

# Using weekly open defect reports as an indicator for software process efficiency

## Theoretical framework and a longitudinal automotive industrial case study

Niklas Mellegård

RISE Viktoria

niklas.mellegard@ri.se

### ABSTRACT

Well-defined, informative and cheap indicators are important in any software development organization that needs to evaluate aspects of its development processes and product quality. This is especially true for large organizations and for organizations developing complex products; for example automotive safety functions where mechanical, electronic and software systems need to interact. In this paper we describe defect backlog profiles as a well-defined, cheap and informative indicator. We define defect backlog profiles in terms of ISO/IEC 15939, provide a theoretical framework for interpretation, and finally present an evaluation in which we applied the indicator in a longitudinal case study at an automotive manufacturer. In the case study, we compare the software integration defect backlog profile for the active safety component released in 2010 to the profile for the following generation of the same component released in 2015. The results are then linked to a number of process and product changes that occurred between the two product generations. We conclude that defect backlog profiles are cheap in terms of data collection and analysis, and can provide valuable process and product quality information although with limitations.

### CCS CONCEPTS

- **Software and its engineering** → **Software defect analysis**;
- *Information systems* → *Data mining*

### KEYWORDS

defect backlog profile, indicators, ISO/IEC15939, time-to-market, continuous integration, agile development process

### ACM Reference format:

N. Mellegård 2017. Using weekly open defect reports as an indicator for software process efficiency. In *Proceedings of IWSM/Mensura '17*, October 25–27, 2017, Gothenburg, Sweden (MENSURA '17) DOI: 10.1145/3143434.3143463

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [Permissions@acm.org](mailto:Permissions@acm.org).  
*IWSM/Mensura '17, October 25–27, 2017, Gothenburg, Sweden*

© 2017 Copyright is held by the owner/author(s). Publication rights licensed to ACM.  
ACM ISBN 978-1-4503-4853-9/17/10...\$15.00  
<https://doi.org/10.1145/3143434.3143463>

### 1 INTRODUCTION

Defect reports are a valuable source of measurement data, as these reports are typically available in any project of complexity regardless of whether there was an intention to measure or not—the data is essentially available at no additional cost.

Although inevitable during development, defects are typically undesirable as they consume development resources that could otherwise be spent on, for example, ensuring faster time-to-market or developing more sophisticated functionality; analysis of defect reports can thus reveal important information about various aspects of process efficiency and product quality.

However, there are many attributes in defect reports that can be measured and many ways to analyse and interpret them. Adding to the challenges, defects and how these are reported are highly contextual; certainly between development organizations, but possibly also between projects within an organization, and even depending on the development phase.

Defect backlog profiles (DBP) [1], [2] are one way of analysing defect reports, where the number of open defects per time period is counted and plotted as a graph. The shape of the graph can reveal information about the timing of increased workload due to defects. This, in turn, can provide evidence regarding efficiency and effectiveness of development practices, although interpretation requires contextualization in terms of the specific development characteristics of the project in focus.

In this paper, we address the research question

RQ *Can Defect Backlog Profiles reveal relevant and valid information about a software development process?*

We address the research question by first defining defect backlog profiles in terms of ISO/IEC 15939 [3], and then by providing using related work, a theoretical framework for interpretation. For validation, we present a longitudinal industrial case study in which DBP was applied, and the results interpreted according to the theoretical framework.

In the case study, we used DBP as an indicator to assess effects of process improvement between two generations of the ADAS (Advanced Driver-Assistance System) for a car manufacturer. In 2010 a DBP was generated for generation 3 of the ADAS node, and interviews were conducted with involved staff [2]. With generation 4 of the node, in addition to major functional growth, there were considerable platform and process changes. With the release of the fourth generation of the ADAS node, we followed up on the 2010 study by generating a DBP and a new round of interviews.

## 2 BACKGROUND AND RELATED WORK

This section defines defect backlog profiles as an indicator and provides a theoretical framework for interpretation.

### 2.1 Defect Backlog Profiles

Defect backlog profiles are generated by aggregating the number of open defects per time period (per week in our case), calculated by extracting the creation and closing dates for each defect. The profile is then generated by plotting a graph of open defect reports over the project life-cycle (see Figure 5).

