Software development metrics and reporting [closed]

Asked 14 years, 11 months ago Modified 4 years, 3 months ago

Viewed 7k times

33

Closed. This question needs to be more focused. It is not currently accepting answers.

Want to improve this question? Update the question so it focuses on one problem only by editing this post.

Closed 4 years ago.

Improve this question

I've had some interesting conversations recently about software development metrics, in particular how they can be used in a reasonably large organisation to help development teams work better. I know there have been Stack Overflow questions about which metrics are good to use - like this one, but my question is more about which metrics are useful **to which stakeholders**, and **at what level of aggregation**.

As an example, my view is that code coverage is a useful metric in the following ways (and maybe others):

- For a team's own internal use when combined with other measurements.
- For facilitating/enabling/mentoring teams, where it might be instructive when considered on a team-by-team basis **as a trend** (e.g. if team A and B have coverage this month of 75 and 50, I'd be more concerned with team A than B if the previous month they'd had 80 and 40).
- For senior management when presented as an aggregated statistic across a number of teams or a whole department.

But I **don't** think it's useful for senior management to see this on a team-by-team basis, as this encourages artifical attempts to bolster coverage with tests that merely exercise, rather than test, code.

I'm in an organisation with a couple of levels in its management hierarchy, but where the vast majority of managers are technically minded and able (with many still getting their hands dirty). Some of the development teams are leading the way in driving towards agile development practices, but others lag, and there is now a serious mandate from the top for this to be the way the organisation works. A couple of us are starting a programme to encourage this. In this sort of an organisation, what sort of metrics do you think are useful, to whom, why, and at what level of aggregation?

I don't want people to feel their performance is being assessed based on a metric that they can artificially

influence; at the same time, the senior management are going to want some sort of evidence that progress is being made. What advice or caveats can you provide based on experience in your own organisations?

EDIT

We are definitely wanting to use metrics as a tool for organisational improvement not as a tool for individual performance measurement.

code-coverage metrics

Share

Improve this question

Follow

edited May 23, 2017 at 12:33

Community Bot

1 ● 1

asked Jan 6, 2010 at 20:24

David M

72.8k ● 13 ● 163 ● 187

It's worth noting that, in general, one definition of a software developer could be that when presented with an arbitrary measure of their performance, they find some way to game it. This is not generally malicious, but just something in the developer mentality, we like systems and finding neat ways round them. – Paddy Jan 14, 2010 at 12:07

1 Yes, exactly. Not just software developers, but in any profession, if you use a metric to measure performance it gets gamed. Look at the National Health Service targets here in the UK...! – David M Jan 14, 2010 at 12:44

There are some really useful answers here, and I almost wish I could split the bounty. However, I'll have to accept one of them... – David M Jan 15, 2010 at 14:13

10 Answers

Sorted by: Highest score (default)

47

+150

A tale from personal experience. Apologies for the length.

A few years ago our development group tried setting "proper" measurable objectives for individuals and team leaders. The experiment lasted for just one year, because hard metrics didn't really work very well for individual objectives (see my question on the subject for some links and further discussion).

Note that I was a team leader, and involved in planning it all with my technical boss and the other team leaders, so the objectives weren't something dictated from on high by clueless upper management -- at the time we really wanted them to work. It is also worth noting that the bonus structure inadvertently encouraged competition between developers. Here are my observations on the things we tried.

Customer-visible issues

In our case, we counted outages on the service we provided to customers. In a shrink-wrapped product it might be the number of bugs reported by customers.

Advantages: This was the only real measure that was visible to upper management. It was also the most objective, being measured outside the development group.

Disadvantages: There weren't that many outages -- just around one per developer for the whole year -- which meant that failing or exceeding the objective was a matter of "pinning blame" for the few outages that did occur in each team. This led to bad feeling and loss of morale.

Amount of work completed

Advantages: This was the only *positive* measure. Everything else was "we notice when bad things happen," which was demoralising. Its inclusion was also necessary because, without it, a developer who did nothing all year would exceed all the other objectives, which clearly wouldn't be in the interests of the company. Measuring the amount of work completed checked the natural optimism of developers when estimating task size, which was useful.

