How do you bring a failing project back on track? [closed]

Asked 16 years, 2 months ago Modified 7 years ago Viewed 4k times

20

Closed. This question does not meet Stack Overflow guidelines. It is not currently accepting answers.

This question does not appear to be about programming within the scope defined in the help center.

Closed 10 years ago.

Improve this question

You must have heard the archetypical story of a failing/failed project:

1. A team of inexperienced programmers work 24x7
2. Bugs are fixed only to introduce new bugs
3. Customer is screaming that he could not even do the basic stuff (Saving/Querying) etc.
4. Programmers used to having the spec handed down struggle to improvise
5. No automated unit tests aggravate the situation
6. Architecture document that looked nice on paper was not followed in practice

7. Third party components used become bottlenecks not having been tested for fitness in the first place
8. Milestone after milestone missed
9. The team is not able to come up with a delivery date as nobody agrees as to the quantum of work actually needs to be done
10. No technical leadership / or a Cowboy Coder that can take on the technical issues

Now, If you were to be brought in as #10 what would be your first steps?

Update: First of all: Thanks to you all for chipping in. Well... I'm being brought in as #10. I was the original Architect anchoring the solution when we made the proposal to the client. Then, unfortunately, I couldn't take on the delivery responsibilities as I was assigned somewhere else. :)

Let's say it's a webification of an existing desktop application. I'm now being brought in as #10. Running away, sadly, is not an option. I'm sure this can still be reversed by following agile best practices and just wanted to tap the community for ideas.

The larger question perhaps is this: If the development team does not have specs but only the (baselined) code for a running application, the original solution called for looking at the code and extracting business rules on the fly. Now, the inexperienced programmers are reluctant to

look at VB 6.0 code and want documents! So how do you fight this if you were to instate Agile processes?

project-management development-process

Share

Improve this question

Follow

edited Apr 5, 2012 at 16:03

casperOne

74.5k ● 19 ● 188 ● 259

asked Oct 24, 2008 at 17:51

Vyas Bharghava

6,510 ● 9 ● 42 ● 62

1 Many of us have been there. Some good advice below. Good luck. – WW. Oct 26, 2008 at 6:51

What language is "anchoring the solutioning"? – Jay Bazuzi Jan 11, 2009 at 1:17

@Jay: haha... "Anchoring the solutioning process"... Yeah... you've a better suggestion for 'solutioning'? Pl. feel free to edit and educate. Thanks. – Vyas Bharghava Jan 13, 2009 at 19:21

This question appears to be off-topic because it is about project management, which is no longer within the scope of the site. – Brad Larson May 31, 2014 at 17:50

20 Answers

Sorted by: Highest score (default)

23

Vyas, I feel like I could have written this question. My previous job involved resurrecting a KVM project that had failed after a year's development. Specs were in the form of a user manual and developers' experience with similar products. I ended up teaching C to 3 assembly programmers and re-architecting from scratch. We brought the product successfully to market in 4 months. (Then I resigned. Go figure.)

Some of the things I'd do again, particularly with an inexperienced team:

1. A team of inexperienced programmers work 24x7
10. No technical leadership / or a Cowboy Coder that can take on the technical issues

- Give them a (short!) break from the project to "recharge." Maybe a day, maybe an afternoon, or maybe a long lunch on you. It will mark the end of the "old" project and the beginning of success.
- Get their agreement to work their butts off when they return, and promise that you will be their go-to guy, cheerleader, and flak jacket. You, collectively, are a team, and your job is to forge their path, eliminate distractions, and lead them.
- Plan an immediate success, no matter how small, and maintain a "can-do" attitude.

8. Milestone after milestone missed
9. The team is not able to come up with a delivery date as

nobody agrees as to the quantum of work actually needs to be done
3. Customer is screaming that he could not even do the basic stuff (Saving/Querying) etc.

