aODC: Agile Orthogonal Defect Classification and Analysis for Quality Improvement

Eric Abuta[1] and Jeff Tian[2*]

[1] Client Computing Group, Intel Corporation, Portland, Oregon, USA.
[2] Department of Computer Science, Southern Methodist University, Dallas, Texas, USA.

* Corresponding author. Tel.: +1 214-768-2861; email: tian@smu.edu
Manuscript submitted March 4, 2025; revised April 11, 2025, accepted April 28, 2025, published May 22, 2025.
doi: 10.17706/jsw.20.1.24-36

Abstract: In-process feedback is essential in providing useful information to stakeholders to improve software quality in an agile development environment, where development decisions are often made quickly with access to limited measurement data and timely feedback is needed for progress monitoring and for quality assurance, all under fluid, rapidly changing market conditions. This study adapts the original Orthogonal Defect Classification (ODC), initially developed and deployed in large commercial software systems following the traditional waterfall process, to aODC, or our adapted ODC for agile development, and demonstrates its ability to provide valuable in-process feedback for a semiconductor software using data that is normally available from the agile development process in a small company. To assess the impact of aODC, 1) we first define our defect and quality metrics, including total defect count, in-field defects discovered by customers, defect distribution, product reliability, and reliability growth; 2) we then quantify the baseline using these metrics for the early versions of this software system prior to the deployment of aODC; and 3) lastly, we quantify the quality improvement using the same metrics after aODC deployment. The comparison results show: 1) a more than 50% reduction in total defects and a 16% reduction of defects found by customers; 2) a significantly higher share of defects discovered in the early part of the process by the developers, at 38%, up from 24.7% in the baseline, and a significantly lower share of defects discovered later by the testers, at 46.5%, down from 66.3% in the baseline; and 3) a higher reliability, with a success rate of 0.914 compared to 0.884 in the baseline, and a more significant reliability growth, quantified by the purification level of 0.99 as compared to 0.91 in the baseline. These results demonstrate that aODC, our adapted ODC to the agile development environment, offers valuable early in-process feedback leading to quantifiable quality improvement.

Keywords: Orthogonal Defect Classification (ODC), defect analysis, software reliability, agile development, in-process feedback and improvement.

1. Introduction

Defect classification and analysis is a useful process of classifying defects into meaningful categories tailored for specific software products or systems under study, and then analyzing them to identify problematic areas, usually those associated with abnormally high numbers or shares of defects, for correction, mitigation, and quality improvement. This can be achieved by using techniques such as Orthogonal Defect Classification (ODC), a systematic classification scheme based on defect data related to execution failures, internal faults, as well as possible causes, collected from testers, inspectors, and developers, to provide in-

process feedback to stakeholders [1]. ODC has been successfully used in large scale software development projects, where the traditional waterfall process is typically used, in identifying problems and improving software quality. It can potentially be adapted to improve quality for various other systems, as demonstrated by studies adapting ODC for NASA (National Aeronautics and Space Administration) and Web-based applications [2, 3]. However, ODC has not been adapted to provide the same type of benefit for small companies who predominantly follow some variation of the agile development process to deal with the rapidly changing market environment, compressed development schedule, and limited access to measurement data [4, 5].

This study uses the standard defect data normally available to many small software development organizations to carry out an adapted ODC analysis. By systematically pairing the data availability with the original ODC data attributes and attribute values, we develop aODC, our adapted ODC suitable to the agile development environment, to meet our needs for quality assurance and improvement. The defect data, used as the basis for developing our aODC and as a case study to validate our approach, are from May 2001 through June 2014. This study is a continuation of our previous work on defect analysis and software reliability modeling for a semiconductor Optical Endpoint Detection (OED) software system [6–8].

The remainder of this paper is organized as follows: Section 2 describes the background of this study and provides a problem statement. Section 3 outlines our solution strategy and research methodology, as well as our defect/quality metrics and the hypotheses to be used to validate our approach. Section 4 presents our aODC, or the adapted ODC for agile development. Section 5 establishes the baseline for defects by analyzing the defect data for product versions before the deployment of aODC. Section 6 presents the results on defect metrics after the deployment of aODC, comparing them to the baseline. Section 7 compares results using our quality metrics before and after the deployment of aODC. Section 8 summarizes our paper, discusses its implications, and outlines possible future work.

2. Background and Problem Statement

In contrast with large software development companies, where the traditional waterfall process is typically used and the corresponding defect data can be classified and analyzed using the original ODC [1], small companies typically employ the agile development process to deal with the dynamic, fluid, and volatile market environment and compressed development cycle, accompanied by various other limitations on data [4, 5]. The development environment for the semiconductor Optical Endpoint Detection (OED) software system in this study is a typical case in this latter category.

OED is an endpoint detection system that communicates to a process tool through a customer's chosen communication method. OED sends commands to a spectrometer on how to collect and transmit data. It then runs the appropriate proprietary algorithm in the customer's recipe. When the condition to stop the process is reached, it sends an appropriate command to the process tool to stop the process.

