Empirical study of Software Quality Evaluation in Agile Methodology Using Traditional Metrics

Kumi Jinzenji
NTT Software Innovation Canter
NTT Corporation
Tokyo, Japan
jinzenji.kumi@lab.ntt.co.jp

Takashi Hoshino
NTT Software Innovation Canter
NTT Corporation
Tokyo, Japan
hoshino.takashi@lab.ntt.co.jp

Laurie Williams
Department of Computer Science
North Carolina State University
williams@csc.ncsu.edu

Kenji Takahashi
NTT Innovation Institute, Inc.
kt@ntti3.com

Abstract— **The use of highly iterative software development methodologies, such as Agile and Lean, have been growing. However, these methodologies do not explicitly provide practices for managing and measuring software quality. This deficiency may prevent software development organizations in critical domains from transforming from traditional development to highly iterative development. These organizations may need to manage quality during development and may desire to compare with the quality of a large number of products produced in the past. In this paper, we focus on the reliability aspects of software quality and discuss the applicability of conventional reliability metrics to iterative development. First, we defined the general process structure in iterative development. Then, we present an associated quality evaluation scheme closely. Our experimental results indicate that traditional quality metrics were applicable to iterative development and that the iterative development achieved maturity sufficient for the commercial release.**

Keywords-iterative development; software reliability; quality metrics

I. INTRODUCTION

The use of iterative software development methodologies, such as Agile [1] and Lean [2], is increasing to satisfy the need to respond to fast moving market demand and for gaining market share. The traditional software development methodology, often represented by Waterfall, employs a sequential process consisting typically of requirement definition, specification, design, construction, unit testing, integration testing, and system testing [3]. A tester can only evaluate and a user can only see working software towards the end of this type of process. This type of process yields "big-bang" projects and may lead to losses in time, cost and also market opportunity. On the other hand, a highly iterative software development creates small pieces of software, often called "user stories" in Agile, over time, which eases the problems mentioned above.

Unfortunately, the iterative development methodologies these methodologies do not explicitly provide practices for managing and measuring quality and reliability, as described in ISO/IEC 9126 [4]. This deficiency may prevent software development organizations in critical domains from transforming from traditional development to highly iterative development. These organizations may need to manage quality during development and may desire to compare with the quality of a large number of products produced in the past. In some industrial cases, management has hampered the transformation due to their perception of uncertainty of software quality in iterative development. Therefore, an in-process quality evaluation scheme is strongly desired.

In waterfall and spiral software development projects, the majority of the testing is done at the end of the development process. Quality artifacts, such as defect reports, can be related to the time of discovery during the testing cycle and to the associated software components. These artifacts and metrics can then be used to predict latent bugs and product reliability. However, often in agile-style iterative development, testing occurs and defects are discovered and fixed before the end of the relatively short iterations rather than being evaluated at the end of the lifecycle close to product release. As a result, comparing predicted software quality with products produced in the past is difficult. In addition, the quality of iteratively developed software is often primarily confirmed by demonstrating that the acceptance criteria are satisfied. However, existing acceptance criteria may not be enough to test all functionalities.

However, we hypothesize that highly iterative development does not yield worse quality than sequential development. In highly iterative development, more important functionalities are generally constructed and tested earlier. These important functionalities are regression tested in subsequent construction of additional code. *Our hypothesis is that iterative development can yield code with better "maturity" than sequential development, if testing is*

done appropriately. Maturity is defined in ISO/IEC 9126, as freedom of failures caused by faults existing in the software itself. The purpose of our study is to examine this hypothesis experimentally and to detect factors and conditions.

In NTT laboratories, most of software is prototype. Some of the software is provided into real service and product for the business companies, who require strict quality of software. Much development for the prototype software is still Waterfall-based because the quality evaluation scheme has been established, and significant data is available comparison to the past product quality. However, this process precludes the ability to examining the prototypes in the market earlier, hampering real innovation in the future. Therefore, NTT is seeking a solution efficient for both quality and agility [5].

Section 2 defines the processes underlying iterative software development. Section 3 discusses artifacts and metrics appropriate for software quality evaluation including maturity. Section 4 details our experiments and their results, with quality metrics, quality control chart and reliability growth curve. In section 5, we conclude our paper by mentioning future work.

II. Related Work

Some former studies [6-11] concerned about software quality in agile and highly-iterative development. Mnkandla et al. introduced an innovative technique for evaluating agile methodologies and determined which factors of software quality were improved [6]. Kunz et al. described a quality model, distinct metrics and their implementation into a measurement tool for quality management [7]. Olaqgue et al. discussed the fault-proneness of object-oriented classes of highly iterative processes [8]. Roden et al. performed empirical studies and examined several Bansiya and Davis quality factor model [9]. A recent study [10] focuses on the metrics and maturity of iterative development. These studies provided the new evaluation viewpoint of software quality and the advantages of agile methodologies were shown in their experiment. However, their results can not be compared with past products measured by traditional metrics. At NTT, the inability to compare has been been an obstacle for transformation from traditional methodology to a highly iterative methodology.

III. Quality Evaluation Process Definition

Fig.1 defines general process for five development types:

- completely sequential/waterfall development (Type A-1)
- sequential/spiral development (Type A-2)
- hybrid sequential development including integration testing (Type B-1)
- hybrid sequential development including specification and integration testing (Type B-2)
- hybrid development with highly iterative process (Type B-3)
- highly iterative development (Type C).

