Testing in the eXtreme Programming (XP) Paradigm: A Case Study

By Marnie L. Hutcheson

Abstract

The application in this case study is a new Web application developed for a Fiduciary Trust company. The company had been assured by their eXtreme developers that testers were unnecessary in this Web project. Since the risk of project failure would be measured in millions of dollars in lost business, someone thought it a good idea to risk a few hundred on a second opinion. This turned out to be a good risk for them and a really good opportunity to measure the cost of not testing.

This paper gives the account of what happened when this application went on line, and presents a review of the test log, and the bugs reported. Because of the nature of the project and the type of failures that occurred, this project provided a unique opportunity to calculate the cost of not testing and the value of testing in terms of customer support dollars that would have been spent without testing and the that were saved because of testing.

Introduction

In order to compete in today's market; it is a business imperative to develop the Web. Not only must a company move their existing products on to the Web but they also have to develop new products directly for the Web. Market pressures in the global Internet environment are intense; there is a constant pressure to produce cheaper, faster, and smarter software. This trend will continue to escalate over the next few years as the world moves into mobile Internet computing.

We are all being forced to re-evaluate the processes we use to create software products. The "heavy-weight" or traditional approaches to producing quality software are largely too cumbersome and too expensive to be competitive in this environment. --And they do not ensure that the end product will satisfy the users. The "light-weight" approaches, descendents of the Rapid Application Techniques, RAD, called Agile approaches, are geared to be very responsive to the needs of this environment and as a result have had more success producing software for today's Internet. Agile approaches in general and eXtreme Programming, XP, in particular are focused on finding new more efficient ways to deliver high-quality software, fast.

As budgets and schedules shrink, everyone in the development effort is feeling pressure to demonstrate the value that they add to the product. Testers in particular are being asked to show how they add value to the development process and how much that value is worth to the company. XP takes this situation to the extreme by asserting that testers are unnecessary and even counter productive. –That they do not add any appreciable value to the product. This paper gives an account of in informal test effort conducted on a recent real world XP project.

Besides providing a good look at how an XP project delivers, it also provides a good opportunity to measure and quantify the value that testing added to the project.

Background on eXtreme Programming (XP)

The Internet lends itself to small lightweight applications by its very nature. The current trend is to write very small modular components that perform a particular task and call them as needed. These program components may be spawned from vastly different architectures and may be called different names by different vendors; I will call them Web Services. Web Services are based on standards set forth by the W3C, including; html, XML, SOAP, and UDDI.

XP evolved in this standards based, small application production environment. It is a “small project” methodology, it’s not known to scale well (6-12 developers is considered ideal). As such it is ideally suited to Internet development projects. For this and several other reasons, XP has many adherents among Web developers.

XP is an iterative development methodology that is based on 12 basic tenets. The source for this listing is Kent Beck’s book: eXtreme Programming Explained: Embrace Change

1. Customer is at the center of the project
2. Small releases
3. Simple Design
4. Relentless Testing
5. Refactoring (adjust code to be improve the internal structure, make it clean, simple, remove redundancies etc)
6. Pair programming
7. Collective ownership
8. Continuous integration
9. 40- hour work week
10. On-site customer
11. Coding standards
12. Metaphor - development is guided by a story view of how the system will work.

The adoption of these methods is often driven from the bottom up by developers who are frustrated by the constraints placed on them by the more formal processes associated with traditional plan driven approaches to developing software. XP has quickly become very popular in development communities as a way to maximize their efficiency and produce products that satisfy their market.

As with any new approach there are strong points and weak points in the process. Several of each are illustrated in this case study. Before presenting the study I want to discuss some of the fundamental differences between the traditional and the agile approaches.

Plan-Driven Versus Agile Methods

Agile and plan-driven teams use different approaches to design and implement software. The two most important differences in this particular project are;

1. The difference in spending priorities

2. The absence of testers

The Difference in Spending Priorities

The plan driven methodology believes in spending up front to acquire information that will be used to formulate the design, so that they are as well informed as possible before they commit to a plan. This is called *money for information* (MFI).

The Agile method chooses the design early and modifies it as they go. So they reserve budget to help them adjust and absorb the changes and surprises that will certainly come. This is called *money for flexibility* (MFF).

Spending money for information, or MFI, is generally considered to be part of the plan-driven management strategy. Reserving money for flexibility, or MFF, is considered to be part of the RAD/Agile management strategy (see Figure 1).