The overall product development and release cycles resemble that in agile development [4, 5]. The feature sets are controlled through code freezes and branches (major versions). Once a targeted major feature is implemented and necessary verification carried out, a branch would be created. All the software changes are controlled by a Software Change Control Board (SCCB) that determines changes to be made based on customer needs and company priorities. Upgrades and deployments are end-user controlled. This raises challenges on applying needed defect fixes unless compelling evidence is presented.

The data used for this study are the individual defects discovered and associated information extracted from a defect reporting database, covering the period from May 2001 to June 2014. The OED/agile data are organized by branches and months, with the branches denoted by the first digit of the release numbers. For example, branch 4.X contains releases 4.01, 4.1, 4.2 etc. Typically for OED and for agile development in general,

there are several incremental or rolling releases per year, as compared to a single release developed over several years for large software systems [1]. Each branch contains between 477K and 848K lines of source code (477-848 KSLOC).

The magnitude of defects reported in the early years of the system's life cycle caused concerns to stakeholders. Due to the negative impact to customers, management was particularly concerned about the relatively larger number of defects discovered in the field by customers. Consequently, ODC was adapted to analyze OED/agile defect data and multiple recommended changes were implemented to address these concerns, such as through enhanced unit tests and additional regression tests.

Historically, most testing activities had been left to testers, with very limited unit tests carried out by developers. This led to a prolonged cycle of defect-fixing and re-testing. In addition, unit tests had been proven to be vital in catching logic errors that were difficult to catch in the system test environment. Consequently, a policy change was implemented in OED to require developers to carry out significantly more unit tests before transitioning to other forms of testing performed by professional testers. This policy change would also offer more opportunities to collect early defect data to provide actionable feedback to the development process.

Therefore, under this particular product development environment with limited data availability, there is a strong need to analyze the available defect data to provide actionable feedback. However, the suitability and applicability of the original ODC are in doubt due to the significant differences in the development environments and data availability. Similarly, previous work in adapting ODC beyond traditional commercial software systems to NASA and Web-based systems [2, 3] cannot be used directly due to similar reasons. However, a similar approach along the line of the previous work is possible: namely, adapting the original ODC to the agile development environment to fulfill the need for actionable feedback for quantifiable quality improvement, and demonstrating its applicability and effectiveness through a case study on OED.

3. Solution Strategy: Methodology, Metrics, Models, and Hypotheses

Our overall solution strategy consists of three major stages: 1) development of a new defect classification scheme, aODC, or our adapted ODC for agile development, by adapting the original ODC to the needs of OED/agile development under its unique environmental constraints; 2) deployment of aODC to classify and analyze defect data from OED, and to provide actionable feedback; and 3) quantification of the impact of aODC on defects and product quality. This three-staged solution strategy, illustrated in Fig. 1, can be considered as an adaptation of the quality improvement paradigm [9] to achieve our goal of quantifiable quality improvement.

To develop aODC, our adapted ODC for OED/agile development, we first analyze the needs for defect classification and analysis under this specific environment, particularly under the constraint of limited data availability. This analysis can be guided by the Goals-Questions-Metrics (GQM) paradigm [9], where the needs and the environmental constraints can be aligned with the Goals and the Questions in GQM, while the specific defect attributes and attribute values can be aligned with the Metrics in GQM. On the other hand, the needed defect attributes and attribute values for this environment can be identified and organized through a systematic examination of the existing defect attributes and values in the original ODC [1], as illustrated in Fig. 1 by the pairing of ODC attributes to the limited data available from OED/agile. Once aODC is constructed, we could deploy it in OED, which we did for 5.X branch and thereafter. The analysis of defect data collected from OED according to aODC and the feedback provided to its development process form a feedback loop, also illustrated in Fig. 1.

To quantify the impact of aODC on defects and product quality, several defect and quality metrics need to be defined and assessed for OED releases before (4.X branch and earlier) and after (5.X branch and later) the

deployment of aODC. The defect metrics of interest to our environment and obtainable from available data include the following:

1. *Total defect count* by release and aggregated by year, which includes both defects reported by customers after product release (in-field) and defects discovered during product development (in-house).
2. *Number of in-field defects* reported by customers after product release. Arguably, in-field defects should be more of a concern to software development organizations, as is the case for OED development (see Section 2 above).
3. *Distribution of in-house defects*. It is desirable to have a front-loaded distribution, i.e., more defects discovered early in the agile development process by programmers and fewer defects discovered later in the process by testers. In general, earlier discovery of defects would minimize the chances for additional defect injections due to chain-effect. The defect detection and fixing effort would also be reduced due to the closer proximity between the defect sources and the expertise needed to identify and fix these defects, such as in the case with logic errors in OED described in Section 2.

