Measurement In DevOps Standard: Proposed Improvements

Alexandra LAPOINTE-BOISVERT[1], Sylvie TRUDEL[1] and Jean-Marc DESHARNAIS[2]

[1]*Dept. of Computer Science, UQAM, Montreal, Canada*
[2]*ÉTS, Montreal, Canada*

Abstract

Context. DevOps is about improving the software delivery performance. Lord Kelvin stated: "*If you can not measure it, you can not improve it.*" For this reason, well-defined measurement is particularly important when using DevOps. **Method.** This paper critically examines the measurement process outlined in the ISO/IEC/IEEE-32675 DevOps standard and proposes several improvements based on measurement-related standards (ISO-15939, ISO-25021, VIM, etc.). **Findings.** The measures (base or derived) stated in the DevOps standard are neither characterized nor defined, as described in several standards. Some measures given as examples are ambiguous. Also, the DevOps standard claims to follow ISO/IEC/IEEE-12207, but some expected measures are missing (e.g. effort and size). This article proposes a way to define the different measures and explains why having effort and size measures is important in the context of DevOps. **Conclusion.** The actual measurement process, as described in ISO/IEC/IEEE-32675, could be improved by better defining the base measures, including effort and size, using a measurement method, defining the derived measures using a measurement function, and avoiding ambiguous measures.

Keywords

ISO-32675, base measures, derived measures, DevOps, software measurement, software measurement process

1. Introduction to the ISO-32675 and ISO-15939 standards

1.1. DevOps standard summary

ISO/IEC/IEEE-32675:2022, an international standard for "*DevOps: Building Reliable and Secure Systems Including Application Build, Package, and Deployment*" [1], is about how DevOps can address challenges of the "*accelerated development methodologies*" to "*achieve end-user goals for increased productivity and quality.*" The standard covers the DevOps concepts [1, p. 17-23] and the relation of software life cycle processes to DevOps [1, p. 23-87]. The DevOps concepts section describes four processes: agreement processes, organizational project-enabling processes, technical management processes, and technical processes. Our main interest is in the measurement process [1, sec. 6.3.7] as part of the technical management processes.

IWSM-Mensura 2024, September 30 - October 4, 2024, Montreal, Canada
lapointe-boisvert.alexandra@courrier.uqam.ca (A. LAPOINTE-BOISVERT); trudel.s@uqam.ca (S. TRUDEL); jean-marc.desharnais@etsmtl.net (J. DESHARNAIS)
https://professeurs.uqam.ca/professeur/trudel.s/ (S. TRUDEL)
0009-0009-9498-8489 (A. LAPOINTE-BOISVERT); 0000-0002-4983-1679 (S. TRUDEL); 0000-0001-8712-8696 (J. DESHARNAIS)

1.2. Measurement process purpose

The DevOps standard cites ISO/IEC/IEEE 12207:2017 [2], which states that the purpose of the measurement process is "*to collect, analyze, and report objective data and information to support effective management and demonstrate the quality of the products, services, and processes.*" Within a DevOps context, this means coping with a high amount of information ingested at a fast collection rate, such as telemetry data. Therefore, automated and quantifiable measurements are required to support management decisions effectively. The standard highlights 15 "*typical measures for availability and performance that can be useful in DevOps applications*" [1, 6.3.7.1]. The measures are shown in Table 1.

1.2.1. Measures mentioned in the DevOps standard

The following measures are included within ISO-32675 as examples of DevOps implementation.

Table 1
List of the measures given as examples in the DevOps standard

#	Name of the measure
1.	Deployment frequency
2.	Change lead time and volume
3.	Change failure rate
4.	Defect rate
5.	Mean time to recovery (MTTR)
6.	Mean time to detection (MTTD)
7.	Issue volume (number of issues) and resolution time
8.	Time to approval
9.	Time to patch vulnerabilities
10.	Logging availability
11.	Retention control compliance
12.	Software assurance requirements findings count
13.	Application traffic
14.	Attack vector details (IP, stack trace, time, rate of attack)
15.	Resource utilization

These measures provide insights into the DevOps process. However, our analysis reveals a lack of explicit definitions and the use of standardized measurement methods. The absence of such details hinders the comparability and reproducibility of results across different organizations (see section 1.4). Furthermore, several implicit measures can be found under a single term. This ambiguity can make it difficult to determine the exact object being measured.

