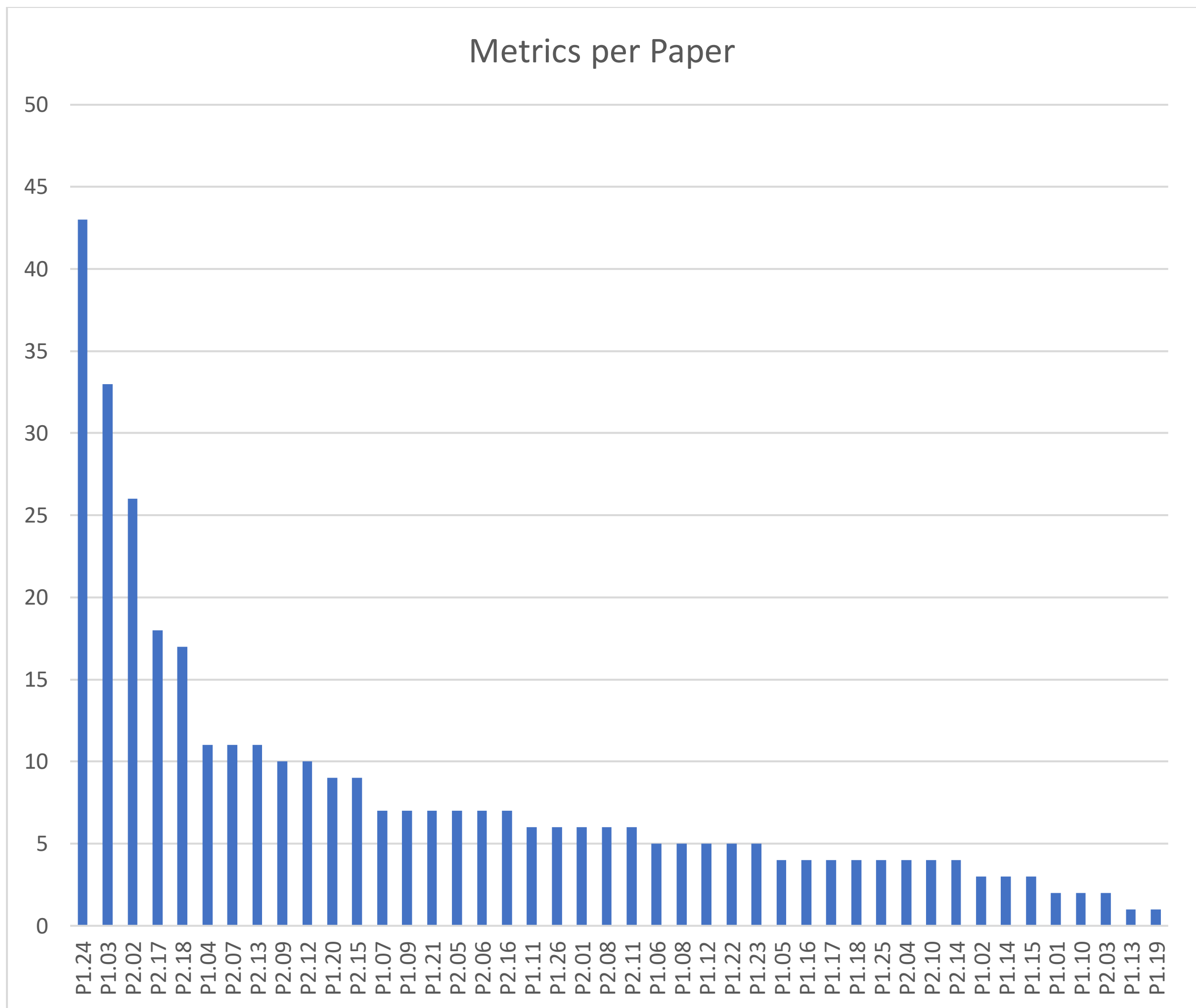



*Figure 15 - Metrics per paper*

# 8. Team Performance Model

In this section, we will introduce a new model for assessing team performance, based on the concepts discovered in the structured literature review, the discussions with experts, and the systematic mapping of their results. This model assesses the performance of a team along four different axes, being *process*, *people*, *technical* and *product*. These perspectives were derived from a final encoding pass over the aspects of software development. This encoding has yielded an additional perspective, called *enterprise*, which focusses on metrics aimed at measuring how well a whole organization is performing. This perspective is left out, however, of the model for *team* performance.

Each of these perspectives has a single key metric that adheres to the SHOUT model of metric strength, and is thus completely outcome-oriented. Consequently, the resulting measurements tell an individual team whether or not they are performing well on an individual perspective, but do not tell us anything about how to improve it. Additional metrics are required to provide a team with the necessary pulls and levers to actively navigate towards becoming a truly high-performance team. This is, however, part of our future research as indicated in section 11.2.

The rest of this chapter introduces the set of strong candidate metrics in section 8.1, and each of the perspectives and its key metric in more detail in section 8.2, after which we will outline the predicted input correlation between the four key metrics in section 8.3.

## 8.1 Candidate Metrics

First, *figure 16* shows the set of metrics that were considered *strong* during the evaluation with experts, adhering to all five SHOUT qualities for metric strength, alongside the aspects of the software development process which they target. Note that aspects that do not have any metrics that can be considered *strong*, are left out of this overview.

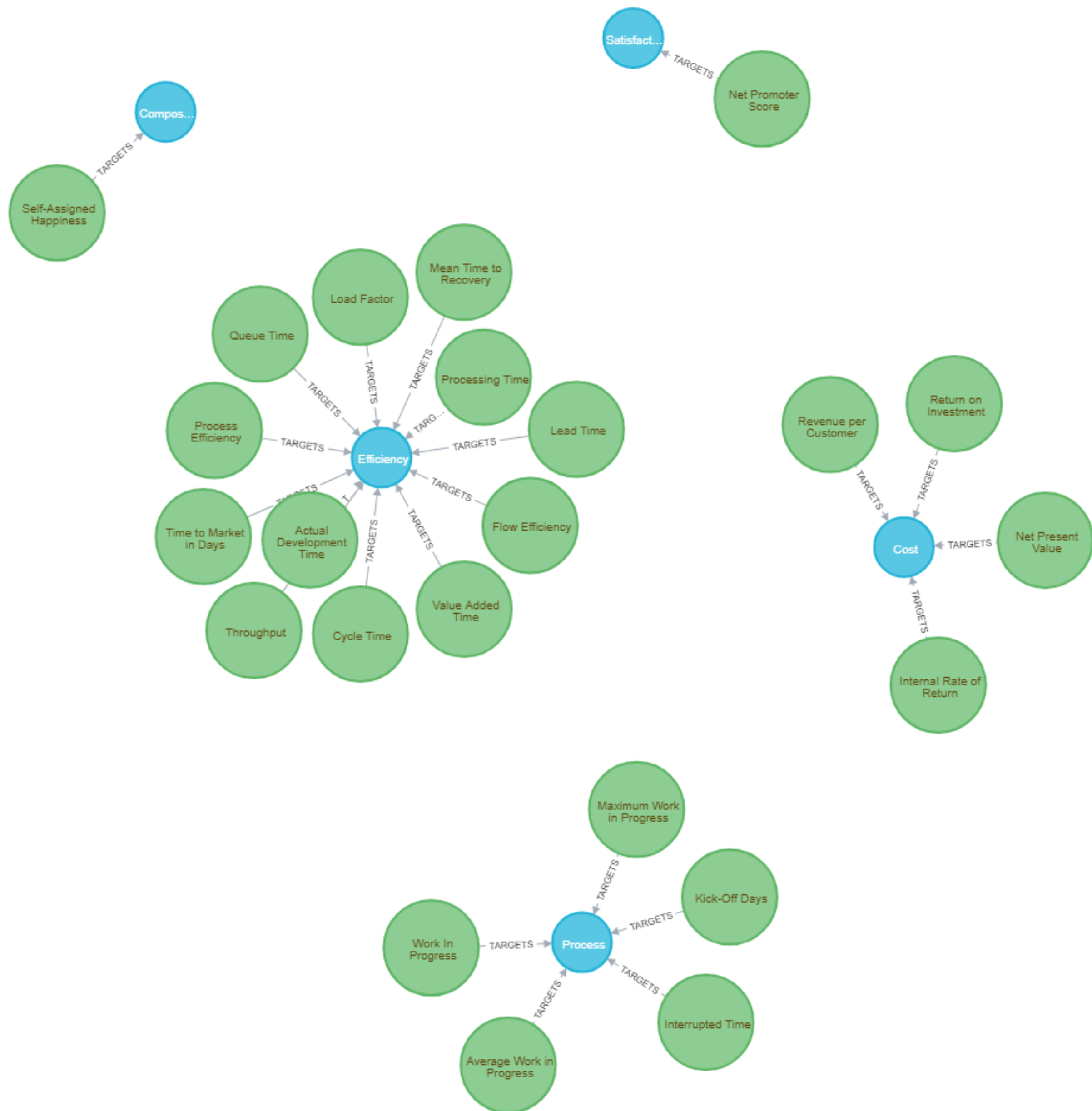



*Figure 16 - SHOUT metrics per aspect*

# 8.2 Perspectives

## 8.2.1 Process

According to Lean Manufacturing, the best manufacturing processes are optimized to reduce waste. In our team performance model, we state that a team's process is performant when it maximizes added-value, while minimizing wasted resources. The *strong* metric of *Process Efficiency*, introduced

in section 4.2.5, measures the percentage of total time spent adding value, and is used as the key metric for the *process* perspective on team performance.

### 8.2.2 People

In our model of team performance, we hold true the axiom that *the members of a team need to feel good about themselves and their company in order to become a high performance team.* The *Employee Happiness* metric, introduced in section 3.4.14.17, measures this sense of purpose, belonging and satisfaction that the experts believe is a necessary ingredient to high performance, and is used as the key metric for the *people* perspective team performance.

### 8.2.3 Technical

High technical performance allows a team to translate concepts into profitable products and services in minimal time. This maximization of speed, alongside the minimization of required effort, is perfectly encapsulated in the *Small Correct Change Into Production* metric introduced in section 4.2.4, and is thus used as the key metric for the *technical* perspective on team performance.

### 8.2.4 Product

Doing the right thing is equally important as (if not more important than) doing the thing right. High performance in the *product* perspective means maximizing the value in the eyes of the customers. The *Net Promoter Score* metric, introduced in section 3.4.14.11, measures how many more people love the product or service you've created, than the amount of people that hate it, and is used as the key metric for the *product* perspective on team performance.

## 8.3 Input Correlation

The key metrics introduced in the previous sections were chosen, not solely because they encapsulate their respective high-performance aspects closely, but also because their formulae share little to no input data-points. This is advantageous because this helps to isolate the cause-and-effect relationship between an organizations attempt to improve, and the difference in their measurement outcomes.

*Figure 17* below shows the four key metrics alongside the identified input data-points, and shows little interconnectivity. The only shared input data-point is the timestamp at which a particular work-item is finished, which is used by both the *Process Efficiency* metric, and the *Small Correct Change Into Production* metric. However, the input data-point represents a very dissimilar concept in each of these metrics. In *Process Efficiency*, the *finished* timestamp marks the end of the process, and concludes the period of examination for that particular work-item, while in *Small Correct Change Into Production*, it marks the start of that process.

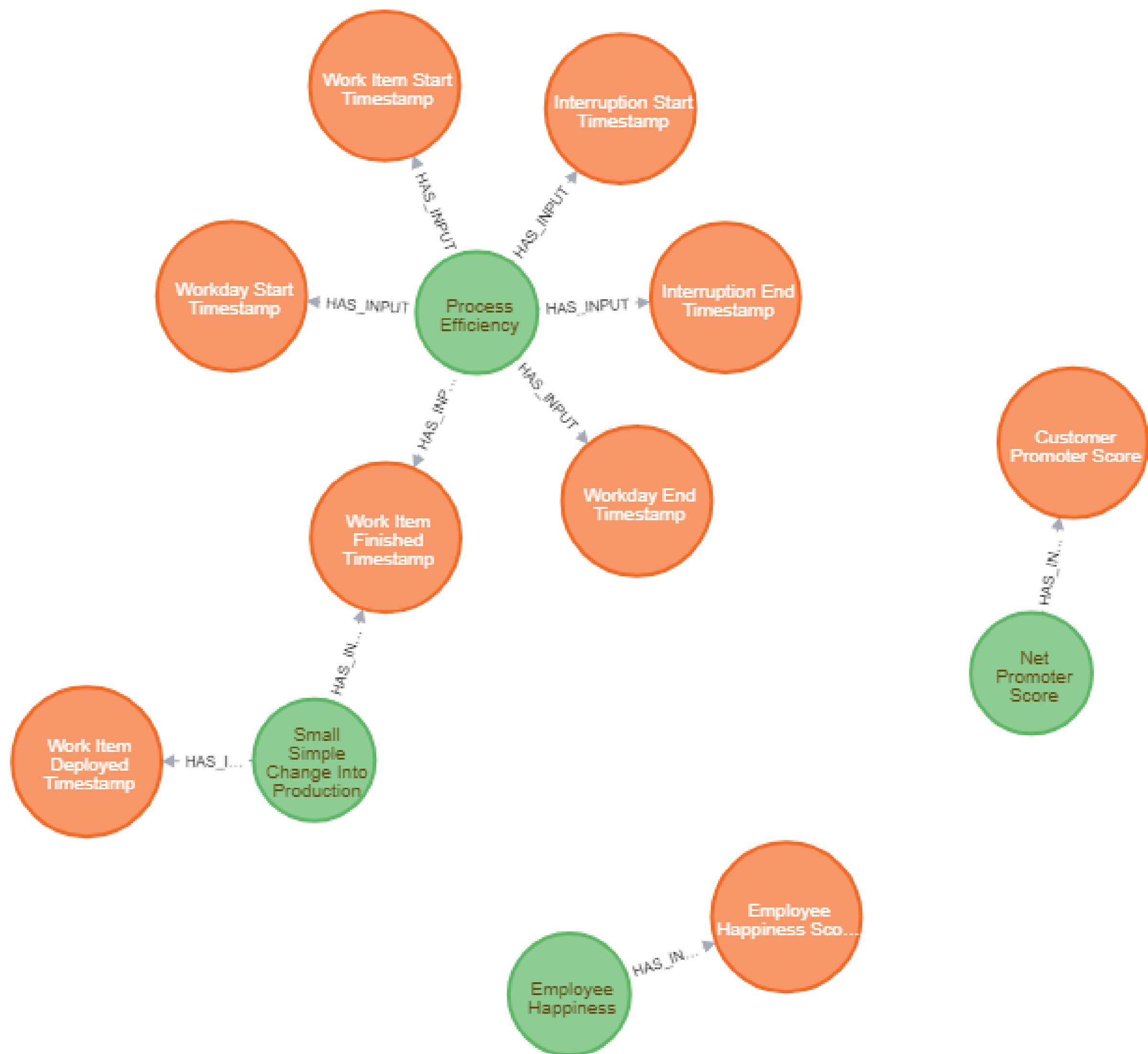



*Figure 17 - Key metrics and their input data-points*

# 8.4 Tooling

A web-based measuring tool has been created to facilitate the measurement of each of the four key metrics. In this section, we will quickly go over the capabilities of this tooling set. The tooling has been made available through https://www.diamondagile.net/la.

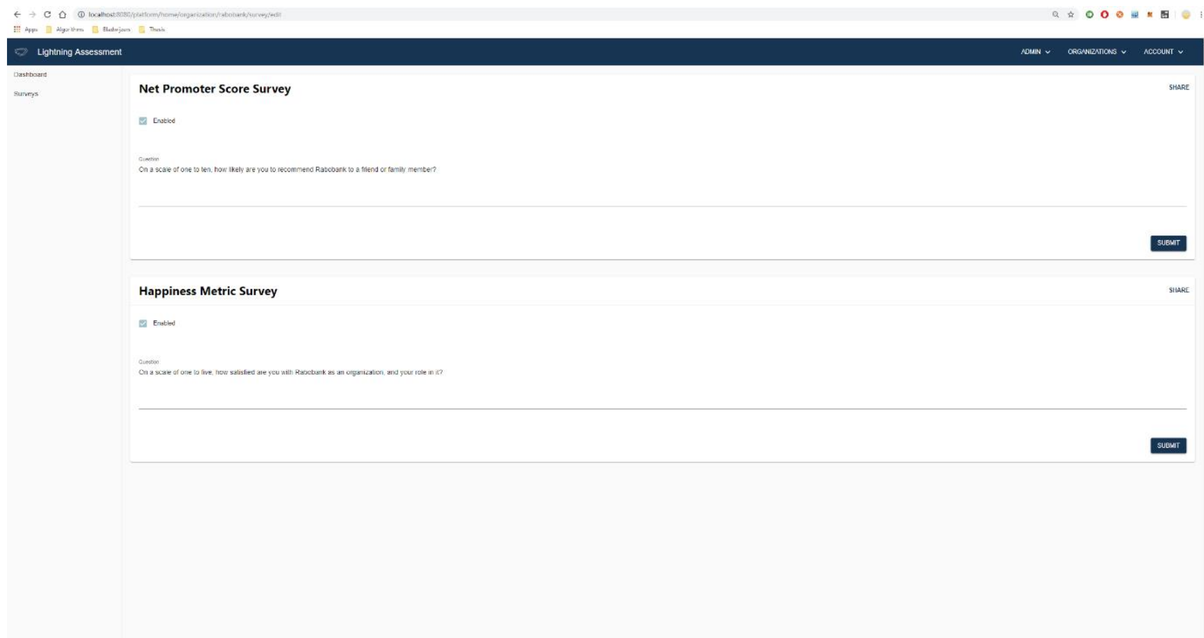



*Image 1 - Survey Maintenance*

*Image 1* above shows how an organization can create an account and set up customer-facing surveys for the *Net Promoter Score* and *Employee Happiness Score*. Here, the user can specify the question that is presented to the customer, as well as whether the survey is currently enabled or not. The resulting survey pages are shown below in *image 2* and *3*.

*Image 2 - Happiness Metric survey for Rabobank*

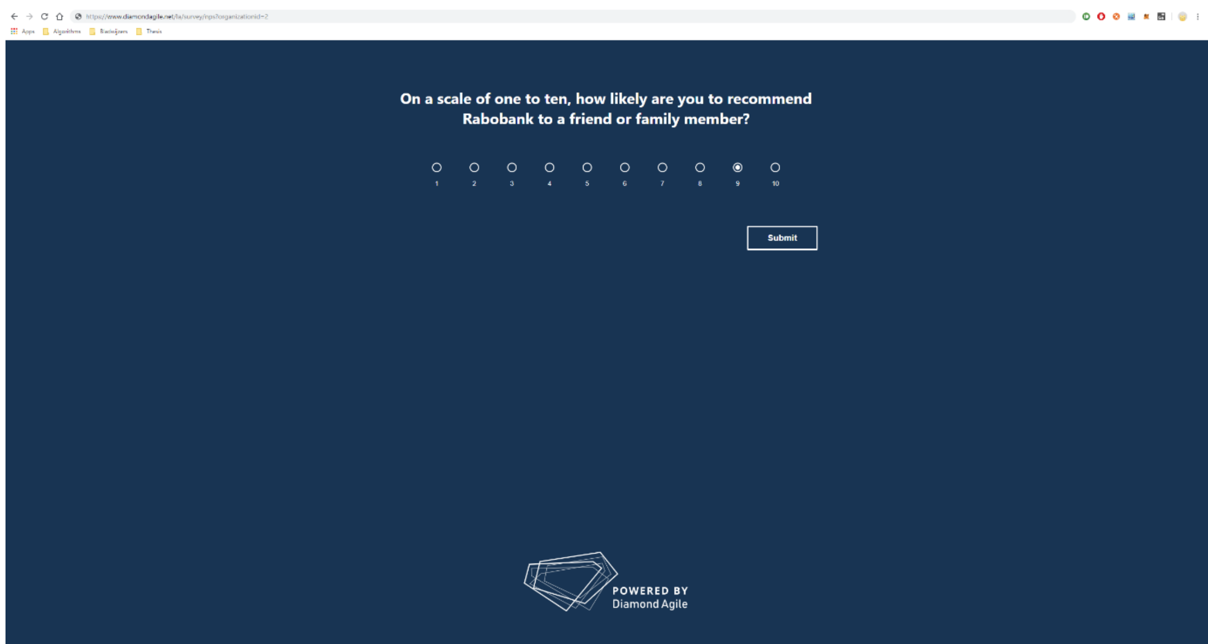



*Image 3 – Net Promoter Score survey for Rabobank*

At the same time, the platform allows for and details the submission of bulk uploaded data on stories, meetings, happiness scores and promoter scores. *Image 4* below shows the detailing of the stories bulk upload, with its required JSON format and an example input file.

Lightning Assessment

ADMIN ORGANIZATIONS ACCOUNT

**Explanation**

The stories data is uploaded through a .json file, containing a collection of story objects. For each story, a start and end date and time should be present, from which Process Efficiency and Simple Correct Change Into Production can be calculated. Below you will find a format definition which can be used to validate the correctness of your input file, as well as an example of such an input file on which you can base your data collecton.

**Format**


```
{
  "definitions": {},
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://example.com/root.json",
  "type": "array",
  "title": "The Root Schema",
  "items": {
    "$id": "#/items",
    "type": "object",
    "title": "The Items Schema",
    "required": [
      "interval"
    ],
    "properties": {
      "interval": {
        "$id": "#/items/properties/interval",
        "type": "object",
        "title": "The Interval Schema",
        "required": [
          "timestampStart",
          "timestampEnd"
        ],
        "properties": {
          "timestampStart": {
            "$id": "#/items/properties/interval/properties/timestampStart",
            "type": "string",
            "title": "The Timestampstart Schema",
            "default": "",
            "examples": [
              "2019-05-01T09:00:00.000"
            ],
            "pattern": "^(.*)$"
          },
          "timestampEnd": {
            "$id": "#/items/properties/interval/properties/timestampEnd",
            "type": "string",
            "title": "The Timestampend Schema",
            "default": "",
            "examples": [
              "2019-05-01T17:00:00.000"
            ],
            "pattern": "^(.*)$"
          }
        }
      }
    }
  }
}
```


**Example**


```
[
        {
                "interval": {
                        "timestampStart": "2019-04-01T09:00:00.000",
                        "timestampEnd": "2019-04-01T15:00:00.000"
                }
        },
        {
                "interval": {
                        "timestampStart": "2019-04-02T12:00:00.000",
                        "timestampEnd": "2019-04-02T16:00:00.000"
                }
        }
]
```


*Image 4 - Bulk upload format details*

Finally, these measurements can be used to generate a dashboard containing indications on how well the organization or team is doing on each of the key metrics. *Image 5* shows a generated performance dashboard with fictitious, generated data. You will notice a fifth perspective here, called *Enterprise*. This perspective was added for the context of an entire organization, as opposed to a single team, and is thus not included in this thesis on *team* performance, while it is available in the online tooling module. The dashboard shows, for each of the five perspectives, whether the organization or team is performing good (white), bad (orange) or mediocre (yellow).

*Image 15 - Generated dashboard*

Note that the tooling module is a work-in-progress, and has focussed primarily on authentication, authorization, API development and initial user experience for the duration of this thesis. In future work, the generated dashboard will require a substantial UX overhaul to fit in with the design of the web UI, and additional features will have to be added for generating dashboards for a particular timespan or sprint. At the time of delivering this thesis, Rabobank is about to start onboarding some +- 70 teams onto the platform for initial use.

# 9. Validation

In this chapter, we will detail the execution of an initial and superficial validation of the newly devised model for team performance. The aim of this validation is to gauge the perceived clarity, relevance and completeness of the model among software development experts in the field. This validation was performed using a small *Google Forms* survey that was distributed via various online software development communities on *Reddit*, resulting in 34 answers from various professionals in the field of software development. These results are outlined below, and finally discussed at the end of this chapter.

## 9.1 What kind of role(s) do you have within your organization?

*Figure 18* below shows the distribution of their roles within their respective companies. Here, respondents could select multiple options, and with a total of 65 selections over 34 responses, each respondent selected an average of 1.91 roles.

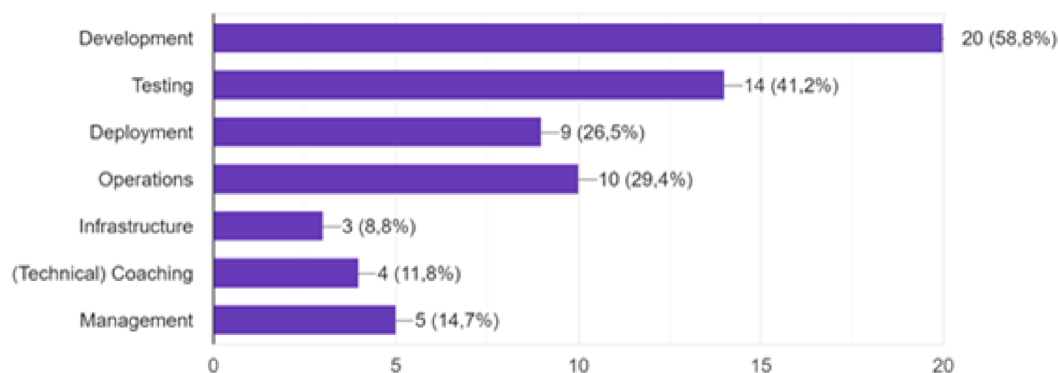



*Figure 18 - Respondent roles*

## 9.2 Does your organization measure the performance of your software development process in any way?

*Figure 19* below shows that in nearly 80% of the cases, the target organization measures the performance of the software development process in one way or the other. An additional 12 percent did not know, while only 8.8% signalled a definitive no. Here, respondents could only select a single answer.

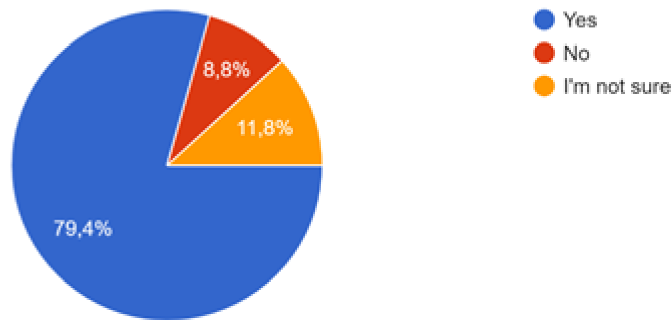



*Figure 19 - Percentage of respondents measuring software development performance*

## 9.3 Does your organization measure the performance of your software development performance?

For the respondents that indicated that their software development process performance was measured, 96% indicated that the performance was measured on the level of their team. Individual, departmental and organizational measures were less common, with 38%, 76% and 34% respectively, as shown below in *figure 20*. Here, again, respondents could select multiple options, and with 64 responses over 34 respondents, each respondent selected an average of 1.88 options.