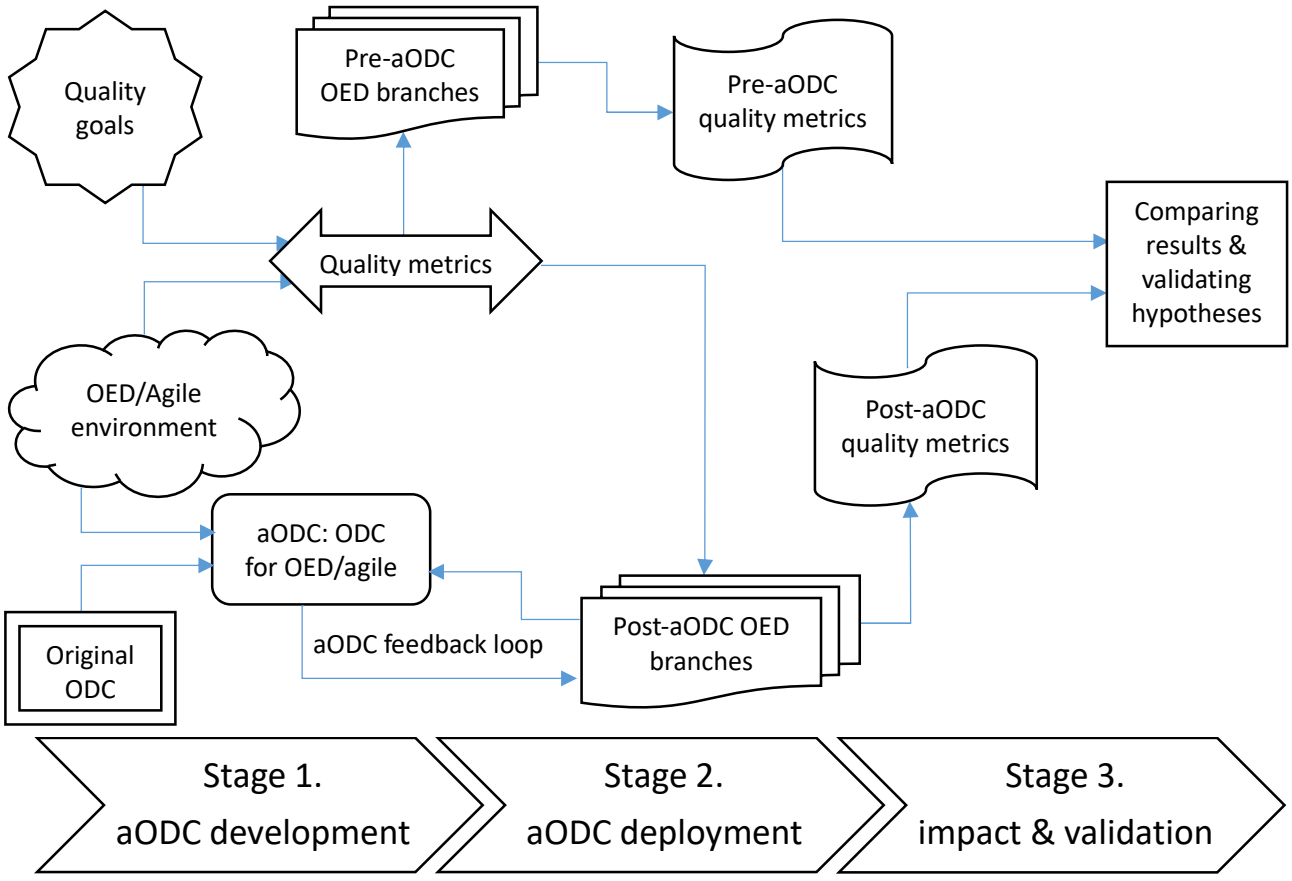

Fig. 1. Solution strategy and steps.

Our quality metric selected is product reliability, or the probability of failure-free operations of a software system for a given period of time or a given set of input under a specific environment [10, 11]. Reliability is a quality metric from the customer's perspective, characterizing the likelihood of problem-free operations desirable to its customers. Reliability can be assessed and predicted based on defect, timing, and input data using various reliability models, including time domain Software Reliability Growth Models (SRGMs) and Input Domain Reliability Models (IDRMs) [10, 11]. The specific SRGMs used in this study are Goel-Okumoto

(GO) [12] and Musa-Okumoto (MO) [13] models, with:

$$\text{GO SRGM} \qquad \mu(t) = a\left(1 - \mathrm{e}^{-bt}\right)$$

$$\text{MO SRGM} \qquad \mu(t) = \frac{1}{\Theta}\ln(\lambda_0 \Theta t + 1)$$

where $\mu(t)$ is the expected number of defects at time t; a, b, λ_0, and Θ are the model parameters.

For a given set of input or test runs, typically over a short period of time, Input Domain Reliability Models (IDRMs) [10, 11] can be used to assess system reliability, such as the Nelson model [14] used in this study:

$$\text{Nelson IDRM} \qquad R = \frac{S}{N} = \frac{N-F}{N} = 1 - \frac{F}{N}$$

where R is the estimated reliability or the success rate, S is the number of successful executions or test runs, N the total number of runs, and F the number of failed runs.

Reliability growth, or the improvement in reliability due to defect discovery and fixing as testing progresses, can be visualized by the cumulative failure arrival curve over time bending towards the upper-left corner and a flattened tail [15]. To quantify reliability growth, we use the purification level ρ [11], the ratio of failure rate reduction over a given test period, define by:

$$\rho = \frac{\lambda_0 - \lambda_T}{\lambda_0} = 1 - \frac{\lambda_T}{\lambda_0}$$

where λ_0 is the failure rate at the start of testing and λ_T is the failure rate at the end of testing, calculated from SRGMs fitted to defect data over time. A higher value of ρ indicates a more significant reliability growth.

