Empir Software Eng (2017) 22:1634–1683
DOI 10.1007/s10664-016-9440-x

EXPERIENCE REPORT

Assessing the quality of industrial avionics software: an extensive empirical evaluation

Ji Wu[1] · **Shaukat Ali**[2] · **Tao Yue**[2,3] · **Jie Tian**[1] · **Chao Liu**[1]

Published online: 1 July 2016

Abstract A real-time operating system for avionics (RTOS4A) provides an operating environment for avionics application software. Since an RTOS4A has safety-critical applications, demonstrating a satisfactory level of its quality to its stakeholders is very important. By assessing the variation in quality across consecutive releases of an industrial RTOS4A based on test data collected over 17 months, we aim to provide a set of guidelines to 1) improve the test effectiveness and thus the quality of subsequent RTOS4A releases and 2) similarly assess the quality of other systems from test data. We carefully defined a set of research questions, for which we defined a number of variables (based on available test data), including *release and measures of test effort*, *test effectiveness*, *complexity*, *test efficiency*, *test strength*, and *failure density*. With these variables, to assess the quality in terms of number of failures found in tests,

Communicated By: Brian Robinson

This paper is the extended version of the conference paper published in the 24th IEEE International Symposium on Software Reliability Engineering (ISSRE 2013).

✉ Ji Wu
wuji@buaa.edu.cn

Shaukat Ali
shaukat@simula.no

Tao Yue
tao@simula.no

Jie Tian
tianjie@sei.buaa.edu.cn

Chao Liu
liuchao@buaa.edu.cn

[1] School of Computer Science and Engineering, Beihang University, Beijing, China

[2] Simula Research Laboratory, Oslo, Norway

[3] University of Oslo, Oslo, Norway

we applied a combination of analyses, including trend analysis using two-dimensional graphs, correlation analysis using Spearman's test, and difference analysis using the Wilcoxon rank test. Key results include the following: 1) The number of failures and failure density decreased in the latest releases and the test coverage was either high or did not decrease with each release; 2) increased test effort was spent on modules of greater complexity and the number of failures was not high in these modules; and 3) the test coverage for modules without failures was not lower than the test coverage for modules with failures uncovered in all the releases. The overall assessment, based on the evidences, suggests that the quality of the latest RTOS4A release has improved. We conclude that the quality of the RTOS4A studied was improved in the latest release. In addition, our industrial partner found our guidelines useful and we believe that these guidelines can be used to assess the quality of other applications in the future.

Keywords Software Quality assessment · Real time operating system · Avionics software · Industrial case study

1 Introduction

A real-time operating system for avionics (RTOS4A) has several nonfunctional properties (e.g., safety, reliability and real-time properties) that require high confidence in its quality. One way of building such confidence is to incrementally collect evidence by statistically analyzing test data generated from different types of testing (e.g., functional and nonfunctional testing) at various levels (e.g., unit, integration). The results of such analyses can demonstrate a satisfactory level of the quality of the RTOS4A to its relevant stakeholders, including application developers and system integrators.

To provide evidence of the RTOS4A quality, we report the quality assessment of an industrial RTOS4A based on test data. The development of the industrial RTOS4A conforms to the process specified in the DO-178B standard (RTCA/DO-178B (1992)) to ensure the safety of avionics software in practice. The standard requires the following three levels of testing: 1) requirement-based low-level testing (RLLT), focusing on testing software components against their low-level requirements that capture the specification of functions according to DO-178B, 2) requirement-based software integration testing (RSIT) focusing on testing interactions among software components, and 3) requirement-based hardware/software integration testing (RHSIT) focusing on testing the deployment of software on a targeted platform(s). Our industrial partner is a local producer of RTOS4A products in China and provides RTOS4A-associated services in the avionics domain. The industrial partner has a Capability Maturity Model level 3 qualification, uses the DO-178B standard and other relative software engineering standards (e.g. ISO/IEC 12207 (1995)) to guide its RTOS4A development activities, and conducts all three types of testing. Our industrial partner does not wish to disclose its name since it believes that its competitors could misinterpret the quality of its RTOS4A based on the analyses reported in this paper. For the three types of testing, we obtained 17 months of test data for eight consecutive RTOS4A releases.

Though conducting reliability prediction based testing data has been very popular, the validation of the prediction result is usually difficult without field data. In this paper, our overall goal is to provide experience in assessing variation in the quality of consecutive RTOS4A releases based on test data rather than failures collected from actual RTOS4A field operation. We refer to the IEEE standard 829–2008 (IEEE Std 829-1998 (1998)) for the definition of

software quality, i.e., the degree to which the RTOS4A meets its specified requirements. To achieve this overall goal, we access multiple dimensions of RTOS4A test data, including the effectiveness of testing (e.g., number of failures), the characteristics of test cases (e.g., the number of lines of code of a test case), and the characteristics of the RTOS4A implementation (e.g., the number of lines of code of the implementation). We define two categories of measures to collect and analyze test data: atomic measures and derived measures. Atomic measures involve readily available test data, such as the number of failures found in a release and the coverage of the lines of code of a release achieved in testing. Derived measures are defined based on atomic measures and allow one to study multiple atomic measures simultaneously, such as the number of failures found per test case and coverage achieved per test case.

We define seven research questions to achieve the overall goal. To answer these research questions, we analyze the test data with three types of analyses: 1) trend analysis using two-dimensional (2D) scatter plots to determine the patterns of data collected with various measures along the releases, 2) correlation analysis to determine the correlations between the measures within a release using a nonparametric Spearman's correlation test (Hair et al. 2006), and 3) difference analysis using the Wilcoxon signed-rank test (Rice 2006) to determine the differences in various measures between a release for modules with failures observed and modules without failures observed. Based on the results of these analyses, we provide the recommendations only based on what we observed from the collected data. The recommendations proposed can be used to: 1) determine the variation in the quality of an RTOS4A release and 2) provide quality indicators that can be used to improve the quality of releases through testing. The recommended guidelines provide insights on the quality of RTOS4A releases to our industrial partner, who found these guidelines very useful. We expect that these guidelines will be useful for other practitioners in the avionics domain to assess the quality of their systems based on test data collected during the development and testing phases.

This paper is an extension of a conference paper (Wu et al. 2013) and the key differences from the conference version include the following: 1) we focused on quality instead of reliability assessment in this paper; 2) We added a set of derived measures for more in-depth analyses to answer two new research questions; 3) we further increased the confidence of the analyses, performing Wilcoxon signed-rank tests for the various measures; 4) we added additional details on how we collected the data for the industrial case study; and 5) in addition to improving existing guidelines from the previous version, we provide guidelines on collecting and validating data and suggestions on how to further improve software quality.

The rest of the paper is organized as follows: Section 2 introduces our industrial case study of an RTOS4A, followed by the planning and design of the case study (Section 3). We present analyses and results in Section 4, guidelines and discussion in Section 5, and threats to validity in Section 6. Our case study is compared with case studies from the literature in Section 7. Finally, we conclude our paper in Section 8.

2 Description of the case study

Our case study system is an industrial RTOS4A with high quality requirements, in particular the safety and reliability, raised by its users, which are companies developing avionics application systems. An RTOS4A provides services and application programming interfaces for such applications as task management, time management, health management, communication and synchronization, error handling, input and output, memory management, and task

scheduling. Due to confidentiality issues, we cannot provide a detailed description of the data regarding the requirements, design, and exact size of the system. However, the development and testing process of the RTOS4A complies with DO-178B.

Our industrial partner has a development group responsible for developing an RTOS4A and releasing it together with its requirement specifications to an independent testing group within the same organization. Each RTOS4A release consists of a set of modules (e.g., 51 modules in the first release), each composed of several functions. All the functions of the system are mapped to specific low-level requirements. The independent testing group of our industrial partner spent about 17 months on the three types of testing (i.e., RLLT, RSIT, and RHSIT) for the system. According to our industrial partner, the test data for RLLT and RSIT are not separated when the testing was performed; therefore, we treat these two types of testing as a whole in our analyses. During the testing phase, we were able to collect test data from the three types of tests conducted on eight consecutive releases, including three types of coverage, namely, statement, branch, and modified condition/decision coverage (MCDC). Note that coverage analysis is considered mandatory according to DO-178B, which clearly requires that the three types of coverage should all be 100 % for safety-critical software, which is the case in our case study. If some code blocks cannot be covered by RLLT, RSIT, and RHSIT, additional verification should be conducted to achieve 100 % coverage according to the description in Section 6.4, Software Testing Process, of DO-178B. In the context of our industrial case study, two additional verification methods (i.e., code inspection and fault simulation) were used to inspect or execute uncovered statements, branches, and conditions/decisions governing branches by RLLT, RSIT, and RHSIT, thus achieving the 100 % coverage objective. According to our industry partner, the code inspection method is conducted manually to check uncovered program elements for possible hidden defects in a static analysis approach, while fault simulation is conducted by injecting faults at runtime using the necessary tools. Due to practical challenges during data collection from the additional verification steps, we were only able to obtain data for RLLT, RSIT, and RHSIT. Therefore, as shown in the figures and tables in the rest of the paper, the three types of coverage from RLLT, RSIT, and RHSIT do no satisfy the 100 % coverage objective as required by DO-178B. However, the final coverage achieved was 100 % after applying the two verification methods as presented in the test report of our industrial partner.

3 Case study design

In this section, we present the design of our case study. In Section 3.1, we define the overall goal and objectives of this case study. The research questions are presented in Section 3.2. In Section 3.3, we present the measures used to assess the quality of the RTOS4A, based on the collected test data, followed by a clarification of the scope of the study in Section 3.4. In Section 3.5, we justify the analyses and statistical tests applied in the case study. Finally, we present our data collection process in Section 3.6.

3.1 Overall goal and objectives

Our overall goal is to provide a set of recommendations for assessing the quality of consecutive RTOS4A releases based on collected test data. Based on the overall goal, we would like to achieve two objectives:

O1: Assess variation in quality based on the effectiveness of testing.
O2: Assess variation in quality based on characteristics of test and RTOS4A implementation.

3.2 Research questions

According to the two objectives defined in Section 3.1, we define seven research questions that are answered using the three assessment activities A1 to A3, shown in Table 1. For O1, we performed A1 to answer RQ1 and RQ2, using test effectiveness (i.e., the number of failures and test coverage) to assess variation in quality. For O2, we have two assessment activities, A2 and A3, as shown in Table 1, where A2 focuses on assessing the variation in quality based on test effort (e.g., the number of test cases) and RTOS4A complexity (e.g., the number of lines of code). Note that the data for these measures are readily available. Assessment activity A3 focuses on studying the variation in quality based on the measures derived from the atomic measures used in A2, such as test efficiency (i.e., amount of test effort required to find a failure), test strength (i.e., amount of test effort spent per line of code), and failure density (i.e., number of failures found per line of code).

We define the following seven research questions:

- **RQ1:** How does the quality of releases vary with testing effectiveness?
- **RQ2:** How does the quality of releases vary with the correlation of multiple effectiveness dimensions?
- **RQ3:** How does the quality of releases vary with test effort?
- **RQ4:** How does the quality of releases vary with RTOS4A complexity?
- **RQ5:** How does the quality of releases vary with the correlation between RTOS4A complexity and test effort?
- **RQ6:** How does the quality of releases vary with testing efficiency?
- **RQ7:** How does the quality of releases vary with the correlation of multiple efficiency dimensions?

3.3 Quality assessment measures

To conduct the assessment activities, we define a set of measures to answer the research questions. In Section 3.1.1, we introduce the conceptual model to classify various measures and show how these measures are related to each other. Section 3.3.2 presents the formulas to compute the values for the measures based on the data collected.

Table 1 Design of the case study

Objective	Research question	Assessment activity
O1	*RQ1, RQ2*	A1: Assess whether quality increases or decreases with the change of test effectiveness
O2	*RQ3, RQ4, RQ5*	A2: Assess how test effort and complexity affect quality
	RQ6, RQ7	A3: Assess how test efficiency, test strength, and failure density affect quality

3.3.1 Conceptual model of measures

Figure 1 shows a conceptual model of the measures as a UML class diagram. There are two types of measures: *atomic* and *derived*. For an *atomic* measure, data are readily available, such as the number of failures found and complexity of the RTOS4A. On the other hand, we define a *derived* measure based on two *atomic* measures.

Based on the available data, we define four groups of *atomic* measures: *release*, *TEFM* (test effectiveness measure), *TEM* (test effort measure), and *CM* (complexity measure). Three groups of *derived* measures are *FD* (failure density), *TS* (test strength), and *TFM* (test efficiency measure). For example, *FD* is derived from two atomic measures: *NoFai* (number of failures) and *LoC* (lines of code). Table 2 summarizes the abbreviation, group name, and the names of the measures in the group for each group of measures.

a) *Atomic Measures*

In this section, we briefly describe each atomic measure.

i. *Release*

This measure represents an integer number uniquely identifying each RTOS4A release. We define this measure as a nominal measure for categorizing other measures (Fenton and Pfleeger 1996). In our case study, we have eight releases and this measure has a nominal value of R1 to R8.

ii. *Test Effectiveness Measure (TEFM)*

The measure *TEFM* involves two types of atomic measures: the number of failures *(NoFai)* and *Test Coverage,* as shown in Table 2 and Fig. 1. The measure *NoFai* is the number of failures observed for functions in a release, while *Test Coverage* measures the code coverage achieved by the execution of a test suite and is classified into three types: statement coverage (*Stmt_Cvg*), branch coverage (*Branch_Cvg*), and modified condition/decision coverage (*MCDC_Cvg*). Several empirical studies have shown that a test suite with higher coverage leads to the detection of more failures (Hutchings et al. 1994; Malaiya et al. 1994, 2002; Fenton and Ohlsson 2000).