- **Take small bites!** Break each piece down as far as possible, then deal with the small components. You'll identify "gotchas" early and be better able to scope the whole project.
- **Define your interfaces.** Anytime you can isolate a chunk, **do it.** This allows parallel development, because you've already decided on parameters, preconditions, assumptions, what happens inside, and return values. You can stub it out, and *build other modules and tests independently.*
- **Prioritize.** Focus on the defects and issues that affect the customer first. New features come last. If necessary, defer features rather than delivering buggy code.
- **Assign responsibilities.** Volunteers are preferred, each in his/her area of expertise, but *one* person must be accountable for each task.
- **Track defects,** and record everything that will help you reproduce, locate, and fix them. Document any that remain at delivery time, so the customer won't be surprised.

4. Programmers used to having the spec handed down struggle to improvise

6. Architecture document that looked nice on paper was not followed in practice

- *You* will create the spec details as you go, each piece just before it's needed. It needn't be pretty, complete, or even written, as long as everyone understands the current task and you've got the big picture.
- Discuss the implementation, one piece at a time, when the developer is ready to code it. Write the skeleton yourself if necessary, and let the team fill in the "guts." You want to keep them focused on each task, without "improvising."
- Be available to answer questions as they arise. Your primary goal is to keep the team productive.

2. Bugs are fixed only to introduce new bugs
5. No automated unit tests aggr[a]vate the situation

- Plan and start unit testing ASAP. If possible, enlist resources outside the team.
- Fix small problems before they grow larger--or get hidden. Confidence in each small piece builds confidence in the whole.

7. Third party components used become bottlenecks not having been tested for fitness in the first place

- Brainstorm solutions when you're not coding. Don't let them stop your progress if at all possible. Can you encapsulate or code around them? Replace them?

General suggestions:

- **Stay ahead of the team.** Anticipate and try to solve problems before your team hits them. Gather any necessary information before it's needed.
- **Communicate constantly.** Make it clear that you want no surprises, and solicit concerns, questions, status, roadblocks, *etc* throughout each day. Encourage collaboration and share "discoveries" across the team.
- **Celebrate every success.** Compliment a clever solution, bring donuts when a problem is solved, demonstrate a new working feature ... anything that shows the team you appreciate them.
- **Get each task done, then move ahead.** Don't waste time tweaking, enhancing, or reworking anything that isn't a direct barrier to success.
- **Keep your promises** to the team, the customer, and your management.

Good luck -- please keep us posted!

answered Oct 26, 2008 at 6:27

Adam Liss

48.3k ● 13 ● 113 ● 152

Adam, that was straight from the heart... Thanks for your valuable advice. – Vyas Bharghava Oct 26, 2008 at 17:13

@willcodejavaforfood: You made my day! I'm in the US but will keep you in mind if I come across a "remote" Java

project. Great handle, too! – Adam Liss Jan 7, 2009 at 4:48

Really nice answer. I'll up vote this. Although, I see this as the "Blue State" answer. See my new answer as the "Red State" answer and then decide how to mix the two to your own tastes! :-) – Tall Jeff Jan 11, 2009 at 1:27

12

Run away or find a new job. This is a death march and they need a scape goat.

> Often, the death march will involve desperate attempts to right the course of the project by asking team members to work especially grueling hours, weekends, or by attempting to "throw (enough) bodies at the problem" with varying results, often causing burnout.

Share Improve this answer

Follow

edited Oct 26, 2008 at 14:26

Mitch Wheat

300k ● 44 ● 477 ● 550

answered Oct 24, 2008 at 17:53

Maxime Rouiller

13.7k ● 9 ● 60 ● 108

It is a valid solution, I agree. – Jason Oct 24, 2008 at 17:55

Agreed. If your other 9 team-members suck badly enough, you'll have no choice but to do their work for them anyways. Not fun. – username Oct 24, 2008 at 18:53

#1: Make sure this is not a "Death March"
– Vyas Bharghava Oct 24, 2008 at 21:14

This is a valid solution, but completely out of proportion to the problem. – HTTP 410 Oct 27, 2008 at 13:01

9

Freeze releases, and start fixing issues with the program.... deal with the customer complaints by priority (the business side of the company can prioritize) and get the program running. Once you get the biggest issues out of the way, start cleaning up the code. Assign tasks to other developers, and start enforcing coding practices on all new code.

