ISSN 2250-3153

Walking Skeleton Strategy in a Test Driven Development

Hina Kousar, Kavitha Kumar

MS Computer Science, Christ University

Abstract- The objective of this paper is to outline the implementation of the Walking Skeleton Strategy in the environment of the test driven development. It is often observed that the project build is time consuming which, delays the project accomplishment, this paper will help us understand how the implementation of Walking Skeleton Strategy is time and cost effective by automating the project build, deploy and test end-to-end.

Index Terms- automated; development; skeleton; code; iterative; end-to-end testing; integration; acceptance test; build; deploy; coverage; agile.

INTRODUCTION

Walking Skeleton Strategy is a small implementation of the system or a slice of code that performs a small end-to-end function [1] that can be built, tested and deployed automatically. It may not use the final structure but it should link together the main architectural components. The functionality and the architecture can then evolve in parallel.

This pattern or strategy was first found in 1994 by 'Alistair Cockburn'. This approach will involve in analyzing and collecting all the required components for starting a project. Once it is observed that the collected components are sufficient then the basic functionality of the project is built. Later the project would evolve to reach the final stage. It also has automated unit testing, integration testing, acceptance testing, static analysis and code coverage.

Test Driven Development will be process of development, which involves iteration of very short development cycle. First an automated test case will be written by a developer for a specified change or new function then writes a minimum amount of code to pass the test and finally improve the design of the code or alter the codes internal structure without changing its external behavior [2] [16].

This paper talks about how test driven development will be implemented with walking skeleton method and also talk about TDD unit testing techniques provide an overview on how to create the walking skeleton. Having a structural element connected and a sample piece of function running on it. More functionality will be developed in parallel with revising the architecture to be more robust.

WHY WALKING SKELETON

A. To build an Effective and Robust Software

Without testing any code is a "Bad Code". It doesn't matter how efficient the programmer's code will be. But it is very difficult that one could possibly foresee all the possible errors during the implementation. Hence, in this approach with automated tests the characteristics and behavior of the code can be changed in a fast manner and verify it along with the implementation. Without the tests it is not known whether the code is getting better or going worse.

B. Early-Stage Technology Deliverables

- Automated Build
- Automated Deploy
- Automated Acceptance Tests
- Automated Unit and Integration Tests
- Automated Code Coverage
- Automated Static Analysis
- Continuous Integration

CREATING A WALKING SKELETON

A walking skeleton as the name says it is an end-to-end implementation of the system with only "bare bones" of functionality. It will be an iterative approach. At first iteration the walking skeleton is developed and in the later iterations the skeleton will be fleshed out with more functionalities and capabilities, as shown in Figure 1.

Developers use number of techniques such as agile development, scrum and extreme programming to maintain the pace of development without solving all the possible issues [20]. In place of external systems in the project facades are used before the final interface is available. Mock screens and stubs can be used to fill the incomplete developments. Migration of full database is not required at the beginning or before any technical task begins.

If the system has lot of modules to be interacted, all the modules should be identified early in the process which helps to establish connection between the modules mutually. Without a complete understanding or knowledge on the modules present in the application it is tedious task to manage changes across the distributed system. The integration of the modules requires strong knowledge of the system.

The application build using walking skeleton technique demonstrates potential functionality of the application.

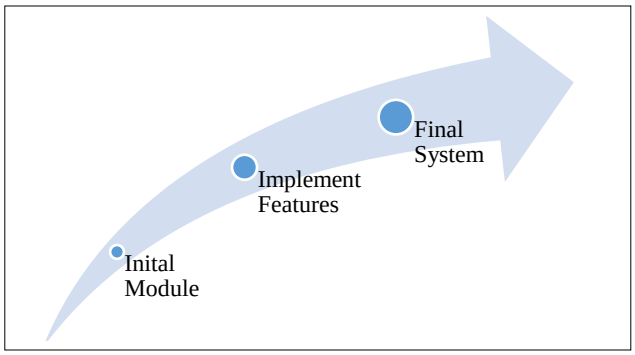

Figure 1: Walking Skeleton Approach

C. Review User Stories

User stories are short description of user ideas or desired capability of customers. Use cases are written on sticky notes and arranged on walls or tables to facilitate planning and discussion. Large user stories are known as epics. These stories are reviewed and to simplify it is split into multiple smaller user stories before it is used. Stories can be split into multiple small units based on its complexity to fit in single iteration. The iteration continues till it satisfies a high level acceptance test [14].