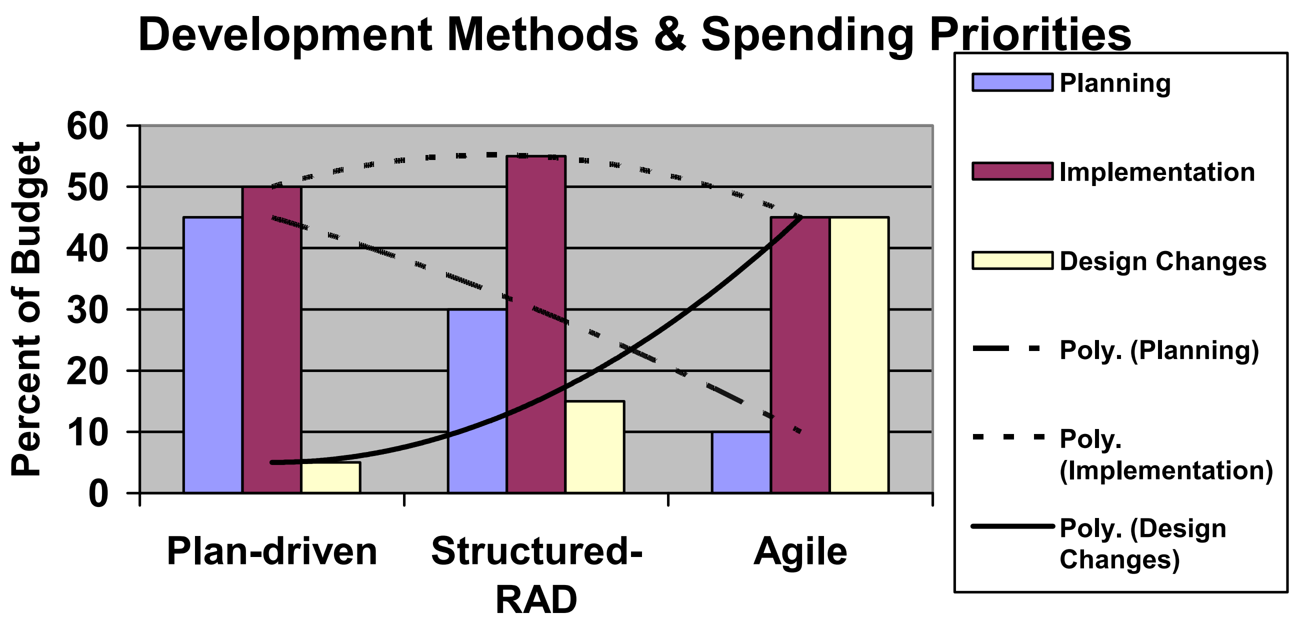


Figure 1: Development methods and their spending priorities

The plan-driven management is willing to spend money for information, (MFI) to mitigate risk. When the outcome is uncertain as with new technologies, or systems MFI is used for unpredictable and the "all or nothing" scenarios. For example; when we don't know actual usage patterns, or system limits. These are things that can be measured or simulated in advance of development. That way there is time to change course before the project fails.

The problem in developing Web applications is that the fast pace of development and the fluid nature of the Internet create an environment with **too many unknowns**. Companies using traditional approaches typically find themselves over budget, behind schedule, and worst of all, they often fail to produce software that satisfies the users. After all, how can static plan with static goal hit a constantly moving target. Too often it is the case that by the time the application is implemented and rolled out, the needs have changed and the product is not satisfactory.

The Agile approach implements in small increments, produces a functioning product or feature set as fast as possible, evaluates it, refines it and iterates through the process again. –And the product evolves as it goes. A functioning product, albeit a limited one, is available from very early in the development process. By reserving funds for flexibility, the Agile effort is gambling that they can fix those last minute problems. This flexibility was a key element of the success of the project in this case study.

The Absence of Testers

Even though item 4 of the XP tenets lists “relentless testing” as one of the basic tenets; it does not say “who” will do this testing. Apparently this item is open to interpretation. The first XP project I was involved with cast me in the double role of Tester and Technical Writer. It worked very well and was the subject of the paper, The Team with the Frog in Their Pond (1999). In that project each feature team consisted of a developer, a tester and a business partner (the customer). I was completely taken by surprise to hear that testers were “unnecessary in this 401k Web application XP project. All testing was done by the developers, and the customer. There was no end user testing, no integration testing, nor was there any process flow or structural testing performed until I began working on the project.

