Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 13, Number 11 (2017), pp. 7725-7734

http://www.ripublication.com

Component Based Testing using the Software Development Cycle

R. Saradha
Research Scholar, Bharathiar University,
Coimbatore India.

Dr. X. Mary Jesintha
Research Supervisor, Bharathiar University,
Professor, Department of MCA,
Vivekananda Institute of Engineering and Technology for women
Thiruchengode, India.

Abstract

The IT community has figured out how to design and implement exceptionally reusable classes amid the previous ten years. This was constrained by writing down the distinctive design patterns and characterizing the component based approach. Presently another aspect turns out to be increasingly vital: do these classes act like it has been characterized amid the design-phase? Existing test-benches depend on the approach that test-classes must be implemented or additional test code in the implementation must be written. This code will incorporate a similar measure of errors than the implementation code so another sort of characterizing test cases can be useful to maintain a strategic distance from this errors. Characterizing the tests graphically e.g. by writing down a XML record has the favorable position that developers can indicate the test in a language independent design that rearranges compatibility tests. These documents depict what ought to be finished amid the test without implementing the necessary steps. The records can be made before all elements of the component that ought to be tested have been implemented.

Keywords: User Groups, Test Cases, CrashIt, Contracts

1. INTRODUCTION

Many individuals relate "testing" with demonstrating that an application runs correctly. To be sure, testing is a destructive process with the point of findinig bugs and consequently crashing an application in a secure testing environment. Utilizing this definition as a beginning stage, the made test tool should persuade programmers to identify more bugs lastly to make better projects. As indicated by these thoughts it was named CrashIt . Integrating the test into the design-process is one key-factor for a succesful test process and subsequently basic for a reliable and stable product. CrashIt rearranges the integration of the test-design into early phases of the development-cycle. CrashIt was designed and executed in cooperation by Egon Valentini and Gerhard Fliess. The primary modules of the framework were produced together.There a few methodologies for unraveling the prerequisites of such test applications. Be that as it may, no test-application which could go with engineers through the entire design and execution cycle of a software project exists. CrashIt comprises of a Java-framework, which can consequently test Java-components on the premise of experiments. After the sum total of what test had be run it produces a test-report. In CrashIt a programmed test doesn't cover the era of experiments. Programmed test implies for this situation that it runs the a test method made out of: 1. testing of single components, 2. loading, interfacing agreeing components by means of Contracts and testing the rootcomponent. (Contracts are utilized to check, regardless of whether conditions between two components hold amid correspondence), 3. loading, combining components to a total application and endeavoring to discover bugs, as well. CrashIt is a framework which underpins developers and test-engineers amid the unit-and the integration-trial of software components. The point of the project is to rouse developers to compose test-cases in light of their determinations. These testcases will be processed by this framework, which at last returns a report about the entire test. A test2 of a component-based framework comprises of 1. checking the conduct of every component (= component-test or unit-test) and 2. monitoring the correspondence between components to distinguish misusages and check the conduct of components in a system (= integrationtest). The framework really gives the abilities to characterize tests (test-cases, experiment arrangements, design), run tests and log their outcomes, check correspondence amongst components and make reports about tests.A new way to deal with make modular software which is also correct and robust, comprises in characterizing contracts between agreeing components. This approach is likewise known under the equivalent word Design by Contract. The benefit of Design by Contract is that most components shrivel, on the grounds that through predefined conditions it can be underestimated that a few cases don't show up when the component will be processed. Supplementary checks can subsequently be dropped. As a result, when the program diminishes, it is less demanding keep a diagram of its components and to keep up the code. Amid a test CrashIt underpins Design by Contract through the likelihood to embed Contracts between components. These

Contracts are straightforward Java classes which needed to satisfy some basic statutes that are caused by the reality, that Java don't bolster various legacy from classes. Contracts need to: actualize the Contract interface, and to execute either the interfaces of the two components or they must be gotten from one component and actualize the interface of the second one CrashIt can stack contracts dynamically. It is conceivable to convey them for extra tests however as a rule they are not some portion of a bundled application.