D. Write and Automate First Acceptance Test – Write a test which will fail.

TDD (Test Driven Development) is different from traditional development. When there is a need to implement a new feature. The first thing that should be checked is whether the existing design is the possible best design for the user to implement the functionality.

A Walking Skeleton is finalized code built for production including regression tests and is intended to develop with the system. Once the system is developed and started running. The system will continue running for the rest of the project even though it leads to Incremental Re-architecture [3]

Compared to the other way of coding in this technique the programmer instead of writing code first and testing it, writes the test code first. Then, writes the functional code keeping the test into consideration. This is done in small steps at first one test and small functional code is written by programmer. A programmer coding in test driven development method should not write a new function until there is a test which fails because the functionality is not present. There is no code until the test is in place for the selected functionality. Once the test is in place then the programmer is required to ensure that the test suite passes even though it fails many times. TDD [18] can be challenging at initial stages when a programmer is working on it for the first time, it is easy to forget and write code covering the functionality without writing a new test.

There are two types of test driven development:

- Acceptance test driven development (ATDD) [19].
- Developer test driven development (DTDD).

Here the focus will be on ATDD. When the testers write a single acceptance test on his preferred terminology with enough production functionality or code the tests to execute it. Single acceptance test is also known as behavioral specification. ATDD specifies detailed requirements that can be executed for solution on required time. ATDD is also called Behavior Driven Development (BDD) [4].

An automated acceptance test provides the measurements of the objects in progress during the development. They provide a fair idea on the complexity involved in the implementation of the features. Whenever a change will be made in the source code the automated acceptance test has to run automatically. Acceptance criteria should also be automated. It will be very vital to have a tool which will integrate the build process and build server without any manual intervention, as shown in Figure 3.

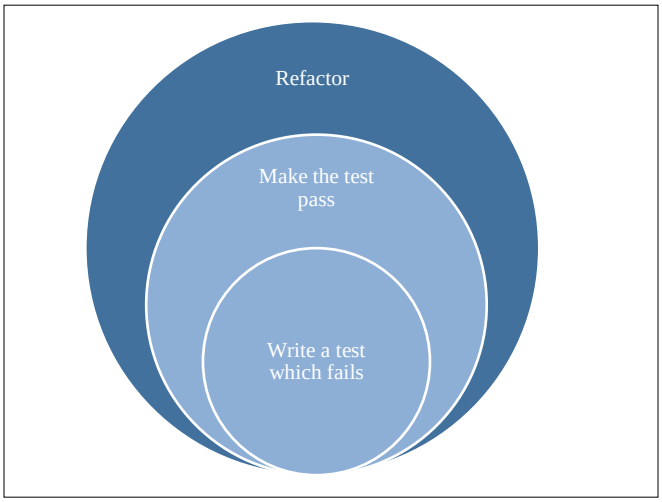

Figure 2: TDD Process

1) Basic Flow of TDD

Given functional requirements, TDD software engineers develop production code through rapid iterations of the following:

- Small numbers of automated unit test cases are written.
- Test cases written should be run and made sure that all the test fail since code for new features is still not added.
- Implement new code for written tests such that when the tests are executed all the test cases pass.
- After refactoring (making) necessary changes ensure that the new unit test cases pass when these are re-run and no duplicate test case exist.
- In regression testing re-run all the test cases in the code base to ensure that all the test cases run are passed and code updates did not break any of the previous working functionalities.
- If any of the test fails the build is reverted to avoid excessive debugging after which build is fixed by taking small modules. [21]

This iterative developmental cycle is used to develop a piece of new functionality [5], as shown in Figure 2.