If you can do whatever you want, then look at what the real issues are and deal with them. If that means putting together a new team to develop the software all over from scratch, so be it. But you should try to at least fix the major bugs. Don't bother introducing new features, they only compound the problem, and a program that doesn't work and the problems aren't dealt with lose you clients.

Share Improve this answer

Follow

answered Oct 24, 2008 at 17:56

Elie
13.8k ● 25 ● 78 ● 128

good answer, sounds like you've been there – Simon Oct 24, 2008 at 19:11

No kidding, but the approach works. Sometimes you can fix the existing product, sometimes not, but the most important

step is to STOP AND THINK about what the core problems are and solve them. – Elie Oct 24, 2008 at 19:28

5

Number 10 is obviously the worst problem, or at least the root of all others. Find someone with some creativity and ability to deliver a project, and give them free reign to do anything - including start over.

Share Improve this answer

Follow

answered Oct 24, 2008 at 17:54

Kendall Helmstetter Gelner

75k ● 26 ● 131 ● 151

Starting over is nearly always absolutely the wrong thing to do (joelonsoftware.com/articles/fog0000000069.html). – Joris Timmermans Jan 13, 2009 at 9:47

5

I hope you are getting paid really well. In any case, my plan would be something like these steps in the following order:

0) Stop adding features or functions across the team. Allow bugs to be addressed while the following steps are taken up to step 5, then stop bug fixing & resume feature development:

1) Apply what I call the *Inverse Staffing Law*: Weaker team members slow down the better and faster ones and generally a late software project needs people removed, not added. So, you need to assess the quality of the team

members as individual contributors. Eliminate weaker staff from the team because presumably there are some. This is best done by reviewing their code and examining their bug fixes and figure out who is making the code worse vs. better and chop them for the team. This is not a time to mentor, you are going to need the best folks to have a change of "fixing" the situation in a optimal period of time. If you can't fire them or reassign them, have them getting coffee or something for everyone else left.

2) Assess the code itself. Identify areas of the code that are not constructed well and/or not well abstracted. If a area code is not constructed well and/or is obviously brittle at it what it is supposed to do, target it for a re-write. This feels painful at this point, but it will save you time in the long run. Recurring bugs and/or history of fixes will help identify the code that can't be salvaged. If a code area or module is fundamentally constructed well, but not abstracted well at the interface level, it should be suitable for re-factoring. This will save significant time and is useful code. Keep a list of the re-write areas, the re-factor areas, and the suitable areas.

3) Define a new reasonable architecture that you believe will result in a robust and complete solution to where you want to eventually be in features and functions. The architecture might not be optimal as starting clean, but in effect match up what you have with where you would like to be.

4) Work with the stake holders to decide what will make an acceptable first release attempting to table as many features as possible for "later" releases. Maybe you can't cut anything, but if you can, now is the time to do it.

5) Stop the background bug fixing efforts and assign the defined work out to the (remaining) team to estimate out a reasonable new implementation plan of the rest of the functionality. They need to own the schedule. Roll up the schedule and be fairly conservative. Now you have a reasonable prediction of when you could actually have something workable and robust.

6) Implement the remaining features and then harden up the release by tackling the remaining bugs. I am assuming all the normal good software development practices are observed here like source control, unit tests, etc.

7) Remove as many barriers as possible to keep the team cranking out stuff as fast possible.

8) Monitor for issues, and assist by getting your hands dirty where ever your can. Offer to take on the nastier issues to the extent you can help and still keep all members of the team as productive.

Good Luck!

answered Jan 11, 2009 at 1:18

Tall Jeff

9,984 ● 7 ● 46 ● 61

My answer here is the "Red State" answer. Alternatively, see the well written "Adam Liss" answer as the "Blue State" version of this. – Tall Jeff Jan 11, 2009 at 1:30

@Jeff: If only the world we live in excludes politics and focus instead on getting the job done! There're bosses that work against you, the 'weaker' staff that collude... How do you handle bosses that will rather let go of the client than do the right thing by removing staff that don't measure up!
– Vyas Bharghava Jan 13, 2009 at 19:13

@Vyas - Well it certainly helps to be the boss, I suppose. But aside from that, if you work for a boss that isn't focused on results, I'd say find a better job. It's now or later because the company is ultimately going downhill if the management isn't focused on results. – Tall Jeff Jan 21, 2009 at 23:51

+1 for "have them getting coffee or something for everyone else left" – Andrei Rînea Jan 18, 2011 at 8:38

3

This isn't about technical leadership any more, it's now about project management.

