A NOVEL METHOD FOR REDUCING TESTING TIME IN SCRUM AGILE PROCESS

Dr. Kiran Kumar Jogu[1], Dr. K. Narendar Reddy[2]

[1]IBM India Software Lab, Hyderabad, India
[2]CVR College of Engineering, Dept. of CSE, Hyderabad, India

ABSTRACT

Recently, the software development in the industry is moving towards agile due to the advantages provided by the agile development process. Main advantages of agile software development process are: delivering high quality software in shorter intervals and embracing change. Testing is a vital activity for delivering a high quality software product. Often testing accounts for more project effort and time than any other software development activities. Testing strategies for conventional process models are well established, but these strategies are not directly applicable to agile testing without modifications and changes. In this paper, a novel method for agile testing in the scrum software development environment is proposed and presented. The sprint and testing activities which form the context for the proposed testing method are presented. The proposed method is applied on two cases studies. The results indicated that the testing time can be reduced considerably by applying the proposed method.

KEYWORDS

Agile software development, scrum, software industry, novel method for testing, testing time reduction.

1. INTRODUCTION

Traditional software development process models are being used for long time in software development. Present business demands the software products to be delivered in shorter intervals and software development environment having capability to embrace change at any stage of development. Traditional process models have difficulty in responding to change which often contributes success or failure of a software product [1]. Software requirements are dynamic which are driven by industry market forces. Agile approach to software development is suitable to such situations [2], [3]. Hence, more software companies are making a transition to agile software process models from traditional software development process models. Some of the key factors for success in an agile testing approach are: adopting an agile mindset, automating regression tests, collaborating and obtaining feedback from customer [4]. Some issues may arise when transition is made from traditional development to agile development. Common issues for agile models after migration from traditional models were identified in [5]. They are related to testing, test coverage, coordination overhead, and software release. In this paper we focused on testing related issues. Agile methods employ short iterative cycles, with prioritizing the requirements which actively involve users. Agile process models are iterative, incremental, self organizing and emergent [6]. One of the agile process models which is being used in the software industry is "scrum". Scrum agile process model is defined in [7], [8]. In agile software development, testing is a vital activity for delivering a high quality software product to the customers. Often testing accounts for more project effort and time than any other software development activities. Since testing plays a major role in the success of the product, it is given a lot of importance in software

DOI : 10.5121/ijsea.2016.7503

development. Testing strategies for conventional process models are well established, but these strategies are not directly applicable to agile testing without modifications and changes. One of the important current research areas is the agile software testing strategies. The main objective for any agile testing strategy is to reduce the testing time and at the same time ensuring the software quality. In this paper, a novel method for agile testing in the scrum software development environment is proposed and applied on two case studies. The results indicate that by applying this method testing time can be reduced.

The remainder of this article is structured as follows. Related work is briefly described in Section 2. In Section 3, the proposed testing method is described. In Section 4, case studies are presented. Subsequently, conclusions are presented and future directions are proposed.

2. RELATED WORK

Software industry is transitioning to agile methodologies from traditional approaches. One of the popular agile process models which is being used in software companies is "scrum". Scrum main characteristic is, continuous deployment of working product increment after each sprint. As per the survey on agile methods given in [9], 54% of the software companies who are using agile methods are using Scrum. In the survey conducted by [10] on agile projects in different countries found that six critical factors contribute to agile project success. These factors are: agile software engineering techniques, customer involvement, project management process, team environment, team capability, and delivery strategy. One of the attributes related to the critical factor "agile software engineering techniques" is testing strategies. To address the above mentioned critical factor and its associated attribute, currently research is being carried out on agile testing strategies [11], [12]. In this direction, authors of this paper proposed a novel testing method for scrum agile software development environment.

3. AGILE SOFTWARE DEVELOPMENT USING SCRUM

To provide consumers with continuous deployment of new features rapidly with the capability of embracing change at any stage of development, scrum is ideally suited for this purpose [7], [8], [13]. The scrum agile model is an iterative, incremental process of planning, development, testing, and deployment. In scrum at the end of each sprint a working increment is released and deployed. In XP (eXtreme Programming) at the end of an iteration, the working product may not be available. Hence, scrum leads to continuous deployment when compared to XP. Due to scrum's main characteristic of continuous deployment, software industry is transitioning to scrum agile software development. The scrum model is depicted in the Fig. 1 which is adopted from [7]. The model shown in Fig. 1, is depicting the artifacts of their underlying activities. The main framework activities of the agile process model are: Creation of product backlog, Planning (Creation of sprint backlog and expanding the sprint backlog), and Sprint (consists of development activities). The scrum activities are performed by the scrum team which consists of product owner, development team, and scrum master. Product owner is responsible for creating and maintaining the requirements in product backlog. He/she creates stories for the requirements in the product backlog. Development team is responsible for developing the product by implementing the features in sprint backlog. The development team is cross functional. Cross functional means, team is responsible for design, development, testing, and deployment. The responsibility of the scrum master is to ensure that the scrum process is followed properly by the team.

