Engaging Testers Early and Throughout the SDLC

Mark L. Gillenson, Xihui Zhang, Sandra Richardson
Department of Management Information Systems
Fogelman College of Business and Economics
University of Memphis, Memphis, TN 38152
{mark.gillenson, xihui.zhang, sandra.richardson}@memphis.edu

Abstract

Early in the history of software development, testing was confined to testing the finished code. However, as the practice of software development has evolved, there has been increasing interest in expanding the role of testing upwards in the SDLC stages. This paper presents a set of models for expanding testing throughout the SDLC. It includes comparisons of the models as well as personnel requirements for each one. It also addresses the roles of the testers at the various SDLC stages.

1. Introduction

Computer software is a kind of product that, like other “manufactured” products, must be developed in stages. New application software has to be conceived, it has to be justified, the development must be found to be feasible within stated parameters, its requirements have to be analyzed, it has to be designed, the design has to be implemented, the implementation has to be tested, and the software has to be maintained over time. We have learned over time that collaborative and cross-functional development teams are very helpful in new product development. There are many examples of collaborative product design, development and manufacturing in traditional manufactured goods. One, in which manufacturing engineers were brought into the new product design process was recounted in [23], in a discussion of camera design and development.

Just as traditional manufactured products can benefit greatly from having representatives of all of the groups that will be responsible for their success at the various stages of development, participate at all of those stages, so, too, can computer applications. In particular, we suggest that personnel from the systems testing organization should be part of the new software development process at all stages and for two major reasons and these are the two guiding propositions in this proposal. First, they can lend their expertise to “testing” or evaluating the “product” of each application development stage, whether it be a set of requirements, a software design, or a program. Second, they can influence the program design and implementation to make the job of testing the finished code more manageable and more effective, which will be to everyone’s benefit. The sum of these will have the additional effect of having testers see themselves as stakeholders in producing high quality, defect-free products rather than just being the people who find bugs.

2. Literature review

Software development and testing has been with us for decades. In the early days, writing a program was as restricted to analyzing and coding [14]. There were no professional testers involved, and testing was usually performed by developers or users to ensure the code ran correctly and met the expectations. Eventually, software development life cycle models came into being. Life cycle models divide the process of software development and testing into well-defined phases or stages, typically including analysis, design, coding, testing, and implementation [10]. Testing originally emerged as a distinct phase, and in some organizations, began to be carried out by professional testers. Testing was limited to the testing of code, after the code development was finished.

More recently, there has been a trend in software development and testing to engage testers earlier in the development process [13], [17], [22]. Also, there has been a further realization that testing is indispensable to ensuring software quality [2]. Furthermore, integrated testing early in the life cycle has the potential to catch and fix defects early in software development. This practice is very rewarding because fixing a defect
The software development and testing life cycle models include sequential models (e.g., waterfall model, traditional V model, PV model), progressive models (e.g., phased model), iterative models (e.g., spiral model), and agile models (e.g., XP). The waterfall model includes a set of sequential phases that flow downwards like a waterfall. The traditional V model has the same set of phases as that of the waterfall model. However, with the traditional V model, each testing activity is mapped to some development activity [8]. With both the waterfall model and the traditional V model, testers are involved only in the traditional testing phase, and testing doesn’t get started until coding is completed. This makes it very difficult to go back to previous phases when changes have to be made or new features have to be added. Thus, using the waterfall model or traditional V model tends to cause a project to run over time and over budget [17].

The PV model is similar to the traditional V model. However, the PV model separates each testing activity into two parts: test planning and test execution. Test plans (i.e., test specifications) are developed along with each development activity, and the test execution is exercised in reverse order after coding is done [15]. The PV model tries to engage testers earlier in SDLC, and compared to the traditional V model, it has the advantage that test plans can be completed early, during the development phases, and therefore the overall testing time can be shortened.

The progressive model, the iterative model, and the agile model are all iterative and incremental approaches to software development, which require full integration of testing and development [16]. Their intent is to produce high quality software in a cost effective and timely manner, and in the meantime, meet the changing needs of the end users.