In terms of ISO/IEC 15939 [3] (fig. A.1, p. 21), extracting creation and closing dates from the defect reports (the *Entity*) corresponds to applying a *Measurement Method*, to arrive at *Base Measures*. The *Measurement Function* calculates the defect life-time as a *Derived Measure*. The *Analysis Model* aggregates the number of open defects per designated time slot over the project life-time, which constitutes the *Indicator*. The indicator can be plotted as a graph to facilitate interpretation. In section 2.2, we elaborate on what *Information Need* DBP can contribute to, and provide a theoretical framework to support interpretation.

### 2.2 Development processes and Defect Backlog Profiles

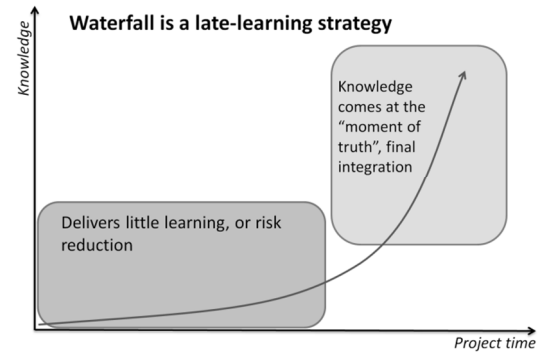
The DBP graph provides a direct indication of peaks in workload due to defects, as an open defect requires resources for analysis and resolution [4]—thus contributing to an undesired workload.

Ideally, a DBP graph should resemble a bell curve [1], [5] (assuming defects are promptly resolved), as in initial project phases there is heavy feature development with limited focus on testing. Once the product is approaching feature completion, V&V typically intensifies and defects are reported in larger numbers, resulting in an increase in open defects (comprising the bell curve peak). In the final phases of the project, fewer undetected defects are left in the product and work is focused on resolving the ones found, thus the number of open defects should decline.

This ideal shape is rarely the case in reality, where any number of contextual factors may influence the defect inflow. Still, it is a well-established fact that identifying and resolving defects in early development phases are less costly than in later phases [6]. Thus, a DBP with a peak skewed to the left would be preferred to one skewed to the right. More precisely, the peak should be close in time after the point of feature completion; an earlier peak might indicate lack of testing, while a later peak might suggest insufficient testing in earlier development phases.

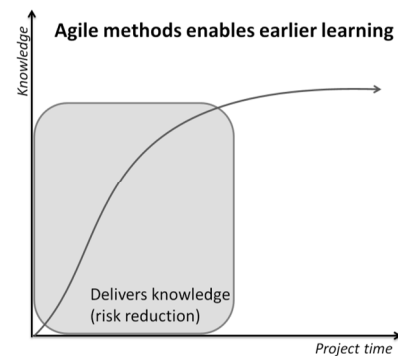
From a theoretical perspective, the development process model in use should reasonably affect the shape of the DBP. Consider waterfall process models compared to agile process models where continuous integration is a key enabler.

With waterfall process models the development phases are typically performed in sequence. This has the effect that assumptions made during the specification and design phases are evaluated late. Waterfall processes can therefore be considered late-learning strategies, as illustrated in Figure 1 [7].



**Figure 1. Knowledge-growth with a big-bang integration strategy (adapted from Cockburn [7])**

Agile process models on the other hand, typically iterate the phases throughout the project, thus providing shorter feedback cycles. Issues can therefore be detected and resolved earlier [7], thus enabling earlier learning as illustrated in Figure 2.