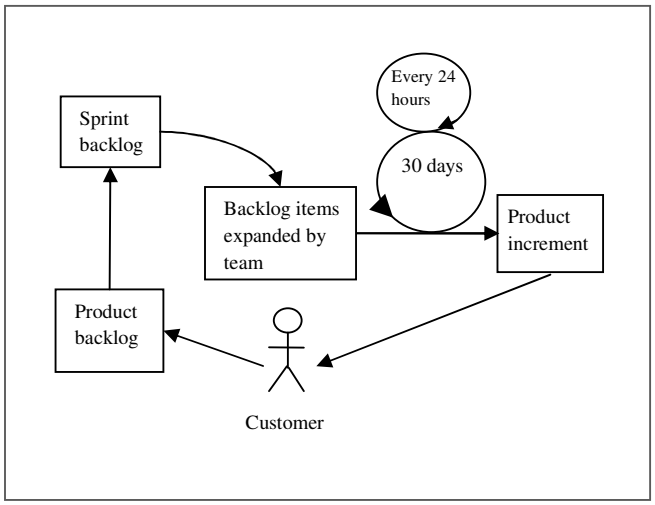

Figure 1. Scrum agile process model

The scrum activities lead to the following artifacts: product backlog, sprint backlog, task list to achieve sprint backlog, and working software product increment respectively. These artifacts are briefly discussed below.

Product backlog: The required product features or requirements identified by customer are added to product backlog. Features are prioritized as desired by the customer. The main source of agility in scrum model is the prioritized requirements list, which is flexible product backlog [8], [14]. Changes are inevitable. As the needs of the customers change the product backlog is continuously reprioritized. Hence, the software development is flexible. New features are selected from the backlog continuously and integrated and released as a working product increment at the end of the sprint. This means that one can deliver with increasing functionality more frequently, which provides flexibility and the opportunity for adaptive planning [8].

Sprint backlog: During first part of planning, product owner and development team together decides which features (user stories) will be part of the next sprint. The high priority features from product backlog are given preference. These features in this backlog are addressed during the sprint. Typical time-box for a sprint is 30 days. The changes (addition of new features) to the features in the ongoing sprint will not be accepted. But, changes (new features) can be added to the product backlog while the sprint is in progress.

Expanded sprint backlog: During second part of planning, development team analyses the user stories (features) in the sprint backlog and divides each user story in its tasks. These tasks are handled by different development team members during sprint.

Working software product increment: During sprint, development activities are carried out iteratively. Scrum meetings are held daily, typically of 15 minutes duration. Team discusses about the progress and what to be done in next 24 hours. At the end of sprint (30 days), working software product increment is delivered (deployed). Delivered product is evaluated by the customer to ensure that the features in the sprint backlog are implemented. Testing is important, because it is carried out to ensure the software product quality. Testing consumes most of the time during the sprint. Any reduction in testing time will help to deliver the product increment in the given time box of the sprint. Hence, the authors of this paper focused on a novel method to reduce the testing time. Testing activities and proposed testing method for scrum model are given in the following section.

3.1 Testing Activities in Scrum

Scrum is a framework for developing software products [15]. Various processes and techniques can be proposed and employed within the framework. Scrum framework specifies the following activities: planning, (Creation of sprint backlog and expanding the sprint backlog), Sprint (consists of activities which can deliver a working software product increment implementing sprint backlog features in a given time-box(typically 30 days)). To propose a method for testing, first the sprint activities need to be considered and identified. One of the possible set of sprint activities can be eXtreme programming (XP) type development activities. The XP development activities could be: design, test driven development and refactoring, integration and regression testing, and validation testing before release. XP activities may not produce a working product increment after completing iteration(s) (in a given time-box). This may be because of the fact that this model is not based on predefined time-box based product release, hence the authors of this paper considered sprint activities which can deliver the working software product in predefined time-box. Sprint activities that are considered, are shown in Fig. 2. The activities are: design, development (coding), and testing. They are performed iteratively to produce a working product increment in a given time-box (sprint). The sprint activities are carried out iteratively to implement the features (user stories) in sprint backlog. The team for sprint contains scrum master and development team. Development team is cross-functional. They will be able to perform design, coding, and testing (unit testing and integration testing). Some of the development team members (testers) can be specifically meant for regression and functional testing. The responsibilities of the testers are: to plan and update test cases for sprint stories, automate test scripts if possible, execute the tests and report defects, and run regression tests and functional tests at the end of the sprint. Testers are also responsible for testing non-functional tests such as load testing and performance testing.