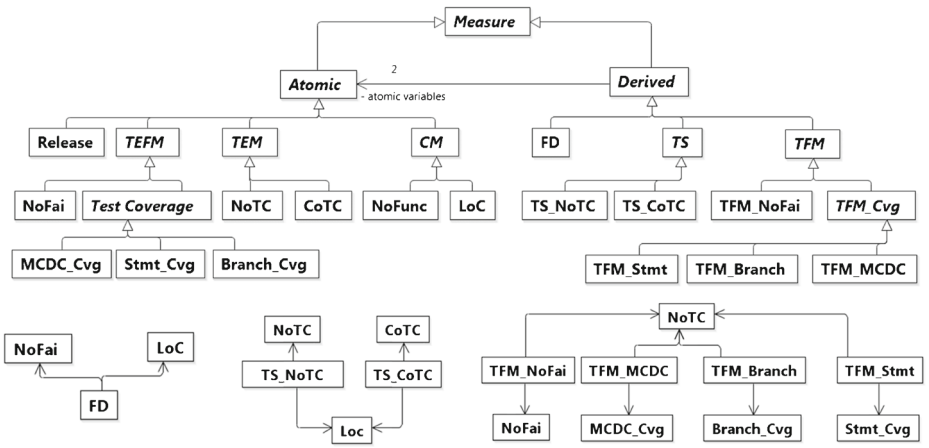

Fig. 1 Conceptual model of measures in a UML class diagram

Table 2 Measure design

Abbreviation	Group of measures	Name(s) of measure(s)
Release	Release of the RTOS4A	None
TEFM	Test effectiveness	Number of failures (*NoFai*), Test Coverage
TEM	Test effort	Number of test cases (*NoTC*), complexity of test cases (*CoTC*)
CM	Complexity of the RTOS4A	Number of function (*NoFunc*), lines of code (*LoC*)
TFM	Test efficiency	Failure-oriented efficiency (*TFM_NoFai*), coverage-oriented efficiency (*TFM_Cvg*)
TS	Test strength	Test effort per *LoC* including an *NoTC*-based sub-measure (*TS_NoTC*) and a *CoTC*-based sub-measure (*TS_CoTC*)
FD	Failure density	*NoFai* found per *LoC*
Test Coverage	Structural coverage achieved	Statement coverage (*Stmt_Cvg*), branch coverage (*Branch_Cvg*), modified condition/decision coverage (*MCDC_Cvg*)
TFM_Cvg	Coverage-oriented efficiency	*Stmt_Cvg*-, *Branch_Cvg*-, or *MCDC_Cvg*-based sub-measure (*TFM_Stmt*, *Branch_Cvg,* or *MCDC_Cvg*)

iii. *Test Effort Measure (TEM)*

The measure *TEM* measures test effort in terms of the number of test cases, their complexity, and time spent executing them. Based on the availability of data in our case study, we selected the number of test cases (*NoTC*) and the complexity of test cases (*CoTC*) to measure test effort. The measure *CoTC* is defined as the number of lines of code (*LoC*) in a test case, once again commonly used to measure code complexity (Fenton and Neil 1999).

iv. *Complexity Measure (CM)*

The complexity of the RTOS4A is measured based on the number of functions (*NoFunc*) and the number of lines of code (*LoC*) of a release, given data availability.

b) *Derived Measures*

There are three derived measures, as shown in Fig. 1: the test efficiency measure (*TFM*), test strength measure (*TS*), and failure density (*FD*).

i. *Test Efficiency Measure (TFM)*

The measure *TFM* is defined by dividing *TEFM* by *NoTC*. Based on the definition of *TEFM*, we define two types of test efficiency measure: failure-oriented *TFM_NoFai* and coverage-oriented *TFM_Cvg*. In addition, we have three types of coverage and we therefore define three additional measures: *TFM_Stmt*, *TFM_Branch,* and *TFM_MCDC,* corresponding to *Stmt_Cvg*, *Branch_Cvg*, and *MCDC_Cvg*, respectively.

ii. *Test Strength Measure (TS)*

The *TS* measure determines the effect of a system's complexity (in terms of *LoC*) on test effort. Since we have *NoTC* and *CoTC* for measuring test effort, we define two measures for *TS*: *NoTC/LoC (TS_NoTC)* and *CoTC/LoC (TS_CoTC).*

iii. *Failure Density Measure (FD)*

With *FD*, we study the effect of *NoFai* and the complexity of a release (i.e., *LoC*) together, via *NoFai/LoC*. Note that *FD* differs from the commonly used failure density function used in the context of reliability models, which is used to derive failure distributions (Musa et al. 1987) by computing the probability of failures within a given time duration.

3.3.2 Computation of measures

Table 3 presents the formulas for calculating the measures, where *x*/*y*/*z* refers to the function/module/release measures (e.g., *LoC*), *k* refers to a function in a module, *j* refers to a module in the *i*th release, m_i refers to the number of modules in the *i*th release, and n_j refers to the number of functions in the *j*th module. Therefore, x_{ijk} stands for a measure for function *k* in module *j* in the *i*th release, y_{ij} stands for a measure for module *j* in the *i*th release, and z_i represents a measure for the *i*th release.

3.4 Scope of the study

In the context of our case study, we provide indications for assessing variations in quality based on test data for consecutive releases of an RTOS4A based only on the measures shown in Fig. 1 and Table 3. The measures are applied at three levels: the function, module, and release levels. Since the analyses at the release and module levels are very comprehensive, as the first step, we report only the release and module levels' analyses. The *atomic* measures for functions (e.g., the *LoC* of a function) were measured directly based on the raw data, whereas the measures for modules and releases were derived based on the measures for functions and modules, respectively.

3.5 Justification of analyses and statistical tests

Incorporating testing metrics (e.g., the number of test cases, test coverage) to conduct software quality assessment is becoming a common practice (Fujii et al. 2011). Many factors could potentially affect the variations of quality and some of them might be not covered in the data collected from tests. It is also possible that some observation in data cannot be validated due to noise in the data. Therefore, quality assessment without checking the collected data by using empirical analyses (Wohlin et al. 2000) will be highly probable to lead to invalid results. To overcome this, we precisely define measures to capture various factors covered in the collected data, based on which we systematically conduct mathematical analyses, i.e. trend analysis, correlation analysis and difference analysis to answer the defined research questions.

Table 3 Formulas for computing measures

Measure	Release(z)	Module(y)	Func(x)
NoFai, NoTC, CoTC, LoC	$z_i = \sum_{j=1}^{m_i} y_{ij}$	$y_{ij} = \sum_{k=1}^{n_j} x_{ijk}$	x_{ijk}
Stmt_Cvg, Branch_Cvg, MCDC_Cvg	$z_i = \sum_{j=1}^{m_i} y_{ij} \,/ m_i$	$y_{ij} = \sum_{k=1}^{n_j} x_{ijk} \,/ n_j$	x_{ijk}
NoFunc	$z_i = \sum_{j=1}^{m_i} y_{ij}$	$y_{ij} = \sum_{k=1}^{n_j} x_{ijk}$	$x_{ijk} = 1$
TS_NoTC	$z_i = \sum_{j=1}^{m_i} y_{ij} \,/ m_i$	$y_{ij} = \sum_{k=1}^{n_j} x_{ijk} \,/ n_j$	$x_{ijk} = NoTC_{ijk}/LoC_{ijk}$
TS_CoTC	$z_i = \sum_{j=1}^{m_i} y_{ij} \,/ m_i$	$y_{ij} = \sum_{k=1}^{n_j} x_{ijk} \,/ n_j$	$x_{ijk} = CoTC_{ijk}/LoC_{ijk}$
TFM_NoFai	$z_i = \sum_{j=1}^{m_i} y_{ij} \,/ m_i$	$y_{ij} = \sum_{k=1}^{n_j} x_{ijk} \,/ n_j$	$x_{ijk} = NoFai_{ijk}/NoTC_{ijk}$
TFM_Stmt	$z_i = \sum_{j=1}^{m_i} y_{ij} \,/ m_i$	$y_{ij} = \sum_{k=1}^{n_j} x_{ijk} \,/ n_j$	$x_{ijk} = Stmt\_Cvg_{ijk}/NoTC_{ijk}$
TFM_Branch	$z_i = \sum_{j=1}^{m_i} y_{ij} \,/ m_i$	$y_{ij} = \sum_{k=1}^{n_j} x_{ijk} \,/ n_j$	$x_{ijk} = Branch\_Cvg_{ijk}/NoTC_{ijk}$
TFM_MCDC	$z_i = \sum_{j=1}^{m_i} y_{ij} \,/ m_i$	$y_{ij} = \sum_{k=1}^{n_j} x_{ijk} \,/ n_j$	$x_{ijk} = MCDC\_Cvg_{ijk}/NoTC_{ijk}$
FD	$z_i = \sum_{j=1}^{m_i} y_{ij} \,/ m_i$	$y_{ij} = \sum_{k=1}^{n_j} x_{ijk} \,/ n_j$	$x_{ijk} = NoFai_{ijk}/LoC_{ijk}$

- Trend analysis. We determine the patterns of test data collected via measures across the eight releases by plotting 2D scatter plots. In a 2D plot, the *x*-axis represents *Release*, while the *y*-axis represents different measures (excluding *Release*), such as the number of failures and coverage. Such analysis provides an indication of quality variation across the releases.
- Correlation analysis. Correlation analysis studies the correlation (positive/negative) between two variables (e.g., x and y) and, moreover, whether such a correlation is statistically significant. A positive/negative correlation between x and y means that increasing the value of x correlates with an increasing/decreasing value of y. We use the nonparametric Spearman's test (Hair et al. 2006) and report the correlation coefficient (ρ) and the significance (p-value). A value of $\rho = 0$ indicates no correlation between x and y, $\rho > 0$ indicates a positive correlation, and $\rho < 0$ indicates a negative correlation. We choose the significance level of 0.05 to denote that a p-value lower than 0.05 suggests a statistically significant correlation.
- Difference analysis. Typically, in the statistics analysis literature, difference analysis studies the differences in the distributions of a single measure between two distinct groups of modules (Rice 2006). In our case study, some modules have no failure (NF_Modules) in any of the releases (Section 4.1) and others do (F_Modules). We compare their differences based on various measures. We apply the Wilcoxon signed-rank test, a nonparametric test commonly used to compare the differences of two samples that 1) are unbalanced, 2) have different standard deviations, and 3) do not have a normal distribution (Rice 2006). We choose the significance level of 0.05 and, therefore, if a p-value is lower than 0.05, the difference is significant. We also report the z-value to indicate the direction in which the result is significant. For example, a value of $z > 0$ suggests that the mean value of a particular measure for F_Modules is greater than that for NF_Modules.

As shown in Table 4, we use 2D analysis to study the variation in quality in terms of the test effectiveness measures *NoFai* and *Test Coverage* across the releases. The correlation analysis is applied to answer all the research questions except for RQ1. The difference analysis is applied to all the measures except for *Release*, *NoFai*, and the derived measures dependent on *NoFai* (Fig. 1 and Table 2). We use [x, y] to indicate that the correlation between x and y is being tested (e.g., [*NoFai*, *Test Coverage*]) for RQ2 and [(x, y), z] to indicate that the correlation between x and z and the correlation between y and z are being tested (e.g., [(*FD*, *TS*, *TFM*), *TEFM*]). The symbol × means that no analysis is applied.

To answer the research questions, we conduct statistical analyses on multiple releases. We use 2D analysis to observe the overall pattern of variations in the measurement of measures

Table 4 Analyses to answer the research questions

Objective	Activity	Research question	Trend analysis	Correlation analysis	Difference analysis
O1	A1	*1*	*2D*	×	×
		2	*2D*	*[NoFai, Test Coverage]*	*Test Coverage*
O2	A2	*3*	*2D*	*[TEM, TEFM]*	*TEM*
		4	*2D*	*[CM, TEFM]*	*CM*
		5	*2D*	*[CM, TEM]*	×
	A3	*6*	*2D*	*[(FD, TS, TFM),TEFM]*	*TS, TFM_Cvg*
		7	*2D*	*[TS, TFM]*	×

along the releases. We check whether two different measures (e.g., *CM* and *TEM*) correlate with each other using correlation analysis and whether such correlation is statistically significant. In addition, we also check whether statistically different results can be observed for a measure (that is not defined based on *NoFai*, e.g. *TEM*) in the two groups of NF_Modules and F_Modules based on difference analysis. When comparing multiple samples as in our case, there is always a risk of drawing false conclusions that can be dealt with the False Discovery Rate (FDR) as suggested in (Benjamini and Hochberg 1995) to adjust the original *p*-values obtained after the correlation and difference analyses (Yekutieli and Benjamini 1999). In this paper, we only report the FDR adjusted *p*-values instead of the original ones. Using the results of correlation and difference analyses, we propose the recommendations to analyze how the quality of an RTOS4A varies across the releases, and how other measures affect the quality. The results of correlation analysis and difference analysis are investigated in three ways: (1) to assess the relations among the measures used to answer a specific research question; (2) to formulate recommendations based on the observed data; (3) to propose guidelines to indicate the variations in the quality, and further actions to improve the quality, if necessary.

3.6 Data collection

We collected the data from the documents provided by our industry partner according to the measures defined in Section 3.3, as shown in Table 5. Since a large amount of data was scattered throughout those documents, we collected data automatically by using scripts. In addition, the scripts automatically checked for data completeness and consistency. A data set is complete if all the relevant data are present in the retrieval data. For example, if, for a given function, no information about *LoC* is available in the automatically retrieved data, the relevant data for the function are missing. The same data from different data sources must be consistent. For example, the number of test cases is available from several data sources, including test execution log files and test coverage reports, and therefore the data should be consistent in both data sources.

As shown in Table 5, we had access to five types of documents to collect the test data. Our industry partner used the LDRA Testbed tool (http://www.ldra.com/index.php/en/products-a-services/ldra-tool-suite/ldra-testbed) to generate the source code analysis reports and test case coverage reports. We developed data collection scripts of over 2000 lines of Python. Figure 2 summarizes the data collection process.

Table 5 Documents used for automated data collection

Data source	Data collected
Source code analysis report for each release (.html)	List of modules for each release, *LoC* and *NoFunc* for each module in each release
Design document (.doc)	List of modules for each release and *NoFunc* for each module in each release
Source code of test suites for each release (.c)	List of test cases for each release and *LoC* for each test case in each release
Test execution log for each release (.txt)	List of test cases executed for each release and *NoFai* for each test case executed in each release
Test coverage report for each release (.html)	List of modules tested in each release and the three types of coverage for each module tested in each release

As shown in Fig. 2, there are six manual steps for data collection, that is, activities M1 to M6, and four automatic steps, that is, activities A1 to A4. We manually checked the completeness of the documents (M1), asked our industry partner to complete the documents when necessary (M2), checked whether the data items that could be retrieved from the documents covered all the measures (M3), developed scripts to retrieve data (M4), implemented consistency checking rules (M5), and checked the correctness of data retrieval (M6). To ensure the correctness of the data items collected, in M6, we asked our industry partner to review them and manually check some sections in the documents to see if the corresponding data items were correctly presented in the collected data set. Activity A1 retrieves the data items automatically retrieved using the scripts written in M4. In A2, the scripts automatically check the correctness of the data, and, in A3, the consistency of data is automatically checked using the consistency rules implemented in M5. Finally, the collected data are automatically formatted into an Excel file (A4) for the analyses.

