What is the fascination with code metrics? [closed]

Asked 16 years, 2 months ago Modified 5 years, 1 month ago

Viewed 13k times

89

I've seen a number of 'code metrics' related questions on SO lately, and have to wonder what the fascination is? Here are some recent examples:

- what code metrics convince you that provided code is crappy
- when if ever is number of lines of code a useful metric
- writing quality tests

In my mind, no metric can substitute for a code review, though:

- some metrics sometimes may indicate places that need to be reviewed, and
- radical changes in metrics over short time frames may indicate places that need to be reviewed

But I cannot think of a single metric that by itself always indicates 'good' or 'bad' code - there are always exceptions and reasons for things that the measurements cannot see.

Is there some magical insight to be gained from code metrics that I've overlooked? Are lazy programmers/managers looking for excuses not to read code? Are people presented with giant legacy code bases and looking for a place to start? What's going on?

> Note: I have asked some of these questions on the specific threads both in answers and comments and got no replies, so I thought I should ask the community in general as perhaps I am missing something. It would be nice to run a metrics batch job and not actually have to read other people's code (or my own) ever again, I just don't think it is practical!

EDIT: I am familiar with most if not all of the metrics being discussed, I just don't see the point of them in isolation or as arbitrary standards of quality.

code-metrics

Share

Improve this question

Follow

edited May 23, 2018 at 14:37

Scath
3,814 ● 10 ● 31 ● 40

asked Oct 12, 2008 at 19:23

Steven A. Lowe
61.1k ● 19 ● 135 ● 204

2 My code metric for C# code is `the number of StyleCop warnings + 10 * the number of FxCop warnings + 2 to the power of the number of disabled warning types`. Only after the value of that metric is as small as possible, is it worth for a human to start reviewing the code (in my opinion). In sum: sophisticated tools rather than simplistic formulas can help improve code quality. This is probably off-topic though. – Hamish Grubijan Aug 4, 2010 at 16:09

@Alfred - `I just don't see the point of them in isolation or as arbitrary standards of quality.` - Who would think of using metrics in isolation or as arbitrary standards of quality? – luis.espinal Jul 13, 2011 at 14:50

@luis that was my edit, based on several questions that spawned this question - the answer is "management", primarily – Steven A. Lowe Jul 14, 2011 at 15:37

18 Answers

Sorted by: Highest score (default)

88

The answers in this thread are kind of odd as they speak of:

- "the team", like "the one and only beneficiary" of those said metrics;
- "the metrics", like they mean anything in themselves.

1/ Metrics is not for *one* population, but for *three*:

- developers: they are concerned with *instantaneous* **static code metrics** regarding static analysis of their code (cyclomatic complexity, comments quality, number of lines, ...)
- project leaders: they are concerned with *daily* **live code metrics** coming from unit test, code coverage, continuous integration testing
- business sponsors (they are always forgotten, but they are the stakeholders, the one paying for the development): they are concerned with *weekly* **global code metrics** regarding architectural design, security, dependencies, ...

All those metrics can be watched and analyzed by all three populations of course, but each kind is designed to be better used by each specific group.

2/ Metrics, by themselves, represent a ***snapshot*** of the code, and that means... nothing!

It is the combination of those metrics, and the combinations of those different levels of analysis that may indicate a "good" or "bad" code, but more importantly, it is ***the trend of those metrics*** that is significant.

That is the *repetition* of those metrics what will give the real added value, as they will help the business managers/project leaders/developers to ***prioritize*** amongst the different possible code fixes

---

In other words, your question about the "fascination of metrics" could refer to the difference between:

- "beautiful" code (although that is always in the eye of the beholder-coder)
- "good" code (which works, and can prove it works)

So, for instance, a function with a cyclomatic complexity of 9 could be defined as "beautiful", as opposed of one long convoluted function of cyclomatic complexity of 42.

BUT, if:

- the latter function has a *steady* complexity, *combined with* a code coverage of 95%,
- whereas the former has an *increasing* complexity, *combined with* a coverage of... 0%,

one could argue:

- the the latter represents a "*good*" code (it works, it is stable, and if it need to change, one can checks if it still works after modifications),
- the former is a "*bad*" code (it still need to add some cases and conditions to cover all it has to do, and there is no easy way to make some regression test)

So, to summarize:

> a single metric that by itself always indicates [...]