1.3. Presentation of the measurement information model

The maturity of the software process relies on key relationships in the measurement information model as part of the ISO-15939 standards [3]. The ISO-15939 standard describes three perspectives from base measures, derived measures, and indicators closely related through three

measurement activities: data collection, data preparation, and data analysis to interpret the data of an information product. Figure 1 shows our view of the three perspectives.

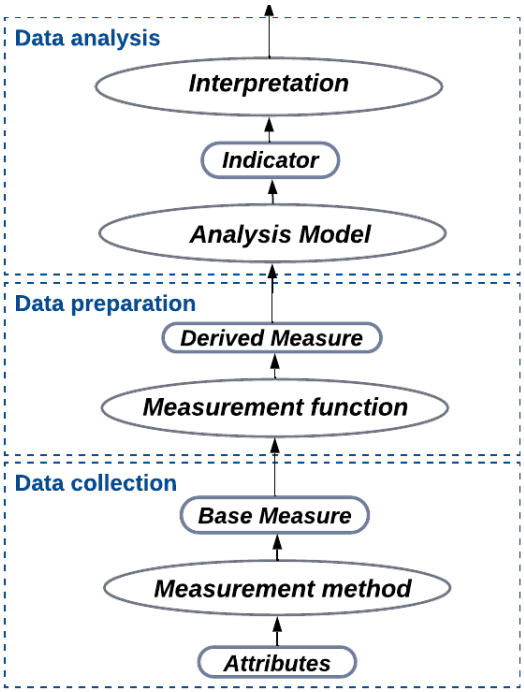

Figure 1: The three perspectives of the measurement information model

The information model indicates the construction steps to obtain a data analysis model. This approach can be found in the following standards: ISO-15939 [3], ISO-25021 to ISO-25025 [4, 5, 6, 7, 8], and the VIM [9] standards. The first three standards were compared by Desharnais et al. [10].

It is not uncommon that, in the software field, we tend to start by suggesting a certain number of derived measures without taking the time to define the base measures. For example, the software quality measure ISO-9126 [11] was created by ignoring the definition of base measures, as described in Desharnais et al. [12].

To ensure clarity and establish a common ground for discussion, we provide some concept definitions. A more extensive list of definitions regarding the measurement elements of these three perspectives can be found in Annex A of the ISO-15939 standard.

Entity: "*It is an object that is to be characterized by measuring its attributes*" [3]. Examples include source code, design documents, and software requirement specifications (SRS). Objects can be further classified into products, processes, and resources (e.g. people and tools). An entity includes attributes that must be measured in order to be relevant in a measuring process.

Attribute: The attributes of an entity may or may not be measurable. It can be measured by human or automated means. First, we choose the attributes most relevant to the information needs of the measurement user.

Base measure: A base measure "*captures information about a single attribute*" [3]. When

collecting data on assigning values to core measures.

Measurement method: It "*is a logical sequence of operations, described generically, used in quantifying an attribute with respect to a specified scale*" [3].

Derived measure: The combination of two or more base measures is a derived measure. We must distinguish algorithms (for example, SQuaRe) from a derived measure which captures information on several attributes (or entities).

Measurement function: It is the formula that joins the base measures. This formula must be clearly defined. This formula must consider the measurement scales. For example, ordinal values cannot be added.

Analysis model: This model combines a number of derived measures linked to decision criteria. For example, improve productivity by X% or even make an estimate with a confidence factor of plus or minus Y%.

Indicators: They are the basis of analysis and decision-making. This is what is presented to decision-makers. Since "*measurement is always based on imperfect information, therefore quantifying the uncertainty, precision or importance of indicators is an essential part of presenting the true value of the indicator*" [3].

1.4. Issues related to the proposed measures

According to ISO-15939, the maturity of the software process relies on key relationships in the measurement information model. To interpret the data of an information product, one perspective is to have base measures, derived measures, and indicators closely related (data preparation). The second perspective is about data analysis. The tendency is to look at the data analysis perspective and forget the data preparation perspective. Our aim in this article is data preparation.

