LOGIN

PatchSpace Blog

Agile, Lean, Systems Thinking and Software Design

« Back to blog

Blog home
Company site

Ash Moran

Posted 10 years ago
February 3, 2012 at 1:29 PM

27982 views

Why You Shouldn't Hire More Developers

Juan Palacio (@juan_palacio) has kindly translated this article into Spanish as Por qué no necesitas contratar más programadores. Offers of translations to other languages are gratefully received.

The smouldering situation

You’re the lead developer in a team of five. You’re all burnt out. Each of you is in the office from early morning til late into the evening, trying to hack away at the relentlessly growing backlog. In fact, evenings are better for work, because in the day you’re swamped with bug reports and operational issues, and developers rarely get time to work on “new” features. Marketing just signed up a new client with a whole load of new feature requests, and you barely have time to speak to the clients already in production. What’s more, you’re losing more and more of your time in the day to meetings, trying to get the situation under control. You’re probably pulling your hair out thinking “we just don’t have enough time to do all this!”. You need to do something, so you think.

On the back of an envelope you sketch out the situation. Your 5 developers are putting in at least 60 hours a week each, that’s 300 developer-hours a week in total. Out of that 300 hours, you estimate you’re spending 50 hours a week bug-fixing, 30 hours a week on ops issues and 20 hours a week on meetings. 100 hours a week before you get to the new features – that’s a third of your time! But look at the backlog: since the new client came on board, it’s not going down, it’s going up!

The conclusion is obvious: there’s just too much work on. You’re already all working overtime, so you need more people. If you had just two new developers, they could handle the bug fixes and ops issues, and still have time to chip away at the backlog. (Well, they’d only be paid for 40 hours a week each, but they’ll soon pick up the corporate culture of going the extra mile, right?) The solution is now even more obvious: you go to your line manager and ask to start recruiting. Right?

Wrong. The very last thing you want to do in this situation is hire more developers.

The new hire stokes the fire

Your first new developer, Alice, starts on a Monday. She ties up another developer for the whole first day getting her machine partially installed. Tuesday morning she goes off on her own because everyone is in a meeting, but then has to spend the afternoon unpicking what she did because it turns out you use a custom build of one tool. Bob knew this, but forgot to document it because some time last month he was … called into an meeting at short notice. Wednesday you set Alice loose on an easy bug-fix task. It takes her all day as she learns to navigate the code, but she commits it and moves on. Thursday and Friday she spends trying to implement one of the easy features in the backlog, but over half that is spent with another developer because first an old bug got in the way. (It might be in the bug tracker, but since it hit 200 open tickets, nobody really checks it any more.) Anyway, a week goes by, and the work goes out in a Friday evening deploy.

There is a weekend. You’ve all learnt to turn off over the weekend, overtime hasn’t crept in that far yet.

Monday is chaos. In fixing the first bug she tackled, Alice changed something she thought was an error, but was actually an obscure edge case of a business rule. Nobody reviewed it because they lost enough time helping her get set up, and everybody knew it was easy anyway! So the deploy is rolled back. Conversation quickly reveals that the feature Alice committed on Friday was design on top of her misunderstanding of the business rules. Now someone in the team has to do a thorough code review. Even without counting the hours attached to this, it’s clear the team is significantly behind, and in a large or complex code base, there is no reason to believe this will improve soon.

Is this all Alice’s fault? Could she have tried harder? Or is the system at fault?

The bottleneck

What is throttling the performance of this company? It’s clear it’s not in marketing – they’re bringing in clients quicker than the software can be rolled out. And it’s not in analysis – the requirements are building up faster than the developers can turn them into code. (We’ll assume for now that these requirements are actually effective.) It’s not even operations – a week’s work went out on Friday evening, and even if it broke the business rules, it was operational. That leaves us with the bottleneck squarely in development. So if development is the bottleneck, why was it wrong to start hiring developers, to increase the capacity of this overstretched skill?