2. LITERATURE REVIEW

Baker et al exhibit two algorithms for component selection and prioritization for the following discharge problem where components will determine special, additional features or functions of the following discharge in the evolution of the software. **Voas** portrayed that the component based software development approach can potentially be utilized to lessen software development costs, increment flexibility, collect frameworks quickly and decrease the spiral upkeep load related with the help and redesign of extensive frameworks. **Vitharana et al** proposed that the weightage must be given to the cost effectiveness without breaking a sweat of get together model in software ventures when the business strategy is minimal effort and component based development. He also pointed that the outline models with evaluated technical features bolster the component designer's business strategy. Software architecture based approach in which the architectural analysis and code blend are consolidated together with a specific end goal to efficiently and accurately amass a framework out of a set of already implemented components. **Inverardi and Tivoli** have proposed a way to deal with the integration problem in the CBSD setting is to form framework by accepting a all around characterized architecture style such that it is conceivable to detect the integration anomalies. **Bucchiarone et al** have proposed a software architecture based approach in which architectural analysis and code blend are joined together with a specific end goal to gather a framework out of a set of already implemented components. All the more critically for integration process, it ought to be conceivable to give a get together guide, as well as to foresee or calculate the qualities of a gathering from those of its constituent components. **Osterweil et al** deduced in their paper as "The quality of software items is plainly unsuitably poor. An extensive and challenging system of long haul inquire about is expected to help building up the innovations required". **Krishnamurthy and Mathur** evaluated the reliability of component based software framework. Their approach is based on the test data and experiments. For each experiment the execution way is distinguished, the way reliability is calculated utilizing the reliability of the components accepting an arrangement association. The reliability of the application is the average gauge of the reliability of each test way. This approach does not consider component interface faults.

3. USER GROUPS

CrashIt was intended for three diverse user groups that thus request unique necessities:

3.1 Framework-developers

Framework deveopers compose modules for the framework. They know its center usefulness and can extend it or integrate the framework into other software development-tools.

3.2 Component-developers

Component developers compose components to be tested by CrashIt.

3.3 Test-engineers

Test engineers start unit-or integration-tests over component(s). This user gather can compose finish test-configurations. Besides, test-engineers analyze test-results and attempt to give indications where bugs could be found, as well. Proposes that component-developers should test, as well as clients, since they know the necessities of the product.

4. PARTS OF TEST

CrashIt can be utilized as a remain solitary application or as an ant-task. The descriptive test information are put away in a few documents. CrashIt utilizes XML-files3 that incorporate these vital test-configurations. A test is isolated into various parts and each part is characterized in an individual record.

4.1 Test-case :

It models one method call. It contains

- how the call must be finished by characterizing the TestClass,
- what ought to be finished by characterizing the method and every single required parameter,
- how the outcome ought to be checked (ResultChecker).

4.2 Test-sequence :

It consolidates a few test-cases to a sequence. These sequences incorporate flow-

control data that can be utilized to skip tests if a sudden outcome shows up. The sequences characterize the test-cases and allude to a connection description that is utilized to set up a test scenario. To begin with, this connection description is be utilized to initialize and associate the required components. After this has been done the test-sequence will be executed. This is important to guarantee that each test-sequence utilizes new components that do exclude data of more established test runs. The connection description is spitted in to a meaning of components and its relations. It can also incorporate contract-components that can be utilized as proxies between two components.

4.3 Test-configuration :

It characterizes an entire test scenario, including distinctive testsequences and configurations. The sequence of the characterized test-sequences incorporates additionally data for the flow-control, so it is conceivable that some test-sequences are be skipped or - relying upon a few outcomes - different testsequences are be invoked.

5. CATEGORIZATION OF CONTRACTS

There are various categories of contracts. For implementing systems that deal with contracts it is important to define categories for these types.

5.1 Linear Contracts

Linear contracts are the least difficult sort. Each state has just a single transition that closures in another state. It is exceptionally easy to deal with so much contracts, as there is an unmistakable method for how a component that backings this sort of contracts must be utilized.