Using the defect and quality metrics defined above, we can compare the impact of aODC in three steps, also illustrated in Fig. 1: 1) We can establish a pre-aODC baseline by determining the defect and quality metrics values based on the data from 4.X branch and earlier; 2) post-aODC assessment can be carried out using the same set of metrics on the data from 5.X branch and later, after aODC has been deployed therein; and 3) we compare the two sets of results and draw some conclusions about the impact of aODC. This quantitative comparison will be used to validate three main hypotheses regarding the desired impact of aODC:

1. *Defect impact hypothesis* (H_d): If in-process feedback is provided in the life cycle of a system through the use of aODC, then the overall defect count would decrease (H_{d1}). A related hypothesis is that in-field defect count reported by customers would decrease too (H_{d2}).
2. *Early defect discovery hypothesis* (H_e): Deployment of aODC would increase the share percentage of early defects discovered by developers and decrease the share percentage of late defects discovered by testers.
3. *Reliability impact hypothesis* (H_r): Deployment of aODC would lead to increased reliability (H_{r1}) and more significant reliability growth (H_{r2}).

These hypotheses will be tested by comparing the post-aODC defect and quality metrics against the pre-aODC baseline to validate our approach and its positive impact on quantifiable quality improvement.

4. aODC: Adapting ODC for OED/Agile

The original ODC groups defect attributes into two major categories, the opener and closer sections. The opener section refers to the information collected when a defect is first detected by a tester, an inspector, or other development personnel, and a defect record is created (or *opened*). The closer section refers to the information collected when a defect gets resolved (or *closed*) by a developer, with concurrence by its discoverer/opener or the manager in charge. The former includes several ODC attributes: activity, trigger,

severity, impact, discovered-by, time-of-discovery, etc. For the closer section, there are several attributes: target, defect-type, age, source, fix-type, fix-action, phase-injected, etc.

We have adapted the standard ODC attributes to the OED/agile system analysis needs under agile development in our adapted ODC, or aODC. Table 1 highlights the ODC attributes that are meaningful to the OED/agile system, categorized under opener and closer sections just like in the original ODC. Under the opener section, we have adapted the original ODC attributes {*Defect Removal Activity, Triggers, Severity,* and *Discovered-by*} as {*Activity, Trigger, Severity,* and *Discovered-by*} attributes respectively for the OED/agile system in aODC. Similarly in the closer section, we have adapted the original ODC attributes {*Defect Type*, *Source*, and *Target*} as {*Defect Type*, *Branch,* and *Component*} attributes respectively in aODC.

Table 1. Adapted ODC attributes in aODC for OED/Agile

Section	**Attribute**		**Attribute Value**
	Original	**OED/Agile**	**OED/Agile**
Opener	Defect Removal Activity	Activity	Unit Test System Test Customer Usage Other Activities
	Triggers	Trigger	Logic Hardware Backward Compatibility Other Triggers
	Severity	Severity	Production Stop Average Minor
	Discovered-by	Discovered-by	End User Developer Tester Other Personnel
Closer	Defect Type	Defect Type	Algorithm Communication Processing Other Functions
	Source	Branch	4.X branch 5.X branch 6.X branch Other branches
	Target	Component	Processor Computational Component Sensor Communicator Tool Communicator RF Communicator

The original *Defect Removal Activity* attribute covers defects found during activities such as design review, code inspection, and different types of testing during product development. The corresponding aODC attribute *Activity* only covers defects discovered during *Unit Test* and *System Test* activities applicable to OED/agile. Defects discovered outside of the development-cycle are typically tracked separately in the original ODC. However, in agile development, user participation in the development process and the incremental or rolling product releases would blur the line separating in-process and out-of-process defect discoveries. Therefore, we also use *Customer Usage* attribute value to track defects reported based on the customer's usage and *Other Activities* attribute value to track other defects found based on activities not accounted for by the above attribute values.

The original *Triggers* attribute relates a defect to the specific circumstance of it discovery, such as the test case or the usage scenario that led to the detection of this particular defect. Our corresponding aODC attribute

Trigger covers specific triggers from unit and system test only, including *Logic, Hardware, Backward Compatibility,* and *Other Triggers* attribute values to track defects detected while testing for logic problems in the source code, hardware problems, backward compatibility related problems, and other problems not falling into the above categories.

The original *Severity* attribute measures the negative impact of a defect. For the corresponding aODC attribute *Severity*, we use *Production Stop* and *Average* attribute values to track the major and average level of impactful defects respectively. The remainder of the defects are classified as *Minor* severity defects.

The original *Discovered-by* attribute records the personnel who discovered the defects. Since this attribute is applicable to the OED/agile system, we adopted the original attribute as is, while using aODC specific attribute values *End User, Developer, Tester,* and *Other Personnel* to track defects reported by these personnel groups respectively.

The original ODC *Defect Type* attribute in the closer section uses attribute values *Algorithm/Method, Interface/O-O Messages, Timing/Serialization*, etc., to indicate the specific type of internal problems and corresponding corrections applied to the defects being fixed. This defect attribute is applicable to the OED/agile system. Therefore it is used as is in aODC, but with updated attribute values *Algorithm, Communication, Processing,* and *Other Functions* to track defects related to proprietary algorithms, protocol and general communication, general data processing, and other minor functions not already categorized.

