# What are the most useful software development metrics? [closed]

Asked 16 years, 2 months ago Modified 5 years, 9 months ago

Viewed 23k times

27

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

Closed 12 years ago.

I would like to track metrics that can be used to improve my team's software development process, improve time estimates, and detect special case variations that need to be addressed during the project execution.

Please limit each answer to a single metric, describe how to use it, and vote up the good answers.

process metrics

Share

Improve this question

edited Aug 8, 2012 at 14:49

Follow

Bill the Lizard
**405k** ● 211 ● 572 ● 889

asked Oct 9, 2008 at 22:08

kchad
**289** ● 1 ● 3 ● 8

What does "improve my team's software development process" mean? I know what time estimates are. What are "special case variations"? How do you define them? – S.Lott Oct 9, 2008 at 22:29

For process improvement my thought is that once the metrics are in place changes to how we develop the software can be made and the metrics would help determine if the process change was a good one. – kchad Oct 10, 2008 at 2:58

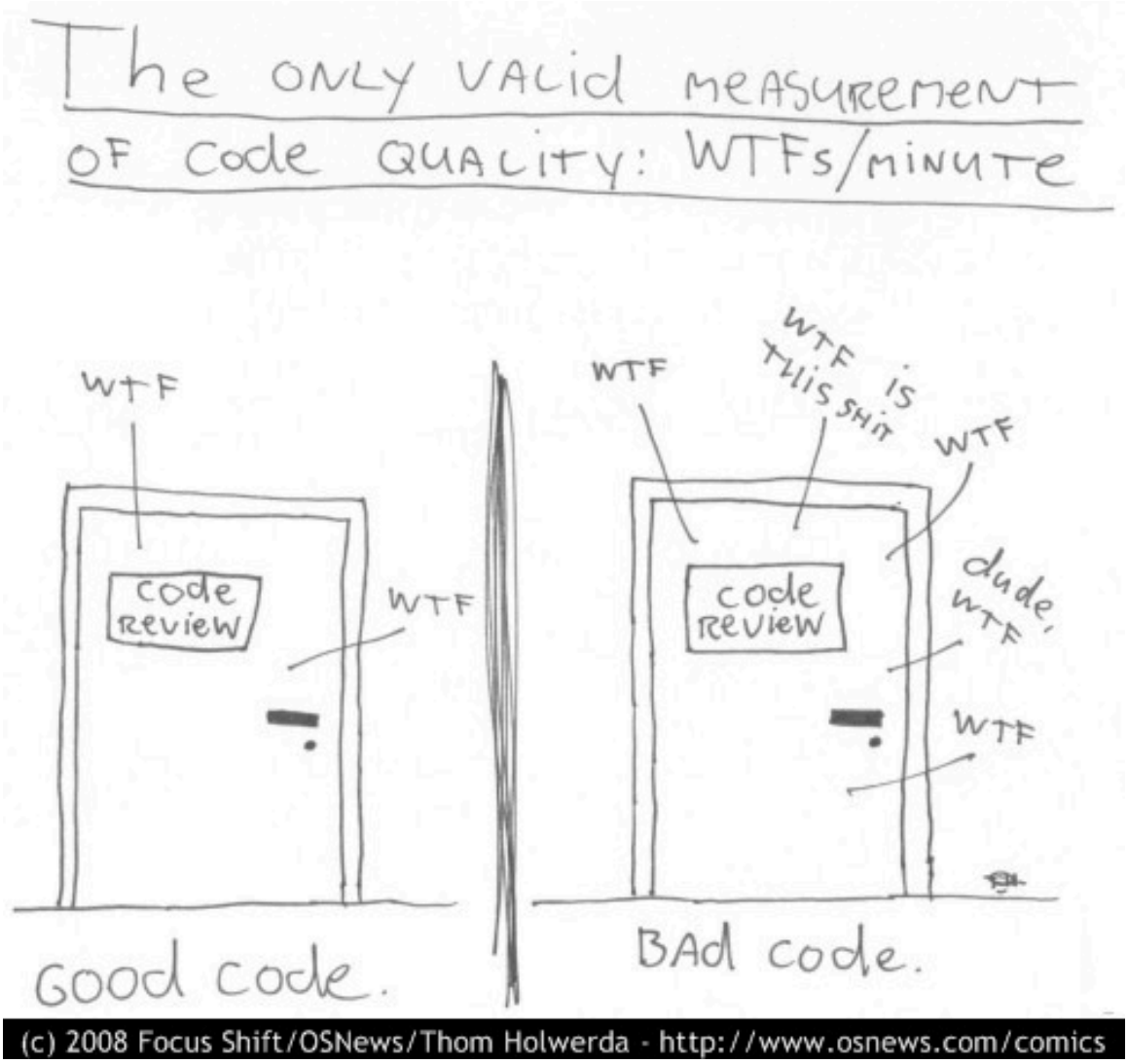
"special case variations" comes from what I remember from reading David Anderson's Agile Management book. An example of what I was getting at here was tracking a metric like WIP over time and seeing a sudden increase would warrant an investigation - there might be a problem there. – kchad Oct 10, 2008 at 3:04