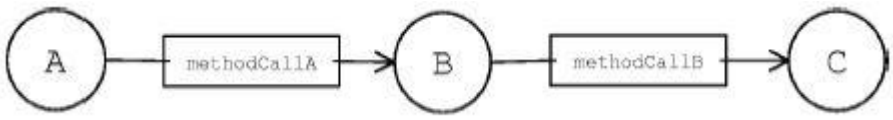

Figure 1: Linear contract

5.2 Pseudo Linear Contracts

This sort is slightly more complex than the linear sort. At least one states have at least one transitions that end either in a similar state or in a state that has not been gone to. Building a framework that can deal with such contracts must take care of two issues:

(i)There are distinctive methods for achieving a state, (ii) How regularly is a transition called that finishes in a similar state.

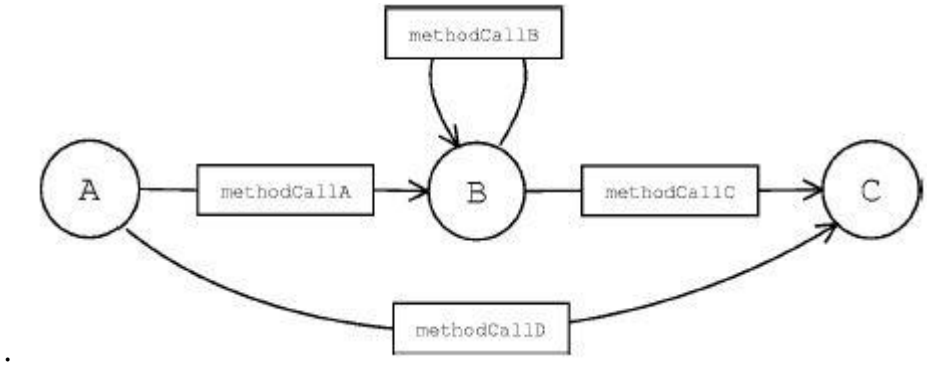

Figure 2: A pseudo linear sequence

5.3 Looped Contracts

Developers ought to abstain from composing contracts that have this conduct, yet now and then it is essential for a component to have such an agreement. A framework for these contracts must take care of a few issues: There are distinctive methods for achieving a state, How frequently is a transition called that closures in a similar state, How to distinguish loops, Finding all approaches to achieve a state, Finding the most brief approach to achieve a state.

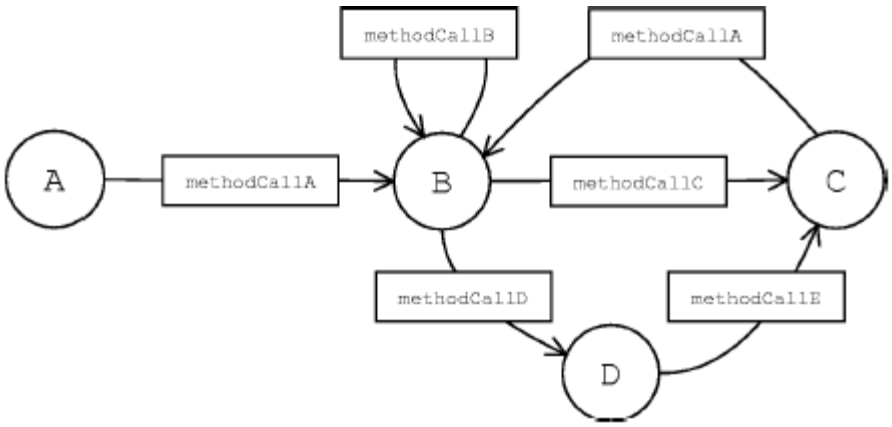

Figure 3: A Looped Sequence

6. USING CRASHIT IN THE SOFTWARE DESIGN CYCLE

CrashIt can be utilized as a part of various software development models. It won't give a solitary path to all development models however it give thoughts how CrashIt

can be utilized amid the development cycle. Test-benches are generally firmly bound to extreme programming however all different models likewise require test-benches to check the development progress.