The assumptions behind hiring

To explain this situation I’m going to make explicit some of the tacit assumptions that often underly hiring in an overworked team. This is quite a crucial point, as much of the shared mindset in software organisations is tacit, and influences decisions without ever being held accountable. It is comparable to the difference between invisible work and work visualised on, for example, a kanban board. (Note that even organisations that use kanban boards often have other, unvisualised work.) The following is not an exhaustive list, but it will serve the point. Many teams *act as if the following are true*:

- Developers are fungible
- Productivity is proportional to developer-hours
- Fixing bugs is valuable
- The requirements are all necessary

Developers are fungible

Tom deMarco calls this The Myth of the Fungible Resource (in Slack). Many factory and warehouse jobs are largely fungible, in that the time to bring someone up to full productivity is inconsequential (hours or days). This is not true of development, where even if a new hire knows the programming language, framework and even the generic business domain, it will still take a long time for tacit knowledge of the codebase to flow into his head.

I don’t think developers actually believe they are fungible (at least, none I’ve met would say so), yet I’ve seen teams hiring as if this assumption was valid. Any time you act as if a new developer working alone will immediately increase team productivity, you are acting as if it was true. This tacit assumption is in contradiction to what most developers will explicitly state the nature of their work is like. In a contradiction, at best one side is right.

Productivity is proportional to developer-hours

There are two forms of this assumption: first, the idea that a developer working a 10-hour day will be 25% more productive than a developer working an 8-hour day; second, the idea that a team of 10 developers is 25% more productive than a team of 8.

To address the first, remember that the nature of software development is creating new knowledge, which I explained previously in the post Why Can’t Developers Estimate Time?. One consequence of this is that development is a creative task that involves constantly making logical decisions. (For example, is it time to break up this long block of code? To use XML or JSON? To replace the application framework?) As explained in the article Do You Suffer From Decision Fatigue?, the human brain has a limited capacity to make these types of choice, and once tired, it will take shortcuts. The feeling of “I just want to go home” may be causing you to introduce bugs. Using overtime as evidence the team has too little capacity is therefore in contradiction to what scientific studies show. That one side of this contradiction paints a picture of developer heroism does not make it any more true.

The second form of the productivity-time assumption is based on the idea that the productivity of a team scales linearly. This is not true for the simple reason that the complexity in managing a team is not the number of people involved but the paths and amount of communication involved. Compare, for example, how easy it is to get 50 people to pass a ball down a line, versus getting even 5 people to agree on the menu for a meal in a Chinese restaurant.

Fixing bugs is valuable

Bugs are, by definition, something the system was not intended to do. There are times when nobody knows if an idea will work (this is the realm of the Lean Startup). But there are many, many defects in the world where the developers had, at the time they wrote the bug into the system, the knowledge to determine the behaviour was wrong, yet for some reason they didn’t. Imagine you’ve taken your otherwise immaculate car in to have the brakes replaced, and when you drive it afterwards it starts pulling to one side. Exactly how much value would you see in having the wheels re-aligned, even if it was done for free?

When these sorts of bugs are fixed, what is actually happening is not work, but rework. The developer must load the knowledge of that bit of code into his head, including the requirements, the way it is implemented, the dependencies it has, and then make the change. Even in the case where the bug fix is purely the addition of code, and not changing existing code, the developer must still repeat the process of understanding the subsystem to make that addition. When a new developer is doing this, they may have to learn from scratch a whole area of code, along with any tacit knowledge required for it, and then cross their fingers they don’t break anything (if a test suite catches a bug here, that knowledge has already been made explicit). If bug-fixing is waste, fixing bugs introduced while bug-fixing is doubly so. I call it *whack-a-mole development*, a term I’m deeply saddened hasn’t caught on yet.

If your team is spending time any significant amount of time fixing bugs, it has much more capacity than you realise. That’s not to say it’s an easy reserve to tap into, but it is there. The attitude that bugs are inevitable is harmful, as it will give strength to the tacit assumption that fixing bugs is valuable.

The requirements are all necessary

I’ve saved this for last as it has a different nature to the other assumptions: it necessarily involves decisions made outside the development team. Unless the team is making the software entirely for itself, someone else will be involved specifying the development work being done. If it turns out that 30% of the features in your software are never used or unnecessary, then at least 30% of the development time is pure waste. (It may be more, due to the complexity of managing the larger codebase, and the waste due to bugs in the surplus code.) However, as many teams are contractually obliged to deliver a fixed spec without reference to the value of the features in that spec, this may be a difficult source of waste to fix. Because of this, I won’t say much more about it. It is in any case usually easier to get someone to clean our their garden shed if you can show you can keep your bedroom tidy first.

