**ISBN** 978-981-11-7861-0
**Proceedings of 2018 the 8th International Workshop on Computer Science and Engineering (WCSE 2018)**
*Bangkok, 28-30 June, 2018, pp.652-665*
doi: 10.18178/wcse.2018.06.109

# Proposing Pre Test-Case Testing Technique for Quality Nightly Builds

Usman Zuberi[1+] and M Asim Ali [2]

[1] Head of Process Engineering, VentureDive, Karachi, Pakistan
[2] Assistant Professor, SZABIST, Karachi, Pakistan

**Abstract.** This paper intends to highlight a problem that exist in list of local and foreign software firms related to releasing a quality internal build to quality assurance (QA) teams. The problem that exists is that the developers usually claim that they have executed a unit test while releasing a build to QA, where in actual, the execution of unit test cannot ensure absence of obvious and basic bugs. This at times results in situations like to and fro loop of builds between development and QA. In addition, this also delays actual testing efforts by QA teams and delays in execution of planned test cycles which eventually causes slippage in project timelines set by an associated Project Manager hence shaking the overall complete timeline. To verify the problem different companies were contacted and interviewed locally and internationally. Most of the individuals interviewed from different firms faced similar issues and agreed to a point that some list of cases should be executed before a build could be released to QA teams. In addition, our proposed solution was applied to different projects in different software houses and after applying solution we found that our new testing technique not only contributed towards reduced number of new, obvious and crash bugs but also bugs from previous functionality and number of test cycles were reduced.

**Keywords:** quality builds, quality assurance, reduce QA time, unit testing, nighty builds, software testing, Pre Test Cases.

## 1. Introduction

While developing a software application no matter either it's a web, desktop or mobile application, quality is of utmost concern for which various steps are taken in different organizations. The main motivation behind writing this paper is to come up with a new testing technique to ensure a release of quality QA builds or working software versions. Usually developers do execute unit tests before releasing a build to QA but unit does not and cannot ensure the absence or presence of any functionality or feature instead it ensures absence of any loop holes in code. Each developer writes his or her own unit tests and therefore unit tests or test cases only contain what developer is aware of which adds a risk of appearance of obvious of basic bugs once the build has been handed over to QA. Developers release nightly builds even after executing unit tests. Builds at times have feature and crash issues which results in various to and fro release cycles between the development and QA teams and the same time shakes actual planned test cycles that could create problems for associated project managers. Each time if an obvious or crash bug appears in a test environment, there's a chance that regression effort gets increased.

The aim is to develop or propose a testing technique that would be a combination of functional, smoke and integration testing that would be executed by the development before releasing a build to QA. These cases would be developed by QA and delivered to Development before any new release. The technique is simple enough which can be implemented by the QA or the developers. The cases would contain but not limited to new functionality, instead the QA team would identify some specific set of cases based on the main architecture of an application; these cases would be executed along with new feature set cases each time the build needs to be released to QA.

[+] Corresponding author.
*E-mail address*: usmanzuberi@msn.com.

Another intention behind proposing a testing technique is to avoid a problem instead of fixing it. According to Albert Einstein ""A clever person solves a problem. A wise person avoids it."" [1]. Also through this paper, our core intention is to check both the technical specifications along with the business requirements before releasing the build to QA. For this purpose, we ensure both the verification (technical specifications) and validation (business requirements) before the actual testing phase.

The testing technique that we are talking about is related to high-level tests which are there as a second line of test defence once the unit testing has been done. If you get a failure in a high level test, not just do you have a bug in your functional code; you also have a missing unit test. Thus whenever you fix a failing end-to-end test, you should be adding unit tests too. Guide for Author (Use "Header 1" Style)

## 1.1. Proposed technique

The technique contains following steps to be followed:

1. QA Team would either in isolation or with the help of developers would identify a set of test cases based on main architecture that would be executed each time along with the new test cases.
2. A document would be developed containing two sections, one for test cases that would be executed each time the developer needs to release a build along with the cases for new features.
3. Builds would be categorized as batches and cases for new features would be documented against associated batches.
4. Before releasing a build, QA team would be informed about the kind and type of features that would be released in a particular build
5. Once the developer is done with the unit testing then he or she would deploy the build, working software or specific version within the QA/test environment.
6. Execute the cases related to our proposed testing technique within the QA environment, if any of the case gets failed then the developer would first fix the related bug or issue then deploy it within the QA/Test environment then retest it against an associated case and finally release the build to QA.
7. Along with the release notes, the developer would attach a document related to propose testing technique.