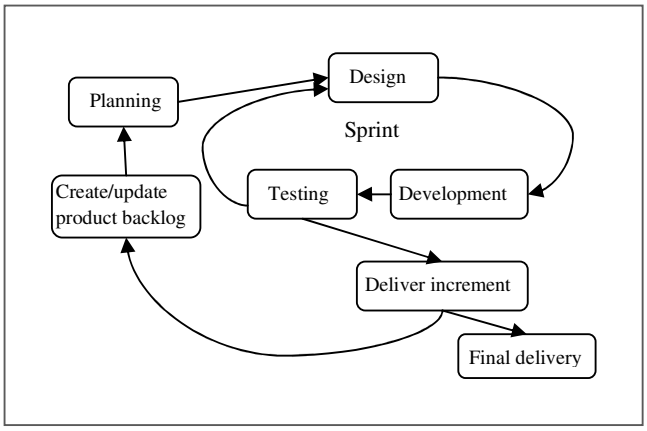

Figure 2. Sprint activities in scrum

The testers in scrum agile software development participate in scrum ceremonies which includes sprint review, planning, daily and retrospective meetings. The testing activities for scrum model are depicted in Fig. 3. Testing strategy contains: unit testing, continuous integration, and regression testing which are carried out during the sprint. Whereas, functional and non-functional testing and user acceptance testing is carried out at the end of the sprint. The testing tasks during a sprint are incremental and iterative. Unit testing is done by the developer for finding the logical errors in a module. The bugs found in unit testing are debugged before integrating with other modules. Continuous integration is performed daily. Continuous integration enables to complete the increment in the scheduled sprint time. Regression testing is done after every integration test to ensure that newly integrated module has not introduced any new bugs. Functional test cases are created based on sprint backlog stories and executed at the end of the sprint.

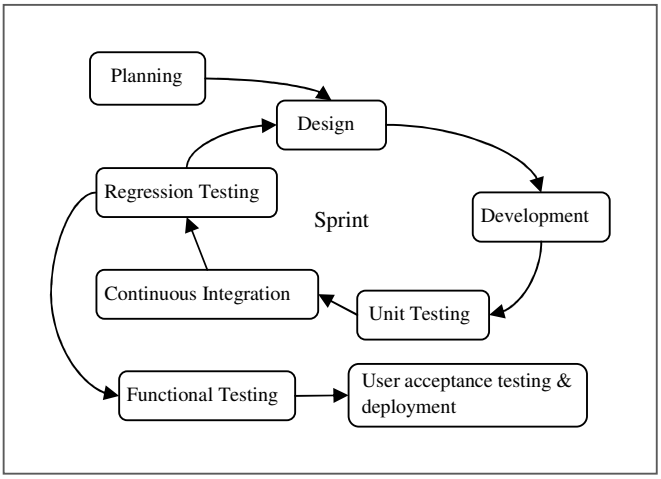

Figure 3. Testing activities in scrum

Testing activities are automated. These testing tasks are conducted repeatedly and frequently, hence, automation will help to reduce the testing time. Since these tests are conducted iteratively on small number of features they increase the likely hood of finding bugs early in the project in intermediate releases (sprints) and in turn reduces the likely hood of magnifying and propagating the bugs to the final product. Because of this fact the quality of software product is better in agile software development. During deployment the product increment is tested by the user which is known as user acceptance testing (UAT) to ensure that all the user stories specified in the sprint backlog are actually implemented. In addition to testing functional requirements, it is essential to test non-functional requirements. Some of the typical non-functional requirements are: load testing, security testing, and performance testing. Tools are used for testing non-functional requirements. These non-functional tests are executed at the end of the sprint. Software testing automation is key for the agile testing. Irrespective of agile methodology, testing automation becomes the core of agile testing [12]. The purpose of software testing automation is to automate software testing activities. Manual testing is time consuming. Manual testing is not suitable for scrum agile testing where continuous deployment is required in shorter intervals. Moreover, since testing tasks are conducted iteratively during a sprint, through testing automation testing time can be reduced considerably. Tools are available to automate all the testing activities. With automation, testing efficiency can be improved and testing time can be reduced which enables to deploy the working product increments in shorter intervals.

3.2 Novel Method for Test Suite Reduction

The proposed testing method is based on the sprint activities and testing activities given in Fig. 2 and Fig.3. The proposed novel method for testing for scrum process model is given in Fig. 4. The proposed novel method is aimed at reducing the test cases during functional testing and regression testing. The proposed method contains two phases:

Activity 1: Deriving reduced functional test suite
Activity 2: Deriving reduced regression test suite.

The proposed method is shown in Fig. 4. In Activity 1 the “Reduced Functional Test Suite” is derived and in Activity 2 the “Reduced Regression Test Suite” is derived by applying a regression

test selection method on the "Reduced Functional Test Suite" that is derived in the Activity 1. How to apply the proposed method is depicted in Fig. 5.

Activity 1: Deriving Reduced Functional Test Suite

During functional testing a large number of test cases are derived by applying various testing techniques to test complete functionality of a software product. This test suite contains test cases to test functionality, boundary values, stress, and performance of the software product. Majority of these test cases will be test cases that test the functionality and boundary values. The Activity1 of the proposed method is focused on reducing test cases considering test cases that test functionality and boundary values.

As part of Activity 1, two aspects functionality and boundary value testing are viewed together. Single test case situations are identified considering functionality and boundary values which can be tested in single test case(s) so as to design minimal test cases.

Activity 2: Deriving Reduced Regression Test Suite

Regression testing process involves selecting a subset of the test cases from the original test suite and if necessary creates some new test cases to test the modified software. In Activity 1 (Fig. 4), the "Reduced Functional Test Suite" is derived. In Activity 2, existing regression test selection technique is applied to derive the "Reduced Regression Test Suite" from the "Reduced Functional Test Suite" of Activity 1. This reduced regression test suite covers the same functionality as the original regression test suite that is derived without applying our method.