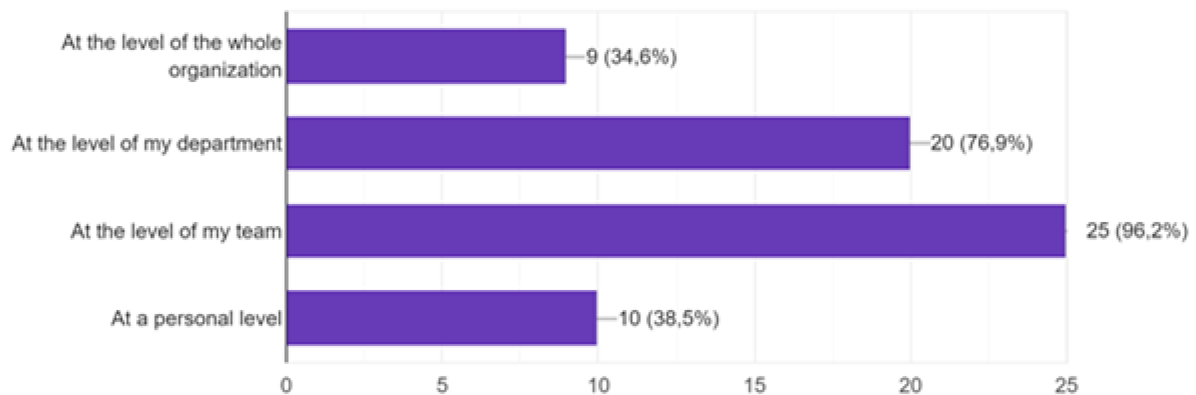



*Figure 20 - Measurement granularity*

## 9.4 In terms of a gain in efficiency, how much of an increase has the application of these measurements brought you?

In terms of the fruits of their labour, over half of all respondents signalled not knowing how much more performant their software development process has become as a result of using the

measurements, as shown below in *figure 21*. For those who could give an indication, it seems like a 10% to 100% increase in performance was most prevalent. Here, respondents could only select a single option, and the total amount of responses is 27. The discrepancy between 27 and 34 is due to the fact that those who signalled not knowing whether or not their performance is measured, or signalled a definitive no (7 people in total), were not asked this question.

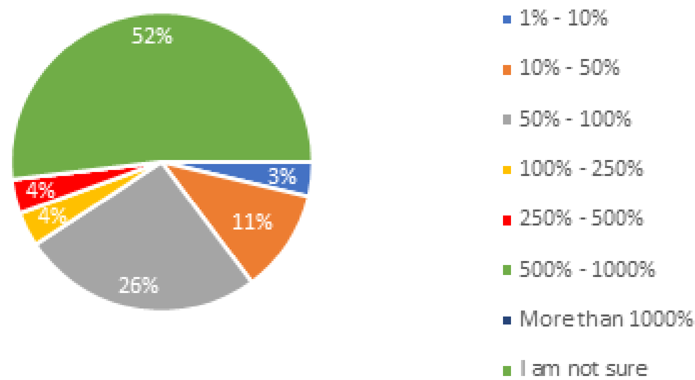



*Figure 21 - Measurement application advantages*

## 9.5 What software development method does your team use?

The overwhelming amount of respondents used Scrum or Kanban as their software development method, with over 80% of the results combined, as shown below in *figure 22*. Here, again, the total amount of responses equals 27, as respondents could only select a single software development method.

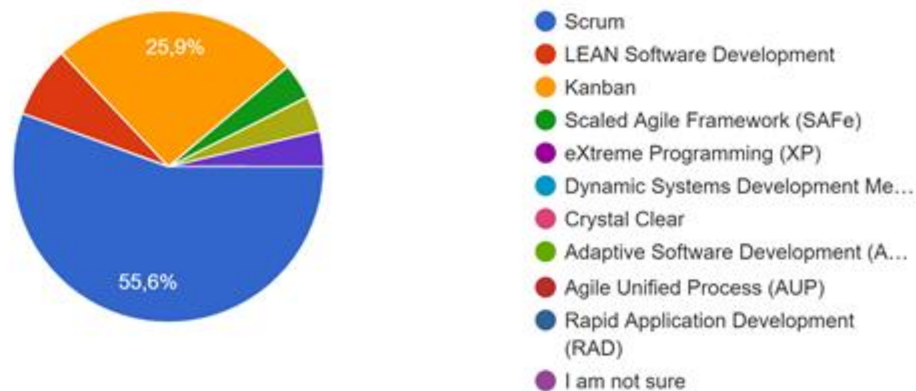



*Figure 22 - Software development methods*

## 9.6 What software development metrics were used in their software development process?

When asked "What software development metrics were used in their software development process", the answers denoted below in *table 12* were provided by the respondents. This question was asked to see if the structured literature review, as well as the expert inquiry, had failed to identify other industry-used metrics. These responses, however, did not identify any new metrics or concepts.

The concepts that have been mentioned have been encoded to show story point estimates, velocity, metrics related to cycle-times, work-in-progress, test metrics, and others. In total, story point estimates were mentioned 22 out of 26 times. This is not surprising, as 22 out of 27 people signalled using Scrum or Kanban. Similarly, 16 out of 26 responses mentioned story point velocity. Metrics that relate to cycle-times, such as lead-, queue-, value-added-, and interrupted-times, were mentioned in 14 out of 26 responses. Work-in-progress metrics, or ones that limit them, were mentioned in 12 out of 26 responses, while testing related metrics were mentioned in only three responses. Finally, metrics such as the *Net Promoter Score*, *Overtime per Iteration*, *On-Time Delivery*, *Burndown Rate*, *Discovered Effort*, *Function Points*, *Targeted Value Increase*, and *Discarded Stories were only mentioned once.*

| |
|---|
| Story points and velocity. |
| Velocity, story point and hours spent |
| Story points, story point velocity, burndown rate, work in progress and cycle time |

| Story points, and velocity per sprint and hours. |
|---|
| I am not sure about all of them but I suspect just the regular scrum stuff is used by our teams, like story point estimates and velocity and time spent per story etc. |
| Story points and development time |
| Story points and velocity, work in progress, net promoter scores, on-time delivery. |
| Test coverage, test growth as opposed to source code growth, and standard Scrum/Kanban things like story points and work-in-progress. |
| The usual Kanban things like flow, amount of stories in a swimlane, size estimates, etc. |
| Things like cycle time, value added time, work in progress, time spent in meetings, overtime per iteration, story points and velocity. |
| We measure flow, the amount of stories that are in each swimlane simultaneously, the average cycle time for stories, and we do effort estimations in story point. |
| Story points. |
| We measure story points and story point velocity, the amount of work in progress, how long each story is in a particular state, how long it's on the backlog before its included in a sprint, how many sprints it takes on average to implement a story, and how often we change story point estimates during a sprint (discovered effort). |
| We mainly use story points, but for some legacy projects we also still employ function points. |
| Story points. |
| We restrict the amount of work in progress, and estimate the effort required in terms of story points. |
| We measure code coverage, the growth of tests versus the code base, and the required time to run the entire test base. |
| Work in progress, story points, velocity, average time spent in swimlane. |
| Story points, velocity. |
| We measure how long a story is in a particular state, how many stories were in a particular state at the same time, how often a state was at "full-capacity", the average cycle-time for a story, how long a story has had to wait before being picked up, and how many stories are thrown away. |
| Unit test coverage, how many of the builds during the day are successful, we measure our velocity and the growth in our velocity. We also have stuff like PMD for quality assessments. |
| Story points and the sprint velocity. |
| The amount of work that is in progress at the same time, the story points and our velocity. |
| Story point velocity and hours spent developing a story |
| We measure velocity, story points, maximize swimlane capacity, cycle time and hours spent on the level of teams. |
| Story point estimates, hours spent, remaining effort and velocity. |

*Table 12 - Used software development metrics*

## 9.7 How would you assess the strength of a particular software development metric?

Similarly, the question "How would you assess the strength of a particular software development metric?" yielded the following responses, denoted below in *table 13*. This question was asked to identify additional qualities of strong metrics that the experts might have overlooked. The quality of *effectiveness* was mentioned an overwhelming amount of times, with 17 out of 27 responses mentioning the quality in one form or another. 4 out of 27 respondents looked for adoption of the

metric in the industry, while only 2 mentioned the impact that measuring it would have on the performance of the team under investigation. Similarly, only 2 respondents mentioned simplicity or intuitiveness of the metric. Additionally, 3 respondents mentioned validity as a primary quality for metric strength. Finally, 3 respondents mention the quality of usefulness or applicability.

| |
|---|
| We would expect a strong metric to be widely adopted and to have proven itself in the industry. |
| Proven effectiveness. |
| How well it works in practice, so if it yields results. |
| How ubiquitous it is in the field. |
| How well they work. |
| How well the measurements correspond with the reality. |
| Simplicity and effectiveness. |
| I feel like you should be able to notice growth fairly quickly once adopting a metric. If it does not help soon, you should let it be. |
| How applicable the results are to the problems we face. |
| If big tech companies are using it. |
| How much impact it has on the performance of a team. |
| How intuitive it is and how accurate the results are. |
| The impact on the process. |
| A strong metric should have meaningful, measurable impact on your performance. If you cannot measure that the adoption of the metric has brought you an increase of performance, the metric is weak; or at least it's weak in your context. |
| Theoretical validity. |
| How much it has improved our process. |
| I would not know but I would usually just do whatever other successful teams are doing. |
| If it's academically validated or has been shown in industry to be beneficial. |
| We look at how well we can use the results to improve our process. |
| How good the results are from using it. |
| I think the strength of a metric comes from the benefits it provides for the process. If it does not bring enough benefit, it's not a very strong metric. |
| How effective it is. |
| Research. |
| How well they work when applied to your process. |
| How useful it is for management and the teams. |
| Usefulness to upper management. |
| How much better the process is when they’re used. |

*Table 13 - Metric strength assessment qualities*

## 9.8 Relevance

Figure *23* below shows the individual responses on the perceived relevance of each of the five qualities.

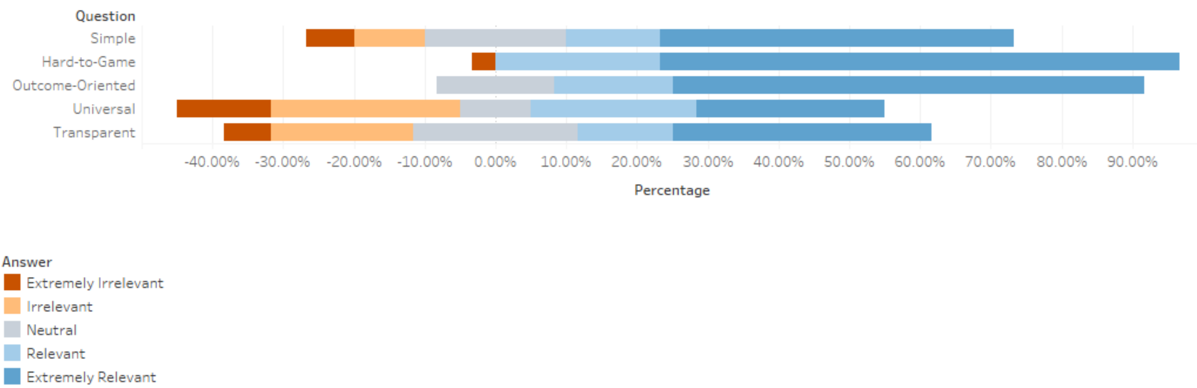



*Figure 23 - Relevance of SHOUT qualities*

## 9.9 Clarity

In terms of clarity of definition, all of the qualities were received well. *Figure 24*, shown below, shows the individual results.

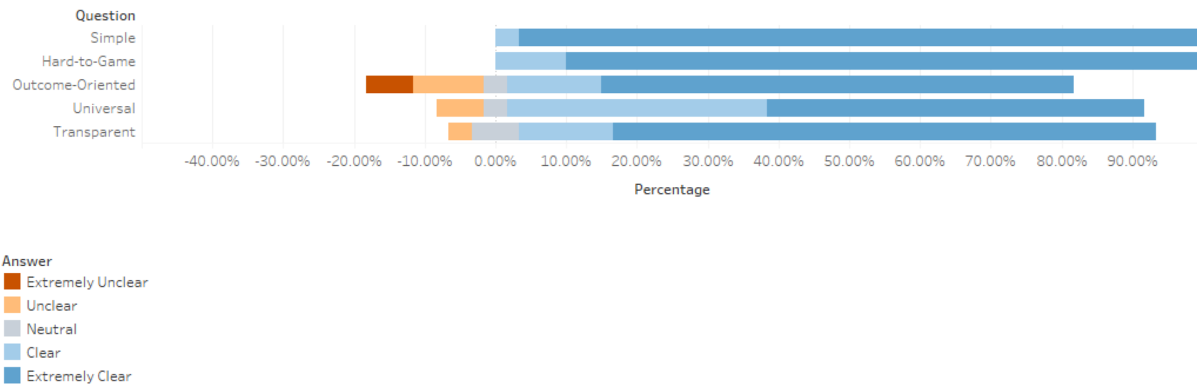



*Figure 24 - Clarity of SHOUT quality definitions*

## 9.10 Completeness

*Figure 25*, shown below, shows how well the respondents think the five qualities encompass everything that a *strong* metric should have.

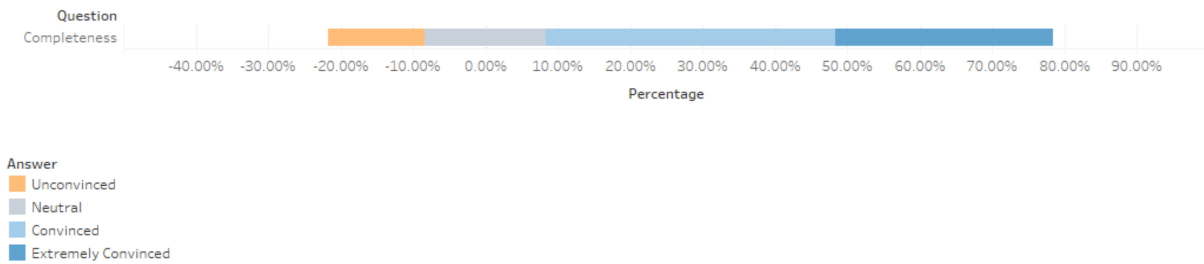



*Figure 25 - Perceived completeness*

## 9.11 Summary

With a total of 34 respondents, the set is too small to draw any significant conclusions from our preliminary validation. Additionally, the participants were drafted from online software development communities on Reddit, and were subsequently not validated in terms of whether they actually were software development professionals or not. While this leads to a validation that might not yield any conclusive results on the model's validity, it is an interesting first step towards gauging the relevance, clarity and completeness of the model.

It is surprising to see that a significant proportion of respondents (21%) signalled not knowing whether they measure performance, or definitively stating that they don't measure performance at all. Similarly, over half of the respondents (51%) who *did* measure performance, had no idea about what they had factually gained in doing so.

The open questions did not yield any additional software development metrics, which yields us some confidence in having a fairly exhaustive set - especially considering the largest systematic literature review that we found on the subject had only discovered 43 software development metrics, as opposed to our set of 197 metrics. The second open question did, however, mention the quality of *effectiveness* an overwhelming amount of times. At this point in time, it is difficult to say with certainty whether *effectiveness*, as meant by the respondents, corresponds with the *outcome-oriented* quality in the SHOUT model for metric strength. While there are definite similarities between the two, we can only be certain after a proper discussion with the advocates of *effectiveness*.

In terms of relevance, the model was received fairly well. The qualities of *simple*, *hard-to-game* and *outcome-oriented* showed significant positive results, with median scores of 4.5, 5 and 5 respectively. The *universal* and *transparent* qualities, however, were only received as fairly relevant, with median

scores of 3.5. In terms of *Net Promoter* Scores, the qualities received a score of 46%, 93%, 83%, 10% and 23% respectively, which seems to support the same conclusion. Similarly, in terms of clarity, the model was received extremely well. Here, all qualities have a median score of 5, and *Net Promoter Scores* of 100%, 100%, 63%, 83% and 86% respectively. Finally, the model was perceived as fairly complete as well, with a median score of 4, and a *Net Promoter Score* of 56%.

# 10. Discussion

In this chapter, we discuss the findings of our study. We start with a discussion of the structured literature review results, the expert inquiry, and the systematic mapping, followed by a discussion of the SHOUT model for metric strength and the model for team performance, and we finish the chapter by listing the threats to the validity of these results.

## 10.1 Metrics

### 10.1.1 Structured Literature Review

The structured literature review yielded a large set of metrics, hinting at a large body of knowledge for software development metrics. The collected work, spanning more than 40 individual papers on the subject and over 1000 potential candidates, shows a healthy distribution over venues and publishers, giving us no reason to suspect any form of venue or publisher bias.

The study found 197 individual metrics, which is more than 4.5 times as many as the largest literature review on the subject that we found (P1.24), giving us adequate reason to believe that our current work has added significant value to the field of measuring software development processes, by the results of the structured literature review alone. This seems to have been a necessary endeavour, seeing as the resulting set of keywords hint at an industry that lacks a clearly defined lexicon of standardized terms, with lots of synonyms and very little overlap between papers. Similarly, when looking at the set of authors working on the included work, we see that they rarely publish more than one paper on the subject, with the most prevalent expert being Jeff Sutherland at three included papers. This also hints at a field that lacks well-known and prominent experts on the subject. Given the fact, however, that our inclusion criteria stated that a paper should mention a *new*, *previously unmentioned* software development metric, we cannot be all too sure about the latter two conclusions.

Surprisingly, no *golden age* of software development metric research can be identified, as the field has seen continuous and consistent attention since its inception. The distribution of metric mentions does, however, show a focus of research on complexity, quality and efficiency metrics, with 146 metrics targeting just these three aspects of the software development process. Similarly, a significant amount of metrics seem to have input data-points coming from work-items and their lifecycle, as well the source code, with 52 out of 118 inputs originating from just these three input categories.

In terms of metric strength, according to the newly introduced SHOUT model of metric strength, it is surprising to see that five out of ten aspects failed to yield *any* strong metrics. Even more surprising is the fact that complexity and quality are among them, while 98 such metrics were identified. While we expected this to be because they were not classified as *universal* (and thus only adhering to a SHOT model of metric strength), we found that most often, they were not classified as *outcome-oriented* instead. This is not surprising, as code quality and complexity metrics can be excellent tools to maintain a high level of maintainability and clarity, but optimizing them does not necessarily correlate with increased business value. Similarly, such metrics can fairly easily be gamed, with various adverse effects. The *lines of code per method* metric, for example, can be kept artificially low by limiting it to one per method, but this might severely hurt readability and maintainability. The efficiency aspect, however, has yielded 12 *strong* metrics, most of which come from LEAN software development or manufacturing. Most of these metrics target various aspects of the life-cycle of a work-item (e.g. lead-time, queue-time, cycle-time, interrupted-time, and value-added-time). Similarly, the *Work-in-Progress* metrics that were encountered in the process aspect of software development, also have their roots in LEAN manufacturing or software development.

According to the distribution of qualities over metrics, the *hard-to-game* quality appears to be the hardest quality to inhibit for a metric, with just 26.3% of the encountered metrics adhering to it. Similarly, only 31.4% of the encountered metrics have shown to be *outcome-oriented*, making it the second hardest quality to adhere to. Finally, just 23 out of 197 metrics can be considered *strong*, being only 11.6% of the entire set of encountered software development metrics. This hints at the necessity of an accurate model for metric strength, as well as the need to keep quality in mind when devising new software development metrics. While the review has yielded a large set of metrics, it has yielded no model for determining metric strength or quality. The goal-question-metric model came closest, but focusses on what makes a metric good for a particular organization's *context* instead. A model for metric strength is thus a welcome addition to the field of software development metrics.

### 10.1.2 Expert Inquiry

The expert inquiry was done with a small group of experts, yet the group consisted of very prominent and prevalent experts in the field, with lots of experience and expertise between them. We found that it was surprisingly easy for a small group of experts to unanimously and quickly determine whether or not a metric could be considered *strong* or not, even without the SHOUT model for metric strength in place.

The inquiry yielded six additional metrics that were not identified through the structured literature review and its snowballing process. It is interesting to note that all six metrics could be considered *simple*, *hard-to-game*, *outcome-oriented* and *universal*. Now that their definitions, as well as their data-points have been clearly and unambiguously defined in this work, they can also be considered to be *transparent*. This means that all of the metrics retrieved from the expert inquiry can now be considered *strong* metrics, and can now be used by software development teams to determine some aspects of their performance.

*Context Concurrency, Priority Focus* and *Degree of Swarming* show clear similarities with Kanban, where the amount of work-in-progress is limited in order to prevent an abundance of context switching and to stimulate a focus on the highest current priority. Additionally, *Degree of* Swarming shows similarities with the rise of *pair programming,* and the move away from the stereotypical independent and anti-social software developer. *Small Correct Change Into Production* and *Innovation Income* can both be considered as very simple, fast indicators of general technical and organizational performance, while in-depth analysis would require other, more complex and time-consuming metrics. Finally, it is interesting to note that *Process Efficiency* is a strong metric, while all of its inputs can *also* be considered *strong*, hinting at a very promising application that will need to be validated in future empirical research.

### 10.1.3 Systematic Mapping

The systematic mapping has proven to be very helpful in analysing and interpreting the results of the structured literature review and the expert inquiry. While the axial-encoding would most likely have yielded different results if performed by other researchers, we feel like it has fulfilled its purpose adequately. At the same time, however, we feel very strongly that potentially many more patterns and insights can be extracted from the systematic mapping, or with a potentially different axial-encodings. For this reason, we have decided to publish the data set in its entirety on https://www.silvester-consultancy.com/portfolio/thesis/download/systematic-mapping.

## 10.2 Models

### 10.2.1 Model for Metric Strength

The SHOUT model for metric strength was received fairly well by the participants of the small validation survey. In their responses, the participants signalled the definitions of the qualities to be very clear, with high median values, just as the relevance of these qualities. In the end, the model was

thought to reasonably encompass every quality that a metric should have in order to be considered *strong*, with a median score of 4 and a *Net Promoter Score* of 56%.

The model does, however, need a larger-scale validation in the industry, with a larger set of verified participants, whereas the current validation was just a small probe into the general reception of the model.

### 10.2.2 Model for Team Performance

The model for team performance shows very little correlation based on shared input data-points, with only the timestamp at which a work-item has finished being used for both *Small Simple Change Into Production* and *Process Efficiency*. As stated in chapter 8, however, the input data-point is used for widely different things, and represents different concepts in both metrics. The resulting model has, however, not been validated in this study, and so reception and performance of the model is difficult to gauge.

## 10.3 Threats to Validity

In this section, we will analyse the apparent threats to the validity of our research. In their work, Zhou et al. (2016) identified various common threats to the validity of systematic literature reviews in the field of software engineering. The following section details the common threats to validity that are applicable to our context, and mentions the considerations that we have adhered to in order to ensure the validity of our work to the largest feasible extent.

#### *10.3.1 Non-specification of settings*

In order to circumvent the threat of *non-specification of settings*, we have ensured to properly document and mention the venues, search strings, and query settings with which the searches were performed. Due to the size and scope of the snowballing process, and the limited resources available to us in this study, we have had to make some concessions regarding the reproducibility of the snowballing process, which has in turn lowered the validity of our results slightly.

#### *10.3.2 Inappropriate search methods*

Subsequently, the threat of *inappropriate search methods* has been circumvented by performing the searches automatically, yet following the automated search with a manual snowballing procedure, in order to ensure that we did not miss some large part of the body of knowledge.

#### *10.3.3 Incomprehensive venues or databases*

The *incomprehensive venues or databases* threat, states that the review might miss relevant work due to not including important resource databases. To circumvent this threat to some extent, the search, and subsequent snowballing, has been duplicated on multiple academic search engines. However, due to time and resource constraints, this effort duplication was limited to only two of the most prevalent academic search engines available today, being Google Scholar and Scopus. Additionally, we have made sure to accurately and appropriately document our inclusion and exclusion criteria, as to ensure that the work being performed is valid and reproducible.

#### *10.3.4 Culture bias*

In order to circumvent the *culture bias* threat, we have ensured to include any work that meets our inclusion criteria, regardless of apparent author nationality or cultural heritage. Because some of this bias may be unconsciously exerted, we have attempted not to inspect or make deductions about the cultural background of a paper's author(s), until we had determined whether or not the paper meets our inclusion criteria. Once inclusion criteria were met, the work could no longer be excluded from the results based on nationality or cultural heritage.

#### *10.3.5 Hidden work*

While we have set out to include all of the relevant work, in some cases this was simply not possible due to paywall protection. We have attempted to retrieve such papers using the University of Utrecht's proxies, and contacting the authors directly if those proxies could not successfully retrieve the work either. However, due to the limited availability of time and resources, work of unresponsive or unwilling authors has ultimately not been included, thus marginally reducing the validity of our results.

#### *10.3.6 Primary study duplication*

In order to circumvent the threat of *primary study duplication*, papers that were included in more than one result set, or published in more than one journal, have been manually identified and removed. Additionally, a prevention module in the systematic mapping application has actively ensured that no duplicate work could have been inserted into the systematic mapping.

#### *10.3.7 Publication bias*

The *publication bias*, which states that positive results are more likely to be published than negative results, as well as the fact the reproduction papers are less likely to be published than new work, was not circumvented in this study. This is largely due to the fact that academia has only recently

acknowledged this problem and brought forward solutions in order to combat it (for instance by launching journals that actively aim to publish papers regardless of whether their results were successful or not, or whether the paper is a reproduction or not). Because our systematic review ought to be timespan-agnostic, we have concluded that eliminating this threat would do substantially more harm than good to the validity of our results. Thus, we have still adhered to the inclusion criteria of work being peer-reviewed and published, instead of including grey/white work, or limiting our search to result- and type-agnostic journals.

#### *10.3.8 Subjective quality assessment and lack of expert evaluation*

The *subjective quality assessment* bias is a significant threat to the validity of our results. While you would prefer quality assessment to occur based on prior research, no such prior work was found to exist for every discovered metric. In order to ensure that the quality assessments exhibit the least amount of subjectivity, we have validated the resulting model by attempting to reach consensus within the focus-group of prevalent experts in the field. Using the same validation construct, the *lack of expert evaluation* threat is circumvented.

# 11. Conclusion

The strength of Agile software development has largely been acknowledged by academia and industry, but an accurate way of measuring the exact benefits of adopting Agile has yet to be uncovered. In this chapter, we conclude our attempt to develop a new model for measuring software development team performance, and describe its impact on the field. Here, we also attempt to answer our primary research question and its sub-questions, and also outline some limitations in the current work. Finally, we posit some potential future work, following the implications of the current work..

## 11.1 Research Questions

***11.1.1 Which software development metrics already exist today?***

In this study, we performed a structured literature review as to determine what software development metrics exist today, resulting in 191 software development metrics. In order to ensure that no metrics were overlooked, we performed an expert inquiry in which we asked prevalent experts in the field of software development whether they thought the resulting list was complete, resulting in an additional 6 metrics.