4 Results and analyses

In this section, we answer the seven research questions. First, in Section 4.1, we briefly provide statistics for the case study, followed by seven sections (Section 4.2 to Section 4.8) answering each research question individually. When answering each research question, we at first introduce and conclude what can be observed from the collected data using the three analyses

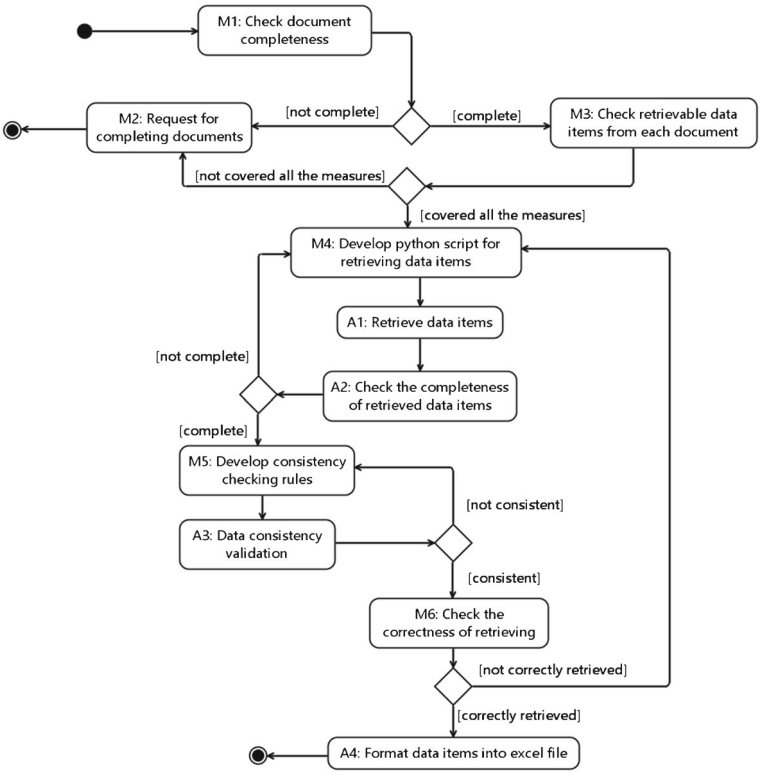

Fig. 2 Process to collect data from the documents

Table 6 Statistics for releases R1 to R8

Release	*#Modules*	*#Removed_Modules*	*NoFai*	$NoFai_{RM}$	$NoTC_{L+S}$	$NoTC_{HS}$	$NoTC_{RM}$	$NoTC_{AM}$
R1	51	0	166	0	2904	1043	0	0
R2	51	0	88	0	3069	1069	0	0
R3	70	16	26	11	3942	5759	611	867
R4	77	21	26	11	4209	5622	825	1098
R5	77	21	33	11	4236	0	825	1080
R6	56	0	18	0	3446	0	0	0
R7	56	0	9	0	3473	0	0	0
R8	56	0	7	0	3536	0	0	0

when applicable, i.e. trend analysis, correlation analysis and difference analysis; then a set of recommendations are proposed only based on the collected data.

4.1 Descriptive statistics

In Table 6, we present the key characteristics of the eight releases, where *#Modules* denotes the number of modules in a release and *#Removed_Modules* denotes the number of modules removed in a release compared to the preceding release. In R3 and R4, new modules were added and some modules were removed in R5. A total of 56 modules of R5 remained in R6 and 21 ones were removed. The measure $NoFai_{RM}$ is the number of failures found for the modules added in releases R3 to R5 but removed in R6, $NoTC_{L+S}$ is the number of test cases used in RLLT and RSIT, and $NoTC_{HS}$ is the number of test cases used in RHSIT (Section 1).

A total of 19 new modules were added in R3 (70–51), seven (77–70) modules were added in R4, and 21 (77–56) modules were removed in R6. Among the 21 modules removed in R6, 16 were added in R3 and five in R4. This is why 16 modules in the *#Removed_Modules* column of Table 6 for R3 were removed in R6, whereas 21 modules in R4 and R5 were removed in R6. There are two reasons for removing these 21 modules: 1) They are library modules in the C language providing library functions for user programs, do not have many interactions with the operating system, and are optional in the industry partner's release plan; 2) in these modules, the industry partner continuously found 11 failures in R3 to R5 for a small ratio of test cases, namely, 6.3 %, 8.4 %, and 19.5 %,[1] respectively. For this reason, the company removed these modules in R6, since they were not stable and did not contribute to key functionalities of the system. In addition, these modules are independent of other parts of the system.

In Table 6, we can see from the $NoTC_{AM}$ and $NoTC_{HS}$ columns for R3 and R4 that there were 1965 (867 + 1098) test cases for the new modules, but 11,381 (5759 + 5622) test cases for RHSIT. This means that, in R3 and R4, the main test effort was spent on RHSIT rather than testing the new modules. Since most of the failures were fixed in R3 in one of the hardware platforms targeted, the testing group started to conduct RHSIT with the other three targeted hardware platforms (for a total of four, as discussed in Section 1) in R3 and R4. As a result, 5759 test cases in R3 and 5622 test cases in R4 were developed for RHSIT. Note that all the

[1] This is the result of $NoTC_{RM}$ divided by the sum of $NoTC_{L+S}$ and $NoTC_{HS}$ for rows R3 to R5, respectively, in Table 6.

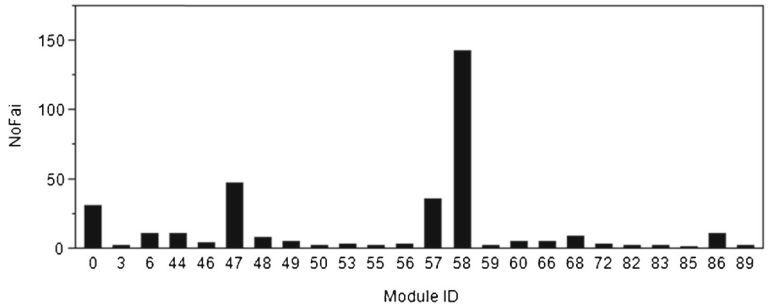

Fig. 3 Histogram of *NoFai* in the modules (F_Modules)

removed library modules were not taken out in RHSIT. Since the RTOS4A system was observed to be stable, based on the number of failures found in all four targeted platforms in R3 and R4, the testing group decided not to continue RHSIT from R5 on.

Based on the data for all eight releases, we find failures in 24 of 77 modules, as shown in Fig. 3.

In Fig. 3, *NoFai* represents the total number of failures observed in a module for all eight releases. We see that most of the failures were observed in just a few modules, such as module 58 and module 47. The eight modules with the highest numbers of failures together account for 306 failures, that is, 82 % of total failures (373). This means that over 80 % of the failures were in about 10 % of the modules (8/77 ≈ 10.4 %). This is a typical Pareto phenomenon frequently reported in the literature (Fenton and Ohlsson 2000), that is, most failures typically reside among a small number of modules of a software system.

4.2 Answer to RQ1

Recall that RQ1 aims to determine how RTOS4A quality varies with testing effectiveness as new versions of the RTOS4A are released (Section 3.1). To answer RQ1, we use 2D trend analysis, as shown in Fig. 4.

Results and conclusion Figure 4 shows *NoFai* is very high (166) for R1, which is normal for an initial release, since the RTOS4A was not mature then. As Fig. 4 shows, *NoFai* decreases with the latest release versions, although we observe a small increase in R5. In R8, *NoFai* decreases to a minimum, suggesting that the quality of the software has become stable. This kind of pattern is common and expected in practice, since the latest releases are usually more stable than earlier ones because the faults exposed in the earlier ones by the existing tests have already been fixed.

As shown in Fig. 4, all three types of coverage (i.e., *Stmt_Cvg*, *Branch_Cvg*, and *MCDC_Cvg*) are above 80 %. Recall from Section 2 that additional testing was performed to reach the 100 % coverage required by the DO-178B standard. We note a sudden drop in coverage in R3, which is due to the fact that, though a significant number of test cases were introduced in the release (9701), only about 8.9 %[2] of them were dedicated to the (19 = 70–51) new modules (Table 6). This implies that these modules were not sufficiently tested and overall coverage was dropped to about 89 % for *Stmt_Cvg*, 81 % for *Branch_Cvg*, and 80 % for *MCDC_Cvg*.

[2] Calculated as $NoTC_{AM}$ divided by the sum of $NoTC_{L+S}$ and $NoTC_{HS}$ for row R3 in Table 6.

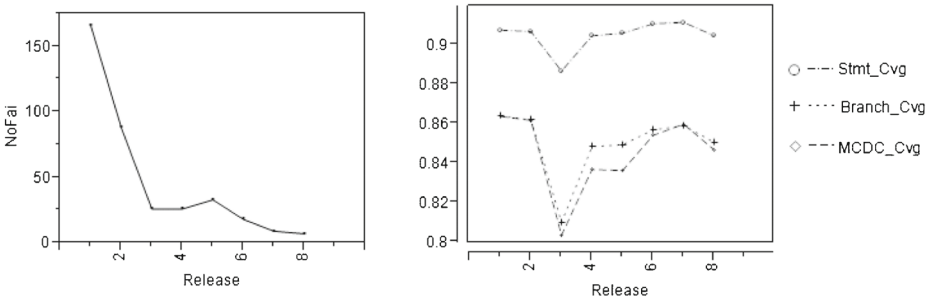

Fig. 4 2D trend analyses of *NoFai* and *Test Coverage* across releases

Recommendation As an initial step to assess quality, we plot the relations between the releases and test effectiveness measures (i.e., *NoFai*, *Stmt_Cvg*, *Branch_Cvg*, and *MCDC_Cvg*). Such plots provide an initial insight into the quality of the RTOS4A. We recommend interpreting the plots as follows.

1) The decrease of *NoFai* in the latest releases shows that quality is improving. A possible explanation is that the system's testing has reached a point where all failures due to offsetting bugs have been fixed successfully.
2) The stable and high percentage of *Test Coverage* throughout the releases suggests high confidence for quality.

4.3 Answer to RQ2

RQ2 aims to determine how various test effectiveness measures (*TEFM*) correlate with each other. We use 2D trend analysis, correlation analysis, and difference analysis to answer this research question. In Fig. 5, we plot a 2D scatter chart to illustrate the relation between *NoFai* and *Test Coverage* for the releases. We report the results of the Spearman's correlation test results in Table 7. Since NF_Modules and F_Modules cannot be compared based on *NoFai*, we perform difference analysis for the three types of test coverage, as shown in Table 8.

Results and conclusion Figure 5 shows that *NoFai* is very high for the earlier releases and then decreases for the latest releases; however, coverage stays almost stable for all the releases.

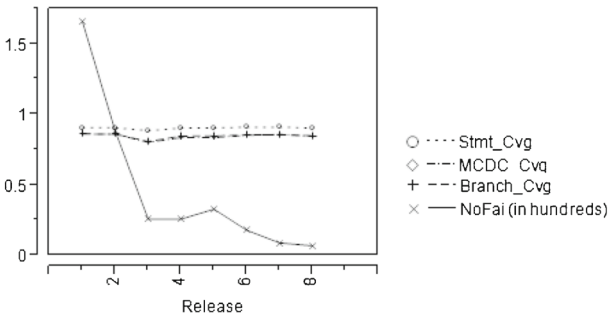

Fig. 5 Measures *NoFai* and *Test Coverage* across releases

Table 7 Correlation between *NoFai* and *Test Coverage* across the modules

Coverage	R1		R2		R3		R4		R5		R6		R7		R8	
	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value
Stmt_Cvg	−0.12	0.4046	−0.26	0.1074	−0.27	0.0685	−0.34	0.0176	−0.28	0.0596	−0.26	0.0938	−0.21	0.1448	−0.21	0.1448
Branch_Cvg	−0.17	0.2365	−0.28	0.1900	−0.18	0.2289	−0.29	0.0824	−0.20	0.2275	−0.20	0.2290	−0.17	0.2365	−0.17	0.2365
MCDC_Cvg	−0.11	0.4377	−0.23	0.2374	−0.15	0.3017	−0.34	0.0192	−0.25	0.1092	−0.21	0.2374	−0.18	0.2766	−0.14	0.3281

Table 8 Difference in *Test Coverage* for the F_Modules and NF_Modules

Coverage	R1		R2		R3		R4		R5		R6		R7		R8	
	z	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value
Stmt_Cvg	−0.75	0.4520	−0.75	0.4520	−1.21	0.3029	−2.18	0.0819	−2.16	0.0819	−1.49	0.2189	−1.60	0.2186	−2.18	0.0819
Branch_Cvg	−1.23	0.4346	−1.03	0.4872	−0.11	0.9161	−1.49	0.4346	−1.38	0.4346	−0.76	0.5099	−0.89	0.4957	−1.70	0.4346
MCDC_Cvg	−0.88	0.5042	−0.69	0.5571	0.23	0.8187	−1.30	0.5042	−1.20	0.5042	−0.92	0.5042	−1.11	0.5042	−1.54	0.5042

Table 7 shows that all the Spearman's coefficient values (ρ) for all the releases and all three types of test coverage are negative. This suggests that increasing the test coverage does not necessarily lead to finding more failures, as suggested in (Inozemtseva and Holmes 2014). At the same time, one can observe that most of the FDR adjusted *p*-values are greater than 0.05, suggesting that the correlation is not statistically significant. To avoid drawing conclusion about the quality of a system only based on test coverage, which may not always lead to more failures as discussed in (Marick 1999), we also conduct additional analyses based on the relationships between test coverage and test effort, complexity of module, and test strength, etc.

We observe that almost all the FDR adjusted *p*-values are greater than 0.05 for the three types of test coverage in Table 8. This means that the difference in coverage between F_Modules and NF_Modules for the three types of test coverage was not significant.

Recommendation Based on the above results, we can conclude that RTOS4A quality has been improved since *NoFai* has been decreased and *Test Coverage* has been stabilized. In general, we can say that the quality of the system has been improved if 1) an increase in the coverage does not lead to observing more failures and 2) no significant difference is found in *Test Coverage* between the modules without failures observed and the modules with failures observed.

4.4 Answer to RQ3

RQ3 is defined to determine how the quality of the releases varies with *TEM* (i.e., *NoTC* and *CoTC*). To answer this research question, we further define the following two sub-research questions (RQ3.1 and RQ3.2):

- **RQ3.1**: How does the number of failures of releases vary with test effort?
- **RQ3.2**: How does the test coverage of releases vary with test effort?

We use 2D plots to show the trends of *TEM* and *TEFM* across the releases. To evaluate the correlation between *TEM* and *TEFM*, we conduct a Spearman test. Difference analysis is performed to determine the difference in *TEM* between the F_Modules and NF_Modules.

4.4.1 Answer to RQ3.1

Recall that RQ3.1 aims to determine how various test effort measures (i.e., *NoTC* and *CoTC*) affect *NoFai*. Figure 6 shows the relations of *NoTC* and *CoTC* with *NoFai* for the releases in 2D scatter plots. We also conduct correlation analyses between the *TEM* and *NoFai* values of the modules for each release, as reported in Table 9. The difference analysis for *TEM* is reported in Table 10.

Results and conclusion The left plot in Fig. 6 shows that *NoTC* is less than 5000 for most of the releases, except for R3 and R4, which contain more than 9000 test cases (Table 6). The difference is due to the fact that (as discussed in Section 4.1), in these two releases, RHSIT was conducted with the other three targeted hardware platforms, which leads to the significant increment in *NoTC*.