Disadvantages: The measure of "work completed" was based on estimates provided by the developers themselves (usually a good thing), but making it part of their objectives encouraged gaming of the system to inflate estimates. We had no other viable measure of work completed: I think the only possible valuable way of measuring productivity is "impact on the company bottom line," but most developers are so far removed from direct sales that this is rarely practical at an individual level.

Defects found in new production code

We measured defects *introduced* into new production code during the year, as it was felt that bugs from previous years should not count against any individual in this year's objectives. Defects spotted by internal quality teams were included in the count even if they didn't impact customers.

Advantages: Surprisingly few. The time lag between the introduction of a defect and its discovery meant that there was really no immediate feedback mechanism to improve code quality. Macro trends at a team level were more useful.

Disadvantages: There was a heavy focus on the negative, since this objective was only invoked when a defect was found and we needed someone to blame for it. Developers were reluctant to record defects they found themselves, and a simple count meant that minor bugs were as bad as severe problems. Since the number of defects per individual was still quite low, the number of minor and severe defects didn't even out as it might with a larger sample. Old defects were not included, so the group's reputation for code quality (based on *all* bugs found) did not always match the measurable introduced-this-year count.

Timeliness of project delivery

We measured timeliness as the percentage of work delivered to internal QA teams by the stated deadline.

Advantages: Unlike counting defects, this was a measure that was under immediate, direct control of the developers, as they effectively decided when the work was complete. The presence of the objective focused the mind on completing tasks. This helped the team commit to realistic amounts of work, and improved the perception by internal customers of the development group's ability to deliver on promises.

Disadvantages: As the only objective directly under the developers' control, it was maximised at the expense of code quality: on the day of a deadline, given the choice between saying a task is complete or doing further testing to improve confidence in its quality, the developer would choose to mark it complete and hope any resulting bugs never come to the surface.

Complaints from internal customers

To gauge how well developers communicated with internal customers during development and subsequent support of their software, we decided that the number of complaints received about each individual would be recorded. The complaints would be validated by the manager, to avoid any possible vindictiveness.

Advantages: Really nothing I can recall. Measured at a sufficiently large group level it becomes a more useful "customer satisfaction" score.

Disadvantages: Not only highly negative, but also a subjective measure. As with other objectives, the

numbers for each individual were around the zero mark, which meant that a single comment about someone could mean the difference between "infinitely exceeded" and "did not meet".

General comments

Bureaucracy: While our task management tools held much of the data for these metrics, there was still quite a lot of manual effort involved to collate it all. The time spent obtaining all the numbers was not enjoyable, generally focused on negative aspects of our work and may not even have been reclaimed by increased productivity.

Morale: For the measures where individuals were blamed for problems, not only did those with "bad" scores feel demotivated, but so did those with "good" scores, as they didn't like the loss in team morale and sometimes felt they were ranked higher not because they were better but because they were luckier.

Summary

So what did we learn from the episode? In later years we tried to re-use some of the ideas but in a "softer" way, where there was less emphasis on individual blame and more on team improvement.

- It is impossible to define objectives for individual developers that are objectively measurable, add

value to the company and cannot be gamed, so don't bother to try.

- Customer issues and defects can be counted at a wider team level, *if* the location of the defect is unequivocally the responsibility of that team -- that is, you don't ever have to play the "blame game".
- Once you measure defects only at the level of responsibility for a code module, you can (and should) measure old bugs as well as new ones, since it is in that group's interest to eliminate all defects.
- Measuring defect counts at a group level increases the sample size per group, and so anomalies between minor and severe defects are smoothed out and a simple "number of bugs" measure can mean something, such as to see if you are improving month-on-month.
- Include something that upper management care about, because keeping them happy is your primary purpose as a development group. In our case it was customer-visible outages, so even if the measure is sometimes arbitrary or seemingly unfair, if it's what the bosses are measuring then you need take notice too.
- Upper management don't need to see metrics they don't have in their own objectives. This way it avoids the temptation to blame individuals for errors.

- Measuring timeliness of project delivery *did* change developer behaviour and put a focus on completing tasks. It improved estimation and allowed the group to make realistic promises. If it were easy to collect the timeliness information then I would consider using it again at a team level to measure improvement over time.

All of this doesn't help when you are required to set measurable objectives for individual developers, but hopefully the ideas will be more useful for team improvement.