All development types generally have seven development processes: requirement definition/evolution, specification, design, construction, unit testing, integration testing and system testing. Type A, B and C generally stand for sequential, hybrid and highly-iterative development, respectively.

All development types except completely sequential has iterations. We define the processes in the iteration as "value production process" or especially in Agile "user story process". Each small user requirement is realized as code in the value production process. The value production process can be defined as *a process unit that provides a certain value for users*. The value production process has further two types: "complete" and "incomplete". The former type has complete small sequential development process except requirement definition/evolution, while the latter does not. In addition, user requirements are defined once and appropriately evolved in hybrid development with highly iterative process.

In this paper, we call simply Type A as sequential development, while Type B and C as iterative development.

A. Development type and characteristics

Table 1 shows the definition of each development type and main characteristic from the viewpoint of user interaction:

- The team can always get code for release.
- The team/customer can evolve requirement.
- The team/customer can confirm functionalities and obtain feedback by product demonstration.
- The team can confirm success of integration.

The first two characteristics can be seen in the iterative development, while the latter two one can be seen in almost all the development type. Type A-1 and A-2 represents Waterfall, while Type B-3 and C represent Agile. Hybrid Type-B1 and Type-B2 are sometimes called "Water-scrum-fall". Water-scrum-fall stands for combination of Waterfall and Scrum. Water-scrum-fall especially starts and ends with waterfall process such as requirement and system testing, while the middle process incorporating iterations.

B. Value production process/user story process

Fig. 2 describes the detail of what we refer to as the value production process. The input of the process is value definition and acceptance criteria. Value definition is called as "user story" in Agile. The output of the process is code and testing result. The code is the output of value definition, while the testing result is that of acceptance criteria.

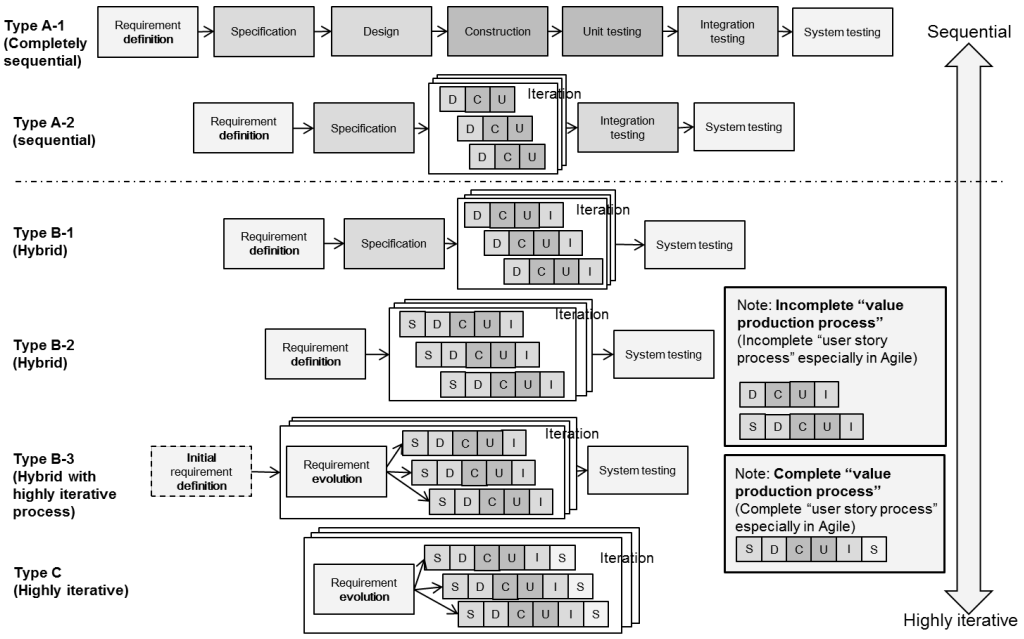

Fig.1 General process definition

Table 1 Characteristics of development process type

Index	Definition of development type	Typical development	Characteristics from a viewpoint of interaction by "users"			
			Always get code for release	Evolve requirement	Confirm functionalities by demonstration	Confirm success of integration
Type A-1	Completely sequential development	Waterfall	---	---	---	---
Type A-2	Sequential development	Waterfall	---	---	---	(X)
Type B-1	Hybrid development	Water-scrum-fall	---	---	X	X
Type B-2	Hybrid development	Water-scrum-fall	---	---	X	X
Type B-3	Hybrid development including highly iterative process	Agile	---	X	X	X
Type C	Highly iterative development	Agile	X	X	X	X

x : satisfying the characteristic, (x): maybe satisfying the characteristic, -: not satisfying the characteristic

C. *Three elements for defining outline of development*

Table 2 shows the three important elements used in defining processes: release term, iteration term and iterative process.

The release term identifies the period of development. The iteration term is a "timebox" for small development and contains itself a value production process. Especially in Agile methodology, the iteration term is called a "sprint," which includes sprint planning at the beginning of iteration and demonstration and retrospective at the end of user story process. The iterative process has to be determined to define the development type. These three elements determine the general outline of development.