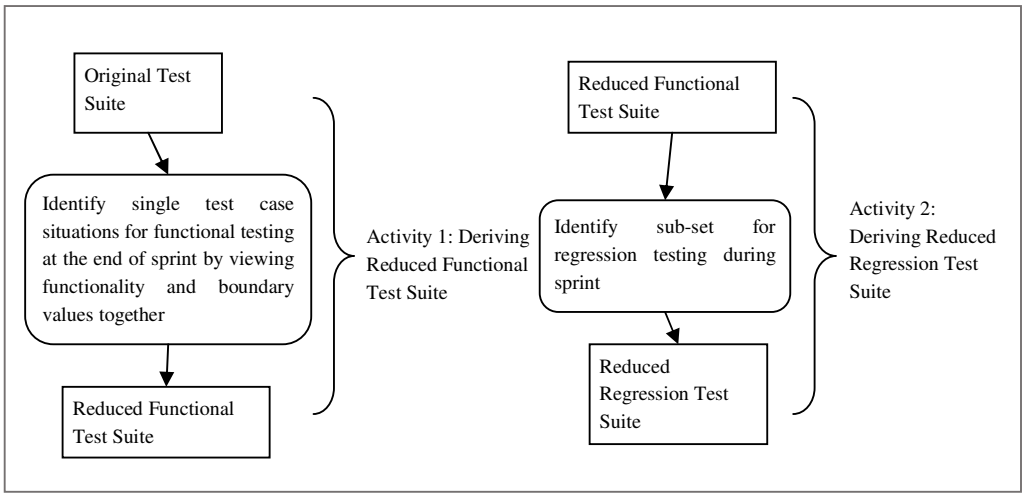

Figure 4. A novel test suite reduction method for agile process

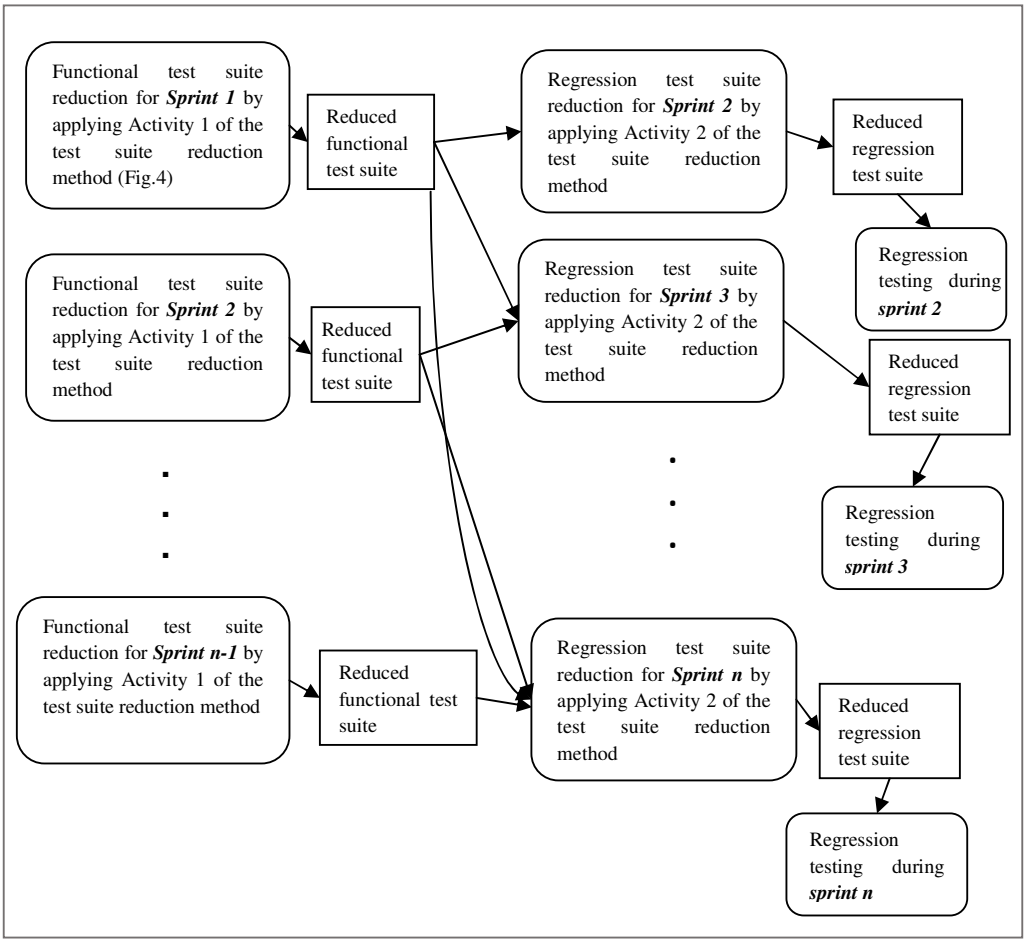

Figure 5. Applying proposed test suite reduction method during different sprints

3.2.1 Algorithm to Compute Total Minimized Test Suites

Total number of test cases that are reduced by applying the proposed method in functional testing and regression testing can be computed using the algorithm given in Fig. 6. Total number of test cases reduced in functional testing are calculated for n sprints, whereas for regression testing for n-1 sprints (Sprint number 2 to sprint n). The algorithm to compute reduced (minimized) test suites is given in Figure. 6. The algorithm contains one outer loop which iterates n times (number of sprints) and each outer loop contains an inner loop which iterates for j times (number of iterations in a sprint).