I like the "Does it work?" test. ;) All funny business aside, I don't believe there is a single metric that is useful. You have to judge by the end product, time, stability, maintainability... Everything factors in. – willasaywhat Oct 10, 2008 at 14:32

## 26 Answers

Sorted by: Highest score (default)

51




(source: osnews.com)

Share Improve this answer

Follow

edited Mar 7, 2019 at 18:23

Glorfindel

**22.6k** ● 13 ● 89 ● 116

answered Mar 20, 2009 at 8:04

Mikeage

**6,554** ● 4 ● 38 ● 55

---

**ROI.**

13

The total amount of revenue brought in by the software minus the total amount of costs to produce the software. Breakdown the costs by percentage of total cost and isolate your poorest performing and most expensive area in terms of return-on-investment. Improve, automate, or eliminate that problem area if possible. Conversely, find your highest return-on-investment area and find ways to amplify its effects even further. If 80% of your ROI comes from 20% of your cost or effort, expand that particular area and minimize the rest by comparison.

Costs will include payroll, licenses, legal fees, hardware, office equipment, marketing, production, distribution, and support. This can be done on a macro level for a company as whole or a micro level for a team or individual. It can also be applied to time, tasks, and methods in addition to revenue.

This doesn't mean ignore all the details, but find a way to quantify everything and then concentrate on the areas that yield the best (objective) results.



edited Mar 20, 2009 at 7:49

answered Mar 19, 2009 at 22:51


VirtuosiMedia
53.3k ● 21 ● 98 ● 140

+1 Although I have to admit that I am AMAZED to see someone think of this! – Mark Brittingham Mar 19, 2009 at 22:56

Not a software metric by itself AFAIK. but a good one anyway +1 – SDReyes Sep 9, 2010 at 8:23

12

Inverse code coverage

Get a percentage of code not executed during a test. This is similiar to what Shafa mentioned, but the usage is different. If a line of code is ran during testing then we know it might be tested. But if a line of code has not been ran then we know for sure that is has not been tested. Targeting these areas for unit testing will improve quality and takes less time than auditing the code that has been covered. Ideally you can do both, but that never seams to happen.

Share Improve this answer

Follow

answered Oct 9, 2008 at 22:19

Rontologist

3,558 ● 1 ● 22 ● 24

This is better, but I'm not sure about this either. This is from a Java perspective, but there are lots of things that are of 0 importance to test. Accessors and Mutators would be the prime example, but there are others. How would you deal with those? – SCdF Oct 9, 2008 at 22:21

@SCdF - We do not include any generated code in our code coverage on my team unless someone has a strong opinion about it. Most getters and setters are generated from the IDE

for example, and we do not include them in our metrics.
– Rontologist Oct 9, 2008 at 22:25

Ahh, fair enough then :) – SCdF Oct 9, 2008 at 22:29

Could you point to any specific tools that do this?
– VirtuosiMedia Mar 23, 2009 at 14:49

I have been using EMMA for the projects that I have been on, and targeting classes with the lowest coverage manually.
– Rontologist Mar 25, 2009 at 20:43

11

"improve my team's software development process": Defect Find and Fix Rates

This relates to the number of defects or bugs raised against the number of fixes which have been committed or verified.

I'd have to say this is one of the really important metrics because it gives you two things:

- 1. Code churn. How much code is being changed on a daily/weekly basis (which is important when you are trying to stabilize for a release), and,
- 2. Shows you whether defects are ahead of fixes or vice-versa. This shows you how well the development team is responding to defects raised by the QA/testers.

