

## Testing processes in business-critical chain software lifecycle

Gautham Reddy N  
Tata Consultancy Services Ltd,  
Hyderabad, Andhra Pradesh, India  
gautham.reddy@tcs.com

### Abstract:

The business-critical chain lifecycle is an agile software development lifecycle that aims at aligning the software project deliverables to attain the business objectives based on business priorities. The traditional software development projects work on the assumption that all 'equal effort consumers' would be treated equally and worked upon. The agile methodology ensures that the delivery cycles are reduced thus introducing agility in the way business is supported by underlying technologies. The proposed lifecycle model introduces new variables pertaining to the business value generation each finished piece of code would produce. Hence the software project processes have to be modified to cater to it.

Even the usual agile lifecycle testing strategies need to be modified to suit the proposed model. The test plans, test estimations, test resource management, quality control, regression plans and automation road map and plans have to be customized to cater to the new life cycle model. The secondary project management activities such as risk management, procurement management, etc also may to be modified with respect to the testing processes.

My paper aims at using the critical chain principles and proposes a software lifecycle model that can cater to business priorities and aligning the testing processes not only to development cycles but also to the actual business value created. In this paper I would take a case study and compare and contrast when project uses the regular models and this new model. I would also provide guidelines to use this lifecycle model and modify regular project management activities to cater to the new model with emphasis on the testing processes. The paper would try to provide the ideal scenarios; in project teams, in consulting firms and in new customers and expectations; where such a model could provide high impact on the way consulting companies can do successful projects and creating more value to the customers.

### Keywords:

BCCSL - Business-Critical Chain Software Lifecycle, Business critical chain, Critical chain component, Critical chain candidature

## 1 Introduction

In today's world of software development, projects more than ever before are turning to agile development methodologies. Of the many factors influencing this shift, one major factor is business impact the software would produce and quantification of the business value the individual pieces of the software would provide. The software management model, which I proposed is an attempt to customize the regular software development and project management techniques to cater to such new variables. The business-critical chain lifecycle is an agile software development lifecycle that aims at aligning the software project deliverables to attain the business objectives based on business priorities.

This paper aims at using the Critical Chain Principles and proposes a software lifecycle model that can cater to business priorities and aligning the testing processes not only to development cycles but also to the actual business value created. In this paper, I would provide customization required for the testing processes to cater to this lifecycle.

## 2 Business Critical Chain Software Lifecycle – an overview

The business-critical chain lifecycle is an Agile software development lifecycle that aims at aligning software project deliverables to attain the business objectives based on business priorities. Traditional software development projects work on the assumption that all ‘equal effort consumers’ would be treated equally and worked upon. Equal effort consumers’ means are those development activities which consume equal effort to complete. The Agile methodology ensures that the delivery cycles are reduced thus introducing agility in the way business is supported by underlying technologies. The proposed lifecycle model introduces new variables pertaining to the business value each finished piece of code would produce. Hence the software project processes have to be modified to cater to it.

This model is made such that either it can be build from scratch or use any of the standard agile methodologies and customize to suite the business priorities.

The core of this lifecycle is to identify and prioritize software components that can be independently created. Like in SCRUM for example, if use case based estimation is used, the first activity is to prioritize all the use cases and rank them and create a ready stack. This prioritization generally includes the functional importance and the technical feasibility of the use case. Both these variables decide the rank of the use case in the stack.

Business Critical Chain Lifecycle introduces an extra variable called the ‘Critical Chain Component’. Critical chain means the stack of components built in such a way that these together form the core functionality/technology framework of the application to be built. All other functionality is like an add-on to these critical chain components. Each use case should be analyzed and verified whether it can be the part of the critical chain. Hence, now the prioritization of use cases would require 3 variables functional importance, technical feasibility and critical chain candidature. This extra variable penetrates into all the project management, software development steps (requirement analysis, design, programming, testing) for this agile project and given the higher priority.