D. *Activities for estimating regression testing cost*

Generally, testing cost are calculated as the sum of time of creating test cases, such as functional, performance, usability, stress, security, integration, system and so on, and the time of running them. In highly-iterative development, a large amount of regression tests are required to guarantee the quality, so we focused on the regression testing cost. Table 3 indicates the three elements that determine the cost of regression testing: integration frequency, regression testing frequency and the amount of regression tests. These three elements impact the testing cost directly. Some projects achieve integration tests and all regression tests after every build if most of the tests are automated. Some projects can achieve only one set of integration tests and few regression tests in iteration due to the need to perform non-functional testing manually

Therefore, we can outline the development and estimate the testing cost by those six elements.

IV. SOFTWARE QUALITY IN DEVELOPMENT WITH ITERATIVE PROCESS

We propose a quality evaluation scheme for the iterative software development. The development type/process determines the means of getting valid quality artifacts to calculate metrics in these iterative processes for total quality evaluation. In sequential development, the functionality is almost stable when development and unit tests are completed. However, in highly-iterative development, the functionality is always changing until the end of the last iteration. We have to get valid and useful artifacts to evaluate metrics for software quality. In addition, regression testing also has to be considered, because the purpose of regression testing is significantly different between sequential and highly-iterative development.

A. Artifacts for quality evaluation

Fig. 3 outlines the quality evaluation for the iterative development. The artifacts necessary for software quality evaluation are code and test cases. For quality evaluation in iterative development, we must also collect defect reports traceable to the extracted location and extracted time and the test cases related to the executed time. These artifacts can be acquired in the same way as with traditional developing method. For example, each bug is recorded in association with the enclosing software code, such as "class". The artifacts are acquired from all value production processes, and then they are summed by a process, such as unit testing

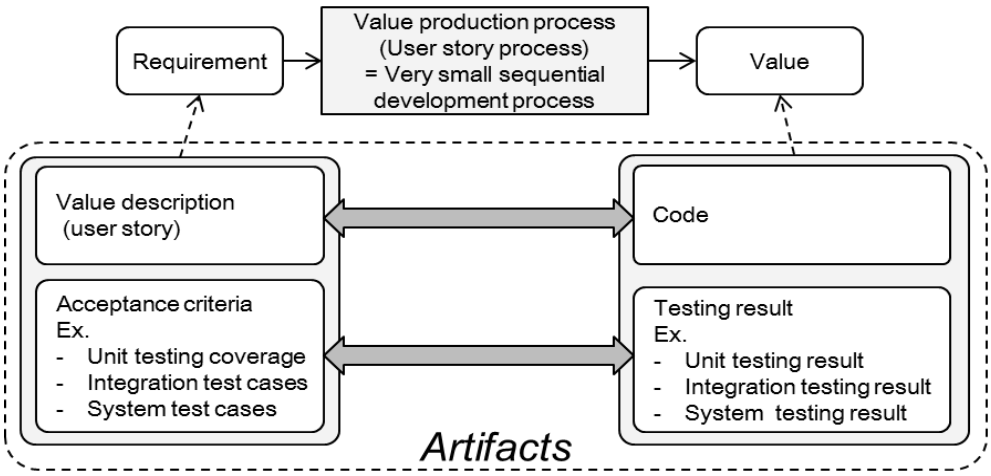

Fig. 2 Value production process (User story process)

Table 2 Three important elements for general process definition

	Release term	Iteration term	Iterative process
Explanation	To decide term of software development	To decide timing of retrospectives and demo	To define general process
Examples	- Every iteration - Every N iterations	- N weeks - M days	- All process - All process without requirement definition and system testing

Table 3 Three activities for determining regression testing

	Integration frequency	Regression testing frequency	Amount of regression tests
Explanation	To decide timing of integration	To decide timing of regression testing	To decide the amount of regression testing
Examples	- Continuous integration every building code in an iteration - Several integration in an iteration -Single integration in an iteration	-Every building code -Several times in an iteration -Once in a sprint	- All - Selective (Viewpoints are required)