***11.1.2 What constitutes a strong software development metric?***

The results of this endeavour were structured in a systematic mapping, and discussed with the experts in order to determine what makes them strong or weak. From this discussion, a new model for metric strength was developed, identifying five qualities that a metric should possess in order to be considered *strong*. These qualities state that a *strong* metric should (a) be simple to explain and simple to measure, (b) be difficult to optimize without increasing business value (c) correlate strongly with increased business value when optimized, (d) be useable in multiple contexts, without confusing edge-cases, and (e) have an unambiguous and transparent definition of its data points, as well as how those data points are used in its calculations. We have dubbed these qualities *simple, hard-to-game, outcome-oriented, universal,* and *transparent* respectively, and together, these qualities spell the acronym SHOUT.

***11.1.3 What set of software development metrics is most suitable for measuring team performance?***

Finally, this model was used to identify strong metrics in the result set of the structured literature review and the expert inquiry. From this set of strong metrics, we have created a new model for

measuring software development team performance. This model is based on the *Process Efficiency*, *Employee Happiness, Net Promoter Score* and *Small Simple Change Into Production* metrics, targeting the *process*, *people, product* and *technical* perspectives of the software development process respectively. This model has not been validated in this study, but initial analysis have shown that little correlation between these metrics is to be expected, based on their shared input data-points.

#### *11.1.4 How can we measure the performance of a software development team?*

Finally, by answering our three sub-questions, we are able to answer our primary research question of how we can measure the performance of a software development team. The final answer to this question is thus to use *strong* software development metrics, utilizing *independent input-data-points* in order to isolate cause-and-effect relationships, while targeting *multiple aspects* of the software development process. In this thesis, we have presented a model for assessing the strength of a software development metric, as well as a model for measuring team performance, based on *strong* metrics, sharing little input data-points and targeting four different aspects of the process. These models can help organizations assess the performance of their software development teams. Finally, we have introduced automated tooling in order to help organizations measure these four key metrics.

## 11.2 Limitations

#### *11.2.1 Limited Google Scholar starting set*

There are several limitations in our execution of this research. First and foremost, we have had to make some concessions as to how thorough our manual search for candidate work could be. Here, we have limited the initial collection of candidate work from Google Scholar to just the first 10 results, instead of incorporating the whole result set. This may have, in the end, led to less valid results, due to not having exhausted the entire existing body of knowledge. However, as we have found more than 4.5 times as many metrics as the largest literature review we have found on the subject, we feel very confident that the extent to which these factors threaten the validity of our results is fairly minimal.

#### *11.2.2 Limiting inclusion criteria*

Similarly, our inclusion criteria of needing to mention a *new* software development metric, as opposed to just *any* software development metric, has a significant influence on the validity of our results. The possibility exists that we have missed a substantial portion of the existing body of knowledge, due to potential separate clusters that our practice may have missed due to this inclusion criteria. A reproduction study would be wise to broaden this inclusion criteria to mentioning *any* software

development metric, but we fear that this will substantially increase the effort required to properly perform the study.

#### *11.2.3 Initial focus on efficiency*

Additionally, we set out to perform this literature review with an initial focus on *efficiency* metrics. For this reason, the search queries that were executed on the Google Scholar and Scopus search engines, were deliberately biased to target software development metrics targeting *efficiency*. Only after having performed the searches, and having seen the amount and quality of the results, did we decide to register *all* software development metrics. This bias in search queries might have caused us to mis significant clusters of metrics in the body of knowledge on software development metrics.

#### *11.2.4 Limited model validation*

Finally, the validation of the SHOUT model for metric strength cannot be considered thorough and complete. The participants of the validation survey were reached through social-media, and therefore not verified to be software development professionals. Additionally, the model for team performance has not seen any validation in this study at all, which calls for future work investigating the effectiveness of the model in, for example, separate case-studies.

## 11.3 Future Work

#### *11.3.1 Thorough model validation*

With this study, we have set a first step towards enabling organizations to measure the performance of a software development team. We have not, however, proven that this model for team performance is accurate or valid. In future work, we plan to validate the model in an industry setting using case-studies in which the model's accuracy is validated. Only after this has happened, can mainstream adoption potentially occur.

Similarly, the validation of the SHOUT model for metric strength has yet to see a thorough validation of its capacities. While we have performed a small survey on these qualities, this was solely meant as an initial probing into their perceived clarity, relevance and completeness, and additional, more thorough validation is required in order to draw any significant conclusions.

#### *11.3.2 Additional analysis of the systematic mapping*

Additionally, we have acquired and systematically mapped a substantial part of the available body of knowledge on software development metrics. While this mapping served its purpose in our research more than adequately, we feel very strongly that there are additional patterns and insights to be discovered within it. We have therefore opted to open-source the results, in order to enable other researchers to draw their own conclusions from them.

#### *11.3.3 Investigate the effectiveness quality*

The preliminary validation of the model for metric strength brought forward an additional quality that many seem to associate with *strong* software development metrics, namely *effectiveness*. Future work could benefit from determining what exactly respondents mean with *effectiveness*, whether it is the same as *outcome-oriented*, or whether it might be a potential sixth quality for *strong* software development metrics.

#### *11.3.4 Multidisciplinary approach*

Additionally, it might prove beneficial to approach future work from a multi-disciplinary perspective, as the fields of psychology, sociology and even anthropology might have valuable insights into what qualities contribute to the strength of a metric. In this study, a focus on software development was used, but a broader view might yield a more robust and universal model for metric strength or team performance.

#### *11.3.5 Broader inclusion criteria*

Finally, the inclusion criteria of having to mention *new* software development metrics, as opposed to just *any* software development metric, is a significant blow to the validity of our results. While we have found more than 4.5 times as many software development metrics than any other literature review we have found on the subject, we feel that we will still have potentially missed numerous other metrics due to this inclusion criteria. A thorough reproduction of this literature review will have to broaden this inclusion criteria to state that a work will be included if it mentions *any* software development metric, but this will increase the required effort, time and resources substantially.

# 12. Acknowledgements

I would like to thank Jan Martijn van der Werf from the University of Utrecht for his assistance on the inception, design and execution of this study. His guidance has elevated the academic validity and rigor of this study tremendously. Additionally, I would like to thank Sietse Overbeek for providing me with valuable feedback and pointers along the way. Similarly, I would like to thank all of the experts that participated in the expert inquiry and focus groups for their time and effort, and for helping us synthesize and extract their tacit knowledge into the newly created models for metric strength and team performance. I would like to specifically thank Kyle Aretae for his extraordinary contributions to the development of these models. Finally, I would like to thank Frank Verbruggen for an extraordinarily fruitful and informative collaboration, that I won't soon forget.

# 13. References

Achara, A., Garg, D., Singh, N. & Gahlaut, U. (2019). Plant effectiveness improvement of overall equipment effectiveness using autonomous maintenance training: - A case study. Om 2019 International Journal of Mechanical and Production Engineering Research and Development (pp. 103-112). IEEE.

Leffingwell, D. (2018). *SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Enterprises*. Addison-Wesley Professional.

Agarwal, M., & Majumdar, R. (2012). Tracking scrum projects tools, metrics and myths about agile. Int J Emerg Technol Adv Eng, 2, 97-104.

Aggarwal, K. K., Singh, Y., Kaur, A., & Malhotra, R. (2006). Empirical Study of Object-Oriented Metrics. Journal of Object Technology, 5(8), 149-173.

Ahbrahamsson, P., Conboy, K., & Wang, X. (2009). 'Lots done, more to do'. The current state of agile systems development research.

Ahmed, A., Ahmad, S., Ehsan, N., Mirza, E., & Sarwar, S. Z. (2010, June). Agile software development: Impact on productivity and quality. In Management of innovation and technology (ICMIT), 2010 IEEE international conference on (pp. 287-291). IEEE.

Albrecht A.J. (1979). Measuring Application Development Productivity. In *Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium*, Monterey, California, pp. 83–92.

Alfraihi, H., Lano, K., Kolahdouz-Rahimi, S., Sharbaf, M., & Haughton, H. (2018, October). The Impact of Integrating Agile Software Development and Model-Driven Development: A Comparative Case Study. In International Conference on System Analysis and Modeling (pp. 229-245). Springer, Cham.

Banker, R. D., & Kemerer, C. F. (1992). Performance evaluation metrics for information systems development: A principal-agent model. Information Systems Research, 3(4), 379-400.

Bates, E. (2003). A Capability Maturity Model-Based Approach to the Measurement of Shared Situational Awareness. InterSymp–2003, 101.

Olszewska, M., Heidenberg, J., Weijola, M., Mikkonen, K., & Porres, I. (2016). Quantitatively measuring a large-scale agile transformation. *Journal of Systems and Software*, *117*, 258-273.

Beer, A., & Felderer, M. (2018, June). Measuring and improving testability of system requirements in an industrial context by applying the goal question metric approach. In Proceedings of the 5th International Workshop on Requirements Engineering and Testing (pp. 25-32). ACM.

Bhardwaj, M., & Rana, A. (2016). Key Software Metrics and its Impact on each other for Software Development Projects. ACM SIGSOFT Software Engineering Notes, 41(1), 1-4.

Boehm, B., Abts, C., & Chulani, S. (2000). Software development cost estimation approaches—A survey. *Annals of software engineering*, *10*(1-4), 177-205.

Burgin, M., & Debnath, N. (2005). Quality of software that does not exist. Paper presented at the 20th International Conference on Computers and their Applications 2005, CATA 2005, 441-446. Retrieved from www.scopus.com

Burgin, M., & Debnath, N. (2008). Testing: Organization and evaluation. Paper presented at the 23rd International Conference on Computers and their Applications, CATA 2008, 203-208. Retrieved from www.scopus.com

Caballero, E., Calvo-Manzano, J. A., & San Feliu, T. (2011, June). Introducing scrum in a very small enterprise: A productivity and quality analysis. In European Conference on Software Process Improvement (pp. 215-224). Springer, Berlin, Heidelberg.

Sutherland, J. (2014). *Scrum: the art of doing twice the work in half the time*. Currency.

Calefato, F., & Lanubile, F. (2011). A Planning Poker Tool for Supporting Collaborative Estimation in Distributed Agile Development. In *6th International Conference on Software Engineering Advances (ICSEA 2011)* (pp. 14-19).

Calikli, G., Bener, A., Aytac, T., & Bozcan, O. (2013, October). Towards a metric suite proposal to quantify confirmation biases of developers. In 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 363-372). IEEE.

Canedo, E. D., & Costa, R. P. D. Methods and metrics for estimating and planning agile software projects.

Cardozo, E. S., Neto, J. B. F. A., Barza, A., França, A. C. C., & da Silva, F. Q. (2010, April). SCRUM and Productivity in Software Projects: A Systematic Literature Review. In EASE.

Castro, J. A. O., & Jaimes, W. A. (2017). Dynamic impact of the structure of the supply chain of perishable foods on logistics performance and food security. Journal of Industrial Engineering and Management, 10(4), 687-710.

Chesworth, A. A., Rannow, R. K., Ruiz, O., DeRemer, M., Leite, J., Martinez, A., & Guenther, D. (2016, February). Novel fiber fused lens for advanced optical communication systems. In Terahertz, RF, Millimeter, and Submillimeter-Wave Technology and Applications IX (Vol. 9747, p. 97471P). International Society for Optics and Photonics.

Choobineh, J., Anderson, E., & Barry, E. (2009). Some Throughput Metrics for (SOA) Application Development.

CN, B. (2008). A farmers market at the local sugar mill: Lean versus agile. In *Proc S Afr Sug Technol Ass* (Vol. 81, pp. 68-71).

Coelho, E., & Basu, A. (2012). Effort estimation in agile software development using story points. *International Journal of Applied Information Systems (IJAIS*), 3(7).

Cohn, M., & Ford, D. (2003). Introducing an agile process to an organization [software development]. Computer, 36(6), 74-78.

Coley, C. (2019, March). Building a Rig State Classifier Using Supervised Machine Learning to Support Invisible Lost Time Analysis. In SPE/IADC International Drilling Conference and Exhibition. Society of Petroleum Engineers.

Coraggio, L. (1990). *Deleterious effects of intermittent interruptions on the task performance of knowledge workers: A laboratory investigation* (Doctoral dissertation, University of Arizona).

Cuatrecasas-Arbos, L., Fortuny-Santos, J., & Vintro-Sanchez, C. (2011). The Operations-Time Chart: A graphical tool to evaluate the performance of production systems–From batch-and-queue to lean manufacturing. Computers & Industrial Engineering, 61(3), 663-675.

Cui, J., Ren, L., Zhang, L., & Wu, Q. (2015, June). An optimal allocation method for virtual resource considering variable metrics of cloud manufacturing service. In ASME 2015 International Manufacturing Science and Engineering Conference (pp. V002T04A013-V002T04A013). American Society of Mechanical Engineers.

Damm LO, Lundberg L, Wohlin C. Faults-slip-through—A concept for measuring the efficiency of the test process. Software Process: Improvement and Practice 2006; 11(1):47–59.

Dascalu, S. M., Brown, N., Eiler, D. A., Leong, H. W., Penrod, N. A., Westphal, B. T., & Varol, Y. L. (2005). Software Modeling of S-Metrics Visualizer: Synergetic Interactive Metrics Visualization Tool. In Software Engineering Research and Practice (pp. 870-876).

de Wardt, J., Chapman, C. D., & Behounek, M. (2012). Well Construction Automation-Preparing for the Big Jump.

Dehghanian, P., Aslan, S., & Dehghanian, P. (2018). Maintaining electric system safety through an enhanced network resilience. IEEE Transactions on Industry Applications, 54(5), 4927-4937.

Dick, M., Drangmeister, J., Kern, E., & Naumann, S. (2013, May). Green software engineering with agile methods. In *Green and Sustainable Software (GREENS), 2013 2nd International Workshop on* (pp. 78-85). IEEE.

Diebold, P., Lampasona, C., & Taibi, D. (2013, October). Moonlighting Scrum: An agile method for distributed teams with part-time developers working during non-overlapping hours. In Eighth International Conference on Software Engineering and Advances, IARIA (pp. 318-323).

Domínguez-Mayo, F. J., Escalona, M. J., Mejías, M., Ross, M., & Staples, G. (2012). Quality evaluation for model-driven web engineering methodologies. Information and Software Technology, 54(11), 1265-1282.

Downey, S., & Sutherland, J. (2013, January). Scrum metrics for hyperproductive teams: how they fly like fighter aircraft. In 2013 46th Hawaii International Conference on System Sciences (pp. 4870-4878). IEEE.

Dutoit, A. H., & Bruegge, B. (1998). Communication metrics for software development. *IEEE transactions on Software Engineering*, (8), 615-628.

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. Information and software technology, 50(9-10), 833-859.

Dyba, T., & Dingsoyr, T. (2009). What do we know about agile software development?. IEEE software, 26(5), 6-9.

F. A. Fontana, P. Braione, and M. Zanoni. Automatic detection of bad smells in code: An experimental assessment. Journal of Object Technology, 11(2):5:1–38, 2012.

Fenton, N. E., & Neil, M. (2000, May). Software metrics: roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 357-370). ACM.

Fitzgerald, B., Musiał, M., & Stol, K. J. (2014, May). Evidence-based decision making in lean software project management. In Companion proceedings of the 36th international conference on software engineering (pp. 93-102). ACM.

Frakes, W., & Terry, C. (1996). Software reuse: metrics and models. *ACM Computing Surveys (CSUR)*, *28*(2), 415-435.

Gebre, B. A., & Pochiraju, K. (2017, November). Ball Drive Configurations and Kinematics for Holonomic Ground Mobility. In ASME 2017 International Mechanical Engineering Congress and Exposition (pp. V014T07A018-V014T07A018). American Society of Mechanical Engineers.

Germani, M., Mengoni, M., & Peruzzini, M. (2012). An approach to assessing virtual environments for synchronous and remote collaborative design. Advanced Engineering Informatics, 26(4), 793-813.

Gilb, T. (2004, June). 1.6. 1 Software Project Management: Adding Stakeholder Metrics to Agile Projects. In *INCOSE International Symposium* (Vol. 14, No. 1, pp. 183-190).

Greening, D. R. (2010, January). Enterprise scrum: Scaling scrum to the executive level. In System Sciences (HICSS), 2010 43rd Hawaii International Conference on (pp. 1-10). IEEE.

Prechelt, L. (2019). The Mythical 10x Programmer. In *Rethinking Productivity in Software Engineering* (pp. 3-11). Apress, Berkeley, CA.

Grimaldi, P., Perrotta, L., Corvello, V., & Verteramo, S. (2016). An agile, measurable and scalable approach to deliver software applications in a large enterprise. International Journal of Agile Systems and Management, 9(4), 326-339.

Grimaldi, P., Perrotta, L., Corvello, V., & Verteramo, S. (2016). An agile, measurable and scalable approach to deliver software applications in a large enterprise. International Journal of Agile Systems and Management, 9(4), 326-339.

H. K. N. Leung and L. White, A cost model to compare regression test strategies, in Proc. Conference on Software Maintenance, 1991, pp. 201–208.

H.A. Sahraoui, R. Godin, and T. Miceli, “Can metrics help to bridge the gap between the improvement of oo design quality and its automation?”, In Proc. International Conference on Software Maintenance, pages 154–162, October, 2000.

Halstead, Maurice H. (1977). Elements of Software Science. Amsterdam: Elsevier North-Holland, Inc. ISBN 0-444-00205-7.

Hannay, J. E., & Benestad, H. C. (2010, September). Perceived productivity threats in large agile development projects. In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (p. 15). ACM.

Hartmann, D. and Dymond, R. (July 2006) Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value. In Proceedings of the Conference on AGILE 2006.

He, X., Avgeriou, P., Liang, P., Li, Z.: Technical debt in MDE: a case study on GMF/EMF-based projects. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, pp. 162–172. ACM (2016)

Henderson-Sellers, B. (1996), "*Object Oriented Metrics -Measures of Complexity*", Henderson-Sellers, B., Prentice Hall, Upper Saddle River, NJ, 1996.

Henry, S., & Kafura, D. (1981). Software structure metrics based on information flow. *IEEE transactions on Software Engineering*, (5), 510-518.

Hevner, A. R., Collins, R. W., & Garfield, M. J. (2002). Product and project challenges in electronic commerce software development. ACM SIGMIS Database: the DATABASE for Advances in Information Systems, 33(4), 10-22.

Hughes, M. (2012). A lean, green, school bus making machine-The evolution of Thomas Built Buses shows how greenfield development can become an environmental and business superstar. Industrial Engineer, 44(5), 28.

Huijgens, H., & van Solingen, R. (2013, October). Measuring Best-in-Class Software Releases. In 2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement (pp. 137-146). IEEE.

Jackson, T. W. (2015). Moving forward with digital reliability assessments. Paper presented at the 9th International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies, NPIC and HMIT 2015, , 3 2381-2386. Retrieved from www.scopus.com

Jeffery, R., Ruhe, M., & Wieczorek, I. (2001). Using public domain metrics to estimate software development effort. In *Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International* (pp. 16-27). IEEE.

Kafura, D., & Henry, S. (1981). Software quality metrics based on interconnectivity. *Journal of systems and software*, *2*(2), 121-131.

Keeling, T., Clements-Croome, D., & Roesch, E. (2015). The effect of agile workspace and remote working on experiences of privacy, crowding and satisfaction. Buildings, 5(3), 880-898.

Kemerer, C. F., & Paulk, M. C. (2009). The impact of design and code reviews on software quality: An empirical study based on psp data. IEEE transactions on software engineering, 35(4), 534-550.

Khadem, M., Ali, S. A., & Seifoddini, H. (2008). Efficacy of lean metrics in evaluating the performance of manufacturing systems. International Journal of Industrial Engineering, 15(2), 176-184.

Khoshgoftaar, T. M., & Munson, J. C. (1990). Predicting software development errors using software complexity metrics. *IEEE Journal on Selected Areas in Communications*, *8*(2), 253-261.

Koru, A. G., & El Emam, K. (2009). Theory of Relative Dependency: Higher Coupling Concentration in Smaller Modules and its Implications for Software Refactoring and Quality. IEEE software.

Kunz, M., Dumke, R. R., & Zenker, N. (2008, March). Software metrics for agile software development. In 19th Australian Conference on Software Engineering (aswec 2008) (pp. 673-678). IEEE.

Kupiainen, E., Mäntylä, M. V., & Itkonen, J. (2015). Using metrics in Agile and Lean Software Development–A systematic literature review of industrial studies. Information and Software Technology, 62, 143-163.

Lano, K. C., Alfraihi, H. A. A., Kolahdouz Rahimi, S., Sharbaf, M., & Haughton, H. (2018). Comparative case studies in agile model-driven development. In International Conference on Model Driven Engineering Languages and Systems (MODELS): the 4th International Workshop on Flexible Model Driven Engineering (pp. 203-212). Copenhagen, Denmark.

Lee, Y. T. T., Lee, J. Y., Riddick, F., Libes, D., & Kibira, D. (2013). Interoperability for virtual manufacturing systems. International Journal of Internet Manufacturing and Services, 3(2), 99-120.

Li, Z., & Li, X. (2019). A Multi-objective Binary-encoding Differential Evolution Algorithm for Proactive Scheduling of Agile Earth Observation Satellites. Advances in Space Research.

Lind, R. K., & Vairavan, K. (1989). An experimental investigation of software metrics and their relationship to software development effort. *IEEE Transactions on Software Engineering*, *15*(5), 649-653.

Liu, X., & Wang, W. (2005, November). On the characteristics of spectrum-agile communication networks. In *New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005. 2005 First IEEE International Symposium on* (pp. 214-223). IEEE.

Mahnic, V. and Zabkar, N. (October 2008). Using COBIT indicators for measuring Scrum-based software development. Wseas transactions on computers. 10(7). pp. 1605 - 1617

Maurer, F., & Martel, S. (2002, March). On the productivity of agile software practices: An industrial case study. In Proceedings of the International Workshop on Global Software Development(Vol.

Melo, C. D. O., Cruzes, D. S., Kon, F., & Conradi, R. (2013). Interpretative case studies on agile team productivity and management. Information and Software Technology, 55(2), 412-427.

Melo, C., Cruzes, D. S., Kon, F., & Conradi, R. (2011). Agile team perceptions of productivity factors. In 2011 Agile Conference (pp. 57-66). IEEE.

Meso, P., & Jain, R. (2006). Agile software development: adaptive systems principles and best practices. *Information systems management*, *23*(3), 19-30.

Minkiewicz, A. (1998), "*Measuring Object Oriented Software with Predictive Object Points*,", PRICE Systems, 1998.

Moreau, D. R., & Dominick, W. D. (1989). Object-oriented graphical information systems: Research plan and evaluation metrics. Journal of Systems and Software, 10(1), 23-28.

Moser, R., Abrahamsson, P., Pedrycz, W., Sillitti, A., & Succi, G. (2008). A case study on the impact of refactoring on quality and productivity in an agile team. In Balancing Agility and Formalism in Software Engineering (pp. 252-266). Springer, Berlin, Heidelberg.

Moser, R., Pedrycz, W., & Succi, G. (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th international conference on Software engineering* (pp. 181-190). ACM.

Nujoom, R., Wang, Q., & Mohammed, A. (2018). Optimisation of a sustainable manufacturing system design using the multi-objective approach. The International Journal of Advanced Manufacturing Technology, 1-20.

Oddone, R., & Chen, L. (2014). Challenges and Novel Solutions for SoC Verification. ECS Transactions, 60(1), 1191-1195.

Olague, H. M., Etzkorn, L. H., Gholston, S., & Quattlebaum, S. (2007). Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. *IEEE Transactions on software Engineering*, *33*(6), 402-419.

Oliveira, M. F., Redin, R. M., Carro, L., da Cunha Lamb, L., & Wagner, F. R. (2008, April). Software quality metrics and their impact on embedded software. In 2008 5th International Workshop on Model-based Methodologies for Pervasive and Embedded Software (pp. 68-77). IEEE.

Onyemechi, C. (2013). Port efficiency modelling in the post concessioning era: The role of logistics drivers, agile ports and other perspectives. Pomorstvo, 27(2), 279-283.

Oza, N., & Korkala, M. (2012, March). Lessons Learned In Implementing Agile Software Development Metrics. In UKAIS (p. 38).

Padmini, K. J., Bandara, H. D., & Perera, I. (2015, April). Use of software metrics in agile software development process. In *Moratuwa Engineering Research Conference (MERCon), 2015*(pp. 312-317). IEEE.

Padmini, K. J., Kankanamge, P. S., Bandara, H. D., & Perera, G. I. U. S. (2018, May). *Challenges Faced by Agile Testers: A Case Study.* In 2018 Moratuwa Engineering Research Conference (MERCon) (pp. 431-436). IEEE.

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems, 24*(3), 45-77.

Petersen, K. and Wohlin, C. 2011. Measuring the flow in lean software development, Softw. Pract. Exper., 41, 975- 996.

Pham, T. M. (2018, July). Optimization Model and Algorithm for Dynamic Service-Aware Traffic Steering in Network Functions Virtualization. In 2018 IEEE Seventh International Conference on Communications and Electronics (ICCE) (pp. 107-112). IEEE.

Razzak, M. A., Noll, J., Richardson, I., Canna, C. N., & Beecham, S. (2017, November). Transition from plan driven to SAFe®: periodic team self-assessment. In International Conference on Product-Focused Software Process Improvement(pp. 573-585). Springer, Cham.

Rosenberg, L. H., & Hyatt, L. E. (1997). Software quality metrics for object-oriented environments. Crosstalk journal, 10(4), 1-6.

Rosero, R. H., Gómez, O. S., & Rodríguez, G. (2016). 15 years of software regression testing techniques—A survey. International Journal of Software Engineering and Knowledge Engineering, 26(05), 675-689.

S. Demeyer, S. Ducasse, O. Nierstrasz, "Finding Refactorings via Change Metrics", In Proceedings of the 15th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA'00, Minneapolis, USA, 2000

Sahu, A. K., Sahu, N. K., & Sahu, A. K. (2017). Performance estimation of firms by GLA supply chain under imperfect data. In Theoretical and Practical Advancements for Fuzzy System Integration (pp. 245-277). IGI Global.

Savola, R. M., Frühwirth, C., & Pietikäinen, A. (2012). Risk-Driven Security Metrics in Agile Software Development-An Industrial Pilot Study. *J. UCS*, *18*(12), 1679-1702.

Scott, E., & Pfahl, D. (2017). Exploring the individual project progress of scrum software developers doi:10.1007/978-3-319-69926-4_24 Retrieved from www.scopus.com

Seaman, C., & Guo, Y. (2011). *Measuring and monitoring technical debt.* In Advances in Computers (Vol. 82, pp. 25-46). Elsevier.