The 401(k) Web Project

This was a Web-based application that would provide 401(k) retirement plans for self-employed small-business people. Up until this new product line, the benefits of a 401(k) retirement plan had been the exclusive province of large businesses. Not only had there not been any opportunity for self-employed persons to have a 401(k) program, but there was also a significant number of now self-employed persons whose 401(k) retirement plans had been frozen in the 401(k) plan of their last major employer. People with their 401(k) plans in this situation couldn’t contribute to them or borrow from them. These assets were effectively unavailable to their owners until they rolled them into an IRA, which could accept contributions but from which they could not borrow. The other option was to close the fund and remove their money, and pay a significant penalty and income tax on the amount.

This new product would allow self-employed people to set up and administer their own 401(k) plan. It would also allow them to transfer in their lame duck 401(k) holdings, contribute to it, and borrow against it. The response to the plan offering was expected to be huge.

The new product was created for the Fiduciary Trust (the customer) by an Internet Service Vendor, ISV who also hosted the application as the Application Service Provider, ASP. The end user is the individual who subscribes to the plan.

Testing Criteria

I became involved with this project as a potential customer, not as a hired tester. As a self employed Web consultant the Web based 401k product was very appealing to me. But I switched from customer to “Professional Tester” before I got past the first screen. Since it was my money, and there were no other offerings of this type available, and I decided to pursue the opportunity to test this application and report my findings in order to see what would happen.

I was not given any instructions or description of how the process worked. I approached the project armed only with my ID, my newly issued password, my Internet-connected PC, and a blank document open in Word. I used the MITs test methods, a risk based test methodology described in my new book Software Testing Fundamentals. Since I wasn’t given any specific requirements, I adopted the fundamental risk requirements for all fiduciary dealings:

- No money can be created or destroyed (or lost or prematurely withdrawn).

The main risk if any of these conditions should occur can include financial instability, audits by government agencies, criminal proceedings, and high-stakes litigation. I used this one requirement and these risks to rank the bugs that I found.

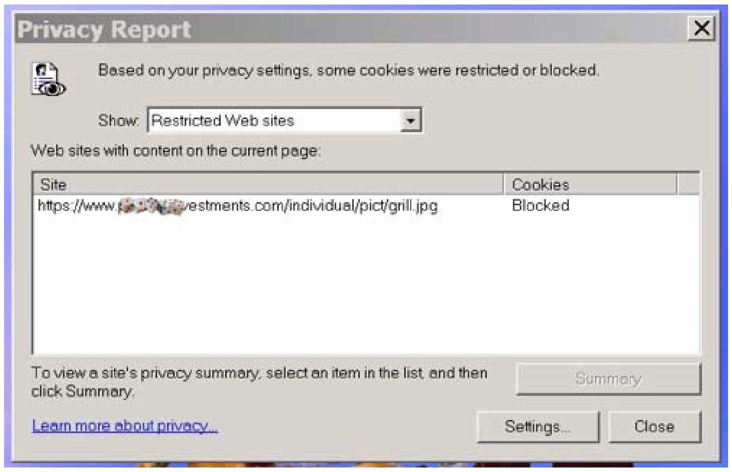
Errors Reported While Testing Plan Enrollment

My first test was to log on and enroll in the plan. Within a few hours I had identified three fatal errors in the application and two fatal errors in the process. In four hours of testing, I logged 20 issues. Of these, 15 required a call to customer service to resolve so that I could continue, and 5 were of a nature that would cause the customer to lose confidence in the institution. I found all of these issues simply trying to enroll in the program.

1. Could not log in on **www.maincompany.com/individual401k** - cookie error on redirect.

   Problem Description: The main corporate web site was using cookies to redirect the subscriber to the application which was hosted by the ASP / developers. These cookies were rejected by both the IE 6 and IE 5 browsers when set at the normal default security settings. These two browsers represent 70-80 percent of all web traffic to all the web sites hosted by Ideva. That means that probably **70 – 80 percent of all potential users would fail to log on – every time**. The disclaimer on the log on window states that the application is optimized for the Netscape browser. Statistics from the last half of 2002 show that all versions of Netscape amounted to less that 10% of all traffic to these sites.

   The logon failed to complete because it violated the security settings in the browser; creating an endless logon loop where the logon succeeded but the no other page could be displayed. This caused the system to redisplay the logon page. However, no error message was displayed. The only clue to the actual events was a small icon displayed in the bottom border of the browser. --Which was not an obvious source of information. Some exploration of the browser policy and security settings revealed the actual problem, see Figure 2. Non-technical users would have been completely stymied – and would have failed to log. See the sidebar on first time user failures in Calculating the Value of Testing below.

   Privacy Report

   Based on your privacy settings, some cookies were restricted or blocked.