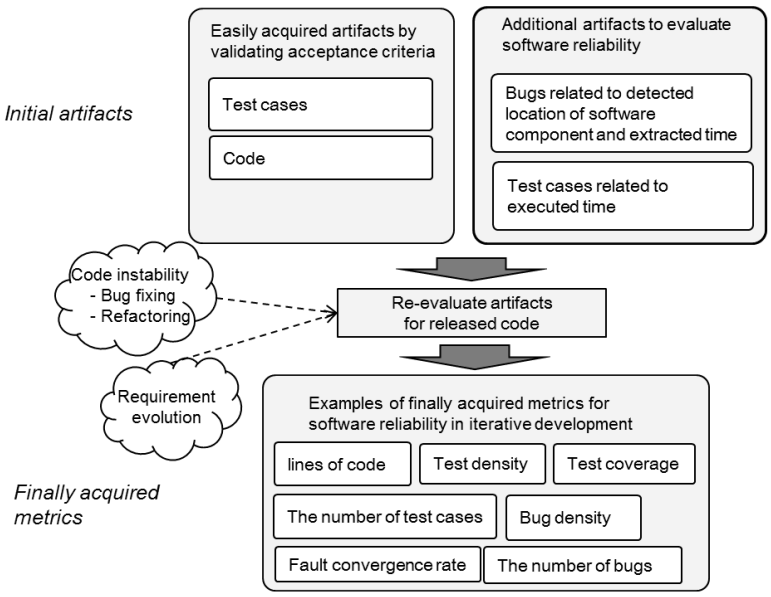

Fig. 3 Quality metrics acquired scheme in iterative process

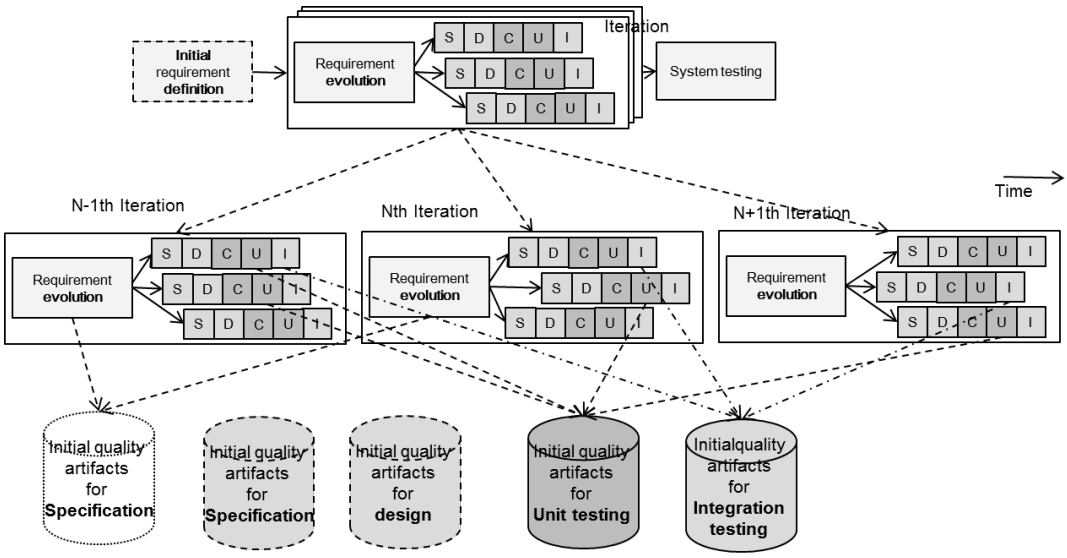

Fig. 4 Summarization for metrics

and integration testing in the way described in Fig. 4. If the artifacts of specification or design process are also required to evaluate quality, they can be accumulated and summed in the same way.

Now, we focus on regression testing, which is applied for bug fixing, requirement evolution, and refactoring. There is a significant difference in the purpose of regression testing between sequential and highly-iterative development. In sequential development, the code is basically stable after integration. In this case, each integration test (or functional test) can achieve two purposes, functional confirmation and non-impact on other code at the same time. Therefore, in this case, the number of bugs is counted and the time used in regression testing especially bug fixing in quality evaluation.

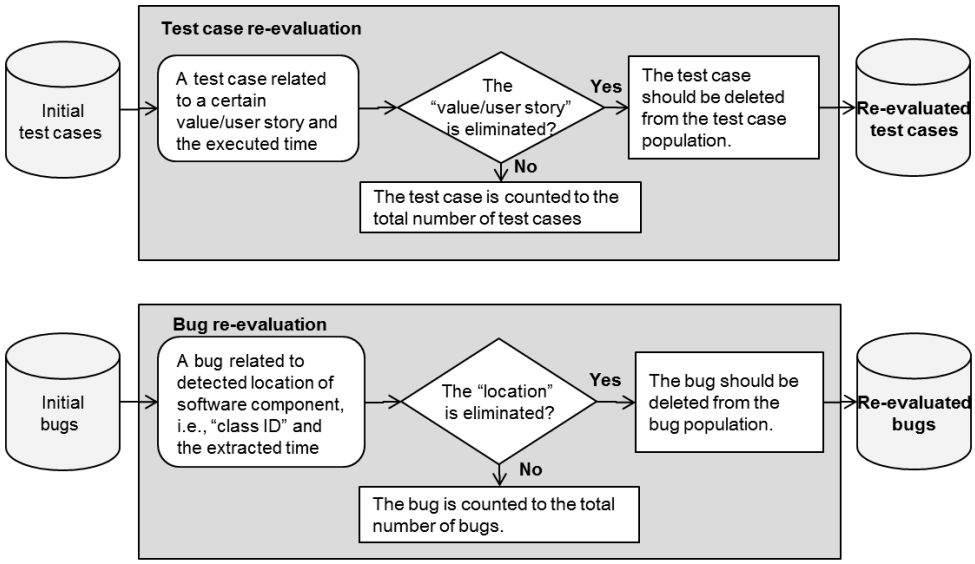

Fig. 5 Code stability based artifacts re-evaluation

Table 4 Typical quality metrics for testing process

No.	Testing process	lines of code	Test coverage (C0,C1)	The number of testing cases	Test density	The number of bugs	Bug density	Bug convergence rate
1	Unit testing	X	X	X	X	X	---	---
2	Integration testing	X	---	X	X	X	X	X
3	System testing	X	---	X	X	X	X	X

On the other hand, in highly-iterative development, code is not stable during iteration, so regression testing should be executed appropriately to confirm the non-impact on other code. A certain test case can be executed over and over during the iteration (particularly with automated testing), but the confirmation viewpoint is always different. In this case, the number of bugs is counted in the same way as with traditional methodology, and the time executed regression tests also should be considered. Therefore, the all test cases executed including regression testing are recorded with the execution time even if the amount of time consuming in regression testing is huge in comparison to the sequential development. In general, highly-iterative agile methodologies do not prescribe this defect data collection.