In a field study of thirty-two business application systems, McKeen [10] found that “systems which spent more time in the analysis phase required less time to code, resulted in greater user satisfaction, and were developed in agreement with established budgets and deadlines” (p. 47). Bringing testers into the analysis phase has the potential to aid in this endeavor. Waligora and Coon [22] presented quantitative evidence that, by starting testing earlier in the development life cycle, project performance, in terms of cost, cycle time, and quality, was improved without sacrificing quality.

Cusumano and Selby [4] noted that “since the late 1980s, Microsoft has used variations of the synch-and-stabilize approach to build Excel, Office, Publisher, Windows 95, Windows NT, Word, Works, and other products” (p. 60). Two of the major features of the synch-and-stabilize approach are 1) product development and testing is done in parallel, and 2) there are frequent synchronizations (daily builds) and intermediate stabilizations (milestones). These two features require “[teams] of programmers and testers [who] frequently synchronize and periodically stabilize the changes they make to products in progress” (p. 53). Pyhajärvi and Rautiainen [13] viewed testing as the means to maximize customer satisfaction and provide feedback for process refinement, in addition to finding defects. They argued that testing is “an integral activity in software development” and they recommended that “testing should be included early in software development.”

Vijay [21] provides us two very convincing reasons why we should bring testing earlier into software development. (1) “Testing activity consumes 40% of the whole project time.” If we want to cut down the cycle time of the SDLC, the testing phase is a “more lucrative timesaving phase as compared to [the rest of the] SDLC.” (2) Some of the testing activity has to wait until coding is complete, however, “around 60-75% of the activities (like the test-plan, test design) don’t require any code.” Thus, if we can bring testers earlier into the SDLC, let them work on the test plan and test design, this would dramatically decrease the total cycle time of SDLC.

In another relevant vein, there is literature on manufacturing engineers being involved in new product development. For example, Nemetz and Fry [12], in talking about flexible manufacturing organizations, describes manufacturing personnel as being part of a multifaceted team. Milgrom and Roberts [11], talking about flexible manufacturing organizations, says, “… products are developed by teams composed of designers, process engineers, and manufacturing managers.” Several books have been written on new product development, including [3], [19], [20], and [23]. In particular, as mentioned earlier, Wheelwright and Clark [23] describe the development of the Kodak FunSaver I single-use camera. Sample quotes from this description include, “Concurrent with these decisions about product line architecture, the development team began to focus on issues of manufacturing.” “Design and manufacturing tasks had to be carefully integrated.” “To deal with these challenges, a manufacturing engineer was co-located with several parts designers and a tool designer, all of whom were dedicated to the project.” “… and part manufacturing people could easily gain early access to part and tool
design decisions, enabling them to estimate costs and identify and evaluate automation options.”

The broader issue is that of cross-functional or matrix organizations. Ford and Randolph [5] provide a good survey of the literature up to that point. Griffin [6] said, “… the use of cross-functional teams, is one of the factors that has been derived through a grounded theory approach by multiple researchers as purporting to decrease new product development cycle time.” A sampling of other papers on this topic include [1] and [7].

There is one more basis for this work of which we are aware. Jordan Lewis’ book Trusted Partners [9] addresses the idea, at several levels, of interpersonal relationships being a key factor in a successful business enterprise. The concept in the current proposal would be not only for the testers embedded in earlier stages of the SDLC seeing themselves as stakeholders, but also for the analysts and designers in the earlier stages seeing the testers as “trusted partners” who can help them in their work and add value to the outputs of those stages.

3. How to engage testers early and throughout the SDLC

There can be variations in the stages of the SDLC as practiced by different companies. For example, the first stage may include business justification, feasibility in several dimensions, and determining information requirements. The analysis stage can be taken to varying levels of detail. The design stage may be split between code design and database design, or it may be one unified process. In fact, the stages of the SDLC can vary in name, in number, and in content to suit the needs of individual organizations. Generally, different SDLC variations are ultimately similar in their effect. For our purposes, we will define five SDLC phases or stages, as follows:

- **Business Requirements:** The set of specifications of what the business unit expects the application to accomplish.
- **Software Requirements:** The systems analysis stage in which the business requirements are translated into graphical formats that show process and data flows.
- **High-Level Design:** The specification of the code modules and their functions, and the flow of data among the code modules.
- **Detailed Design:** The design of the functions within each code module.
- **Implementation:** The programming of each code module.