## 1.2. Approach

All The approach we have taken is that we have introduced a testing technique i.e. a combination of functionality, smoke and integration testing between implementation and testing phases once the unit testing has been conducted by the concerned developer. Figure 1 shows that our technique is applicable between the phases of coding and testing.

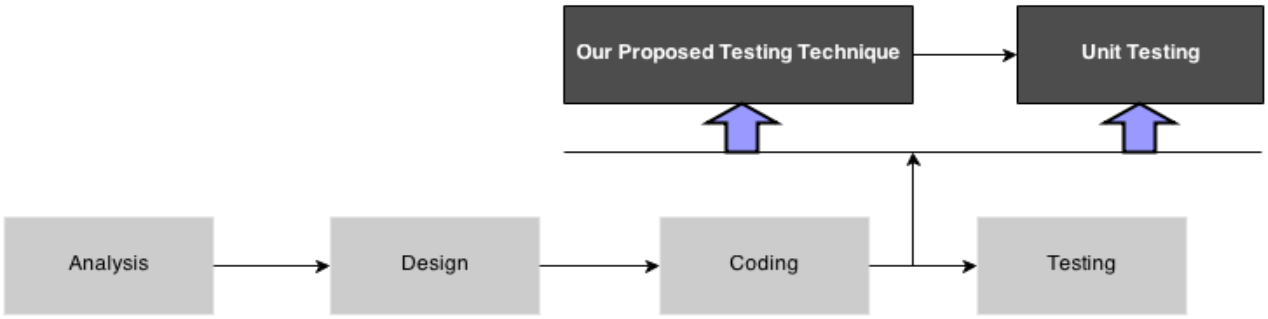



Fig. 1: Our proposed testing technique within the development cycle

A practical execution of how this approach would be applied within a production environment has been shown in Fig 2 below:

The following flow diagram explains the process of our proposed test technique. First the identification of test cases is done. If there are any test cases then the document is made which contains the test case of functionality. Then the developer unit tests these features and after that the same features are again tested by

the tester. If all the test cases pass then then the build is released to the QA otherwise the bugs are fixed and same process is repeated.

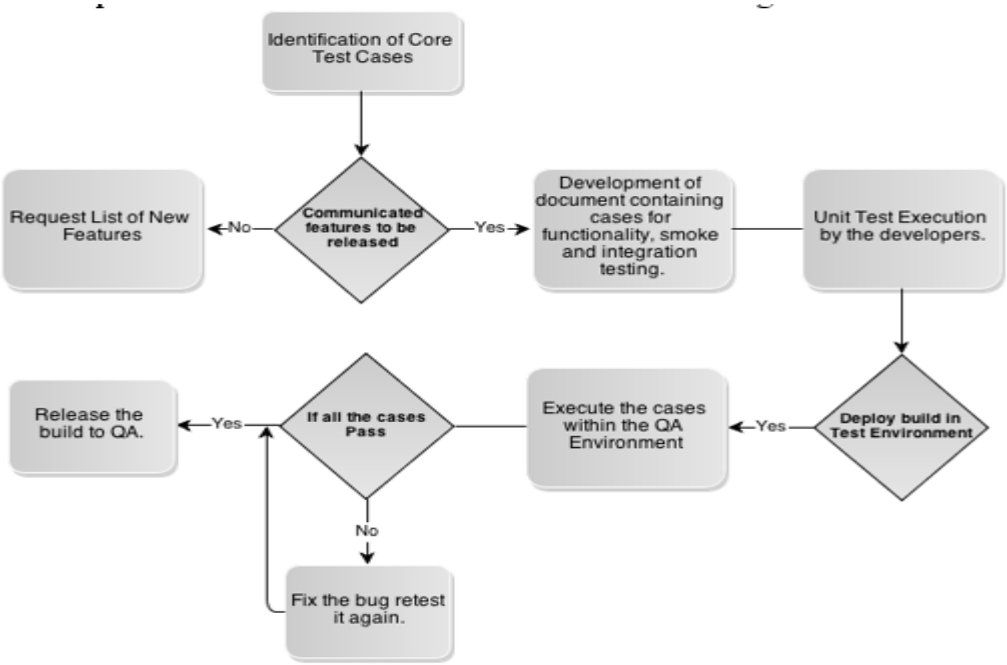



Fig. 2: Pictorial representation of our proposed testing technique

### 1.3. Brief description of results

1. Most of the individuals agreed to a point that introduction of any testing technique or process containing a combination of cases that would be executed before the release of working software to QA would help the QA teams and also the project managers.

2. We found that most of the companies don't have any such practices or techniques applied within their organizations due to which the QA personnel's along with the project managers does face the problems that we tried to solve through our work.