   Show: Restricted Web sites

   Web sites with content on the current page:

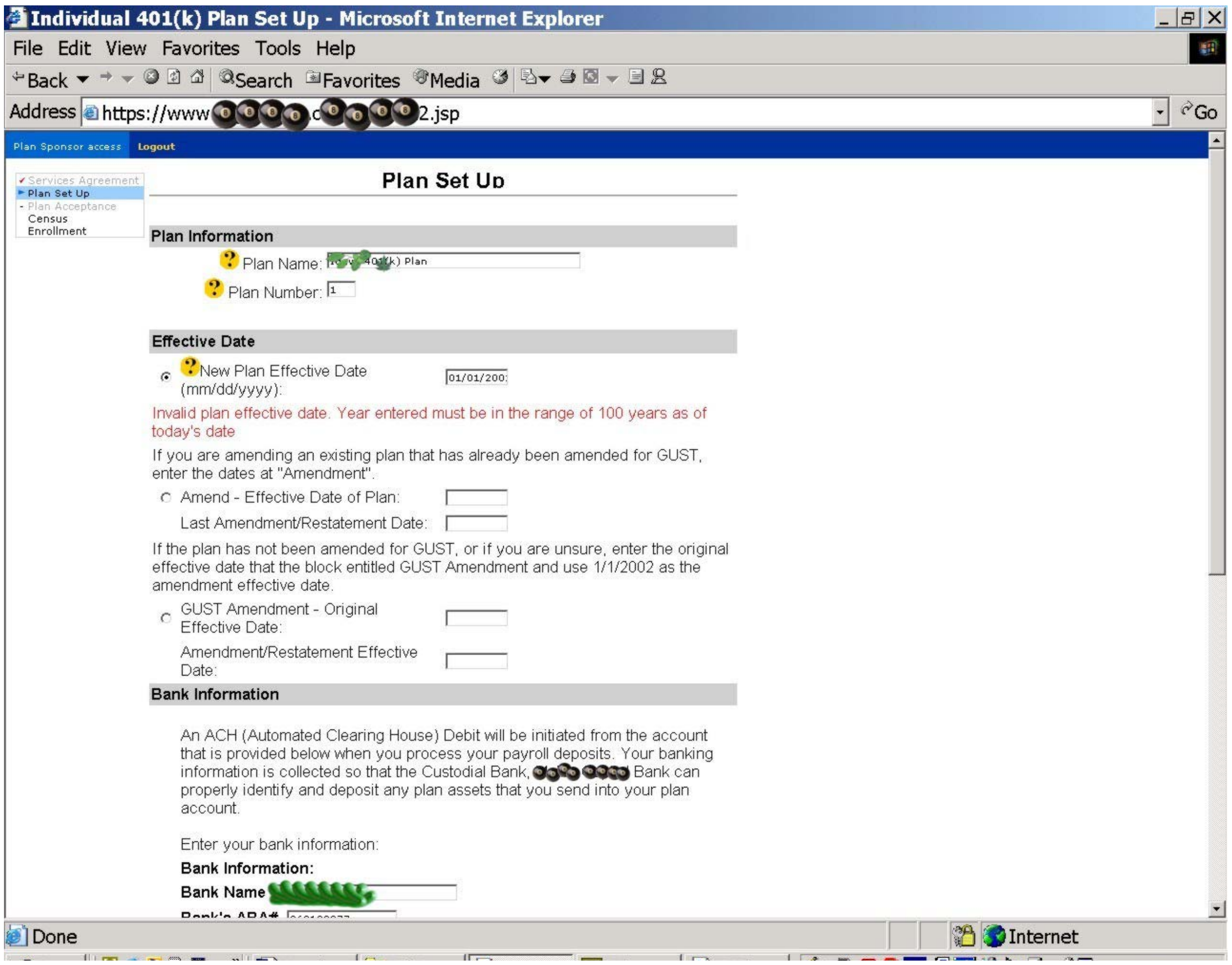
Site	Cookies
https://www.[redacted]vestments.com/individual/pict/grill.jpg	Blocked

   To view a site's privacy summary, select an item in the list, and then click Summary.

   Figure 2. The logon failure

   The workaround: the ASP was not using these cookies, so it was possible to navigate directly to their logon URL and log on to the application. The problem still persists on the main web site.

NOTE: Three months later, the problem still existed on the main web site. The lead developer at the ISV tried repeatedly to get the companies IT personnel to fix the problem but with no success. Apparently the core issue was related to support for the Netscape browser over IE.