You as the technical lead will just be shifting deckchairs on the Titanic. So here's what I would do if I was the de-facto **project manager**.

1) Identify the project sponsors and stakeholders - both the official ones and the **real** ones.

2) Go to them and request that the project "goes dark" for a week.

3) If they don't agree, walk away from this project.

4) If they do agree, call a project time-out for a week - everything stops.

5) Spend that whole week talking to the important people on the project to identify the real project state.

6) Whilst engaged in those discussions, start formulating a project recovery plan, emphasising possible trade-offs between scope, schedule, budget, and personnel.

7) At the end of the week, decide which (if any) of your possible project scenarios are feasible.

8) Take the best of these scenarios back to the project sponsors and stakeholders, and start negotiating.

9) When a way forward is agreed, reboot the project and pray - possibly not in that order.

edited Oct 26, 2008 at 14:03

answered Oct 25, 2008 at 18:31

HTTP 410
17.6k ● 14 ● 79 ● 129

2

Common sense has already been pointed out to you by Maxim (Quit the death march). But if for reasons unknown you wish to persist, let me regale you with my experience in a similar situation - perhaps it might come useful.



It was my first job in a sleepy old town where good computer jobs where hard to come by and I despertely needed one immediately after college. I was hired coz the management thought i was enthusiastic enough and might be better than nothing (I offered to bring in my own comp to save them a cost of giving me a PC and offered to work for the experience alone)

The project had been abandoned by its creators due to the death march situation and had gone away after deleting all the comments in the code and performing other obfuscations. Nobody knew win32 / MFC stuff either.

I simply started studying the code on good old paper and pencil (lots of rubbing and corrections) until within 20 days time i knew the entire code including the variables by heart and what and where things where happening.

Armed with this knowledge i was able to make a critical piece working which had eluded everyone before. Of-course this was nothing but a drop in the ocean but it enabled the management to buy the clients confidence "smart fellow - got him with great difficulty - already got x working - u will have ur stuff working within y time".

Once the client was convinced and we where able to buy some time, some pressure was taken away. This got some hope back into the team and we started to hammer away for good. 6 months later i got promoted to project lead and 9 months later we had our fix shipment (lots of

progress demos and a visibly more and more satisfied client in between).

As you can see, the elements of success are not directly duplicatable. But i would summarize that you need to breath some hope into the project first - show some progress and win confidence - that of your peers, management and the client. Once that is in place the technical stuff should be corrected too - there is nothing to replace this part of the equation.

If that does not seem likely, all that hard work (oh yes - lots and lots of work like you never imagined - why do you think its called a death march) would be a waste and you had better quit even before you start.

I had no choice and i was hot blooded and desperately need a job. The technical details where something icould work magic upon, and everthing just clicked into place. I really earned a lot of good will and self respect with that piece of work but in the long run its just a story i can narrate with great aplomb and nothing more except for those few in the know.

Things might be different for you but its for you to decide.

Good luck

edited Oct 24, 2008 at 18:37

answered Oct 24, 2008 at 18:31

computinglife
4,391 ● 1 ● 23 ● 18

You were promoted to project lead 9 months out of college? – Tim Oct 24, 2008 at 19:45

Agreed, if there're no expectations, generally we succeed. ;) Not realistic when a 5-6 member team gets billed by the hour.... :) The customer needs value for every $ billed. – Vyas Bharghava Oct 24, 2008 at 21:16

@Tim - yes i got promoted to project lead 9 months out of college ;) - crazy isnt it - as far as the management was concerned i was making things happen for them & they couldnt care less. Speaks volumes for the mess they were in & their relative in-experience. (founded by *techies* from AT&T) – computinglife Oct 31, 2008 at 8:40

@Vyas, the idea is not to do away with expectations BUT to fix something concrete right away and win back the clients confidence. Of what use is reams & reams of time sheets, if client believes nothing useful is being done? Build the confidence & get tech on track & u might yet scrape through. – computinglife Oct 31, 2008 at 8:44

1

- Make sure you *aren't* the scapegoat
- Cut scope creep
- Trim functionality "requirements"
- Implement a faster dev cycle (maybe Agile/Scrum/XP/whatever)

Share Improve this answer Follow

answered Oct 24, 2008 at 17:55

warren

33.4k ● 23 ● 89 ● 128

1

If you can, run away.