3. Upon implementing our solution, we found that this new testing technique greatly contributed towards less number of bugs. For instance, new and obvious bugs, crash bugs, and issues from previous functionality and test cycles compared to a build on which this new testing technique is not applied.

### 1.4. Contributions

1. Document against the proposed technique containing combination of cases.
2. Cases would be written to cover integration, functionality and smoke testing.
3. Introduction of a technique that would ensure that existing functionality based upon the core architecture of an application is working (integration testing) and also the new features have no obvious or crash issues (functionality +Smoke testing).
4. Execution of the proposed technique would ensure the verification and validation information definitions.
5. Cases that the developers have missed during unit testing would be detected during an execution of this proposed technique.
6. Additional layer of testing between an implementation and test phase.
7. Project managers could plan there builds more appropriately.
8. QA can set definite timelines for execution of major test efforts.

## 2. Related Work

In this paper author describes [1] regression testing technique into four areas in order to reduce the effort taken by in terms of cost and time in order to execute different regression test cycles. First, effective test

cases were identified that would be executed for changed part of software application only instead of executing or rerun the cases that are not directly or indirectly associated with parts of the software that has been changed. Second, technique that would rerun or re use cases for subsequent versions of the software. Third, technique that would re-execute test cases by monitoring current executions and gathering test input data. Fourth, fixing existing test cases and transforming them into new ones against the software changes made. Reports estimate that regression testing consumes as much as 80% of the overall testing budget and can consume up to 50% of the cost of software maintenance.

In this paper authors [2] describes the difference verification and validation like if the software is working is as desired which is actually verification and then if software works as per the conditions set that is validation. Authors differentiates between umbrella software testing techniques like static and dynamic testing.

Techniques and then detailed different techniques appearing under these parent techniques. Among different testing principles, authors briefly discussed an important point related to our research "start testing early", which helps in fixing list of issues during early stages of development and reduces extra efforts in finding errors in later stages of development cycle.

In this paper authors [3] describes different fundamentals related to software testing like quality is not part of test team. Test team cannot improve quality instead they measure quality of a software application. Designing tests before coding of particular feature begins would improve the quality as it would help the developers to think about their software designs and architecture well before starting an actual development of a feature of complete software applications.

In this paper authors [4] describes different testing techniques applied at different levels within the development cycle. Unit testing is conducted at lowest level or more specifically testing within the code. Unit is the smallest testable piece of software. Integration testing is applied when two or more tested units are combined together. Through integration testing, it has been ensured if combined units are working fine as a larger structure. A system testing is conducted to check end to end flow of an application, both functional and non-functional attributes such as reliability, Security and maintainability are checked. Beside different testing levels authors also highlighted static and dynamic analysis along with functional and structural testing techniques.

In this paper authors [5] discusses different testing techniques appearing under white box and black box testing techniques. While explaining the testing techniques, authors created a main umbrella of software testing under which he displayed correctness testing, reliability testing, security and performance testing. Under correctness layer he presented black box and white box techniques and in between it has been pictorially displayed where grey box testing technique. While explaining white box testing technique, some of the pros and cons have been discussed; White box testing reveals errors in hidden code, developer gives detailed reasoning about their technical implementations. Beside pros white box testing has some disadvantages like it's expensive and only covers what is included within the script. In black box testing, the tester does not have to go into actual code of an application, both the developers and tester works independently, on larger units, it's more effective as compared to white box testing.

In this paper authors [6] describes different basics related to software testing best practices like functional specifications document which provide an external view of an object easing testers to develop different test cases. Functional test variations and writing down different variations of cases to uncover maximum states within a software application. Internal beta builds to release a working product or an application to limited number of users to get feedback about a feature or larger unit, fix bugs or feedbacks received from users before a final release or shipment could be made. Advantage of nightly builds including, if major regression testing occurs due to bugs or issues recently generated then they are captured earlier. Regression testing can be resumed in parallel while the developers are in process of fixing up those bugs. Latest release is available to developers and testers sooner.

In this paper authors [7] describes about writing a Good Test Case, designing or writing a good test case is more of a complex art instead of an activity. The complexity comes from different sources; test cases facilitates an individual to uncover information. Test cases would be good in variety of ways and we cannot

classify a test case as good in all of them. Based on a statement that test case must be reasonably capable of revealing information, the boundary or horizon of test cases changes as the program gets more stable.

Authors [8] and [9] discussed about the reliability and acceptability of critical systems. Author [8] developed two scenarios one completely free from errors and the other defining an upper bound on failure acceptable failure probability. Author [9] works specifically on the improvement of a process quality using fault reporting approaches.