B. Quality artifacts re-evaluation

The quality artifacts initially acquired, the defect reports and the test cases have to be re-evaluated because of the software instability inherent in highly-iterative development. Fig. 5 shows an algorithm for quality artifacts re-evaluation. If a defect belongs to code or functionality that is dropped, the bug should not be counted, the associated test cases should be deleted. The test cases related to value/user stories eliminated by requirement evolution or refactoring should not be counted. After quality artifacts re-evaluation, we can calculate the quality metrics using these quality artifacts.

The cost of acquiring additional artifacts and re-evaluating these artifacts cannot be ignored for the practical purpose, so some means for automatically recoding artifacts may be required. Time-related artifacts are essential to evaluate the maturity of the software, while re-evaluation of artifacts can be skipped, if the code is apparently stable. We can acquire "approximate" quality software if quality artifacts are not re-evaluated.

C. Maturity for iterative development

After re-evaluating the quality artifacts, we can acquire the quality metrics and observe the maturity of the software. Table 4 shows the typical quality metrics used for testing in highly-iterative development. They are almost the same as those in sequential development. This similarity helps the transformation from traditional development to highly-iterative, because the same quality measurement is available to compare to the quality of previously-developed software.

The software reliability, especially maturity, also can be obtained through the temporal trends of re-evaluated artifacts. For example, software reliability can be described by the reliability growth model, such as the Gomperz model as is done with sequential development [12].

Table 5 Project profiles

	Example A	Example B	Example C
New/improve	Improvement	Improvement	Improvement
Lines of code	53 klines (Newly developed code: 16klines)	58 klines (Newly developed code: 7klines)	62.7 klines (Newly developed ode: 5.6klines)
Development type	Type B-1	Type B-2	Type B-2
Release term	3 months	4 months	6 months
Iteration term	2 to 3 weeks	2 to 7 weeks	3 to 5 weeks
Iterative process	Construction/unit testing/integration testing	Design/construction/unit testing/integration testing	Design/construction/unit testing/integration testing
Sequential process	Requirement/design/system testing	Requirement/system testing	Requirement/system testing
The number of iteration	3	3	3
Integration frequency	Every code building (Continuous integration)	Every code building (Continuous integration)	Every code building (Continuous integration)
Regression frequency	Every code building	Once in an iteration	Once at the end of all iterations
Amount of regression tests	All	All	Selective
Test automation in iterative process	Automated	Automated	Automated

Table 6 Quality metrics

No.	Process	Quality metrics	Sample 1	Sample 2	Sample 3
1	Unit testing	C1 Coverage [%]	89.5	89.8	93.6
2		The number of test cases	1,402	3,030	3,372
3		The number of classes tested	175	175	343
4		Test density per class	8.0	9.5	9.8
5	Integration testing	The number of bugs	35	35	29
6		The number of test cases	472	1,704	1845
7	(Regression testing)	The number of bugs	1	0	0
8		The number of test cases	5,418	12,585	8,862
9	System testing	The number of bugs	0	0	1
10		The number of test cases	5	6	6

V. EXPERIMENTS, RESULTS AND DISCUSSION

A. *Sample project profile*

Three sample projects are experimentally examined. The sample software has been improved though the three projects. The software is a simple data converter. The software consists of five components: input, core processing, output production, logic control and user interface (UI).

Table 5 details the three project profiles. All projects are the hybrid development with iterations (Types B-1 and B-2). Example A achieves some improvement but it contains code refactoring, so it can be regarded as scratch development. On the other hand, example B and C are typical improvement, which code includes the code produced in the former projects. Software architecture had already fixed before the first project began, and it has never changed through the three projects. The requirements or user stories were not changed at all while developing in the both projects. The major difference among these projects is the frequency and the amount of regression testing. Example A has a regression testing in every build, while example B and C has it at the end of each iteration. Example A and B utilizes the all test cases as regression testing, while example C partially selects the regression test cases. Therefore, example A has the most cost in regression testing, while example C has the least cost.

B. *Experimentally-acquired metrics*

Table 6 shows the quality metrics of example A, B and C. The metrics are re-evaluated after the project finished. No bugs of eliminated classes are counted in integration testing.

Example A has much more regression tests than original integration tests, because regression testing is carried out after every build.

C. *Temporal trend in integration testing*

These examples are hybrid type and they have iterative process. We focus on the iterative process, the integration testing, and evaluate them in this section.

Fig. 6, 7 and 8 plot the daily trend in integration testing process units for sample A, B and C, respectively. The X-axis shows date, while the y-axis plots temporally accumulated number of bugs and the number of executed test cases. The black bars indicate the regression tests, while the white bars describe the original integration tests.