6.1 Development-models

There are distinctive models that depend on various - at times chronicled - approaches. Regularly they concentrate on processes, milestones and activities, yet all other methodology elements impact the entire development process.

6.1.1 CrashIt and the waterfall model

Planning the tests-scenario ought to be done as ahead of schedule as would be prudent yet in this model it is helpful to begin by then, where the bits of the system have been outlined. Now it is conceivable to characterize the intermediaries between the segments that can be utilized for monitoring the communication between them. The following stage is to characterize the tests for every segment amid its plan. Writing the tests previously the code has been composed is a piece of eXtreme Programming however fits exceptionally well even in this development model. On the off chance that it is easy to characterize tests for the segment it is most likely even easy to utilize it. Amid the implementation of every part it can be tested by utilizing the characterized test-cases in the past phase. The system-and integration-tests should be possible by joining parts by utilizing the connection-configuration.

6.1.2 CrashIt and the spiral model

CrashIt can be utilized amid each cycle when the present implementation must be checked. As indicated by its utilization in the waterfall model it can be utilized as a part of the related strides.

6.1.3 CrashIt and prototyping

CrashIt can be utilized to confirm the prototypes after one stage has been done. This model makes the need of integration tests vital. in any case, it might make enormous endeavors adjust the unit test between two circles, in light of the fact that the interfaces may firmly change amid a re-outline. The descriptive approach of characterizing the tests that is utilized as a part of CrashIt may permit a quicker refresh of the test-cases that execute a test for the new form.

6.1.4 CrashIt and XP

Testing is one focal idea in XP so CrashIt can be utilized, as a test-seat in this approach to confirm that an element has been effectively implemented. Additionally, in this model the descriptive approach that is utilized by CrashIt to characterize the test-cases streamlines the changing of the test-case on the grounds that the descriptive information are free from the present implementation. A test can be run notwithstanding when a few interfaces or classes have not been completely adjusted in light of the fact that no test-class must be compiled. In JUnit for instance a test must be run if the sum total of what interfaces have been completely adjusted. Else it is unrealistic to arrange the test-class.

7. OVERALL ARCHITECTURE

The general architecture of CrashIt takes after the fundamental rule that it is to be conceivable to supplant everything that is valuable to trade and to give a straightforward method for integrating new components or extensions. This requirements powers the framework to help a declarative configuration that can be broadly adjusted without recompiling components of the framework. It is conceivable to compose its own configuration-subsystem yet a full-highlighted XML-based form is a piece of the real CrashIt implementation.

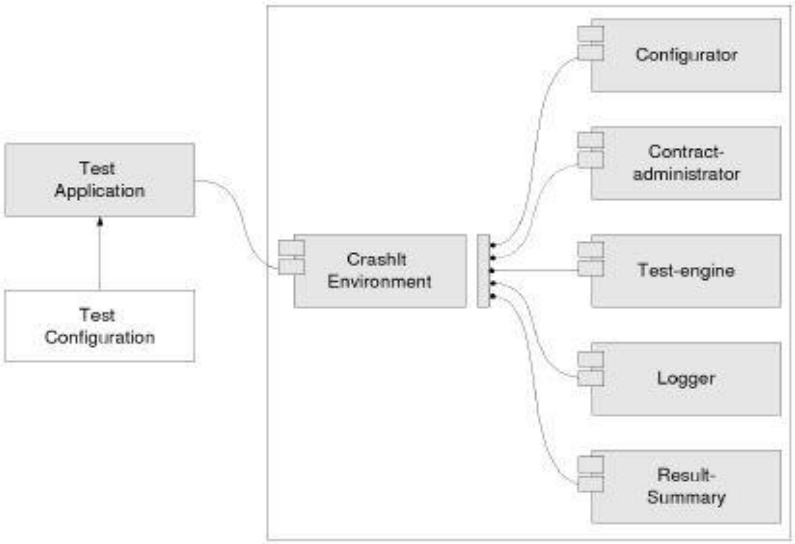

Figure 4: CrashIt modules

Over the Figure 4 comprises of a few fundamental parts: Namely,

Configurator : This segment is in charge of initializing a few sub-components. It along these lines is utilizes a configuration protest including all data on the configuration. The configuration can stack and interface the components that are essential for the test.