In this paper author [10] explain the problems and criticism of quality models. This usually happens where the quality requirements are not clearly stated. To solve this problem, the author proposed a usage scenario of quality models based on the critique of existing quality models. From this usage scenario the quality requirements are derived which are used to guide further quality model development.

### 2.1. How the work presented in this paper is different from related work?

So far, the kind of related work that we have studied or gone through is different from what we are thinking of, we proposed a testing technique that ensures bohem's informational definition for verification and validation, placed between the development and test phase from the development cycle; added as an additional layer once unit testing has been done, helps in avoiding multiple to and fro releases between the developers and QA due to obvious and basic bugs, a combination of functional + Smoke and integration testing techniques. We need to execute these cases for new and existing features. QA/test teams would work on related test cases in isolation or with the help of developers. Our testing technique is not only related to integration testing, functional testing, and smoke testing instead collection of these three techniques to ensure smooth and easy release of the builds to QA teams.

## 3. Methodology

The main intention of this paper was to overcome the issues occurring while the developers release build to QA. During the online surveys and live interviews with different professionals within the industry, we came across a point that QA and Project Manager individuals do face a situation where the build released to them have obvious bugs which results in continuous loops between the development and test which causes more test iterations and change in test and project plans which results in more testing efforts. Moreover, at times, due to absence of any such process or technique, the bugs or issues fixed by the developers gets re opened once when released, this happens because developers in some instances claim that they have executed the unit test but in reality they only fix just the required patch within the code and does not re test them before releasing it back to QA.

For this purposed we proposed a new testing technique, that would be applied between the development and test phases while the developer needs to release a build to QA. This technique would be executed till the build has not reached an Alpha or Beta stage from the release life cycle. For this purpose, we have focused on dynamic analysis of given software by inheriting mike cohen's testing pyramid where he has worked over adding a more micro level testing beside a unit test by dividing the software testing process in three different sections from a pyramid and breaking the test process into unit, service and UI. We, instead of adding any additional layer at the unit level have introduced a test technique at high level which acts as a second line of test defence. If there is a failure in a high level test, not just there can be bugs in functional code but also there can be missing unit tests. Thus whenever a failing end-to-end test is fixed the unit tests should be added too.

By executing a combination of such testing technique we could easily ensure the boehm's information definition for verification and validation i.e. building the product right and building the right product through the cases that would be documented against the combination of testing techniques that we have used.

Through this paper our intention is to check for both the technical specifications along with the business requirements before releasing the build to QA. For this purpose, we would ensure both the verification (technical specifications) and business requirements (validation) before the actual testing phase. Though we have unit testing in place through which we could ensure technical Design and coding [4] but to ensure functional design as well we have introduced integrated test cases to be executed before release to QA that would ensure functional design [4].

After reading this paper, some of the people might argue that in presence of Acceptance Test Driven development technique, why one should apply this technique, the answer is, acceptance test driven development is more of an automated testing technique where user stories are transformed into unit tests or automated test suites that are executed and checked against some acceptance criteria before the developers could start the development. In some environments, same test suites are automated by the QA teams and executed as part of scheduled jobs each time the build has to be released but this also caters only the integration part of the working product and does not the functionality of new features.

### 3.1. Benefits of this proposed Pre Test Case Testing technique

The proposed testing technique would really benefit the production environment before a build could be released to QA for testing. We have tried to introduce integration and Functionality + Smoke Test Cases which would ensure that before a build travels to QA it has already been verified and validated at an additional layer. By applying actual or major testing efforts only, it would be easier for the Project Managers and QA individuals to keep the deliveries over given timelines and better plan different internal and external builds. Moreover, our proposed testing technique would benefit both the test construction and test execution phases from the test cycles.