Share Improve this answer

Follow

edited May 23, 2017 at 12:01

Community Bot

1 ● 1

answered Jan 13, 2010 at 15:06

Paul Stephenson

69.3k ● 9 ● 52 ● 51

1 Hey, don't apologise for the length of it. Much appreciated! – David M Jan 13, 2010 at 15:07

3 +1 for a lifetime of wisdom condensed. I wish I could give you +2 – Chris Huang-Leaver Jan 15, 2010 at 14:27

19

The key thing about metrics is knowing what you are using them for. Are you using them as a tool for improvement, a tool for reward, a tool for punishment,



etc. It sounds like you're planning to use them as a tool for improvement.

The number one principle when setting metrics is to keep the information relevant so that the person receiving it can use it to make a decision. Most likely a senior manager cannot dictate the micro level of whether you need more tests, less complexity, etc. But a team leader can do that.

Therefore, I don't believe a measure of code coverage is going to be useful to management beyond the individual team. At the macro level, the organisation is probably interested in:

- Cost of delivery
- Timeliness of delivery
- Scope of delivery & external quality

Internal quality won't be high on their list of things to cover off. It's a development team's mission to make it clear that internal quality (maintainability, test coverage, self-documenting code, etc) is a key factor in achieving the other three.

Therefore you should target metrics to more senior managers which cover off those three such as:

- Overall Velocity (note that comparing velocity between teams is often artificial)

- Expected vs Actual scope delivered to agreed timelines
- Number of production defects (possibly per capita)

And measure things like code coverage, code complexity, cut 'n' paste score (code repetition using flay or similar), method length, etc at a team level where the recipients of the information can really make a difference.

Share Improve this answer

Follow

edited Jan 7, 2010 at 8:56

David M

72.8k ● 13 ● 163 ● 187

answered Jan 6, 2010 at 20:35

mopoke

10.7k ● 2 ● 33 ● 32

Yes, motivation of measuring is definitely for improvement rather than reward or punishment. I was adamant about that before I accepted the role. – David M Jan 6, 2010 at 20:59

It's a good answer - I was hoping for a range of responses however, so hope you won't be offended if I open it up with a bounty? – David M Jan 7, 2010 at 12:30

@mopoke - happens when the first is so thorough... ;) – David M Jan 7, 2010 at 20:51

Don't mistake length for thoroughness :-) – mopoke Jan 7, 2010 at 21:07

A metric is a way of answering a question about a project, team or company. Before you start looking for the

4

answers, you need to decide what questions you want to ask.

Typical questions include:

- what is the quality of our code?
- is the quality improving or degrading over time?
- how productive is the team? Is it improving or degrading?
- how effective is our testing?
- ...and so on.

Each question will require a different set of metrics to answer. Collecting metrics without knowing what questions you want answered is at best a waste of time and at worst counterproductive.

You also need to be aware that there is an 'uncertainty principle' at work - unless you are very careful the act of collecting metrics will change people's behaviour, often in unexpected and sometimes detrimental ways. This is especially so if people believe they are being evaluated on the metrics, or worse still have the metrics tied to some reward or punishment scheme.

I recommend reading Gerald Weinberg's Quality Software Management Vol 2: First Order Measurement. He goes into a lot of detail on software metrics, but says the most important are often what he calls "Zero Order Measurement" - asking people their opinion on how a

project is going. All four volumes in the series are expensive and hard to get hold of, but well worth it.

Share Improve this answer

Follow

answered Jan 12, 2010 at 8:48

Dave Kirby

26.5k ● 5 ● 70 ● 84

4

Software writing

- What must be optimised?

CPU(s) use, memory(s) use, memory cache(s) use, user time use, code size at run-time, data size at run-time, graphics performance, file access performance, network access performance, bandwidth use, code conciseness and readability, electricity use, (count of) distinct API calls used, (count of) distinct methods and algorithms used, maybe more.

- How much must it be optimised?

It must be optimised the minimum reasonable amount (except in areas where surpassing acceptance test criteria is desirable) required to pass acceptance tests, facilitate maintenance, facilitate audit and meet user requirements.

("... for legal/illegal input test data and legal/illegal test events in all test states at all required test data volumes and test request volumes for all current and future test integration scenarios.")

