Code Churn

Yegor Bugayenko

Lecture #13 out of 24
80 minutes

The slidedeck was presented by the author in this YouTube Video

History of Version Control Systems (VCS)

- The **Librarian** by ADR: created in 1969
- **Panvalet** by Computer Associates: 1969
- **SCCS** by Bell Labs: 1973
- **RCS** by GNU: 1982
- **CVS** by Dick Grune: 1986
- **ClearCase** by IBM: 1992
- **Subversion** by Apache: 2000
- **BitKeeper** by BitMover Inc.: 2000
- **Arch** by Thomas Lord in GNU: 2001
- **Git** by Linus Torvalds: 2005

- **Mercurial** by Olivia Mackall: 2005

Sebastian G. Elbaum

"We can measure the increase or decrease in system complexity as measured by a selected metric, code delta, or we can measure the total amount of change the system has undergone between builds, code churn."

— J. C. Munson and S. G. Elbaum. Code Churn: A Measure for Estimating the Impact of Code Change. In *Proceedings of the International Conference on Software Maintenance*, 1998. doi:10.1109/icsm.1998.738486

What is “Churn”?

Wikipedia says that “churn rate (also known as attrition rate, turnover, customer turnover, or customer defection) is a measure of the proportion of individuals or items moving out of a group over a specific period.”

Barrel Butter Churn, Victorian, Original, $95.00

Motivating Example

Commit #1:

0

```
class Book
  private final int id;
  public Book(int it)
    this.id = i;
```

Commit #2:

+5

```
class Book
  private int id;
  public Book(int it)
    this.id = i;
  public int getId()
    return this.id;
  private int setId(int i)
    this.id = i;
```

Commit #3:

+2 / -2

```
final class Book
  private final int id;
  public Book(int it)
    this.id = i;
  public int getId()
    return this.id;
```

Delta: +3 / 0, Churn: +7 / -2

BUILD	RCM	DELTA	CHURN
1	184643.38	5.72	7.12
2	184648.81	5.43	16.78
3	185702.10	1053.28	1547.93
4	185857.61	155.50	187.11
5	186236.58	378.97	429.30
6	186261.81	25.23	294.18
7	186083.21	-178.60	791.87
8	186726.70	643.50	836.71
9	187072.37	345.65	426.24
10	187157.08	84.71	86.74
11	189945.91	2788.85	2804.02
12	190000.81	54.89	59.28
13	190007.10	6.29	6.38
14	190012.49	5.39	5.48
15	190069.36	56.87	75.08
16	190066.48	-2.88	3.86
17	190067.65	1.17	1.17
18	190067.81	0.16	0.21

Table 2: Evolution Data for QTB

"A limitation of measuring code deltas is that it doesn't give an indicator as to how much change the system has undergone. If several software modules are removed and are replaced by modules of roughly equivalent complexity, the code delta for the system will be close to zero."

Source: J. C. Munson and S. G. Elbaum. Code Churn: A Measure for Estimating the Impact of Code Change. In *Proceedings of the International Conference on Software Maintenance*, 1998. doi:10.1109/icsm.1998.738486

Table 5
Code delta and code churn for fault index

ρ	Δ_ρ	∇_ρ
12 to 13	98.96	233.99
13 to 14	242.99	377.33
14 to 15	376.34	666.22
15 to 16	462.30	482.10
16 to 17	404.97	604.26
17 to 18	107.11	4387.03
18 to 19	−23.79	156.08
19 to 20	368.12	373.15
20 to 21	368.99	1027.32
21 to 22	76.71	1446.61
22 to 23	−211.42	1055.70

"In the case of code churn, what is important is the absolute measure of the amount of code that has been modified. From the standpoint of fault insertion (FI), removing a lot of code is probably as catastrophic as adding a similar amount."

Source: Gregory A. Hall and John C. Munson. Software Evolution: Code Delta and Code Churn. *Journal of Systems and Software*, 54(2):111–118, 2000. doi:10.1016/S0164-1212(00)00031-5

Nachiappan Nagappan

"Our case study provides strong support for the following conclusion: increase in relative code churn measures is accompanied by an increase in system defect density."

— Nachiappan Nagappan and Thomas Ball. Use of Relative Code Churn Measures to Predict System Defect Density. In *Proceedings of the 27th International Conference on Software Engineering*, pages 284–292, 2005. doi:10.1145/1062455.1062514

Table A10
Designer assignment, designer turnover, and code churn

Product	Pre-release			
	No. of designers	Designers turnover	Code churn	Development time (months)
P(C1)	28	45	23,830	24
P(F1)	37	50	422,964	16
P(I1)	57	90	222,945	30
P(E1)	36	52	207,412	17
C3	23	22	194,937	16
P(F3)	33	29	425,593	15
P(I3)	38	35	112,424	8
P(E10)	66	93	633,629	13

"During the pre-release phase, there seems to be a positive relationship between designer turnover and the impact of change (i.e., the ratio between code churn and product growth for a particular product). A similar relationship does not seem to hold for post-release changes."

Source: Samuel A. Ajila and Razvan T. Dumitrescu. Experimental Use of Code Delta, Code Churn, and Rate of Change to Understand Software Product Line Evolution. *Journal of Systems and Software*, 80(1): 74–91, 2007. doi:10.1016/j.jss.2006.05.034

Georgios Gousios

"We examined 291 carefully selected Ruby, Python, Java and Scala projects (in total, 166,884 pull requests)... The number of total lines changed by pull requests is on average less than 500 (95% percentile: **1227**, 90% percentile: 497, 80% percentile: 168) with a median number of 20."