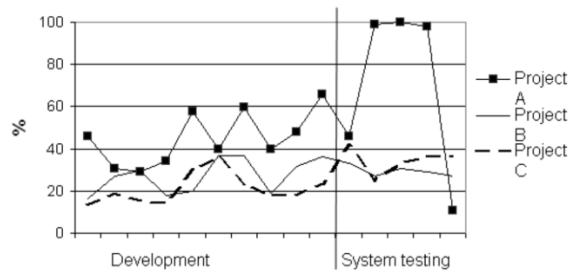
**Figure 2. Knowledge-growth with an early integration strategy (adapted from Cockburn [7])**

*Learning* or *risk reduction*, in the context of this argumentation, is essentially testing the product or part of it in its operational environment (or a sufficiently precise approximation)—it is testing that facilitates the developers to learn whether the system, with its current requirements, design and implementation, behaves as intended. In that sense, detected defects are an indication of the acquisition of new knowledge, as they signify a deviation from what is expected [8]. Therefore the shape of the DBP can be used to assess the effectiveness of the development process to deliver knowledge in time about the product in its operational environment. Presumably, this would mean that a DBP graph for a waterfall process should have peaks in late project phases whereas an agile process should have a flatter graph.

Support for this presumption is provided by Korhonen who examined the defect inflow profile<sup>1</sup> for an organization that transitioned from a waterfall to an agile development model.

<sup>1</sup> The defect inflow profile measures the number of newly created defect reports per week. In the special case that all defects are created and closed in the same week, an inflow profile would be identical to a DBP. More importantly in the current context, peaks in defect inflow would be present in both types of profiles.

Figure 3 shows the inflow profile for three projects developed by the same team. Project A followed a waterfall process model, and projects B and C an agile. From the figure it is evident that there is a large inflow of defects in the system testing phase; this is consistent with the late learning strategy (see Figure 1). Conversely, for projects B and C, no such peak is present; consistent with the early learning enabled by the agile process, as argued above.



**Figure 3. Defect inflow profiles (from [9]). Project A used a waterfall model, while in B and C the organization had transitioned to agile methods. The vertical line represents the feature-freeze milestone.**

### 3 CASE STUDY

A longitudinal case study was conducted at the active safety department of a car manufacturer (OEM, Original Equipment Manufacturer). In the study, the development of the initial releases for generations 3 and 4 of the active safety node (ADAS node) was examined. The first observation took place in 2010 [2] and the second in 2017.

Defect backlog profiles were used in the study for the purpose of assessing effects of process and product changes that were implemented before the development of generation 4 of the ADAS node. For each of the two observations in the study, interviews with managers, developers and testers were done to support interpretation of the DBP.

While the study revealed several interesting results, in this paper the case study is mainly used for assessing DBP as a viable indicator, and as an experience report of applying DBP.

In the following sections, we briefly describe automotive software systems and their development, and then describe in more detail the development of the ADAS node; more specifically, contrasting the development of generation 3 and 4.

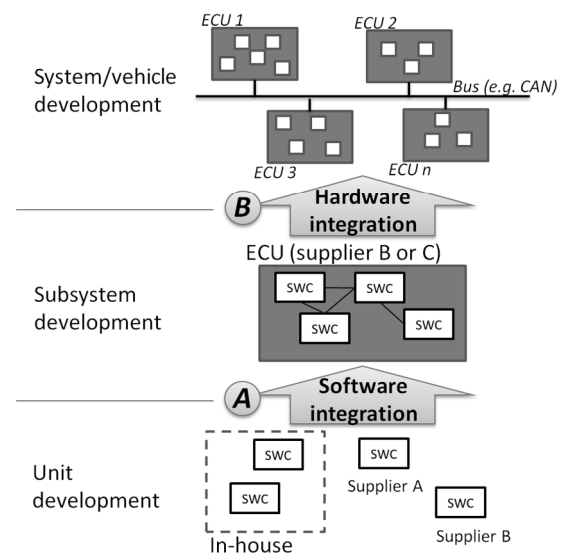
#### 3.1 Automotive Software

Automotive active safety systems present a number of specific software engineering challenges. The software does not exist in isolation, but is rather part of a cyber-physical system with strong dependencies on mechanical and electrical systems—each with different development characteristics. For example, an active safety function such as collision avoidance will need to identify relevant objects in front of the vehicle using various sensors, to distinguish in a complex traffic situation what presents a potential danger, and to actuate the brake system with appropriate force. Not only does this present algorithmic

challenges, proper functioning is also dependent on the quality of sensors—including their potential degradation over time or due to specific weather and light conditions—as well as on the characteristics of the brake system which in turn is dependent on the dynamics of the complete vehicle. Adding to the challenges, the operational environment is complex—for instance, the variety of road types, traffic participants, light and weather conditions.