Having established an SDLC framework with which to work, we now turn to the nature of the participation of the testers at each SDLC stage. This breaks down into two possibilities, as we noted earlier, one or both of which can be practiced. One, which is applicable at all of the stages, is the idea of testing the output of the stage. Do the requirements make sense and do they meet a set of accepted standards (and what are those standards?)? Do the diagrams that result from the systems analysis stage flow correctly and make sense? And here again, have the systems analysis diagrams been constructed to meet accepted standards and what are those standards? Are the program design specifications and the database design acceptable?

The second possibility regarding the nature of the participation of the testers is directed toward the creation of software that will lend itself to being more easily and effectively tested. This would begin in the systems analysis stage and engage heavily in the systems design stage. This could have profound implications in software testing in streamlining the process in general and in specifics such as determining the test datasets to use.

There are a number of factors to consider in formulating a model to use in integrating testers into the application development team and process. An initial factor is whether a company believes in embedding testers at all levels of the SDLC. As we have stated, we believe that the arguments for doing so are compelling and so for our purposes we will assume that this is the case.

Moving on, then, one factor in deciding which model to use is the skill sets of the individuals in the testing department or organization. We will assume that any tester assigned to represent the testing organization in an application development project is skilled in testing in at least one of the application development stages. This can even be extended back into the education and training backgrounds of the individuals. It is as difficult to imagine a tester without a business background leading a business requirements review, as it is imagining a tester without a programming background leading a code review. But, pursuing this further, the Tester Embedding Model chosen will also depend on whether the company’s testers tend toward breadth of skill or depth of skill. Are the individual testers expected to have skills that range from business to technical skills or are their skill sets expected to be narrowly focused?

Another factor is the amount of resource the company is willing to invest in testing. This certainly will depend on the company’s commitment to testing and on the size of the application development project.
Generally, in this regard, more would seem to be better, however even a company that takes testing seriously would not want to overwhelm the application development teams with testers.

With the foregoing as background, we propose five “Tester Embedding Models” for embedding testers throughout the application development process.


As the name implies, in The Single Tester Model, one tester is assigned to an application development project and stays with it through all of its stages. This has the advantage of continuity as the one person begins learning about the project from the very beginning of the business requirements phase and continues building her project knowledge through each successive stage. The disadvantages of this model include a project over dependency on one person and the expectation that the one person must be well-versed in a breadth of skills ranging from requirements analysis to program design and programming. Assuming you can find such a person, if she gets sick or leaves the company, the project is left in the lurch. And, if The Single Tester Model is attempted without a sufficiently broadly skilled tester, then clearly principle of having strong testing expertise at each development stage will have been defeated.

3.2. Model 2: “The Specialist Model”

If The Single Tester Model is one extreme, then The Specialist Model is the other extreme. In The Specialist Model, a different, highly specialized tester works on the application development project in each of its stages. Presumably, each tester is a true expert in the work being done at their particular development stage and thus the advantage is the testing expertise that can be applied at each stage. Conversely, each tester does not have to have a broad skill base. However, there are some disadvantages, too. One disadvantage is that each Specialist must learn the nature and details of the project when they cycle onto the project. Another disadvantage is the lack of communication between the testers in the different stages. Indeed, it is this lack of communication that inspires certain aspects of the next three models.

3.3. Model 3: “The Leapfrog Model”

The Leapfrog Model is designed to overcome some of the problems associated with both The Single Tester Model and The Specialist Model. The Leapfrog Model begins with Tester A, who is a requirements testing specialist, working on the Business Requirements development stage. Tester A continues working on the project in the Software Requirements stage, where she is joined by Tester B, whose expertise is more geared towards systems analysis and high-level software design. As both Testers A and B work on the Software Requirement stage, Tester A is able to gradually transfer her project knowledge to Tester B. At the end of the Software Requirements stage, Tester A leaves the project. At the beginning of the High-Level Design stage, Tester B is joined by Tester C, whose expertise is focused on both high-level and detailed software design. Similarly, at the end of the High-Level Design stage, Tester B leaves the project and at the beginning of Detailed Design stage, Tester C is joined by Tester D, whose expertise is focused on detailed program design and programming. Tester C leaves the project at the end of the Detailed Design stage. An advantage of The Leapfrog Model includes having two testers working on each development stage except for the first and last stages. At each of the intermediate stages there is the opportunity for one tester to gradually transfer his project knowledge to the next tester. However, as with The Specialist Model, The Leapfrog Model assumes the availability of a stable of relatively specialized testers and, with two testers involved at each of the intermediate stages, it is even more resource intensive.