Pictorial Interpretation of Proposed Testing Technique

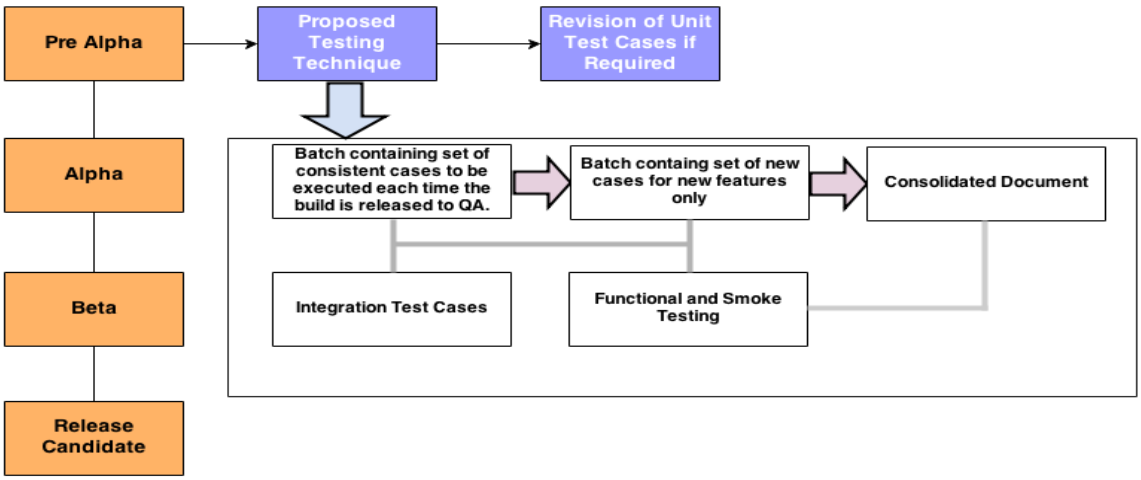



Fig. 3: Pictorial representation to show the presence of our testing technique at release life cycle

The above figure explains that there would be an additional phase before the build/release could be given to QA team during which Developers would run test case and ensure that application is at least 60% functional. What would happen is, there will be a set of consistent (that needs to be executed all the time) along with new test cases (against newly developed feature only). This would also allow them to ensure that Unit Tests written are not just validating the code instead business logic is working fine as well.

### 3.2. Experimental methodology

In order to validate our testing technique, we have applied the technique over number of applications in two different scenarios. One where the developers have released the build without executing the combination of cases that we are working on and one with the combination of cases. We would compare the results of execution on following measures:

1. Number of Obvious Bugs
2. Number of Bugs re Opened within the Test Environment
3. Number of bugs from previously working functionality.
4. Number of Critical and Crash Issues.
5. Number of test cycles executed due to presence of obvious and crash issues.

## 4. Discussion and Results

## 4.1. Survey results

In this section result of data analysis is presented. All the data has been collected and processed against a problem mentioned in associated section. In order to justify the solution proposed for a problem, surveys have been collected and also the solution has been executed at different in current and new projects to verify the results against the proposed technique. Online surveys have been collected from associated IT professionals from different software companies.

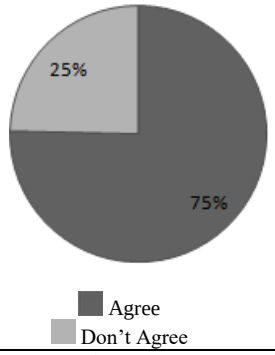
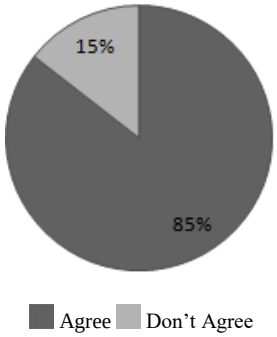
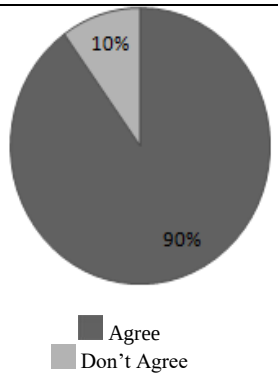
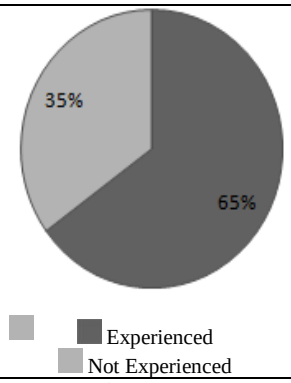
## 4.2. Response rate