If not, you need to stop all activities that make the project unstable - including coding and fixing defects.

Assess where you are

Break up the requirements into much smaller "milestones"

Read some practical books (Mcconnell's "Software Project Survival Guide" comes to mind.

Identify all the problems and risks. Communicate all those to all involved.

Work on each piece one at a time.

Celebrate improvements and milestones as they are reached.

Good luck. Your scenario sounds pretty bad. It may not be salvageable - and things have to change to get better.

Share Improve this answer

Follow

answered Oct 24, 2008 at 18:02

Tim

20.4k ● 24 ● 122 ● 219

If you really had to get it on track (if bailing isn't an option)

1

Start off by accepting that it's a failure in management. You might then want to go on to implementing a strict but light process.

I'd suggest some form of Agile, since it's the easiest to successfully implement without a GURU, but you have to be VERY strict about it, including Pairing, Ruthless Refactoring, Reviews, Spiking functionality, Visibility, TDD, one-week cycles, 8-hour workdays (Yes, longer than 8 tends to harm productivity more than help, as you seem to have noticed)...

Don't be cutting anything out either. Parts of Agile rely on other parts--without the pairing, refactoring and testing you cannot eliminate upfront design (one of the biggest agile failures).

Don't forget about the management side of it. One week iterations to start (demo EVERY week). Constant adaptation. Very short stand-ups every day to address issues. (Keep to 15 minutes max, table longer issues, etc) Burndown charts, core-team with a client on it.

You can't just have a 15 minute meeting every week and 2 week iterations and call it Agile, but if you do it right, it just MIGHT give you a chance. You might get a GOOD agile consultant in to train you on getting started.

Also, constantly evaluate what works and what doesn't. Be prepared to fix what doesn't work. Weekly meetings to analyze that weeks' development successes and failures.

Overall it CAN work, and can bring a flailing team into line, but it's not trivial. The nicest part is that you can implement it without taking huge chunks of time out of your current development. You just keep developing, but you do it better.

Share Improve this answer

Follow

edited Oct 24, 2008 at 18:54

tloach

8,040 ● 1 ● 35 ● 44

answered Oct 24, 2008 at 18:08

Bill K

62.8k ● 18 ● 112 ● 158

1

Tough situation, you have zero customer trust and basically can't be successful under that situation, no matter what.

For all intents and purposes the project needs a reboot; the unfortunate fact is that incumbant shops usually don't get this oppurtunity to start over and re-evaluate everything that is there.

I hate to say it, but you need to halt development and spend a month working out what went wrong...

The result needs to be a plan for a feasible 6month - 1year delivery really making them focus on what the must-haves are and real trade studies on your third party components. And trashing the code base needs to be an option; start a new source control project and when you

get to a particular module port peices that make sense and leave the garbage behind.

Agile is great and all, and a valid approach once you get a real plan in place; but its not going to fix a broken relationship with your customer... or all the junk that's already there.

Share Improve this answer Follow

answered Oct 24, 2008 at 19:01

Patrick

1

Here's the summary of *key learning* after reading through your experiences:

Maxim
1: Make sure this is not a "Death March"

Ellie
2: Make sure what's delivered works 3: Refactor & Realgin codebase to Architecture / Best practices 4: Look at what are the real issues: Is the team technically competenet to deliver?

Kendall
5: Ensure availaibility of Technical Leadership

Bill K
6: Implement Agile Processes (At least automated unit tests if not TDD, short iterations that make progress visible) 7: Get customer buy-in 8: Be prepared to throw out what cannot work (wishful thinking aside)

Warren

9: Make sure the team memebers that remain given a chance to start over

Tim

10: Motivate team and as improvement becomes visible reward them

jsl4980

11: You need buy-in on schedule from your team (most imp.) & customer [This raises more questions. What if your customer asks whether the team is competent enough to stick to your schedule? What if you yourself know that the timelines the team is proposing just shows their lack of understanding]

Ather

12: Is the team commited?