TDD delivers:

- Automatically testable software

- Higher test coverage
- Lower defect rates
- Reliable refactoring

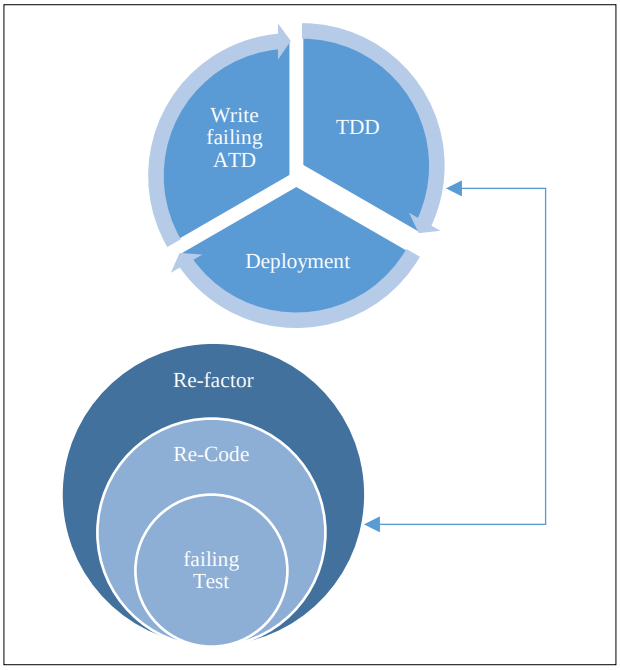

Figure 3: ATDD Process

E. Automate Build and Package

The process of scripting or automating different tasks during the software development is termed as Automatic Build. Compiling the source code to binary code is known as binary code packaging. Compiling and linking of source code modules is required to create the final deployable object. Build will be automated and it does not require direct human intervention.

In a distributed process advanced system build automation technique provides remote assistance for processing the automation. The calls to the compiler and linkers can be sent across multiple locations to improve the speed of the build. The machine intelligence of distributed systems will understand source code dependencies and will send it to dependent compliers and linkers in different systems. Tools are used to discover the relationship between the source code and its dependent machine they are Electric Cloud ElectricAccelerator, [6] Rational ClearMake distributed [7] and Platform LSF lsmake [8] based on user-configuration. An example of automatic build is shown in Code.1.

Automate Build and Package delivers:

- Removal of redundant tasks
- Improved product quality
- Reduced bad builds.
- Accelerates compile and link processing.

```
DATA lo_nd_cn_dept TYPE REF TO if_wd_context_node.
DATA lt_cn_dept TYPE wd_this->elements_cn_dept.

*   navigate from <CONTEXT> to <CN_DEPT> via lead sel
ection

lo_nd_cn_dept = wd_context-
>get_child_node( name = wd_this->wdctx_cn_dept ).

lo_nd_cn_dept-
>get_static_attributes_table( IMPORTING table = lt_cn_dept
 ).
```

Code.1. Automatic Build in Webdynpro ABAP

Automatic build helps the programmer to concentrate more on the business logic than spending time in built, staging, deploying, creating objects, compiling source code, backing up old versions, tagging new versions which leads to software faults [17]. Subsequently the programmer will be left with more time to enhance the beauty of the application interfaces or make it more user friendly.

F. Automate Deployment

Deployment in a real time application is a complex task which is costly, error prone, time consuming and should be repeated in different environment when done manually [4].

To overcome all the above hardships automated deployment tools are being used such as deployment manager and octopus deploy. These tools will integrate the systems and provide key steps for deploying number of application services, databases and business intelligence tools. This integration behavior makes the configuration and maintenance of the deployment cost effective and easy. One can have a clear picture on when and at what time the deployment will occur for each deployment. It holds information of the server and can easily match the configuration to deploy the objects which were deployed to the same configured servers. When installing or deploying software in a new test environment the existing set up can be used as once it is automated, it is repeatable.

G. Implement Features

All the features are not implemented during the initial stage of creating walking skeleton. All the requested features by the client will be implemented in this phase. In the iterative development and testing phases of walking skeleton approach it always involves lot of interaction from the client to the project and the project teams. So, the requirements of the project and the client's expectation is known at the very early stage, which is easy for the team to code the minimal requirement in the initial model and later develop the other requirements for the system to evolve[10] in its characters and behavior.