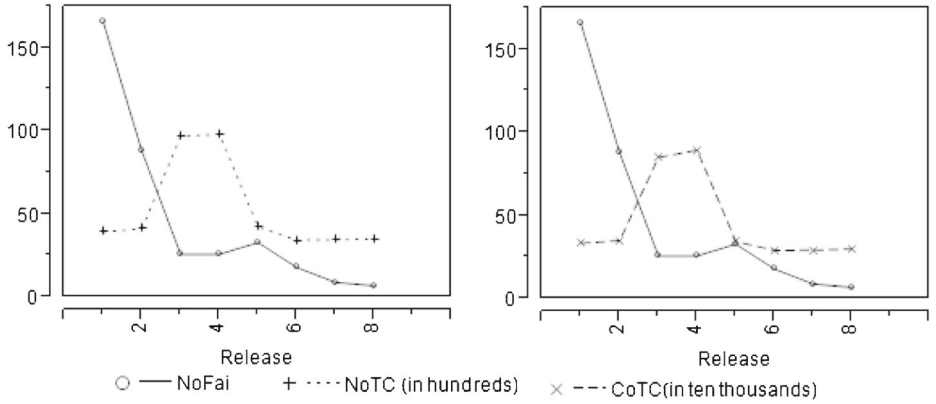

Fig. 6 Plots of *TEM* and *NoFai* across releases

As shown in Table 9, for each release, the values (ρ) of Spearman's coefficient for all the modules in the release are greater than zero, indicating a positive correlation between *NoTC* and *NoFai*. This means that we observe more failures when *NoTC* is increased. Moreover, all the FDR adjusted *p*-values are lower than 0.05, suggesting that the correlation is statistically significant. This phenomenon is commonly observed in testing in which the increased *NoTC* typically reveals more failures. However, note that, as observed in Section 4.2, *NoFai* decreases in the latest releases, even with the same *NoTC,* suggesting that the quality of the system was improved in the latest releases.

The right plot in Fig. 6 shows that *CoTC* is similar for all the releases except R3 and R4, but is quite high for these two releases because of the reason discussed in Section 4. The value of *NoFai* is high for the earlier releases (R1 to R3) and then decreases and remains stable for the latest releases. In Table 9, all the Spearman coefficients ρ are greater than zero, indicating a positive correlation between *CoTC* and *NoFai,* and all *p*-values are less than 0.05, indicating that the correlation is statistically significant. This result means that, as *CoTC* increases, more failures are observed but, as we observed in Section 4.2 when answering RQ1, *NoFai* is reduced in the latest releases. Again, this means that the quality of the system is improving.

The results of the difference analysis of *TEM* for NF_Modules and F_Modules are shown in Table 10. All the FDR adjusted *p*-values for *NoTC* and *CoTC* are less than 0.05 (in fact less than 0.0001), suggesting that testers put different degrees of test effort for NF_Modules and F_Modules. The positive *z*-statistics indicate that much less test effort was put in the NF_Modules, based on the discussion of the results presented in Section 3.5. Based on the results reported in Table 8, no significant difference in *Test Coverage* between F_Modules and NF_Modules was found. Therefore, less test effort for the NF_Modules does not invalidate the increasing quality of the system. We discuss the reasons for less test effort for the NF_Modules in Section 4.5.

4.4.2 Answer to RQ3.2

This sub-research question studies how various test efforts measures (i.e., *NoTC* and *CoTC*) affect *Test Coverage*. Figure 7 shows the relations between *NoTC* (or *CoTC*) and *Test Coverage*. We also conduct Spearman's correlation analysis between *TEM* and *Test Coverage* of the modules in each release (Table 11).

Table 9 Correlation between *TEM* and *NoFai* for all the releases

TEM	R1		R2		R3		R4		R5		R6		R7		R8	
	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value
NoTC	0.63	0.0004	0.49	0.0008	0.36	0.0024	0.37	0.0016	0.42	0.0004	0.44	0.0012	0.39	0.0024	0.40	0.0024
CoTC	0.63	0.0004	0.48	0.0010	0.34	0.0045	0.33	0.0030	0.41	0.0008	0.45	0.0010	0.40	0.0027	0.40	0.0027

Table 10 Difference in *TEM* between the F_Modules and NF_Modules

TEM	R1		R2		R3		R4		R5		R6		R7		R8	
	z	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value
NoTC	4.55	<0.0001	4.58	<0.0001	4.26	<0.0001	4.35	<0.0001	4.38	<0.0001	4.39	<0.0001	4.35	<0.0001	4.35	<0.0001
CoTC	4.61	<0.0001	4.59	<0.0001	4.13	<0.0001	4.21	<0.0001	4.36	<0.0001	4.39	<0.0001	4.34	<0.0001	4.33	<0.0001

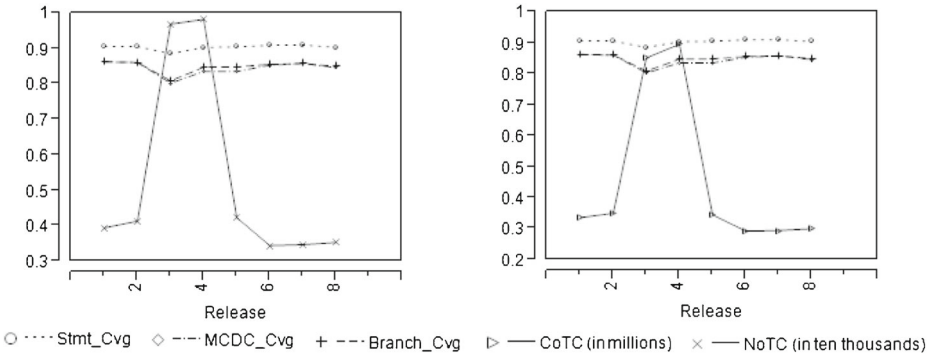

Fig. 7 Plots of *TEM* and *Test Coverage* across the releases

Results and conclusion Figure 7 shows that *NoTC* and *CoTC* are similar for all the releases, except R3 and R4, which are quite high for these two releases (Section 4.1). For each type of coverage and each release, all the values of the Spearman coefficient ρ are less than zero (Table 11), indicating a negative correlation between *NoTC* (and *CoTC*), and *Test Coverage* and most of the FDR adjusted *p*-values are less than 0.05, suggesting the correlation is statistically significant. This result means that, even with less *NoTC* or *CoTC*, high test coverage was achieved for all the releases except for R3 and R4. However, note that, even for these releases, the coverage was greater than 80 %. This finding suggests that the test cases in our case study were very effective and thus managed to achieve high coverage.

Recommendations To assess RTOS4A reliability, we suggest studying a) the trend of test effectiveness and test effort measures using 2D plots, b) the correlation between various test effectiveness and test effort measures using Spearman's correlation test, and c) the difference between the F_Modules and NF_Modules in terms of test effort measures using the Wilcoxon signed-rank test. Based on our experience, we recommend the following practices.

1) An increase in *NoTC* or *CoTC* must detect more failures; otherwise there are two potential explanations: a) the test cases are ineffective or redundant and therefore different test strategies must be used to develop new test cases or b) the quality of the system has been improved. If, with similar values of *CoTC* and *NoTC* across the latest releases, the number of failures has been decreased (as the case in our case study), this suggests that the quality of the system is improving.
2) Check *TEM* and *Test Coverage*. If *Test Coverage* is low or not stable, increased values of *NoTC* or *CoTC* must achieve greater test coverage; otherwise this means that test cases are ineffective, since they repeatedly cover the same program constructs (e.g., statements in statement coverage). If *Test Coverage* is high and stable and an increased value of *NoTC* or *CoTC* does not lead to increasing *NoFai*, this implies that quality is improved.
3) Study *TEFM* and *TEM* together. If *Test Coverage* is increased or is stable and reduced *NoFai* are observed across releases, then we can conclude that the quality of the system is improving.
4) Check the differences in test effort and test effectiveness for modules with and without failures observed. If the test coverage for the modules without failures observed is statistically equal to or higher than that for the modules with failures observed, we can conclude that the former do not invalidate the improvement of

Table 11 Correlation between *TEM* and *Test Coverage* for all the releases

TEM	Coverage	R1		R2		R3		R4		R5		R6		R7		R8	
		ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value
NoTC	*Stmt_Cvg*	−0.33	0.0173	−0.33	0.0173	−0.30	0.0161	−0.36	0.0024	−0.40	0.0006	−0.48	0.0005	−0.49	0.0004	−0.52	0.0004
	Branch_Cvg	−0.39	0.0118	−0.37	0.0118	−0.17	0.1672	−0.27	0.0182	−0.30	0.0119	−0.34	0.0119	−0.36	0.0119	−0.41	0.0119
	MCDC_Cvg	−0.39	0.0061	−0.38	0.0069	−0.17	0.1571	−0.35	0.0030	−0.38	0.0016	−0.44	0.0016	−0.46	0.0012	−0.48	0.0012
CoTC	*Stmt_Cvg*	−0.37	0.0109	−0.35	0.0122	−0.29	0.0162	−0.35	0.0029	−0.38	0.0012	−0.47	0.0005	−0.49	0.0004	−0.52	0.0004
	Branch_Cvg	−0.42	0.0080	−0.39	0.0112	−0.16	0.1921	−0.28	0.0160	−0.29	0.0156	−0.33	0.0156	−0.35	0.0156	−0.42	0.0080
	MCDC_Cvg	−0.42	0.0030	−0.41	0.0037	−0.17	0.1717	−0.34	0.0037	−0.36	0.0028	−0.44	0.0019	−0.46	0.0012	−0.50	0.0004

quality. Otherwise, more test effort on modules without failures observed is required to increase the test coverage and could thereby result in finding more failures.

4.5 Answer to RQ4

This research question is defined to determine how the quality of releases varies with RTOS4A complexity. We answer this research question based on its two sub-research questions, RQ4.1 and RQ4.2.

- **RQ4.1**: How does the number of failures of releases vary with RTOS4A complexity?
- **RQ4.2**: How does the test coverage of releases vary with RTOS4A complexity?

4.5.1 Answer to RQ4.1

RQ4.1 aims to determine how the complexity measures of the releases affect the values of *NoFai*. Recall that we use two specific complexity measures (i.e., *NoFunc* and *LoC*, as shown in Fig. 1 and Table 2) to measure the complexity of a release. Therefore, the 2D plot in Fig. 8 illustrates their relations with *NoFai* for each release. We report the results of the correlation analyses between the *NoFunc* and *LoC* values and the *NoFai* values of the modules for each release in Table 12. In addition, we report the results of the difference analysis of *NoFunc* and *Loc* in Table 13.

Results and conclusion Figure 8 shows that *NoFunc* does not change much across the releases; however, *NoFai* is quite high for the earlier releases and decreases with the later releases, suggesting improved RTOS4A quality. We note an increase in *NoFai* for R5 again followed by a decrease in *NoFai* for R6. Recall that, in R6, the 16 modules that were added in R3 and the five modules added in R4 were removed in R6 (Section 4.1). In Table 12, all the values of the Spearman coefficient ρ are greater than zero, indicating a positive correlation between *NoFunc* and *NoFai*. Moreover, all the *p*-values are less than 0.05, suggesting statistically significant correlations.

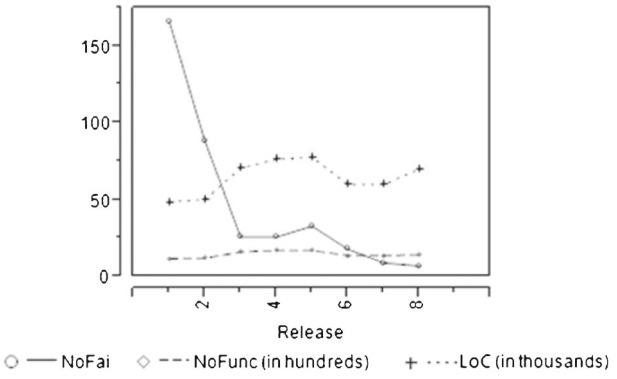

Fig. 8 Plots of *CM* and *NoFai* across releases

Table 12 Correlation between *CM* and *NoFai* of the modules for all the releases

CM	R1		R2		R3		R4		R5		R6		R7		R8	
	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value
NoFunc	0.72	0.0004	0.54	0.0004	0.30	0.0021	0.28	0.0021	0.32	0.0045	0.43	0.0021	0.40	0.0026	0.40	0.0026
LoC	0.50	0.0008	0.43	0.0025	0.38	0.0021	0.39	0.0011	0.41	0.0008	0.43	0.0018	0.40	0.0028	0.39	0.0033

Table 13 Differences in *CM* for the F_Modules and NF_Modules

CM	R1		R2		R3		R4		R5		R6		R7		R8	
	z	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value
NoFunC	5.36	<0.0001	5.36	<0.0001	4.51	<0.0001	4.63	<0.0001	4.62	<0.0001	5.16	<0.0001	5.10	<0.0001	5.10	<0.0001
LoC	3.58	0.0016	3.56	0.0016	3.15	0.0018	3.16	0.0018	3.12	0.0018	3.23	0.0018	3.20	0.0018	3.17	0.0018

Figure 8 also shows that *LoC* increases along the release timeline, especially for R3 to R5, since more modules were added in these three releases. However, *LoC* again decreases for R6, since some of the added modules in the previous three releases were removed (Section 4.1). The correlation analysis is shown in Table 12, where all the values of the Spearman coefficient ρ are greater than zero (also, all FDR adjusted *p*-values < 0.05), indicating a positive correlation between *LoC* and *NoFai* of the module that is statistically significant.

The results of the difference analysis (Table 13) show that the NF_Modules have significantly less complexity than the F_Modules, since all the FDR adjusted *p*-values are less than 0.05 and all the *z*-statistics are positive. This may explain why the test effort for the F_Modules is significantly higher than for the NF_Modules (Table 10) because our industry partner considered the complexity of modules as a factor to allocate test effort. Furthermore, we investigate the test strength by considering these two measures together in Section 4.7.

4.5.2 Answer to RQ4.2

RQ4.2 aims to determine how various complexity measures affect *Test Coverage*. In this section, we report the results in Fig. 9 to show the impact of *NoFunc* and *LoC* on *Test Coverage*. We also conduct correlation analyses between *CM* and *Test Coverage* and report the results in Table 14.

Results and conclusion Figure 9 shows that *Test Coverage* increases in the latest releases, especially for R3 to R5, for the reason discussed in Section 4.1. Figure 9 also shows that *NoFunc* is similar across all the releases; however, all three types of coverage are stable for all the releases (above 80 %, on average). In Table 14, there are three rows in the coverage column for *NoFunc* and *LoC* for each of the releases. We find that all the values of the Spearman coefficient ρ, except that for *NoFunc* and *MCDC_Cvg* in R3, are less than zero, indicating a negative correlation between *NoFunc* and *Test Coverage*, as well as between *LoC* and *Test Coverage,* and most of the FDR adjusted *p*-values are less than 0.05, indicating that the correlation is statistically significant. This result means that *Test Coverage* decreases as *NoFunc* or *Loc* increases. We also observe that the *p*-values for *LoC* are much smaller than those for *NoFunc*, which means the correlation between *Loc* and *Test Coverage* is statistically significant.