## 3 Testing processes in business-critical chain software lifecycle (BCCSL)

Testing process in business critical chain software lifecycle is of crucial importance than any other Agile models. The crux for success of the project depends highly on the testing efficiency. It is highly recommended that testing be taken out of the regular project management and its strategic, tactical and operational processes are managed separately, thereby making testing a new project in itself. This paper also illustrates the test management required in BCCSL.

### 3.1 Calling it testing is a misnomer

Calling the entire process testing would be a misnomer. It involves all the three tiers of management specified above, only the operational part being the actual testing. Quality is to be maintained high in all phases and the last effort being actual testing. The correct



word would be “Quality Assurance”. Calling it quality assurance allows us to better quantify the QA effort by calculating the ‘Cost of Quality’.

### **3.2 Testing processes in BCCSL**

The emphasis of this section would be to customize each such strategic, tactical and operational processes specific to testing to suite the BCCSL. The following sub sections describe each of the testing processes in an agile software development project, that need customization.

#### **3.2.1 Test plan**

In this lifecycle model, testing is of core importance and the test plan is finished first and on top of it the development plan is laid out. This implies that initially the test plan is created and the test iteration is decided. And later, the development plan is laid such that the development iterations are matched with the test iteration. This process will go on iteratively, until a correct balance is reached. But more importance given to the test iteration provides the advantage position to reduction of cost of quality. It can be considered close to the Test Driven Development (TDD) methodology used commonly in many Agile projects, but where as TDD is more of a programmer driven where the programmer completes all the unit test cases first and then starts development, this is more of test management perspective where the project management model is directed by the test management and later with development management.

#### **3.2.2 Test estimations**

The test estimation process help create the ‘Test Iteration plan’. Similar to the development estimation, there are quite a few techniques available for test estimation. But one technique that is most suitable to Agile development projects is use-case based estimations. Similar to the programming estimation, test estimation also divides the use cases into simple, medium, complex based on functional complexity, technical complexity and critical chain candidature. To enable the team to better estimate, use cases can be broken down into test cases or in some cases test stories. Test story can be considered as a quicker way to document the test cases. Instead of writing conventional test cases, a test suite from a user perspective can be written using which the tester would test the entire flow of functionality. Although, test stories cannot substitute the test cases at all times, adopting test story documentation wherever possible would be beneficial. Historic information and/or iterative estimation (i.e., re-estimating after iteration based on the iteration experience) are most suitable to make the estimates as realistic as possible.

#### **3.2.3 Test resource management**

The resource management needs to be broken off the normal project management, where it is for the entire project, and managed separately through test resource management. The main reason for this is because iteration length and frequency depends on the test team’s confidence but not on the development team. (See 3.1.3) The resource ramp up and ramp down of test team is proportional to the work load to be taken for the testing of the project. The ideal resource management would start with test lead, senior test

members included right from the start of the project, then a smooth ramp up, steady resource maintenance and a smooth ramp down at the end of the project.

Most agile models inherently follow this sequence in resource management. But the BCCSL needs customization of test resource management. The difference is how individual resources are placed in the work break down structure (WBS).

- a) The test lead, senior test member (if possible) be a part of the project from the start. This gives a definite advantage to the test team as it would have a strong role in requirement analysis, design and project management.
- b) The work break down structuring needs careful customization, as it is required that the best of the test resources work on the critical chain components. If for example, two use cases have same criticality (see 3.1.3), but one of them belongs to the critical chain components, and if the test team have two members ready to be allocated to each of it, the effort required would be the same but as one of the use case is a critical chain component, it is required that this use case be tested by the senior (or should I say more efficient) of the two test team members. There would be no change in effort spent but there would be a quality difference. This is called 'Buffer of Quality' and is given to the critical chain components. This is a project management cover to make sure that the critical chain components have better and efficient resource to work on and reduce the cost of quality.
- c) Automation engineers need to be placed in the project much ahead of time when compared to the norm. As automation is of core importance in BCCSL (see 3.1.8), the automation engineers would be part of the requirement analysis, design. This helps them better understand the components they need to automate, create efficient reusable scripts, also help design/UI teams to choose components (like Ajax over newer frameworks etc) that are easily automatable