Client's satisfaction plays a very important role during this phase. If there is a negative feedback from the client it will lead to a lot of escalations which is time consuming. This will impact the deliverables and cause damage to the cost planning. One has to be very careful while implementing the features.

H. Automate Static Analysis

Automating static analysis is very important as it forms a large portion of the testing. It is fast to analyze what effect has taken place on system by an immediate change and cost effective for simple checks. In this analysis there is an attempt to check all the vulnerabilities in the static code using Taint Analysis, Data flow Analysis and Lexical Analysis.

Taint Analysis and Data flow Analysis are discussed below:

- Taint Analysis - It is the ability to monitor the code as it executes [15]. This method forms a filtering mechanism for automatic detection of overwrite attacks and internet attacks.
- Data Flow Analysis – Method used to draw conclusion of the data. It is used to collect the dynamic information of the system when the system is in the static state. Data flow analysis consists of three areas to cover. They are basic block, control flow analysis and control flow path.
- Based on the complexity of the application unit test cases and component test cases are written.

I. Automate Code Coverage

Code coverage approach is to ensure that all the statements and paths in the code are executed. This may not find bugs but can identify areas of code which are not covered in test case. It is a useful metric to ensure high code coverage. Once this is been automated it is very helpful for testing the code in regression cycles.

Adaptive random testing can achieve higher code coverage than testing with the same number of test cases. [12] [9]

It includes coverage types as:

- Statement Coverage: A statement is covered if it is executed. Advantage of having is it identifies any of the block not executed in the code.

```
If(x > 20)
Dies "This is a dead end";
```

Code. 2. Example for Statement Coverage.

- Branch Coverage: It is also known as decision coverage. The aim of branch coverage is to cover every branch or decision in the code.

```
If(x)
        Print "a";
Else
        Print "b";
```

Code. 3. Example for Branch Coverage.

- Path Coverage: This will be done to ensure that all the branches are covered during testing in the code which may not happen in branch coverage. It ensures all the test decision outcomes are independent of one another. This makes path coverage robust.

```
public class PathExample {
public int returnInput{int x, boolean condition1,
                                        boolean
condition2,
                                        boolean
condition3}{
        if {condition1} {
        x++;
        {
        if {condition2} {
        x--;
        }
        if {condition3} {
        x=x;
        }
        return x;
}
```

Code. 4. Example for Path Coverage.

- Condition Coverage: It is to cover evaluation of the Boolean expressions. It measures the conditions independently.

J. Integration Tests

Integration testing is done to test individual software components and detect interface defects. Once the components are tested these are available for system testing. Here individual tested modules are combined. It is done to uncover the problems in combined modules. Code inspections can be used for verifying and validating system [11].

Types of integration testing:

- Top down Testing: The approach is to test the integrated modules from top to bottom. Complex pieces are specified and broken down into small pieces for testing. This method detects unseen design errors at early stages of testing which avoids extensive re-designing at later stage of testing.
- Bottom up Testing: This approach is opposite of top down method. Modules are tested at lowest level of hierarchy and these are integrated to form larger sub modules until a complete top level module is formed. This type of testing is well suited for object oriented systems in which individual objects may be tested using their own test drivers.
- Big Bang Approach: This approach combines both top down and bottom up approach. It involves integrating the modules to complete software system. The unit modules are combined to form bigger modules and tested with top down approach.

TECHNIQUES TO CREATE WALKING SKELETON

Walking skeleton technique varies with the system being developed. In case of a client - server system it will be a single screen connected for navigating to database and back to screen. In a front end application system it acts as a connection between the platforms and compilation takes place for the simplest element of the language. In a transaction process it is walking through a single transaction.