The reality of the busy team

Remember that we came here because Alice was brought in to increase the capacity of an “overworked” team. Yet we’ve seen that the assumptions underlying the need to hire her were false:

- The team is not running at full capacity, it is spending at least 25% of its time on rework and avoidable maintenance, even taking into account overtime
- The team is not even producing maximum quality given the existing skills of the team, because some of the bugs were introduced due to developers being fatigued and over-stressed
- Alice can’t be brought in to give immediate relief, because the communication overhead actually reduces productivity in the short term at least

A note on team sizes

You may hold a valid reservation about my bold statement that you shouldn’t hire more developers: increasing capacity isn’t the only reason you may want to do so. A very valid one is redundancy, as very small teams are vulnerable to Murphy’s Law. If your only developer is run over by a bus, your project is in immediate jeopardy. (It was in jeopardy before, it just took a bus to show it.) Then again, it is possible to have a team of 10 devastated by a single errant bus incident, if the team has formed knowledge silos.

Christopher Allen’s article The Dunbar Number as a Limit to Group Sizes explains some of the consequences of various team sizes.

Small team sizes may be less of a risk than they appear though. In my personal experience, developers are very rarely run over by buses. And they very rarely leave because of pay. But developers do very frequently leave because of unsatisfactory working conditions. If you’re the manager of a situation like the one in the story, one of them has probably told you so, in as many words.

There’s also another situation you might want to increase the size of your team: when the person you’re bringing in has the knowledge and experience to help improve everyone else’s effectiveness. In this case, though, their responsibilities will have to extend far beyond pure development.

What to do

The first thing is step back and check if you’re trying to solve a problem fundamentally caused by systematic waste by throwing more effort at it. This is akin to putting more sailors on water-bailing duty when the ship’s engineer should be welding the hull shut. Fred Brooks stated Brooks’s Law over thirty years ago: “Adding manpower to a late software project makes it later”. Please don’t ignore the past unless you want to turn your office into a historical re-enactment. I’ve had someone personally tell me “We have a perfect graph showing velocity going down as we started adding more people!”.

Improving the productivity of a software team is hard. It involves understanding the business, the team, the history, the obstacles blocking progress. It is a complex, context-sensitive problem. This being a blog post, one already in need of a TL;DR summary, I’ll just point you in the direction of a suitable body of knowledge, and suggest you read The Goal.

We see the world filtered by the metaphors we hold. The Goal (by Eli Goldratt) shows how our common assumptions blind us to the real causes of the problems we face every day. It has sold millions of copies, has been used in thousands of corporations, and is taught in hundreds of colleges and universities. The Goal is the archetypal book on how to focus on what matters. It will take you only a couple of days to read, and will teach you to see the real source of bottlenecks in your organisation. (This is not an affiliate link.)

I’ll end with a rule of thumb though: when faced with a situation like the one described above, try to exploit what you already have before throwing more effort and money at the problem. You’ll often realise you can be more effective with the people and resources you already have, once you discover the real reason things are going wrong.

Thanks for reading

My name is Ash Moran. I’m a software developer and agile coach, and owner of PatchSpace Ltd (Twitter). If you have any feedback, questions, or would like to know *more about my services, feel free to contact me at ash.moran@patchspace.co.uk, or continue the discussion in the comments.*

Upvote 1 Tweet

Share 0

Like this post? Subscribe by email »

29 responses

Great article Ash, don't assume there's a bottleneck until you address areas of overproduction elsewhere :)

— Captain Crom

Captain Crom - Heh, very good point. I've assumed for the example here that there isn't massive overproduction of requirements for the reason stated above. I think I've only seen that once myself. As it happens though, the answer would still involve not hiring any more developers!

— Ash Moran

The same problem space is discussed in quite broadly (and quite entertaining) on joelonsoftware.com...