Individual 401(k) Plan Set Up - Microsoft Internet Explorer

Address https://www[redacted]2.jsp

Plan Sponsor access Logout

✓ Services Agreement
► Plan Set Up
- Plan Acceptance
Census
Enrollment

Plan Set Up

Plan Information

Plan Name: [redacted] 401(k) Plan

Plan Number: 1

Effective Date

● New Plan Effective Date (mm/dd/yyyy): 01/01/200

Invalid plan effective date. Year entered must be in the range of 100 years as of today's date

If you are amending an existing plan that has already been amended for GUST, enter the dates at "Amendment".

○ Amend - Effective Date of Plan:

Last Amendment/Restatement Date:

If the plan has not been amended for GUST, or if you are unsure, enter the original effective date that the block entitled GUST Amendment and use 1/1/2002 as the amendment effective date.

○ GUST Amendment - Original Effective Date:

Amendment/Restatement Effective Date:

Bank Information

An ACH (Automated Clearing House) Debit will be initiated from the account that is provided below when you process your payroll deposits. Your banking information is collected so that the Custodial Bank, [redacted] Bank can properly identify and deposit any plan assets that you send into your plan account.

Enter your bank information:

Bank Information:

Bank Name [redacted]

Bank's ABA#

Done Internet

Figure 3: The Plan Setup page

2. Plan Page Interface Problems -See Figure 3
   2.1. “GUST” not explained in paper instructions but is clarified in online.
   2.2. Several fields are to short to display all the required input info, i.e. the date field in the screen pix. - – notice the error message about the year needing to be within 100 years of today's date.
   2.3. Type face too small for age > 40 – the font size was set to 8 points, and could not be changed by the user.
   2.4. Bank ABA # in the online interface is called ACH R/T on bank checks (bottom of screen), required a call to CS to get the translation.
   2.5. No way off page – There is no continue or next or back button on this page.
3. Plan setup page function and capability problems

3.1. Downloaded the 61 page PDF and machine locked up during printing. This error repeated several times, it seems to be contention between acrobat reader and outlook. Later analysis showed that it the lock up was due to mal-formed coding introduced by the application logic as it encoded the agreement into PDF format. No agreement created with these errors could ever print. Further, both the document, stored in memory and the online session were lost when the machine locked up.

3.2. After rebooting and coming back to the site, it was no longer possible to get back to the plan setup page, so could not get another copy of the plan agreement.

3.3. After the lock up and reboot, the next time I logged on to the plan setup, the system took me to the Census page. There was no way to get back to the plan setup page to complete the set up process, and no indication that there would be an opportunity to agree later in the process. The system took me to a new set of pages, and the outcome of the agreement was unknown.

3.4. **The system had erroneously decided that I had agreed to the terms of the plan**. -- But I had never succeeded in getting to the agreement page, because of the problems with PDF locking up the PC and prematurely ending the session with the Plan setup application.

3.4.1. The plan document is 61 pages, not likely that anyone is going to read it and agree in the same session – but there is no way to reenter the set up process.

4. During enrollment, allocation selection is conducted using percentages, but the allocation field doesn't accept the % sign. The error message talks about "whole numbers" only. - user confusion will occur.

5. I entered in my beneficiaries’ social security numbers with dashes between number groups. The system (gave no instructions for or against, and no data example was presented) inserted more dashes when I submitted the page. The resulting social security number was corrupted.

5.1. Next, I couldn't edit the beneficiaries’ social security number. The solution was to delete and re-add the beneficiary.

5.2. There is no way off this page

6. Information windows don't close and don't have a button or link. And they don't come forward when a new topic is selected. They remain behind the browser, so it looks like the help function is unresponsive or absent.

7. There is no closure on the plan setup process. After census, you can enroll, then there is no wrap up, and since I never agreed to anything, the process has failed. I can't find a way back to log on as a plan owner either. So how can I administer the plan? (The top menu has only two options, Participant access and Log out.) The options in the left menu are now all geared to the employee, not the employer. The developer tells me that I should have an option in the top menu to switch to the administrative role.

8. The best system issue: Both the Company site and the ASP site disappeared from DNS in the second week of testing.

9. The best process flow issue: the bank in charge of processing the funds, both contributions and rollovers, lost the rollover check, because they didn’t know how to process it.

10. MISC. The administrator’s instructions were completely out of date and actually belonged to a different product, so following the instructions in the administrators guide caused serious errors in the users plan.

Results Reporting

Most of the issues were first reported over the phone, but I also prepared a comprehensive written report, which was widely circulated among the developers and the agents. This document was used as the agenda in several conference calls between the developers, customer service, and me. All the reports that I submitted were also circulated to the project team at the fiduciary trust.