According to Figure 1, the measures outlined by DevOps standard show that:

A. The measures are not characterized (e.g. base measure, derived measure, or indicator).
B. The base measures are not defined.
C. The measurement functions of derived measures are not defined.
D. Some measures cannot be qualified due to insufficient precision.
E. Some measures are ambiguous.

A. The list does not provide any distinction between the different measures (e.g. base measures, derived measures, or indicators). Without characterization, the measures do not allow consistent and reliable data collection.

B. The DevOps standard does not define the base measures. However, it is impossible to use them coherently since organizations could apply different definitions of these base measures, resulting in different or incomparable results from one organization to another or from one project to another. This article proposes several base measures for DevOps (see Table 3) and suggests improving the measurement process. Finally, we cannot find any measure of size and effort that could be useful for measuring the velocity as defined in Section 4.

C. To create a derived measure, a well-defined measurement function is essential. This function mathematically transforms one or more base measures into the desired derived mea-

sure. For instance, the change failure rate is a derived measure calculated using the following measurement function:

$$X = A/B \tag{1}$$

where:

X = represents the change failure rate
A = represents the number of failures
B = represents the number of changes

Of the 15 provided measures, the majority are derived measures (for example, MTTR, MTTD) as defined by the ISO-15039 standard, some contain base measures (for example, attack vector details), and some are too imprecise for qualification (example, resource utilization). Some contain two different measures (for example, change lead time and volume).

D. Without a precise identification of the measures, it is difficult to reproduce different measurements and interpret them as base measures or in combination with derived measures. The lack of consistency when using base measures in data collection can affect data preparation and analysis. Several standards have been proposed for this [3, 9, 4].

E. Ambiguous measures can lead to misrepresenting, hindering benchmarking, and offering unclear improvement targets. The change in lead time and volume is an example of an ambiguous measure. First, it combines time and volume, which are two completely different concepts. Secondly, there is no clear definition of when the start time begins and where the end time stops to adequately measure the lead time. Lead time is usually defined by the time it takes to go from a customer making a request to the request being satisfied. However, Forsgren et al. who promoted the lead time as one of the four DevOps key measures for measuring performance, separates the lead time into two parts: the time it takes to design and validate a product or feature and the time it takes to deliver the feature to customers. The latter is defined as the product delivery lead time and is the one measured as "the lead time". This can be misleading for an unadvised reader.

Another example is the "defect rate". Fenton and Bieman [14, p .450] mention that "*The defect density measure is sometimes incorrectly called a defect rate*" and it could lead to defining defect rate as the number of defects found during a time period or per software component (e.g. module, API, micro-service, etc.) or as a number of defects found per software size (e.g. per one thousand source lines of code - KSLOC or per one hundred function points). The latter being the derived measure of "defect density". Separating the measures and defining the base measures and the measurement function would allow a better comprehension of the DevOps performance and benchmarking.

It is not the first time that a standard has proposed several measures that are mainly derived measures (constructed by a measurement function) but with no definition of their related base measures.

The ISO-9126 series was analyzed in 2011 by Desharnais et al. [12] and was since replaced by the ISO-25000 series [15]. The SQuaRE series provides a framework for quality measures at a high level [16]. Also, at a lower level, a standard [4] proposes a process to define the different quality elements of measures (base measures) and several definitions.

Those measures did not cover the size of the product and indirectly covered the effort using the word "time", which often means "duration". Size (e.g. functional size) is helpful in the requirements part of the life cycle. Size is also beneficial for establishing the velocity of the development and implementation. According to this definition, DevOps covers all aspects of the life cycle as defined in ISO-12207. The ISO 32765 standard's sub-section 5.1, Values of DevOps [1, 5.1], mentioned that "*DevOps seeks to achieve a balance between velocity and system reliability and stability*".

2. Presentation of DevOps

ISO-32675 defines DevOps as a "*set of principles and practices which enable better communication and collaboration between relevant stakeholders for the purpose of specifying, developing, and operating software and systems products and services, and continuous improvements in all aspects of the life cycle*".