Following are the techniques which can be used to create a walking skeleton:

- Methodology Shaping: Gathering information about prior experiences and using it to come up with the starter conventions. Following two steps are used in this technique:
    1) Project interviews
    2) Methodology shaping workshop
- Reflection Workshop: A particular workshop format for reflective improvement. In the reflection workshop team members discuss what is working fine, what improvements are required and what unique things will be added next time.
- Blitz Planning: Every member involved in project planning notes all the tasks on the cards, which will then be sorted, estimated and strategized. Then the team decides on the resources such as cost, time and discuss about the road blocks.[22]
- Delphi Estimation: A way to come up with a starter estimate for the total project. A group consisting of experts is formed and opinions are gathered with an aim to come up with highly accurate estimates.[23]
- Daily Stand-ups: A quick and efficient way to pass information around the team on a daily basis. It is a short meeting to discuss status, progress and setbacks. The agenda is to keep meetings short. This meeting is to identify the progress and road blocks in the project.
- Agile Interaction Design: A fast version of usage-centered design where multiple short deadlines are used to deliver working software without giving important consideration to activities of designing [24]. To simplify the user – interface test LEET a record/Capture tool is used.[25]
- Process Miniature: A learning technique as any new process is unfamiliar and time consuming. When the process is complex more time is required for new team members to understand how different parts of the process fit. Time taken to understand the process is reduced with use of Process Miniature.
- Side-by-Side Programming: An alternative of pair programming is "Programming in pairs". Here two people work on one assignment by taking turns in providing input and mostly on a single workstation. It results in better productivity and cost consumed for fixing bugs is less [26]. Programmers work without interfering in their individual assignments and review each other's work easily.
- Burn Charts: This tool is used to estimate actual and estimated amount of work against the time.[3]

CONCLUSION

Walking Skeleton Strategy first and foremost reduces human intervention in the all the phases of development by automating all the processes like build, deployment, installation, test, integration and acceptance. It is time efficient and cost effective as the whole process of building a software system is being automated. The delivery of the software system can be assured on time with reduced cost and quality code which is mostly spent on the resources if there were manual work. Hence, this approach plays a vital role in early software system deliverables.

This system is not the replacement for the current manual and automated system available but can be implemented to remove time and cost constraints to build robust applications.

Developers spend more time in solving the unseen issues and do not cover them while solving the other issues. The final solution is rarely known at the beginning. A walking skeleton is a working system that evolves with implementing the requirements. A big design is not seen upfront; it is not a frame work to prove a technical concept. Subsequent planning sessions will be helpful to uncover the unseen issues.

REFERENCES

[1] Alistair Cockburn, "In Practice (Strategies & Techniques)" in Crystal Clear A Human-Powered Methodology For Small Teams including The Seven Properties of Effective Software Projects, 2004

[2] Martin Fowler, "Improving the Design of Existing Code" Addison-Wesley Professional publications, June 28th 1999.

[3] Woodward, C.J.; Cain, A.; Pace, S.; Jones, A.; Kupper, J.F., "Helping students track learning progress using burn down charts," Teaching, Assessment and Learning for Engineering (TALE), 2013 IEEE International Conference on , vol., no., pp.104,109, 26-29 Aug. 2013

[4] Scott W. Ambler Introduction to Test Driven Development (TDD) [online]. Available: http://www.agiledata.org/essays/tdd.html. (Accessed: 08 December 2013).

[5] Williams, L.; Maximilien, E.M.; Vouk, M., "Test-driven development as a defect-reduction practice," Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on , vol., no., pp.34,45, 17-20 Nov. 2003
doi: 10.1109/ISSRE.2003.1251029

[6] DR. Dobb's. Building Scalable Web architecture and distributed systems [Online]. Available: http://www.drdobbs.com/web-development/building-scalable-web-architecture-and-d/240142422. (Accessed: 12 December 2013)

[7] Dr. Dobb's. Take My Build, Please. [Online]. Available: http://www.drdobbs.com/take-my-build-please/184415472 (Accessed: 12 December 2013).

[8] LSF User's Guide - Using lsmake [Online] .Available: http://users.cs.fiu.edu/~tho01/psg/3rdParty/lsf4_userGuide/10-lsmake.html (Accessed: 20 December 2013).

[9] Williams, T.W.; Mercer, M.R.; Mucha, J.P.; Kapur, R., "Code coverage, what does it mean in terms of quality?," Reliability and Maintainability Symposium, 2001. Proceedings. Annual , vol., no., pp.420,424, 2001.