Serrador, P., & Pinto, J. K. (2015). Does Agile work?—A quantitative analysis of agile project success. *International Journal of Project Management*, *33*(5), 1040-1051.

Shah, S. M. A., Papatheocharous, E., & Nyfjord, J. (2015). Measuring productivity in agile software development process: a scoping study. In Proceedings of the 2015 International Conference on Software and System Process (pp. 102-106). ACM.

Sirkiä, R., & Laanti, M. (2013). Lean and agile financial planning. *White Paper*, *24*, 2013.

Sjøberg, D. I., Johnsen, A., & Solberg, J. (2012). Quantifying the effect of using kanban versus scrum: A case study. IEEE software, 29(5), 47-53.

Song, R., Tang, H., Mason, P. C., & Wei, Z. (2013, November). Cross-layer security management framework for mobile tactical networks. In MILCOM 2013-2013 IEEE Military Communications Conference (pp. 220-225). IEEE.

Subramanya, S., Mustafa, Z., Irwin, D., & Shenoy, P. (2016, March). Beyond energy-efficiency: evaluating green datacenter applications for energy-agility. In Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (pp. 185-196). ACM.

Sunil, P. U., Barve, J., & Nataraj, P. S. V. (2017). Mathematical modeling, simulation and validation of a boiler drum: Some investigations. Energy, 126, 312-325.

Suri, N., Bradshaw, J. M., Carvalho, M. M., Cowin, T. B., Breedy, M. R., Groth, P. T., & Saavedra, R. (2003, May). Agile computing: Bridging the gap between grid computing and ad-hoc peer-to-peer

resource sharing. In CCGrid 2003. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003. Proceedings. (pp. 618-625). IEEE.

Sutherland, J., Harrison, N., & Riddle, J. (2014, January). Teams that finish early accelerate faster: a pattern language for high performing scrum teams. In System Sciences (HICSS), 2014 47th Hawaii International Conference on (pp. 4722-4728). IEEE.

Sutherland, J., Schoonheim, G., & Rijk, M. (2009, January). Fully distributed scrum: Replicating local productivity and quality with offshore teams. In 2009 42nd Hawaii International Conference on System Sciences (pp. 1-8). IEEE.

Sutherland, J., Schoonheim, G., Rustenburg, E., & Rijk, M. (2008, August). Fully distributed scrum: The secret sauce for hyperproductive offshored development teams. In Agile 2008 Conference (pp. 339-344). IEEE.

Tegarden, D., Sheetz, S., Monarchi, D., "Effectiveness of Traditional Software Metrics for Object-Oriented Systems", Proceedings: 25th Hawaii International Conference on System Sciences, January, 1992, pp. 359-368.

Triki, I., El-Azouzi, R., & Haddad, M. (2016, June). NEWCAST: Anticipating resource management and QoE provisioning for mobile video streaming. In 2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM) (pp. 1-9). IEEE.

Tsatsoulas, A., Barkate, J., Baylis, C., & Marks, R. J. (2016, May). A simplex optimization technique for real-time, reconfigurable transmitter power amplifiers. In 2016 IEEE MTT-S International Microwave Symposium (IMS) (pp. 1-4). IEEE.

Verbruggen, F., Sutherland, J., van der Werf, J. M., Brinkkemper, S., & Sutherland, A. (2019, January). Process Efficiency-Adapting Flow to the Agile Improvement Effort. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*.

von Alan, R. H., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS quarterly*, 28 (1), 75{105.

Wang, A., Jin, Z., & Xu, W. (2016, August). A programmable analog-to-information converter for agile biosensing. In Proceedings of the 2016 International Symposium on Low Power Electronics and Design (pp. 206-211). ACM.

Wang, X., & Garcia-Luna-Aceves, J. J. (2011). Collaborative routing, scheduling and frequency assignment for wireless Ad Hoc networks using spectrum-agile radios. Wireless Networks, 17(1), 167-181.

West, D., Gilpin, M., Grant, T., & Anderson, A. (2011). Water-scrum-fall is the reality of agile for most organizations today. *Forrester Research*, 26.

White, K. (2017). “Effing” the military: a political misunderstanding of management. Defence Studies, 17(4), 346-358.

Wiart, B., Peyronnet, P., Moity, N., & Pradeilles, F. (2002). SimEC3: Innovative Simulation Based Acquisition Tool for the France’s Cooperative Fighting System (Vol. 16). RTO-MP-MSG-035.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering* (p. 38).

Z. Li, P. Liang, P. Avgeriou, N. Guelfi, and A. Ampatzoglou. An empirical investigation of modularity metrics for indicating architectural technical debt. In Proceedings of the 10th international ACM SIGSOFT conference on Quality of Software Architectures, pages 119–128. ACM, 2014.

Zammori, F., Braglia, M., & Frosolini, M. (2011). Stochastic overall equipment effectiveness. International Journal of Production Research, 49(21), 6469-6490.

Zhang, Y., & Tanniru, M. (2005). Business Flexibility and Operational Efficiency-Making Trade-Offs in Service Oriented Architecture. AMCIS 2005 Proceedings, 237.

Zhou, X., Jin, Y., Zhang, H., Li, S., & Huang, X. (2016, December). A map of threats to validity of systematic literature reviews in software engineering. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)* (pp. 153-160). IEEE.

# Appendix A - Google Scholar Candidates

| Query | Candidate | Title | Reference | Citations |
|---|---|---|---|---|
| | | | | |
| **Software Development Metrics** | | | | |
| | **GS.1.01** | Predicting software development errors using software complexity metrics. | Khoshgoftaar & Munson, 1990 | 309 |
| | **GS.1.02** | Using public domain metrics to estimate software development effort. | Jefferey, Ruhe & Wieczorek, 2001 | 209 |
| | **GS.1.03** | Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes | Olague, Etzkorn, Gholston & Quattlebaum, 2007 | 328 |
| | **GS.1.04** | Software development cost estimation approaches - A survey | Boehm, Abts & Chulani, 2000 | 725 |
| | **GS.1.05** | An experimental investigation of software metrics and their relationship to software development effort | Lind & Vairavan, 1989 | 138 |
| | **GS.1.06** | Software quality metrics based on interconnectivity. | Kafura & Henry, 1981 | 146 |
| | **GS.1.07** | Software structure metrics based on information flow. | Henry & Kafura, 1981 | 1046 |
| | **GS.1.08** | Communication metrics for software development | Dutoit & Bruegge, 1998 | 85 |
| | **GS.1.09** | Software metrics: roadmap | Fenton & Neil, 2000 | 440 |
| | **GS.1.10** | Software reuse: metrics and models | Frakes & Terry, 1996 | 447 |
| | | | | |
| **Agile Efficiency Metrics** | | | | |
| | **GS.2.01** | Does agile work? A quantitative analysis of agile project success | Serrador & Pinto, 2015 | 246 |
| | **GS.2.02** | Risk-driven security metrics in agile software development - An industrial pilot study. | Savola, Frühwirth & Pietikäinen, 2012 | 15 |
| | **GS.2.03** | A farmers market at the local sugar mill: lean versus agile | Bezuidenhout, 2008 | 14 |
| | **GS.2.04** | Green software engineering with agile methods | Dick, Drangmesiter, Kern & Naumann, 2013 | 49 |

| | | | | |
|---|---|---|---|---|
| | **GS.2.05** | A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction | Moser, Pedrycz & Succi, 2008 | 483 |
| | **GS.2.06** | Use of software metrics in agile software devlopment processes | Padmini, Bandara & Pererea, 2015 | 14 |
| | **GS.2.07** | On the characteristics of spectrum-agile communication networks | Lui & Wang, 2005 | 65 |
| | **GS.2.08** | Lean and agile financial planning | Sirkiä & Laanti, 2013 | 3 |
| | **GS.2.09** | Software project management: Adding stakeholder metrics to agile projects | Gilb, 2004 | 7 |
| | **GS.2.10** | Agile software development: adaptive systems principles and best practices | Meso & Jain, 2006 | 187 |
| | | | | |
| **Scrum Productivity Metrics** | | | | |
| | **GS.3.01** | Scrum metrics for hyperproductive teams: how they fly like fighter aircraft | Downey & Sutherland, 2013 | 44 |
| | **GS.3.02** | Enterprise scrum: Scaling scrum to the executive level | Greening, 2010 | 22 |
| | **GS.3.03** | Introducing scrum in a very small enterprise: A productivity and quality analysis | Caballero &Calvo-Manzano & Feliu, 2011 | 12 |
| | **GS.3.04** | Tracking scrum projects tools, metrics and myths about agile | Agarwal & Majumdar, 2012 | 12 |
| | **GS.3.05** | Fully distributed scrum: Replicating local productivity and quality with offshore teams | Sutherland, Schoonheim & Rijk, 2009 | 69 |
| | **GS.3.06** | Teams that finish early accelerate faster: a pattern language for high performing scrum teams | Sutherland, Harrison & Riddle, 2014 | 16 |
| | **GS.3.07** | On the productivity of agile software practices: An industrial case study | Maurer & Martel, 2002 | 32 |
| | **GS.3.08** | Moonlighting Scrum: An agile method for distributed teams with part-time developers working during non-overlapping hours | Diebold, Lampasona & Taibi, 2013 | 15 |
| | **GS.3.09** | Fully distributed scrum: The secret sauce for hyperproductive offshored development teams | Sutherland, Schoonheim, Rustenburg & Rijk, 2008 | 89 |

| | | | | |
|---|---|---|---|---|
| | **GS.3.10** | Measuring productivity in agile software development process: a scoping study | Shah, Papatheocharous & Nyfjord, 2015 | 6 |
| | | | | |
| **Agile Productivity** | | | | |
| | **GS.4.01** | Interpretative case studies on agile team productivity and management | Melo, Cruzes, Kon & Conradi, 2013 | 79 |
| | **GS.4.02** | Agile software development: Impact on productivity and quality | Agmed, Ahmad, Ehsan, Mirza & Sarwar, 2010 | 63 |
| | **GS.4.03** | Empirical studies of agile software development: A systematic review | Dybå & Dingsøyr, 2008 | 2116 |
| | **GS.4.04** | A case study on the impact of refactoring on quality and productivity in an agile team | Moser, Abrahamsson, Oedrycz, Sillitti & Succi, 2008 | 101 |
| | **GS.4.05** | Agile team perceptions of productivity factors | Melo, Cruzes, Kon, Conradi & 2011 | 52 |
| | **GS.4.06** | On the productivity of agile software practices: An industrial case study | Maurer & Martel, 2002 | 32 |
| | **GS.4.07** | Perceived productivity threats in large agile development projects | Hannay & Benestad, 2010 | 31 |
| | **GS.4.08** | Introducing an agile process to an organization | Cohn & Ford, 2003 | 282 |
| | **GS.4.09** | SCRUM and Productivity in Software Projects: A Systematic Literature Review. | Cardozo, Neto, Barza, Franca & Da Silva, 2010 | 47 |
| | **GS.4.10** | What do we know about agile software development? | Dybå & Dingsøyr, 2009 | 249 |

# Appendix B - Scopus Candidates

| Query | Candidate | Title | Reference | Citations |
|---|---|---|---|---|
| | | | | |
| **Software Development Metrics** | | | | |
| | **SC.1.01** | Key software metrics and its impact on each other for software development projects | Bhardwaj & Rana, 2016 | 2 |
| | **SC.1.02** | Moving forward with digital reliability assessments | Jackson, 2015 | 0 |
| | **SC.1.03** | Evidence-based decision making in lean software project management | Fitzegrald, Musial & Stol, 2014 | 13 |
| | **SC.1.04** | Challenges and novel solutions for SoC verification | Oddone & Chen, 2014 | 0 |
| | **SC.1.05** | Towards a metric suite proposal to quantify confirmation biases of developers | Calikli, Bener, Aytac & Bozcan, 2013 | 5 |
| | **SC.1.06** | Some throughput metrics for (SOA) application development | Choobineh, Anderson & Barry, 2009 | 3 |
| | **SC.1.07** | Software reliability assessment based on agile software development metrics | Dascula et al., 2005 | 4 |
| | **SC.1.08** | Testing: Organization and evaluation | Burgin & Debnath, 2008 | 6 |
| | **SC.1.09** | Quality of software that does not exist | Burgin & Debnath, 2005 | 3 |
| | **SC.1.10** | Software modelling of S-Metrics Visualizer: Synergetic interactive metrics visualization tool | Dascula et al., 2005 | 1 |
| | **SC.1.11** | Product and Project Challenges in Electronic Commerce Software Development | Hevner, Collins & Garfield, 2002 | 14 |
| | **SC.1.12** | Performance evaluation metrics for information systems development: A principal-agent model | Banker & Kemerer, 1992 | 99 |
| | **SC.1.13** | Object-oriented graphical information systems: Research plan and evaluation metrics | Moreau & Dominick, 1989 | 55 |
| | | | | |
| **Agile Efficiency Metrics** | | | | |
| | **SC.2.01** | A multi-objective binary-encoding differential evolution algorithm for proactive scheduling of agile earth observation satellites | Li & Li, 2019 | 0 |

| | | | | |
|---|---|---|---|---|
| | **SC.2.02** | Building a rig state classifier using supervised machine learning to support invisible lost time analysis | Coley, 2019 | 0 |
| | **SC.2.03** | Plant effectiveness improvement of overall equipment effectiveness using autonomous maintenance training: - A case study | Achara, Garg, Singh & Gahlaut, 2019 | 0 |
| | **SC.2.04** | Optimization model and algorithm for dynamic service-aware traffic steering in network functions virtualization | Pham, 2018 | 0 |
| | **SC.2.05** | Maintaining Electric System Safety Through An Enhanced Network Resilience | Dehghanian & Aslan, 2018 | 16 |
| | **SC.2.06** | Measuring and improving testability of system requirements in an industrial context by applying the goal question metric approach | Beer & Felderer, 2018 | 1 |
| | **SC.2.07** | Optimisation of a sustainable manufacturing system design using the multi-objective approach | Nujoom, Wang & Mohammed, 2018 | 5 |
| | **SC.2.08** | The impact of integrating agile software development and model-driven development: A comparative case study | Alfraihi, Lano Kolahdouz-Rahimi, Sharbaf & Haughton, 2018 | 0 |
| | **SC.2.09** | Comparative case studies in agile model-driven development | Lano, Alfraihi, Kolahdouz-Rahimi, Sharbaf & Haughton, 2018 | 0 |
| | **SC.2.10** | Real-time detection of under-reamer failure: An example of agile data analytics development and deployment | De Wardt, Chapman & Behounek, 2012 | 4 |
| | **SC.2.11** | “Effing” the military: a political misunderstanding of management | White, 2017 | 1 |
| | **SC.2.12** | Performance estimation of firms by G-L-A supply chain under imperfect data ( Book Chapter) | Sahu & Sahu, 2017 | 7 |
| | **SC.2.13** | Ball drive configurations and kinematics for holonomic ground mobility | Gebre & pochiraju, 2017 | 2 |
| | **SC.2.14** | Mathematical modelling, simulation and validation of a boiler drum: Some investigations | Sunil, Barve & Nataraj, 2017 | 12 |
| | **SC.2.15** | Dynamic impact of the structure of the supply chain of perishable foods on logistics performance and food security | Castro & Jaimes, 2017 | 7 |
| | **SC.2.16** | A simplex optimization technique for real-time, reconfigurable transmitter power amplifiers | Tsatsoulas, Barkate, Baylis & Marks, 2016 | 2 |

| | | | | |
|---|---|---|---|---|
| | **SC.2.17** | A Programmable Analog-to-Information Converter for Agile Biosensing | Wang, Jin & Xu, 2016 | 1 |
| | **SC.2.18** | Anticipating resource management and QoE provisioning for mobile video streaming | Triki, El-Azouzi & Haddad, 2016 | 9 |
| | **SC.2.19** | 15 Years of Software Regression Testing Techniques - A Survey | Rosero, Gómez & Rodríguez, 2016 | 14 |
| | **SC.2.20** | Beyond energy-efficiency: Evaluating green datacenter applications for energy-agility | Subramanya, Mustafa, Irwin & Shenoy, 2016 | 5 |
| | **SC.2.21** | An agile, measurable and scalable approach to deliver software applications in a large enterprise | Grimaldo, Perrotta, Corvello & Verteramo, 2016 | 2 |
| | **SC.2.22** | Novel fiber fused lens for advanced optical communication systems | Chesworth et al., 2015 | 0 |
| | **SC.2.23** | The effect of agile workspace and remote working on experiences of privacy, crowding and satisfaction | Keeling, Clements-Croome & Roesch, 2015 | 5 |
| | **SC.2.24** | An optimal allocation method for virtual resource considering variable metrics of cloud manufacturing service | Cui, Ren, Zhang & Wu, 2015 | 5 |
| | **SC.2.25** | Cross-layer security management framework for mobile tactical networks | Song, Tang, Mason & Wei, 2013 | 3 |
| | **SC.2.26** | Interoperability for virtual manufacturing systems | Lee, Lee, Riddick, Libes & Kibira, 2013 | 5 |
| | **SC.2.27** | Port efficiency modelling in the post concessioning era: The role of logistics drivers, agile ports and other perspectives | Onyemechi, 2013 | 0 |
| | **SC.2.28** | Quality evaluation for Model-Driven Web Engineering methodologies | Domínguez-Mayo, Escalona, Mejías, Ross & Staples, 2012 | 26 |
| | **SC.2.29** | An approach to assessing virtual environments for synchronous and remote collaborative design | Germani, Mengoni & Peruzzini, 2012 | 17 |
| | **SC.2.30** | Risk-driven security metrics in agile software development - An industrial pilot study | Savola, Frühwith & Pietikäinen, 2012 | 17 |
| | **SC.2.31** | A lean, green, school bus making machine | Hughes, 2012 | 2 |
| | **SC.2.32** | Stochastic overall equipment effectiveness | Zammori, Braglia & Frosolini, 2011 | 57 |

| | | | | |
|---|---|---|---|---|
| | **SC.2.33** | The Operations-Time Chart: A graphical tool to evaluate the performance of production systems - From batch-and-queue to lean manufacturing | Cuatrecasas-Arbos, Fortuny-Santos & Vitro-Sanchez, 2011 | 35 |
| | **SC.2.34** | Collaborative routing, scheduling and frequency assignment for wireless Ad Hoc networks using spectrum-agile radios | Wang & Garcia, 2011 | 13 |
| | **SC.2.35** | The theory of relative dependency: Higher coupling concentration in smaller modules | Koru & El Emam, 2009 | 7 |
| | **SC.2.36** | Collaborative Routing, scheduling and frequency assignment for Wireless ad hoc Networks using spectrum-agile radios | Wang & Garcia-Luna-Aceves, 2011 | 13 |
| | **SC.2.37** | Efficacy of lean metrics in evaluating the performance of manufacturing systems | Khadem, Ali & Seifoddini, 2008 | 44 |
| | **SC.2.38** | Business flexibility and operational efficiency -making trade-offs in service oriented architecture | Zhang & Tanniru, 2005 | 6 |
| | **SC.2.39** | SimEC3: An innovative simulation based acquisition tool for the France's cooperative fighting system | Wiart, Peyronney, Moity & Pradeilles, 2002 | 1 |
| | **SC.2.40** | Agile computing: Bridging the gap between grid computing and ad-hoc peer-to-peer resource sharing | Suri er al., 2003 | 35 |
| | **SC.2.41** | A capability maturity model-based approach to the measurement of shared situational awareness | Bates, 2003 | 0 |
| | **SC.2.42** | Using metrics in agile and lean software development a systematic literature review of industrial studies | Kupiainen, Mäntylä & Itkonen, 2015 | 90 |
| | | | | |
| **Scrum Productivity Metrics** | | | | |
| | **SC.3.01** | Methods and metrics for estimating and planning agile software projects | Canedo & Costa, 2007 | 0 |
| | **SC.3.02** | Exploring the individual project progress of scrum software developers | Scoot & Pfahl, 2017 | 0 |
| | **SC.3.03** | Transition from plan driven to SAFe®: Periodic team self-assessment | Razak, Noll, Richardson, Canna & Beecham, 2017 | 1 |
| | **SC.3.04** | An agile, measurable and scalable approach to deliver software applications in a large enterprise | Grimaldo, Perrotta, Corvello, Verteramo, 2016 | 2 |
| | **SC.3.05** | Scrum metrics for Hyperproductive Teams: How they fly like fighter aircraft | Downey & Sutherland, 2013 | 47 |

| | | | | |
|---|---|---|---|---|
| | **SC.3.06** | Measuring best-in-class software releases | Huijgens & Van Solingen, 2013 | 4 |
| | **SC.3.07** | Quantifying the effect of using Kanban versus scrum: A case study | Sjøberg, Johnsen & Solberg, 2012 | 84 |
| | | | | |
| **Agile Productivity** | | | | |
| | **SC.4.01** | Measuring productivity in agile software development process: A scoping study | Shah, Papatheocharous & Nyfjord, 2015 | 5 |

# Appendix C - Snowballing Results

| Iteration | Inclusion | Title | Reference |
|---|---|---|---|
| **Iteration 1** | | | |
| | SN.1.01 | Empirical Study of Object-Oriented Metrics | Minkiewicz, 1998 |
| | SN.1.02 | Lessons Learned In Implementing Agile Software Development Metrics | Oza & Korkala, 2012 |
| | SN.1.03 | Software Metrics for Agile Software Development | Kunz, Dumke & Zenker, 2008 |
| | SN.1.04 | The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data | Kemerer & Paulk, 2009 |
| | SN.1.05 | Software Quality Metrics for Object-Oriented Environments | Rosenburg & Hyatt, 1997 |
| | SN.1.06 | Finding Refactorings via Change Metrics | Demeyer, Ducasse & Nierstrasz |
| | SN.1.07 | Can metrics help to bridge the gap between the improvement of OO design quality and its automation? | Sahraoui, Godin & Miceli, 2000 |
| | SN.1.08 | Measuring the flow in Lean software development | Petersen & Wohlin, 2011 |
| | SN.1.09 | Technical Debt in MDE: A Case Study on GMF / EMF - Based Projects | He, Avgeriou, Liang & li, 2016 |
| | SN.1.10 | A Cost Model to Compare Regression Test Strategies | Leung & White, 1991 |
| **Iteration 2** | | | |
| | SN.2.01 | Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value | Hartmann & Dymond, 2006 |
| | SN.2.02 | Using COBIT Indicators for Measuring Scrum - based Software Development | Mahnic & Zabkar, 2008 |
| | SN.2.03 | Effectiveness of software metrics for object-oriented system | Tegarden, Sheetz & Monarchi, 1992 |
| | SN.2.04 | Faults-slip-through - A Concept for Measuring the Efficiency of the Test Process | Damm, Lundberg & Wohlin, 2006 |
| | SN.2.05 | Automatic detection of bad smells in code: An experimental assessment | Fontana, Braione & Zanoni, 2012 |
| | SN.2.06 | An Empirical Investigation of Modularity Metrics for Indicating Architectural Technical Debt | Li, Liang, Avgeriou & Guelfi, 2014 |
| **Iteration 3** | | | |

| | | | |
|---|---|---|---|
| | SN.3.01 | Software Quality Metrics and their Impact on Embedded Software | Oliveira, Redin, Carro, Da Cuhna Lamb & Wagner, 2008 |
| **Iteration 4** | | | |
| | SN.4.01 | Empirical Study of Object-Oriented Metrics | Aggarwal, Singh, Kaur & Malhotra, 2006 |