Testing such systems is challenging, not only due to the sheer size of the test space, but also due to the fact that realistic prototypes or simulation models of mechanical, electronic and software components are required sufficiently early in the development cycle to allow extensive testing.

Considering the technical implementation of such systems in more detail: a vehicle contains a distributed computer system, with ECUs (Electronic Control Unit) connected by a number of communication buses (see the top layer of Figure 4). Attached to an ECU, there may be a set of sensors (e.g. a camera) and actuators (e.g. ability to trigger the brake system). An ECU has a set of responsibilities in terms of functionality, while a function (e.g. the collision avoidance function) may have functionality deployed on a number of ECUs.



**Figure 4. Overview of software development, and its deployment on the vehicle platform**

Internally, an ECU has an operating system (typically AUTOSAR compliant [10]) and a number of software components (SWC) deployed (illustrated in the middle layer of Figure 4). The SWCs typically realize functionality (such as the collision avoidance function). Normally, a supplier (typically a tier-1 supplier) delivers the hardware together with the operating system. The SWCs deployed on an ECU may be developed in-house or delivered by suppliers (illustrated in the bottom layer of Figure 4). There are thus two levels of integration needed: A) Software integration, where the SWCs are integrated into the ECU (integration point A in Figure 4), and; B)

Hardware integration, where the ECU is integrated with the vehicle system (integration point B in Figure 4).

### 3.2 The ADAS Node

In this paper we have studied the development of the software for the ECU realizing the Advanced Driver-assistance System (ADAS) in two subsequent generations. The ADAS node is responsible for realizing various active safety and driver support functions, such as collision warning, lane keeping aid, and parking assistant. In addition, the node is responsible for a number of sensors (e.g. camera, radar and lidar) and for processing the sensor data.

**Generation 3** of the ADAS (ADAS3) node was released in 2010 and developed according to the typical V-process model [11]. The majority of SWCs were specified by the OEM, but developed by suppliers. The ECU supplier was responsible for the software integration (integration point A in Figure 4), while the OEM was responsible for vehicle integration (point B).

The vehicle platform on which ADAS3 was developed had been in production and continuously improved on for over 10 years. There was therefore much in-house and supplier experience with the platform, as well as many vehicle prototypes on which to experiment during development.

With **generation 4** of the ADAS node (ADAS4), the OEM replaced the vehicle platform. The platform change has internally been described as one of the larger steps taken by the OEM. For ADAS4, this included an entirely new system architecture.

In the same move, the decision was made to develop more of the key software components in-house; prompting the department to adopt software engineering best practices, such as shared software repositories, and automatic build servers. In addition, the need to do earlier and more frequent testing had been identified. Although still applying the V-model, steps were taken to enable earlier and more frequent integration by performing software integration in-house (integration point A in Figure 4). This enabled shorter feedback cycles for the software developed in-house.

Comparing ADAS3 and ADAS4, the functional growth was (according to interviewees) considered higher in ADAS4 than in ADAS3, with more functionality as well as functionality with a higher level of autonomy [12].

### 3.3 Case Study Design

The longitudinal case study was conducted with two observations. The object under study was the development of the ADAS node, where two subsequent generations of the node were observed (released in 2010 and 2015 respectively). Data collection—defect reports and interviews—was conducted in the same way in both observations.

The DBP data comprised software defect reports concerning the ADAS node reported in tests done *after* integration point A (see Figure 4). Note that this does *not* include defects found during unit testing, and that the reports include any issue that may have impact on software components. No additional

filtering was made. The same issue reporting system and procedures was in use in both observations. The defect data were collected and analysed to produce a DBP.

After generating DBP graphs, semi-structured interviews were conducted in order to provide qualitative data on the resulting DBP. The interviewees were first asked open-ended questions about their experiences from the project. They were then presented with the DBP with commentary from the researchers, and asked to relate the shape of the profile to the experiences from development.