Test-engine : This part is capable to run the tests. It utilizes a testconfiguration that holds all data about the test.

ContractAdministrator : This part is in charge of overseeing contracts which are utilized amid a test. These contracts can cooperate with each other and they in this way utilize the contractadministrator which can restore the accessible contracts for a test run.

Logger : CrashIt gives a configurable logging interface that permits to utilize diverse logging mechanisms. An implementation of a logging framework – completed by two understudies of Graz University of Technology - and a connector for utilizing Log4J as logging component in CrashIt are incorporated into the present appropriation.

ResultSummary : This segment can make outlines of what occurred amid the test.

CONCLUSION

Designing and implementing CrashIt and the need of managing points like XP (eXtreme Programming) doubtlessly expanded my resources in designing software. The real form of CrashIt can be viewed as the center of a greater framework that will be executed in future. It will now be utilized as a part of an address about software design and along these lines ideally a considerable measure of criticism can be dealt with in future discharges. This variant is the effective demonstrate that the thoughts of CrashIt can be utilized for testing part based frameworks. The greatest preferred standpoint of CrashIt is up to now is that no test-code must be written and in this way the test-group van begin to design and execute the test parallel to the development group without the need of existing interfaces. An other preferred standpoint of CrashIt is its modular design with the goal that it ought to be anything but difficult to incorporate it into an other environment e.g. testing J2EE applications.

REFERENCE

[1] Latika Kharb: Proposed C.E.M (Cost Estimation Metrics): Estimation of Cost of Quality in Software Testing: International Journal of Computer Science and Telecommunications, Volume 6, Issue 2, February 2015.

[2] Latika Kharb: Assessment of component criticality with proposed metrics, INDIACom-2008: Computing For Nation Development, sponsored by AICTE, IETE, and CSI: INDIACOM-2008

[3] Research Areas of the Software Engineering Group, Retrieved on November 18th 2011

[4] Latika Kharb: Proposing a Comprehensive Software Metrics for Process Efficiency: International Journal of Scientific & Engineering Research, Volume 5, Issue 9, September-2014 78 ISSN 2229-5518.

[5] Latika Kharb: CCTF: Component Certification & Trust Framework: International Journal of Scientific Research in Computer Science and Engineering: Vol-1, Issue-6 ISSN: 2320-7639.

[6] Ivica Crnkovic; Stig Larsson; Michel Chaudron, "Component-based Development Process and Component Lifecycle."

[7] Luiz Fernando Capretz, "Y: A New Component-based software life cycle model", Journal of Computer Science 1 (1): 76-82, 2005, ISSN 1549-3636 © Science Publications, 2005.

[8] K.Kaur; H Singh, "Candidate process models for component based software development", Journal of Software Engineering 4 (1):16-29, Academic Journal Inc, India 2010.

[9] Tomar, P. ; Gill, N.S. , "Verification & Validation of Components with New X ComponentBased Model", in Proceedings of 2010 , Software Technology and Engineering (ICSTE), 2nd International Conference, San Juan, PR, 3-5 Oct.

[10] Lata Nautiyal; Umesh, K; Sushil, C, "Elite: A New Component-Based Software Development Model", Int.J.Computer Techology & Applications,Vol 3 (1),119-124.

[11] Latika Kharb: Complexity Metrics for Component-Oriented Software Systems: ACM SIGSOFT Software Engineering Notes Page 1 March 2008 Volume 33 Number 2. [14] Latika Kharb: An Agent Based Software Approach towards Building Complex Systems, TEM Journal, Volume 4, Number 3, Pg. 287-291, August 2015.