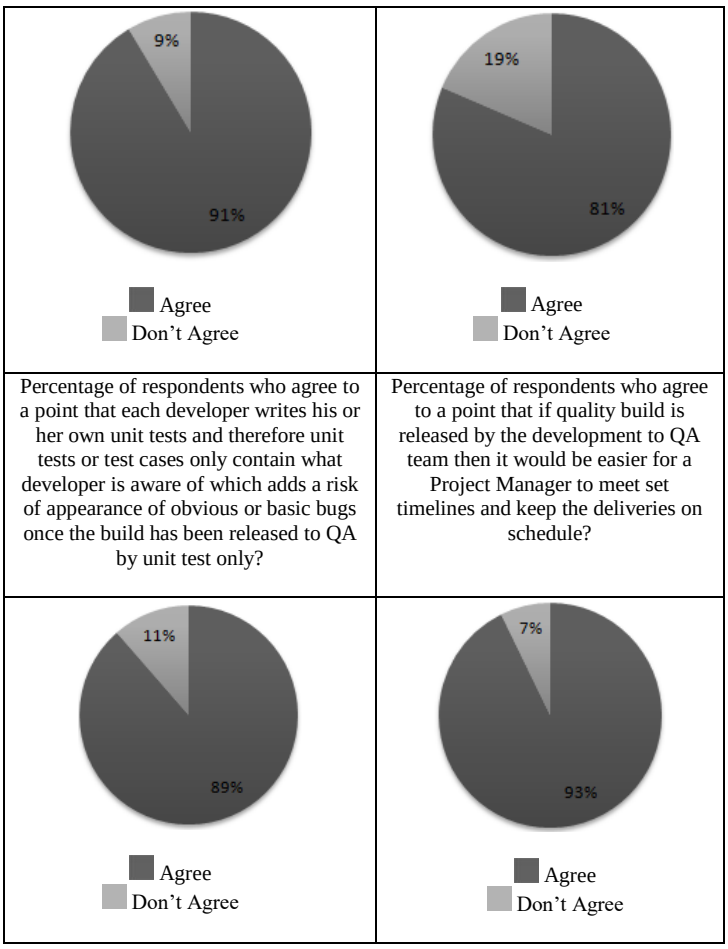
Around 180 surveys were sent to different IT professionals around an IT globe from which Total of 140 responses were collected from different individuals from local and foreign companies through online surveys. Responses have been collected from people working under Project Management, Test Engineering and Development capacities.

## 4.3. Findings

In order to verify our results, we first analysed the results collected through online surveys. In Table 1 we have presented some of the responses collected against core questions to justify the motivation against proposing a solution against stated problem.

Table 1: Online survey results

| | |
|---|---|
| Percentage of respondents who agree that execution of unit testing could not claim that build released to QA do not have any obvious bugs or functionality issues? | Percentage of respondents who agree that obvious bugs/issues caused delay in applying actual testing efforts. |
|  |  |
| Percentage of respondents who agree that a set of cases (combination of smoke, functionality and integration testing's) should be executed before releasing a build to QA. | Percentage of respondents who have experienced a situation where build has been rejected and sent back to development due to obvious or critical bugs, wrong implementation or incomplete features. |
|  |  |
| Percentage of respondents who agree to a point that further the bug gets away from the person, who caused it; the more expensive it becomes to repair. | Percentage of respondents who agree to a point that regression testing consumes as much as 80% of the overall testing budget and can consume up to 50% of the cost of software maintenance? |




## 4.4. Implementation results

Later on we implemented the proposed solution on 16 projects from four different companies. For data set in past and new projects from Mobile, Web and desktop based applications were selected. Also, the proposed solution was tested at both the utility and gaming projects. Moreover, projects with different architectures were picked up; there were projects with/without external interfaces and web services.

While verifying the results on real time projects, it was ensured that projects or data set selected should be from different domains and platforms.

When we implemented our proposed solution at different companies and checked different measures against new bugs, obvious bugs, critical & crash issues, bugs from previous functionality and total number of test cycles the figures drastically motivated us as count against different measures were quite low at builds. In order to verify the results, we released two different builds to QA for all sixteen projects, one at which bunch of cases were executed and one without executing any of the cases. On average the count was reduced by three to four times against all the measures. This happened as developers got specific set of cases which not only they checked while releasing the build to QA but also included the same cases within their Unit Test Cases.

## 4.5. Some findings regarding the benefit of this testing technique:

Some obvious benefits that we observed from this particular testing technique were:

1. Reduced number of new bugs allowed the test team to complete the testing cycles earlier as compared to ones having more bugs.

2. Less number of obvious bugs enabled the test team to apply their major testing efforts instead of verifying the builds, sending them back to development and receiving new builds with patches against obvious bugs.

3. Most of the crash issues were identified within the development environment and not within the sandbox which reduced the count of rejected builds due to crashing or similar critical issues.

4. As we included a set of cases from previously implemented functionality that the developers have to execute each time they need to release a new build, this greatly contributed towards reduced number of bugs from previously implemented functionality. Just to clarify, inclusion of these cases does not mean that test teams would not be required to execute any regression tests instead a particular set of regression would be executed each time the build needs to be released.