2.1. Concepts

DevOps has different concepts. The value of DevOps is the first one: According to ISO-32765, "*DevOps is appropriate for building, packaging, and deploying reliable and secure systems*"[1]. It is more than just combining development and operation. In the standard, there is a recognition for balancing the "V" requirements (i.e. volume, velocity, variety, veracity, value, etc.). The customers and producers must be included.

2.2. Principles

There are also some principles:

- **Business first** means that "*DevOps focuses on business and organizational goals ahead of procedural and technical considerations*";
- **Customer focus**: "*DevOps takes a customer-centric view, prioritizing and designing work to deliver value to the customer, as well as identifying and managing risk*";
- **Left-shift** means that continuous delivery, testing, security, and quality assurance (QA) practices are integrated earlier in the workflow.
- **Continuous everything** means "*using the same practices in development as in operations and sustainment*";
- **System thinking**: DevOps discourages promoting specialists (networking, database specialist, etc.). This means that "*in DevOps, taking a comprehensive view encourages technology professionals to fully understand the system from end to end*".

2.3. DevOps and the organizational culture

According to the DevOps standard, the life cycle is part of the organizational culture [1]. Creating a continuous delivery pipeline is essential within the life cycle process. Each part of the delivery cycle needs to communicate effectively and automatically as much as possible. This is possible

with open standards across telemetry signals. According to Blanco, a respected figure in the field, "*telemetry in software systems allows us to efficiently operate complex architectures to ensure high performance and reliability*" [17]. In the book "Practical Open Telemetry" [17], Blanco gives several examples of how open telemetry can improve the communication between software systems in DevOps, mainly in Chapter 7, which discusses metrics.

Blanco strongly advocates open standards, much like the SI (*Système International*) units. He believes that "*the adoption of SI units [9] as a standard in all scientific fields has boosted research collaboration, leading to faster and safer advancements in technology*". He added: "*unfortunately, standards in software telemetry are in a much more delicate situation, and we don't have to contemplate interactions at a global scale to see these flaws*". Adopting the measurement method and the measurement function approach promoted by different software standards [3, 4] is a perfect base to implement standard units recognized by the software engineers within the life cycle process. Those standards will then become a part of the culture of the organization.

2.4. Relation of software life cycle processes to DevOps

Section 6 of ISO-32675 [1, sect. 6] discusses the relationship between software life cycle processes and DevOps. The section covers the agreement processes, the organizational project-enabling processes, the technical management processes, and the technical processes. The measurement processes are part of the technical management process. The definition of the technical management processes is: "*to establish and evolve plans, to execute the plans, to assess actual achievement and progress against the plans, and to control execution through to fulfillment*". Following and controlling the plan to improve it is an important part of the technical management processes. This is why measurement is also important. As Lord Kelvin said, "*You can't improve what you can't measure*".

We address the following research questions:

- RQ1. How can DevOps measures be defined in conformance with software measurement standards?
- RQ2. Which base measures can be identified in the ISO-32765 DevOps standards?

3. Analysis of the proposed measures

For each of the 15 proposed measures in ISO-32675, we should answer the questions in Figure 2.