# Appendix D - Metric Introductions

| Metric | Origin |
|---|---|
| Access to Foreign Data | (Marinescu, 2001) |
| Accuracy of Estimation | (Downey & Sutherland, 2013) |
| Accuracy of Forecast | (Downey & Sutherland, 2013) |
| Afferent Coupling | (Martin, 1995) |
| Amount of Code Smell Occurrences | (Fowler & Beck, 1999) |
| Amount of Cycles in Dependency Graph | (Kunz, Dumke & Zenker, 2008) |
| Amount of Lines of Generated Code | (He, Avgeriou, Liang & Li, 2016) |
| Amount of Manually Created Lines of Code | (He, Avgeriou, Liang & Li, 2016) |
| Amount of Modified Lines of Generated Code | (He, Avgeriou, Liang & Li, 2016) |
| Attribute Hiding Factor | (Abreu & Carapuça, 1994) |
| Attribute Inheritance Factor | (Abreu & Carapuça, 1994) |
| Average Amount of Defects Carried to Next Iteration | (Hartmann & Dymond, 2006) |
| Average Class-to-Leaf Depth | (Chidamber & Kemerer, 1994) |
| Average Code Review Rate | (Kemerer & Paulk, 2009) |
| Average Design Review Rate | (Kemerer & Paulk, 2009) |
| Average Fault Cost | (Damm, Lundberg & Wohlin, 2006) |
| Average Number of Modified Components per Commit | (Li, Liang, Avgeriou, Guelfi & Ampatzoglou, 2014) |
| Average Number of Stories Added to Iteration | (Oza & Korkala, 2012) |
| Average Number of Stories Removed to Iteration | (Oza & Korkala, 2012) |
| Average Overtime per Day | (Mahnic & Vrana, 2007) |
| Average Overtime per Sprint | (Mahnic & Vrana, 2007) |
| Average Projects per Employee | (Mahnic & Vrana, 2007) |
| Average Work in Progress | (Hall, 1981) |
| Business Value Delivered | (Hartmann & Dymond, 2006) |
| Capacity Utilization | (Petersen & Wohlin, 2011) |
| Cashflow per Iteration | (Hartmann & Dymond, 2006) |
| Change Requests per Requirement | (Petersen & Wohlin, 2010) |
| Check-Ins per Day | (Humble, Read & North, 2006) |
| Class Attribute Import Coupling | (Li & Henry, 1993) |
| Code Abstractness | (Oliveira, Redin, Carro, Da Cunha & Wagner, 2008) |
| Code Instability | (Oliveira, Redin, Carro, Da Cunha & Wagner, 2008) |
| Comment Percentage | (Rosenverg & Stapko, 1999) |
| Common Tempo Time | (Tanner & Roncarti, 1994) |
| Cost Efficiency | (Petersen & Wohlin, 2011) |

| | |
|---|---|
| Cost of Quality | (Gupta & Campbell, 1995) |
| Cost Performance Index | (Mahnic & Vrana, 2007) |
| Coupling Between Objects | (Chidamber & Kemerer, 1994) |
| Coupling Concentration Index | (Koru & El Emam, 2009) |
| Coupling Factor | (Abrey & Carapuça, 1994) |
| Critical Defects Sent by Customers | (Cheng, Jansen & Remmers, 2009) |
| Cycle Time | (De Jong, 1957) |
| Cyclomatic Complexity Metric | (McCabe, 1976) |
| Data Abstraction Coupling | (Li & Henry, 1993) |
| Delivery on Time | (Petersen & Wohlin, 2011) |
| Depth of Inheritance Tree | (Chidamber & Kemerer, 1994) |
| Descendant Method-to-Method Export Coupling | (Sahraoui, Goding & Miceli, 2000) |
| Due Date Performance | (Seidmann & Smith, 1981) |
| Efferent Coupling | (Martin, 1994) |
| Enterprise Velocity | (Greening, 2010) |
| Fault Latency | (Damm, Lundberg & Wohlin, 2006) |
| Faults Slip Through | (Damm, Lundberg & Wohlin, 2006) |
| Focus Factor | (Downey & Sutherland, 2013) |
| Fulfilment of Scope | (Mahnic & Vrana, 2007) |
| Halstead Complexity Metric | (Maurice, 1977) |
| Ideal Days | (Angioni et al., 2006) |
| Impediments per Work-Item | (Mahnic & Vrana, 2007) |
| Implemented Versus Wasted Requirements | (Petersen & Wohlin, 2010) |
| Improvement Potential | (Tanaka, Sakamoto, Kusumoto, Matsumoto & Kikuno, 1995) |
| Index of Inter-Package Extending | (Abdeen, Ducasse & Sahraoui, 2011) |
| Index of Inter-Package Extending Diversion | (Abdeen, Ducasse & Sahraoui, 2011) |
| Index of Inter-Package Usage | (Abdeen, Ducasse & Sahraoui, 2011) |
| Index of Inter-Package Usage Diversion | (Abdeen, Ducasse & Sahraoui, 2011) |
| Index of Package Changing Impact | (Abdeen, Ducasse & Sahraoui, 2011) |
| Index of Package Goal Focus | (Abdeen, Ducasse & Sahraoui, 2011) |
| Information-Based Cohesion | (Lee, Liang, Wu & Wang, 1995) |
| Internal Efficiency | (Grimaldo, Perrotta, Corvello & Verteramo, 2016) |
| Kick-Off Days | (Grimaldo, Perrotta, Corvello & Verteramo, 2016) |
| Lack of Cohesion of Methods | (Chidamber & Kemerer, 1994) |
| Lead Time | (Krafcik, 1988) |
| Locality of Attribute Accesses | (Lanzá & Marinescu, 2006) |
| Maximum Nested Block Depth | (Wichmann & Cox, 1992) |
| Message Passing Coupling | (Henderson-Sellers, 1996) |
| Method Hiding Factor | (Abreu & Carapuça, 1994) |
| Method Inheritance Factor | (Abreu & Carapuça, 1994) |

| | |
|---|---|
| Method-to-Method Export Coupling | (Sahraoui, Goding & Miceli, 2000) |
| Net Promoter Score | (Reichheld & Frederick, 2003) |
| Non Compliance Index | (Padmini, Bandara & Perera, 2015) |
| Normalized Amount of Code Smell Occurrences | (Fowler & Beck, 1999) |
| Normalized Distance from Main Sequence | (Oliveira, Redin, Carro, Da Cunha & Wagner, 2008) |
| Number of Bounce Backs | (Middleton, Taylor, Flaxel & Cookson, 2007) |
| Number of Generated Files | (He, Avgeriou, Liang & Li, 2016) |
| Number of Inherited Methods per Class | (Henderson-Sellers, 1996) |
| Number of Manually Created Files | (He, Avgeriou, Liang & Li, 2016) |
| Number of Modified Generated Files | (He, Avgeriou, Liang & Li, 2016) |
| Number of Overridden Methods per Class | (Henderson-Sellers, 1996) |
| Number of Static Methods per Class | (Oliveira, Redin, Carro, Da Cunha & Wagner, 2008) |
| Number of Static Variables per Class | (Oliveira, Redin, Carro, Da Cunha & Wagner, 2008) |
| Percentage of Adopted Work | (Downey & Sutherland, 2013) |
| Percentage of Found Work | (Downey & Sutherland, 2013) |
| Percentage of Modified Generated Files | (He, Avgeriou, Liang & Li, 2016) |
| Percentage of Modified Generated Lines of Code | (He, Avgeriou, Liang & Li, 2016) |
| Polymorphism Factor | (Abreu & Carapuça, 1994) |
| Predictive Object Points | (Minkiewicz, 1997) |
| Process Efficiency | (Sutherland, Harrison & Riddle, 2014) |
| Processing Time | (Krafcik, 1988) |
| Queue Time | (Krafcik, 1988) |
| Regression Test Cycle Time | (Manila, 2013) |
| Response for a Class | (Chidamber & Kemerer, 1994) |
| Reuse Ratio | (Henderson-Sellers, 1996) |
| Running Tested Features | (Abbas, Gravell & Wills, 2010) |
| Schedule Performance Index | (Mahnic & Vrana, 2007) |
| Self-Assigned Happiness | (Sutherland, Harrison & Riddle, 2014) |
| Smoke Test Cycle Time | (Manila, 2013) |
| Specialization Index | (Henderson-Sellers, 1996) |
| Story Point Velocity | (Downey & Sutherland, 2013) |
| Success at Scale | (Downey & Sutherland, 2013) |
| System Analysis Cost | (Leung & White, 1991) |
| Targeted Value Increase | (Downey & Sutherland, 2013) |
| Task Time | (Krafcik, 1988) |
| Technical Efficiency | (Grimaldo, Perrotta, Corvello & Verteramo, 2016) |
| Test Execution Cost | (Leung & White, 1991) |
| Test Result Analysis Cost | (Leung & White, 1991) |

| | |
|---|---|
| Test Selection Cost | (Leung & White, 1991) |
| Throughput | (Krafcik, 1988) |
| Thumbs Up Rule | (Grimaldo, Perrotta, Corvello & Verteramo, 2016) |
| Time to Market in Days | (Stata, 1980) |
| Value Added Time | (Krafcik, 1988) |
| Value Delivered Over Time | (Petersen & Wohlin, 2011) |
| Weighted Method per Class | (Chidamber & Kemerer, 1994) |
| Win Loss Record | (Downey & Sutherland, 2013) |
| Work Capacity | (Downey & Sutherland, 2013) |
| Work Effectiveness | (Mahnic & Vrana, 2007) |
| Yesterday's Weather | (Downey & Sutherland, 2013) |

# Appendix E - Metrics per Paper

| Paper | Encountered Metrics |
|---|---|
| P1.01 | Maximum Amount of Team Members, Hours per Function Point, Function Points |
| P1.02 | Function Points, Predictive Object Points, Weighted Methods per Class, Person Hours, Amount of Team Members |
| P1.03 | Delivery on Time, Work Capacity, Unit Test Coverage, Percentage of Adopted Work, Mean Time to Recovery, Lines of Code per Unit of Time, Halstead Complexity Metrics, Cyclomatic Complexity, Defect Density, Story Point Velocity, Focus Factor, Percentage of Found Work, Accuracy of Forecast, Targeted Value Increase, Success at Scale, Win/Loss Record, Smoke Test Cycle Time, Regression Test Cycle Time, Defect count, Faults Slip-Through, Lead Time, Work in Progress, Queue Time, Cost of Quality, Defect Severity Index, Open Defect Severity Index, Defect Slippage Rate, Requirement Clarity Index, Sprint Level Effort Burndown, Non Compliance Index, Accuracy of Estimation, Net Promoter Score |
| P1.04 | Sprint Level Effort Burndown, Story Point Velocity, Work Capacity, Focus Factor, Percentage of Adopted Work, Percentage of Found Work, Accuracy of Estimation, Accuracy of Forecast, Targeted Value Increase, Success at Scale, Win/Loss Record, Function Points |
| P1.05 | Net Present Value, Story Point Velocity, Enterprise Velocity, Person Hours |
| P1.06 | Ideal Days, Sprint Level Effort Burndown, Story Point Velocity, Defects per Iteration, Amount of Tests, Standard Violations |
| P1.07 | Story Point Velocity, Lines of Code per Unit of Time, Function Points, Person Months, Person Hours, Open Defect Count, Unit Test Coverage, Hours per Story Point |
| P1.08 | Story Point Velocity, Yesterday's Weather, Mean Time to Recovery, Self-Assigned Happiness, Process Efficiency |
| P1.09 | Person Hours, Lines of Code per Unit of Time, New Classes per Release, New Methods per Release, New Features per Release, New Lines of Code per Release, Defects Fixed per Release |
| P1.10 | Function Points, Lines of Code per Unit of Time, Hours per Function Point |
| P1.11 | Lines of Code per Unit of Time, Person Hours, Lack of Cohesion of Methods, Weighted Methods per Class, Statements per Method, Response for Class |
| P1.12 | Maximum Amount of Team Members, Function Points, Person Hours, Person Months, Story Point Velocity, Defect Count |
| P1.13 | Work in Progress |
| P1.14 | Lines of Code per Unit of Time, Halstead Complexity Metrics, Cyclomatic Complexity, Function Points |
| P1.15 | Lines of Code per Unit of Time, Lack of Cohesion of Methods, Cyclomatic Complexity |
| P1.16 | Cyclomatic Complexity, Person Months, Defects per Iteration, Amount of Tests, Unit Test Coverage |
| P1.17 | Lines of Code per Unit of Time, Cyclomatic Complexity, Duplicate Expressions, Lack of Cohesion of Methods |
| P1.18 | Lines of Code per Unit of Time, Cyclomatic Complexity, Duplicate Expressions, Lack of Cohesion of Methods |
| P1.19 | Lines of Code per Unit of Time, Amount of Tests |

| | |
|---|---|
| P1.20 | Maximum Amount of Team Members, Amount of Team Members, Work Capacity, Ideal Capacity, Person Hours, Kick-Off Days, Technical Efficiency, Internal Efficiency, Number of Scrum Teams per Project, Interrupted Time |
| P1.21 | Work in Progress, Average Work in Progress, Maximum Work in Progress, Task Time, Lead Time, Queue Time, Defects per Iteration |
| P1.22 | Lack of Cohesion of Methods, Defect Count, Lines of Code per Unit of Time, Depth of Inheritance Tree, Coupling Concentration Index |
| P1.23 | Cycle Time, Value Added Time, Lead Time, First Time Yield, Average Work in Progress |
| P1.24 | Story Point Velocity, Sprint Level Effort Burndown, Test Pass Rate, Defect Count, Amount of Tests, Running Tested Features, Work in Progress, Critical Defects Sent by Customers, Open Defect Count, Test Failure Rate, Test Pass Rate, Remaining Task Effort, Team Effectiveness, Check-Ins per Day , Defects per Iteration, Number of Defects Found by Tests, Net Promoter Score, Revenue per Customer, Cycle Time, Business Value Delivered, Mean Time to Recovery, Unit Test Coverage, Test Growth Ratio, Standard Violations, Release Level Effort Burndown, Cost Performance Index, Common Tempo Time, Number of Bounce Backs, Customer Satisfaction, Lead Time, Processing Time, Queue Time, Change Requests per Requirement, Defect Slippage Rate, Implemented Versus Wasted Requirements, Number of Requests from Customers, Requirements Inventory Size, Number of Requirements per Feature, Throughput, Percentage of Completed Stories, Load Factor, Actual Development Time, Due Date Performance, Flow Efficiency |
| P1.25 | Lines of Code per Unit of Time, Function Points, Time to Market in Days, Cost per Function Point, Cost per Story Point, Amount of Team Members |
| P1.26 | Lead Time, Lines of Code per Unit of Time, Work in Progress, Maximum Work in Progress, Queue Time, Churn |
| P2.01 | Lines of Code per Unit of Time, Weighted Methods per Class, Methods per Class, Number of Children, Depth of Inheritance Tree, Lines of Code per Method, Coupling Between Objects, Number of Instance Variables per Class, Predictive Object Points |
| P2.02 | Stories per Day per Developer, Cost per Iteration, Tests per Story, Defects per Story, Test Runtime, Manual Tests per Story, Mean Time to Recovery, Average Amount of Defects Carried to Next Iteration, Unit Test Coverage, Builds per Day, Build Runtime, Percentage of Successful Builds, Cycle Time, Net Present Value, Return on Investment, Story Point Velocity, Ideal Days, Number of Stories per Iteration, Amount of Open Work Items, Average Number of Stories Added to Iteration, Average Number of Stories Removed from Iteration, Cyclomatic Complexity, Lines of Code per Unit of Time, Running Tested Features, Defect Slippage Rate, Duplicate Expressions, Percentage of Dead Code |
| P2.03 | Parameters per Method, Amount of Cycles in Dependency Graph |
| P2.04 | Mean Time to Recovery, Amount of Lines of Code, Defect Density, Average Design Review Rate, Average Code Review Rate |
| P2.05 | Cyclomatic Complexity, Amount of Lines of Code, Comment Percentage, Weighted Methods per Class, Response for Class, Lack of Cohesion of Methods, Coupling Between Objects, Depth of Inheritance Tree, Number of Children |
| P2.06 | Messages per Method, Statements per Method, Lines of Code per Method, Weighted Methods per Class, Number of Instance Variables per Class, Depth of Inheritance Tree, Number of Children, Number of Inherited Methods per Class, Number of Overridden Methods per Class |
| P2.07 | Average Class-to-Leaf Depth, Number of Children, Number of Overridden Methods per Class, Number of Inherited Methods per Class, Number of Methods Added per Class, |

| | |
|---|---|
| | Specialization Index, Coupling Between Objects, Data Abstraction Coupling, Information-Flow Based Inheritance Coupling, Class Attribute Import Coupling, Descendant Method-to-Method Export Coupling, Method-to-Method Export Coupling |
| P2.08 | Common Tempo Time, Capacity Utilization, Delivery on Time, Cost Efficiency, Value Delivered over Time, Value Added Time |
| P2.09 | Number of Generated Files, Number of Modified Generated Files, Number of Manually Created Files, Percentage of Modified Generated Files, Amount of Lines of Generated Code, Amount of Modified Lines of Generated Code, Amount of Manually Created Lines of Code, Percentage of Modified Generated Lines of Code, Amount of Code Smell Occurrences, Normalized Amount of Code Smell Occurrences |
| P2.10 | System Analysis Cost, Test Selection Cost, Test Execution Cost, Test Result Analysis Cost |
| P2.11 | Business Value Delivered, Story Point Velocity, Return on Investment, Internal Rate of Return, Cashflow per Iteration, Net Present Value |
| P2.12 | Story Point Velocity, Work Effectiveness, Schedule Performance Index, Defect Density, Cost of Failure to Control, Fulfilment of Scope, Average Overtime per Sprint, Average Projects per Employee, Impediments per Work-Item, Mean Time to Recovery, Personnel Turnover |
| P2.13 | Comment Percentage, Amount of Lines of Code, Weighted Methods per Class, Depth of Inheritance Tree, Number of Children, Coupling Between Objects, Response for Class, Lack of Cohesion of Methods, Efferent Coupling, Afferent Coupling, Code Instability, Code Abstractness, Normalized Distance from Main Sequence |
| P2.14 | Faults Slip-Through, Fault Latency, Improvement Potential, Average Fault Cost |
| P2.15 | Access to Foreign Data, Locality of Attribute Accesses, Foreign Data Providers, Weighted Methods per Class, Lack of Cohesion of Methods, Amount of Lines of Code, Lines of Code per Method, Cyclomatic Complexity, Halstead Complexity Metrics, Parameters per Method |
| P2.16 | Index of Inter-Package Usage, Index of Inter-Package Extending, Index of Package Changing Impact, Index of Inter-Package Usage Diversion, Index of Inter-Package Extending Diversion, Index of Package Goal Focus, Average Number of Modified Components per Commit |
| P2.17 | Afferent Coupling, Efferent Coupling, Code Instability, Lack of Cohesion of Methods, Code Abstractness, Normalized Distances from Main Sequence, Depth of Inheritance Tree, Number of Overridden Methods per Class, Number of Instance Variables per Class, Number of Classes, Number of Interfaces, Methods per Class, Number of Packages, Parameters per Method, Number of Static Variables per Class, Number of Static Methods per Class, Amount of Lines of Code, Cyclomatic Complexity, Lines of Code per Method, Maximum Nested Block Depth, Weighted Methods per Class |
| P2.18 | Response for Class, Number of Instance Variables per Class, Methods per Class, Weighted Methods per Class, Coupling Between Objects, Data Abstraction Coupling, Message Passing Coupling, Coupling Factor, Lack of Cohesion of Methods, Information Based Cohesion, Method Hiding Factor, Attribute Hiding Factor, Number of Children, Depth of Inheritance Tree, Method Inheritance Factor, Attribute Inheritance Factor, Number of Overridden Methods per Class, Polymorphism Factor, Reuse Ratio, Specialization Ratio |

# Appendix F - Papers per Metric

| Metric | Papers |
|---|---|
| Access to Foreign Data | P2.15 |
| Accuracy of Estimation | P1.03, P1.04 |
| Accuracy of Forecast | P1.03, P1.04 |
| Actual Development Time | P1.24 |
| Afferent Coupling | P2.13, P2.17 |
| Amount of Code Smell Occurrences | P2.09 |
| Amount of Cycles in Dependency Graph | P2.03 |
| Amount of Lines of Generated Code | P2.09 |
| Amount of Manually Created Lines of Code | P2.09 |
| Amount of Modified Lines of Generated Code | P2.09 |
| Attribute Hiding Factor | P2.18 |
| Attribute Inheritance Factor | P2.18 |
| Average Amount of Defects Carried to Next Iteration | P2.02 |
| Average Class-to-Leaf Depth | P2.07 |
| Average Code Review Rate | P2.04 |
| Average Design Review Rate | P2.04 |
| Average Fault Cost | P2.14 |
| Average Number of Modified Components per Commit | P2.16 |
| Average Number of Stories Added to Iteration | P2.02 |
| Average Number of Stories Removed to Iteration | P2.02 |
| Average Overtime per Day | |
| Average Overtime per Sprint | P2.12 |
| Average Projects per Employee | P2.12 |
| Average Work in Progress | P1.21, P1.23 |
| Build Runtime | P2.02 |
| Builds per Day | P2.02 |
| Business Value Delivered | P1.24, P2.11 |
| Capacity Utilization | P2.08 |
| Cashflow per Iteration | P2.11 |
| Change Requests per Requirement | P1.24 |
| Check-Ins per Day | P1.24 |

| | |
|---|---|
| Churn | P1.26 |
| Class Attribute Import Coupling | P2.07 |
| Code Abstractness | P2.13, P2.17 |
| Code Instability | P2.13, P2.17 |
| Comment Percentage | P2.05, P2.13 |
| Common Tempo Time | P1.24, P2.08 |
| Cost Efficiency | P2.08 |
| Cost of Quality | P1.03 |
| Cost Performance Index | P1.24 |
| Cost per Function Point | P1.25 |
| Cost per Iteration | P2.02 |
| Cost per Story Point | P1.25 |
| Coupling Between Objects | P2.01, P2.05, P2.07, P2.13, P2.18 |
| Coupling Concentration Index | P1.22 |
| Coupling Factor | P2.18 |
| Critical Defects Sent by Customers | P1.24 |
| Cycle Time | P1.23, P1.24, P2.02 |
| Cyclomatic Complexity Metric | P1.03, P1.14, P1.15, P1.16, P1.17, P1.18, P2.02, P2.05, P2.15, P2.17 |
| Data Abstraction Coupling | P2.07, P2.18 |
| Defect Count | P1.03, P1.12, P1.22, P1.24 |
| Defect Density | P1.03, P2.04, P2.12 |
| Defect Severity Index | P1.03 |
| Defects Fixed Per Release | P1.09 |
| Defect Slippage Rate | P1.03, P1.24, P2.02 |
| Defects Per Iteration | P1.06, P1.16, P1.21, P1.24 |
| Defects per Story | P2.02 |
| Delivery on Time | P1.03, P2.08 |
| Depth of Inheritance Tree | P1.22, P2.01, P2.05, P2.06, P2.13, P2.17, P2.18 |
| Descendant Method-to-Method Export Coupling | P2.07 |
| Due Date Performance | P1.24 |
| Duplicate Expressions | P1.17, P1.18, P2.02 |
| Efferent Coupling | P2.13, P2.17 |
| Enterprise Velocity | P1.05 |
| Fault Latency | P2.14 |
| Faults Slip Through | P1.03, P2.14 |
| First Time Yield | P1.23 |
| Flow Efficiency | P1.24 |
| Focus Factor | P1.03, P1.04 |
| Foreign Data Providers | P2.15 |
| Fulfilment of Scope | P2.12 |
| Halstead Complexity Metric | P1.03, P1.14, P2.15 |

| | |
|---|---|
| Hours per Function Point | P1.01, P1.10 |
| Hours per Story Point | P1.07 |
| Ideal Capacity | P1.20 |
| Ideal Days | P1.06, P2.02 |
| Impediments per Work-Item | P2.12 |
| Implemented Versus Wasted Requirements | P1.24 |
| Improvement Potential | P2.14 |
| Index of Inter-Package Extending | P2.16 |
| Index of Inter-Package Extending Diversion | P2.16 |
| Index of Inter-Package Usage | P2.16 |
| Index of Inter-Package Usage Diversion | P2.16 |
| Index of Package Changing Impact | P2.16 |
| Index of Package Goal Focus | P2.16 |
| Information-Based Cohesion | P2.18 |
| Information-Flow Based Inheritance Coupling | P2.07 |
| Internal Efficiency | P1.20 |
| Internal Rate of Return | P2.11 |
| Impediments | P1.20 |
| Kick-Off Days | P1.20 |
| Lack of Cohesion of Methods | P1.11, P1.15, P1.17, P1.18, P1.22, P2.05, P2.13, P2.15, P2.17, P2.18 |
| Lead Time | P1.03, P1.21, P1.23, P1.24, P1.26 |
| Lines of Code per Method | P2.01, P2.06, P2.15, P2.17 |
| Lines of Code (per Unit of Time) | P1.03, P1.07, P1.09, P1.10, P1.11, P1.14, P1.15, P1.17, P1.18, P1.19, P1.22, P1.25, P1.26, P2.01, P2.02 |
| Load Factor | P1.24 |
| Locality of Attribute Accesses | P2.15 |
| Manual Tests per Story | P2.02 |
| Maximum Amount of Team Members | P1.01, P1.12, P1.20 |
| Maximum Nested Block Depth | P2.17 |
| Maximum Work in Progress | P1.21, P1.26 |
| Mean Time to Recovery | P1.03, P1.08, P1.24, P2.02, P2.04, P2.12 |
| Message Passing Coupling | P2.18 |
| Messages per Method | P2.06 |
| Method Hiding Factor | P2.18 |
| Method Inheritance Factor | P2.18 |
| Method-to-Method Export Coupling | P2.07 |
| Net Present Value | P1.05, P2.02, P2.11 |

| | |
|---|---|
| Net Promoter Score | P1.03, P1.24 |
| New Classes Per Release | P1.09 |
| New Features Per Release | P1.09 |
| New Lines of Code Per Release | P1.09 |
| New Methods Per Release | P1.09 |
| Non Compliance Index | P1.03 |
| Normalized Amount of Code Smell Occurrences | P2.09 |
| Normalized Distance from Main Sequence | P2.13, P2.17 |
| Number of Bounce Backs | P1.24 |
| Number of Classes | P2.17 |
| Number of Defects Found by Tests | P1.24 |
| Number of Generated Files | P2.09 |
| Number of Inherited Methods per Class | P2.06, P2.07 |
| Number of Interfaces | P2.17 |
| Number of Manually Created Files | P2.09 |
| Number of Methods Added per Class | P2.07 |
| Number of Modified Generated Files | P2.09 |
| Number of Overridden Methods per Class | P2.06, P2.07, P2.17, P2.18 |
| Number of Packages | P2.17 |
| Number of Requests From Customers | P1.24 |
| Number of Requirements per Feature | P1.24 |
| Number of Scrum Teams on Project | P1.20 |
| Number of Static Methods per Class | P2.17 |
| Number of Static Variables per Class | P2.17 |
| Number of Stories per Iteration | P2.02 |
| Open Defect Count | P1.07, P1.24 |
| Open Defect Severity Index | P1.03 |
| Parameters per Method | P2.03, P2.15, P2.17 |
| Percentage of Adopted Work | P1.03, P1.04 |
| Percentage of Completed Stories | P1.24 |
| Percentage of Dead Code | P2.02 |
| Percentage of Found Work | P1.03, P1.04 |
| Percentage of Modified Generated Files | P2.09 |

| | |
|---|---|
| Percentage of Modified Generated Lines of Code | P2.09 |
| Percentage of Successful Builds | P2.02 |
| Person Hours | P1.02, P1.05, P1.07, P1.09, P1.11, P1.12, P1.20 |
| Person Months | P1.07, P1.12, P1.16 |
| Personnel Turnover | P2.12 |
| Polymorphism Factor | P2.18 |
| Predictive Object Points | P1.02, P2.01 |
| Process Efficiency | P1.08 |
| Processing Time | P1.24 |
| Queue Time | P1.03, P1.21, P1.24, P1.26 |
| Regression Test Cycle Time | P1.03 |
| Release Level Effort Burndown | P1.24 |
| Remaining Task Effort | P1.24 |
| Requirement Clarity Index | P1.03 |
| Requirements Inventory Size | P1.24 |
| Response for a Class | P1.11, P2.05, P2.13, P2.18 |
| Return on Investment | P2.02, P2.11 |
| Reuse Ratio | P2.18 |
| Revenue per Customer | P1.24 |
| Running Tested Features | P1.24, P2.02 |
| Schedule Performance Index | P1.24, P2.12 |
| Self-Assigned Happiness | P1.08 |
| Smoke Test Cycle Time | P1.03 |
| Specialization Index | P2.07 |
| Reuse Ratio | P2.18 |
| Sprint Level Effort Burndown | P1.03, P1.04, P1.06, P1.24 |
| Standard Violations | P1.06, P1.24 |
| Statements per Method | P1.11, P2.06 |
| Stories per Day per Developer | P2.02 |
| Story Point Velocity | P1.03, P1.04, P1.05, P1.06, P1.07, P1.08, P1.12, P1.24, P2.02, P2.11, P2.12 |
| Success at Scale | P1.03, P1.04 |
| System Analysis Cost | P2.10 |
| Targeted Value Increase | P1.03, P1.04 |
| Task Time | P1.21 |
| Team Effectiveness | P1.24 |
| Technical Efficiency | P1.20 |
| Test Execution Cost | P2.10 |
| Test Failure Rate | P1.24 |
| Test Growth Ratio | P1.24 |
| Test Pass Rate | P1.24, P1.24 |
| Test Result Analysis Cost | P2.10 |

| Test Runtime | P2.02 |
|---|---|
| Test Selection Cost | P2.10 |
| Tests per Story | P2.02 |
| Throughput | P1.24 |
| Thumbs Up Rule | P1.03 |
| Time to Market in Days | P1.25 |
| Unit Test Coverage | P1.03, P1.07, P1.16, P1.24, P2.02 |
| Value Added Time | P1.23, P2.08 |
| Value Delivered Over Time | P2.08 |
| Weighted Method per Class | P1.02, P1.11, P2.01, P2.05, P2.06, P2.13, P2.15, P2.17, P2.18 |
| Win Loss Record | P1.03, P1.04 |
| Work Capacity | P1.03, P1.04, P1.20 |
| Work Effectiveness | P2.12 |
| Work In Progress | P1.03, P1.13, P1.21, P1.24, P1.26 |
| Yesterday’s Weather | P1.08 |