— Georgios Gousios, Martin Pinzger, and Arie van Deursen. An Exploratory Study of the Pull-Based Software Development Model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355, 2014. doi:10.1145/2568225.2568260

Rudolf Ferenc

"We can conclude that committing files with higher cumulative code churn values (i.e. those of longer change history) is more likely to result in negative maintainability change."

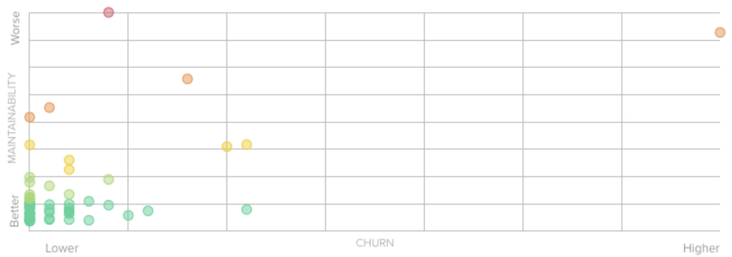
— Csaba Faragó, Péter Hegedűs, and Rudolf Ferenc. Cumulative Code Churn: Impact on Maintainability. In *Proceedings of the 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 141–150. IEEE, 2015. doi:10.1109/scam.2015.7335410

Tobias Olsson

"Non-normal code churn can be a tangible and measurable effect of architectural violations in source code. However, more research is needed to better understand this connection, e.g., if violations are the cause, the size of the effect, the cost of refactoring, and whether it is generalizable to other system."

— Tobias Olsson, Morgan Ericsson, and Anna Wingkvist. The Relationship of Code Churn and Architectural Violations in the Open Source Software JabRef. In *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*, 2017. doi:10.1145/3129790.3129810

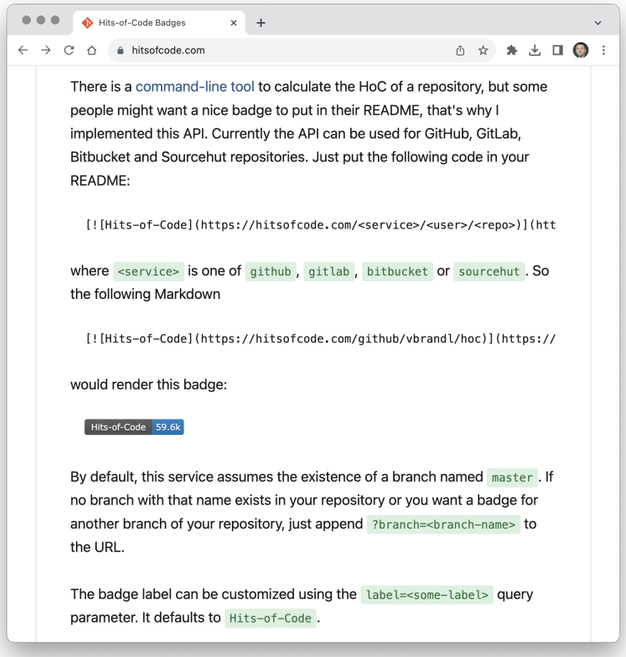
CodeClimate.com Interpretation of Code Churn



"When we have finished analyzing your default branch, you will see churn metrics for the files in your repository. This churn metric is approximately the number of times a file has changed in the last 90 days, calculated by counting the number of distinct versions of that file across commits from that time."

Source: their blog

HitsOfCode.com



Created by Valentin Brandl (@vbrandl) and inspired by the blog post of mine [Bugayenko, 2014].

Bibliography

Samuel A. Ajila and Razvan T. Dumitrescu. Experimental Use of Code Delta, Code Churn, and Rate of Change to Understand Software Product Line Evolution. *Journal of Systems and Software*, 80(1):74–91, 2007. doi:10.1016/j.jss.2006.05.034.

Yegor Bugayenko. Hits-of-Code Instead of SLoC. `https://www.yegor256.com/141114.html`, 11 2014. [Online; accessed 15-12-2024].

Csaba Faragó, Péter Hegedűs, and Rudolf Ferenc. Cumulative Code Churn: Impact on Maintainability. In *Proceedings of the 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 141–150. IEEE, 2015. doi:10.1109/scam.2015.7335410.

Georgios Gousios, Martin Pinzger, and Arie van Deursen. An Exploratory Study of the Pull-Based Software Development Model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355, 2014. doi:10.1145/2568225.2568260.

Gregory A. Hall and John C. Munson. Software Evolution: Code Delta and Code Churn. *Journal of Systems and Software*, 54(2):111–118, 2000. doi:10.1016/S0164-1212(00)00031-5.

J. C. Munson and S. G. Elbaum. Code Churn: A Measure for Estimating the Impact of Code Change. In *Proceedings of the International Conference on Software Maintenance*, 1998. doi:10.1109/icsm.1998.738486.

Nachiappan Nagappan and Thomas Ball. Use of Relative Code Churn Measures to Predict System Defect Density. In *Proceedings of the 27th International Conference on Software Engineering*, pages 284–292, 2005. doi:10.1145/1062455.1062514.

Tobias Olsson, Morgan Ericsson, and Anna Wingkvist. The Relationship of Code Churn and Architectural Violations in the Open Source Software JabRef. In *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*, 2017. doi:10.1145/3129790.3129810.