I created an inventory of the features that I tested; the logic flow map of enrollment (not presented in this paper), and a bug report complete with pictures. The logic flow map clearly showed two of the fatal errors. These could have been detected in a design review. The report also contained my cost / savings analysis attributable to the test effort. It was the cost / savings estimate that got the most attention.

Calculating the Value of Testing

There were two areas where the value of testing could be demonstrated in this effort. The first is in terms of the cost to the CS departments for helping each customer through the enrollment process. (This neglects the rest of the application and the bugs most probably lurking there.) The second way to show the value of the test effort is in terms of customer retention with respect to the logon process. I will discuss the CS savings first because it was very well received by management. Budget and potential or unexpected overruns are something that managers seem to understand vividly, while “lost customers” or “potential lost customers” seem to be a bit less “real” to them.

Cost Savings to CS as a Result of the Test Effort

The thing that makes this effort so important for demonstrating risk and its associated costs is that all of these issues were 100 percent reproducible. They were going to occur during ***every*** enrollment. So ***every*** customer was going to experience them. Consequently, calculating the value of testing or the risk of not testing in terms of dollars saved was doable, and the results were impressive and defensible.

Developers have long disputed this kind of cost savings statement on the basis that you can’t be sure how many users will be affected. If you have to wait until you have collected actual usage stats from production before you can justify an estimate, this usually takes months and by that time, it is hard to get a hearing.

I kept logs of my time spent testing, talking, and reporting. The total came to 8 billable hours. For this analysis, I first calculated the cost of the test effort. There were no expenses, so the invoice was for time only:

The cost of my testing: 8 hours X $125 per hour = $1000

The next step was to calculate how much cost was incurred by the company for supporting the end user, me, through this process. I had spent 215 minutes on the phone with the ASP and the trust company customer service accomplishing the enrollment process. Their support costs for call-in customers on a per-minute basis were estimated at $5 per minute. This figure includes all the costs associated with handling a call—floor space, telecommunications, salaries benefits, and so on. (Note: this estimate is very conservative, larger organizations support significantly higher costs per minute in their call centers.) This gives a typical cost per (computer-literate, Web-savvy) customer of:

215 minutes per customer X $5 per minute = $1075 per customer

So, they would probably make back their investment on testing the first time a customer did not have to call customer service to get through the process of setting up their plan. This estimate doesn't take into account all the potential customers who would not become customers because of the number of issues they would have to resolve in order to set up their plan. Nor does it take into account the ones who would simply fail, give up, and go away without calling customer service. --More on this topic in a moment.

Next, I asked how many customers they expected to sign up in the first month. (I ignored the cost to fix the bugs and assumed for the sake of estimation that all the bugs I found would be fixed within the first month.) The company had planned a mass mailing to 40,000 targeted prospective customers. They expected a record response to the long-awaited new plan, so estimates went as high as 20 percent response to the mailing, or 8000 customers for the plan. (A 5 to 10 percent response would be the normal estimate.) So, I calculated a range from a 5 percent response, or 2000 customers at the low end, to a 20 percent response, or 8000 customers at the high end.

For simplicity, I will round off the estimated customer support cost per customer to get through the plan setup stage (calculated in the preceding text) to $1000 per customer. The savings in customer support alone, due to testing this one feature, is somewhere between

$1000 * 2000 customers = $2,000,000 and

$1000 * 8000 customers = $8,000,000

Note: Conversely and conservatively, the risk of not testing is somewhere between $2 and $8 million *in the first month*.

If no testing is done, this range represents the budget that they had better have ready for customer support if they want the product to succeed.

The User's Response to First Time Logon Failure

Logon failure statistics gathered in 1993, 1997, 2000, 2001, 2002 the first time user's response to logon failure, see Table 1. The "Other" category originally was for users that fell through the cracks and couldn't be accounted for. Starting in 2000 the "self help" web sites and automated password reissues became common and have had profound impact on user satisfaction, they are listed in the "other" category.

Year	Network	Leave and don't come back	Call Customer Support	Email Customer Support	Other
1993	Prodigy	50%	45	0	5
1997	Internet IE4	60	5	30	5
2000	Internet e-business	40	15	30	20
2002	Internet e-business	40	15	15	30
2000	Intranet	40	25	25	10
2002	Intranet	20	20	30	30

Table 1: Study results of the user's response to first time logon failure over time

The User's response to first time logon failure is possibly the single biggest reason for startup failure in new e-business offerings in the late 1990s. The statistics show that in 1997 over half of all the users who failed in their first attempt to log on to the product or service, left and never returned. The data collected in 2002 suggest that this number has been reduced to about 40%, this seem to be largely due to the improved automated online self-help web sites available now with almost all online services. Typically, the user can ask for a new id and or password by filling out a form, and perhaps answering a security question. Typically, the new information is automatically sent to the user via email. The Pie graphs in Figure 3 illustrate this change in user behavior between 1997 and 2001.