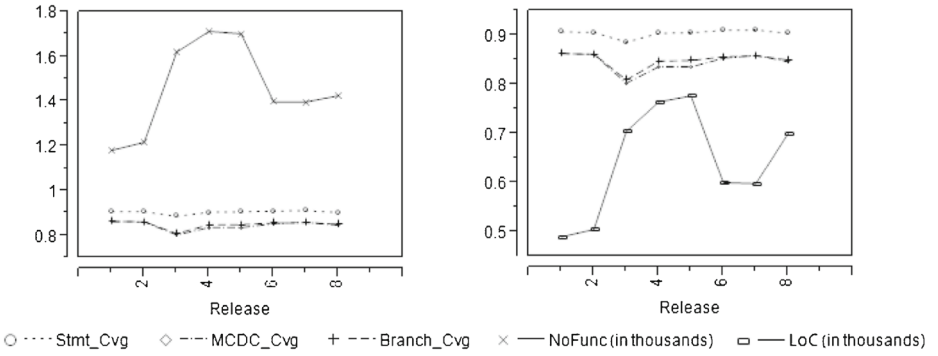

Fig. 9 Plots of *CM* and *Test Coverage* across the Releases

Table 14 Correlation between *CM* and *Test Coverage* for the modules

CM	Coverage	R1		R2		R3		R4		R5		R6		R7		R8	
		ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value
NoFunc	*Stmt_Cvg*	−0.29	0.0446	−0.29	0.0446	−0.24	0.0463	−0.32	0.0086	−0.31	0.0098	−0.38	0.0086	−0.39	0.0086	−0.44	0.0056
	Branch_Cvg	−0.36	0.0347	−0.35	0.0347	−0.01	0.9421	−0.23	0.0864	−0.21	0.0864	−0.24	0.0886	−0.25	0.0864	−0.33	0.0347
	MCDC_Cvg	−0.35	0.0261	−0.35	0.0261	0.00	0.9854	−0.27	0.0261	−0.26	0.0266	−0.31	0.0261	−0.33	0.0261	−0.39	0.0261
LoC	*Stmt_Cvg*	−0.61	<0.0001	−0.59	<0.0001	−0.61	<0.0001	−0.59	<0.0001	−0.58	<0.0001	−0.66	<0.0001	−0.67	<0.0001	−0.69	<0.0001
	Branch_Cvg	−0.70	<0.0001	−0.67	<0.0001	−0.37	0.0015	−0.50	<0.0001	−0.49	<0.0001	−0.54	<0.0001	−0.55	<0.0001	−0.61	<0.0001
	MCDC_Cvg	−0.69	<0.0001	−0.68	<0.0001	−0.37	0.0017	−0.49	<0.0001	−0.47	<0.0001	−0.50	<0.0001	−0.54	<0.0001	−0.61	<0.0001

Recommendations To assess the quality of a system, we suggest studying the correlation between various complexity and test effectiveness measures using 2D plots to observe their trends and using Spearman's correlation analysis to test the statistical correlation. If the test effort for the F_Modules is significantly greater than for the NF_Modules, we suggest studying the difference between the NF_Modules and F_Modules in terms of *CM*. We recommend the following practices.

1) An increased *NoFunc* or *LoC* in the latest releases correlating significantly with a low *NoFai* suggests either improvement in the quality of the system or insufficient testing. If *Test Coverage* is high and holds well as the new versions are released, the quality of the system has been improved; otherwise, a decreased *Test Coverage* in the latest releases with a low *NoFai* suggests insufficient testing.
2) Without loss of generality, an increased *NoFunc* or *LoC* correlating significantly with a high *NoFai* in the current releases compared to the preceding releases (though not observed in our case study) suggests decreased quality and we recommend greater test effort in future releases.
3) Greater complexity of the NF_Modules observed with lower test effort indicates insufficient testing for the NF_Modules compared to the F_Modules. This suggests greater test effort is required for the NF_Modules.

4.6 Answer to RQ5

RQ5 is defined to determine how quality varies with the correlation between RTOS4A complexity and test effort. If there is evidence showing that the distribution of test effort overlooks the factor of module complexity, drawing a conclusion on improved quality might be biased, because the test on the modules with higher complexity might not be as sufficient as the test on the other modules. Figure 10 is a 2D plot showing the various complexity and test effort measures of the releases. The values of the Spearman coefficient ρ for correlation analyses between the various complexity and test effort measures of the modules are provided in Tables 15 and 16.

Results and conclusion As shown in Fig. 10, with the increase of *LoC* or *NoFunc*, *NoTC* and *CoTC* increase accordingly. We can also observe the same phenomenon in Tables 15 and 16. There are positive correlations between *NoFunc* and *NoTC*, between *NoFunc* and *CoTC*, between *LoC* and *NoTC*, and between *LoC* and *CoTC,* since all the Spearman coefficient values (ρ) are greater than zero and all the FDR adjusted *p*-values are less than 0.05,

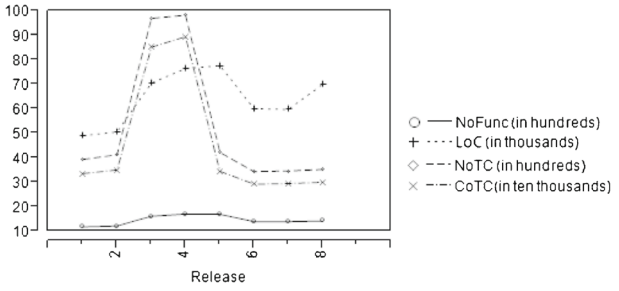

Fig. 10 Plots of CM and TEM across the releases

Table 15 Correlation between *NoFunc* and *TEM* of the modules

TEM	R1		R2		R3		R4		R5		R6		R7		R8	
	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value
CoTC	0.94	<0.0001	0.94	<0.0001	0.86	<0.0001	0.87	<0.0001	0.93	<0.0001	0.95	<0.0001	0.95	<0.0001	0.94	<0.0001
NoTC	0.93	<0.0001	0.94	<0.0001	0.84	<0.0001	0.84	<0.0001	0.93	<0.0001	0.95	<0.0001	0.95	<0.0001	0.94	<0.0001

Table 16 Correlation between *LoC* and *TEM* of the modules

TEM	R1		R2		R3		R4		R5		R6		R7		R8	
	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value
CoTC	0.86	<0.0001	0.86	<0.0001	0.76	<0.0001	0.78	<0.0001	0.88	<0.0001	0.88	<0.0001	0.88	<0.0001	0.85	<0.0001
NoTC	0.87	<0.0001	0.87	<0.0001	0.74	<0.0001	0.75	<0.0001	0.86	<0.0001	0.87	<0.0001	0.87	<0.0001	0.85	<0.0001

suggesting that the correlations are statistically significant. In other words, the higher the complexity of a module, the higher the test effort it received.

Recommendation To avoid the biased evaluation of the quality of a system based on testing, we suggest studying the correlation between *CM* and *TEM* and how *TEM* changes as *CM* changes in all the releases. We recommend the following practices.

1) If increasing *NoFunc* or *LoC* correlates with more test effort to achieve an acceptable level of coverage, we can conclude that the testing was not biased and the quality evaluation would therefore not be invalidated. Otherwise, this suggests insufficient testing and more test effort needs to be spent on modules with a higher *NoFunc* or *LoC* to achieve unbiased quality evaluation.
2) If increasing *NoFunc* or *LoC* correlates with increasing *TEM* and, at the same time, decreasing *Test Coverage*, this means that the current test strategy cannot increase the test coverage of modules with a higher complexity measure by increasing test effort, which therefore suggests a new test strategy to improve *Test Coverage* to increase the chance of uncovering more failures.

4.7 Answer to RQ6

RQ6 is defined to determine how the quality of the releases varies with different test efficiency measures, that is, *FD*, *TFM,* and *TS*. We break up this research question into two sub-research questions (RQ6.1 and RQ6.2), presented as follows.

- **RQ6.1**: How does the number of failures of releases vary with testing efficiency?
- **RQ6.2**: How does the test coverage of releases vary with testing efficiency?

4.7.1 Answer to RQ6.1

RQ6.1 answers how the *FD*, *TFM,* and *TS* of the releases affect *NoFai*. We use 2D plots in Fig. 11 to show the trends of *FD*, *TS*, *TFM,* and *NoFai* for the eight releases. In Table 17, we report the correlation between *TS*, *TFM,* and *NoFai* using Spearman's test. We do not test the correlation between *FD* and *NoFai* or that between *TFM_NoFai* and *NoFai*, since *FD* and *TFM_NoFai* are derived from *NoFai*. We report the results of the Wilcoxon signed-rank test for *TS* and *TFM_Cvg* in Table 18.

Results and conclusion From Fig. 11, we can see that *FD* slightly decreases from R1 to R8, suggesting an increase in quality. The measures *TS_NoTC* and *TS_CoTC* increase from R1 to R3, showing that much more test effort was spent, since the number of modules increases from 51 (in R2) to 70 (in R3), as shown in Table 6. Recall that three additional target hardware platforms were added to RHSIT in R3 and R4 (Section 4.1) and a large percentage of test cases were added for these two releases (59.4 % in R3, 57.2 %[3] in R4). Therefore, test strength in R3

[3] It is computed by $NoTC_{HS}/(NoTC_{HS} + NoTC_{L+S})$.

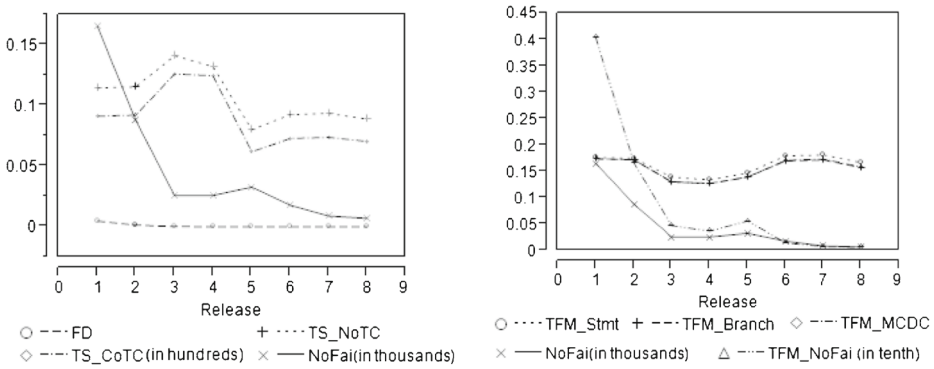

Fig. 11 Plots of *FD*, *TS*, *TFM,* and *NoFai* across the releases

and R4 was much higher than in the other releases. As introduced in Section 4.1, in R5, most of the test effort focused on those modules added in R3 and R4, which decreased the *TS_NoTC* and *TS_CoTC* in R5. When those 21 library modules were removed in R6, *TS* was reinforced and maintained in the latest releases, though we observed a small drop in R8. We also observe in Table 17 that all the *p*-values for these two measures are greater than 0.05, suggesting no statistically significant correlation between *TS* and *NoFai* and, therefore, improved quality.

A large percentage of the test cases in R3 and R4 were designed for RHSIT and not intended to increase *Test Coverage* and, therefore, *TFM_Stmt*, *TFM_Branch,* and *TFM_MCDC* are quite low for these two releases. However, the three types of coverage related to test efficiency increase from R5 to R7. This suggests that quality is improved, since *NoFai* and *TFM_NoFai* decrease from R5 to R8 at the same time. We can see in Table 17 that *TFM_Stmt*, *TFM_Branch*, and *TFM_MCDC* have a negative significant correlation with *NoFai,* since most of the FDR adjusted *p*-values are less than 0.05 and the ρ values are less than zero.

Based on the results of difference analysis in Table 18, we can see that *TS_NoTC* does not show a statistically significant difference between NF_Modules and F_Modules. These results further support our conclusion of improved quality and suggest that there were no significant differences in effort spent on testing the NF_Modules and F_Modules. However, *TS_CoTC* shows a statistically significant difference; that is, the F_Modules, on average, have more complicated test cases than the NF_Modules, since all the FDR adjusted *p*-values (except that in R1) are less than 0.05 and the *z*-statistic is positive. Since the F_Modules have statistically higher complexity in terms of *LoC* and *NoFunc,* as reported in Table 13, and no significant difference in coverage between NF_Modules and F_Modules was found (reported in Table 8), the test cases for the F_Modules use many more combinations of inputs. For the same reason, we can see in Table 18 that the NF_Modules have statistically greater test efficiency in terms of coverage than the F_Modules, since all the FDR adjusted *p*-values are less than 0.05 and the *z*-statistic is negative.

4.7.2 Answer to RQ6.2

RQ6.2 aims to determine how *FD*, *TFM_NoFai,* and *TS* affect *Test Coverage*. Since *TFM_Stmt*, *TFM_Branch,* and *TFM_MCDC* are derived from the three corresponding

Table 17 Correlations between *TS*, *TFM_Cvg,* and *NoFai* of the modules

Measure	R1		R2		R3		R4		R5		R6		R7		R8	
	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value
TS_NoTC	0.11	0.9203	0.01	0.9639	−0.05	0.9203	−0.06	0.9203	−0.39	0.4744	−0.14	0.9203	−0.15	0.9203	−0.18	0.9203
TS_CoTC	0.22	0.8279	0.13	0.8279	0.05	0.8279	0.03	0.8279	−0.03	0.8279	0.07	0.8279	0.03	0.8279	0.03	0.8279
TFM_Stmt	−0.52	0.0442	−0.40	0.0923	−0.35	0.0932	−0.43	0.0501	−0.45	0.0488	−0.61	0.0442	−0.55	0.0442	−0.52	0.0442
TFM_Branch	−0.62	0.0002	−0.49	0.0005	−0.37	0.0020	−0.39	0.0008	−0.44	0.0002	−0.44	0.0008	−0.39	0.0027	−0.40	0.0026
TFM_MCDC	−0.63	0.0002	−0.49	0.0005	−0.37	0.0021	−0.40	0.0006	−0.44	0.0002	−0.44	0.0009	−0.39	0.0030	−0.40	0.0023

Table 18 Differences between *TS* and *TFM_Cvg* for the F_Modules and NF_Modules

Measure	R1		R2		R3		R4		R5		R6		R7		R8	
	z	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value	*z*	*p*-Value
TS_NoTC	0.54	0.5921	0.65	0.5873	1.81	0.2731	1.86	0.2731	1.63	0.2731	1.11	0.4848	0.99	0.4848	0.91	0.4848
TS_CoTC	1.12	0.2626	4.58	<0.0001	4.26	<0.0001	4.35	<0.0001	4.38	<0.0001	4.39	<0.0001	4.35	<0.0001	4.35	<0.0001
TFM_Stmt	−4.53	<0.0001	−4.53	<0.0001	−4.25	<0.0001	−4.45	<0.0001	−4.51	<0.0001	−4.41	<0.0001	−4.38	<0.0001	−4.40	<0.0001
TFM_Branch	−4.51	<0.0001	−4.53	<0.0001	−4.19	<0.0001	−4.39	<0.0001	−4.46	<0.0001	−4.33	<0.0001	−4.29	<0.0001	−4.31	<0.0001
TFM_MCDC	−4.53	<0.0001	−4.51	<0.0001	−4.13	<0.0001	−4.22	<0.0001	−4.32	<0.0001	−4.23	<0.0001	−4.19	<0.0001	−4.29	<0.0001

coverage measures, they must be highly correlated with the coverage measures and therefore we do not include them in the correlation analysis.