5. Number of test cycles on almost all the projects reduced by 37% on average.

Refer to tables below for statistics we received from different companies against list of different projects and measures: (for privacy reason the company names have been kept confidential)

Table 2: Company a results

**New Bugs**

| Batch Testing Not Executed | | Batch Testing Executed | |
|---|---|---|---|
| **Project** | **No.** | **Projects** | **No.** |
| Project A | 15 | Project A | 4 |
| Project B | 24 | Project B | 3 |
| Project C | 16 | Project C | 2 |
| Project D | 35 | Project D | 3 |
| Project E | 25 | Project E | 3 |
| Project F | 15 | Project F | 2 |

**Obvious Bugs**

| Batch Testing Not Executed | | Batch Testing Executed | |
|---|---|---|---|
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 35 | Project A | 8 |
| Project B | 14 | Project B | 5 |
| Project C | 26 | Project C | 10 |
| Project D | 20 | Project D | 5 |
| Project E | 4 | Project E | 0 |
| Project F | 3 | Project F | 0 |

**Critical and Crash Issues**

| Batch Testing Not Executed | | Batch Testing Executed | |
|---|---|---|---|
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 5 | Project A | 0 |
| Project B | 4 | Project B | 0 |
| Project C | 3 | Project C | 1 |
| Project D | 3 | Project D | 1 |
| Project E | 1 | Project E | 0 |
| Project F | 2 | Project F | 0 |

**Bugs from Previous Functionality**

| Batch Testing Not Executed | | Batch Testing Executed | |
|---|---|---|---|
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 12 | Project A | 2 |
| Project B | 9 | Project B | 3 |
| Project C | 6 | Project C | 1 |
| Project D | 8 | Project D | 2 |
| Project E | 6 | Project E | 0 |
| Project F | 5 | Project F | 0 |

**Number of Test cycles**

| Batch Testing Not Executed | | Batch Testing Executed | |
|---|---|---|---|
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 8 | Project A | 3 |
| Project B | 8 | Project B | 2 |
| Project C | 9 | Project C | 1 |
| Project D | 8 | Project D | 3 |
| Project E | 8 | Project E | 3 |
| Project F | 8 | Project F | 1 |

Table 3: Company B results

**New Bugs**

| Batch Testing Not Executed | | Batch Testing Executed | |
|---|---|---|---|
| **Project** | **No.** | **Projects** | **No.** |
| Project A | 45 | Project A | 12 |
| Project B | 60 | Project B | 25 |
| Project C | 120 | Project C | 35 |
| Project D | 110 | Project D | 40 |
| Project E | 35 | Project E | 3 |
| Project F | 130 | Project F | 35 |

**Obvious Bugs**

| Batch Testing Not Executed | | Batch Testing Executed | |
|---|---|---|---|
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 25 | Project A | 4 |
| Project B | 5 | Project B | 0 |
| Project C | 30 | Project C | 6 |
| Project D | 20 | Project D | 4 |
| Project E | 4 | Project E | 0 |
| Project F | 60 | Project F | 8 |

| Critical and Crash Issues | | | |
|---|---|---|---|
| **Batch Testing Not Executed** | | **Batch Testing Executed** | |
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 5 | Project A | 0 |
| Project B | 2 | Project B | 0 |
| Project C | 15 | Project C | 1 |
| Project D | 4 | Project D | 4 |
| Project E | 2 | Project E | 0 |
| Project F | 25 | Project F | 5 |

| Bugs from Previous Functionality | | | |
|---|---|---|---|
| **Batch Testing Not Executed** | | **Batch Testing Executed** | |
| | | | |
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 6 | Project A | 0 |
| Project B | 2 | Project B | 0 |
| Project C | 12 | Project C | 2 |
| Project D | 4 | Project D | 0 |
| Project E | 5 | Project E | 0 |
| Project F | 13 | Project F | 2 |

| Number of Test cycles | | | |
|---|---|---|---|
| **Batch Testing Not Executed** | | **Batch Testing Executed** | |
| | | | |
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 10 | Project A | 3 |
| Project B | 4 | Project B | 3 |
| Project C | 12 | Project C | 3 |
| Project D | 8 | Project D | 3 |
| Project E | 4 | Project E | 3 |
| Project F | 5 | Project F | 3 |

Table 4: Company C results

| New Bugs | | | |
|---|---|---|---|
| **Batch Testing Not Executed** | | **Batch Testing Executed** | |
| **Project** | **No.** | **Projects** | **No.** |
| Project A | 60 | Project A | 15 |
| Project B | 35 | Project B | 8 |

| Obvious Bugs | | | |
|---|---|---|---|
| **Batch Testing Not Executed** | | **Batch Testing Executed** | |
| **Projects** | **No.** | **Projects** | **No.** |

| | | | |
|---|---|---|---|
| Project A | 10 | Project A | 0 |
| Project B | 8 | Project B | 0 |

| Critical and Crash Issues | | | |
|---|---|---|---|
| **Batch Testing Not Executed** | | **Batch Testing Executed** | |
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 5 | Project A | 0 |
| Project B | 2 | Project B | 0 |

| Bugs from Previous Functionality | | | |
|---|---|---|---|
| **Batch Testing Not Executed** | | **Batch Testing Executed** | |
| | | | |
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 25 | Project A | 25 |
| Project B | 12 | Project B | 12 |

| Number of Test Cycles | | | |
|---|---|---|---|
| **Batch Testing Not Executed** | | **Batch Testing Executed** | |
| | | | |
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 12 | Project A | 3 |
| Project B | 10 | Project B | 3 |