— PS

"The myth of man month" issue.
The solution to get around the issue is a process change. Require all features be covered by automated regression tests, before, after every checkins. Run those tests 24x7. For all the projects I worked on with this process, the dev have to always on time, sw quality have been consistently high.

— tonyplee

I think one of the most crucial points there is the requirements. Clients often don't realise that what might be a fairly inconsequential feature can sometimes be a mountain of development work.
Spending more time at the requirements phase on analysis & educating the client as to costs for each feature can pay large dividends.

— AMcDermott

If you have crap processes and add more people into the mix, you get more crap. There isn't anything surprising about that.
30 hours a week on Ops? Alice should have been a full-time ops person. Take over what's being done now, fixing the issues in the process, and increasing the efficiency of everyone involved. If the Ops job isn't taking a full work week, she can pitch in on those "ramp up" instructions that Bob forgot to document.

— j3

I've seen Joel's description of the problem but i do feel Ash has done a great job of explaining this. Im very new to development and this has really opened up my eyes on the matter.

— Khuram Malik

Read the mythical man month. Its still true.

— phpguy

I loved this article!
Working smarter is always the better way to go!

— Sean Behan

"The first thing is step back and check if you’re trying to solve a problem fundamentally caused by systematic waste by throwing more effort at it. "
Beautifully said.

— Ana

Very nice article.

— zachdennis

Great article Ash, I too believe in making existing teams more productive, and working smarter and faster.
It all boils down to good planning as well, including managing scope creep and being realistic on quoted hours. A developer almost always underestimates the amount of work, except the highly experienced ones.

— Adriaan (Entegral)

I would tend to disagree. There are some higher level aspects

worth considering. The first is that you need to hire more people in order to grow. The second is that your process needs to allow subtasks that can be split off and managed as a separate project which could have its own development
team. The third is that all efforts have overhead and cannot
run beyond 70% capacity for any length of time so working
to eliminate overhead beyond this amount is wasteful as it will show up elsewhere. The fourth is that you risk having the
whole team suffer overload if one person get sick or leaves.
The fifth is that you do not have a "long term" view. You need
to add capacity in smooth increments and invest in the new
person, including education and training.

Hire a competent manager. Find someone who understands
the people side of development and these problems should
disappear.

— Tim Daly

I agree Tim, I think Ash's focus on this article is at crunchtime where it is not the best idea to through in more resources and distract everyone. But I agree with you, the long term view should be there to grow and for that you will need to hire people. Yes, your team needs to be flexible to split sub tasks to new less experiences developers

— Adriaan (Entegral)

I pretty much disagree with everything in this post.
The team you describe at the beginning is already fucked. The company is running like amateurs and won't ever succeed.

Work = new work plus rework and bug fixing. So suggesting a dev isn't fully utilised because they're doing some rework or bug fixing is being disingenuous with semantics. This is the kind of weasel words that suits use to lift extra budget to cover their inefficiency.

Software has bugs. They need fixing. That's it. Almost always is it preferable to ship a product which conforms to a known quality/quantity than an unpredictable one with the newest feature crowbarred in.

There is always handholding when recruiting new staff. That's also just life. I agree that hiring at a critical point and expecting to have more productivity is erroneous, but it's is manageable such that there is no less productivity as a result.

Knowledge silos happen. There is never enough time to document everything. Again this can be managed to distribute info, and it's a foolish manager who doesn't effect some form of staff rotation (this happens naturally as devs are easily bored in my experience).

The resource problem you cite above is this: too much work / not enough time. Usually this comes about through overselling, through bad comms between eng and sales, through poor sales discipline. If your product is good, no customer will mind you spending an extra man month making it great. If they do, then your sales guy should walk. It does nobody favours taking on work that's not achievable.

One last thing. This whack a mole thing is a myth. With a very light process it's possible to have every fix managed such that it never hits the stable branch without being reviewed by someone with a clue and/or tested. I'll go further and state that in most products I've managed, and with a very light process, it is VERY RARE to have a defect regression hit core code. Even then sometimes we've taken the decision that the regression is "the right thing to do" as it exposes real defects elsewhere.