The variables used in the algorithm are explained below.

TT_{minFTS} = Total minimized test cases during functional testing after completion of n sprints.
TT_{minRTS} = Total minimized test cases during regression testing after completion of n sprints.
$T_{minRTS[i]}$ = Total minimized test cases during regression testing in i^{th} sprint.
$T_{minFTS[i]}$ = Total minimized test cases during functional testing in i^{th} sprint.
$TS_{minRTS[j]}$ = Total minimized test cases during regression testing in j^{th} iteration of i^{th} print.
TT_{nFTS} = Total number of test cases during functional testing after completion of n sprints (without using proposed approach)
TT_{nRTS} = Total number of test cases during regression testing after completion of n sprints (without using proposed approach)

T_{iFTS} = Total number of test cases during functional testing in i^{th} sprint (without using proposed approach)
T_{iRTS} = Total number of test cases during regression testing in i^{th} sprint (without using proposed approach)

n = number of sprints

m=number of iterations in a sprint

$TT_{minFTS} = 0$

$TT_{minRTS} = 0$

$T_{minRTS[i]} = 0$ (for i equals to 1 to n)

Repeat steps 1 to 3 for i = 1 to n

Step 1: If i equals to 1

Then

Compute $T_{minFTS[i]}$ using proposed method

End If

Else

Repeat step 1.1 for j=1 to m

Step 1.1: Compute $TS_{minRTS[j]}$ Using TT_{minFTS}

$T_{minRTS[i]} = T_{minRTS[i]} + TS_{minRTS[j]}$

j=j+1

Step 1.2: Compute $T_{minFTS[i]}$ using proposed method

End Else

Step 2: $TT_{minRTS} = TT_{minRTS} + T_{minRTS[i]}$

Step 3: $TT_{minFTS} = TT_{minFTS} + T_{minFTS[i]}$

i = i +1

Figure 6. Algorithm to compute total minimized test cases

The algorithm given in Fig. 6 is used to compute minimized test cases for functional testing and regression testing after every sprint. It also computes total minimized test suites for functional testing and regression testing after n sprints. The percentages of reductions in test cases are computed using Eqs. (1) and (2).

Average percentage of reduction in test cases during functional testing (after completion of n sprints) is computed using Eq.(1).

$$AvgT_{redFTS} = \frac{(TT_{nFTS} - TT_{minFTS})}{TT_{nFTS}} \times 100 \qquad (1)$$

Average percentage of reduction in regression test cases (after completion of n sprints) is computed using Eq. (2).

$$AvgT_{redRTS} = \frac{(TT_{nRTS} - TT_{minRTS})}{TT_{nRTS}} \times 100 \qquad (2)$$

4. CASE STUDIES

The proposed approach is applied on two real-world ETL tools which are being used by many customers. The ETL tools are: Teradata ETL DB Component and DB2 ETL DB Component. The final product of the Teradata ETL DB Component was delivered in four sprints and the DB2 ETL DB Component was delivered in three sprints. Sprint is of 30 days duration and after every sprint working increment is deployed.

The Fig. 7 shows the ETL process. The ETL stands for "extract, transform and load", is the set of functions combined into one tool or solution that enables companies to "extract" data from numerous databases, applications and systems, "transform" it to appropriate format, and "load" it into another databases, a data mart or a data warehouse for analysis, or send it along to another operational system to support a business process.

The Fig. 8 shows some attributes of a generalized ETL Database Component write process. In this write process, the source could be an ETL DB Component or a flat file and the target is a ETL DB Component. In the write process, the target ETL DB Component reads data from the source component, connects to the respective database using the connection properties specified and writes that data into the target table.