A low fix rate indicates the team is busy working on other things (features perhaps). If the bug count is high, you might need to get developers to address some of the

defects.
A low find rate indicates either your solution is brilliant and almost bug free, or the QA team have been blocked or have another focus.

Share Improve this answer

Follow

answered Oct 10, 2008 at 14:34

RobS

**9,422** ● 3 ● 37 ● 63

1 I can't believe this had no upvotes, it was my immediate first choice. – Jeff Atwood Mar 20, 2009 at 8:12

I was a bit surprised too! This is a key metric IMHO – RobS Mar 20, 2009 at 10:52

**7**

Track how long is takes to do a task that has an estimate against it. If they were well under, question why. If they are well over, question why.

Don't make it a negative thing, it's fine if tasks blow out or were way under estimated. Your goal is to continually improve your estimation process.

Share Improve this answer

Follow

answered Oct 9, 2008 at 22:13

SCdF

**59.3k** ● 24 ● 79 ● 114

**7**

Track the source and type of bugs that you find.

The bug source represents the phase of development in which the bug was introduced. (eg. specification, design,

implementation etc.)

The bug type is the broad style of bug. eg. memory allocation, incorrect conditional.

This should allow you to alter the procedures you follow in that phase of development and to tune your coding style guide to try to eliminate over represented bug types.

Share Improve this answer

Follow

answered Oct 10, 2008 at 1:15

Andrew Edgecombe

**40.3k** ● 3 ● 38 ● 63

1 One of the very few frustrations I have with our agile methodology is that we never review where defects came from. When one developer "finishes" a feature and then spends half of the next two iterations fixing the wreckage left behind, it feels personally demoralizing. Just more time burned. – rektide Mar 19, 2009 at 22:39

@rektide: We have that where I work as well (we are working hard to improve it). It's a deserved slap in the face if we spend all our time fixing wreckage if we don't make an effort to figure out exactly where in the process defects (as you say) come from. – J M Mar 20, 2009 at 9:39

**7**

Velocity: the number of features per given unit time.

Up to you to determine how you define features, but they should be roughly the same order of magnitude otherwise velocity is less useful. For instance, you may classify your features by stories or use cases. These should be broken down so that they are all roughly the same size. Every

iteration, figure out how many stories (use-cases) got implemented (completed). The average number of features/iteration is your velocity. Once you know your velocity based on your feature unit you can use it to help estimate how long it will take to complete new projects based on their features.

[EDIT] Alternatively, you can assign a weight like function points or story points to each story as a measure of complexity, then add up the points for each completed feature and compute velocity in points/iteration.

Share Improve this answer Follow

edited Oct 10, 2008 at 10:47

answered Oct 9, 2008 at 22:13

tvanfosson

**532k** ● 102 ● 699 ● 798

Have you had success breaking down features to the same size? I have always liked the velocity idea but have had a hard time getting the features the same size. I have to admit I have bought but not yet read Agile Estimating and Planning and the FDD book... – kchad Oct 10, 2008 at 3:11

You can't "measure" features very accurately. You can use Function Points to score their complexity. The Function Point per Time metric is pretty common. – S.Lott Oct 10, 2008 at 10:17

For my purposes, yes -- sort of. I would say that they are all within about an order of magnitude. I have some stories that will take 2-3 hours and some that will take 2-3 days. Most are in the 2-3 days range, which is what I use for my estimates. I

ignore "aspect stories" when estimating. – tvanfosson Oct 10, 2008 at 10:45

4

Track the number of clones (similar code snippets) in the source code.

Get rid of clones by refactoring the code as soon as you spot the clones.

Share Improve this answer

Follow

answered Mar 20, 2009 at 7:55

Anonymous

**3,051** ● 4 ● 25 ● 25

Check out Simian as a tool for finding duplicate code.
– Ola Eldøy May 14, 2009 at 9:44

3

Average function length, or possibly a histogram of function lengths to get a better feel.

The longer a function is, the less obvious its correctness. If the code contains lots of long functions, it's probably a safe bet that there are a few bugs hiding in there.

Share Improve this answer

Follow

answered Oct 9, 2008 at 22:17

Simon Howard

**9,089** ● 5 ● 30 ● 21

number of failing tests or broken builds per commit.