Results and conclusion As we can see in Fig. 12, the three types of test coverage remain stable over 80 %, while *FD* and *TFM_NoFai* decrease. These results are similar to those reported in Section 4.3. We conduct the correlation analysis in Table 19 using Spearman's test. The results show that *FD* and *TFM_NoFai* do not have a statistically significant correlation with the three types of coverage, since almost all the FDR adjusted *p*-values are greater than 0.05.

We can see in Fig. 12 that *TS_NoTC* and *TS_CoTC* drop from R3 to R5 and then increase and remain stable until R8. Recall that 19 and seven new modules were added in R3 and R4, respectively, and RHSIT was conducted in R3 and R4 (Table 6 and the analysis in Section 4.1), leading to a slight drop of *TS_NoTC* and *TS_CoTC* in R3 and R4. In R5, *TS* further decreases, since RHSIT was stopped and a high percentage ($(825+1080)/4236 \approx 45$ %; see Table 6) of test effort was made for the newly added 26 (=19 + 7) modules. Spearman's correlation test in Table 19 shows the positive significant correlation between *TS_NoTC*, *TS_CoTC,* and the three types of coverage since all the FDR adjusted *p*-values are less than 0.05.

According to the results, the increase in *TS* suggests an increase in *Test Coverage*, although the increase in test effort does not suggest an increase in test coverage according to Table 11. Considering the complexity of the system, many functions are designed and implemented to detect and handle errors and faults caused by user applications and hardware devices. These functions are very difficult to cover with the applied testing methods (i.e., RLLT, RSIT, and RHSIT), which are not effective at activating the corresponding errors and faults. Therefore, simply increasing the test effort might not increase coverage. The positive correlation between *TEM* and *Test Coverage* reported in Table 11 suggests that *TS* should be increased to improve coverage. It is true, as noted in Section 2, that the additional tests (e.g., code inspection and simulation tests) were conducted to achieve 100 % coverage. Unfortunately, we do not have the data for the additional tests; therefore, we cannot analyze how the test strength of additional tests is correlated to the three types of test coverage.

Recommendations Based on the evaluation of *FD*, *TEFM*, *TEM,* and *CM*, we recommend the following practices by evaluating how *FD*, *TFM,* and *TS* affect *TEFM*. To conduct such analyses, the derived measures (i.e., *FD*, *TFM,* and *TS*) need to be defined and measured.

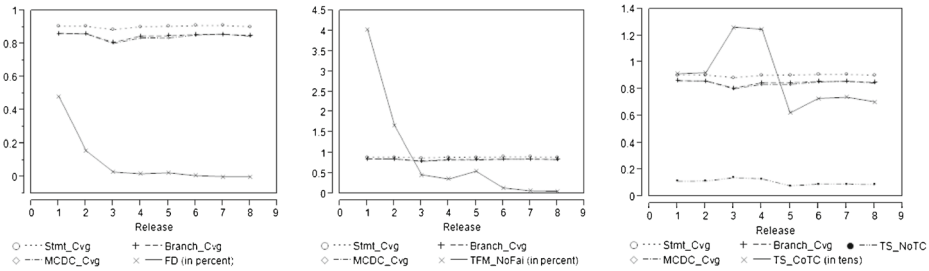

Fig. 12 2D plots of *FD*, *TFM_NoFai*, *TS,* and *Test Coverage* across the releases

Table 19 Correlation between *FD*, *TFM_NoFai*, *TS,* and *Test Coverage* of the modules

Measure	*Test Coverage*	R1		R2		R3		R4		R5		R6		R7		R8	
		ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value
FD	*Stmt_Cvg*	0.44	0.2268	−0.28	0.3631	−0.23	0.3631	−0.44	0.2268	−0.24	0.3631	−0.35	0.3631	−0.22	0.4202	−0.15	0.5532
	Branch_Cvg	−0.67	0.6649	−0.23	0.2725	−0.15	0.2803	−0.29	0.0768	−0.19	0.2725	−0.19	0.2803	−0.16	0.2803	−0.17	0.2803
	MCDC_Cvg	−0.02	0.8727	−0.19	0.3141	−0.12	0.3525	−0.35	0.0160	−0.25	0.1208	−0.20	0.3141	−0.18	0.3141	−0.14	0.3526
TFM_NoFai	*Stmt_Cvg*	0.35	0.2952	−0.31	0.3214	−0.33	0.2952	−0.45	0.2128	−0.23	0.3589	−0.35	0.2952	−0.22	0.4202	−0.14	0.5737
	Branch_Cvg	−0.08	0.5821	−0.24	0.2613	−0.17	0.2613	−0.29	0.0752	−0.19	0.2613	−0.19	0.2613	−0.16	0.2803	−0.17	0.2803
	MCDC_Cvg	−0.04	0.8042	−0.19	0.3141	−0.14	0.3263	−0.35	0.0144	−0.25	0.1232	−0.20	0.3141	−0.18	0.3141	−0.14	0.3423
TS_NoTC	*Stmt_Cvg*	0.68	<0.0001	0.67	<0.0001	0.56	<0.0001	0.36	0.0015	0.45	<0.0001	0.52	<0.0001	0.52	<0.0001	0.45	0.0006
	Branch_Cvg	0.75	<0.0001	0.76	<0.0001	0.45	0.0001	0.35	0.0017	0.46	<0.0001	0.58	<0.0001	0.58	<0.0001	0.53	<0.0001
	MCDC_Cvg	0.76	0.0002	0.76	0.0002	0.44	0.0003	0.35	0.0018	0.41	0.0003	0.46	0.0003	0.46	0.0003	0.47	0.0003
TS_CoTC	*Stmt_Cvg*	0.62	0.0001	0.63	0.0001	0.52	0.0001	0.28	0.0126	0.40	0.0006	0.45	0.0006	0.45	0.0008	0.38	0.0042
	Branch_Cvg	0.69	0.0001	0.70	0.0001	0.42	0.0003	0.29	0.0094	0.43	0.0002	0.54	0.0001	0.53	0.0001	0.46	0.0003
	MCDC_Cvg	0.70	0.0002	0.70	0.0002	0.41	0.0008	0.31	0.0055	0.40	0.0008	0.43	0.0014	0.41	0.0022	0.41	0.0022

1) If *FD* decreases and, at the same time, *Test Coverage* does not decrease in the latest releases, we can conclude that quality is improved. Otherwise, an increased *FD* in the latest releases suggests that more effective bug fixing is required.
2) If there are no significant differences in *TS* and *TFM* between the NF_Modules and F_Modules, we can conclude that quality is improved. Otherwise, we might find failures in the NF_Modules by increasing the test strength and test efficiency.
3) If *NoFai* is increased along with increased *TFM_Cvg* or *TS*, quality is decreased and, therefore, more test effort is needed to improve quality. Otherwise, *NoFai* decreased along with increased *TFM_Cvg* or *TS* suggests improved quality.

4.8 Answer to RQ7

RQ7 determines how various *TS* and *TFM* measures are correlated with each other. Since the difference analysis of the derived measures is reported in Section 4.7, we conduct the 2D analyses in Fig. 13 and the correlation analysis in Table 20 to answer this research question.

Results and conclusion As we can see in Fig. 13, *TFM_NoFai* is high in R1 and then drops to a value near zero in R8, although a small increase is observed in R5. We also find that the three types of coverage (*TFM_Stmt*, *TFM_Branch,* and *TFM_MCDC*) remain quite stable for all the releases, with almost the same levels and trends. This result suggests that quality is improved, since more defects were removed from the system, the efficiency to detect new failures decreases, and the efficiency to maintain the level of *Test Coverage* remains stable. The value of *TS* does not change too much across the releases. The trends of *TS_CoTC* and *TS_NoTC* were introduced in Section 4.7.1.

According to the correlation analysis shown in Table 20, there are no statistically significant correlations between the two *TS* measures and the four *TFM* measures, since all the FDR adjusted *p*-values are greater than 0.05. This means that testers did not improve test efficiency by increasing test strength. Generally, higher test efficiency can improve quality to a certain level in a shorter time. Considering that RTOS4A testing lasted over 17 months and test efficiency to achieve test coverage remained stable across all the releases, we can conclude that quality is improved.

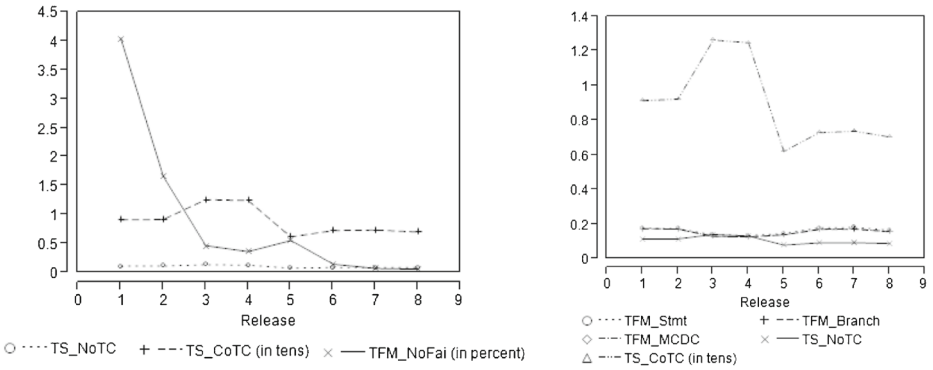

Fig. 13 Plot of *TS* and *TFM* across the releases

Table 20 Correlation between *TS* and *TFM* of the modules

TS	TFM	R1		R2		R3		R4		R5		R6		R7		R8	
		ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value	ρ	*p*-Value
TS_NoTC	*TFM_NoFai*	0.14	0.9979	0.05	0.9979	0.05	0.9979	0.03	0.9979	−0.06	0.9979	0	0.9992	−0.02	0.9979	−0.02	0.9979
	TFM_Stmt	0.07	0.6443	0.09	0.6236	−0.23	0.4792	−0.18	0.5092	0.07	0.6236	0.13	0.6236	0.13	0.6236	0.09	0.6236
	TFM_Branch	0.07	0.6231	0.09	0.5913	−0.22	0.5216	−0.17	0.5418	0.08	0.5913	0.15	0.5418	0.15	0.5418	0.12	0.5913
	TFM_MCDC	0.07	0.6339	0.09	0.6339	−0.22	0.5088	−0.17	0.5260	0.07	0.6339	0.12	0.6339	0.13	0.6339	0.09	0.6339
TS_CoTC	*TFM_NoFai*	0.20	0.9633	0.11	0.9633	0.02	0.9633	0	0.9633	−0.05	0.9633	0.06	0.9633	0.02	0.9633	0.03	0.9633
	TFM_Stmt	−0.04	0.9994	−0.05	0.9994	−0.28	0.1680	−0.23	0.1880	0.03	0.9994	0.01	0.9994	0.02	0.9994	0	0.9994
	TFM_Branch	−0.03	0.8567	−0.04	0.8567	−0.27	0.1920	−0.22	0.2192	0.04	0.8567	0.03	0.8567	0.04	0.8567	0.02	0.8567
	TFM_MCDC	−0.03	0.9691	−0.04	0.9691	−0.27	0.1744	−0.22	0.2068	0.03	0.9691	0.01	0.9691	0.02	0.9691	0	0.9691

Recommendation If test strength increases, *TFM_NoFai* decreases to a very small value, and, at the same time, *TFM_Cvg* remains stable, we can conclude that the quality of RTOS4A has been improved. Otherwise, an increase in *TS* correlating with an increase in *TFM_NoFai* or a decrease in *TFM_Cvg* suggests it is necessary to add more test effort to uncover greater numbers of failures or to increase *Test Coverage*.

5 Guidelines to assess quality

Figure 14 shows the three sets of guidelines that we devised based on answering the research questions and our experience of working with the industrial case study of the RTOS4A: 1) data collection and measurement, 2) quality improvement identification, and 3) quality improvement suggestions.

As shown in Fig. 14, we recommend three main activities under data collection and measurement, namely, measure definition, data collection and validation, and measurement and analysis. To assess quality, we first recommend defining the research questions to achieve the research objectives (Section 3.1) and defining the necessary measures (e.g., as proposed in Section 3.3). With these defined measures, data can be collected from the documents based on a particular development process (e.g., DO-178B in our case study). Data collected from real applications are usually inconsistent and incomplete (e.g., as discussed in Section 3.6). Therefore, it is important that the data be validated before quality assessment. We recommend applying trend analysis to determine the change of a measure (e.g., *NoFai*) along software releases, correlation analysis to determine how two measures (e.g., *NoFai* and *NoTC*) are correlated with each other, and difference analysis to determine whether a measure (e.g., *Test Coverage*) is observed statistically different in the NF_Modules and F_Modules. These three types of analyses provide the observations for variation in quality. Based on the results of applying the analyses and proposed recommendations during answering the research questions, we propose the respective indicators to identify whether quality was improved or not, and how to improve quality in further if necessary. The rationale to propose the indicators to identify improved quality is to look into the change of *NoFai* and *Test Coverage* between releases, the change of *Test Coverage*, and the differences of other relevant test measures in NF_Modules and F_Modules. If there is a statistical positive correlation between a test measure x (e.g. number of test cases) and a test effectiveness measure y (e.g. test coverage), then a suggestion would be formulated to increase x (i.e. to add more test cases) to increase y (i.e. to increase test coverage) and then to improve the quality; or if a test measure (e.g. number of test case) or test coverage in NF_Modules is found statistically lower than the one in F_Modules, then a suggestion would be formulated to increase test effort or to increase test coverage for modules in NF_Modules.

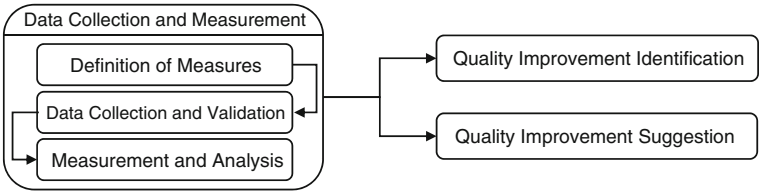

Fig. 14 Guidelines to assess quality

5.1 Guidelines to identify improved quality

Based on our results, we provide 9 indicators (D1 to D9) to recognize improved reliability in release R_i compared with the previous release R_{i-1}, as shown in Table 21.