My biggest gripe with your article is that is works against the nature of making great products professionally. Defects happen and need fixing. Changes need making at the last minute. Products are under resourced and oversold. Companies hire new staff. All these things happen and will continue to happen.

It's better not to write a list of ways to avoid these things (you will fail), and instead to admit that they will happen, understand them, and work with these conditions to mitigate their negative effects.

— Neil

I've reread this and I think I'm guilty of answering several questions that weren't asked but were touched upon. My answer is "don't get there in the first place", which somewhat makes itself immediately redundant in the context of the post!

No, you can't hire your way out of trouble. That said, hiring good people is so hard, I would never turn down the chance of open headcount.

I still stand by my comments that your process should protect you from whack-a-bug / dangerous commits, and if you hire someone who's good but are worried about them bothering your busy engineers then they can be banned from hassling them for the crunch period. Good devs are, to quote n-billion job specs "good self-starters" and will make progress without input from devs experienced in your codebase.

That's it. I'll reread this in an hour and add another correction, obviously. :)

— Neil

The one thing I saw no mention of is what turned this situation around in many of the shops I have been involved with - that is QA. Once QA was implemented - the volume of bugs was reduced or at least caught before they jeopardized production. It also lent an air of accountability that was not there prior. Before QA came in - the goal was to get it to the client to satisfy a contract. Now it became complete the work to pass QA. Knowing the work was to be more scrutinized meant more care was taken completing the tasks. The added advantage was that the QA staff became a source of industry knowledge when new hires did occur.
The specifications were an important part of the solution. QA needed the specification to determine if the task had been completed correctly. We found that with a proper specification, development was better able to speak to the complexities involved and as a result - better price the task to the client. This allowed them to make better decisions about the features to be added - even if the time was being comped. The client was better able to prioritize the features that needed to added.

As my CEO used to say 9 women can't make a baby in a month - adding more developers will not shorten the timeline in a crisis situation. Hiring a new person requires time to find the right person as well as time to bring them to speed. If the business is growing - a new person may be in order - but the hire should be handled with open eyes and a solid plan

— pamshaw

Neil - thanks for your replies.

You're right, the situation I describe is dire, but tragically not that uncommon. However, many companies manage to limp along like this. If big enough or somehow managing to charge enough, they can carry teams performing this poorly for a very long time.

I'm not suggesting that the team "isn't fully utilised because they're doing some rework or bug fixing". Rather the opposite, as if anything I'd say that they are _overutilised_. What Lean Product Development teaches us is that to run a software team at this level of utilisation leads to very large queues of work, and the delay this introduces causes considerable economic damage (rework both introduces extra work and makes individual tasks take longer to release).

Regarding a "known quality/quantity", it's impossible to prove the absence of bugs, although it is possible to say that a certain category of bug is absent or highly unlikely to be present. For example, referential integrity in a relational database prevents all bugs relating to inconsistent references. Other things can be handled with a pattern or process, for example a code review by a DBA might pick up a poor choice of data type where the check could not be automated. Or a lint tool run as part of the build might pick up dangerous language style. The fewer categories of bugs are creeping through, the less variation in lead time is caused by defects. Immature teams can lose a lot of capacity to trivial issues. Mature teams have fewer defects, but also they tend to be of the far more surprising kind, with less things that can be picked up by simple inspection.

As for "overselling" , "bad comms between eng and sales", "taking on work that's not achievable" etc, I've specifically avoided talking about that in this case. I'm not saying it's not a cause of problems, but this post was focussed on cases where the bottleneck lies more clearly within development. It is certainly possible (as Captain Crom pointed out), to cause an artificial bottleneck by pushing too much work downstream. But even then, the first step is still not to hire more developers.

I dispute that the "whack a mole thing is a myth". I have seen myself companies that re-introduce defects over time. If your company does not suffer this problem, count yourself lucky!

Please note that I'm not denying that "defects happen and need fixing", nor any of the other problems you list. But with incremental improvement it is possible to increase team effectiveness and discover its true stable capacity. That "defects happen and need fixing" in no way absolves you of responsibility to find the systemic causes of these defects and remove them. The reason so many teams fail to mature is because of a "shit happens" attitude.