2

Share Improve this answer

Follow

answered Oct 9, 2008 at 22:15

Ali Shafai

**5,161** ● 8 ● 36 ● 50

---

interdependency between classes. how tightly your code is coupled.

2

Share Improve this answer

Follow

answered Oct 9, 2008 at 23:36

Ali Shafai

**5,161** ● 8 ● 36 ● 50

---

Track whether a piece of source has undergone review and, if so, what type. And later, track the number of bugs found in reviewed vs. unreviewed code.

2

This will allow you to determine how effectively your code review process(es) are operating in terms of bugs found.

Share Improve this answer

Follow

answered Oct 10, 2008 at 1:18

Andrew Edgecombe

**40.3k** ● 3 ● 38 ● 63

---

If you're using Scrum, the backlog. How big is it after each sprint? Is it shrinking at a consistent rate? Or is stuff

2

being pushed into the backlog because of (a) stuff that wasn't thought of to begin with ("We need another use case for an audit report that no one thought of, I'll just add it to the backlog.") or (b) not getting stuff done and pushing it into the backlog to meet the date instead of the promised features.

Share Improve this answer

Follow

answered Oct 10, 2008 at 10:21

S.Lott

**391k** ● 82 ● 517 ● 788

---

2

[http://cccc.sourceforge.net/](http://cccc.sourceforge.net/)

Fan in and Fan out are my favorites.

Fan in: How many other modules/classes use/know this module

Fan out: How many other modules does this module use/know

Share Improve this answer

Follow

answered Oct 10, 2008 at 10:37

Ronny Brendel

**4,835** ● 5 ● 37 ● 55

---

2

> improve time estimates

While Joel Spolsky's Evidence-based Scheduling isn't per se a *metric*, it sounds like exactly what you want. See [http://www.joelonsoftware.com/items/2007/10/26.html](http://www.joelonsoftware.com/items/2007/10/26.html)

Share Improve this answer

Follow

answered Mar 19, 2009 at 22:20

Jonas Kölker
7,827 ● 3 ● 45 ● 52

---

2

I especially like and use the system that Mary Poppendieck recommends. This system is based on three holistic measurements that must be taken as a package (so no, I'm not going to provide 3 answers):

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

I don't need more to know if we are in phase with the ultimate goal: providing value to users, and fast.

Share Improve this answer

Follow

answered Jan 6, 2010 at 20:44

Pascal Thivent
570k ● 140 ● 1.1k ● 1.1k

number of similar lines. (copy/pasted code)

1

Share Improve this answer

Follow

answered Oct 9, 2008 at 22:14

Ali Shafai

**5,161** ● 8 ● 36 ● 50

1

improve my team’s software development process

It is important to understand that metrics can do nothing to improve your team’s software development process. All they can be used for is measuring how well you are advancing toward improving your development process in regards to the particular metric you are using. Perhaps I am quibbling over semantics but the way you are expressing it is why most developers hate it. It sounds like you are trying to use metrics to drive a result instead of using metrics to measure the result.

To put it another way, would you rather have 100% code coverage and lousy unit tests or fantastic unit tests and < 80% coverage?

Your answer should be the latter. You could even want the perfect world and have both but you better focus on the unit tests first and let the coverage get there when it does.

Share Improve this answer

Follow

answered Oct 10, 2008 at 14:25

Flory

2,849 ● 20 ● 31

I agree! My intention is to use the metrics as feedback - a way to detect potential problems or potential areas of process that could be improved. I have read that any single metric can be manipulated (and will if used as a measure of performance). I expect the best result from a combo of metrics. – kchad Oct 12, 2008 at 12:54

I disagree. Similar to the idea that attitude affects behavior and vice versa, tracking metrics will force you to evaluate your processes. No one is going to say "wow that metric sucks", and then do nothing about it. That being said metrics have their place in your understanding of the process, but should only be a piece of the puzzle. In other words, you need to be smart enough to know the limits of your metrics. – Trent Mar 11, 2011 at 22:01

1

Most of the aforementioned metrics are interesting but won't help you improve team performance. Problem is your asking a management question in a development forum.