13: Do you formally QA?

Patrick

14: Start over, redesign and reconform to Architecture/Design best practices for modules yet to be developed.

answered Oct 24, 2008 at 21:25

Vyas Bharghava

6,510 ● 9 ● 42 ● 62

The summary has 14 items. You can't do them all. So, what's the first step?

Here's what you have to do first -- get **one** thing improved.

- You've got fundamental quality issues. (#2-5)
- You've got architecture and component issues. (#6, 7)
- You've got schedule problems. (#1, 8, 9)

You can tackle quality. Formal unit testing, heading toward TDD can help. This might be hard because architecture issues slow testing down.

You can tackle architecture. This might be harder because it will probably involve rework that will not appear deliverable. But it may fix quality issues. Or, it may be compounded by fundamental testing problems.

You can tackle schedule. Without other corrections (i.e., quality or architecture) you may not get any traction with fixing schedule issues.

I think that overall improvements in people's attitudes come from starting with *one* success -- any success -- as early as possible. What's the lowest-hanging fruit?

- One long-standing bug? One unit test suite to find and fix that bug?

- One major architectural feature? Would a diagram that everyone can post in their cube help? How about a presentation clarify things?
- One new use case? One new feature that actually works?

Share Improve this answer

Follow

answered Oct 25, 2008 at 1:53

S.Lott

391k ● 82 ● 517 ● 788

1

Here's a good book on the subject:

Catastrophe Disentanglement: Getting Software Projects Back on Track

Share Improve this answer

Follow

answered Jan 6, 2009 at 11:40

John D. Cook

30k ● 10 ● 69 ● 94

1

First off, be resolved that you may fail -- if you can't accept that, don't take the challenge. And that includes being a scapegoat (it does happen). Management won't look at it in those terms (i.e. they're not intentionally/consciously 'setting you up'). But that is a reality of a corporate environment; if you take on the responsibility (often with more pay than those that don't), then your head is for the block if things don't work out. You have to be ready to stick with it for the long haul too. I was once placed on a client site for 8 months to fix a

waning project. And as you saw, one of the other blog-posters here spent 9 months before a release version was ready.

Now, assuming you are okay with the possibility of it going all pear-shaped in spite of your efforts, this is what I suggest:

- a bug tracking system is going to be your number one best friend, it will allow you to regain a semblance of control. you can't hope to understand a complex system as a whole, so 'chunking' it will help. and a bug tracking system allows you to unitize problems and distribute them to the other guys you are working with.
- you have got both technical and political challenges to deal with. the technical generally aren't so bad because you're a coder and you know how to do this. the political ones are much trickier, you're at the helm of a ship thats gone hopelessly off-course, and you're in the Bermuda triangle. the biggest challenge is often stemming the tide of negative sentiment amongst the client (e.g. client: "these cow-boys don't know what they are doing", "they promised me this and didn't deliver", "i have no confidence in these guys to any more").
- for starters, apologize to the customer and tell them in concrete terms what you are doing to do to re-right their project, e.g. you: "I'm sorry about the delay on your project, I'm getting stuck into it now. I've looked

at the project history, and personally, I would be angry too if I was paying good money for this system. the first thing I'm going to tackle is..." <- bingo, you've just taken responsibility for the project which means there's no turning back - its all or nothing now.

- a few other people have said it here, and I agree; stop adding new features. what they haven't mentioned is that you may have to do this to keep the client happy (remember, there's a technical and political side to the challenge).
- understand the business domain as best you can. read through any requirements documents you can get your hands on. you are at a massive disadvantage by coming onto the project late since you don't know what was originally discussed. the devil is in the detail. this is what sunk me on a late projects I wasn't able to salvage, everyone was on edge, and i missed a minor requirement. at the time, it wasn't a big deal and could have been corrected easily, but politically speaking, it was the straw that broke the camels back. one tactic which may help is to go out on client site for a few weeks.
- understand that time is money. its not just a technical issue. the client has paid for something which isn't right or has not been delivered. your company has expended resources, possible having already used up all the project budget - the business is now losing money. and this is where the issue of new features come in again, yes - people are saying don't add them, stablise. but adding new features can be a

politically helpful tactic, management will be happy because new money is coming in for off-spec work.