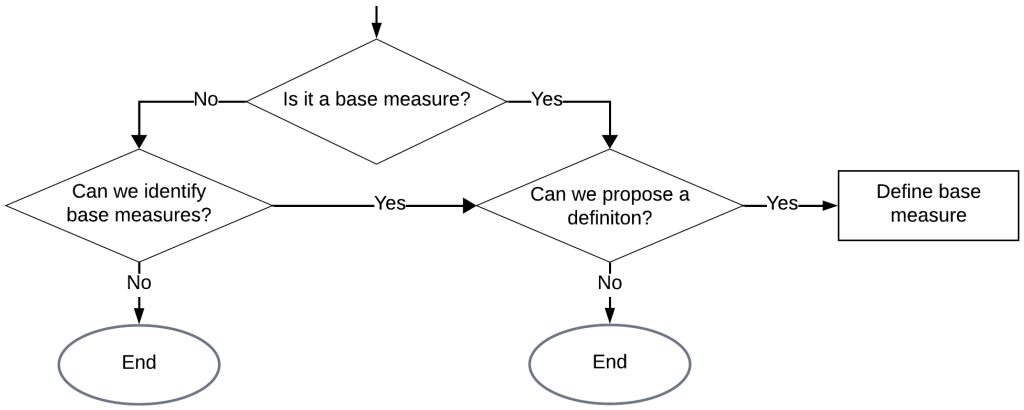

Figure 2: Analysis of the proposed measures

After applying model in Figure 2, we obtained the results shown in Table 2 and Table 3. Table 2 qualifies each measure outlined in the DevOps standard. It first outlines the type of measure, i.e. if it is a base measure, a derived measure, or if it is undefined and it illustrates the base measures associated if applicable. Table 3 will be explained in sub-section 3.1.

Table 2
Qualification of the suggested measures

#	Name of the measure (example)	Type of measure	Base measure(s)
1.	Deployment frequency	Derived	Number of deployment Time period (hours, days, weeks)
2.	Change lead time and volume	Derived	Number of changes (volume) Time duration between x and the change being deployed to customers
3.	Change failure rate	Derived	Number of changes causing a failure Number of changes
4.	Defect rate	Derived	Number of known defects Time period Number of software components
5.	Mean time to recovery (MTTR)	Derived	Recovery of service duration Number of recoveries
6.	Mean time to detection (MTTD)	Derived	Time duration between the detection and the start time of a failure Number of failures
7.	Issue volume (number of issues) and resolution time	Base & Derived	Number of issues Duration of resolution
8.	Time to approval	Derived	Duration of approval Number of items to approve
9.	Time to patch vulnerabilities	Derived	Duration of patching Number of vulnerabilities
10.	Logging availability	Undefined	NA
11.	Retention control compliance	Undefined	NA
12.	Software assurance requirements findings count	Undefined	NA
13.	Application traffic	Undefined	NA
14.	Attack vector details (IP, stack trace, time, rate of attack)	Undefined	NA
15.	Resource utilization	Undefined	NA

In measure 2, change lead time and volume, x can refer to the time when a customer make a request or the time it takes for work to be implemented, tested and delivered into production [13].

The results show that the base measures are not described and therefore not defined and that 6 measures out of 15 can not be qualified: logging availability, retention control appliance, software assurance requirements finding counts, application traffic, attack vector details, and resource utilization.

We also note that there is no mention of the velocity as a measure. The velocity is a derived measure composed of the size over a duration. The measurement function could be, for example, the number of SLOC or the number of function points delivered in a 2-week sprint.

3.1. Definition of the base measures

Figure 1 indicates that to obtain a base measure, one must use a measurement method. According to ISO-15939, a measurement method is "*a logical sequence of operations, described generically, used in quantifying an attribute with respect to a specified scale"* [3, p. 4].

In this section, we analyze each measure and provide, if possible, the name(s) of the base(s) measure(s) and a definition of each base measure (3). The following information will be collected according to the measurement method as described in [3]:

- name of the base measure (a unique name);
- definition. What is intended to know by making the definition of this base measure?; and,
- scale type (nominal, ordinal, interval, or ratio).

To apply the measurement function correctly, it is necessary to know the measurement scale. The scale type is defined in different standards [3, 9], and it describes "*the nature of the relationship between values on the scale*" [3, p. 23]. We will use the four types commonly defined and described in ISO-15939 as:

- nominal: the measurement values are categorical (example: specific role name of team members, such as analyst, programmer, etc.);
- ordinal: the measurement values are rankings (example: degree of satisfaction or priority level to fix an issue);
- interval: the measurement values have equal distances corresponding to equal quantities of the attribute (examples: cyclomatic complexity, temperature); and
- ratio: the measurement values have equal distances corresponding to equal quantities of the attribute where the value of zero corresponds to none of the attribute (example: number of requirements, number of defects, etc.).

Table 3 summarizes the collected information according to the list of measures identified in the DevOps standard.