For each of the two observations, three interviews were conducted. In both observations, the quality manager and the lead engineer responsible for the entire ADAS node, and one function developer responsible for developing a feature (i.e. a set of SWCs) deployed on the ADAS node were interviewed. Notes were taken during the interview for analysis, and all conclusions were later validated with the interviewees. For the first observation, focus of the interview was on the late peak in the DBP (see Figure 5). For the second observation, the focus was to contrast the DBPs from the two generations of the node, and to relate the differences to changes in the development practices.

## 4 RESULTS

The results are presented here in the form of a DBP and a sample of interview notes for ADAS3 and ADAS4 respectively. The results are then interpreted and discussed in the section 5.

### 4.1 ADAS Generation 3

Figure 5 shows the DBP for the ADAS3 node. As can be seen in the graph, there seems to be an apex just after the *Feature Complete* milestone as expected. However, there is also a significant peak late in the project, coinciding with the *Pre-production* milestone.

This late peak seems consistent with a waterfall development process, as argued previously. The late-learning hypothesis is supported by the timing of the peak, as it coincides with the first vehicle built with production hardware—suggesting that V&V activities in earlier project phases were not sufficiently effective. Interview results further support the hypothesis, where one developer stated “*to some extent, there was no point in testing [note, on subsystem and system level] that much before we had the production hardware; we wouldn't have gotten much useful results before that*”.

Another contributing factor, found in the interviews, was related to the integration process. As previously described, the software integration (integration point A in Figure 4) was done by the supplier at predetermined intervals (typically, in the order of several weeks apart). This had the effect that software changes (whether in-house or by suppliers) could not easily be tested until the next delivery cycle—effectively prolonging the feedback loop for such changes.

The late peak as a significant contributor of undesired workload was also confirmed in the interviews. In fact, one quality manager asked to use Figure 5 as part of a report, as it

was considered to provide evidence for what had previously been a notion in ADAS3, as well as in prior projects.

Furthermore, in a separate study [13], we conducted a deeper examination of a sample of defects from the late peak. From that study, it seemed that the defects were not predominately typical integration issues, but rather issues that could have been identified on unit test level. This further suggests that ineffective early V&V activities may have contributed to the late peak in the DBP graph, where shorter feedback cycles for software changes might be a promising target for improvement.

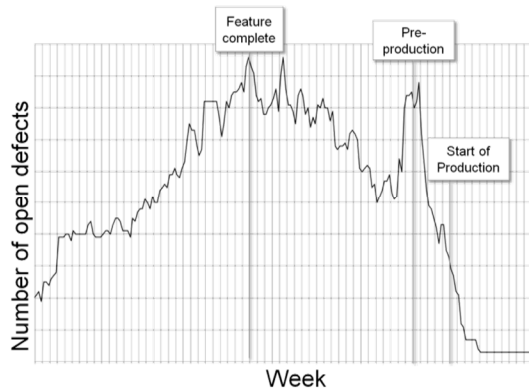


Figure 5. DBP from ADAS generation 3

#### 4.2 ADAS Generation 4

Figure 6 shows the DBP for the ADAS4 node. As can be seen in the graph, the shape is similar to the one for ADAS3 but with the late peak notably absent.

Note that there seems to be a considerable amount of open defects at start of production. This is not to be taken to mean that the product is released with defects—rather, these may be deviations from requirements where the product is good enough for release, but open defect reports are kept in order to track updates for future (service) releases.

The interviewees described the development of ADAS4 as very challenging mainly due to the new vehicle platform, but also due to applying new ways of working.

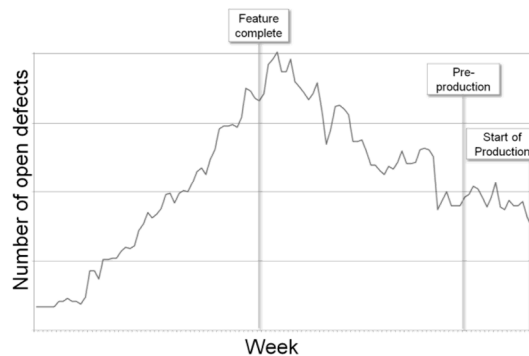


Figure 6. DBP from ADAS generation 4

Comparing to the DBP from ADAS3, interviewees confirmed that the workload in the later development phases were indeed

less challenging in generation 4 than in generation 3. When asked what the main contributing factor was, interviewees were in agreement that the ability to do in-house software integration—internally referred to as “short-loops” (integration point A in Figure 4)—was significant. In addition to allowing the in-house software engineers to get fast feedback on software changes, it also enabled automatic (continuous) software integration and testing on (sub-)system level.