: not much, except that the code may be more "beautiful", which in itself does not mean a lot...

> Is there some magical insight to be gained from code metrics that I've overlooked?

Only the *combination* and *trend* of metrics give the real "magical insight" you are after.

Share Improve this answer

Follow

edited Dec 23, 2010 at 15:47

Craig P. Motlin
26.7k ● 18 ● 103 ● 127

answered Oct 12, 2008 at 19:49

VonC
1.3m ● 558 ● 4.7k ● 5.6k

8 "Beautiful" code may well be easier to maintain - and if so, THAT has a lot of value! – Richard T Dec 28, 2008 at 5:07

23

I had a project that I did as a one person job measured for cyclomatic complexity some month ago. That was my first exposure to these kind of metrics.

The first report I got was shocking. Almost all of my functions failed the test, even the (imho) very simple ones. I got around the complexity thing by moving logical sub-task into subroutines even if they have been called only once.

For the other half of the routines my pride as a programmer kicked in and I tried to rewrite them in a way that they do the same, just simpler and more readable. That worked and I was able to get most down to the customers yclomatic complexity threshold.

In the end I was almost always able to come up with a better solution and much cleaner code. The performance did not suffered from this (trust me - I'm paranoid on this, and I check the disassembly of the compiler output quite often).

I think metrics are a good thing if you use them as a reason/motivation to improve your code. It's imortant to know when to stop and ask for a metric violation grant though.

Metrics are guides and helps, not ends in itself.

Share Improve this answer

Follow

answered Oct 12, 2008 at 19:51

Nils Pipenbrinck

86.2k ● 33 ● 155 ● 223

18

The best metric that I have ever used is the C.R.A.P. score.

Basically it's an algorithm that compares weighted cyclomatic complexity with automated test coverage. The algorithm looks like this: `CRAP(m) = comp(m)^2 * (1 – cov(m)/100)^3 + comp(m)` where comp(m) is the cyclomatic complexity of method m, and cov(m) is the test code coverage provided by automated tests.

The authors of the afore mentioned article (please, go read it...it's well worth your time) suggest a max C.R.A.P. score of 30 which breaks down in the following way:

```
Method’s Cyclomatic Complexity        % of
coverage required to be
                                      below CRAPpy
threshold
------------------------------        ------------
--------------------
0 – 5                                   0%
10                                     42%
15                                     57%
20                                     71%
25                                     80%
30                                    100%
31+                                   No amount of
testing will keep methods
```

```
                                         this complex
out of CRAP territory.
```

As you quickly see, the metric rewards writing code that is not complex coupled with good test coverage (if you are writing unit tests, and you should be, and are not measuring coverage...well, you probably would enjoy spitting into the wind as well). ;-)

For most of my development teams I tried really hard to get the C.R.A.P. score below 8, but if they had valid reasons to justify the added complexity that was acceptable as long as they covered the complexity with sufficient tests. (Writing complex code is always very difficult to test...kind of a hidden benefit to this metric).

Most people found it hard initially to write code that would pass the C.R.A.P. score. But over time they wrote better code, code that had fewer problems, and code that was a lot easier to debug. Out of any metric, this is the one that has the fewest concerns and the greatest benefit.

Share Improve this answer Follow

edited Oct 25, 2019 at 16:33
Simeon Leyzerzon
19k ● 10 ● 59 ● 91

answered Jul 27, 2010 at 17:06

RMatthews
375 ● 3 ● 7

For me the single most important metric that identifies bad code is cyclomatic complexity. Almost all methods in

13

my projects are below CC 10 and bugs are invariably found in legacy methods with CC over 30. High CC usually indicates:

- code written in haste (ie. there was no time to find an elegant solution and not because the problem required a complex solution)
- untested code (no one writes tests for such beasts)
- code that was patched and fixed numerous times (ie. riddled with ifs and todo comments)
- a prime target for refactoring

Share Improve this answer Follow

answered Oct 12, 2008 at 21:36

Goran
6,826 ● 9 ● 43 ● 57

10

A good code review is no substitute for a good static analysis tool, which is of course not substitute for a good set of unit tests, now unit tests are no good without a set of acceptance tests......