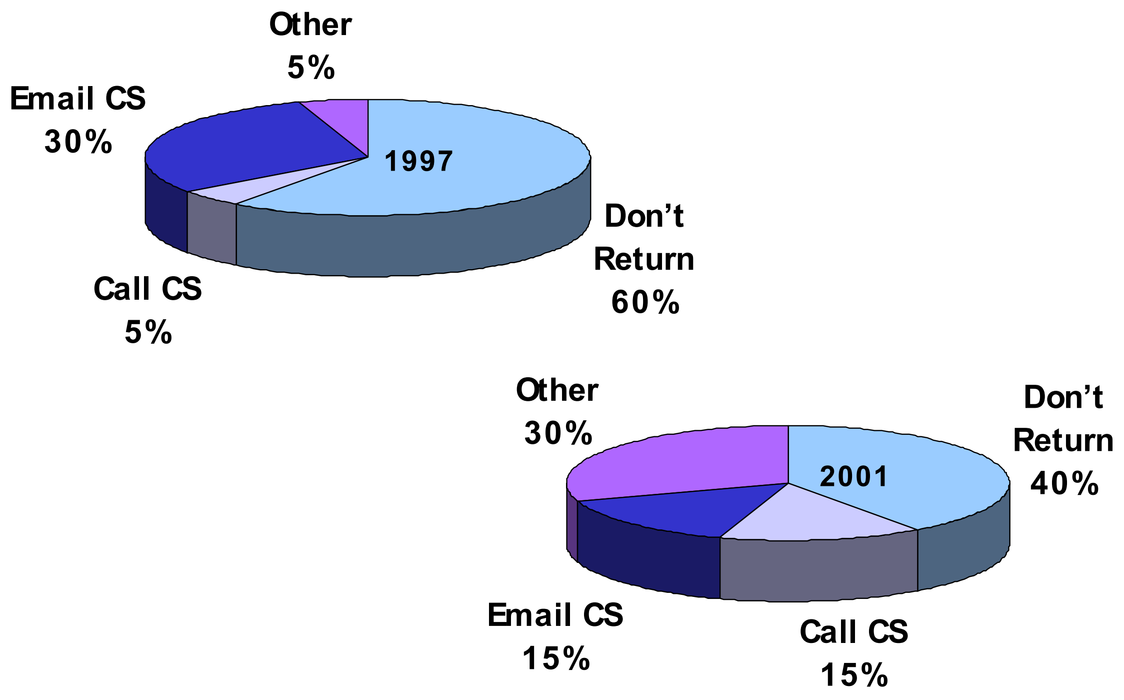


Figure 3. Change in user behavior between 1997 and 2001

Some Thoughts on This Project

This was a small application. I walked into it cold and discovered many places where it was broken or disconnected, and I reported them to the maker. There was no planning and no estimation done. I just started exploring. The effort certainly saved the company millions of dollars at the very least. But this is not the best way to conduct a test effort nor is it the best way to ensure that testing adds value to the project.

The first problem is that in a larger project this would not be possible. While I could manage the complexity of this one function in my head and in my notes, in a larger project, there are too many complexities to succeed using this method. Like the hero Theseus in the Maze of the Minotaur, I need a string to help me find my way. This lifeline is a set of best practice risk based testing methods.

The next point is that this was an ad hoc effort; a one-time deal. Again, ad hoc efforts don't work on a large scale. --Which brings me to the next problem, the fact is that the work that I do is based on years of disciplined professional practice, and defensible risk based methods using solid measurement. Management only saw the results of this discipline, not the years of careful exercise that made it possible. They seemed to be left with the impression that anyone with a modicum of experience could produce a similar set of results. This is not the case, and it is this shallow interpretation of the test professional that has led us to the current situation where developers can say that testers are not necessary and not adding value to the product, --and be believed.

The biggest thing wrong with this whole scenario is that it gave management the wrong message, even though it was unintentional. For some reason, they seemed to believe that even though I only tested one-tenth of the application, I had discovered most or all of the bugs and they were now safe to proceed with the deployment. So they were very happy with the savings provided by the test effort, and they deployed the marketing materials that would bring in the main stream customers.

