What I've Learned...

From Failure

by Reginald Braithwaite

What I’ve Learned From Failure

A quarter-century of experience shipping software, distilled into fixnum bittersweet essays

raganwald

This book is for sale at http://leanpub.com/shippingsoftware

This version was published on 2014-10-08

Tweet This Book!

Please help raganwald by spreading the word about this book on Twitter!

The suggested hashtag for this book is #shippingsoftware.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#shippingsoftware

Contents

Preface

The chapters of this book originally appeared as online essays and blog posts. I decided to publish these essays as an e-book as well as online. This format doesn't replace the original online essays, it's a way to present these essays in a more coherent whole that's easier to read consecutively. I hope you like it.

–Reginald "Raganwald" Braithwaite[1], Toronto, Christmas 2011

copyright notice

[1] http://braythwayt.com

[2] http://creativecommons.org/licenses/by-nd/3.0/

About this sample

This sample edition of the book includes the first full chapter, “What I’ve Learned From Failure.” The full book adds chapters about customer-centric design (“The Not So Big Design”), models of project management (“Which Theory Fits The Evidence?”), product-oriented development (“Trial and Error with a Feedback Loop”), and more.

I hope you enjoy the sample. Remember, there’s a money-back guarantee on the full book, so please do read it from cover to cover.

the price of the book

> Legend has it that Pablo Picasso was sketching in the park when a bold woman approached him.
>
> “It’s you, Picasso, the great artist! Oh, you must sketch my portrait! I insist!!” So Picasso agreed to sketch her. After studying her for a moment, he used a single pencil stroke to create her portrait. He handed the women his work of art.
>
> “It’s perfect!” she gushed. “You managed to capture my essence with one stroke, in one moment. Thank you! How much do I owe you?”
>
> “Five thousand dollars,” the artist replied. “B-b-but, what?” the woman sputtered. “How could you want so much money for this picture? It only took you a second to draw it!”
>
> To which Picasso responded, “Madame, it took me my entire life.”

What I've Learned From Failure

> I have been fired from more jobs than most people have had.–Mark Cuban

Why does failure matter?

It's a funny thing. After more than twenty-five years of drawing a paycheque for creating software, people generally want to hire me because they want me to duplicate the successes I've had. The model seems to be "Just do the things you've done successfully before, and you'll be successful now."

My experience is that this has never worked on its own. Success in software development is at least as much about avoiding failure modes as it is about "best practices." I conjecture it's because software development on a commercial scale is so hard that almost any mistake will sink a project if left uncorrected or even worse, actively encouraged.

> We tend to seek easy, single-factor explanations of success. For most important things, though, success actually requires avoiding many separate causes of failure.—Jared Diamond

With that in mind, I've taken a little time to jot down some thoughts about situations where I've personally failed. I'm not going to tell you about some theoretical anti-pattern, or relate some broken thing I've fixed, I'm going to share things that caused me to leap from the deck of a burning boat to avoid drowning.

> If you decide to run with the ball, just count on fumbling and getting the shit knocked out of you, but never forget how much fun it is just to be able to run with the ball.–Jimmy Buffett

Some of them, in retrospect, would be comical if it wasn't for the human misery, damaged careers, and money wasted on failed projects. Or worse, in my opinion, the opportunity cost of putting good people to work on things that never end up delighting the world. I weep for what might have been.

The four most important causes of failure

> Things which matter most must never be at the mercy of things which matter least.—Johann Wolfgang Von Goethe (1749-1832)

The first thing I've learned from failure is that the four things which matter most are:

1. The quality of the people doing the development
2. The expected value of the product to its stakeholders
3. The fitness of the proposed solution

4. The quality of project management and expectations communication

In my experience, you need all four working to have a successful project. I've personally failed when even one of those four things was bad and not corrected immediately. If two, three, or all four were wrong, my discovery is that I've been unable to avert disaster. (This list obviously doesn't cover all of the factors needed for business success: I'm just talking about getting the software to ship).

Now that I've learned this, I have four new things to evaluate when placed in charge of a new project. And regardless of what I'm told, I'm going to investigate these four things every time, right away, without fail.

I've never seen a project where strength in one area made up for weaknesses in others. I've never personally seen a great technology platform, for example, that magically enabled low-quality developers to produce commercial-quality results.

And don't talk to me about XP being a magic bullet: all of the good XP teams I've seen happened to have quality developers, a valuable objective, decent technology, and yes, good project management.