Table 3
Identification of several base measures according to several standards

Name of the base measure	Definition	Type of scale
Approval (number of)	Written notification by an authorized representative that an information item appears to satisfy requirements and is complete [18].	Ratio
Change (number of)	The modification of an existing application comprising additions, changes and deletions [18].	Ratio
Defect (number of known)	Imperfection or deficiency in a work product or a project component. In generic terms, it can refer to a fault or a failure [18].	Ratio
Deployment (number of)	Phase of a project in which a system is put into operation and cutover issues are resolved [18, 3.1113]	Ratio
Failure (number of)	Termination of the ability of a product to perform a required function or its inability to perform within previously specified limits [4]	Ratio
Issue (number of)	Refer to a wide range of challenges, problems, or concerns that arise during the software development life-cycle (SDLC).	Ratio
Recovery (number of)	The return to normal operation after a hardware or software failure [18].	Ratio
Resolution	Refers to the process of addressing, fixing, or resolving issues, bugs, or problems that have been identified within a software system.	Ratio
Time (duration)	The time in calendar units between the start and finish of a scheduled activity [19].	Interval
Time period	It signifies the real-world time that elapses, independent of the software development process. A time period could represent a number of hours, days, weeks (or sprints), months, etc.	Interval
Vulnerability (number of)	Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source. [20]	Ratio

Table 3 illustrates that most of the base measures list in table 2 are already defined in different known standards.

3.2. Measurement function

The measurement process within the DevOps standard does not propose measurement functions. A measurement function "*is an algorithm or calculation that combines two or more base measures*". For example, the deployment frequency is the number of deployments for a specific time period (i.e. a specific duration).

$$X = A/B \tag{2}$$

where:

X = represents the deployment frequency;
A = represents the number of deployments;
B = represents the time period during which the deployments happened.

In practice, the following example could serve as an expected result: one deployment per week (or per hour, per day, per sprint, per month, per year).

In Table 4, we did propose measurement functions for each of the 9 defined measures.

Table 4
Measurement function of the defined measures

#	Name of the measure	Base measure(s)	Measurement function
1.	Deployment frequency	A. Number of deployments B. Time period	A/B
2.	Change lead time and volume	A. Number of changes B. Time duration	∑B / A
3.	Change failure rate	A. Number of changes causing a failure B. Number of changes	A/B
4.	Defect rate	A. Number of known defects B. Number of software components C. Time period	A/B or A/C
5.	Mean time to recovery (MTTR)	A. Recovery of service duration B. Number of recoveries	∑A / B
6.	Mean time to detection (MTTD)	A. Start time of failure B. The amount of time it takes to identify the origin of a single failure after it occurs C. Number of failures	∑(B - A) / C
7.	Issue volume and resolution time	A. Number of issues B. Duration of resolution	∑B / A
8.	Time to approval	A. Number of items to approve B. Duration of approval	∑B / A
9.	Time to patch vulnerabilities	A. Number of vulnerabilities B. Duration of patching	∑B / A

4. Discussion

4.1. Measuring the performance

A "software development project" usually has a start and end date consistent with its budget. Applying DevOps, many organizations are adopting the "software development product mode", where a budget is established for a time period (e.g. the current year, semester, or month). In both modes (project or product), the size of the related software product is required to measure velocity and productivity, whether this size is physical (SLOC) or functional (function points). However, in modern projects and DevOps, it is quite usual that software system components are written in different programming languages [21, 22]. This makes the number of SLOCs an inappropriate measure of performance unless the performance needs to be measured separately by technology. In most cases, adopting a technology-independent size measurement method, such as COSMIC [23], would be more appropriate. Nonetheless, the size measure is not mentioned directly in the ISO-32675 DevOps standard's list of measures.

The DevOps standard mentioned "work units" in the "velocity" definition [1, p. 14], giving examples such as "story points", which represent "*a largely approximate level of perceived effort*" [24]. Other given "work units" examples in this "velocity" definition are also subjective or incomparable values, such as delivered features, functions, user stories, use cases, or requirements (from one of these work units to another, the size may vary greatly). In [13, chap. 2], the velocity based on user story points is declined as a weak subjective measure linked with various unwanted behaviours from team members.

The product's size and related effort for a given time period must be included to measure productivity. Therefore, the list of proposed measures needs to mention these measures and be updated accordingly. The standard proposes a certain number of measures that are not necessarily linked to each other. A performance model with its characteristics is missing (for example, deployment, security, validation, etc.). This is because, at a high level, we cannot find a model of relationships like the one in quality measures [16]. For this reason, it is difficult to determine whether all measures are relevant or whether some measures are missing. For example, comparing performance between different projects and organizations is difficult if there is no common definition of the performance measures. Some performance-related measures are not available to measure velocity and productivity, which are also part of the performance, namely the software development process performance.