[10] Ward Cunningham. "The WyCash Portfolio Management System" ACM SIGPLAN OOPS Messenger, Vol 4, April 1993 (Accessed: 12 December 2013).

[11] Paul, R., "End-to-end integration testing," Quality Software, Proceedings .Second Asia-Pacific Conference on , vol., no., pp.211,220, 2001.

[12] Tsong Yueh Chen; Fei-Ching Kuo; Huai Liu; Wong, W.E., "Code Coverage of Adaptive Random Testing," Reliability, IEEE Transactions on , vol.62, no.1, pp.226,237,March2013.

[13] Jez Humble; David Farley, "Reliable Software Releases through Build, Test, and Deployment Automation". Anatomy of Deployment Pipeline. Ch:5, pp - 113.

[14] Nykanen, K.; Tarkiainen, M., "Nokia requirements and user story for the project MOTION," Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on , vol., no., pp.357,358, 2002
doi: 10.1109/ICDCSW.2002.1030794.

[15] Schwartz, E.J.; Avgerinos, T.; Brum ley, D., "All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask)," Security and Privacy (SP), 2010 IEEE Symposium on, vol., no., pp.317, 331, 16-19 May 2010 doi: 10.1109/SP.2010.26.

[16] Soares, G.; Gheyi, R.; Massoni, T., "Automated Behavioral Testing of Refactoring Engines," Software Engineering, IEEE Transactions on , vol.39, no.2, pp.147,162, Feb. 2013

[17] Hilford, V.; Lyu, M.R.; Cukic, Bojan; Jamoussi, A.; Bastani, F.B., "Diversity in the software development process," Object-Oriented Real-Time Dependable Systems, 1997. Proceedings., Third International Workshop on , vol., no., pp.129, 136, 5-7 Feb 1997 doi: 10.1109/WORDS.1997.609943.

[18] Cauevic, A.; Punnekkat, S.; Sundmark, D., "Quality of Testing in Test Driven Development," Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the , vol., no., pp.266, 271, 3-6 Sept .2012 doi: 10.1109/QUATIC.2012.49.

[19] Negara, N.; Stroulia, E., "Automated Acceptance Testing of JavaScript Web Applications," Reverse Engineering (WCRE), 2012 19th Working Conference on , vol., no., pp.318, 322, 15-18Oct.2012.

[20] James F Kile, "An investigation into the effectiveness of agile software development with a highly distributed workforce" (January 1, 2007). ETD Collection for Pace University. Paper AAI3271868. Retrieved on 2014-01-15

[21] Erdogmus, Hakan; Morisio, Torchiano. "On the Effectiveness of Test-first Approach to Programming". Proceedings of the IEEE Transactions on Software Engineering, 31(1). January 2005. (NRC 47445). Retrieved 2014-01-15.

[22] Alistair Cockburn.(2009) Blitz Planning [Online]. Available: http://agile2009.agilealliance.org/node/1690/ (Accessed: 20 December 2013).

[23] Lilja, K.K.; Laakso, K.; Palomki, J., "Using the Delphi method," Technology Management in the Energy Smart World (PICMET), 2011 Proceedings of PICMET '11: , vol., no., pp.1,10, July 31 2011-Aug. 4 2011.

[24] Silva da Silva, T.; Selbach Silveira, M.; Maurer, F., "Ten Lessons Learned from Integrating Interaction Design and Agile Development," Agile Conference (AGILE), 2013 , vol., no., pp.42,49, 5-9 Aug. 2013.

[25] Hellmann, T.D.; Hosseini-Khayat, A.; Maurer, F., "Supporting Test-Driven Development of Graphical User Interfaces Using Agile Interaction Design," Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on , vol., no., pp.444,447, 6-10 April 2010.

[26] Dewan, P.; Agarwal, P.; Shroff, G.; Hegde, R., "Distributed side-by-side programming," Cooperative and Human Aspects on Software Engineering, 2009. CHASE '09. ICSE Workshop on , vol., no., pp.48,55, 17-17 May 2009