People

> I think the root of your mistake is saying that macros don't scale to larger groups. The real truth is that macros don't scale to stupider groups.–Paul Graham on the Lightweight Languages mailing list

I've been involved with strong teams and weak teams, and the weak teams always failed. Weak teams have individuals whose performance is weak. The strongest indication of a weak team is the realization that if you were to quit and start your own business, you wouldn't try to poach any of your colleagues.

Painful experience has taught me some of the signs that a team doesn't have the chops to perform up to par. The first sign of a weak team is poor hiring practices.

Developing software is a difficult job. It requires a panoply of strengths. Hiring good people is never as simple as interviewing three people with "five years of J2EE" on their résumés and making an offer to the best of the three. Strong teams have almost impossibly high hiring standards. Strong teams will always leave a desk empty rather than settling for less than the best.

Another sign of a weak team is poor development hygiene. There are dozens of development practices that seem trivial to the inexperienced outsider or to the manager focusing on "big wins." Examples of development hygiene include source code versioning, maintenance of an accurate bug or issue database, significant use of automated testing, continuous integration, and specifications that are kept current (whether incredibly detailed or high-level overviews).

One team I audited were not just unwilling, but were actually unable to build a product that was in sustaining development. In other words, the product was in the field, in use by customers, and the team were not able to rebuild it from source. They were issuing all of their bug fixes as patches on their existing binaries. This was not a good sign.

(c) 2003 Steve Gregory

Does this mean that nothing can be done if the team is weak? Not exactly. Some of the time I've had the authority to replace members of the team. I've always had the ability to set an example and suggest practices. But sometimes I've thought that an organization would be unreceptive to calls for change. And for want of courage, projects have been lost.

The bottom line is that when I've failed to recognize weakness in the team and/or failed to take immediate and decisive action to bring the team up to world-class strength, I've failed.

> Argue with idiots, and you become an idiot.–Paul Graham

> If you compete with slaves, you become a slave.–Norbert Weiner

I've mentioned that I've failed with weak teams. Would you believe I've compounded this failure by failing with weak stakeholders? Whenever I've had stakeholders who didn't have the horsepower or the will to recognize that a project was in trouble, I've wound up in the E.R. having the brick dust removed from my forehead.

> A chicken and a pig decided to open a diner together. The pig asked the chicken what they should call their new restaurant. The chicken suggested "Ham and Eggs." The pig thought about it for a while, then decided she didn't want any part of the venture. "You," she told the chicken, "would only be interested in serving breakfast. I'd be committed."–as told by Ken Schwaber

Getting away from weak teams, another source of failure is the omnipresent threat of "chickens." A chicken is not necessarily a weak individual, but a sign of a weak management structure. A chicken is an individual who has significant authority over your project, but does not make a personal commitment to the success of the project. Significant authority includes the authority to impose constraints on the team.

Even a single chicken can take a project out. Chickens are a special case of "external dependencies." Special, because they are often politically entrenched. I've worked with teams where the pay scale was determined by an edict from H.R. They were literally prevented from hiring top talent, and it wasn't a question of budget: they did not have the freedom to replace three mediocre programmers with two good programmers for the same price.

Another situation involved a team that were continually pestered to include functionality and architecture for "strategic" reasons by a Business Development person. Although senior management made the importance

of the strategic functionality clear, they were unwilling to relax tactical requirements like the ship date or the target revenues. They had to constantly manage the "chicken" in order to succeed.

I've managed around chickens here and there, but I've failed to deliver a successful project whenever I've failed to limit the effect of chickens on the management of projects.

Action

> Always dive down into a problem and get your hands on the deepest issue behind the problem. All other considerations are to dismissed as "engineering details"; they can be sorted out after the basic problem has been solved.–Chris Crawford

The next thing I've learned from my failures is something familiar to the test-driven development crowd. It's mandatory to fail early. You need to know you're in trouble right away. That's essential when taking over an existing project or starting something new. You have to find out how you're doing within weeks. Not quarters, not months. The longer you wait, the more inertia the failure will have.

You have to come in, take over, and establish some incredibly short-term goals and be prepared to take action based on the project's performance. I've learned that there's no such thing as too little time between milestones. Looking back at projects where I've failed, many contained some uncertainty or risk that I didn't address immediately.

(c) 2010 Jim Pennucci

In one case there was a critical piece of functionality that was so important the entire architecture was designed around it. The CTO and every developer swore it was the greatest thing since sliced bread. I made a back-of-the-envelope risk calculation and scheduled testing of that functionality to begin three months before the project was to be delivered.