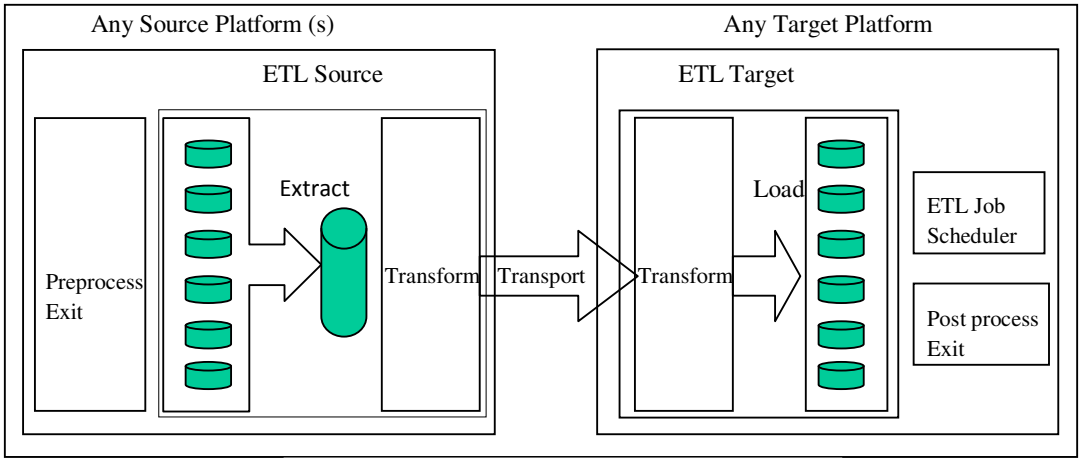

Figure 7. The ETL process

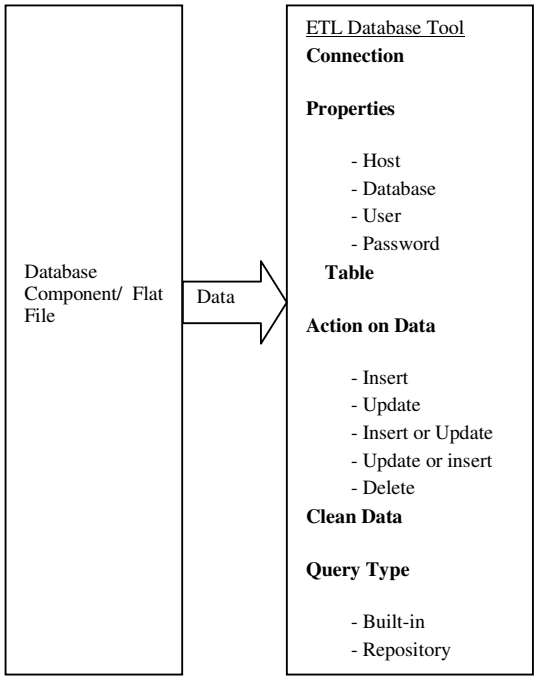

Figure 8. The ETL database component write process

4.1 Teradata ETL DB Component

This section describes the application of the proposed approach on the Teradata ETL DB Component and how the test cases are reduced using the proposed approach. The same approach is applied to all the sprints. The Fig.9 shows the metadata of the table '*sampletable'* used in the Teradata ETL DB Component case study. This is a Teradata table that contains 5 columns. The *col1* is integer type, *col2* is character type, *col3* is varchar type, *col4* is float type and *col5* is date type.

Activity 1: Deriving Reduced Functional Test Suite

The Table 1 shows some sample test cases for the Teradata ETL DB Component write process. Each of these test cases tests a single functionality or scenario of the Teradata ETL DB Component to ensure that

```
CREATE SET TABLE Sample table NO
FALL BACK, NO BEFORE JOURNAL,
NO AFTER JOURNAL,
CHECKSUM=DEFAULT
   (
   col1 INTEGER,
   col2 CHAR(9) CHARACTER SET LAT NOT
CASESPECIFIC,
   col3 VARCHAR(9) CHARACTER SETLATIN NOT
CASESPECIFIC,
   col4 FLOAT,
   col5 DATE FORMAT 'YY/MM/DD')
 PRIMARY INDEX (col1);
```

Figure 9. Meta data of the sample table

the particular attribute or function is working properly. The Table 2 shows some sample boundary value test cases for the Teradata ETL DB Component write process. Each of these test cases tests a single column or data type to ensure the boundary values of that data type are written properly to the target table. The steps involved in the activity 1 are applied to the case study (Teradata ETL DB Component) to derive reduced functional test suite.

View Two Aspects Together:

Many test cases are designed to test complete functionality of a software product. These test cases include: test cases that focused on functionality (T_f), Boundary Value test cases (T_b) , Stress test cases (T_s), Performance test cases(T_p) and other test cases (T_o) like negative test cases. The total test cases (TT_{nFTS}) is computed using Eq. (3).

$$TT_{nFTS} = T_f + T_b + T_s + T_p + T_o \qquad (3)$$

Most of the test cases in this test suite (TT_{nFTS}) belong to functional test cases and boundary value test cases. The proposed approach focused on these test cases.

Identify Single Test Case Situations:

The test case TC_{f1} tests the functionality of the Teradata ETL DB Component when the attribute 'Action on Data' is set to 'Insert' and the test case TC_{b1} tests the INTEGER data type boundary value that is written to the target Teradata table. By using the proposed approach in Phase 1, the test cases TC_{f1} and TC_{b1} are viewed together and designed a single test case TC_{m1} (Table 3) that covers both aspects (functionality and boundary values). The minimized test case set designed is shown in the Table 3.

Table 1. Functional test cases

Test Case ID	Description	Preconditions	Expected Result	Test Status	Comments
TCf_1	Test on writing the data to the target table with Action on data = Insert		The job should add new rows to the target table and stop if duplicate rows are found.		
:	:	:	:		
TCf_i	Test on writing the data to the target table with Action on data = Update		The job should make changes to existing rows in the target table with the input data.		