The original *Source* attribute indicates the defects origins in the context of development history, such as from in-house base code, external vendor code, reusable software components or services in the form of APIs (application program interfaces), or other code sources. This attribute can be mapped to the *Branches* in aODC for OED/agile, to indicate specific defect origins identified with specific product branches of OED and the corresponding code base. We use *Branch* attribute values *4.X branch, 5.X branch,* and *6.X branch,* to indicate the defects originated from the 4.X, 5.X, and 6.X released branches respectively. The discontinued branches 1.X, 2.X, and 3.X are tracked using *Other Branches* attribute value.

The original ODC *Target* attribute indicates the subpart of a product, such as a specific unit, module, component, or even a sub-system, where the specific defect (or product-internal fault) is fixed. For OED/agile, this product sub-part is identified by its *Component* attribute in aODC, with its attribute values *Processor, Computational Component, Sensor Communicator, Tool Communicator,* and *RF Communicator* to track defects associated with different sets of modules grouped together to form several executable or dynamic linked libraries for Processor, Computational Component, Sensor Communicator, Tool Communicator, and RF Communicator respectively.

5. Baseline Defect Analysis

We began with retroactive defect distribution analysis of the baseline for all the aODC attributes identified in Table 1. The analysis results for the *Defect Type* and *Discovered-by* attributes showed the most interesting patterns among all the attributes we examined.

The objective of using the *Defect Type* attribute is to examine defect distribution across the major functions, including *Algorithm*, *Communication*, and *Processing* from Table 1 and described in the previous section. The key questions for this analysis are: 1) which function is more defect prone, or which *Defect Type* is dominant, and 2) whether this result conforms to our expectations.

Fig. 2 highlights defects for the three major functions of the OED/agile system prior to aODC deployment. The processing related defects were the most reported. This was not surprising, as the amount of data processed was high, leading to a higher share of defects. The next most reported category was communication related defects. Initially, this was a surprise as more algorithm defects were expected than communication defects. On further analysis, we realized that the algorithms were initially developed on a separate platform,

and they had already gone through extensive validation activities. Thus, the fundamental principles of the algorithms were solid. This led us to conclude that the defects reported in this category were more likely related to the migration and integration process of the algorithms from their original platform to the new environment.

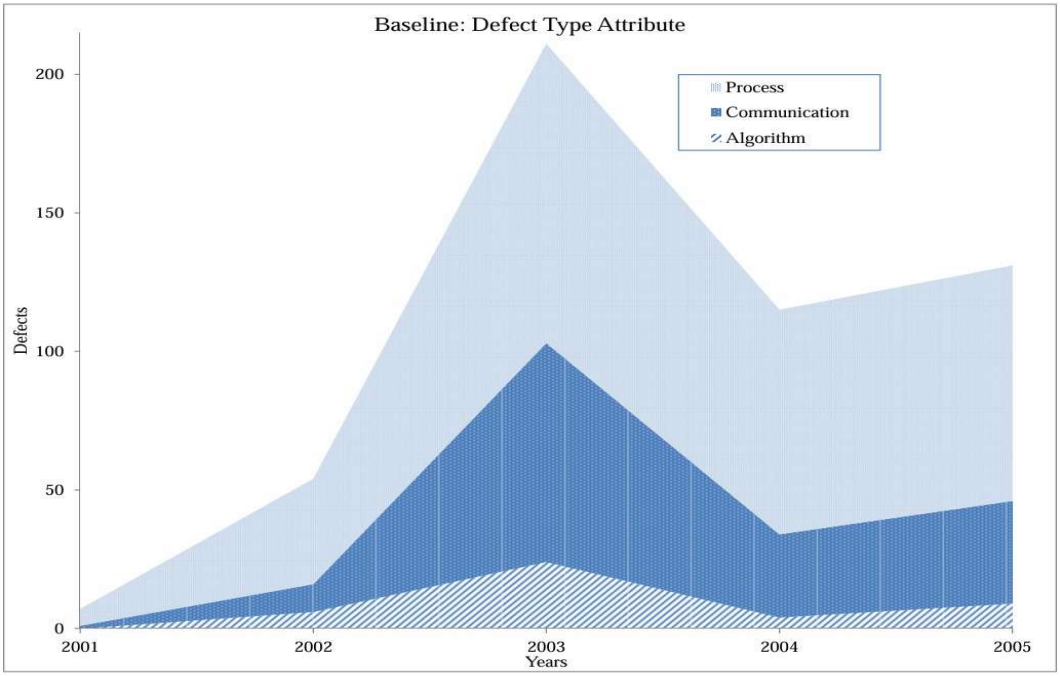

Fig. 2. Baseline defect type attribute.

Next, we examine *Discovered-by* attribute for the defect distribution between in-house and in-field defects. This analysis addresses one of the major concerns for management of limiting the number of defects remaining in OED which are later discovered by end-users. For in-house defects, *Discovered-by* attribute further groups the defects by the testers or developers who discovered them. The results are summarized in Table 2, excluding the partial years of 2001 and 2014 to keep the yearly data comparable.