Code metrics are another tool to put into your tool box, they are not a solution in their own right they are just a tool to be used as appropriate (with of course all the other tools in your box!).

Share Improve this answer Follow

answered Oct 12, 2008 at 19:42

Scott James
684 ● 1 ● 5 ● 12

6

People are drawn to the idea of mechanistic ways to understand and describe code. If true, think of the ramifications for efficiency and productivity!

I agree that a metrics for "code goodness" is about as sensible as a metric for "good prose." However that doesn't mean metrics are useless, just perhaps misused.

For example, *extreme* values for some metrics *point the way* to possible problems. A 1000-line-long method is *probably* unmaintainable. Code with zero unit test code coverage *probably* has more bugs that similar code with lots of tests. A big jump in code added to a project just before release that isn't a third-party library is *probably* cause for extra attention.

I think if we use metrics as a suggestion -- a red flag -- perhaps they can be useful. The problem is when people start measuring productivity in SLOC or quality in percentage of lines with tests.

Share Improve this answer Follow

answered Oct 12, 2008 at 19:31

Jason Cohen

83k ● 26 ● 110 ● 114

6

My highly subjective opinion is that code metrics expresses the irresistable institutional fascination with being able to quantify something inherently unquantifiable.

Makes sense, in a way, at least psychologically - how can you make decisions on something you can't evaluate or understand? Ultimately, of course, you can't evaluate quality unless you're knowledgeable about the subject (and are at least as good as the what you're trying to evaluate) or ask someone who's knowledgeable, which of course just puts the problem back one step.

In that sense, maybe a reasonable analogy would be evaluating college entrants by SAT scores, it's unfair and misses every kind of subtlety but if you need to quantify you've got to do something.

Not saying I think it's a good measure, only that I can see the intitutional irresistability of it. And, as you pointed out, there are probably a few reasonable metrics (lots of 500+ line methods , high complexity-probably bad). I've never been at a place that bought into this,though.

Share Improve this answer Follow

answered Oct 12, 2008 at 19:31

Steve B.
57.2k ● 12 ● 97 ● 134

There's one code metric I believe in.

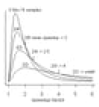
6

I'm working on a big system. When a single new requirement comes to me, I set about coding it up. When I'm done and got the bugs worked out, I check it into the version control system. That system does a diff, and counts up all the changes I made.

The smaller that number is, the better.

Share Improve this answer Follow

answered Nov 22, 2008 at 14:50



Mike Dunlavey
40.6k ● 15 ● 94 ● 138

2 That would be true considering that the previous code was developed by yourself or another developer with equivalent or better skills. If on the other side the code was developed by a developer with fewer skills, an increasing number of lines in the diff (changed lines + new lines + lots of deleted lines) may actually mean an improvement in the code as you are getting rid of poor quality code. – Alfred Myers Sep 22, 2009 at 21:50

@Alfred: Sure. I'm talking ideal-world, and averaged over a number of requirement changes. Here's an example of what I'm talking about, and it does have a learning curve: stackoverflow.com/questions/371898/… – Mike Dunlavey Sep 23, 2009 at 11:37

How do you know you did a good job if you have no baseline to compare it to? – J S Sep 29, 2009 at 18:17

Metrics and automated tests aren't meant to be a replacement for full code reviews.

5

They just speed things up. With an automated checker, it's very easy to see which conventions you've forgotten to follow, that you're using the designated packages and methods, etc. You can see what you can fix without using other people's time.

Managers also like metrics them because they feel they're getting an exact figure on productivity (though that's often not really the case) and they should be able to juggle people better.

Share Improve this answer Follow

answered Oct 12, 2008 at 19:31

Oli
240k ● 65 ● 226 ● 303

5

Measurements are only useful if:

- The team developed them
- The team agreed to them
- They are being used to identify a specific area

In general, any metric that doesn't fit into that will suffer from the team optimizing to it. You want to measure lines of code? By gosh, watch how many they can write! You want to measure code coverage, by golly, watch me cover that code!

I think metrics can be useful for identifying trends, and in fact, I've seen some useful ones, such as plotting when the build breaks, code churn (number of lines of code changing throughout the project) and other things. But if the team isn't coming up with them, or they don't agree or understand them, you are likely in a world of hurt.