- I'd recommend against you or your coding crew working ridiculous hours to deliver. if you normally leave at 5pm, leave at 6.30pm or 7pm instead. you and your coding boys can consistently maintain an hour or two of extra work for many weeks on end and perhaps 4-5 hours over the weekend. working until 9pm or 10pm every night will result in burn-out in roughly 2 weeks (some can go longer). after that point, your extra time on the project is doing more harm then good. in the unlikely event your boss takes issue with this, make a choice; do what they ask (i.e. work more hours), or say "I've already committed extra hours to working on this project - I'm here for the long haul and im going to get this project done if its the death of me. but that is the limit of how much time I'm willing to put in. i have other commitments to keep outside of work" <- *but be ready for the consequences* (remember, political situation as much as a technical one).
- there are people here that are saying "stop and write a spec, stop and do this..." - I'm sorry guys, i just cant agree with you here, its unrealistic. the project is already stagnating, the last thing management or the client wants to here is "we have to stop everything and...". I've tried this before, where I've said to the client and management "the bugs will keep coming until we stop and i write up a detailed system test plan. it will take me two weeks" - the client didn't

want to pay for this, and management wasn't willing to wear the cost. as it happened, the bugs kept coming.

- learn to 'juggle' - you have to map out tasks ahead of time so programmers aren't waiting on you. this will generally mean you do less coding yourself. generally this is best achieved by having a project schedule before coding starts. programmers should know what they are doing next after they finish what they are currently working on, and they shouldn't be coming to ask you "what do i work on next?", they should already know.
- build-in recovery utilities, especially if the software has recurring problems which are hard to pin-down. for example; it may take 12 hours to track down a bug and fix it, it may take 2 hours to put in utility (read 'hack') to fix the problem for the time-being. time and momentum are of the essessen, and unfortunately bandaid fixes may be needed.
- be very observant of the clients mood. they need to know you are 'on their side' (e.g. client: "the product is unacceptable", you: "i agree, i would kick our asses to if i was in your position. all i can tell you is im on it and wont rest until its all working"). when the client is back on your site, they will actually start helping you. for instance, they may shield you from pressure from your management.
- lead your guys by example. something along the lines of "I'm staying back a bit to work on the project,

> I'd appreciate the help if your willing to stay back too" and "i know its not our mess, but we're still going to clean it up anyway. i want the client to get some good quality software". programmers could generally care less about the company that got them into this situation, but they may care if its about one of their own or the client ('may').

many of the suggestions I've seen here assume a fairly high degree of power (e.g. 'stopping the project to restart it properly' or 'say no to new features') - you are starting the project already hamstrung, and as a programmer, you will traditionally have less power to affect change then a true manager. this doesn't mean 'give up/don't try' - it just means you are going to have to be creative and do things you don't normally do (i.e. use 'soft' or people skills).

a lot of people here are saying bail on the project, run for the hills. I have been on 3 hopelessly late projects to date. i managed to fix 2, and 1 I couldn't fix. personally, it doesn't bother me to take on a late project. after all, the worst that can happen is you get fired :)

edited Dec 18, 2017 at 12:25

Sinister Beard

3,680 ● 13 ● 62 ● 99

answered Oct 28, 2008 at 6:25

louism

1,639 ● 12 ● 31

Louism, Thank you. I believe, this is also the hallmark of stackoverflow: The sheer enthusiasm of the community. I'm gratified and thankful to all the people that has taken the time to shared their experience here. Will read through carefully and offer my take later today. – Vyas Bharghava Oct 28, 2008 at 20:51

0

If you were involved in the project from the beginning, I hate to say it, but the company should replace you (and the entire team).

It should be reanalyzed with a competent team with real project management processes and lead by a project manager with experience in this situation.