Table 2. Number of Defects and Percentage Share

Year		In-House				In-Field		Total
		Developer		Tester		Customer		
Baseline (2002–2005)	2002	9	17%	32	62%	11	21%	52
	2003	46	22%	157	75%	7	3%	210
	2004	19	16%	86	73%	13	11%	118
	2005	51	40%	61	48%	15	12%	127
	Avg.	31.3	24.7%	84	66.3%	11.5	9.1%	126.8
Post-aODC (2006–2013)	2006	36	31%	68	58%	14	12%	118
	2007	27	25%	78	72%	3	3%	108
	2008	31	49%	18	29%	14	22%	63
	2009	13	35%	9	24%	15	41%	37
	2010	25	47%	17	32%	11	21%	53
	2011	15	39%	18	47%	5	13%	38
	2012	24	50%	12	25%	12	25%	48
	2013	17	57%	10	33%	3	10%	30
	Avg.	23.5	38.0%	28.8	46.5%	9.6	15.6%	61.9

The top portion of Table 2 shows the pre-aODC baseline total defect count and the number and percentage share of the defects discovered by developers, testers, and customers during the normal unit test, system test, and in-field use respectively. The tester's share of total defect discoveries was significantly higher than other

categories between 2002 and 2004, ranging from 62% to 75% of the total number of defects. The gap closed considerably in 2005, although the tester's percentage share still remained fairly high, at 48%. These results agree with a known fact that prior to the deployment of aODC, developers were not performing extensive unit tests before passing the product to the testers. This observation contributed to the policy change in OED to require developers to carry out significantly more unit tests before transitioning to system testing performed by professional testers.

The percentage share of defects reported by the customers was relatively low, ranging from 3% to 21%, and stayed relatively consistent over the years. These results alleviated management's concern of having small percentage share of defects discovered in-field by the customers.

6. Reducing Defects and Improving Early Discovery of Defects

Next, we examine the effect of aODC deployment by comparing the defect metrics results for post-aODC against the pre-aODC results.

The bottom portion of Table 2 highlights the post-aODC total defect count and the number and percentage share of the defects discovered by each personnel group. Comparing the total number of defects, we can see a general trend of decreasing defect count post-aODC and over time. The average total defect count per year is reduced from the original 126.8 pre-aODC to 61.9 post-aODC, a reduction of more than 50%, thus validating our hypothesis H_{d1}. In addition, the average number of in-field defects reported by customers is reduced from 11.5 to 9.6 after deployment of aODC, a 16% reduction, thus validating our hypothesis H_{d2}.

On the other hand, the customer's percentage share of the defects increased from an average of 9.1% pre-aODC to 15.6% post-aODC. However, it remained relatively low and steady over time, ranging from 3% to 25% in the post-aODC time period, with an outlier occurring in 2009 at 41%. The outlier was aligned with the last major recession when customers were heavily involved with research and development of new processes that led to more newly discovered defects. If that outlier is excluded, the customer percentage share of defects would be reduced to an average of 13.5%, a moderate increase over 9.1% pre-aODC. Overall, the total number of defects discovered by the customers were still much fewer compared to those discovered in-house, and followed a downward trend from pre-aODC to post-aODC (see H_{d2} discussion above).

Table 2 also shows the trend of the percentage share of the defects discovered by each personnel group. Generally speaking, the percentage share of the defects reported by the developers increased post-aODC, with the average percentage share increasing from 24.7% pre-aODC to 38% post-aODC. On the other hand, the tester's average percentage share of total defects decreased from 66.3% pre-aODC to 46.5% post-aODC. This supports the hypothesis H_e: Deployment of aODC, accompanied by the shift to more unit tests in OED/agile, the share percentage of early defects discovered by developers would increase.

7. Improvement in Reliability and Reliability Growth

For reliability comparison of pre-aODC baseline and post-aODC branches, we took the 4.X and 5.X data and applied the Nelson model to the last branch release data. The modeling results give us Nelson reliability values of 0.884 and 0.914 for 4.X and 5.X respectively. These results validate our reliability impact hypothesis H_{r1}, i.e., deployment of aODC would lead to increased reliability.

For reliability growth comparison between pre-aODC and post-aODC branches, we used the defect trend data within the 24 months for 4.X and 5.X branches respectively. Fig. 3 shows the cumulative reported defects over the 2230 and 2198 test runs over the respective 24 month periods for the 4.X and 5.X branches. Visually examining the defect trend, we can see more reliability growth in 5.X than in 4.X, as visualized through a more pronounced bending towards the upper-left corner and a significantly flattened defect arrival curve for 5X at the tail-end in comparison with the curve for 4.X.