D. *Fault convergence*

Fig. 9, 10 and 11 show the trend in bug number (running total) and an approximation of a reliability growth model, i.e. Gompertz model, in integration testing for examples A, B and C, respectively. The X-axis shows the accumulated number of test cases that have been executed, while the y-axis plots bug number (running total). Most of the test cases are automatically executed and each testing time is the almost the same. This automation allows us to regard that the accumulated number of executed test cases represents the total testing time at the current point. Actually, they are not equal but proportional. We provide two graphs in Fig. 9 and three graphs in Fig. 10 and 11. Fig. 9-graph1, 10-graph1 and 11-graph1 show only integration testing result except regression testing, while Fig.9-graph2, 10-graph2 and 11-graph2 describe integration testing result including regression testing for the current development project except regression testing for diverted code. Fig. 10-graph3 and 11-graph3 consider all tests including regression tests for both newly added code in the current project but also diverted code in the past projects. .

In example A of Fig.9, the fault convergence rate in the graph2 is 99.4% if all integration tests and regression tests in the current project are considered. Actually, no bug related to example A was present until example B was completed. The convergence rate is 81.9% in graph1, which is worse than the graph2.

In example B of Fig.10, the same characteristics can be seen in the graphs1 and 2. The convergence rate is 99.9 % in graph2 if the current development approach employs regression tests. The convergence rate is 95.1 % in the graph1, which is worse than the graph2. Moreover, the convergence rate is 99.4% in graph3 if all tests including all regression tests are considered.

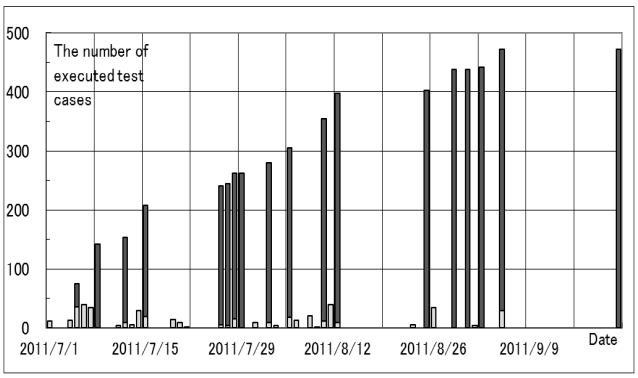

Fig.6 Temporal trend of accumulated test cases

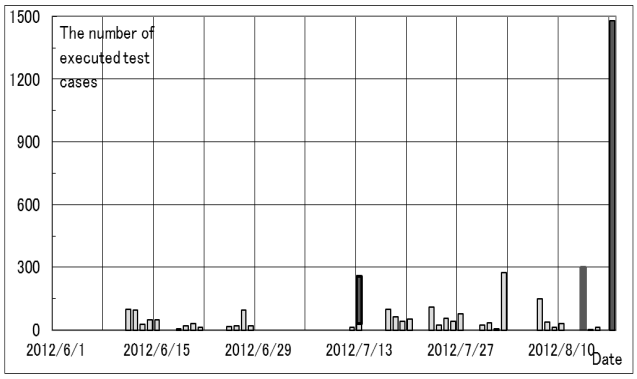

Fig.7 Temporal trend of accumulated test

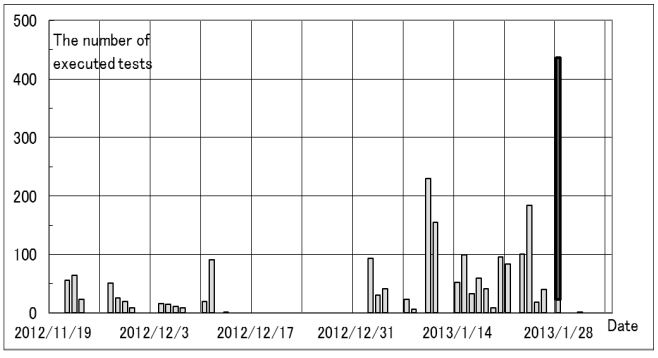

Fig.8 Temporal trend of accumulated test cases for

In example C of Fig.11, the same characteristics can be seen. The fault convergence rate is 67.5%, 90.4%, and 87.8%, respectively. The graph2 scores highest.

Although, example B and C has much fewer regression tests than example A from the viewpoint of regression testing, these three graphs of each example have similar tendency in terms of their fault convergence rate. Especially in example B and C, the graph2 written by the current project's test result only achieved the best fault convergence rate up to around 99%. The graph3 of them apparently shows the temporal trend of extracted bugs does not fit the reliability growth model, so regression tests for diverted code produced in the former projects might be ignored, because the code is stable in the current project and have already tested in the past when maturity of the "current developing code" is examined. The huge number of test cases of diverted code might bother the fault convergence estimation of currently developing code. Therefore, regression testing is a key issue in describing software maturity, so we need more examples to find appropriate condition of regression testing.