4.2. Measuring the quality

Several quality-related measures are proposed in the DevOps standard. It is expected that quality measures be used to improve quality aspects of the software development process and of the software product. In that case, the "Defect density" measure could be added to the DevOps standard, as the number of known defects per software size, such as per 100 COSMIC Function Points (CFP). Also, the evolution of the value of the set of quality-related measures should be looked at over time to observe their variation and confirm if the software process improvements are bringing the expected results.

4.3. Comments on measuring performance and quality

One can argue that measuring the delivered software's functional size is a manual task that would take extra effort. Forsgren et al. [13, chap. 4] define a list of five (5) DevOps key principles, where the third one relates to the automation of any repetitive manual task. Functional size measurement should also be automated under this key principle. There are available tools that automate software functional size in COSMIC function points from several sources: requirements written in natural language (French and English) [25], Java code developed with the Spring MVC framework [26], and Java code of business applications [27, 28]. Also, current research projects aim to automate functional size measurement from other DevOps artifacts (for example, automated tests and API requirements) [21].

4.4. Threats to validity

This article aims to answer those two questions:

- RQ1. How can DevOps measures be defined in conformance with software measurement standards?
- RQ2. Which base measures can be identified in the ISO-32765 DevOps standards?

To answer the question of compliance of measurement standards in software engineering, we consulted three recognized standards, including two in software engineering [3, 4] and one based on the metric system or SI [9]. These three standards emphasize the importance of using a measurement method to define a base measure and a measurement function method to define a derived measure. The DevOps standard has been shown not to use these methods. It only lists the measures without qualifying them (base or derived), see Table 1.

To answer the second question, we analyzed the proposed measures to find the base measures that could be extracted from the measures listed (Table 2, "Name of the measure" column) and, when possible, define them (Table 3).

We went further and applied, where possible, the measurement function method (Table 4). We found that we could apply the functional measurement method to only sixty percent (60%) of the measures (9 out of 15) listed in the DevOps standard, meaning that 6 measures out of 15 (40%) were too ambiguous to be defined.

Many other measures are proposed for DevOps in different publications [29, 30, 13, 31]. They propose different measures adapted to the immediate situation and do not refer necessarily to their related measurement method and measurement function.

5. Conclusion, recommendations and future works

The actual measurement process in the DevOps standard shows that the measurement process needs to be more mature. To improve this measurement process, it is necessary to:

- Better define the base measures using a measurement method [3];
- Better define the derived measures using a measurement function;
- Avoid ambiguous measures, i.e. the ones that are difficult to classify as base measures or derived measures;

- Provide a performance model relationship, like SQuaRE quality model [15].

DevOps measures can be defined in conformance with software measurement standards. Table 2 shows that DevOps measures can first be categorized as base measures, derived measures, or indicators. Table 3 shows that base measures identified according to Figure 2 can be defined. Finally, Table 4 summarizes the list of base measures identified in the DevOps standard and the measurement functions associated with the ISO-32765 DevOps measure.

When DevOps becomes more mature, it will be necessary to have a global view of the measurement processes like those in SQuaRE [15, 16]. This could take the form of a DevOps measurement model suggesting a number of measures to be calculated automatically. A suggestion of tools could be provided, but not without considering the rapid evolution of tools. In future works, researchers could be interested in applying Blanco's telemetry idea, using the measurement method and measurement function approaches.

References

[1] I. O. for Standardization, Iso/iec/ieee 32675:2022 information technology – devops – building reliable and secure systems including application build, package and deployment, 2022.

[2] I. O. for Standardization, Iso/iec 12207:2017 systems and software engineering – software life cycle processes, 2017.

[3] I. O. for Standardization, Iso/iec/ieee 15939 systems and software engineering – measurement process, 2017.

[4] I. O. for Standardization, Iso 25021:2012 systems and software engineering – systems and software quality requirements and evaluation (square) – quality measure elements, 2012.