# Appendix G - Full Aspect Encoding

| Aspect Encoding | | Metric |
|---|---|---|
| **Axial Encoding** | **Open Encoding** | |
| **Efficiency** | **Time** | Actual Development Time |
| | | Hours per Function Point |
| | | Hours per Story Point |
| | | Ideal Capacity |
| | | Ideal Days |
| | | Interrupted Time |
| | | Lead Time |
| | | Load Factor |
| | | Person Hours |
| | | Person Months |
| | | Processing Time |
| | | Queue Time |
| | | Task Time |
| | | Technical Efficiency |
| | | Value Added Time |
| | **Rework** | Average Amount of Defects Carried to Next Iteration |
| | | Defects Fixed Per Release |
| | | Mean Time to Recovery |
| | **Cycle Times** | Build Runtime |
| | | Builds per Day |
| | | Check-Ins per Day |
| | | Common Tempo Time |
| | | Cycle Time |
| | | Regression Test Cycle Time |
| | | Smoke Test Cycle Time |
| | | Test Runtime |
| | | Time to Market in Days |
| | **Delivery** | Delivery on Time |
| | | Due Date Performance |
| | | Enterprise Velocity |
| | | Stories per Day per Developer |
| | | Story Point Velocity |
| | | Targeted Value Increase |
| | | Throughput |
| | | Yesterday's Weather |
| | **Flow** | Flow Efficiency |

| | | |
|---|---|---|
| | | Process Efficiency |
| | **Impediments** | Impediments per Work-Item |
| | | Interrupted Time |
| | **Burndown** | Release Level Effort Burndown |
| | | Remaining Task Effort |
| | | Sprint Level Effort Burndown |
| | **Presumed** | Internal Efficiency |
| | | Kick-Off Days |
| | | Team Effectiveness |
| | | Work Effectiveness |
| | **Effort** | Focus Factor |
| | | Lines of Code (per Unit of Time) |
| | | New Classes Per Release |
| | | New Features Per Release |
| | | New Lines of Code Per Release |
| | | New Methods Per Release |
| | | Number of Stories per Iteration |
| | | Percentage of Completed Stories |
| | **Cohesion** | Index of Package Goal Focus |
| | | Information-Based Cohesion |
| | | Lack of Cohesion of Methods |
| | | Parameters per Method |
| | **Coupling** | Access to Foreign Data |
| | | Afferent Coupling |
| | | Average Number of Modified Components per Commit |
| | | Class Attribute Import Coupling |
| | | Code Instability |
| | | Coupling Between Objects |
| | | Coupling Concentration Index |
| | | Coupling Factor |
| | | Data Abstraction Coupling |
| | | Descendant Method to Method Export Coupling |
| | | Efferent Coupling |
| | | Foreign Data Providers |
| | | Index of Inter-Package Usage |
| | | Index of Inter-Package Usage Diversion |
| | | Index of Package Changing Impact |
| | | Information-Flow Based Inheritance Coupling |
| | | Locality of Attribute Accesses |
| | | Message Passing Coupling |
| | | Method to Method Export Coupling |

| | | |
|---|---|---|
| **Complexity** | | Normalized Distance from Main Sequence |
| | | Response for a Class |
| | | Weighted Method per Class |
| | **Dependencies** | Amount of Cycles in Dependency Graph |
| | **Code Generation** | Amount of Lines of Generated Code |
| | | Amount of Manually Created Lines of Code |
| | | Amount of Modified Lines of Generated Code |
| | | Number of Generated Files |
| | | Number of Manually Created Files |
| | | Number of Modified Generated Files |
| | | Percentage of Modified Generated Files |
| | | Percentage of Modified Generated Lines of Code |
| | **Encapsulation** | Attribute Hiding Factor |
| | | Method Hiding Factor |
| | | Number of Interfaces |
| | **Inheritance** | Attribute Inheritance Factor |
| | | Average Class-to-Leaf Depth |
| | | Code Abstractness |
| | | Depth of Inheritance Tree |
| | | Descendant Method to Method Export Coupling |
| | | Index of Inter-Package Extending |
| | | Index of Inter-Package Extending Diversion |
| | | Information-Flow Based Inheritance Coupling |
| | | Method Inheritance Factor |
| | | Number of Inherited Methods per Class |
| | | Number of Methods Added per Class |
| | | Number of Overridden Methods per Class |
| | | Polymorphism Factor |
| | | Reuse Ratio |
| | | Specialization Index |
| | | Specialization Ratio |
| | **Cyclomatic Complexity** | Cyclomatic Complexity Metric |
| | | Halstead Complexity Metric |
| | **Expression Tree** | Duplicate Expressions |
| | | Lines of Code per Method |
| | | Maximum Nested Block Depth |
| | | Messages per Method |
| | | Statements per Method |
| **Risk** | **Clarity** | Requirements Clarity Index |
| | | Success at Scale |
| **Size** | **Effort** | Percentage of Adopted Work |
| | | Percentage of Found Work |

| | | |
|---|---|---|
| | | Lines of Code (per Unit of Time) |
| | | Lines of Code per Method |
| | | Number of Classes |
| | **Components** | Number of Interfaces |
| | | Number of Packages |
| | | Number of Static Methods per Class |
| | | Number of Static Variables per Class |
| | | Accuracy of Estimation |
| | | Accuracy of Forecast |
| | **Estimation** | Percentage of Adopted Work |
| | | Percentage of Found Work |
| | | Predictive Object Points |
| | | Churn |
| | | New Classes Per Release |
| | **Code Churn** | New Features Per Release |
| | | New Lines of Code Per Release |
| | | New Methods Per Release |
| | | Test Growth Ratio |
| | **Fulfilment** | Fulfilment of Scope |
| | | Percentage of Completed Stories |
| **Quality** | **Anti-Patterns** | Amount of Code Smell Occurrences |
| | | Halstead Complexity Metric |
| | | Non Compliance Index |
| | | Normalized Amount of Code Smell Occurrences |
| | | Normalized Distance from Main Sequence |
| | | Parameters per Method |
| | | Percentage of Dead Code |
| | | Standard Violations |
| | **Defects** | Average Fault Cost |
| | | Critical Defects Sent by Customers |
| | | Defect Count |
| | | Defect Density |
| | | Defect Severity Index |
| | | Defect Slippage Rate |
| | | Defects Per Iteration |
| | | Defects per Story |
| | | Fault Latency |
| | | Faults Slip Through |
| | | Improvement Potential |
| | | Number of Bounce Backs |
| | | Number of Defects Found by Tests |
| | | Open Defect Count |

| | | |
|---|---|---|
| | | Open Defect Severity Index |
| | | Percentage of Successful Builds |
| | | First Time Yield |
| | **Documentation** | Comment Percentage |
| | **Tests** | Manual Tests per Story |
| | | Number of Defects Found by Tests |
| | | Running Tested Features |
| | | Test Failure Rate |
| | | Test Pass Rate |
| | | Tests per Story |
| | | Unit Test Coverage |
| **Composition** | **Team Composition** | Maximum Amount of Team Members |
| | | Personnel Turnover |
| | | Self-Assigned Happiness |
| | | Work Capacity |
| | **Project Composition** | Number of Scrum Teams on Project |
| **Cost** | **Cost of Performed Work** | Cost Efficiency |
| | | Cost per Function Point |
| | | Cost per Iteration |
| | | Cost per Story Point |
| | | Cost Performance Index |
| | | Value Delivered Over Time |
| | **Cost of Performed Rework** | Average Fault Cost |
| | | Faults Slip Through |
| | | Improvement Potential |
| | **Cost of Quality** | Cost of Quality |
| | | System Analysis Cost |
| | | Test Execution Cost |
| | | Test Result Analysis Cost |
| | | Test Selection Cost |
| | **Financial** | Cashflow per Iteration |
| | | Internal Rate of Return |
| | | Net Present Value |
| | | Return on Investment |
| | | Revenue per Customer |
| | | Schedule Performance Index |
| **Design** | **Requirements** | Change Requests per Requirement |
| | | Implemented Versus Wasted Requirements |
| | | Number of Requirements per Feature |
| | | Requirements Inventory Size |
| | **Reviews** | Average Code Review Rate |
| | | Average Design Review Rate |

| | | |
|---|---|---|
| **Process** | **Story** | Average Work in Progress |
| | | Maximum Work in Progress |
| | | Work in Progress |
| | **Iteration** | Average Number of Stories Added to Iteration |
| | | Average Number of Stories Removed to Iteration |
| | **Team Member** | Average Overtime per Day |
| | | Average Overtime per Sprint |
| | **Project** | Average Projects per Employee |
| | | Business Value Delivered |
| | | Capacity Utilization |
| | **Requirements** | Number of Requests From Customers |
| **Satisfaction** | **Satisfaction** | Net Promoter Score |
| | | Thumbs Up Rule |

# Appendix H - Full Input Encoding

| Input Encoding | | Input |
|---|---|---|
| **Axial Encoding** | **Open Encoding** | |
| **Backlog** | **Backlog** | Amount of Open Defects |
| | | Change Request Time Created |
| | | Amount of Work Items |
| | | Amount of Stories in Iteration |
| | | Amount of Defects |
| | | Amount of Open Work Items |
| **Company** | **Company** | Amount of Customers |
| **Defects** | **Defect Counts** | Amount of Defects |
| | | Defective Units Produced |
| | | Amount of Open Defects |
| | **Defect Cost** | Defect Severity |
| | | Defect Cost |
| | **Defect Discovery** | Defect Discovery Environment |
| **Deployment** | **Build** | Build Cycle Time |
| | | Build Status |
| | **Version Control** | Commit Timestamp |
| **Estimate** | **Size Estimate** | Work Item Story Point Estimate |
| | | Sprint Story Point Original Forecast |
| | | Work Item Use Case Point Estimate |
| | | Adjusted Work Item Story Point Estimate |
| | | Work Item Function Point Estimate |
| | **Clarity Estimate** | Requirement Clarity |
| | **Commitment Estimate** | Adjusted Sprint Forecast |
| **Lifecycle** | **Day Lifecycle** | Planned Workday Start Timestamp |
| | | Planned Workday End Timestamp |
| | | Workday Start Timestamp |
| | | Workday End Timestamp |
| | **Interruption Lifecycle** | Interruption Type |
| | | Interruption End Timestamp |
| | | Interruption Start Timestamp |
| | **Iteration Lifecycle** | Sprint Start Timestamp |
| | | Sprint End Timestamp |
| | **Product Lifecycle** | Commit Timestamp |
| | | Release Date |
| | **Team Lifecycle** | Team Members Added |
| | | Team Members Removed |

| | | |
|---|---|---|
| | **Test Lifecycle** | Test Created Timestamp |
| | | Test Deleted Timestamp |
| | **Work Item Lifecyle** | Planned Work Item Finished Time |
| | | Work Item Start Timestamp |
| | | Work Item Created Timestamp |
| | | Work Item Finished Timestamp |
| | | Work Item Deployed Timestamp |
| **Financial** | **Cost** | Cash Outflow |
| | | Cost of Actual Work Performed |
| | | Cost of Appraisal |
| | | Cost of Budgeted Work Performed |
| | | Cost of Control |
| | | Cost of External Failure |
| | | Cost of Failure Control |
| | | Cost of Internal Failure |
| | | Cost of Prevention |
| | | Defect Cost |
| | | Project Cost |
| | | Work Item Cost |
| | **Revenue** | Cash Inflow |
| | | Project Revenue |
| **Iteration** | **Commitment** | Adjusted Sprint Forecast |
| | | Sprint Story Point Original Forecast |
| | **Delivery** | Amount of Stories in Iteration |
| | | Features Added |
| | | Process Capacity |
| | | Units Produced |
| | **Lifecycle** | Sprint End Timestamp |
| | | Sprint Start Timestamp |
| **Schedule** | **Planned Production** | Scheduled Production |
| | | Process Capacity |
| | **Planning** | Planned Workday Start Timestamp |
| | | Amount of Available Workdays |
| | | Planned Workday End Timestamp |
| | **Unplanned** | Interruption End Timestamp |
| | | Interruption Start Timestamp |
| **Source Code** | **Code Churn** | Lines of Code Deleted |
| | | Lines of Code Added |
| | | Classes Added |
| | | Methods Added |
| | | Lines of Code Edited |
| | **Code Complexity** | Number of Distinct Operators |

| | | |
|---|---|---|
| | | Number of Distinct Operands |
| | | Number of Children |
| | | Total Number of Operators |
| | | Number of Instance Variables per Class |
| | | Total Number of Operands |
| | **Components** | Number of Top Level Classes |
| | | Methods per Class |
| | **Code Coupling** | Control Flow Graph |
| | | Inheritance Tree |
| **Survey** | **Customer Inquiry** | Customer Promoter Score |
| | **Team Member Inquiry** | Happiness Score |
| **Team** | **Team Churn** | Team Members Added |
| | | Team Members Removed |
| | **Team Composition** | Amount of Team Members |
| | | Assigned Project |
| | **Team Delivery** | Units Produced |
| **Test** | **Test Result** | Test Result |
| | **Test Lifecycle** | Test Deleted Timestamp |
| | | Test Created Timestamp |
| | **Test Count** | Amount of Tests |
| **Work Day** | **Day Lifecycle** | Workday Start Timestamp |
| | | Workday End Timestamp |
| | **Planning** | Amount of Available Workdays |
| **Work Item** | **Work Item Count** | Amount of Open Work Items |
| | | Amount of Work Items |
| | **Work Item Estimate** | Work Item Function Point Estimate |
| | | Work Item Story Point Estimate |
| | | Work Item Use Case Point Estimate |
| | | Adjusted Work Item Story Point Estimate |
| | **Work Item Financials** | Work Item Revenue |
| | | Work Item Cost |
| | **Work Item Lifecyle** | Work Item Finished Timestamp |
| | | Planned Work Item Finished Time |
| | | Work Item Deployed Timestamp |
| | | Work Item Start Timestamp |
| | | Work Item Created Timestamp |
| | **Work Item Meta Data** | Work Item Reporter |
| | | Work Item State |
| | | Work Item Resolve |
| | **Work Item Requirements** | Work Item Requirements |

# Appendix I - Keyword Occurrences

| Keyword | Occurrences |
|---|---|
| Productivity | 7 |
| Software Metrics | 6 |
| Software | 5 |
| Measurement | 5 |
| Software Engineering | 5 |
| Software Measurement | 5 |
| Programming | 4 |
| Costs | 4 |
| Companies | 4 |
| Testing | 4 |
| Agile | 3 |
| Refactoring | 3 |
| Scrum | 3 |
| Metrics | 3 |
| Large Scale Systems | 2 |
| Agile Software Development Process | 2 |
| Outsourcing | 2 |
| Lean | 2 |
| Case Study | 2 |
| Agile Development | 2 |
| Efficiency | 2 |
| Lean Manufacturing | 2 |
| Quality Assurance | 2 |
| Software Quality | 2 |
| Coupling | 2 |
| Computer Science | 1 |
| Computer Industry | 1 |
| Robustness | 1 |
| Data Engineering | 1 |
| Software Standards | 1 |
| Interviews | 1 |
| Traditional Software Development Process | 1 |
| Scrum Development Process | 1 |
| Indexes | 1 |
| Estimation | 1 |
| Accuracy | 1 |
| MySpace | 1 |
| Planning | 1 |

| | |
|---|---|
| Agile Metrics | 1 |
| CMMI | 1 |
| Myths About Agile | 1 |
| Tracking Tools | 1 |
| Progress Chart | 1 |
| Burndown Chart | 1 |
| Scrum Metrics | 1 |
| Project Management Tools | 1 |
| Global Communication | 1 |
| Geography | 1 |
| Project Management | 1 |
| Cultural Differences | 1 |
| Continents | 1 |
| Acceleration | 1 |
| Meteorology | 1 |
| Communities | 1 |
| Strong Productivity Gain | 1 |
| Agile Practice | 1 |
| Small Software Company | 1 |
| Industrial | 1 |
| Agile Software Practice | 1 |
| Productvitiy Metric | 1 |
| Software Process | 1 |
| Methodology | 1 |
| Software Development Metrics | 1 |
| Project Duration | 1 |
| Software Size | 1 |
| Effort | 1 |
| Kanban | 1 |
| Empirical Study | 1 |
| Evidence-Based Decision Making | 1 |
| Psychology | 1 |
| Tools | 1 |
| Complexity Theory | 1 |
| Insurance | 1 |
| Model Driven Development | 1 |
| Financial Applications | 1 |
| Agile Model-Driven Development Integration | 1 |
| Prioritization | 1 |
| Minimization | 1 |
| Optimization | 1 |
| Selection | 1 |

| Software Regression Testing | 1 |
|---|---|
| Offshoring | 1 |
| Software Development | 1 |
| Capacity-Based Model | 1 |
| Agile Methodologies | 1 |
| Delivery Process | 1 |
| Scalability | 1 |
| SAFe3.0 | 1 |
| Modeling | 1 |
| Operations Time Chart | 1 |
| Work in Progress | 1 |
| Efficacy | 1 |
| Lean Simulation | 1 |
| Lean Metrics | 1 |
| Systematic Literature Review | 1 |
| Internet Technology Metrics | 1 |
| Performance Measurement | 1 |
| Quality | 1 |
| Time to Market | 1 |
| Software Management | 1 |
| Product Lifecycle Management | 1 |
| Agile Manufacturing | 1 |
| Computer Bugs | 1 |
| Quality Management | 1 |
| Product Design | 1 |
| Lab-on-a-Chip | 1 |
| Computer Society | 1 |
| Job Shop Scheduling | 1 |
| Inspection | 1 |
| Software Performance | 1 |
| Business Continuity | 1 |
| Software Design | 1 |
| Software Evolution | 1 |
| Reverse Engineering | 1 |
| Object-Oriented Frameworks | 1 |
| Error Correction | 1 |
| Computer Aided Software Engineering | 1 |
| Error Detection | 1 |
| Object Oriented Methods | 1 |
| Development Flow | 1 |
| Goal-Question-Metric | 1 |
| Lean Software Development | 1 |

| | |
|---|---|
| Test Cost | 1 |
| Regression Testing | 1 |
| Aggregates | 1 |
| Shape Measurement | 1 |
| Software Development Management | 1 |
| Monitoring | 1 |
| Teamwork | 1 |
| IT Performance Measurement | 1 |
| AGIT | 1 |
| Agile Software Development | 1 |
| COBIT | 1 |
| IT Balanced Scoreboard | 1 |
| IT Indicators | 1 |
| Basis Path | 1 |
| Chidamber and Kemerer Metrics | 1 |
| Cyclomatic Complexity | 1 |
| Object-Oriented | 1 |
| Fault Latency | 1 |
| Software Process Improvement | 1 |
| Fault-Slip-Through | 1 |
| Early Fault Detection | 1 |
| Fault Metrics | 1 |
| Software Quality Evaluation | 1 |
| Code Smell Detection Tools | 1 |
| Code Smells | 1 |
| Architectural Technical Debt | 1 |
| Modularity Metric | 1 |
| Commit | 1 |
| Software Architecture | 1 |
| Unified Modeling Language | 1 |
| Embedded Software | 1 |
| Computer Languages | 1 |
| Embedded System | 1 |
| Software Systems | 1 |
| Computer Architecture | 1 |
| Informatics | 1 |
| Model Driven Engineering | 1 |
| Standards Development | 1 |
| Inheritance | 1 |
| Polymorphism | 1 |
| Object-Oriented Software | 1 |
| Information Hiding | 1 |

| Cohesion | 1 |
|---|---|

# Appendix J - Author Occurrences

| First Name | Last Name | Works |
|---|---|---|
| Jeff | Sutherland | 3 |
| Turgay | Aytac | 2 |
| Ayse | Bener | 2 |
| Ovunc | Bozcan | 2 |
| Gul | Calikli | 2 |
| Hessa | Alfraihi | 2 |
| Kevin | Leno | 2 |
| Shekoufeh | Kolahdouz-Rahimi | 2 |
| Howard | Haughton | 2 |
| Mohammadreza | Sharbaf | 2 |
| Claes | Wohlin | 2 |
| Paris | Avgeriou | 2 |
| Peng | Liang | 2 |
| Zengyang | Li | 2 |
| M | Ruhe | 1 |
| I | Wieczorek | 1 |
| R | Jefferey | 1 |
| Barry | Boehm | 1 |
| Chris | Abts | 1 |
| Sunita | Chulani | 1 |
| Indika | Perera | 1 |
| K | Padmini | 1 |
| H | Bandara | 1 |
| Scott | Downey | 1 |
| Daniel | Greening | 1 |
| Rana | Majumdar | 1 |
| Monika | Agarwal | 1 |
| Maurits | Rijk | 1 |
| Guido | Schoonheim | 1 |
| Neil | Harrison | 1 |
| Joel | Riddle | 1 |
| Frank | Maurer | 1 |
| Sebastien | Martel | 1 |
| Papatheocharous | Efi | 1 |
| Muhammad | Syed | 1 |
| Jaana | Nyfjord | 1 |
| Raimund | Moser | 1 |
| Witold | Pedrycs | 1 |

| | | |
|---|---|---|
| Giancarlo | Succi | 1 |
| Pekka | Abrahamsson | 1 |
| Alberto | Sillitti | 1 |
| Ajay | Rana | 1 |
| Mridul | Bhardwaj | 1 |
| Mariusz | Musial | 1 |
| Klaas-Jan | Stol | 1 |
| Brian | Fitzgerald | 1 |
| Michael | Felderer | 1 |
| Armin | Beer | 1 |
| Omar | Gómez | 1 |
| Glen | Rodríguez | 1 |
| Raúl | Rosero | 1 |
| Saverino | Verteramo | 1 |
| Vincenzo | Corvello | 1 |
| Carla | Vintro-Sanchez | 1 |
| Jordi | Fortuny-Santos | 1 |
| Lluis | Cuatrecasas-Arbos | 1 |
| Günes | Koru | 1 |
| Khaled | El Emam | 1 |
| Hamid | Seifoddini | 1 |
| Mohammed | Khadem | 1 |
| Sk | Ahad Ali | 1 |
| Juha | Itkonen | 1 |
| Mika | Mäntylä | 1 |
| Eetu | Kupiainen | 1 |
| Rini | van Sollingen | 1 |
| Hennie | Huijgens | 1 |
| Anders | Johnsen | 1 |
| Dag | Sjøberg | 1 |
| Jørgen | Solberg | 1 |
| Arlene | Minkiewicz | 1 |
| Nilay | Oza | 1 |
| Mikko | Korkala | 1 |
| Martin | Kunz | 1 |
| Reiner | Dumke | 1 |
| Niko | Zenker | 1 |
| Mark | Paulk | 1 |
| Chris | Kemerer | 1 |
| Lawrence | Hyatt | 1 |
| Linda | Rosenburg | 1 |
| Serge | Demeyer | 1 |

| Oscar | Nierstrasz | 1 |
|---|---|---|
| Stéphane | Ducasse | 1 |
| Thierry | Miceli | 1 |
| Houari | Sahraoui | 1 |
| Robert | Godin | 1 |
| Kay | Peterson | 1 |
| Xiao | He | 1 |
| Hareton | Leung | 1 |
| Lee | White | 1 |
| Deborah | Hartmann | 1 |
| Robin | Dymond | 1 |
| Viljan | Mahnic | 1 |
| Natasa | Zabkar | 1 |
| Santanu Kumar | Rath | 1 |
| Jayadeep | Pati | 1 |
| Yeresime | Suresh | 1 |
| Lars Ola | Damm | 1 |
| Lars | Lundberg | 1 |
| Marco | Zanoni | 1 |
| Pietro | Braione | 1 |
| Francesca Arcelli | Fontana | 1 |
| Nicolas | Guelfi | 1 |
| Luis | Lamb | 1 |
| Flavio Rech | Wagner | 1 |
| Luigi | Carro | 1 |
| Marcio | Oliveira | 1 |
| Ricardo Miotto | Redin | 1 |
| Yogesh | Singh | 1 |
| K | Aggarwal | 1 |
| Ruchika | Malhotra | 1 |
| Arvinder | Kaur | 1 |