In practice, we recommend checking all the indicators in Table 21 if possible. The more times one could infer improved quality, the more confidence one would have in the conclusion (i.e., improved quality). Generally, we recommend studying the quality of a system based on various test effectiveness measures and the relations between them. In our case study, quality assessment is mainly based on test coverage and the number of failures, where we expect reasonably high coverage together with decreased numbers of failures in the latest releases.

5.2 Guidelines to improve quality

Improving quality is an iterative process. Based on the guidelines presented in Fig. 14 and the indicators presented in Table 21, one can assess whether quality of a specific release was improved based on test data. In practice, practitioners are mostly concerned with efficiently improving the quality of each release. We propose guidelines to improve the quality of future releases by providing six suggestions (S1 to S6) and 16 indicators (D10 to D25), as shown in Table 22.

The indicators of D10 to D15, D18 to D21, and D25 are self-explanatory; however, we discuss the rest below.

- D16: Recall that test strength is defined as the ratio of test effort to *LoC* and *TFM_NoFai* is defined as the ratio of *NoFai* to test effort, as shown in Fig. 1. This means that the increase

Table 21 Indicators to identify improved quality in a release by comparison with its preceding release

# Indicator	Explanation
D1	For a release R_i, the number of failures (*NoFai*) and failure density (*FD*) have decreased (Section 4.2, Section 4.7) and test coverage is either high or the same as for R_{i-1} (Section 4.2).
D2	For a release R_i, the test coverage for the NF_Modules is not lower than for the F_Modules (Section 4.3, Section 4.4).
D3	For a release R_i, increasing test coverage does not correlate with increasing the number of failures (*NoFai*) (Sections 4.3 and 4.4).
D4	For a release R_i, increasing test effort (*TEM*) does not correlate with increasing the number of failures (*NoFai*) or decreasing test coverage (Section 4.4).
D5	For a release R_i, test coverage is either higher than or the same as for R_{i-1} and increasing test effort (*TEM*) does not correlate with increasing the number of failures (*NoFai*) (Section 4.4).
D6	For a release R_i, higher complexity (*CM*) of module does not correlate with more failures (*NoFai*) uncovered and test coverage is either higher than or the same as for R_{i-1} (Section 4.5).
D7	For a release R_i, the test strength (*TS*) and test efficiency (*TFM*) of the NF_Modules are not lower than the corresponding values of the F_Modules (Section 4.7).
D8	For a release R_i, increasing test strength (*TS*) or coverage-oriented test efficiency (*TFM_Cvg*) correlates with decreasing the number of failures (*NoFai*) uncovered (Section 4.7).
D9	For a release R_i, increasing test strength (*TS*) does not correlates with increasing the test efficiency measured using the number of failures (*TFM_NoFai*) and test efficiency based on coverage (*TFM_Cvg*) remains stable, as for R_{i-1} (Section 4.8).

Table 22 Indicators and suggestions to improve the quality of releases

# Indicator	Explanation	Suggestion
D10	For R_i, the test coverage achieved is lower than that for R_{i-1} (Section 4.2).	S1: Increase test effort to improve test coverage, uncover more failures, or increase test strength.
D11	For R_i, increasing test coverage correlates with increasing the number of failures (*NoFai*) (Section 4.3, Section 4.4).	
D12	For R_i, increasing test effort (*TEM*) correlates with uncovering more failures (*NoFai*) (Section 4.4).	
D13	For R_i, test coverage is lower than for R_{i-1} and increasing test effort (*TEM*) correlates with a decrease in test coverage (Section 4.4).	
D14	For R_i, increasing test effort (*TEM*) or test strength (*TS*) correlates with increasing test coverage (Sections 4.4 and 4.7).	
D15	For R_i, increasing coverage-oriented test efficiency (*TFM_Cvg*) or test strength (*TS*) correlates with detecting more failures (*NoFai*) (Section 4.7).	
D16	For R_i, increasing test strength correlates with increasing failure-oriented test efficiency (*TFM_NoFai*) (Section 4.8).	
D17	For R_i, increasing test effort (*TEM*) does not correlate with detecting more failures (*NoFai*), but increasing test strength (*TS*) does (Sections 4.4 and 4.7).	S2: Since increasing test effort does not improve test effectiveness or test strength, use a new test strategy.
D18	For R_i, test coverage remains stable as for R_{i-1} and increasing test strength (*TS*) correlates with increasing the test effort (Section 4.7).	
D19	For R_i, the test coverage of the NF_Modules is statistically lower than the test coverage of the F_Modules (Section 4.3).	S3: Increase test effort or improve test coverage or test strength for the NF_Modules.
D20	For R_i, the test effort (*TEM*) of the NF_Modules is statistically lower than that of the F_Modules, while the module complexity of the NF_Modules is statistically greater than the module complexity of the F_Modules (Section 4.4).	
D21	For R_i, the test strength (*TS*) or test efficiency (TFM) of the NF_Modules is statistically lower than the corresponding values of the F_Modules (Section 4.7).	
D22	For R_i, increasing module complexity (*CM*) correlates with either increasing the number of failures (*NoFai*) or decreasing test coverage (Section 4.5).	S4: Increase test effort on the modules with higher complexity.
D23	For R_i, increasing module complexity (*CM*) does not correlates with increasing test effort (*TEM*) (Section 4.6).	
D24	For R_i, increasing module complexity (*CM*) correlates with increasing test effort (*TEM*) but decreasing test coverage (Section 4.6).	S5: Apply a new test strategy to improve the coverage of the modules of higher complexity.
D25	The number of failures (*NoFai*) uncovered or failure density (*FD*) is not decreased in R_i compared to its previous release, R_{i-1} (Sections 4.2 and 4.7).	S6: Improve the effectiveness of bug fixing.

in the number of failures uncovered (i.e., increased *TFM_NoFai*) is even higher than the increase in test effort (i.e., increasing test strength); increasing the test effort is then expected to detect more failures.

- D17: Increasing test effort does not correlate with detecting more failures but increasing test strength does. This means that adding more test cases or increasing the complexity of

test cases does not correlate with detecting more failures. A new test strategy should be defined to develop test cases that consider the complexity of modules.

- D22: Increasing module complexity correlates with either increasing the number of failures uncovered or decreasing test coverage. This means that modules of higher complexity should draw more test effort to detect defects or to increase test coverage.
- D23: Increasing module complexity correlates with decreasing test effort. This means that test efforts for modules of higher complexity are less sufficient and therefore need to be increased.
- D24: Increasing module complexity correlates with decreasing test coverage and increasing test effort. This means that testers intend to expend more test effort on modules of higher complexity but fail to maintain the test coverage. Therefore, a new test strategy is required to focus on the constructs of the code that are not covered by existing test strategies.

6 Threats to validity

Below we report threats to the validity of our case study based on the template reported by (Wohlin et al. 2000).

6.1 Conclusion validity

As with most case studies in software engineering, one main threat to conclusion validity is related to the sample size on which we base our analysis. Note that we collected the test data of eight releases for 17 months and thus believe that the sample size in our case is sufficiently large.

In order to avoid errors introduced in the analyses performed by just one person, all the authors of this paper validated the analyses and their results individually. The final results of the analyses were also presented to developers in our industrial partner who validated the conclusions and recommendations.

6.2 Internal validity

In our case study, we tried to minimize the effect of the other factors on results that were not part of the case study. For example, we used the same tool to automatically generate reports about code coverage—that is, *Stmt_Cvg*, *Branch_Cvg*, and *MCDC_Cvg*—and source code statistics—that is, *LoC* and *NoFunc*—for each release and used the same scripts to retrieve data from documents and reports. In addition, we used the same set of rules to validate the data.

6.3 Construct validity

One possible threat to construct validity in our case study is that we did not use all possible measures, for example, when a failure is found to assess reliability. However, note that we used both atomic measures (i.e., test effort measures, complexity measures, and test effectiveness measures) and derived measures (i.e., failure density, test efficiency measures, and test strength measures) to assess RTOS4A reliability. These measures are commonly used in the literature to

assess reliability. Moreover, we employed three types of analysis—that is, trend analysis, correlation analysis, and difference analysis—to assess the improvement of reliability and suggest practices to improve tests. These analyses fit the needs for the analysis we are targeting. In the future, we can conduct more detailed analysis by defining additional measures, including time information.

6.4 External validity

The threat to external validity is the most common threat in any case study. Since we analyzed the data for only one RTOS4A case study, one could argue that the results and recommendations cannot be generalized. However, note that we studied eight RTOS4A releases over 17 months and thus our analyses are based on a large amount of data. However, we expect more similar case studies need to be conducted to further support the results and recommendations presented in this paper.

7 Related work

Conducting quantitative software quality assessment can be tracked back to the paper published in 1976 by (Boehm et al. 1976), where a hierarchy of quality characteristics was proposed (later known as the quality model), which was standardized as one part of the ISO 9126–1 standard (ISO/IEC 9126-1 (2001)) and later on the ISO 25010 standard (2011). For analyzing software quality, metrics are often defined and used as indicators to find potential problems and such metrics could also be used to define guidelines for software development and testing (Boehm et al. 1976). In this paper, we use product metrics (e.g., number of functions) and test metrics (e.g., number of test cases) to conduct software quality assessment and identify opportunities to further improve software quality across releases. We applied three different kinds of analyses, i.e., trend, correlation and difference analyses to assess the quality of eight continues releases of an industrial avionics software, based on data collected during the testing process of our industrial partner. Therefore, we limit the scope of the literature review (to be reported in this section) to quantitative quality and reliability analysis based on data collected during testing processes and pay a particular attention to guidelines.

We report some of the related work on reliability analysis using testing data and other metrics for product and process in this paper since we use some common measures, though we focus on assessing quality instead of reliability. Incorporating testing metrics (e.g., the number of test cases, test coverage) to conduct software reliability assessment is becoming a common practice (Fujii et al. 2011), and is also applicable in assessing software quality. Test effort is a measure of how much effort spent on testing, and has impacts on software quality. In this paper, we use two measures to quantify test effort, i.e. number of test cases, and complexity of test case (i.e. lines of code of test case). Size of a smoke GUI test suite was investigated to assess its impact on fault detection effectiveness. The empirical study by testing five software applications shows that the number of faults detected grows with test suite size (Memon and Xie 2005). Several factors have been reported in the literature to measure software reliability. For instance, Zhang and Pham (2000) identified 32 such factors based on a survey of 13 companies. Most commonly used factors are related to program complexity, programmer skill, testing coverage, test effort, and the testing environment. The main contribution of the work by Zhang and Pham is in identifying such factors that can be used to measure reliability, which differs from the aim of our

case study. However, we study the relations among the various factors that contribute to the quality assessment of the RTOS4A based on test data collected from a large-scale real-world industrial project. Schneidewind (1999) uses a set of selected metrics, including time to next failure and failures per thousand lines of code (same as the failure density defined in this paper), to determine variation in quality. In comparison with this work, our study uses test effort measures together with complexity measures and test effectiveness measures to assess quality and the other three types of derived measures (e.g., test efficiency measure). In addition, we use correlation analysis and difference analysis to assess variation in quality.

Software quality is closely related to the software's development and testing. Reliability is treated as a quality attribute by the Software Assurance Technology Center of the National Aeronautics and Space Administration in the United States and metrics for requirements (e.g., lines of text in the requirement specifications), coding (e.g., the number of lines of code), and testing (e.g., requirement coverage) are used to evaluate the impact of the three development phases on reliability (Rosenberg et al. 1998). Four case studies were conducted with four software development teams (three from Microsoft and one from IBM) (Nagappan et al. 2008) to evaluate whether test-driven development can decrease the pre-release defect density using team and product metrics. The results show that it was decreased by 40 to 90 % compared to similar projects that did not use test-driven development. Along this line, the software testing and reliability early warning (STREW) metric suite (Nagappan et al. 2005) was proposed in building a linear regression model to estimate post-release failures in the coding and testing phases in a case study. The STREW metrics suite covers the testing, complexity, and structure of object-oriented programs. Encouraged by the effective prediction of failures with the in-process metrics reported by Nagappan et al. (2005), Nagappan et al. (2006) then conducted a case study with the extended suite of metrics covering the development process (e.g., changed LoC between versions, the frequency of code modifications, the number of developers) and the product (e.g., test coverage and complexity) to create statistical predictors for estimating the post-release failures of Windows XP SP1 and failure-prone binary modules of Windows Server 2003. However, the prediction accuracy of failures for a project obtained from the data of other projects might be different, as the results reported in the industrial case studies of 12 real-world applications (Zimmermann et al. 2009). Differing from these case studies of failure prediction, our work investigates how software quality evolves with consecutive releases and how product and testing factors impact variation in quality. Results of the coverage analysis on the tests of the two real case studies show that testing coverage gives a clear measure of testing quality, and also reveals software quality (Horgan et al. 1994). In our paper, we use much more testing data (e.g. number of test cases) and product data (e.g. number of functions) to investigate the variations of quality along releases.

Software testing is an important technique for developing dependable software systems. A case study was conducted to evaluate the effectiveness of testing to assure quality (Lyu et al. 2003). In the case study, a total of 34 programming teams were engaged to independently develop multiple software versions of a critical flight application and the faults detected in these versions during the unit, integration, and acceptance testing were collected. The metrics employed include the program size, the number of modules, the number of functions, the number of code blocks, the number of decisions, the number of c-use and p-use (Ntafos 1998), and the number of mutants generated. The results indicate that faults could be detected and removed more efficiently with an increase in testing coverage. A similar finding is reported in an empirical study (Mockus et al. 2009) conducted with two dissimilar industrial software development projects (from Avaya and Microsoft). The results show that the increase in test coverage is

correlated with a decrease in the number of failures reported in field operations. However, in our case study, we do not observe a significant correlation between test coverage and the number of failures found in testing, since all the modules have approximately similar test coverage.

The quality of functional test is proposed to measure by its structural coverage (Marick 1991). The efficiency to detect failures by structural coverage and input distribution was compared, the increase in failure detection efficiency drops for test coverage based test selection method when test coverage achieved is up to some level (David and Podgurski 2003). The relationship between the effectiveness of coverage testing and mutation testing evaluate was compared empirically under different testing profiles, and code coverage was found as a moderate indicator of the capability of fault detection (Cai and Lyu 2005). In our paper, we do not assume any relationship between test coverage and fault detection. And in the case study that there is no statistical significant correlation between test coverage and number of failures was found. Since the coverages of statement, branch and MCDC are required by the standard DO-178B in the domain of our case study, we can only use test coverage, number of failures found in testing, and other test measures and product measures to assess the quality of RTOS4A.