## 5 DISCUSSION

DBP are cheap in terms of data collection, as defect reports presumably are available digitally in any project of complexity, and as the attributes needed is creation and closing dates of the reports. Thus, data extraction should be a trivial operation. Furthermore, the DBP graph is informative as it is easy to understand when mapped to project milestones—this was to some extent validated as the interviewees could easily map the DBP shape to project experiences.

However, in addition to being mainly post-hoc rather than in-process, the interpretation and generalization of DBP is limited. The DBP does most likely not produce results that can be directly compared between organizations, or that lends itself to easy scaling from good to bad. Rather, interpretation requires contextualization and intellectual assessment. This also has an impact on generalization. For one, defect reporting procedures may differ between organizations and the V&V process will have a direct impact on the shape of the DBP. Also, the characteristics of the product most likely have an impact on the shape—it may not make sense to contrast a DBP from a pure software product, such as a web site, with one from a cyber-physical system, such as an automotive ADAS node. Furthermore, the information provided by DBP should be considered quantitative as it does not provide any information about the nature of the shape. As case in point, while it is evident from Figure 5 that there is late peak in workload, no clues are given as to what caused it or what type of defects it comprised (other means are needed, such as defect classification [1], [14]), or even if there was a significant increase in required development resources—in our case study, the latter was confirmed through interviews.

Still, there might be some basic universality to the DBP shape; the ideal bell shape as argued in section 2.2, and the fact that defects found early are typically cheaper to resolve [6]. Moreover, general characteristics of the development process should reasonably influence the shape. As described in 2.2, with a late-learning process an increase in late defects are expected, while a flatter DBP would be expected in a more agile process where the product is built and tested incrementally. It is therefore conceivable that a DBP could be used to gauge the effectiveness of the process to deliver early risk reduction.

Incidentally, support for this can be found in the Korhonen study (see Figure 3), although the focus of the study was to examine changes in defect reporting practices when transitioning from a waterfall to an agile development process. Their graph (Figure 3), showing defect inflow rather than a DBP as defined in this paper, indeed shows a late increase in defect

reports late for the waterfall project. For the agile projects, however, flatter shapes can be seen.

Further support for the influence of development practises was found in our case study, where the interviewees stated that a significant improvement in the ADAS4 development process had been the short-loops; essentially continuous integration on subsystem level. This allowed engineers to build and test software in shorter cycles; enabling incremental development. Interestingly, the late peak in workload observed in ADAS3 was absent in the DBP from ADAS4. Still, firm conclusions cannot be drawn as there were a large number of platform and process changes done between the development of ADAS3 and ADAS4.

Even though interviews are required in order to gain understanding about the reasons behind the shape of a DBP, interviews does not necessarily make DBP obsolete. On the contrary, the quantitative nature of a DBP would complement interviews with more objective evidence. In addition, compared to interviews DBP require considerably less effort and can be generated automatically as part of the project reporting. Results could, for instance, be used to better target interviews.

## 5.1 Validity Evaluation

The main threats to validity concerns the case study, as it is used to validate DBP as viable indicator. The main threats are to internal, construct and external validity [15]. Regarding *internal validity*, there were a considerable number of changes in the platform, process, and people involved between the observations. It is likely that the shape of the DBP was influenced by factors other than the ones considered. To address this, we provide a theoretical model for interpretation, find support in independent studies, and triangulate with interviews. To mitigate the threat to *construct validity*, we applied the same method in both observations, although different individuals were interviewed in first and second observation. Regarding *external validity*, we can make no claims regarding generalizability as discussed in the previous section. However, our findings are in-line with the theoretical framework [5], [7], [9]. In addition, a thick case description is provided to support in assessing the transferability to other contexts [16].