#### ***3.2.4 Scope creep- deterrent effect on cost of quality***

Agile methodology is used where the scope is not clear and software is developed in an iterative manner. In other words Agile has very high leverage for scope changes. The term 'Scope creep' seems to have no relevance. But this term needs to be modified to suit the agile philosophy. Though scope is not clear, and development is iterative, when a particular iteration is decided and the functionality is chosen for development, ability to freeze on this chunk of functionality (pertaining to this particular iteration) is important for timely completion of the development of this iteration. This is to prevent iteration within iteration. This scope creep within iteration is highly deterrent to the timeliness as well as quality of the deliverable of this iteration. It could so happen that the code of a particular iteration can be scrapped in the future because of a large change in the functionality. In iterative mode of development, success depends on the ability of the team to freeze the scope of that iteration. Many a times the scope creep goes unnoticed or overridden by the management yielding to delivery pressures and reduction in quality of the iteration. But effective tracking of the effects of the scope creep on the quality of the

deliverable puts the team in an advantage position, where it can negotiate options with the facts.

If it is assumed that the iteration time period is fixed and the efforts for each phase in the iteration is also determined at the start of the iteration, then the following are the two new metrics required to be tracked to analyze the deterrent effect of scope creep on the cost of quality.

- a) The number of bugs escaped from each of the defined phases of the iteration is to be tracked against the % variation in scope (or the scope creep) in that iteration. In most cases, there would be a proportional relationship between the two entities. As according to the assumption the effort for all the phases is constant, the scope creep will elongate the design/ code rework until the additional scope is accommodated. This pushes the timelines of all the phases and as the total iteration time period is constant, it would crunch the time period designated for testing. Either the quality is compromised or the team members work overtime to fill up this effect. In both cases the testing efficiency is bound to decrease.

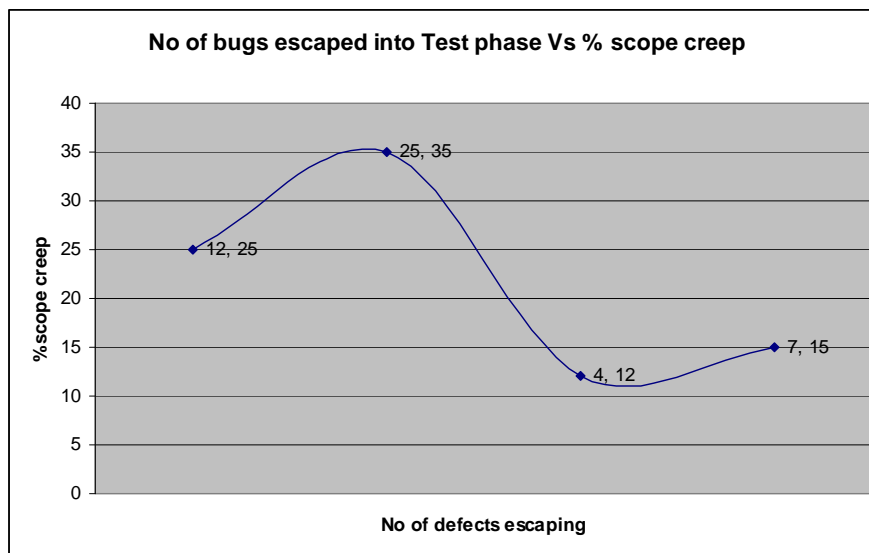


Figure 1: Sample Graph- No of bugs escaped into Test phase Vs % scope creep per iteration

- b) The number of bugs escaped from each of the defined phases of the iteration is to be tracked against the % penetration (into the iteration cycle) of the scope creep into the iteration cycle. In most cases, there would be a proportional relationship between the two entities. As according to the assumption the effort for all the phases is constant, the scope creep in number of days pushes the timelines of all the phases and as the total iteration time period is constant, it would crunch the time period designated for testing.