Share Improve this answer Follow

answered Oct 12, 2008 at 19:32

Cory Foy
7,212 ● 4 ● 32 ● 34

5

Here is some Complexity Metrics from stan4j.

An eclipse class structure analyze tool.

I like this tool and the metrics. I treat the metrics as statistics, indicators, warning messages. Sometime due to some methods or some classes really has some complicated logic made them to be complex, what shall be done is keep an eye on them, review them to see if there is an need to refactor them or review them carefully, due to normally they are error prone. Also I use it as analyze tool to learn source code, due to I like to learn from complex to simple.Actually it includes some other metrics such as Robert C. Martin Metrics, Chidamber & Kemerer Metrics,Count Metrics But I like this one best

Complexity Metrics

Cyclomatic Complexity Metrics

Cyclomatic Complexity (CC) The cyclomatic complexity of a method is the number of decision points in the method's control flow graph incremented by one.

Decision points occur at if/for/while statements, case/catch clauses and similar source code elements, where the control flow is not just linear. The number of (byte code) decision points introduced by a single (source code) statement may vary, depending e.g. on the complexity of boolean expressions. The higher the cyclomatic complexity value of a method is, the more test cases are required to test all the branches of the method's control flow graph.

Average Cyclomatic Complexity Average value of the Cyclomatic Complexity metric over all methods of an application, library, package tree or package.

Fat Metrics The Fat metric of an artifact is the number of edges in an appropriate dependency graph of the artifact. The dependency graph type depends on the metric variant and the chosen artifact:

Fat The Fat metric of an application, library or package tree is the edge count of its subtree dependency graph. This graph contains all the artifact's children in the package tree hierarchy, thereby also including leaf packages. (To see the appropriate graph in the Composition View, the Structure Explorer's Flat Packages toggle has to be disabled. The Show Libraries toggle has to be enabled if the chosen artifact is a library, otherwise it has to be disabled.)

The Fat metric of a package is the edge count of its unit dependency graph. This graph contains all top level classes of the package.

The Fat metric of a class is the edge count of its member graph. This graph contains all fields, methods and member classes of the class. (This graph and the Fat value are only available if the code analysis was performed with Level of Detail Member, not Class.)

Fat for Library Dependencies (Fat - Libraries) The Fat for Library Dependencies metric of an application is the edge count of its library dependency graph. This graph contains all libraries of the application. (To see the appropriate graph in the Composition View, the Structure Explorer's Show Libraries toggle has to be enabled.)

Fat for Flat Package Dependencies (Fat - Packages) The Fat for Flat Package Dependencies metric of an application is the edge count of its flat package dependency graph. This graph contains all packages of the application. (To see the appropriate graph in the Composition View, the Structure Explorer's Flat Packages toggle has to be enabled and the Show Libraries toggle has to be disabled.)

The Fat for Flat Package Dependencies metric of a library is the edge count of its flat package dependency graph. This graph contains all packages of the library. (To see the appropriate graph in the Composition View, the Structure Explorer's Flat Packages and Show Libraries toggles have to be enabled.)

Fat for Top Level Class Dependencies (Fat - Units) The Fat for Top Level Class Dependencies metric of an application or library is the edge count of its unit dependency graph. This graph contains all the top level classes of the application or library. (For reasonable applications it is too large to be visualized and thus can not be displayed in the Composition View. Unit dependency graphs may only be displayed for packages.)

Share Improve this answer

Follow

edited Oct 25, 2019 at 16:32

Simeon Leyzerzon
19k ● 10 ● 59 ● 91

answered Aug 18, 2011 at 12:58

Clark Bao
1,693 ● 3 ● 22 ● 40

2

Metrics may be useful to determine the improvement or degradation in a project, and can certainly find style and convention violations, but there is no substitute for doing peer code reviews. You can't possibly know the quality of your code without them.

Oh ... and this assumes that at least one of the participants in your code review has a clue.

Share Improve this answer

Follow

answered Oct 12, 2008 at 19:33

Steve Moyer
5,733 ● 1 ● 26 ● 34

2

I agree with you that code metrics should not substitute a code review but I believe that they should complement code reviews. I think it gets back to the old saying that "you cannot improve what you cannot measure." Code metrics can provide the development team with quantifiable "code smells" or patterns that may need further investigation. The metrics that are captured in most static analysis tools are typically metrics that have been identified over the course of research in our field's short history to have significant meaning.