Here are a few metrics: Estimates/vs/actuals at the project schedule level and personal level (see previous link to Joel's Evidence-based method), % defects removed at release (see my blog: http://redrockresearch.org/?p=58), Scope creep/month, and overall productivity rating (Putnam's productivity index). Also, developers bandwidth is good to measure.

Share Improve this answer

Follow

answered Jun 28, 2009 at 6:52



Mike J Berry

---

1

Every time a bug is reported by the QA team- analyze why that defect escaped unit-testing by the developers.

Consider this as a perpetual-self-improvement exercise.

Share Improve this answer

Follow

answered Jun 28, 2009 at 13:19

RN.

**1,007** ● 4 ● 15 ● 31

---

1

I like Defect Resolution Efficiency metrics. DRE is ratio of defects resolved prior to software release against all defects found. I suggest tracking this metrics for each release of your software into production.

Share Improve this answer

Follow

answered Jun 13, 2010 at 15:38

Mark Kofman

**639** ● 4 ● 12

---

1

Tracking metrics in QA has been a fundamental activity for quite some time now. But often, development teams do not fully look at how relevant these metrics are in relation to all aspects of the business. For example, the typical tracked metrics such as defect ratios, validity, test productivity, code coverage etc. are usually evaluated in

terms of the functional aspects of the software, but few pay attention to how they matter to the business aspects of software.

There are also other metrics that can add much value to the business aspects of the software, which is very important when an overall quality view of the software is looked at. These can be broadly classified into:

1. Needs of the beta users captured by business analysts, marketing and sales folks
2. End-user requirements defined by the product management team
3. Ensuring availability of the software at peak loads and ability of the software to integrate with enterprise IT systems
4. Support for high-volume transactions
5. Security aspects depending on the industry that the software serves
6. Availability of must-have and nice-to-have features in comparison to the competition
7. And a few more….

Share Improve this answer

Follow

answered Sep 19, 2011 at 12:02

Ricky Ponting

21 • 3

Code coverage percentage

0

Share Improve this answer

Follow

answered Oct 9, 2008 at 22:12

Ali Shafai

5,161 ● 8 ● 36 ● 50

I would strongly argue against this. Coverage just means you've executed that line, and thus it must compile. It doesn't say either that it's relevant to test or that it's correct. – SCdF Oct 9, 2008 at 22:14

you mean not having it at all is better? at least if you get 10%, you know it's not even executed... – Ali Shafai Oct 9, 2008 at 22:16

I'm saying that when you make code coverage a metric then it's just a hoop that developers jump through. They can they say "See, we have 100% coverage!" when in reality what you want is each distinct piece of logic to have separate unit tests that validate it. That is far more important than coverge – SCdF Oct 9, 2008 at 22:18

0

If you're using Scrum, you want to know how each day's Scrum went. Are people getting done what they said they'd get done?

Personally, I'm bad at it. I chronically run over on my dailies.

Share Improve this answer

Follow

answered Oct 10, 2008 at 10:18

S.Lott

391k ● 82 ● 517 ● 788

0

Perhaps you can test CodeHealer

CodeHealer performs an in-depth analysis of source code, looking for problems in the following areas:

- **Audits** Quality control rules such as unused or unreachable code, use of directive names and keywords as identifiers, identifiers hiding others of the same name at a higher scope, and more.
- **Checks** Potential errors such as uninitialised or unreferenced identifiers, dangerous type casting, automatic type conversions, undefined function return values, unused assigned values, and more.
- **Metrics** Quantification of code properties such as cyclomatic complexity, coupling between objects (Data Abstraction Coupling), comment ratio, number of classes, lines of code, and more.

Share Improve this answer

Follow

answered Jan 6, 2010 at 20:36



Hugues Van Landeghem

6,783 ● 4 ● 36 ● 61

-1

Size and frequency of source control commits.

Share Improve this answer

answered Oct 10, 2008 at 14:36

Follow

dacracot

**22.3k** ● 28 ● 109 ● 159

Sounds like a sneaky way of implementing a LOC metric. – JohnFx Mar 19, 2009 at 22:40

@JohnFx, what if the commits are actually *deleting* code, as they sculpt the simplest, most elegant code possible... (or, say, refactoring). – John C Jun 13, 2010 at 15:53

I'm not saying that source control commits are a bad thing. Just that they aren't a good metric of progress. The could just as easily be developmentstruction. – JohnFx Jun 13, 2010 at 19:41