- Why the minimum reasonable amount?

Because optimised code is harder to write and so costs more.

- What leadership is required?

Coding standards, basic structure, acceptance criteria and guidance on levels of optimisation required.

How can success of software writing be measured?

- Cost
- Time
- Acceptance test passes
- Extent to which acceptance tests it is desirable to surpass are surpassed
- User approval
- Ease of maintenance
- Ease of audit
- Degree of absence of over-optimisation

What cost/time should be ignored in assessing aggregate performance of *programmers*?

- Wasted cost/time incurred because of requirements (inc architecture) changes
- Extra cost/time incurred because of deficiencies in platforms/tools

But this cost/time should be included in assessing aggregate performance of *teams* (inc architects, managers).

How can success of architects be measured?

Other measures plus:

- Instances of "avoiding early" being affected by deficiencies in platforms/tools
- Degree of absence of changes in architecture

Share Improve this answer Follow

edited Jan 13, 2010 at 16:45

answered Jan 13, 2010 at 0:22

martinr
3,802 ● 2 ● 19 ● 16

2

As I said in What is the fascination with code metrics?, metrics include:

- **different populations**, meaning the scope of interest is not the same for developer or for manager
- **trends** meaning any metrics in itself is meaningless without its associated trend, in order to take the decision to act upon it or to ignore it.

We are using a tool able to provide:

- lots of **micro-level metrics** (interesting for developers), with trends.
- lots of rules with multi-level (UI, Data, Code) static analysis capabilities
- lots of **aggregations rules** (meaning those vast number of metrics are condensed in several *domains* of interests, adequate for higher level of populations)

The result is an analysis which can be drilled-down, from high level aggregation *domains* (security, architecture, practices, documentation, ...) all the way down to some line of code.

The current feedback is:

- project managers can get defensive very quickly when some rules are not respected and make their global note significantly lower.
  Each study has to be re-tailored to respect each project quirks.
  The benefit is the definition of a contract where exceptions are acknowledged but rules to be respected are defined.
- higher levels (IT department, stakeholder) use the global notes just as one element of their evaluation of the progress made.
  They will actually look more closely at other elements based on delivery cycles: how often are we able to iterate and put an application into production?, how many errors did we had to solve before that release?

(in term of merges, or in term of pre-production environment not correctly setup), what immediate feedbacks are generated by a new release of an application?

So:

> which metrics are useful to which stakeholders, and at what level of aggregation

At high level:

- the (static analysis) metrics are actually the result of low-level metric aggregations, and organized by domains.
- Other metrics (more "operational-oriented", based on the release cycle of the application, and not *just* on the static analysis of the code) are taken into account
- The actual ROI is achieved through other actions (like six-sigma studies)

At lower level:

- the static analysis is enough (but has to encompass multi-level tiers applications, with sometimes multi-languages developments)
- the actions are piloted by the trends and importance
- the study has to be approved/supported by all levels of hierarchy to be accepted/acted upon (in particular,

budget for the ensuing refactoring has to be validated)

Share Improve this answer Follow

edited May 23, 2017 at 12:33

Community Bot

1 ● 1

answered Jan 11, 2010 at 8:36

VonC

1.3m ● 558 ● 4.7k ● 5.6k

2

If you have some Lean background/knowledge, then I would suggest the system that Mary Poppendieck recommends (that I've already mentioned in this previous answer). This system is based on three holistic measurements that must be taken as a package:

1. Cycle time
    - From product concept to first release or
    - From feature request to feature deployment or
    - From bug detection to resolution
2. Business Case Realization (**without this, everything else is irrelevant**)
    - P&L or
    - ROI or
    - Goal of investment
3. Customer Satisfaction
    - e.g. Net Promoter Score

The aggregation level is product/project level and I believe that these metrics are helpful for **everybody** (developers should never forget that they don't write code for fun, they write code to create value and should always keep that in mind).

Teams may (and actually do) use technical metrics to measure quality standards conformance which are integrated in the Definition of Done (as "no increase of the technical debt"). But high quality is not a end in itself, it's just a mean to achieve **short cycle time** (to be a fast company) which is the real target (with Business Case Realization and Customer Satisfaction).