Share Improve this answer Follow

answered Oct 12, 2008 at 19:37

SaaS Developer
9,895 ● 8 ● 37 ● 45

2

Metrics are not a substitute for code review, but they're far cheaper. They're an indicator more than anything.

Share Improve this answer Follow

answered Oct 12, 2008 at 19:42

Andy Lester
93.5k ● 15 ● 104 ● 159

2

One part of the answer is that some code metrics can give you a very quick, initial stab at an answer to the question: What is this code like?

Even 'lines of code' can give you an idea of the size of the code base you are looking at.

As mentioned in another answer, the trend of the metrics gives you the most information.

Share Improve this answer Follow

answered Oct 12, 2008 at 21:02

quamrana
39.3k ● 13 ● 55 ● 76

2

Metrics of themselves are not particularly interesting. It's what you do with them that counts.

For example if you were measuring the number of comments per line of code what would you consider a good value? Who knows? Or perhaps more importantly, everyone has their own opinion.

Now if you collect enough information to be able to correlate the number of comments per line of code against the time taken to resolve a bugs or against the number of bugs found that are attributed to coding, then you may start to find an empirically useful number.

There is no difference between using metrics in software and using any other performance measure on any other process - first you measure, then you analyse, then you improve the process. If all you're doing is measuring, you're wasting your time.

edit: In response to Steven A. Lowe's comments - that's absolutely correct. In any data analysis one must be careful to distinguish between causal relationship and a mere correlation. And the selection of the metrics on the basis of suitability is important. There is no point in trying to measure coffee consumption and to attribute code quality (although I'm sure some have tried ;-))

But before you can find the relationship (causal or not) you have to have the data.

The selection of the data to collect is based on what process you wish to verify or improve. For example, if you're trying to analyse the success of your code review procedures (using your own definition for "success", be that reduced bugs or reduced coding bugs, or shorter turnaround time or whatever), then you select metrics that measure the total rate of bugs and the rate of bugs in reviewed code.

So before you collect the data you have to know what you want to do with it. If metrics is the means, what is the end?

Share Improve this answer Follow

edited Oct 13, 2008 at 6:43

answered Oct 12, 2008 at 22:58

Andrew Edgecombe
40.3k ● 3 ● 38 ● 63

i would agree, except that what you measure in order to improve is critical. If you want to reduce defects in a manufacturing process but all you measure is the number of defects and the amount of coffee consumed in the break room, you're probably not going to get anywhere
– Steven A. Lowe Oct 13, 2008 at 1:46

in other words there are no established correlations to use as a standard; are you recommending the use of metrics and correlation to establish a standard? if so, can you demonstrate a causal linkage or are we again measuring coffee consumption? – Steven A. Lowe Oct 13, 2008 at 1:47

2

I don't think small changes in metrics are meaningful: a function with complexity 20 is not necessarily cleaner than a function with complexity 30. But it's worth running metrics to look for large differences.

One time I was surveying a couple dozen projects and one of the projects had a maximum complexity value around 6,000 while every other project had a value around 100 or less. That hit me over the head like a baseball bat. Obviously something unusual, and probably bad, was going on with that project.

Share Improve this answer Follow

answered Nov 22, 2008 at 15:22

John D. Cook
30k ● 10 ● 69 ● 94

did you look at the project? what was the cause of the huge difference in the metric? – Steven A. Lowe Nov 22, 2008 at 16:25

The project with 6K complexity started out poorly written, then got worse as it evolved under extreme pressure.
– John D. Cook Jan 12, 2009 at 3:43

1

We're programmers. We like numbers.

Also, what are you going to do, NOT describe the size of the codebase because "lines of code metrics are irrelevant"?

There is definitely a difference between a codebase of 150 lines and one of 150 million, to take a silly example. And it's not a hard number to get.

Share Improve this answer Follow

answered Oct 12, 2008 at 21:40

Thomas David Baker
1,129 ● 10 ● 26

1 there is a difference, one has more lines of code. But so what? That says nothing about the quality of the software... – Steven A. Lowe Oct 13, 2008 at 1:35

2 I know which one I'd choose to work on next ;-) – quamrana Nov 23, 2008 at 22:41