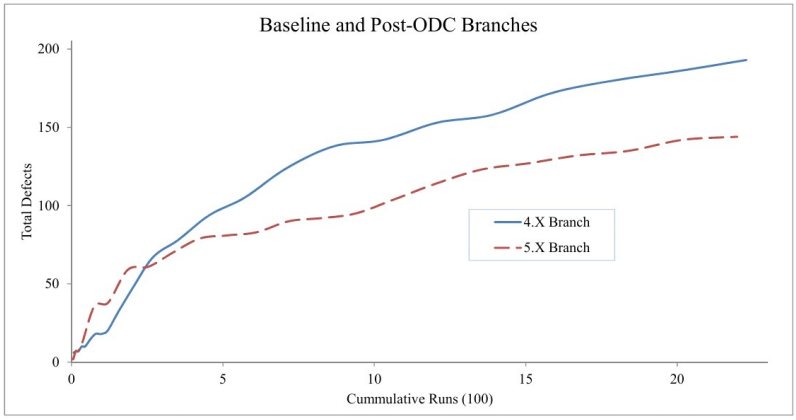

Fig. 3. Reliability growth comparison: BASELINE 4.X vs Post-aODC 5.X

Figs. 4 and 5 show the 4.X and 5.X branches defect data fitted with the GO and MO models respectively. Purification level ρ calculated from each fitted model can then be compared across the branches to quantitatively compare the reliability growth. We calculated the purification level ρ as 0.916 and 0.902 for 4.X by the fitted GO and MO models respectively. Similarly, ρ is 0.991 and 0.994 for 5.X, calculated by the fitted GO and MO models respectively. The overall purification level ρ is significantly higher in the 5.X branch (> 0.99) than in the 4.X branch (around 0.91). These results validate our reliability growth impact hypothesis H_{r2}: If in-process feedback is provided earlier in the life cycle of a system, then the purification level ρ would increase, indicating significantly more reliability growth.

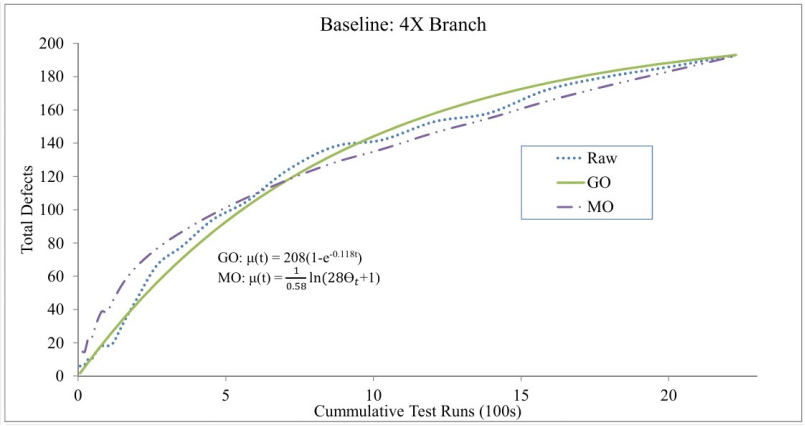

Fig. 4. Fitted GO and MO SRGMs to baseline 4.X data.

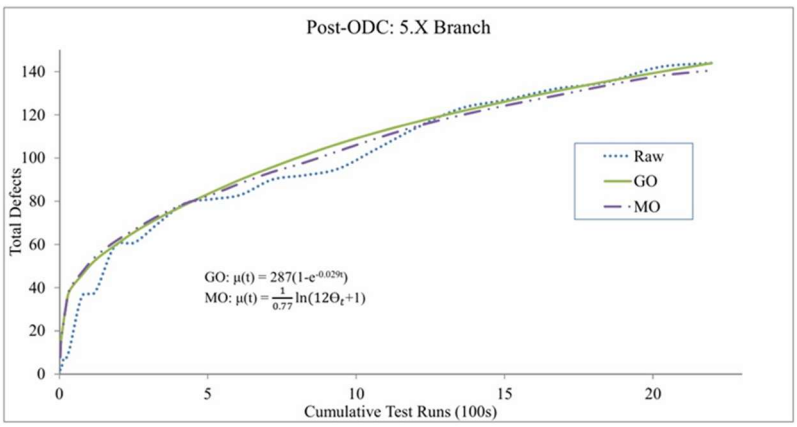

Fig. 5. Fitted GO and MO SRGMs to post-aODC 5.X data.

8. Conclusions and Perspectives

This study was instrumental in providing valuable early feedback about software quality and drive quantifiable quality improvement for a semiconductor software developed in a small company under the agile development process. Similar to other studies on adapting ODC to specific application environments [2, 3], we adapted the original ODC attributes and attribute values [1] to meet this study's need under agile development and its specific environment constraints, such as limited data availability, reduced staffing level and process scope, and compressed schedule for the dynamic market [4, 5].

To assess the impact of this in-process feedback based on aODC, our adapted ODC for agile development, we defined several defect and quality metrics, including total defect count, in-field defects discovered by customers, share of early defects discovered by developers vs late defects discovered by testers, reliability assessed at various points in time, and overall reliability growth. We first quantified the baseline defect and quality metrics before the deployment of our aODC. After deployment and active usage of aODC in the OED product development process for the later product branches, we assessed the results using the same defect and quality metrics. We compared these pre-aODC and post-aODC results and demonstrated that aODC offered valuable early in-process feedback that led to quantifiable quality improvement. In particular, all our hypotheses regarding the impact of aODC have been validated, namely,

1. a more than 50% reduction in total defects (Hypothesis H_{d1}) from an average of 126.8 per year down to 61.9, and a 16% reduction of defects found by customers (Hypothesis H_{d2}) from an average of 11.5 per year down to 9.6;
2. a significantly higher percentage share of defects discovered in the early part of the process by the developers, at 38%, up from 24.7% in the baseline, and a significantly lower percentage share of defects discovered later by the testers, at 46.5%, down from 66.3% in the baseline (Hypothesis H_e);
3. a higher reliability (Hypothesis H_{r1}), with a success rate of 0.914 post-aODC compared to 0.884 in the baseline, and a more significant reliability growth (Hypothesis H_{r2}), quantified by the purification level of 0.99 post-aODC as compared to 0.91 in the baseline.