E. Consideration and Limitations

The results of the experiments show that iteratively developed software can be evaluated by the same quality metrics used in sequential software development. That means there is possibility to evaluate quality using the same quality metrics and it can be the one of driving force to transform traditional development to new development suitable for rapid response to the agile market needs. The experimental result also brings us the knowledge that the total number of test cases reaches several times as the original number of test cases and the effective bugs are extracted early through the project. That means the huge number of test cases are executed than sequential development and it is effective to find and fix bugs earlier. However, a couple of things must be taken into consideration:

- The number of iteration and amount of regression tests

Example projects have only three iterations. According to the experimental result, a certain amount of regression tests are essential, but all regression tests should not always be considered to fit the reliability curve even if the short project. The amount of test cases for regression is increasing gradually as project is advancing. Paying attention to the curve of the extracted bug trend, it creates the small group to fit a certain reliability growth curve by iteration. Therefore, we can have new hypothesis that only regression tests for the current iteration might be considered and software reliability in the current iteration might be evaluated iteration by iteration. Long-term project examples are required.

- Characteristics of the example software

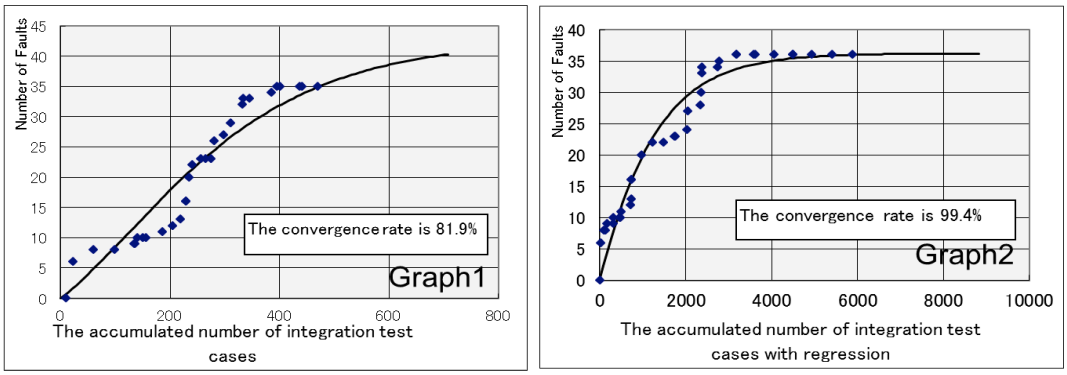

Fig. 9 Fault convergence for example A

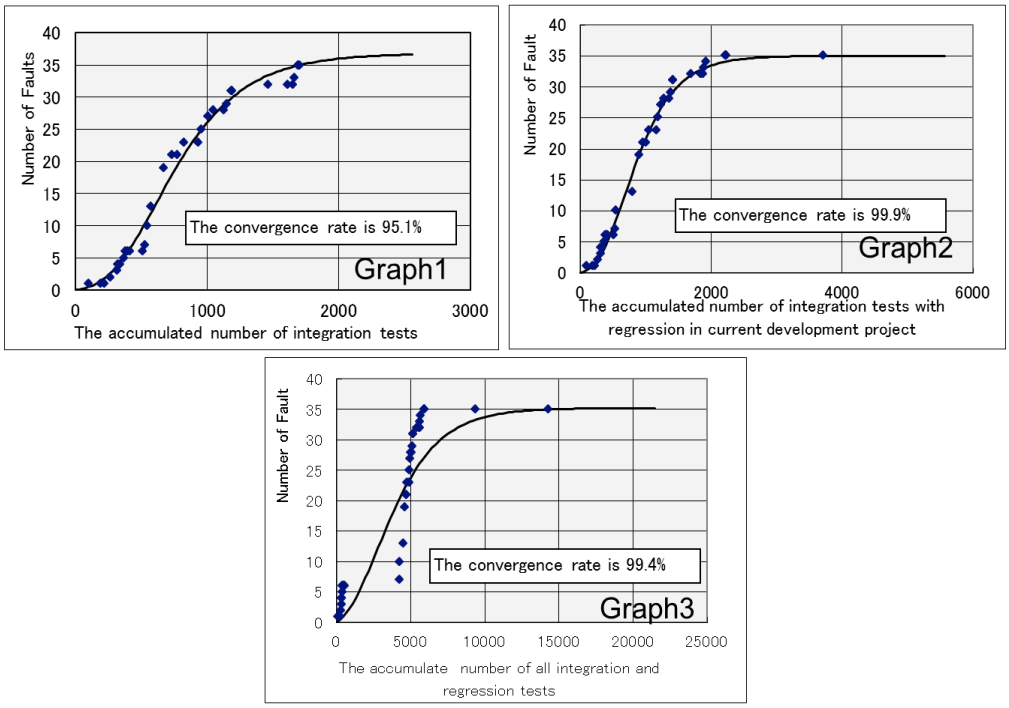

Fig. 10 Fault convergence for example B

The example software's architecture, code and also requirements were very stable during the iteration. This is obviously different from typical example of highly iterative development such as Agile, which has a lot of chance for requirement evolution and code refactoring any time. In this case, several types of project should be examined.

- The extra work for obtaining and re-evaluating artifacts

The programmers have to record all artifacts, especially bugs and test cases related to time, component or class. This extra load must be traded off against agility. One solution is systematic approach to reduce workload. Another solution is very different. We need to return to the hypothesis written at the beginning of this paper. If the development process was defined and the volume and timing of tests including regression was appropriately determined at first, quality might be estimated without obtaining quality artifacts and metrics. That is the real goal for our study.