3.4. Model 4: “The Balanced Bifurcated Model”

In The Balanced Bifurcated Model, there are two testers, A and B. Tester A has a broadly-based systems analysis background that extends to requirements on the one end and to high-level software design on the other. Tester B has a programming background that includes the higher levels of software design. In this model, Tester A begins at the Business Requirements stage and stays with the project through the High-Level Design stage, after which he leaves the project. Tester B joins the project at the High-Level Design stage and continues with it to its conclusion in the Code stage. With both Testers A and B working together in the High-Level Design stage, they have the opportunity to transfer project knowledge from A to B. Their skill bases must be broader than those of the testers in either The Specialist Model or the Leapfrog Model, but not as broad as the testers in The Single Tester Model. While there is a shift back toward the problem of over-dependence on individuals as in The Single Tester
Model, there is also not as much of a resource drain as in either The Specialist Model or the Leapfrog Model.

3.5. Model 5: “The Unbalanced Bifurcated Model”

The difference between The Balanced Bifurcated Model and The Unbalanced Bifurcated Model is the point of hand-off of responsibilities. The principle of The Unbalanced Bifurcated Model is that one tester, Tester A, will work with all aspects of the project through and including the Detailed Design stage. Then, Tester B will join him in the Detailed Design stage and be responsible for testing in the Coding stage. This model heightens the personnel dependency issue, plus Tester A must be very broadly based. The clear advantage of this model is it provides specialized testers whose sole purpose and total focus is to look at the detailed design and then work in the highly technical pursuit of code testing.

There are two possible considerations that could lead to variations in some or all of Models 1-5. One is that as a practical matter, depending on the size of the IT organization, the scope of the development project, and the company’s dedication to testing, the testers described in any of the five models may well serve as “test leads” and bring additional testing personnel into the application development process as required. In all cases, this could obviously be simply a matter of handling the volume of work at hand. Also, in all cases, this decreases the dependency on only one or two people at any given development stage. Naturally, it also increases the amount of resources expended in testing. In addition, in Model 1, The Single Tester Model, and to a lesser extent in Models 4 and 5, the two bifurcated models, it could involve calling in specialists to supplement the skills of the test leads in those models.

The other consideration has to do with the overlap between the testers at the various intermediate stages of development. Models 1 and 2 have no overlap. If we eliminate the overlap in Model 3, The Leapfrog Model it effectively reverts to Model 2, The Specialist Model. The issue is whether in Models 4 and 5, the two bifurcated models, the overlap could be eliminated. This has the advantages of reducing the expenditure of resources and of not requiring the testers to be quite so broad in their skill sets. On the other hand, there will clearly be a cost in losing the transfer of project knowledge from one tester to the next that is provided by the overlap. Perhaps the elimination of the overlap is most appealing in Model 5, The Unbalanced Bifurcated Model. Eliminating the overlap in Model 5 would isolate Tester B who is responsible for code testing. Historically, this responsibility has in many organizations been the only real job for testers. Even in those instances in which testing is embedded throughout the SDLC, the nature of code testing and its specific and highly technical techniques make it a candidate for being largely separable from the other SDLC stages. In fact, as a practical matter in today’s IT environment, code testing can be looked upon as a candidate for outsourcing, which would fit the variation that we might now call The Unbalanced Bifurcated Model Without Overlap.

4. Conclusions and further study

We believe that software development is fertile ground for the use of cross-functional teams, with testers being integral members of those teams at all stages of development. Clearly, work must now commence on further refining our five suggested models and in testing them in some manner.

Overall, the goal is higher quality application code with fewer errors produced. All of this will lead to higher quality applications being introduced into production. An additional contribution will be testers seeing themselves as stakeholders in the quality of the finished applications by virtue of their work throughout the SDLC. This will lead to the further development of systems testing as a recognized and respected specialty within information systems organizations.

5. References