## 6 FUTURE WORK

This paper presents preliminary results from an on-going study at a vehicle OEM. While the focus of the present paper was on defect backlog profiles as a viable indicator of certain aspects of process efficiency, the larger aim of the on-going study is on effects of adopting software development best practices in an organization that previously mainly outsourced its software development. As part of the larger study we will conduct deeper interviews investigating in more details the platform and process changes done between generation 3 and 4 of the ADAS node, and link these changes to observed effects, as well as their influence on the DBP.

## 7 CONCLUSION

In this paper we have examined defect backlog profiles (DBP) as a well-defined, cheap and informative post-hoc indicator for undesired workload. We have defined DBP in terms of ISO/IEC 15939, provided a theoretical framework linking DBP to generic development process characteristics, and provided validation using in a longitudinal industrial case study.

From the case study, we conclude that DBP is a cheap indicator in terms of data collection and analysis. DBP is an informative indicator that is easy to understand, and does seem to reflect relevant development process characteristics. While cheap and informative, DBP has limitations in addition to being post-hoc rather than in-process; interpretation requires contextualization and intellectual assessment.

## ACKNOWLEDGMENTS

This work was done as part of NGEA (Next Generation Electrical Architecture) funded by the Swedish innovation agency VINNOVA (proj.no. 2015-04881). The author would like to thank the OEM and the people that made their time available to participate in study design and interviews as part of the research for this paper; especially Kenneth Lind, Håkan Burden, and Ana Magazinius for valuable input and support.

## REFERENCES

- [1] N. Mellegård, Improving Defect Management in Automotive Software Development, LiDeC—A Light-weight Defect Classification Scheme. Institutionen för data- och informationsteknik, Chalmers Univ. of Tech. 2013.
- [2] N. Mellegård, M. Staron, and F. Törner, *A Light-weight Defect Classification Scheme for Embedded Automotive Software and its Initial Evaluation* presented at the IEEE International Symposium on Software Reliability Engineering (ISSRE 2012), Dallas, Tx USA, 2012.
- [3] International Organization for Standardization, *ISO/IEC 15939 - Systems and Software Engineering – Measurement Process* International Organization for Standardization, 01-Aug-2007.
- [4] P. Hooimeijer and W. Weimer, *Modeling Bug Report Quality* in Proceedings of the Twenty-second IEEE/ACM Intl. Conference on Automated Software Engineering, New York, NY, USA, 2007, pp. 34–43.
- [5] S. H. Kan, *Metrics and Models in Software Quality Engineering*, 1st ed. Addison-Wesley Professional, 1995.
- [6] B. W. Boehm, *Software Engineering Economics*, 1st ed. Prentice Hall, 1981.
- [7] A. Cockburn, *Disciplined learning: The successor to risk management* in CrossTalk, vol. 27, no. 4, pp. 15–18, 2014.
- [8] IEEE, *IEEE Std. 1044-2009. Standard Classification for Software Anomalies*, 09-Nov-2009.
- [9] K. Korhonen, *Evaluating the Effect of Agile Methods on Software Defect Data and Defect Reporting Practices - A Case Study* in 2010 Seventh International Conference on the Quality of Information and Communications Technology, 2010, pp. 35–43.
- [10] AUTOSAR, <http://www.autosar.org>, 23-May-2017. [Online]. [Acc. 23-May-2017].
- [11] K. Forsberg and H. Mooz, *The Relationship of Systems Engineering to the Project Cycle* in Eng. Manag. J., vol. 4, no. 3, pp. 36–43, Sep. 1992.
- [12] SAE International, *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles* SAE International, Standard J3016\_201609, 2016.
- [13] N. Mellegård, M. Staron, and F. Törner, *A Light-Weight Defect Classification Scheme for Embedded Automotive Software* in Chalmers University of Technology, Göteborg, Technical Report 2012:04, ISSN:1654-4870, 2012.
- [14] R. Chillarege et al., *Orthogonal defect classification-a concept for in-process measurements* IEEE Trans. Softw. Eng., vol. 18, no. 11, pp. 943–956, 1992.
- [15] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*. Boston MA: Kluwer Academic Publisher, 2000.
- [16] R. K. Yin, *Case Study Research: Design and Methods*, 4.ed. London SAGE '09.