VI. Conclusion and Future Work

In this paper, we discussed software quality based on our hypothesis that iterative development can realize higher maturity than sequential development, if the testing is done enough and appropriately. To confirm this idea, we proposed the traditional-metrics-based quality evaluation scheme and evaluated the metrics of three sample projects to prove it experimentally and detect the factors and conditions. In the future, we have to examine and analyze several types of examples to prove and evolve our hypothesis and finally to reach the goal that we can describe

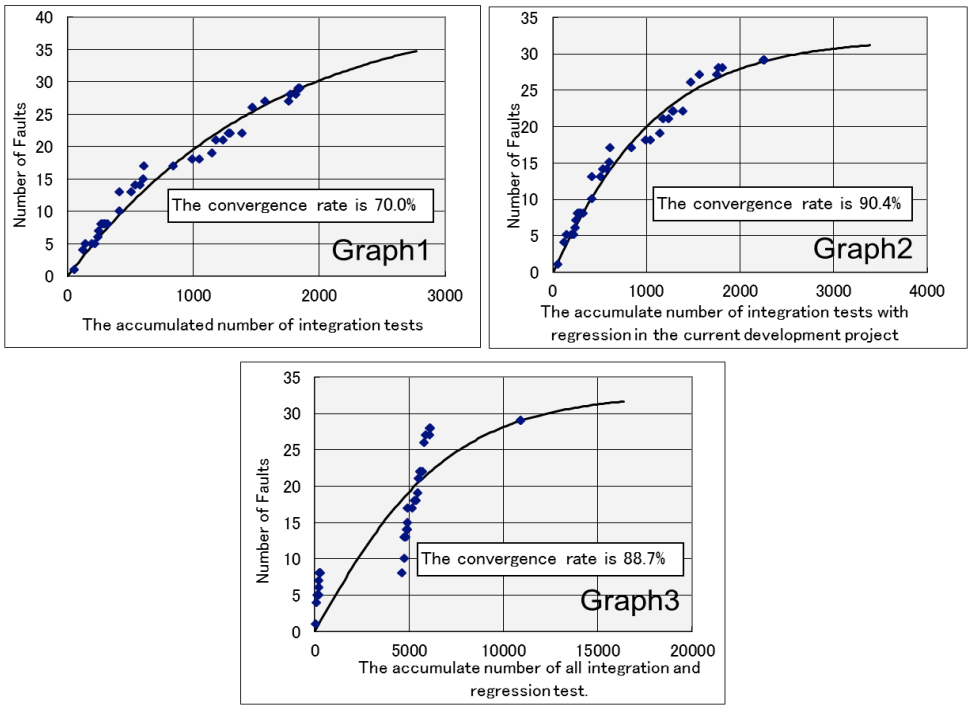

Fig. 11 Fault convergence for example C

the quality of software iteratively developed by simply defining process structure and determining test volume and timing without recourse to traditional metrics.

ACKNOWLEDGMENT

We wish to thank Mr. H. Tanno and Ms X. Zhuang for providing the sample code, and Ms. T. Tanaka, Mr. H. Andoh and Ms. C. Takahashi for organizing and visualizing the data, and all colleagues of NTT Software Innovation Center for giving us this chance to conduct our studies.

REFERENCES

[1] Kent Beck, et al., "Manifest for Agile Software Development".

[2] Poppendieck, M and Poppendieck T., "Lean Software Development: An Agile Toolkit", Addison Wesley, 2003.

[3] W. Royce, "managing the Development of Large Software Systems," Proceedings of IEEE WESCON 26, 1970.

[4] ISO/IEC TR 9126-2 "Software engineering – Product quality – Part2 External metrics"

[5] K.Jinzenji, T. Hoshino, L. Williams, K. Takahashi, "Metric-based quality evaluations for iterative software development approaches like Agile," IEEE International Symposium on Software Reliability Engineering 2012 (Industrial Track).

[6] E. Mnkandla, B. Dwolatzky, "Defining Agile Software Quality Assurance", Processdings of the International Conference on Software Engineering Advaces, 2006.

[7] M. Kunz, R. Dunke, N. Zenker, "Software Metrics for Agile Sftware Development", pp.673-678, the19th Australian Conference on Software Engneering, 2008.

[8] H. Olague, L. Etzkorn, A. Gholston, S. Quttlebaum, "Emprical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes", Vol.33 No.6, IEEE Transaction on Software Engineering, June 2007.

[9] P. Roden, S. Virani, L. Etzkorn, S. Messeimer, "An Empirical Study of the Relationship of Stability Metrics and QMOOD Auality Models Over Software Development Using Highly Iterative or Agile Software Processes", pp.171-179, Seventh IEEE International Working Conference on Source Code Analysis and Manipulation, 2007.

[10] S. Jeon, M. Han, E. Lee, K. Lee, "Quality Attribute driven Agile Development", pp.203-210, Nineth International on Software Engineering, Management and Applications, 2011.

[11] T. Fujii, T, Dohi and T, Fujiwara, "Towards quantitative software reliability assessment in incremental development processes," pp.41-50, Proceedings of ICSE 2011.

[12] Sakata, K. (1974), Formulation for predictive methods in software production control – static prediction and failure rate transition model – (in Japanese), Trans. On IECE of Japan, 57-D, 277–283.