This study has shown practical ability of adapting the original ODC and using it to provide valuable in-process feedback, to reduce defects, and to improve quality and reliability for a semiconductor software under the agile development environment. In general, adapting existing models and techniques to work effectively in specific application environments and generalizing them to work for heterogeneous systems will help improve quality, productivity, and customer satisfaction for a wider variety of systems, such as command-control-communication systems, embedded systems, smart devices, communication networks, critical infrastructures, clouds and service computing systems, etc. Most of these systems are developed using the latest development technologies and are more likely to adopt the agile development process instead of the traditional waterfall process [4, 5], making our aODC for the agile development environment more suitable and more easily adaptable than the original ODC to these systems for quantifiable quality improvement.

As a follow-up to this study, we would like to explore the possibility of a tighter and smoother integration of our aODC into the agile development process to provide timely feedback and suggestions, as an integral part of the process instead of as an add-on to the process. Instead of relying on a dedicated analyst to classify defect data according to aODC, software developers, testers, and inspectors can be pre-trained to classify defects on the spot so that aODC data would become immediately available. Previous work on automatically classifying defect based on defect descriptions in natural language [16] can potentially be adapted and integrated into the agile development process to reduce the training and operational cost of aODC deployment. In addition, automated ODC data analysis along the line of [17] has the potential to reduce the time delay in providing timely feedback and to reduce the analysis cost performed by a dedicated analyst.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Both authors conducted the research and wrote the paper. Eric Abuta implemented aODC in OED and collected detailed data. All authors had approved the final version.

Funding

This research was supported in part by the NSF Grant #1126747, Raytheon and NSF Net-Centric I/UCRC.

Acknowledgment

This paper contains material expanded from unpublished portions of the doctoral thesis by the first author [8]. We thank the anonymous reviewer for his/her insightful and constructive comments.

References

[1] Chillarege, R., Bhandari, I., Chaar, J., Halliday, M., Moebus, D., Ray, B., & Wong, M. -Y. (1992). Orthogonal defect classification—A concept for in-process measurements. *IEEE Transactions on Software Engineering, 18(11),* 943–956.

[2] Lutz, T. R., & Mikulski, C. (2003) Better analysis of defect data at NASA, *Proceedings of the 15th International Conference on Software Engineering and Knowledge (SEKE '03)* (pp. 607–611).

[3] Ma, L., & Tian, J. (2007). Web error classification and analysis for reliability improvement. *Journal of Systems and Software, 80(6)*, 795–804.

[4] Alahyari, H., Svensson, R. B., & Gorschek, T. (2017). A study of value in agile software development organizations. *Journal of Systems and Software*, *125,* 271–288.

[5] Gren, L., Torkar, R., & Feldt, R. (2017). Group development and group maturity when building agile teams: A qualitative and quantitative investigation at eight large companies. *Journal of Systems and Software, 124*, 104–119.

[6] Abuta, E., & Tian, J. (2018). Reliability over consecutive releases of a semiconductor optical endpoint detection software system developed in a small company. *Journal of Systems and Software, 137,* 355–365.

[7] Abuta, E., & Tian, J. (2018). Defect classification and analysis in a small company, *Proceedings of the 31st International Conference on Computer Applications in Industry and Engineering (CAINE 2018)* (pp. 175–182).

[8] Abuta, E. (2019). *Long Term Software Quality and Reliability Assurance in a Small Company*. DE Praxis, Southern Methodist University, Dallas, Texas, USA.

[9] Basili, V. R., & Rombach, H. D. (1988). The TAME project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering, 14(6)*, pp. 758–773.

[10] Lyu, M. R. (1995). *Handbook of Software Reliability Engineering*. McGraw-Hill.

[11] Tian, J. (1998). Reliability measurement, analysis, and improvement for large software systems. *Advances in Computers, 46,* 159–235.

[12] Goel, A., & Okumoto, K. (1979). Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability, 28(3)*, 206–211.

[13] Musa, J. D., & Okumoto, K. (1984). A logarithmic Poisson execution time model for software reliability measurement, *Proceedings of the 7th Int. Conf. on Software Engineering* (pp. 230–238).

[14] Nelson, E. (1978). Estimating software reliability from test data. *Microeletronics and Reliability, 17(1),* 67–73.

[15] Tian, J. (2023). Accelerated defect discovery and reliability improvement through risk-prioritized testing for Web applications. *Journal of Software, 18(3)*, 159–171.

[16] Huang, L., Ng, V., Persing, I., Chen, M., Li, Z., Geng, R., & Tian, J. (2015). Automated generation of orthogonal defect classifications. *Automated Software Engineering, 12(1)*, 3–46.

[17] Menzies, T., Lutz, R., & Mikulski, C. (2003). Better analysis of defect data at NASA, *Proceedings of the 15th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'03)* (pp. 607–611).