Share Improve this answer

Follow

edited May 23, 2017 at 11:45

Community Bot

1 ● 1

answered Jan 12, 2010 at 0:05

Pascal Thivent

570k ● 140 ● 1.1k ● 1.1k

2

This is a bit of a side note to the main question, but I had a very similar experience to Paul Stephensons answer above. One thing I would add to that is about collection of data and visibility of metrics.

In our case, the development director was meant to collate a bunch of data from various disparate systems and distribute individual metric results once a month. This

often didn't happen, as it was a time consuming job and he was a busy man.

The results of this were:

1. Unhappy developers, as performance bonuses were based on metrics and people didn't know how they were getting on.
2. Some time consuming multiple entry of data into various different systems.

If you are going down this route, you need to be sure that all metric data can be collated automatically and is easily visible to those it affects.

Share Improve this answer

Follow

answered Jan 14, 2010 at 12:16

Paddy

33.8k ● 15 ● 81 ● 120

1

One of the interesting approaches that's currently getting some hype is Kanban. It's fairly Agile. What's particularly interesting is that it permits a metric of "work done" to be applied. I havn't used/encountered this in actual practice yet, but I'd like to work towards getting a kanban-ish flow going at my job.

Share Improve this answer

Follow

answered Jan 8, 2010 at 20:47

Paul Nathan

40.2k ● 30 ● 120 ● 215

I'm familiar with Kanban cards from books about lean processes. Worth further reading though. Thanks.
– David M Jan 8, 2010 at 20:54

Kanban is incredibly flexible and really gives you the ability to define your own processes with a firm underpinning. I'd definitely recommend this great paper introducing Kanban: blog.crisp.se/henrikkniberg/2009/04/03/1238795520000.html
– mopoke Jan 9, 2010 at 2:26

1

Interestingly I just finished reading PeopleWare, and the authors strongly discourage individual metrics being made visible to superiors (even direct managers), but that aggregate metrics *should* be very visible.

As far as code specific metrics I think it's good for a team to know the state of the code at the current time, and to know the trends affecting the code as it matures and grows.

The question is obviously not focussed on .NET, but I think the .NET product NDepend has done a lot of work to define and document common metrics that are useful.

The documentation section on metrics is educational reading, even if you're not doing .NET.

Share Improve this answer

Follow

answered Jan 13, 2010 at 3:13

John Weldon
40.7k ● 11 ● 96 ● 130

Thanks John. I'm already familiar with NDepend - my own team in this organisation is a .NET one, but the department as a whole is mixed, with probably slightly more Java than .NET going on. – David M Jan 13, 2010 at 10:01

1

Software metrics have been with us for a long time and as best I can tell nothing to date has emerged individually or in aggregate that is capable of guiding projects during development. The nut of the problem is that we want to use objective measures and these can only measure what has happened, not what is happening or about to happen.

By the time we have measured, analyzed and interpreted some series of metrics we are reacting to things that have already gone wrong, or very occasionally, gone right. I don't want to underplay the importance of learning from objective metrics but I do want to point out that this is a reactive not a pro-active response.

Developing a "confidence index" may be a better way of monitoring whether project is on-track or headed for trouble. Try developing a voting system where a reasonable number of representatives from each project area of interest are asked to anonymously vote their confidence from time to. Confidence is voted in two areas: 1) Things are on-track 2) Things will continue to be on-track or get back on-track. These are purely subjective measurements from people closest to the "action". Feed the results into a Kanban type chart where the columns

represent voting areas and you should have a pretty good idea where to focus your attention. Use question 1 to evaluate whether management reacted to the previous voting cycle appropriately. Use question 2 to identify where management should focus next.

This idea is based on each of us having a comfort level within our own area of responsibility. Our confidence level is a product of experience, knowledge within our domain of expertise, the number and severity of problems we are facing, the amount of time we have to accomplish our tasks, the quality of the information we are working with and a whole bunch of other factors.

MBWA (Management By Walking Around) is often touted as one of the most effective tools we have - this is a variation of it.

This technique is not much use at the level of individual teams because it only reflects the general mood of the team. Kind of like using someone's watch to tell them the time. However, at higher levels of management it should be quite informative.

answered Jan 13, 2010 at 17:38

NealB

16.9k ● 2 ● 42 ● 61

requirement helps protect this question from spam and non-answer activity.