Without a test effort to warn them of these problems in advance, the company would have discovered these issues in public and in volume. Yet they had not thought it necessary to spend even 1 percent of this potential loss on a test effort to prevent it. This type of decision is typical of the I feel lucky approach to software development that I talked I first described in 1993 in Software Testing Methods and Metrics. The I feel lucky approach is characterized by an entrepreneurial management, with a shallow understanding of the technology, making decisions based on their hunches, rather than real data, or research. This decision is also proof that the I feel lucky approach to software development is still very much in use today.

Why does eXtreme Programming work?

There are several reasons that XP projects can be deemed, “successful”. Not the least of which is the fact that in this brave new Internet world, no one is measuring how much business was lost because of the failures in the software, so any apparent success adds to the bottom line, and few of the failures are counted at all.

If XP succeeds in fielding a viable product it is due to the MFF strategy that goes along with this approach, and the full dress rehearsal given to the software while the first users test it. The successful XP projects that I have seen keep a full contingent of developers at the ready when they roll out the software to a select few users. This gives them the lead time that they need to fix major issues before too many people see the thing. And the users doing the testing get a feeling that they have a very important status, after all, the developers are changing the application to suit them. This generates good will, and gives them a stake in the product.

This contrasts sharply with the Plan-driven approach where the developers who wrote the code have invariable been assigned to their next project as soon as their code is complete, leaving no one who is familiar with the code to fix the bugs. This is especially true and problematic in large integration projects where integration issues are not discovered until well after the code has been turned over. The time required to fix the bug and the cost to fix the bug are both significantly higher than a bug fixed by the original author. Meanwhile, release is delayed, or the application is released and the users become can become disenchanted while waiting for fixes.

Summary

After close inspection, I have decided that though the XP tenets are laudable, they will most likely become the mantra for the latest iteration of the “I feel lucky” approach to software development. There are a couple of weaknesses that I believe will cause many XP projects to fail. First, they don't contain any pledge or commitment to ethical behavior. Nor are there any balancing activities to ensure the delivery of a functioning product. –In short no fundamental metrics are built in, only subjective judgments. Which brings me to the second weakness…

There is an implicit assumption in the tenets that the customer knows what they want and that they are willing, able, and most of all; qualified to know what the product should do and what features it should have in order to achieve that goal. --This is rarely the case. Users expectations are vague and non specific, and worst of all, these ideas vary from user to user. A design environment dominated by the user's expectations and the developer's virtuosity must be strictly managed to prevent scope creep, feature bloat, and other impossible conditions that I call the kid in a candy store syndrome. I find it hard to imagine management successfully anticipating and controlling such an environment without the kind of feedback that a professional test effort provides.

XP's predecessor, the RAD approach, at least pretended to do formal testing before letting the users do the real testing. XP uses a frontal assault on the concept of testing, asserting that the method is so good that no professional testers are necessary. --Which means that no time or effort is wasted before turning it over to the end user who will begin testing it in earnest, whether they mean to or not.

Even if some developers are adamant that no professional testers are required in an XP effort, there will be those more cautious and mature among the sheep who will be willing to risk a bit on an evaluation. The lead developer in my this XP project said this to me:

> "We really need testers on the Internet, people have been writing code and sticking it out there for years without testing anything."

Managers at the ASP and at the trust company didn't originally believe that a test effort by a professional tester would be worthwhile. Until my report, the value of such an effort was a mystery to them at best. It is a sad fact that something that can make such a profound difference to the bottom line of a start up operation as a "test effort" could be so thoroughly discounted and misunderstood. It is my hope that this experience will help other testers to demonstrate the value of their efforts, and that they will pursue every opportunity to demonstrate this value.

References

Agile development methods. See AgileAlliance. "Agile Software Development Manifesto." February 13, 2001. www.agilemanifesto.org

Beck, Kent and Martin Fowler. Planning eXtreme Programming. Reading, Mass.: Addison-Wesley, 2001.

Boehm, Barry. "Get Ready for Agile Methods, with Care." IEEE Computer, January 2002.

Hutcheson, Marnie. Software Testing Fundamentals, NY, NY: Wiley, May 2003

Hutcheson, Marnie Software Testing Methods and Metrics, **www.ideva.com**, 1993-1997

TestersParadise.com – resources and discussions for testers.

Notes on Terms

The "I feel lucky" approach to software development. Term coined by the author, in Sortware Testing Methods and Metrics, 1993 and also discussed in Software Testing Fundamentals, 2003.

Plan-driven. Term coined by Barry Boehm in his article, "Get Ready for Agile Methods, with Care" to describe traditional waterfall-style development methods.