Table 2. Boundary Value Test Cases

Test Case ID	Description	Preconditions	Expected Result	Test Status	Comments
TCb_1	Test on writing the data to col1 with INTEGER data type boundary values		The job should read the INTEGER data type boundary values from input data and write to the target table successfully.		
:	:	:	:		
TCb_j	Test on writing the data to col2 with CHAR data type boundary values		The job should read the CHAR data type boundary values from input data and write to the target table successfully.		

Table 3. The minimized text cases designed using the proposed approach in Activity 1

Test Case ID	Description	Preconditions	Expected Result	Test Status	Comments
TCm_1	Test on writing the data to the target table with Action on data = Insert and col1 contains INTEGER data type boundary values		The job should read the input data, add new rows to the target table successfully and stop if duplicate rows are found.		
:	:	:	:	:	:
TCm_k	Test on writing the data to the target table with Action on data = Update and col2 contains CHAR data type boundary values		The job should read the input data and make changes to existing rows in the target table with the input data		

Logically Proving Single Test Case(s):

Each test case in the minimized test case set described in Table 3 will test the functionality of the Teradata ETL DB Component to ensure that the particular attribute is working properly and also tests the boundary values for various columns in the target table to ensure that the boundary values of that column data type are written properly. For example, the TC_{m1} in the minimized test case that tests whether the Teradata ETL DB Component is working properly when the attribute 'Action on Data' is set to 'Insert' and also tests whether the INTEGER data type boundary value is written to the target table properly which are tested by the two test cases TC_{f1} and TC_{b1}. Since test case TC_{m1} is able to test functionality and boundary values together, it is logically correct to combine TC_{f1} and TC_{b1} together into TC_{m1}. The test cases in the minimized test case set {TC_{m1} – TC_{mk}} described in Table 3 will test the both aspects of functionality and the boundary values of Teradata ETL DB Component, otherwise, without combining requires test cases {TC_{f1}-TC_{fi} } (Table 1) and {TC_{b1}-TC_{bj} }(Table 2)}. In similar way, the proposed approach is also applied on DB2 ETL DB Component.

Showing and Validating the Test Suite Reduction:

After applying the proposed approach in Activity 1, the percentage of test cases reduction is calculated using Equation (1). The third column of Table 4 describes the total number of functional test cases (TT_{nFTS}) before applying the proposed approach in Activity 1, the fourth column describes the total number of test cases in the minimized functional test case suite (TT_{minFTS}) and percentage of test case reduction (AvgTredFTS) after applying the proposed approach in Activity 1 is given in second column of Table 5. In both the case studies the reduced

functional test suites covered all the functionalities and boundary values with same defect coverage as that of original test suites (without applying the proposed method).

Activity 2: Deriving Reduced Regression Test Suite

In Activity 2 of the approach (Fig. 4), an existing regression test selection method is applied on the "Reduced Functional Test Suite" that is derived in Activity 1. Application of Activity 2 resulted in "Reduced Regression Test Suite". The results on two real-world case studies are recorded in Table 4. The ninth column in table 4 describes the number of regression test cases (TT_{nRTS}) that are derived by applying the existing regression test selection method (before applying the proposed method). The tenth column in Table 4 describes the "Reduced Regression Test Suite" (TT_{minRTS}) which is derived by applying the proposed approach. The percentage of regression tests that are reduced by applying the proposed approach is calculated using Equation (2). The percentage of reduction (AvgTredRTS) for various case studies is shown in the third column of the Table 5. Using the algorithm given in Fig. 6, minimized test cases for functional testing and regression testing after every sprint and after n sprints are computed and presented in Table 4. The average percentages of reductions in test cases are computed using Equations. (1) and (2) and given in Table 5.

The proposed approach is applied on the second case study "Db2 ETL DB Component" in the same way and the results are presented in the Tables 4 and 5. The results in Tables 4 and 5 indicate that the application of proposed approach on two real-world case studies has lead to considerable reduction in test cases without affecting the test coverage. The testing time is reduced proportionate to the reduction in test cases.

Table 4. Minimised test cases using proposed testing method

Case Study	Sprint #	T_{iFTS}	$T_{minFTS[i]}$	T_{iRTS}	$T_{minRTS[i]}$	TT_{nFTS}	TT_{minFTS}	TT_{nRTS}	TT_{minRTS}
Teradata ETL DB Component	1	847	644	506	506	847	644	506	506
	2	1270	965	760	586	2122	1609	1266	1092
	3	1186	902	709	546	3308	2511	1975	1638
	4	932	708	557	430	4235	3219	2532	2068
DB2 ETL DB Component	1	1102	827	576	576	1102	827	576	576
	2	1360	1020	712	566	2462	1847	1288	1142
	3	1176	883	615	491	3638	2730	1903	1633

Table 5. Average percentage of reduction in test cases

Case Study	AvgTredFTS	AvgTredRTS
Teradata ETL DB Component	24%	18.5%
DB2 ETL DB Component	25%	14.5%

The results in Tables 4 and 5 indicate that the application of proposed approach on two real-world case studies has lead to considerable reduction in test cases without affecting the test coverage. The testing time is reduced proportionate to the reduction in test cases. The average testing time reduction during functional testing at the end of every sprint is 24.5% and the average testing time reduction during regression testing of every sprint is 16.5%.