— Ash Moran

Ash - The issues you refer to with hiring new developers relate to:
1) process maturity - are there established practices that can easily be transferred to new resources
2) team formation - how does the team assimilate new resources.

-- My current team is about 5 developers short of planned capacity. For a number of reasons we we have not been able to hire enough developers of the right skill and experience.
-- the team however, is not working 60 hour weeks, because we have projected a lower capacity.
-- we have assimilated 2 groups of 3-4 developers over the last 2 sprints, and we do not plan their output in the sprint they join on.
-- our business analyst walks the new guys through the existing application, and we have some canned webex demos for business cases and application highlights.
-- we have existing developers leading layer based practices, and we assign resources to those layers initially, so that they can focus on one aspect to learn initially until they have a broader understanding of the application.
-- We spend as much time collaborating as we do coding, because that is our culture. It ultimately means less re-work, but also less planned velocity.
-- I believe that in this team, we could assimilate new resources every sprint without really upsetting the apple cart - but I hope we can get up to planned capacity soon, as the delivery date is getting uncomfortably close.

P.S. I have been fighting whack-a-mole practices since 2001. The issue is usually caused by developers who do not understand - oo or layer contracts or other design patterns that keep you out of jail.

I always enjoy reading your posts. Mind you, I am the "functional architect" on this product - primary interface between product owner and developers - making sure developers understand "meta-requirements" that form around the details that analysts deliver in user stories.

— Rich Stone

interesting article but probably should have been titled "when you shouldn't hire more developers". only until the system is under control should decisions be made about growing a team.

— eikonne

Rich - thanks for such a detailed description. It sounds like you've got a stable and mature process for adjusting team size without causing excessive disruption, and also a realistic assessment of your capacity. These are a really important capabilities, but ones that are not universal. Hopefully we can raise aware of the problems caused by getting this wrong.

— Ash Moran

@eikonne Yes - that's exactly it! I'm a TOC practitioner, so my mindset is 1) identify the bottleneck, 2) exploit the existing bottleneck capacity, 3) subordinate everything else to this, 4) elevate the bottleneck (by adding more people, etc). You describe doing step 4 in its correct place. However many teams see being out of control and underworked as a sign they are under-staffed, and as a result they rush to elevate the bottleneck (or elevate _something_) too soon. The result is more chaos.
As the article was long enough in itself, I avoided discussion of these related issues. Hopefully the problem context as described is common enough that it's still relevant to many people.

— Ash Moran

yy

— oo

In this article, I understand the policy to hire more developers and how they are very helpful to increase the development business.

— Anna Harris

I agree. But there is no other option but to hire developer for software development. They are the most crucial part in software development. Good work!!!!!!!!!!

— sudhacis

I am a client, and when I posted my project to couple of freelancing sites to understand they are not worthwhile. Most clients redo or repost the projects as the quality is not met in the freelancing sites. I recommend the www.hireprogrammers.in, where they have high-level of expertise and help the clients to get the right person for their projects. They assure high productivity and help in project management also

— Jackthomas

CMS and Template Customization:
CMS customization(Joomla,Oscommerce, Wordpress):

Do you want to take advantage of the latest open source CMSs out there, such as Joomla or Oscommerce? Or have an existing script that needs some modifications and additions? You already own a web site template and want it to be customized? We can help you. If you just bought a Full Flash template we can help you customize that too according to your website requirements.

CMS customization:
United IT Solution Ltd. specializes in customizing open source CMSs like joomla!, mambo, oscommerce, creloaded, magento, zen cart, phpbb, wordpress, OSDate, gallery V2, dolphin, sugarCRM etc. We can customize any of these scripts for you and blend with your existing web site. http://uniteditbd.com/

We love to work with any open source scripts and are inspired by its philosophy. But we specifically love to work with Joomla! We are enamored by its powerful features, modularity and flexibility to use it as a platform to develop small information based content managed website to large portal with many complex and dynamic features that is also scalable and robust.
http://uniteditbd.com/

— sultan

Most Attractive Blog & Useful Important Information I Got This Blog .

— Hire Developer

1 visitor upvoted this post.