Table 5: Company D results

| New Bugs | | | |
|---|---|---|---|
| **Batch Testing Not Executed** | | **Batch Testing Executed** | |
| **Project** | **No.** | **Projects** | **No.** |
| Project A | 50 | Project A | 25 |
| Project B | 35 | Project B | 12 |

| Obvious Bugs | | | |
|---|---|---|---|
| **Batch Testing Not Executed** | | **Batch Testing Executed** | |
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 20 | Project A | 6 |
| Project B | 15 | Project B | 4 |

| Critical and Crash Issues | | | |
|---|---|---|---|
| **Batch Testing Not Executed** | | **Batch Testing Executed** | |
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 15 | Project A | 2 |
| Project B | 12 | Project B | 3 |

| Bugs from Previous Functionality | | | |
|---|---|---|---|
| **Batch Testing Not Executed** | | **Batch Testing Executed** | |
| | | | |
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 15 | Project A | 5 |
| Project B | 22 | Project B | 4 |
| **Number of Test Cycles** | | | |
| **Batch Testing Not Executed** | | **Batch Testing Executed** | |
| | | | |
| **Projects** | **No.** | **Projects** | **No.** |
| Project A | 17 | Project A | 3 |
| Project B | 8 | Project B | 3 |

## 5. Discussion and Results

In order to reduce the testing time and complete the project within the available time duration a new testing technique implemented between the implementation and test phases that would be executed before the developer could release the build to QA. Our proposed testing technique that contained a combination of functionality and smoke test cases for new features only and consistent set of cases from previously implemented functionality that would be executed each time the developer is required to release a build. The results showed that the technique was quite beneficial and drastic difference was observed based on the measures selected for verifying the results. Not only the number of test cycles and build loops between development and QA was reduced but also upon getting a build test teams have enough confidence that build is at least 60% functional and therefore they could apply major test efforts.

## 6. Future Work

The paper was written while considering the risks associated with the waterfall model only. During last few years the water - scrum - fall has emerged as well and therefore execution or iterations against plan or locked scope would reduce the risk of solving all the bugs at a single testing phase.

The same test technique or approach would be enhanced for a system where we lock scope and plan in waterfall model and executions are agile based only. Since integration testing would be done in isolation at end of the project therefore, we would apply this technique during each sprint to ensure that any component or build is tested to an agreed point.

## 7. References


[1] Bentley, J. E. (2005). Software testing fundamentals–concepts, roles, and terminology. Proceedings of SAS Conference, 141–30.

[2] Harrold, M. J. (2009). Reduce, reuse, recycle, recover: Techniques for improved regression testing. 2009 IEEE International Conference on Software Maintenance, 5–5.

[3] Sawant, A. A., Bari, P. H., & Chawan, P. (2012). Software Testing Techniques and Strategies. Journal of Engineering Research & Applications, 2(3), 980–986.

[4] Luo, L. (2001). Software testing techniques. Institute for Software Research International Carnegie Mellon University Pittsburgh, PA, 15232(1–19), 19.

[5] Mohd. Ehmer, K. (2010). Different forms of software testing techniques for finding errors. International Journal of Computer Science Issues, 7(3), 11–16.

[6] Chillarege, R. (1999). Software Testing Best Practices. Writing, 96856, 1–11.

[7] Kaner, C. (2003). What Is a Good Test Case? Software Testing Analysis & Review Conference (STAR East), 1–

16.

[8] Bertolino A., Strigini L. (1997) Acceptance Criteria for Critical Software Based on Testability Estimates and Test Results. In: Schoitsch E. (eds) Safe Comp 96. Springer, London

[9] Arvid, J., & Science, I. (2007). Fault Reporting Processes in Business-Critical Systems. Science and Technology, (Idi), 1–30.

[10] Deissenboeck, F., Juergens, E., Lochmann, K., & Wagner, S. (2009). Software quality models: Purposes, usage scenarios and requirements. Proceedings - International Conference on Software Engineering, 9–14.