A week before the function was to go into test, the technical lead informed me that it didn't work, had never worked, and that no attempt to fix it would work, because there was a major, glaring flaw that had been overlooked. A rewrite would be required.

The stakeholders agreed and appointed a new team to handle the rewrite. Needless to say, the old team's job security suffered a major hit.

Today, older and wiser, I would demand immediate proof of feasibility of all critical pieces of the product, no matter how obvious things may be to everyone else. I should have said "Great! It's a slam dunk! Wonderful, let's schedule a demo next week." At least we would have felt the pain early.

Details

Whenever I've allowed the details of a project to escape me, I've failed. On one project, the technical lead was a Ph.D. and refused to describe his work, saying that although I was the managing product development, he wasn't going to try to explain his rarified code or architecture to a layperson.

> No, I'm not responsible for what happened. I'm accountable for how we dealt with it, but I'm not responsible for it.–Julian Fantino

Needless to say, I was unable to ship a successful project. On another project the CEO would ask me the same question every few days: "draw on my whiteboard who's working on what." I had no trouble with this on that particular project, but looking back there have been projects where I was not tracking people's work on a day to day basis.

And I can tell you, whenever the details of a project have slipped from my grasp, the project has started to drift into trouble. I make no apologies for now insisting on knowing exactly who, what, where, when, and why. There's a big difference between being asked to explain your work in detail and being told how to do your job.

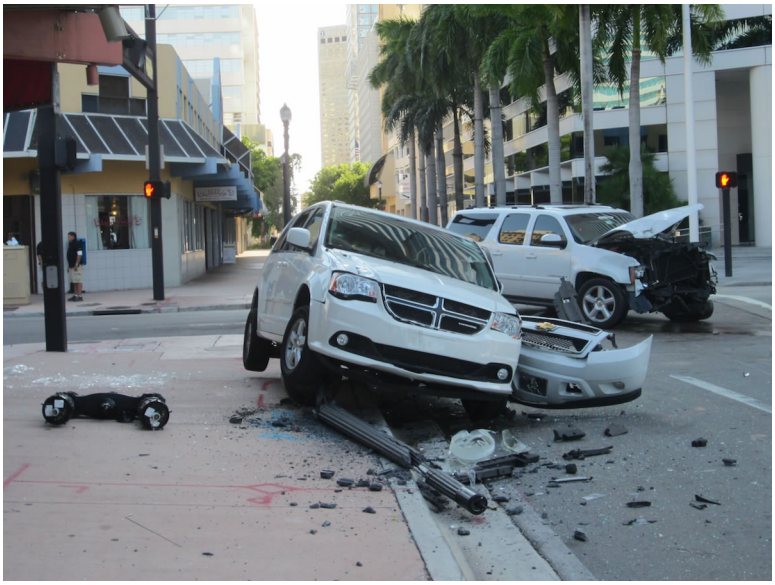
My personal experience is that attention to detail has always accompanied successful projects: Losing track of the details has always accompanied failing projects.

The Schedule

> In most companies if a good quality project ships late then the managers will still get it in the neck whereas if poor quality project ships on time then the managers say "we did our best - obviously the dev team seem to be of a poor standard".–Daniel H. Steinberg

Dates are sacred. I've learned this lesson in good times and bad. Stakeholders treasure good dates. Stakeholders despise bad dates and the people who make flawed promises. That would have been me, more than once.

Every time, the lesson has been clear. Don't get the dates wrong. I'll confess: I don't really think Scrum is an order of magnitude more effective than anything else at producing beautiful, world-changing software. It may be worse. But it does produce software every month, month after month.



(c)2011 Phillip Pessar

And every time I've delivered software on schedule milestone after milestone, my influence and standing with stakeholders has grown. And every time I've missed a date, I've suffered, regardless of whether the late software was demonstrably better than what was originally planned for the missed date.

> If documents don't serve to avoid stupid things, mitigate risks or calculate budgets then what are they for? They're to show you have a "process" and a "paper trail" so that you can get ISO certified. That's all they look for, they don't care if you read them or not.–Skagg on http://discuss.fogcreek.com/joelonsoftware/

Back to the measurable processes. I've learned from failure that stakeholders like to know what's going on. I hate producing useless documentation. The net result is that I've tried to find the happy medium where I generate weekly management reports on projects.

A management report is something that is used to actually make a decision. Everything else is garbage. I've learned that when I haven't had management reports for a project, failure has resulted. Worse, sometimes I've had documents and metrics that were used to justify bad decisions that sank the ship.