The proposed novel method for testing in scrum agile development offers following advantages.

- Method is simple.
- After every sprint functional test suite is minimized

- During sprint regression test suite is minimized.
- At the end of n sprints the average percentage of functional test suite reduction is significant.
- At the end of n sprints the average percentage of regression test suite reduction is significant.
- Reduction in test cases reduced testing time proportionately.
- Reduced testing time leads to deploying working increments quickly.

5. CONCLUSIONS

The software development in the industry is moving towards agile due to the advantages provided by the agile development process. Two main advantages of agile software development process are: delivering the high quality software to the customers in shorter intervals and having the capability of embracing the changes in requirements at any stage of software development. In majority of the situations scrum model is preferred because it delivers working software product increment in a predefined time-box (typically 30 days). Delivering a working product increment in shorter intervals (30 days) gives business advantage to the customers. Testing in agile process model plays a vital role. Testing strategies for traditional process models are well established, but these strategies are not directly applicable to agile testing without modifications and changes. A novel method for agile testing in the scrum software development environment is proposed and presented. The proposed method is applied on two case studies. Results indicate that the regression testing time is reduced by around 16.5% and functional testing time is reduced by around 24.5%. Since main goal of agile process is to deploy working increments at shorter intervals, the proposed method helps to achieve the goal by reducing the testing time.

As part of future work more number of case studies from different domains and applications need to be studied to get further insight into the research areas of agile software testing strategies and methods.

REFERENCES

[1] L.Williams and A.Cockburn,(2003) ”Agile software development: it’s about feedback and change”, IEEE Computer, 36(6), pp. 39-43.

[2] J.Highsmith and A.Cockburn, (2001) “Agile software development: The business of innovation”, IEEE Computer, 34(9), pp-120-127.

[3] C.R.Jakobsen and J.Sutherland, (2009) ”Scrum and CMMI going from good to great”, Agile Conference (AGILE), pp. 333-337.

[4] L.Crispin and J.Gregory, (2009) “Agile testing: A practical guide for testers and agile teams”, Addison-Wesley.

[5] Kai Petersen and Claes Wohlin, (2010) ”The effect of moving from a plan-driven to an incremental software development approach with agile practices: an industrial case study”, Empirical Software Engineering, 15(6), pp.654-693.

[6] B.Boehm and R.Turner, (2005) ”Management challenges to implementing agile processes in traditional development organizations”, IEEE Software, 22(5), pp.30-39.

[7] K.Schwaber and M.Beedle, (2002), Agile software development with scrum, Prentice Hall.

[8] K.Schwaber, (2004), Agile project management with scrum, Microsoft Press.

[9] VersionOne, (2013) “7th Annual State of Agile Development Survey”. Retrieved in November 2013 from http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf.

[10] T.Chow and D.Cao, (2007) ”A survey study of critical success factors in agile software projects”, The Journal of Systems and Software, pp. 961-971.

[11] Theodore D. Hellmann, Abhishek Sharma, Jennifer Ferreira, and Frank Maurer, (2012) ”Agile testing: past, present, and future – charting a systematic map of testing in agile software development”, Agile Conference (AGILE), pp. 55-63.

[12] E.Collins, A.Dias-Neto, and V.F.de Lucena, (2012) "Strategies for agile software testing automation: An industrial experience", IEEE 36th Annual Computer Software and Applications Conference Workshops (COMPSACW), pp. 440-445.
[13] Puneet Agarwal,(2011) "Continuous SCRUM: Agile management of SAAS products", ISEC' 11: Proceedings of the 4th India Software Engineering Conference, February 2011.
[14] K.Lukasiewicz and J.Miler, (2012) "Improving agility and discipline of software development with the Scrum and CMMI", IET Software, pp. 416-422.
[15] K.Schwaber and J.Sutherland, (2013) "The scrum guide: The definitive guide to scrum: The rules of the game", https://www/scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf#zoom=100.

Authors:

Dr. Kiran Kumar Jogu received his Ph.D. in Computer Science & Engineering in the year 2012 from JNTU, Anantapur, Andhra Pradesh. He is a Technical Lead (Senior Software Engineer) at IBM India Software Labs , Hyderabad, Telangana. He has 12 years of software industry and research experience with 15 research publications. His areas of interest include Software Testing, Design, Computational Intelligence based Software Engineering. He is a member of International Association of Engineers and ACM.

Dr.K.Narendar Reddy received his Ph.D. in Computer Science & Engineering in the year 2013 from JNTUA, Anantapur, Andhra Pradesh. He is a Professor in the Department of Computer Science & Engineering at CVR College of Engineering, JNTUH, Hyderabad, Telangana. He has 25 years of teaching and research experience with 17 research publications. His areas of interest include Software Design, Testing, Computational Intelligence based Software Engineering, and Computational Biology. He is a Member of ACM and Institution of Engineers(I).