[5] I. O. for Standardization, Iso 25022:2016 systems and software engineering – systems and software quality requirements and evaluation (square) – measurement of quality in use, 2016.

[6] I. O. for Standardization, Iso 25023:2016 – systems engineering – systems and software quality requirements and evaluation (square), 2016.

[7] I. O. for Standardization, Iso 25024:2015 – systems engineering – systems and software quality requirements and evaluation (square), 2015.

[8] I. O. for Standardization (ISO), I. E. C. (IEC), Iso/iec ts 25025:2021 – information technology – systems and software quality requirements and evaluation (square) – measurement of it service quality, 2021.

[9] I. Vim, International vocabulary of basic and general terms in metrology (vim), International Organization 2004 (2004) 09–14.

[10] J.-M. Desharnais, A. Abran, et al., Software measurement methods: An analysis of two designs, Journal of Software Engineering and Applications 5 (2012) 797.

[11] I. O. for Standardization, Iso 9126:2001 – software engineering – product quality characteristics and their measures, 2001.

[12] J.-M. Desharnais, A. Abran, W. Suryn, Identification and analysis of attributes and base measures within iso 9126, Software Quality Journal 19 (2011) 447–460.

[13] N. Forsgren, J. Humble, G. Kim, Accelerate: the science of lean software and DevOps: building and scaling high performing technology organizations, IT Revolution, 2018.
[14] N. Fenton, J. Bieman, Software Metrics: A Rigorous and Practical Approach, 3rd ed., Project Management Institute, 2017.
[15] I. O. for Standardization, Iso 25000:2014 systems and software engineering – systems and software quality requirements and evaluation (square), 2014.
[16] I. O. for Standardization, Iso 25010:2023 systems and software engineering – systems and software quality requirements and evaluation (square), 2023.
[17] D. G. Blanco, Practical OpenTelemetry: Adopting Open Observability Standards Across Your Organization, Springer, 2023.
[18] I. O. for Standardization, Iso 24765:2017 systems and software engineering – vocabulary, 2017.
[19] P. M. Institute, A Guide to the Project Management Body of Knowledge (PMBOK® Guide), 6th ed., Project Management Institute, 2017.
[20] N. I. of Standards, T. (NIST), vulnerability - glossary | csrc, 2023. URL: https://csrc.nist.gov/glossary/term/vulnerability.
[21] A. Lapointe-Boisvert, S. Mosser, S. Trudel, Towards modelling acceptance tests as a support for software measurement, in: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2021, pp. 827–832. doi:10.1109/MODELS-C53483.2021.00129.
[22] P. Rani, S. Panichella, M. Leuenberger, A. Di Sorbo, O. Nierstrasz, How to identify class comment types? a multi-language approach for class comment classification, Journal of systems and software 181 (2021) 111047.
[23] I. ISO, Iso/iec 19761:2011 software engineering — cosmic: a functional size measurement method, 2011.
[24] S. Trudel, Agile & cosmic: A good integration!, 2019.
[25] B. Gérançon, S. Trudel, R. Kkambou, S. Robert, Software functional sizing automation from requirements written as triplets, ICSEA 2021 (2021) 33.
[26] A. Sahab, S. Trudel, Cosmic functional size automation of java web applications using the spring mvc framework., in: IWSM-Mensura, 2020, p. 7.
[27] M. A. Sag, A. Tarhan, Measuring cosmic software size from functional execution traces of java business applications, in: 2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, 2014, pp. 272–281. doi:10.1109/IWSM.Mensura.2014.29.
[28] N. Chamkha, A. Sellami, A. Abran, Automated cosmic measurement of java swing applications throughout their development life cycle., in: IWSM-Mensura, 2018, pp. 20–33.
[29] G. Kim, J. Humble, P. Debois, J. Willis, N. Forsgren, The DevOps Handbook, Second Edition: How to Create World-Class DevOps Teams and Culture, 2 ed., IT Revolution Press, 2016.
[30] D. Research, A. D. Project, Dora | devops research and assessment, https://dora.dev/, 2024.
[31] PuppetLabs, State of devops report. technical report, puppetlabs, 2021.