So my lesson from these failures is that every project needs a set of regular reports that contain information you'll actually use to make decisions.

Software

> Few projects are cancelled because their designs and implementation weren't complicated enough. Many are cancelled because they become so complicated that no one can understand them anymore; any attempt to change or extend the system results in so many unintended side effects that continuing to extend the system becomes practically impossible.–Steve McConnell

One of the reasons people associate me with "Agile" development approaches is that I'm always trying to simplify, simplify, simplify. This is because almost every time I added something to a milestone, I've gotten burned. It seems like it's always better to say "just finish what we planned, get that 100% functional, and then we'll add foobars."

I recently got burned twice in the same project adding functionality in between milestones. Both times I was sure that the changes were low risk. Both times I burned myself. Now I'm licking my wounds and swearing I'll never, ever break my agile principles of restricting scope changes to between increments.

> It's important to remember that when you start from scratch there is absolutely no reason to believe that you are going to do a better job than you did the first time. First of all, you probably don't even have the same programming team that worked on version one, so you don't actually have "more experience". You're just going to make most of the old mistakes again, and introduce some new problems that weren't in the original version.–Joel Spolsky, "Things You Should Never Do, Part I"

Here's something that I've screwed up repeatedly. Sometimes I've bounced back, sometimes the project has paid the ultimate price. The grand "this time we'll get it right" mantra is absolute garbage.

> It had taken 3 years of tuning to get code that could read the 60 different types of FTP servers, those 5000 lines of code may have looked ugly, but at least they worked.–Lou Montulli, one of the founding engineers at Netscape

Don't talk to me about porting to Java, or new design patterns. If you must refactor, refactor here, and there, and there to solve this, and that, and the other specific problem that has a specific feature or bug attached to it. And show me that you had 100% unit testing coverage on the affected code and completed each refactoring in a day or so and then ran all the unit tests and got a green light.

If you can't do that, you're going to fail. I know it, because I've failed when I didn't do that. And when I cried on a friend's shoulder, he told me "I also made that mistake once, and I suffered the same horrible fate."

Thinking that a major rewrite is going to solve all of your problems is just revisiting my four things that matter most and planning on having one, the fitness of the proposed solution, overpower defects in the people, expected value, and process. It won't happen.

A major rewrite should produce a major new product that offers an order of magnitude more expected value. And you'll need to be 100% sure your team has the horsepower to get the job done and is going to use a process that can handle the load. I say this because I've tried and failed to rewrite entire applications, and I've taken over other people's rewrite projects and failed there too.

Power

> Some days you are the bug, some days you are the windshield.

I've learned a little about politics from failing. What I've learned is that if you stick your neck out and evangelize change, you will be blamed if you do not achieve results. You may or may not care about that. But be aware of the fact that making changes involves spending your personal credibility. If you don't want to lose it, don't ante up: get out of the project.

> Don't have good ideas if you aren't willing to be responsible for them.–Alan Perlis

And if you decide to make changes, have the courage to go 100% with your gut. I've failed more than once when I watered down my convictions in order to appease dissenters. The only thing worse than evangelizing change and failing is looking back and realized you might have succeeded if you'd held firm on your convictions. What a waste!

> Making an employee work and profiting from that work are two different things.–Eliyahu Goldratt

I've seen a number of "sweat shops," and I've worked in several places where long hours and rhinoplastic intimacy with the grind stone were demanded of the team. I can honestly say that hard work makes no long-term difference to failing software development teams.

I disagree with those who say that long hours are 100% detrimental to software development: I've seen lots of situations where people worked around the clock, motivated by passion. But those were successful projects.

I've learned that redoubling effort when a project is in trouble has not fixed the project. The conclusion I draw is that although teams have worked long hours on many successful projects, there is no causal relationship between long hours and success (It's another example of the fallacy of "best practices": copying a single element of a successful project does not guarantee that another project will be improved).

My experience with failing projects is that the exhortation "Ahh, I'm going to have to go ahead and ask you to come in on Sunday, too…" has always been interpreted as punishment, not a meaningful way to fix the project. It has had no effect on under-performing members of the team and tends to strongly demotivate the people who are pulling more than their fair share of the weight.

> It is impossible to sharpen a pencil with a blunt axe. It is equally vain to try to do it with ten blunt axes instead.—Edsger Dijkstra

Good luck convincing stakeholders of this. One of the reasons people love to hand out overtime like candy is that hours in the office are measurable. The team's behind? Make them stay until midnight every night. Even if it doesn't work, the executive handing out this order can be sure she can measure compliance.