None of the original coders should work on the 'new project' of saving it. They can move to other projects (they don't have to be fired) but to get a fresh look at the project, everybody should be replaced.

And of course, management has to understand and be on board with the fact that the project is going to be much later than expected. If management doesn't agree with this (replace team, find experienced leadership, take a step back and start again) then @Maxim is right - get out of there.

Share Improve this answer

Follow

answered Oct 24, 2008 at 17:58

Jason

17k ● 24 ● 89 ● 142

Luckily... No Jason... I was the Architect for this project when we made proposal to the client. But after we won the project, I was not involved in the project at all. So, the vision I had while solutioning was something the team couldn't grasp [Agile development best practices] – Vyas Bharghava Oct 24, 2008 at 18:08

0

1) The first thing I will assess is whether the people on the team are committed to the project or not? If not, it is worthless to do any other thing. Nothing can prevent the disaster unless I get a dedicated and committed team. 2) I'll make sure that there is QA on the team. 3) Come up with a reasonable plan of iterative and incremental releases to the customer. With the mess we are in, there is no way customer can get everything soon. Based on the priorities of customer, we'll deliver smaller increments of functionality to him frequently. This will keep customer engaged, a bit less-edgy since he is seeing something happening.

Share Improve this answer

Follow

answered Oct 24, 2008 at 18:20

Ather

1,600 11 17

What ever you do, do it step by step.

First, it's not about addind features, it's about fixing the app. Don't add anything new. Just refactor. Say no to any new stuff somebody ask you to introduce in the system.

Don't try to improve the whole app. Take your team, make it focus on one aspect at the time, with the best practices you can, especially using unit test.

Use test driven development only. In that case, it will immediately show you what part of the behavior you don't understand (you can't code a test if you don't know what to test.

So here are the road map :

1. Identify the critical part you need to change
2. Isolate the code that implies this behavior
3. Find any occurence of this code in the rest of the code
4. Refactor using this knowledge and massive TDD
5. Integrate, test and fix until this particular part works
6. Go back to step

Make the situation clear to your boss : it will take time, money and will be painfull. Explain why, what you will do, and that you have no other way or it will fail AGAIN.

A above all, don't try to make it clean the first time. Refactor what you can, but don't expect to change the entire architecture of the part you are working on the first time. You will have to iterate the process **on the whole application** several times.

No miracle. Just method and patience.

Share Improve this answer

Follow

answered Oct 26, 2008 at 9:39

Bite code

595k ● 116 ● 309 ● 334

0

Been there, followed these steps:

Stabilize

- gather the real story: how good/bad is the codebase, how good/bad are the developers, what really needs to get done (bare bone min.), when it needs to get done by
- reduce overtime (tired people, good or bad, don't work well)
- remove the bad, input new/good - err on the side of replacement (many could be burnt out and appreciate even a forced change)
- remove access to bad/un-required code (focus on the 20% of the code base that provides the 80% of the value)

- put base code practices in place ensuring only good code is getting in (don't damage the base anymore)

Control

- implement teams focused on the app components (decouple as much as possible)
- put code management, release management, risk management, QA, etc. in place (build your environment so you can succeed)
- get on your clients/sponsors good side - delivery a win, even if it's a somewhat stable very very small release - and then put in change management (control what gets requested)

Move forward

- develop a plan (planning is essential, plans are useless according to Ike - you need to plan to find what is missing and to set a target, but don't expect to tell the future) - continuous planning is required
- aggressively manage your people - good people make good product - make sure you get and retain the best
- refactor over time - clean up code as you go - you may not have the luxury to fix everything at once so do it overtime to provide for a cleaner code base
- move forward bravely - overtime be more aggressive with your deliveries test (but not stress) your team

Share Improve this answer

Follow

answered Jan 10, 2009 at 23:50

meade

23.3k ● 12 ● 33 ● 36

0

Agile refactoring. Identify and prioritize what customer wants and then create the most important stuff in short sprints out of existing code. Good luck man :)

Share Improve this answer

Follow

answered Jan 11, 2009 at 0:38

mannicken

7,123 ● 4 ● 33 ● 38