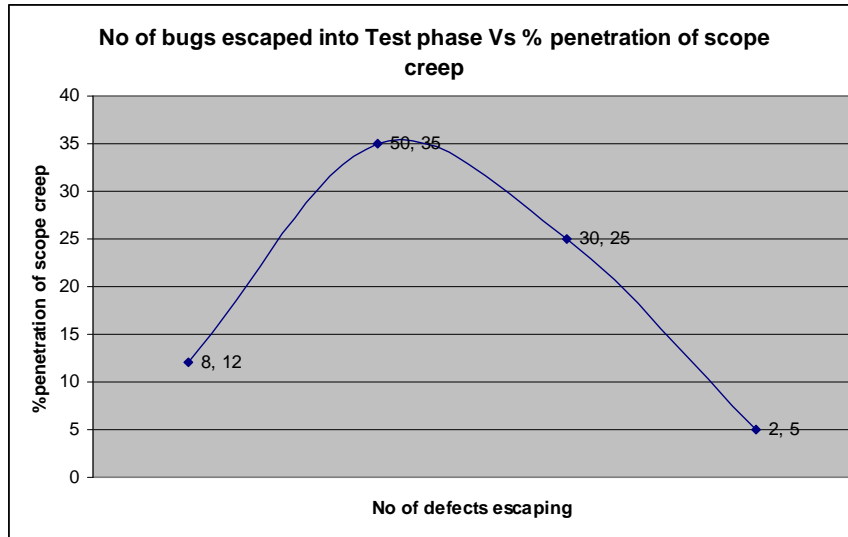


Figure 2: Sample Graph- No of bugs escaped into Test phase Vs % penetration in scope creep

In an ideal case, there should not be any scope creep in iteration. This activity of understanding the effect of scope creep within iteration is important, as would give an indication to the agile team if the duration of their iteration is optimum. As this duration is a function of the functional clarity for the iteration, business value creation and team’s ability. Understanding scope creep (if any) would help us understand which of the above mentioned variables is causing it.

### 3.2.5 Cost of Quality control

The measure of test effectiveness in BCCSL is the Cost Of Quality (COQ). COQ is defined as the total cost incurred to maintain the quality of the software until it is signed off from the customer. Testing is the last of the total efforts to reduce cost of quality. Defect containment at every phase of the software development is an effective COQ reduction strategy. TSP-PSP(team software process – personal software process) has proved that defects escaping from one phase to the next phase in software development would more than linearly increase the effort required to contain it in that phase. A defect containment strategy needs to be put in place in this model, as the defect escaping would be of more deterrent effect than it would be for normal software projects. The best way would be to

- a) Clearly divide iteration lifecycle into requirement analysis, design, development, code review/inspection, testing, and bug fix. Some of these stages can be moved in and out of the regular iteration, but would all exist in the project as a whole.
- b) Have defect containment levels set for each stage. Here, methods from TSP-PSP can be used where defect containment effort can be set and tracked. This conscious effort is to make sure that each member of the team is part of the COQ control.

- c) Test Driven Development: The use of test driven development can be very advantageous in controlling the COQ. The philosophy is to use extensive unit testing to effectively contain the defects in programming phase. In an ideal scenario, all unit test cases are actually written before the code is developed. The finished code will have to pass through all the unit test cases to be deployed onto the development/test servers.
- d) The phase called Code review/inspection is required to contain as many defects as possible in the development phase.

All these efforts may be viewed by the team/ customers as additional effort, but the it has been proven that effort spent on each defect containment prior to testing phase is always less than to contain/fix in the test phase and it is even worse that the defects escapes into production.

### ***3.2.6 Regression strategy***

Regression testing has sizable effort consumption in Agile projects. As development iterations complete, the effort spent on regression testing increases (until automation testing counter-balances it). But the core difference in BCCSL when compared to other agile projects is that the regression strategy is not just regressing the entire iteration, but to first regress the functionality of critical chain candidates in each iteration. Also the strategy should be such that, each regression suite includes the critical chain components included. As the stability increases, if dropping of some test cases is part of the strategy, then all such test cases which belong to non critical chain components of a stabilized iteration can easily be removed from the regression suite. One of the criteria for making a regression suite is make to balance the risks and rewards. As part of that only a minimum set of test cases are identified to cover X% of the product thus giving Y% of confidence that the product is not broken. This selection of test cases and the X, Y% should now also take into account the critical chain candidature. The confidence Y% would be higher if in X% there is more critical chain components. The aim is not only to reduce the risk of the software failing but also to make sure that the critical chain components do not fail.