The bottom line is, it's easy to measure how many axes you're using to sharpen a pencil. When you discover that a blunt axe isn't sharpening the pencil, how do you propose to measure "sharpness"? How do you measure "working smarter"? If we wish to count lines of code, we should not regard them as lines produced but as lines spent. Edsger Dijkstra

So another thing I've learned about failure is that when things start to go wrong, stakeholders want two things:

1. New processes
2. A way to measure compliance with the new processes
3. Overtime meets both criteria nicely, as do other simple panaceas like generating reports with every build or compliance with coding standards.

Fixing failing projects demands lots of things that are easy to measure and some that aren't. I've learned that if you don't control your stakeholder's expectations around change, you'll find yourself fending off demands for things like overtime and reports.

History

> Even when my proposals are seen as significant improvements, they are often rejected on the grounds that they are not intuitive. It is a classic Catch-22: The client wants something that is significantly superior to the competition. But if it is to be superior, it must be different. (Typically, the greater the improvement, the greater the difference.) Therefore, it cannot be intuitive, that is, familiar. What the client wants is an interface with at most marginal differences from current practice… that, somehow, makes a major improvement.—Jef Raskin

When I've been brought in specifically to "work out" a failing project I've failed when I didn't have the authority and support to make major changes. This is saying the same thing I've said several times already, but it needs to be repeated.

Often, the stakeholders have just finished casting someone into the darkness and think that they've cast failure into the darkness with him. Take a moment and look up the definition of the word "scapegoat." They may have symbolically cleansed the project of sin, but the sins remain, and I have inherited them whenever I've allowed the project to continue to do business as usual.

> The most damaging phrase in the language is, "It's always been done that way."—Rear Admiral Grace Hopper

I've heard dozens of variations on the same line (yes, I've failed dozens of times!) The line is "Well, so-and-so failed because he didn't x. But now you're here, you'll get x cleaned up, and we'll start succeeding right away. We were hardly failing, really, a little behind, nothing serious."

Every time I've been told that, things have ended up being seriously dire. No, it wasn't as simple as implementing monthly sprints or formalizing acceptance tests or nightly builds. The rot went right to the core and the stakeholders were usually (unwittingly) enabling it by not understanding or being in denial of the real problems. When people don't see the depth of the problem, they don't accept the importance of making changes.

It has boiled down to something so simple that you've probably heard it described in jest as the definition of insanity. If a project has been doing things a certain way, and the stakeholders are not 100% happy with the results, doing things substantially the same way will not produce substantial changes in the results.

Finishing

> There are two kinds of people in the world: those who finish what they started.

Getting back to failing early, I've learned it's important to completely fail. Get fired. Shoot the project, then burn its corpse. Melt the CVS repository and microwave the backup CDs. When things go wrong, I've often tried to play the hero from start to finish. Guess what? Some projects are doomed no matter what. Some need skills I don't possess. And some need a fresh face.

> The best way to fix a bad project is to not be part of it.—Norman Nunley & Michael Schwern

I've ridden more than one project down in flames, and as painful as it is to 'fess up and admit defeat, it's important to know when to fold your cards and quit. Yes, that sounds defeatist. But most success stories are comebacks from personal failures, not wondrous turn-arounds.

Sometimes you shouldn't finish what you started. Sometimes you shouldn't finish what somebody else started.

But if you avoid the four key causes of failure—people, value, fitness, and management—you will finish the software and it will be wonderful.

(Originally published in January, 2005. The bmx image is taken from http://www.flickr.com/photos/pennuja/5129137471/in/pho
The accident image is taken from http://www.flickr.com/photos/southbeachcars/6205032655/in/photostream/.
The race picture is taken from http://www.flickr.com/photos/gasheadsteve/131492751/in/photostream/.)

About The Author

When he's not shipping Ruby, Javascript and Java applications scaling out to millions of users, Reg "Raganwald" Braithwaite has authored libraries[3] for Javascript and Ruby programming such as Katy, JQuery Combinators, YouAreDaChef, andand, and others.

He writes about programming on [raganwald.com][http://raganwald.com] un-blog as well as general-purpose ruminations on braythwayt.com[4].

contact

Twitter: @raganwald
Email: raganwald@gmail.com

Reginald "Raganwald" Braithwaite

[3] http://github.com/raganwald
[4] http://braythwayt.com