# Appendix K - Metric Quality Assessment

| Name | Simple | Hard to Game | Outcome Oriented | Universal | Transparent |
|---|---|---|---|---|---|
| Access to Foreign Data | FALSE | FALSE | FALSE | FALSE | TRUE |
| Accuracy of Estimation | TRUE | TRUE | FALSE | TRUE | TRUE |
| Accuracy of Forecast | TRUE | FALSE | FALSE | TRUE | TRUE |
| Actual Development Time | TRUE | TRUE | TRUE | TRUE | TRUE |
| Afferent Coupling | FALSE | FALSE | FALSE | FALSE | TRUE |
| Amount of Code Smell Occurrences | FALSE | FALSE | FALSE | FALSE | FALSE |
| Amount of Cycles in Dependency Graph | FALSE | FALSE | FALSE | FALSE | TRUE |
| Amount of Lines of Generated Code | TRUE | FALSE | FALSE | FALSE | FALSE |
| Amount of Manually Created Lines of Code | TRUE | FALSE | FALSE | FALSE | FALSE |
| Amount of Modified Lines of Generated Code | TRUE | FALSE | FALSE | FALSE | FALSE |
| Attribute Hiding Factor | FALSE | FALSE | FALSE | FALSE | TRUE |
| Attribute Inheritance Factor | FALSE | FALSE | FALSE | FALSE | TRUE |
| Average Amount of Defects Carried to Next Iteration | TRUE | TRUE | FALSE | FALSE | TRUE |
| Average Class-to-Leaf Depth | TRUE | FALSE | FALSE | FALSE | TRUE |
| Average Code Review Rate | TRUE | TRUE | FALSE | FALSE | TRUE |
| Average Design Review Rate | TRUE | TRUE | FALSE | FALSE | TRUE |
| Average Fault Cost | FALSE | FALSE | FALSE | FALSE | FALSE |
| Average Number of Modified Components per Commit | FALSE | FALSE | FALSE | FALSE | FALSE |
| Average Number of Stories Added to Iteration | TRUE | TRUE | FALSE | TRUE | TRUE |
| Average Number of Stories Removed to Iteration | TRUE | TRUE | FALSE | TRUE | TRUE |
| Average Overtime per Day | TRUE | FALSE | FALSE | TRUE | TRUE |
| Average Overtime per Sprint | TRUE | FALSE | FALSE | TRUE | TRUE |
| Average Projects per Employee | TRUE | TRUE | FALSE | FALSE | TRUE |
| Average Work in Progress | TRUE | TRUE | TRUE | TRUE | TRUE |
| Build Runtime | TRUE | TRUE | FALSE | FALSE | TRUE |
| Builds per Day | TRUE | FALSE | FALSE | FALSE | TRUE |
| Business Value Delivered | TRUE | TRUE | TRUE | TRUE | FALSE |
| Capacity Utilization | TRUE | TRUE | TRUE | TRUE | FALSE |
| Cashflow per Iteration | TRUE | TRUE | TRUE | TRUE | FALSE |
| Change Requests per Requirement | FALSE | FALSE | FALSE | TRUE | FALSE |
| Check-Ins per Day | TRUE | FALSE | FALSE | FALSE | FALSE |
| Churn | TRUE | FALSE | FALSE | FALSE | TRUE |

| | | | | | |
|---|---|---|---|---|---|
| Class Attribute Import Coupling | FALSE | FALSE | FALSE | FALSE | FALSE |
| Code Abstractness | FALSE | FALSE | FALSE | FALSE | TRUE |
| Code Instability | FALSE | FALSE | FALSE | FALSE | TRUE |
| Comment Percentage | TRUE | FALSE | FALSE | FALSE | TRUE |
| Common Tempo Time | FALSE | FALSE | FALSE | TRUE | TRUE |
| Cost Efficiency | TRUE | FALSE | FALSE | FALSE | TRUE |
| Cost of Quality | FALSE | TRUE | TRUE | TRUE | FALSE |
| Cost Performance Index | FALSE | FALSE | FALSE | TRUE | FALSE |
| Cost per Function Point | TRUE | FALSE | TRUE | TRUE | TRUE |
| Cost per Iteration | TRUE | FALSE | FALSE | TRUE | TRUE |
| Cost per Story Point | TRUE | FALSE | TRUE | TRUE | TRUE |
| Coupling Between Objects | FALSE | FALSE | FALSE | FALSE | TRUE |
| Coupling Concentration Index | FALSE | TRUE | FALSE | FALSE | TRUE |
| Coupling Factor | FALSE | FALSE | FALSE | FALSE | FALSE |
| Critical Defects Sent by Customers | FALSE | FALSE | FALSE | TRUE | FALSE |
| Cycle Time | TRUE | TRUE | TRUE | TRUE | TRUE |
| Cyclomatic Complexity Metric | TRUE | FALSE | FALSE | FALSE | TRUE |
| Data Abstraction Coupling | FALSE | FALSE | FALSE | FALSE | TRUE |
| Defect Count | TRUE | FALSE | FALSE | TRUE | TRUE |
| Defect Density | TRUE | FALSE | FALSE | TRUE | TRUE |
| Defect Severity Index | TRUE | FALSE | TRUE | FALSE | TRUE |
| Defects Fixed Per Release | TRUE | FALSE | FALSE | TRUE | TRUE |
| Defect Slippage Rate | FALSE | FALSE | FALSE | TRUE | TRUE |
| Defects Per Iteration | FALSE | FALSE | FALSE | FALSE | TRUE |
| Defects per Story | TRUE | FALSE | TRUE | TRUE | TRUE |
| Delivery on Time | TRUE | FALSE | TRUE | TRUE | TRUE |
| Depth of Inheritance Tree | TRUE | FALSE | FALSE | FALSE | TRUE |
| Descendant Method-to-Method Export Coupling | FALSE | FALSE | FALSE | FALSE | FALSE |
| Due Date Performance | TRUE | FALSE | TRUE | TRUE | FALSE |
| Duplicate Expressions | TRUE | FALSE | FALSE | FALSE | FALSE |
| Efferent Coupling | FALSE | FALSE | FALSE | FALSE | TRUE |
| Enterprise Velocity | TRUE | FALSE | TRUE | TRUE | FALSE |
| Fault Latency | FALSE | FALSE | FALSE | FALSE | FALSE |
| Faults Slip Through | FALSE | FALSE | FALSE | TRUE | FALSE |
| First Time Yield | TRUE | TRUE | FALSE | TRUE | TRUE |
| Flow Efficiency | TRUE | TRUE | TRUE | TRUE | TRUE |
| Focus Factor | TRUE | FALSE | TRUE | TRUE | TRUE |
| Foreign Data Providers | FALSE | FALSE | FALSE | FALSE | FALSE |
| Fulfilment of Scope | FALSE | FALSE | FALSE | FALSE | FALSE |
| Halstead Complexity Metric | FALSE | TRUE | FALSE | FALSE | TRUE |
| Hours per Function Point | TRUE | FALSE | TRUE | TRUE | TRUE |

| | | | | | |
|---|---|---|---|---|---|
| Hours per Story Point | TRUE | FALSE | TRUE | TRUE | TRUE |
| Ideal Capacity | FALSE | FALSE | FALSE | TRUE | TRUE |
| Ideal Days | TRUE | FALSE | FALSE | TRUE | TRUE |
| Impediments per Work-Item | TRUE | TRUE | FALSE | TRUE | TRUE |
| Implemented Versus Wasted Requirements | TRUE | FALSE | TRUE | TRUE | TRUE |
| Improvement Potential | FALSE | FALSE | FALSE | FALSE | FALSE |
| Index of Inter-Package Extending | FALSE | FALSE | FALSE | FALSE | TRUE |
| Index of Inter-Package Extending Diversion | FALSE | FALSE | FALSE | FALSE | TRUE |
| Index of Inter-Package Usage | FALSE | FALSE | FALSE | FALSE | TRUE |
| Index of Inter-Package Usage Diversion | FALSE | FALSE | FALSE | FALSE | TRUE |
| Index of Package Changing Impact | FALSE | FALSE | FALSE | FALSE | TRUE |
| Index of Package Goal Focus | FALSE | FALSE | FALSE | FALSE | TRUE |
| Information-Based Cohesion | FALSE | FALSE | FALSE | FALSE | TRUE |
| Information-Flow Based Inheritance Coupling | FALSE | FALSE | FALSE | FALSE | FALSE |
| Internal Efficiency | FALSE | FALSE | FALSE | FALSE | TRUE |
| Internal Rate of Return | TRUE | TRUE | TRUE | TRUE | TRUE |
| Impediments | TRUE | TRUE | TRUE | TRUE | TRUE |
| Kick-Off Days | TRUE | TRUE | TRUE | TRUE | TRUE |
| Lack of Cohesion of Methods | FALSE | FALSE | FALSE | FALSE | TRUE |
| Lead Time | TRUE | TRUE | TRUE | TRUE | TRUE |
| Lines of Code per Method | TRUE | FALSE | FALSE | FALSE | TRUE |
| Lines of Code (per Unit of Time) | TRUE | FALSE | FALSE | FALSE | TRUE |
| Load Factor | TRUE | TRUE | TRUE | TRUE | TRUE |
| Locality of Attribute Accesses | FALSE | FALSE | FALSE | FALSE | FALSE |
| Manual Tests per Story | FALSE | FALSE | FALSE | FALSE | FALSE |
| Maximum Amount of Team Members | TRUE | TRUE | FALSE | TRUE | TRUE |
| Maximum Nested Block Depth | TRUE | FALSE | FALSE | FALSE | TRUE |
| Maximum Work in Progress | TRUE | TRUE | TRUE | TRUE | TRUE |
| Mean Time to Recovery | TRUE | TRUE | TRUE | TRUE | TRUE |
| Message Passing Coupling | FALSE | FALSE | FALSE | FALSE | TRUE |
| Messages per Method | TRUE | FALSE | FALSE | FALSE | FALSE |
| Method Hiding Factor | FALSE | FALSE | FALSE | FALSE | TRUE |
| Method Inheritance Factor | FALSE | FALSE | FALSE | FALSE | TRUE |
| Method-to-Method Export Coupling | FALSE | FALSE | FALSE | FALSE | FALSE |
| Net Present Value | TRUE | TRUE | TRUE | TRUE | TRUE |
| Net Promoter Score | TRUE | TRUE | TRUE | TRUE | TRUE |
| New Classes Per Release | TRUE | FALSE | FALSE | FALSE | FALSE |
| New Features Per Release | TRUE | FALSE | TRUE | TRUE | FALSE |

| New Lines of Code Per Release | TRUE | FALSE | FALSE | FALSE | TRUE |
|---|---|---|---|---|---|
| New Methods Per Release | TRUE | FALSE | FALSE | FALSE | TRUE |
| Non Compliance Index | FALSE | FALSE | TRUE | TRUE | FALSE |
| Normalized Amount of Code Smell Occurrences | FALSE | FALSE | FALSE | FALSE | FALSE |
| Normalized Distance from Main Sequence | FALSE | FALSE | FALSE | FALSE | TRUE |
| Number of Bounce Backs | FALSE | FALSE | TRUE | TRUE | FALSE |
| Number of Classes | TRUE | FALSE | FALSE | FALSE | TRUE |
| Number of Defects Found by Tests | TRUE | FALSE | TRUE | TRUE | TRUE |
| Number of Generated Files | TRUE | FALSE | FALSE | FALSE | TRUE |
| Number of Inherited Methods per Class | TRUE | FALSE | FALSE | FALSE | TRUE |
| Number of Interfaces | TRUE | FALSE | FALSE | FALSE | TRUE |
| Number of Manually Created Files | TRUE | FALSE | FALSE | FALSE | TRUE |
| Number of Methods Added per Class | TRUE | FALSE | FALSE | FALSE | TRUE |
| Number of Modified Generated Files | TRUE | FALSE | FALSE | FALSE | TRUE |
| Number of Overridden Methods per Class | TRUE | FALSE | FALSE | FALSE | TRUE |
| Number of Packages | TRUE | FALSE | FALSE | FALSE | TRUE |
| Number of Requests From Customers | TRUE | TRUE | TRUE | TRUE | FALSE |
| Number of Requirements per Feature | TRUE | FALSE | FALSE | TRUE | FALSE |
| Number of Scrum Teams on Project | TRUE | FALSE | FALSE | TRUE | TRUE |
| Number of Static Methods per Class | TRUE | FALSE | FALSE | FALSE | TRUE |
| Number of Static Variables per Class | TRUE | FALSE | FALSE | FALSE | TRUE |
| Number of Stories per Iteration | TRUE | FALSE | FALSE | TRUE | TRUE |
| Open Defect Count | TRUE | FALSE | TRUE | TRUE | TRUE |
| Open Defect Severity Index | TRUE | FALSE | TRUE | FALSE | TRUE |
| Parameters per Method | TRUE | FALSE | FALSE | FALSE | TRUE |
| Percentage of Adopted Work | TRUE | FALSE | TRUE | TRUE | TRUE |
| Percentage of Completed Stories | TRUE | FALSE | TRUE | TRUE | TRUE |
| Percentage of Dead Code | TRUE | TRUE | FALSE | FALSE | TRUE |
| Percentage of Found Work | TRUE | FALSE | TRUE | TRUE | TRUE |
| Percentage of Modified Generated Files | TRUE | FALSE | FALSE | FALSE | TRUE |
| Percentage of Modified Generated Lines of Code | TRUE | FALSE | FALSE | TRUE | TRUE |
| Percentage of Successful Builds | TRUE | TRUE | FALSE | FALSE | TRUE |
| Person Hours | TRUE | FALSE | FALSE | TRUE | TRUE |
| Person Months | TRUE | FALSE | FALSE | TRUE | TRUE |

| | | | | | |
|---|---|---|---|---|---|
| Personnel Turnover | TRUE | TRUE | FALSE | TRUE | TRUE |
| Polymorphism Factor | FALSE | FALSE | FALSE | FALSE | TRUE |
| Predictive Object Points | FALSE | FALSE | FALSE | FALSE | TRUE |
| Process Efficiency | TRUE | TRUE | TRUE | TRUE | TRUE |
| Processing Time | TRUE | TRUE | TRUE | TRUE | TRUE |
| Queue Time | TRUE | TRUE | TRUE | TRUE | TRUE |
| Regression Test Cycle Time | TRUE | TRUE | TRUE | FALSE | TRUE |
| Release Level Effort Burndown | TRUE | FALSE | TRUE | TRUE | TRUE |
| Remaining Task Effort | TRUE | FALSE | TRUE | TRUE | FALSE |
| Requirement Clarity Index | TRUE | FALSE | TRUE | FALSE | TRUE |
| Requirements Inventory Size | FALSE | TRUE | TRUE | TRUE | TRUE |
| Response for a Class | FALSE | FALSE | FALSE | FALSE | TRUE |
| Return on Investment | TRUE | TRUE | TRUE | TRUE | TRUE |
| Reuse Ratio | FALSE | FALSE | FALSE | FALSE | TRUE |
| Revenue per Customer | TRUE | TRUE | TRUE | TRUE | TRUE |
| Running Tested Features | TRUE | FALSE | FALSE | FALSE | TRUE |
| Schedule Performance Index | FALSE | FALSE | FALSE | FALSE | TRUE |
| Self-Assigned Happiness | TRUE | TRUE | TRUE | TRUE | TRUE |
| Smoke Test Cycle Time | TRUE | TRUE | TRUE | FALSE | TRUE |
| Specialization Index | TRUE | FALSE | TRUE | TRUE | TRUE |
| Reuse Ratio | FALSE | FALSE | FALSE | FALSE | TRUE |
| Sprint Level Effort Burndown | TRUE | FALSE | TRUE | TRUE | TRUE |
| Standard Violations | FALSE | FALSE | FALSE | FALSE | FALSE |
| Statements per Method | TRUE | FALSE | FALSE | FALSE | TRUE |
| Stories per Day per Developer | TRUE | FALSE | TRUE | TRUE | TRUE |
| Story Point Velocity | TRUE | FALSE | TRUE | TRUE | TRUE |
| Success at Scale | TRUE | FALSE | FALSE | TRUE | TRUE |
| System Analysis Cost | FALSE | FALSE | FALSE | FALSE | FALSE |
| Targeted Value Increase | TRUE | FALSE | TRUE | TRUE | TRUE |
| Task Time | TRUE | TRUE | TRUE | FALSE | TRUE |
| Team Effectiveness | FALSE | FALSE | FALSE | FALSE | FALSE |
| Technical Efficiency | FALSE | FALSE | FALSE | FALSE | TRUE |
| Test Execution Cost | FALSE | FALSE | FALSE | FALSE | FALSE |
| Test Failure Rate | FALSE | FALSE | FALSE | FALSE | FALSE |
| Test Growth Ratio | FALSE | FALSE | FALSE | FALSE | TRUE |
| Test Pass Rate | FALSE | FALSE | FALSE | FALSE | FALSE |
| Test Result Analysis Cost | FALSE | FALSE | FALSE | FALSE | FALSE |
| Test Runtime | TRUE | TRUE | FALSE | FALSE | TRUE |
| Test Selection Cost | FALSE | FALSE | FALSE | FALSE | FALSE |
| Tests per Story | TRUE | FALSE | FALSE | TRUE | TRUE |
| Throughput | TRUE | TRUE | TRUE | TRUE | TRUE |
| Thumbs Up Rule | TRUE | TRUE | TRUE | TRUE | FALSE |

| Time to Market in Days | TRUE | TRUE | TRUE | TRUE | TRUE |
|---|---|---|---|---|---|
| Unit Test Coverage | TRUE | FALSE | FALSE | FALSE | TRUE |
| Value Added Time | TRUE | TRUE | TRUE | TRUE | TRUE |
| Value Delivered Over Time | TRUE | FALSE | FALSE | FALSE | FALSE |
| Weighted Method per Class | FALSE | FALSE | FALSE | FALSE | TRUE |
| Win Loss Record | TRUE | FALSE | TRUE | TRUE | TRUE |
| Work Capacity | TRUE | TRUE | FALSE | TRUE | TRUE |
| Work Effectiveness | FALSE | FALSE | FALSE | FALSE | FALSE |
| Work In Progress | TRUE | TRUE | TRUE | TRUE | TRUE |
| Yesterday’s Weather | TRUE | TRUE | FALSE | TRUE | TRUE |

# Draft Paper

# Models for metric strength and software development team performance.

Frank Verbruggen
*Hi Efficiency*
frank@diamongagile.net

Teun Kooijman
*Universiteit Utrecht*
teun.kooijman@gmail.com

Jan Martijn van der Werf
*Universiteit Utrecht*
j.m.e.m.vanderwerf@uu.nl

Sietse Overbeek
*Universiteit Utrecht*
s.j.overbeek@uu.nl

***Abstract*** *– This paper addresses the need for a means of comparing the performance of two different software development teams. A structured literature review was performed to identify the software development metrics that exist today, which was later supplemented with the results of an expert inquiry aimed at identifying software development metrics that the review had missed. This yielded a total of 191 distinct software development metrics, identified in 44 included papers, as well as 6 additional software development metrics, identified by the experts. These results were systematically mapped in a graphing database, and analysed in focus groups with the experts in order to extrapolate tacit knowledge about what makes these metrics strong or weak. The extrapolated knowledge was encapsulated in a new model for metric strength, which was subsequently used to assess the strength of the 197 identified software development metrics. This model states that a metric should (a) be simple to explain and simple to measure, (b) be hard to optimize without increasing business value, (c) correlate strongly with increased business value when optimized, (d) be universally applicable in many different contexts, without confusing edge cases, and (e) be transparent in how it's measured and how it's formulae are calculated. A new model of software development team performance was then created, using a diverse set of software development metrics that were all deemed strong, targeted many different aspects of the software development process, and shared little input data-points.*

## 1. Introduction

According to the yearly State of Agile surveys, an increasing amount of software development companies are embracing Agile in order to increase their performance. It's methodologies boast many positive effects, such as more flexible projects, reductions in project duration, increases in adaptation and satisfaction, fewer deadline-transcending projects, and lower overall project costs. One of the most prominent and oft-quoted advantage of the Agile approach is the claim that it makes your development process *more efficient* (Sutherland, 2014), (Prechelt, 2019), (Leffingwell, 2018).

The methodology provides a number of ways to measure a team's performance, such as *Story Point Velocity* or *Focus Factor* (Padmini, Bandara & Perera, 2015), yet it remains to be seen if these are *strong* measures of software development team performance. If they are not, this means that management will not be able to accurately determine which teams are performing admirably, or even extremely well, and which teams are not. At the same time, individual team members will not know whether their team is excelling or failing.

The most likely problem with these performance metrics is in their manner of size estimation. All size estimation is done in terms of *Story Points*. *Story Points* are estimated from the expert opinions of the team that is going to perform the work. First, an initial reference story is set to an arbitrary number of *Story Points*. From here, the team members estimate the *Story Points* of the other stories in a *relative* fashion, comparing the size of the work of the new story to that of the reference story and other, already estimated stories. Their expert opinions are based on relative estimates of the effort required for

implementing the story, but are likely to be coloured by their experience, expertise, technical aptitude, or even ulterior motives. This means that the size estimation of a story in terms of *Story Points*, may differ quite substantially from team to team. Consequently, a comparison between their performance in terms of their *Story Point Velocity* or *Focus Factor*, is inherently flawed, and easy to game.

Until an accurate assessment of software development team performance can be performed, the inability for an organization to accurately determine the benefits that the adoption of Agile has brought them in terms of performance remains. In this paper, we thus attempt to answer the primary research question outlined below.

**RQ:** *How can we measure the performance of a software development team?*

Before we can determine how we can accurately measure the performance of a software development team, we need to determine what constitutes an accurate and objective metric in the first place. The following sub-question addresses this need.

**S1:** *What constitutes a strong software development metric?*

In order to determine what constitutes a valid and accurate efficiency metric, we need to determine what metrics already exist today. Additionally, we will need to extrapolate what makes these metrics valid or accurate. The following sub-question addresses this.

**S2:** *Which software development metrics already exist today?*

Finally, we need to determine what set of software development metrics is most suitable for measuring team performance, based on their strength, their domain, and the potential correlations caused by a shared set of input data-points. The following sub-question addresses this need.

**S3:** *What set of software development metrics is most suitable for measuring team performance?*

## 2. Methods

In our attempt to answer the various research and sub-questions, we will employ a *Grounded Theory* approach, consisting of a data-collection phase, and a data-structuring phase. Afterwards, two new models are constructed, based on discussions and conversations about the collected data with prevalent experts in the field. Finally, the constructed models are subjected to a preliminary validation, gauging their perceived clarity, relevance and completeness.

### 2.1 Structured Literature Review

In this chapter, we will detail the execution and the results of our systematic literature review, consisting of two phases. The first phase identifies a starting set through an automated search process, whereas the second phase aims to identify missing work, based on Wohlin's snowballing technique (2014), until an iteration no longer results in additional discovered relevant metrics. The aim of this review is to discover as many software development metrics as possible. The process denoted in the remainder of this section was performed on Google Scholar, and duplicated on Scopus.

#### *2.1.1 Inclusion Criteria*

The inclusion criteria used for selecting or discarding literature was kept as broad as possible. The selected papers should be written in English, and should be published in a peer-reviewed journal, or presented at a venue which was facilitated by a peer-reviewed journal, such as a conference or workshop. We will not employ inclusion criteria based on year of publication, specific authors or specific journals. The latter two because we want to evade any such bias, and the former because we deem year of publication to be irrelevant to our purpose. The final decision on whether or not to include a piece of literature is done through manual

examination of the candidate work. Here, the abstract of the candidate is examined, and if needed, the paper is thoroughly studied. In this examination, we will look for the presence of metrics in the work, that are deemed relevant to the field of software development. In the context of this review, a *relevant* metric is defined as a metric that can be used to measure any aspect of a software development process.

*2.1.2 Search Queries*

The search for the starting set of literature was performed on Tuesday the 22nd of January, 2019. The result sets for some of these queries on Google Scholar were so large, that complete analysis was unfeasible for the size and scope of this study. This has caused us to make compromises in terms of validity, for the sake of time. This means that, instead of analysing over 3.500.000 results in order to generate a starting set, only the first ten results were considered for inclusion when performing the automated search on Google Scholar. On Scopus, however, the entire result set was considered, as it was significantly smaller. This consolidation is a severe threat to the validity of our results. The search queries performed on both Google Scholar and Scopus are listed below.

- "Software Development Metrics"
- "Agile Efficiency Metrics"
- "Scrum Productivity Metrics"
- "Agile Productivity"

**2.2 Expert Inquiry**

Additionally, an *expert inquiry* was held among some prevalent experts in the field, where the collected set of metrics will be presented and discussed. The aim of this inquiry was to discover additional software development metrics, that have not yet been discovered in the literature review. In this inquiry, we have not made any distinction as to why they were not discovered in the literature review. This could be, for example, because no prior research has been performed on this metric, no peer-reviewed work has been published on the subject, or because the literature review missed it due to not encompassing the entire body of knowledge available in literature today. In this inquiry, the experts were asked about their view on the current state of efficiency and productivity metrics. Additionally, they were asked to think about possible efficiency metrics that we have not encountered yet, for which they would be very interested in seeing measurement results from the industry. In total, four prevalent experts have participated in the focus groups, some of which are seen as true authorities in the field of agile software development. These experts are *Jeffrey Saltz, Jeff Sutherland, Kyle Aretae* and *Frank Verbruggen.*

**2.3 Systematic Mapping**

The software development metrics that were found, as well as the aspects of the software development process that they target, and their individual input data-points, were then processed using the *Grounded Theory* approach of *axial encoding*. Here, the concepts will be encoded into a final set of aspects and input groups.

Additionally, the metrics that were found were systematically mapped in a graphing database, along with the inputs required to calculate them, the papers that mention them, the authors who wrote them, the keywords those papers use, the journals in which they were published, and the publishers who published them. This systematic mapping was subsequently used for the theory building phase that followed.

**2.4 Model Building**

The software development metrics that were found, as well as their aspects and inputs, and the systematic mapping, were presented to and discussed with the experts. In these discussions, we have attempted to extrapolate the expert's tacit knowledge about determining which metrics can be considered *strong*, and which metrics can be considered *weak*. This tacit knowledge was then distilled in a newly devised model for metric strength. All of the encountered metrics can subsequently be assessed on their strength, using the new model.

Subsequently, we constructed a model for team performance, based on additional discussions and conversations with the experts, and the newly devised model for metric strength. This model was to focus on a set of metrics targeting a broad set of software development process aspects, while

sharing a minimum amount of input data-points so that cause-and-effect can be more easily isolated.

# 3. Results

## 3.1 Structured Literature Review

In total, the automated search produced 103 candidate papers to be included in the review. From these 103 candidates, 44 works were selected for inclusion, based on the defined inclusion and exclusion criteria. The subsequent snowballing process identified an additional 18 works eligible for inclusion, bringing the total amount of included papers up to 62.

These papers were published during the time period 1989 to 2018. Together, these studies were written by 113 individual authors, using 166 distinct keywords. They were published in 37 different venues, facilitated by 12 different publishers.

Within these 62 papers, a total of 191 distinct software development metrics were encountered, targeting 10 different aspects of the software development process, while looking at 14 different categories of input for their data-points. The list of encountered metrics can be found in *appendix A*.

## 3.2 Expert Inquiry

The focus groups of the expert inquiry yielded an additional 6 software development metrics, which were not represented in the body of knowledge discovered by the structured literature review.

### *3.2.1 Priority Focus*

The first additional metric, brought forward by *Jeffrey Saltz*, is the *Priority Focus*, which measures the time that an individual team member has spent adding value to the highest priority story backlog item, as a percentage of the total time spent working. The metric can be calculated for each individual team member, by taking the time that the team member has spent working on the highest priority story backlog item on the previous day, and dividing it by the total time that he or she *could* have spent on it. This metric can be calculated on multiple granularities, e.g. per day or per sprint. At the same time, the metric can easily be calculated for entire teams or companies by aggregating the individual measurements into weighted arithmetic means.

This metric can be used to determine a team's capability to *do the most important things first*. Additionally, the metric can yield interesting insights into how well the team is *swarming* on the highest priority story backlog items. The act of *swarming* has been shown to lead to a reduction of waste in software development processes (Verbruggen, Sutherland, van der Werf, Brinkkemper & Sutherland, 2019). The following sections detail the calculation of this metric for an individual team member, and aggregated into an arithmetic mean for an entire team.

The *Member Priority Focus* for sprint $s$ and member $m$, represented by $pf_{sm}$, is given by

$$pf_{sm} = \frac{\sum_{x=1}^{|E_{sm}|} \begin{cases} p_{e_x} == true & |d_{em}| \\ otherwise & 0 \end{cases}}{wc_{sm}}$$

where $E_{sm}$ is the set of the events that occurred in sprint $s$ for member $m$, $wc_{sm}$ is the *Work Capacity* in sprint $s$ for member $m$, as outlined in *section 3.2.7*, $p_{e_x}$ is a Boolean value denoting whether the $x_{th}$ event $e_x$ was marked as targeting the highest current priority, and $d_{em}$ is the set of timestamps included in the duration of event $e$ <u>and</u> the *Work Schedule* of member $m$, as out lined in *section 3.2.7.*

The *Mean Team Priority Focus* for sprint $s$ and team $t$, represented by $\mu_{pf_{st}}$, is given by

$$\mu_{pf_{st}} = \frac{\sum_{m=1}^{|M_{ts}|} pf_{sm}}{|M_{ts}|}$$

where $M_{ts}$ is the set of the members of team $t$ who have participated in sprint $s$, and $pf_{sm}$ is the *Member Priority Focus* for sprint $s$ and member $m$.