### ***3.2.7 Automation strategy***

Automation is of core importance in BCCSL. In a general agile project, automation strategy is designed in such way that, automation starts at about 50-60% of the total project timelines and the depending factors to start automaton are analysis of the % reduction in manual effort and the number of former iterations which are now stable and automatable. These factors would still be of crucial importance in BCCSL, but the additional variable of critical chain candidature is also taken into account. The functionality that are ready to be automated and give almost the same % reduction in manual effort, automating the functionality which is critical chain component increases the leverage point to make sure that the critical chain is testing as much as possible(in regression) with as little effort as possible. In cases of highly stabilized iterations, that only requires part of the test cases to be regressed; only critical chain components may be automated.

The ramp of automation team is also important. The automation engineers would be part of the requirement analysis, design. This helps them better understand the components they need to automate, create efficient reusable scripts, also help design/UI teams to choose components that are easily automatable.

### **3.2.8 Test risk Management**

Traditionally, Risk management has always been part of the project management processes. Having a separate test risk management is advisable for projects which use models like BCCSL, as the risk of inefficient quality assurance is very high with high consequences. Ideally all testing related risks have to be identified at the start of the project (at the test plan creation stage). There could be risks of testability of the critical chain components which would not be required to be identified and mitigated in regular agile projects. All regression, automation related risks need to be properly evaluated and registered. All operational risks possible in testing phase in iterative mode have to be registered. A separate test risk register with all possible risks and mitigation strategies have to be maintained through out the project. A periodic check on the criticality of risks has to be analyzed and the risks re-prioritized as per current situation of the project. It is advised that the effectiveness of the mitigation plans of major risks need to be verified by creating virtual scenarios.

### **3.3 Ideal scenarios**

Implementing a new lifecycle will always come with an overhead. Both in terms of effort required adapting to new philosophies/ principles and the effort required to implement them successfully. BCCSL is no exception from this overhead. It is of foremost importance to understand where such models need to put in place and where traditional lifecycle or traditional agile processes would suffice, or sometimes better suited.

For software consulting companies which wish to cater to customers using BCCSL, the judging factors are

- a) Whether the project teams have been exposed to agile lifecycles in the past and did teams in the past successful in implementing these projects with some amount of customization? This gives the teams ability to adapt to these philosophies more easily.
- b) Are project teams working in agile methodologies displayed in the past, the ability to negotiate with the customers by providing facts and alternate options? These negotiations(of scope, of effort and technical options) are important in agile methodologies, as the parameters keep changing frequently and ability to quickly access, adapt and provide alternate options is critical.

For software clients who want potential vendors to provide software application/product development through outsourcing or partnership, the judging factors are

- a) Are the business requirements/ business roadmap to be built through application/product development? If so, is the quality/ SLA of that application/product is of critical importance to business?

- b) Whether these customers have ever been exposed to agile software development and worked with vendors/partners in custom agile models with part of the management team involved in day to day tracking and negotiations?

#### 4 Citations

- **Book**

PSP, A self improvement process for software engineers, Watts S. Humphrey, Addison-Wesley, Pearson Education

- **Book Chapter**

Chapter 7 , Chapter 8 , Chapter 10

#### References

None

## Biography



### **Gautham Reddy**

Gautham Reddy: Working with TCS for 2.5 years on various projects from a process angle. I have played the role of a SCRUM master for large healthcare development project for a reputed US customer. WON TCS young innovator(Yi)award for 2007. Venugopal Reddy: Working with TCS for the last 11 Years in various capacities like Process, Sales and Delivery. I have worked on different types of projects ranging from Support, conversion, migration and development. Worked with Waterfall model to Agile methodologies. Managed large testing engagements.