The identification of fault-prone modules can guide more effective test design that can potentially improve system quality. The case study reported by Khoshgoftaar et al. (1996) uses a discriminant analysis technique to identify fault-prone modules based on 16 static software product metrics (e.g., the number of statements, the number of comments, the number of function calls, McCabe cyclomatic complexity) and the amount of code changed during debugging in the two consecutive releases of a large-scale telecommunication system with over 38,000 procedures in 171 modules. Unlike our case study, quality was not assessed with test data but, instead, with the number of predicted fault-prone modules in the case study. We also find other, similar empirical analyses where logistic regression models were designed to predict the error-prone modules of Apache 2.0 from data collected from Apache 1.3 (Denaro and Pezze 2002), and of the joint surveillance target attack radar system by using configuration management data, source code and problem reporting system data (Khoshgoftaar and Allen 1999). Metrics including program size, Halstead software science metrics, program structure, and McCabe cyclomatic complexity were used. In comparison with these empirical studies, our work does not propose any regression model but investigates the relations between the factors and failures found and test coverage achieved.

Empirical evaluation is important in understanding phenomena observed in real industrial projects to suggest improvements in quality. A case study was conducted by Fenton and Ohlsson (2000) to check some of the hypotheses about failure distribution, prediction, and failure density based on data collected from two consecutive releases of a switching system development process of Ericsson Telecom AB. The dependent variables defined in the case study include faults in function, system, and site tests and faults in a year of operation. The independent variables of the study include module size, the cyclomatic complexity of the module, and domain-specific metrics (e.g., the number of modified communication signals). This is the closest work to ours, but in a different domain. In the RTOS4A domain, requirements-based testing and MCDC are explicitly required by DO-178B for safety-critical software; therefore, specific development and testing activities are (partially) enforced for such software, thereby leading to phenomena that can be observed from the data. The results of the case study (Fenton and Ohlsson 2000) show that a small number of modules contain most of the faults in the pre-release testing and a very small number of modules contain most of the faults discovered in operation. No evidence was found that module size relates to fault density, as concluded by Basili and Perricone (1984) and Moller and Paulish

(1993), and no evidence shows module complexity to be a good predictor of failure- or fault-prone modules.

Software complexity metrics can be used to conduct quality analyses. Zimmermann and Nagappan report (2008) that about 30 % of the modules that developers considered critical were predicted by using complexity metrics in the empirical study. Kevrekidis et al. (2009) also report that software complexity has a positive correlation with the number of failures found during testing, but no significant correlation is observed with the number of failures found during operation. The case study reported by Kevrekidis et al. (2009) was conducted on the software of an x-ray cardiovascular scanner, using 18 complexity metrics for software size, structure, and criticality (i.e., the number of callers). Furthermore, it was found that "components that have outgoing dependencies to components with higher object-oriented complexity tend to have fewer field failures for Vista,[4] but the opposite relation holds for Eclipse"[5] (Zimmermann et al. 2011). These software complexity findings show that one should be careful in using the cyclomatic metric to conduct any failure prediction. In our case study, we use the number of lines of code instead of cyclomatic complexity to conduct the analyses and we also conduct correlation analyses and difference analyses with the number of lines of code.

8 Conclusion

To assess the quality of consecutive releases of an RTOS4A during its development phase, in this paper, we propose a set of practical guidelines, based on systematic analyses of real test data collected over eight releases of an industrial RTOS4A, for practitioners to improve test effectiveness and therefore the quality of the system during the development phase of the RTOS4A and other, similar kinds of systems. Test data were collected based on various atomic measures, such as test effort (e.g., the number of test cases), test effectiveness (e.g., test coverage), and release complexity (e.g., the number of functions) and derived measures such as failure density, test strength, and test efficiency. We performed trend analysis to study how the measures change with the releases, studied the correlation between various measures by using Spearman's correlation analysis, and investigated the differences in various measures between modules with failures observed and modules without failures observed, using the Wilcoxon signed-rank test. A set of recommendations was proposed in this paper to serve as guidelines to assess whether the quality of the RTOS4A was improved and to suggest ways of further improving quality. These guidelines will be also useful to other practitioners (in addition to our industrial partner, a large RTOS4A product and service provider in China) in the avionics domain to assess the quality of their avionics software during the development phase.

Acknowledgments This research is jointly supported by the Technology Foundation Program (JSZL2014601B008) of the National Defense Technology Industry Ministry, the State Key Laboratory of the Software Development Environment (SKLSDE-2013ZX-12). This work was also supported by the MBT4CPS project funded by the Research Council of Norway (grant no. 240013/O70) under the category of Young Research Talents of the FRIPO funding scheme. Tao Yue and Shaukat Ali are also supported by the EU Horizon 2020 project U-Test (http://www.u-test.eu/) (grant no. 645463), the RFF Hovedstaden funded MBE-CR (grant no. 239063) project, the Research Council of Norway funded Zen-Configurator (grant no. 240024/F20) project, and the Research Council of Norway funded Certus SFI (grant no. 203461/O30) (http://certus-sfi.no/).

[4] Microsoft's name for the operating system. See https://en.wikipedia.org/wiki/Windows_Vista for details.
[5] An open-source platform. See www.eclipse.org for details.

References

Basili VR, Perricone BT (1984) Software errors and complexity: an empirical investigation. Commun ACM 27(1):42–52

Benjamini Y, Hochberg Y (1995) Controlling the false discovery rate: a practical and powerful approach to multiple testing. J R Stat Soc Ser B Methodol 57(1):289–300

Boehm BW, Brown JR, Lipow M (1976) Quantitative evaluation of software quality[C]. In: Proceedings of the 2nd international conference on Software engineering. IEEE Computer Society Press, p 592–605

Cai X, Lyu MR (2005) The effect of code coverage on fault detection under different testing profiles. ACM SIGSOFT Softw Eng Notes 30(4):1–7

David L, Podgurski A (2003) A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. In: Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE). IEEE Computer Society Press, p 442–453

Denaro G, Pezze M (2002) An empirical evaluation of fault proneness models. Software Engineering, 2002. ICSE 2002. In: Proceedings of the 24rd International Conference on. IEEE, 2002.

Fenton NE, Neil M (1999) Software metrics: successes, failures and new directions. J Syst Softw 47:149–157

Fenton NE, Ohlsson N (2000) Quantitative analysis of faults and failures in a complex software system. IEEE Trans Softw Eng 26(8):797–814

Fenton N, Pfleeger SL (1996) Software metrics—A rigorous and practical approach, 2nd edn. International Thomson Computer Press, London

Fujii T, Dohi T, Fujiwara T. Towards quantitative software reliability assessment in incremental development processes[C]. In: Proceedings of the 33rd International Conference on Software Engineering. ACM, p 41–50

Hair JF, Black WC, Babin BJ, Anderson RE, Tatham RL (2006) Multivariate data analysis, vol 6. Pearson Prentice Hall, Upper Saddle River

Horgan JR, London S, Lyu MR (1994) Achieving software quality with testing coverage measures[J]. Computer 27(9):60–69

Hutchings M, Goradia T, Ostrand T et al (1994) Experiments of the effectiveness of dataflow-and controlflowbased test adequacy criteria[C]. In: Proceedings of the 16th international conference on Software engineering. IEEE Computer Society Press, p 191–200

IEEE standard for software test documentation, IEEE Std 829-1998, 16 Dec. 1998

Inozemtseva L, Holmes R (2014) Coverage is not strongly correlated with test suite effectiveness[C]. In: Proceedings of the 36th International Conference on Software Engineering. ACM, p 435–445

International Standard ISO/IEC 12207, Information Technology-Software Life Cycle Processes, International Organization for Standardization, International Electrotechnical Commission, 1995

ISO/IEC 9126-1:2001: Software Engineering – Product Quality. Part 1: Quality Model. Geneva, Switzerland: International Organization for Standardization, 2001

ISO/IEC 25010 (2011) Systems and software engineering — systems and software quality requirements and evaluation (SQuaRE) — system and software quality models[J]. International Organizationfor Standardization, 2011: 34

Kevrekidis K, Albers S, Sonnemans PJM, Stollman GM (2009) Software complexity and testing effectiveness: an empirical study[C]. Reliability and Maintainability Symposium, 2009. RAMS 2009. Annual. IEEE, 2009: 539–543

Khoshgoftaar TM, Allen EB (1999) Logistic regression modeling of software quality. Int J Reliab Qual Saf Eng 6(04):303–317

Khoshgoftaar TM, Allen EB, Goel N, Nandi A, McMullan J (1996) Detection of software modules with high debug code churn in a very large legacy system[C]. Software Reliability Engineering. In: Proceedings, SeventhInternational Symposium on. IEEE, p 364–371

Lyu MR, Huang Z, Sze KS, Cai X (2003) An empirical study on testing and fault tolerance for software reliability engineering[C]. Software Reliability Engineering. ISSRE 2003. 14th International Symposium on. IEEE, p 119–130

Malaiya YK, Li N, Bieman J, Karcich R, Skibbe B (1994) The relationship between test coverage and reliability[C]. Software Reliability Engineering, p 186–195. In: Proceedings, 5th International Symposium on. IEEE,

Malaiya YK, Li N, Bieman J, Karcich R (2002) Software reliability growth with test coverage. IEEE Trans Reliab 45(4):420–426

Marick B (1991) Experience with the cost of different coverage goals for testing[C]. In: Proceedings Pacific Northwest Soft. Quality Conf, p 147–164

Marick B (1999) How to misuse code coverage[C]. In: Proceedings of the 16th Interational Conference on Testing Computer Software, p 16–18

Memon AM, Xie Q (2005) Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software. IEEE Trans Softw Eng 31(10):884–896

Mockus A, Nagappan N, Dinh-Trong TT (2009) Test coverage and post-verification defects: A multiple case study[C]. Empirical Software Engineering and Measurement. ESEM 2009. 3rd International Symposium on IEEE, p 291–301

Moller K-H, Paulish D (1993) An empirical investigation of software fault distribution[C]. Software Metrics Symposium. In: Proceedings., First International. IEEE, p 82–90

Musa JD, Iannino A, Okumoto K (1987) Software reliability: measurement, prediction, application. McGraw-Hill, New York

Nagappan N, Williams L, Vouk M, Osborne J (2005) Early estimation of software quality using in-process testing metrics: a controlled case study. ACM SIGSOFT Softw Eng Notes 30(4):1–7

Nagappan N, Ball T, Murphy B (2006) Using historical in-process and product metrics for early estimation of software failures, Proceedings of the 17th International Symposium on Software Reliability Engineering, pp. 62–74

Nagappan N, Maximilien EM, Bhat T, Williams L (2008) Realizing quality improvement through test driven development: results and experiences of four industrial teams. Empir Softw Eng 13:289–302

Ntafos SC (1998) A comparison of some structural testing strategies. IEEE Trans Softw Eng 6:868–874

Rice J (2006) Mathematical statistics and data analysis, 3rd edn. Nelson Education

Rosenberg L, Hammer T, Shaw J (1998) Software metrics and reliability. Proceedings of the Ninth International Symposium on Software Reliability Engineering

RTCA/DO-178B, Software Considerations in Airborne Systems and Equipment Certification, December 1, 1992

Schneidewind NF (1999) Measuring and evaluating maintenance process using reliability, risk, and test metrics. IEEE Trans Softw Eng 25(6):768–781

Wohlin C, Runeson P, Höst M, Ohlsson M, Regnell B, Wesslén A (2000) Experimentation in software engineering: an introduction. Kluwer Academic Publishers, Norwell

Wu J, Ali S, Yue T, Tian J (2013) Experience report: Assessing the reliability of an industrial avionics software: Results, insights and recommendations[C]. Software Reliability Engineering (ISSRE), IEEE 24th International Symposium on. IEEE, p 218–227

Yekutieli D, Benjamini Y (1999) Resampling-based false discovery rate controlling multiple test procedures for correlated test statistics. J Stat Plann Infer 82(1–2):171–196

Zhang X, Pham H (2000) An analysis of factors affecting software reliability. J Syst Softw 50(1):43–56

Zimmermann T, Nagappan N (2008) Predicting defects using network analysis on dependencygraphs[C]. Proceedings of the 30th international conference on Software engineering. ACM, p 531–540

Zimmermann T, Nagappan N, Herzig K, Premraj R, Williams L (2011) An empirical study on the relation between dependency neighborhoods and failures[C]. Software Testing, Verification and Validation (ICST), 2011 IEEE FourthInternational Conference on. IEEE, p 347–356

Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B (2009) Cross-project defect prediction: a large scale experiment on data vs. domain vs. process, Proceedings of the Seventh Joint Meeting of the European Software Engineering Conference/ACM SIGSOFT Symposium on the Foundations of Software Engineering, p 91–100

Ji Wu Dr., associate professor, assistant dean of School of Computer Science and Engineering (SCSE), Beihang University. He received his Ph.D. degree from Beihang University in 2003, and M.S. degree from Second Research Institute of the China Aerospace Science and Industry Group in 1999. He focuses on the industry-oriented researches, and the main research interests include embedded system and software modeling and verification, software requirement and architecture modeling and verification, safety and reliability assessment, and software testing. He was invited to visit Simula Research Laboratory for one year in 2012.

Shaukat Ali is currently a senior research scientist in the Software Engineering department, Simula Research Laboratory, Norway. He has been affiliated to Simula Research Lab since 2007. He has been involved in many industrial and research projects related to Model based Testing (MBT) and Search Based Software Engineering since 2003. He has experience of working in several industries and academic research groups in many countries including UK, Canada, Norway, and Pakistan. Shaukat has been on the program committees of several international conferences and also served as a reviewer for several software engineering journals.

Tao Yue is now a chief research scientist of Simula Research Laboratory, Oslo, Norway, where she is leading the expertise area of Model Based Engineering (MBE).She is also affiliated to University of Oslo as an associate professor. She has received the PhD degree in the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada in 2010. Before that, she was an aviation engineer and system engineer for seven years. She has around 16 years of experience of conducting industry oriented research with a focus on MBE in various application domains such as Avionics, Maritime and Energy, and Communications in several countries including Canada, Norway, and China. Her present research area is software engineering, with specific interested in requirements engineering, requirements-based testing, model based product line engineering,model based system engineering, model based testing,search based software engineering and empirical software engineering. Dr. Yue has been on the program and organization committees of many international, IEEE and ACM conferences such as ACM/IEEE International Conference on Model Driven Engineering, Languages and Systems, International Conference of Requirements Engineering, and International Systems and Software Product Line Conference. She is also on the editorial board of on the Editorial Board of Empirical Software Engineering Journal. She is PI and CO PI of several national and international research projects.

Jie Tian currently is a Ph.D. student studying at School of Computer Science and Engineering, Beihang University. Her research interests include software reliability analysis and software testing.

Chao Liu Professor, Director of Software Engineering Institute(SEI), Beihang University(BUAA), Beijing, China. His research interests include software quality engineering, software testing, as well as software process improvement. He received his Ph.D. degree and M.S. degree in Computer Software and Theory at Beihang University, and his B.S. Degree in Mathematics at Beijing University of Posts and Telecommunication.