### *3.2.2 Context Concurrency*

The second additional metric, brought forward by *Frank Verbruggen*, is the *Context Concurrency* metric. This metric determines the maximum amount of story backlog items that the team has had to work on concurrently throughout a day, sprint or project. Superfluous context switching can hurt productivity, and keeping the amount of concurrent contexts to switch between to a feasible minimum will help minimize its impact. The metric denotes

the maximum number of stories that were *in progress* at any given time, during a particular period of time.

The *Context Concurrency* of sprint $s$ at timestamp $t$, represented by $cc_{st}$, is given by

$$cc_{st} = |S_t| - |F_t|$$

where $S_t$ is the set of all stories that were started at timestamp $t$, and $F_t$ is the set of all stories that were finished at timestamp $t$.

The *Maximum Context Concurrency* of sprint $s$, represented by $mcc_s$, is given by

$$mcc_s = \bigvee_{t = s_s}^{f_s} cc_{st}$$

where $f_s$ is the timestamp at which sprint $s$ was finished, $s_s$ is the timestamp at which sprint $s$ was started, and $cc_{st}$ is the *Context Concurrency* of sprint $s$ at timestamp $t$.

*3.2.3 Degree of Swarming*

The third additional metric, brought forward by *Jeff Sutherland*, is the *Degree of Swarming*. This metric determines the degree of collaboration and teamwork within the team. It indicates whether team members tend to work on story backlog items individually or in association with other members of the team. It is defined here as the percentage of the team that has performed work on a specific story during a particular day, whether this was two minutes or eight hours.

The *Story Degree of Swarming* on story backlog item $i$ on day $d$, represented by $dos_{id}$, is given by

$$dos_{id} = \frac{|M_{id}|}{|M_d|}$$

where $M_{id}$ is the set of all members who participated in work performed on story $i$ on day $d$, and $M_d$ is the set of all members who were working on day $d$.

The *Mean Day Degree of Swarming* on day $d$, represented by $\mu_{dos_d}$, is given by

$$\mu_{dos_d} = \frac{\sum_{x=1}^{|I_d|} dos_{i_x d}}{|I_d|}$$

where $I_d$ is the set of all story backlog items that were in progress at any time during day $d$, and $dos_{i_x d}$ is the *Story Degree of Swarming* on the $x_{th}$ story backlog item $i_x$ on day $d$.

The *Mean Sprint Degree of Swarming* on sprint $s$, represented by $\mu_{dos_s}$, is given by

$$\mu_{dos_s} = \frac{\sum_{x=1}^{|D_s|} \mu_{dos_{d_x}}}{|D_s|}$$

where $D_s$ is the set of days in sprint $s$, and $\mu_{dos_d}$ is the *Mean Day Degree of Swarming* on the $x_{th}$ day $d_x$.

*3.2.4 Small Correct Change Into Production*

The fourth additional metric, brought forward by *Kyle Aretae*, is the *Small Correct Change Into Production* (SCCIP). This metric looks at the overhead of the act of deploying the product into production. It is defined as the time it takes for a single, extremely simple change to the code base, to be available in the production environment(s). If the target team works with deployment windows, it is assumed that the last deployment window has *just* closed. Kyle has seen this metric range from under 5 minutes in some of the truly high-performance teams, to over a year in some of the worst.

The *Simple Correct Change Into Production* for project $p$ , represented by $sccip_p$, is given by

$$sccip_p = t_d - t_c$$

Where $t_d$ is the timestamp at which the change is available in production, and $t_c$ is the timestamp at which the change was committed.

*3.2.5 Process Efficiency*

The fifth proposed metric, brought forward by *Jeff Sutherland* and *Frank Verbruggen*, is the *Process Efficiency* metric. This metric determines the efficiency of a software development team from the perspective of their work, instead of the individual team members. It is defined as the value-added-time divided by the total time spent working. Here, excellency measures a low throughput time, but could also lead to a low capacity utilization.

The *Story Process Efficiency* for story backlog item $i$, in sprint $s$, represented by $pe_{is}$, is given by

$$pe_{is} = \frac{\sum_{x=1}^{|E_{smi}|} f_{e_x} - s_{e_x}}{ct_i}$$

where $E_{smi}$ is the set of the events that occurred in sprint $s$ for member $m$, targeting story backlog item $i$, $ct_i$ is the *Story Cycle Time* of story backlog item $i$, as outlined in *section 3.2.7,* $f_{e_x}$ is the timestamp at which the $x_{th}$ event $e_x$ has finished, and $s_{e_x}$ is the timestamp at which the $x_{th}$ event $e_x$ has started.

The *Mean Team Process Efficiency* for sprint $s$, represented by $\mu_{pe_s}$, is given by

$$\mu_{pe_s} = \frac{\sum_{x=1}^{|I_s|} pe_{i_x s}}{|I_s|}$$

where $I_s$ is the set of all story backlog items in sprint $s$, and $pe_{i_x s}$ is the *Story Process Efficiency* of the $x_{th}$ story $i_x$ in sprint $s$, as outlined in *section 3.2.7.*

*3.2.6 Innovation Income*

The final proposed metric, brought forward by *Frank Verbruggen* and *Kyle Aretae*, is the *Innovation Income* metric. This metric determines the percentage of an organization's income that's coming from innovations. It posits that if a significant part of the value delivered by an organization comes from recent innovation, the organization has the ability to innovate, and dares to move. Such an organization has the ability to change the way they operate on their markets, and can quickly react to changing circumstances.

The *Innovation Income* $ii$ for organization $o$, denoted by $ii_o$, is given by

$$ii_o = \frac{r_{<2}}{r}$$

Where $r_{<2}$ is the amount of yearly revenue obtained through projects that were released within the last two years, while $r$ is the total amount of yearly revenue for the organization. While the initial cut-off is set at two years, empirical validation might show more optimal values for this threshold.

*3.2.7 Intermediate Metrics*

While the following metrics are not part of the set of metrics suggested by the experts, their values are needed for the calculation of some of the metrics that were. Their definitions are stated below in order to provide an accurate and unambiguous account of how their calculations are done.

The *Work Capacity* in sprint $s$ for member $m$, represented by $wc_{sm}$ is given by

$$wc_{sm} = \sum_{x=1}^{|D_{sm}|} f_{md_x} - s_{md_x}$$

where $D_{sm}$ is the set of days during sprint $s$ on which member $m$ worked on the project, $f_{md_x}$ is the time at which member $m$ stopped working on the $x_{th}$ day $d_x$, and $s_{md_x}$ is the time at which member $m$ started working on the $x_{th}$ day $d_x$.

The *Work Schedule* of member $m$ in sprint $s$, represented by $U_{ms}$, is the union of the intervals of the times that member $m$ worked during sprint $s$, and is given by

$$U_{ms} = \bigcup_{x=1}^{|D_{ms}|} [s_{md_x}, f_{md_x}]$$

where $D_{ms}$ is the set of days that member $m$ worked during sprint $s$, $s_{md_x}$ is the time at which member $m$ started working on the $x_{th}$ day $d_x$, and $f_{md_x}$ is the time at which member $m$ stopped working on the $x_{th}$ day $d_x$.

The *Event Duration* for event $e$ of member $m$, represented by $d_{em}$ is given by

$$d_{em} = \{x \mid x \in U_{ms}, x \in [s_e, f_e]\}$$

where $U_{ms}$ is the *Work Schedule* of member $m$ in sprint $s$, $s_e$ is the time at which event $e$ has started, and $f_e$ is the time at which event $e$ has finished.

The *Story Cycle Time* of story backlog item $i$, represented by $ct_i$, is given by

$$ct_i = f_i - s_i$$

where $f_i$ is the timestamp at which story backlog item $i$ is finished, and $s_i$ is the timestamp at which story backlog item $i$ is started.

Similarly, the *Story Cycle Interval* of story backlog item $i$, represented by $ci_i$, is given by

$$ci_i = \{[s_i, f_i]\}$$

where $f_i$ is the timestamp at which story backlog item $i$ is finished, and $s_i$ is the timestamp at which story backlog item $i$ is started.

The *Mean Team Interruption Count* for sprint $s$, represented by $\mu_{ic_s}$, is given by

$$\mu_{ic_s} = \frac{|I_s|}{|M_s|}$$

where $I_s$ is the set of the interruptions that occurred in sprint $s$, and $M_s$ is the set of the team members who participated in sprint $s$.

The *Mean Team Interruption Duration* for sprint $s$ and team $t$, represented by $\mu_{id_{st}}$, is given by

$$\mu_{id_{st}} = \frac{\sum_{x=1}^{|I_s|} f_{i_x} - s_{i_x}}{c_{i_s}}$$

where $I_s$ is the set of the interruptions that occurred in sprint $s$, $f_{i_x}$ is the time at which the $x_{th}$ interruption $i_x$ was finished, and $s_{i_x}$ is the time at which the $x_{th}$ interruption $i_x$ started.

### 3.3 Model for Metric Strength

In this section, we introduce a new model for metric strength, which describes five qualities that a metric should have in order to be considered a *strong* metric for software development This model for metric strength was developed through in-depth discussion of metric strength with the experts, in which tacit knowledge about *what makes a metric good or bad*, was extrapolated and distilled into explicit knowledge.

These qualities state that a *strong* metric should (**a**) be simple to explain and simple to measure, (**b**) be difficult to optimize without increasing business value (**c**) correlate strongly with increased business value when optimized, (**d**) be useable in multiple contexts, without confusing edge-cases, and (**e**) have an unambiguous and transparent definition of its data points, as well as how those data points are used in its calculations. In the remainder of this study, we will refer to these qualities as simple, hard-to-game, outcome-oriented, universal, and transparent respectively. Together, these criteria spell the acronym SHOUT.

#### *3.3.1 Simple*

The first quality criteria is *simplicity*. This addresses the need for a metric to be simple to explain, measure and interpret. It also takes into account how much effort, in terms of time and energy, is required to take the required measurements. Finally, it takes into account the perceived impact on the productivity of the team under investigation. If taking the required measurements takes only a second, but has to be done many times a day, the overall effort required is low, but the impact on overall team productivity might be too high, because of the numerous interruptions that it would cause.

#### *3.3.2 Hard to Game*

Then, the metric is judged on whether or not its value is hard to game. In the context of this study, *hard to game* is defined as *being difficult to optimize without increasing business value*. This means that we do not truly care whether or not a metric is easy to game or not, as long as the act of gaming still results in the intended increase in business value. An excellent of example of a metric that is hard to game in this sense, is *Work in Progress*. The emergence of the *hard-to-game* quality is not all that surprising, as E.M. Goldratt's '*tell me how you measure me, and I'll tell you how I'll behave'* comes to mind.

#### *3.3.3 Outcome Oriented*

*Strong* metrics should also *show a strong correlation with increased business value when optimized.* This means that the metric should give a clear indication of where that optimum might be, and can reasonably be assumed to increase business value when a process gets closer to that optimum.

#### *3.3.4 Universal*

For a metric to be *universal*, it must be applicable to many different contexts, and not just software development or industrial manufacturing. Similarly, it should not have any confusing edge-cases for specific circumstances, resulting in invalid measurements or values.

#### *3.3.5 Transparent*

Finally, metrics should be transparent, meaning that they should have an explicit and unambiguous definition of their data points. Additionally, all of the metrics should be transparent in the sense that they

should unambiguously define how those data points are used to calculate the final metric value(s).

### 3.4 Model for Team Performance

In this section, we will introduce a new model for assessing team performance, based on the concepts discovered in the structured literature review, the discussions with experts, and the systematic mapping of their results. This model assesses the performance of a team along four different axes, being *process*, *people*, *technical* and *product*. These perspectives were derived from a discussion with the experts of the final encoding pass over the aspects of software development that metrics can target

Each of these perspectives has a single key metric that adheres to the SHOUT model of metric strength, and is thus completely outcome-oriented. Consequently, the resulting measurements tell an individual team whether or not they are performing well on an individual perspective, but do not tell us anything about how to improve it. Additional metrics are required to provide a team with the necessary pulls and levers to actively navigate towards becoming a truly high-performance team. This is, however, part of our future research as indicated in *section 5.3*.

#### *3.4.1 Process*

According to Lean Manufacturing, the best manufacturing processes are optimized to reduce waste. In our team performance model, we state that a team's process is performant when it maximizes added-value, while minimizing wasted resources. The *strong* metric of *Process Efficiency*, introduced in section 4.2.5, measures the percentage of total time spent adding value, and is used as the key metric for the *process* perspective on team performance.

#### *3.4.2 People*

In our model of team performance, we hold true the axiom that *the members of a team need to feel good about themselves and their company in order to become a high performance team.* The *Employee Happiness* metric, introduced in section 3.4.14.17, measures this sense of purpose, belonging and satisfaction that the experts believe is a necessary ingredient to high performance, and is used as the key metric for the *people* perspective team performance.

#### *3.4.3 Technical*

High technical performance allows a team to translate concepts into profitable products and services in minimal time. This maximization of speed, alongside the minimization of required effort, is perfectly encapsulated in the *Small Correct Change Into Production* metric introduced in section 4.2.4, and is thus used as the key metric for the *technical* perspective on team performance.

#### *3.4.4 Product*

Doing the right thing is equally important as (if not more important than) doing the thing right. High performance in the *product* perspective means maximizing the value in the eyes of the customers. The *Net Promoter Score* metric, introduced in section 3.4.14.11, measures how many more people love the product or service you've created, than the amount of people that hate it, and is used as the key metric for the *product* perspective on team performance.

## 4. Discussion

### 4.1 Structured Literature Review

The structured literature review yielded a large set of metrics, hinting at a large body of knowledge for software development metrics. The collected work, spanning more than 40 individual papers on the subject and over 1000 potential candidates, shows a healthy distribution over venues and publishers, giving us no reason to suspect any form of venue or publisher bias.

The study found 197 individual metrics, which is more than 4.5 times as many as the largest literature review on the subject that we found, giving us adequate reason to believe that our current work has added significant value to the field of measuring software development processes, by the results of the structured literature review alone. This seems to have been a necessary endeavour, seeing as the resulting set of keywords hint at an industry that lacks a clearly defined lexicon of standardized terms, with lots of synonyms and very little overlap

between papers. Similarly, when looking at the set of authors working on the included work, we see that they rarely publish more than one paper on the subject, with the most prevalent expert being Jeff Sutherland at three included papers. This also hints at a field that lacks well-known and prominent experts on the subject. Given the fact, however, that our inclusion criteria stated that a paper should mention a *new*, *previously unmentioned* software development metric, we cannot be all too sure about the latter two conclusions.

Surprisingly, no *golden age* of software development metric research can be identified, as the field has seen continuous and consistent attention since its inception. The distribution of metric mentions does, however, show a focus of research on complexity, quality and efficiency metrics, with 146 metrics targeting just these three aspects of the software development process. Similarly, a significant amount of metrics seem to have input data-points coming from work-items and their lifecycle, as well the source code, with 52 out of 118 inputs originating from just these three input categories.

In terms of metric strength, according to the newly introduced SHOUT model of metric strength, it is surprising to see that five out of ten aspects failed to yield *any* strong metrics. Even more surprising is the fact that complexity and quality are among them, while 98 such metrics were identified. While we expected this to be because they were not classified as *universal* (and thus only adhering to a SHOT model of metric strength), we found that most often, they were not classified as *outcome-oriented* instead. This is not surprising, as code quality and complexity metrics can be excellent tools to maintain a high level of maintainability and clarity, but optimizing them does not necessarily correlate with increased business value. Similarly, such metrics can fairly easily be gamed, with various adverse effects. The *lines of code per method* metric, for example, can be kept artificially low by limiting it to one per method, but this might severely hurt readability and maintainability. The efficiency aspect, however, has yielded 12 *strong* metrics, most of which come from LEAN software development or manufacturing. Most of these metrics target various aspects of the life-cycle of a work-item (e.g. lead-time, queue-time, cycle-time, interrupted-time, and value-added-time). Similarly, the *Work-in-Progress* metrics that were encountered in the process aspect of software development, also have their roots in LEAN manufacturing or software development.

According to the distribution of qualities over metrics, the *hard-to-game* quality appears to be the hardest quality to inhibit for a metric, with just 26.3% of the encountered metrics adhering to it. Similarly, only 31.4% of the encountered metrics have shown to be *outcome-oriented*, making it the second hardest quality to adhere to. Finally, just 23 out of 197 metrics can be considered *strong*, being only 11.6% of the entire set of encountered software development metrics. This hints at the necessity of an accurate model for metric strength, as well as the need to keep quality in mind when devising new software development metrics. While the review has yielded a large set of metrics, it has yielded no model for determining metric strength or quality. The goal-question-metric model came closest, but focusses on what makes a metric good for a particular organization's *context* instead. A model for metric strength is thus a welcome addition to the field of software development metrics.

### 4.2 Expert Inquiry

The expert inquiry was done with a small group of experts, yet the group consisted of very prominent and prevalent experts in the field, with lots of experience and expertise between them. We found that it was surprisingly easy for a small group of experts to unanimously and quickly determine whether or not a metric could be considered *strong* or not, even without the SHOUT model for metric strength in place.

The inquiry yielded six additional metrics that were not identified through the structured literature review and its snowballing process. It is interesting to note that all six metrics could be considered *simple*, *hard-to-game*, *outcome-oriented* and *universal*. Now that their definitions, as well as their data-points have been clearly and unambiguously defined in this work, they can also be considered to be *transparent*. This means that all of the metrics retrieved from the expert inquiry can now be considered *strong* metrics, and can now be used by

software development teams to determine some aspects of their performance.

*Context Concurrency, Priority Focus* and *Degree of Swarming* show clear similarities with Kanban, where the amount of work-in-progress is limited in order to prevent an abundance of context switching and to stimulate a focus on the highest current priority. Additionally, *Degree of* Swarming shows similarities with the rise of *pair programming,* and the move away from the stereotypical independent and anti-social software developer. *Small Correct Change Into Production* and *Innovation Income* can both be considered as very simple, fast indicators of general technical and organizational performance, while in-depth analysis would require other, more complex and time-consuming metrics. Finally, it is interesting to note that *Process Efficiency* is a strong metric, while all of its inputs can *also* be considered *strong*, hinting at a very promising application that will need to be validated in future empirical research.

### 4.3 Systematic Mapping

The systematic mapping has proven to be very helpful in analysing and interpreting the results of the structured literature review and the expert inquiry. While the axial-encoding would most likely have yielded different results if performed by other researchers, we feel like it has fulfilled its purpose adequately. At the same time, however, we feel very strongly that potentially many more patterns and insights can be extracted from the systematic mapping, or with a potentially different axial-encodings. For this reason, we have decided to publish the data set in its entirety on

https://www.silvester-consultancy.com/portfolio/thesis/download/systematic-mapping.

### 4.4 Model for Metric Strength

The SHOUT model for metric strength was received fairly well by the participants of the small validation survey. In their responses, the participants signalled the definitions of the qualities to be very clear, with high median values, just as the relevance of these qualities. In the end, the model was thought to reasonably encompass every quality that a metric should have in order to be considered *strong*, with a median score of 4 and a *Net Promoter Score* of 56%.

The model does, however, need a larger-scale validation in the industry, with a larger set of verified participants, whereas the current validation was just a small probe into the general reception of the model.

### 4.5 Model for Team Performance

The model for team performance shows very little correlation based on shared input data-points, with only the timestamp at which a work-item has finished being used for both *Small Simple Change Into Production* and *Process Efficiency*. As stated in chapter 8, however, the input data-point is used for widely different things, and represents different concepts in both metrics. The resulting model has, however, not been validated in this study, and so reception and performance of the model is difficult to gauge.

## 5. Conclusion

### 5.1 Research Questions

#### *5.1.1 Which software development metrics already exist today?*

In this study, we performed a structured literature review as to determine what software development metrics exist today, resulting in 191 software development metrics. In order to ensure that no metrics were overlooked, we performed an expert inquiry in which we asked prevalent experts in the field of software development whether they thought the resulting list was complete, resulting in an additional 6 metrics.

#### *5.1.2 What constitutes a strong software development metric?*

The results of this endeavour were structured in a systematic mapping, and discussed with the experts in order to determine what makes them strong or weak. From this discussion, a new model for metric strength was developed, identifying five qualities that a metric should possess in order to be considered *strong*. These qualities state that a *strong* metric should (a) be simple to explain and simple to

measure, (b) be difficult to optimize without increasing business value (c) correlate strongly with increased business value when optimized, (d) be useable in multiple contexts, without confusing edge-cases, and (e) have an unambiguous and transparent definition of its data points, as well as how those data points are used in its calculations. We have dubbed these qualities *simple, hard-to-game***,** *outcome-oriented, universal,* and *transparent* respectively, and together, these qualities spell the acronym SHOUT.

#### *5.1.3 What set of software development metrics is most suitable for measuring team performance?*

Finally, this model was used to identify strong metrics in the result set of the structured literature review and the expert inquiry. From this set of strong metrics, we have created a new model for measuring software development team performance. This model is based on the *Process Efficiency*, *Employee Happiness, Net Promoter Score* and *Small Simple Change Into Production* metrics, targeting the *process*, *people, product* and *technical* perspectives of the software development process respectively. This model has not been validated in this study, but initial analysis have shown that little correlation between these metrics is to be expected, based on their shared input data-points.

#### *5.1.4 How can we measure the performance of a software development team?*

Finally, by answering our three sub-questions, we are able to answer our primary research question of how we can measure the performance of a software development team. The final answer to this question is thus to use *strong* software development metrics, utilizing *independent input-data-points* in order to isolate cause-and-effect relationships, while targeting *multiple aspects* of the software development process. In this thesis, we have presented a model for assessing the strength of a software development metric, as well as a model for measuring team performance, based on *strong* metrics, sharing little input data-points and targeting four different aspects of the process. These models can help organizations assess the performance of their software development teams. Finally, we have introduced automated tooling in order to help organizations measure these four key metrics.

### 5.2 Limitations

#### *5.2.1 Limited Google Scholar starting set*

There are several limitations in our execution of this research. First and foremost, we have had to make some concessions as to how thorough our manual search for candidate work could be. Here, we have limited the initial collection of candidate work from Google Scholar to just the first 10 results, instead of incorporating the whole result set. This may have, in the end, led to less valid results, due to not having exhausted the entire existing body of knowledge. However, as we have found more than 4.5 times as many metrics as the largest literature review we have found on the subject, we feel very confident that the extent to which these factors threaten the validity of our results is fairly minimal.

#### *5.2.2 Limiting inclusion criteria*

Similarly, our inclusion criteria of needing to mention a *new* software development metric, as opposed to just *any* software development metric, has a significant influence on the validity of our results. The possibility exists that we have missed a substantial portion of the existing body of knowledge, due to potential separate clusters that our practice may have missed due to this inclusion criteria. A reproduction study would be wise to broaden this inclusion criteria to mentioning *any* software development metric, but we fear that this will substantially increase the effort required to properly perform the study.

#### *5.2.3 Initial focus on efficiency*

Additionally, we set out to perform this literature review with an initial focus on *efficiency* metrics. For this reason, the search queries that were executed on the Google Scholar and Scopus search engines, were deliberately biased to target software development metrics targeting *efficiency*. Only after having performed the searches, and having seen the amount and quality of the results, did we decide to register *all* software development metrics. This bias in search queries might have caused us to mis significant clusters of metrics in the body of knowledge on software development metrics.

#### *5.2.4 Limited model validation*

Finally, the validation of the SHOUT model for metric strength cannot be considered thorough and complete. The participants of the validation survey were reached through social-media, and therefore not verified to be software development professionals. Additionally, the model for team performance has not seen any validation in this study at all, which calls for future work investigating the effectiveness of the model in, for example, separate case-studies.

**5.3 Future Work**

*5.3.1 Thorough model validation*

With this study, we have set a first step towards enabling organizations to measure the performance of a software development team. We have not, however, proven that this model for team performance is accurate or valid. In future work, we plan to validate the model in an industry setting using case-studies in which the model's accuracy is validated. Only after this has happened, can mainstream adoption potentially occur.

Similarly, the validation of the SHOUT model for metric strength has yet to see a thorough validation of its capacities. While we have performed a small survey on these qualities, this was solely meant as an initial probing into their perceived clarity, relevance and completeness, and additional, more thorough validation is required in order to draw any significant conclusions.

*5.3.2 Additional analysis of the systematic mapping*

Additionally, we have acquired and systematically mapped a substantial part of the available body of knowledge on software development metrics. While this mapping served its purpose in our research more than adequately, we feel very strongly that there are additional patterns and insights to be discovered within it. We have therefore opted to open-source the results, in order to enable other researchers to draw their own conclusions from them.

*5.3.3 Investigate the effectiveness quality*

The preliminary validation of the model for metric strength brought forward an additional quality that many seem to associate with *strong* software development metrics, namely *effectiveness*. Future work could benefit from determining what exactly respondents mean with *effectiveness*, whether it is the same as *outcome-oriented*, or whether it might be a potential sixth quality for *strong* software development metrics.

*5.3.4 Multidisciplinary approach*

Additionally, it might prove beneficial to approach future work from a multi-disciplinary perspective, as the fields of psychology, sociology and even anthropology might have valuable insights into what qualities contribute to the strength of a metric. In this study, a focus on software development was used, but a broader view might yield a more robust and universal model for metric strength or team performance.

*5.3.5 Broader inclusion criteria*

Finally, the inclusion criteria of having to mention *new* software development metrics, as opposed to just *any* software development metric, is a significant blow to the validity of our results. While we have found more than 4.5 times as many software development metrics than any other literature review we have found on the subject, we feel that we will still have potentially missed numerous other metrics due to this inclusion criteria. A thorough reproduction of this literature review will have to broaden this inclusion criteria to state that a work will be included if it mentions *any* software development metric, but this will increase the required effort, time and resources substantially.

## 6. References


Leffingwell, D. (2018). SAFe 4.5 Reference Guide: *Scaled Agile Framework for Lean Enterprises*. Addison-Wesley Professional.

Padmini, K. J., Bandara, H. D., & Perera, I. (2015, April). *Use of software metrics in agile software development process*. In Moratuwa Engineering Research Conference (MERCon), 2015(pp. 312-317). IEEE.

Prechelt, L. (2019). *The Mythical 10x Programmer*. In Rethinking Productivity in Software Engineering (pp. 3-11). Apress, Berkeley, CA.

Sutherland, J. (2014). *Scrum: the art of doing twice the work in half the time*. Currency.

Verbruggen, F., Sutherland, J., van der Werf, J. M., Brinkkemper, S., & Sutherland, A. (2019, January). *Process Efficiency-Adapting Flow to the Agile Improvement Effor*t. In Proceedings of the 52nd Hawaii International Conference on System Sciences.

Wohlin, C. (2014). *Guidelines for snowballing in systematic literature studies and a replication in software engineering. In Proceedings of the 18th international conference on evaluation and assessment in software engineering* (p. 38).