Measuring the Software Engineering Process

A REPORT

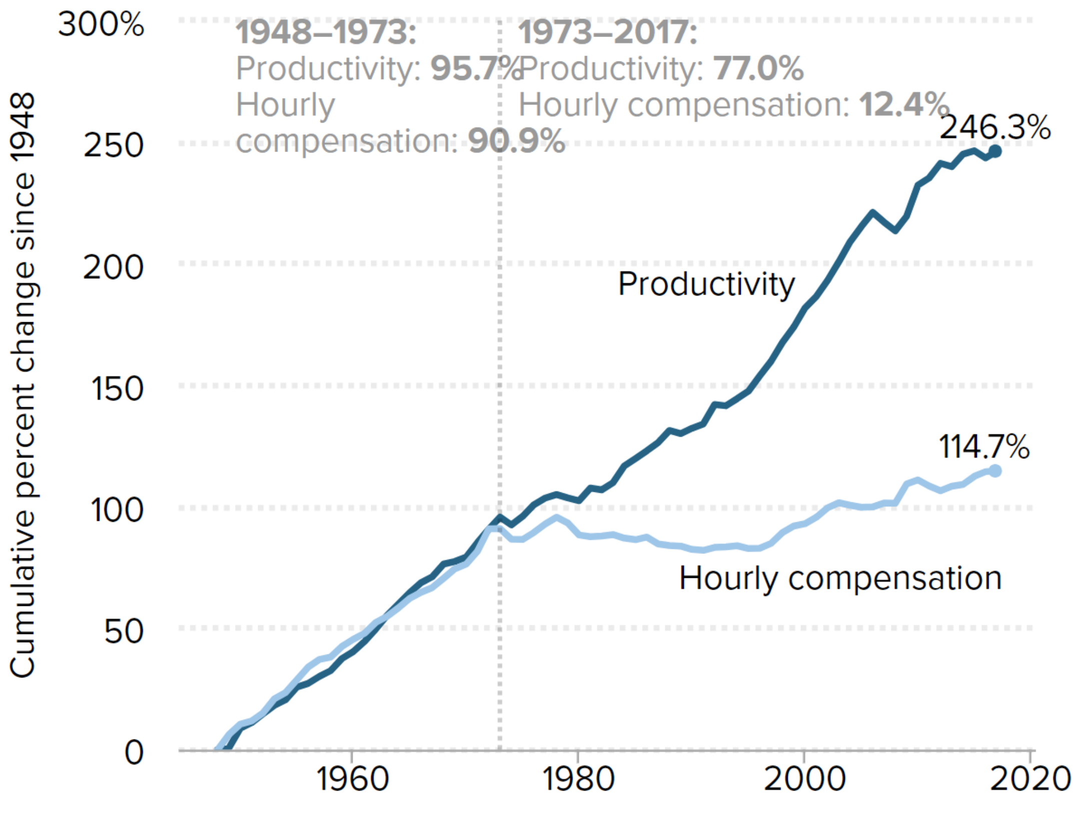
Ben Steacy | CS3012 | 20/11/2018

Introduction

Since the dawn of the information age we have strived towards innovation through computer microminiaturization. This has led to the increased automation of the manufacturing industry using robots and machines instead of skilled manual labour on production lines. Also, there has been a push to routinize many professions by reducing the number of tasks required to complete their work using technology to alleviate some of the manual inconvenience associated with their respective workflows. These steps have striven towards a goal of extracting productivity which according to the Economic Policy Institute has increased 77% in the private sector since 1973.

The gap between productivity and a typical worker's compensation has increased dramatically since 1973

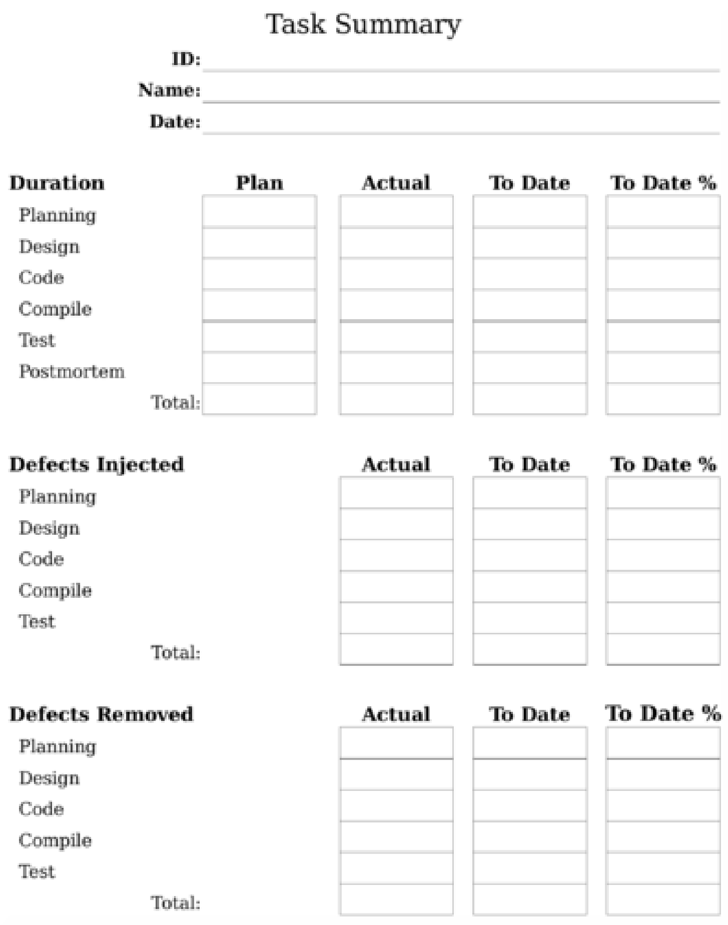
Productivity growth and hourly compensation growth, 1948–2017



But interestingly, the hourly compensation has stagnated since that year. So, there is an obvious financial incentive for companies and corporations to hire only the brightest and most productive for the job. But how do we measure this productivity in something as abstract as software development? There has been an industry wide obsession to measure individual's contributions towards a project or product and despite the research done in this area, there is still no consensus to the assessment of the software engineering process. How much should we attribute to cold hard statistics? What are the effects and ethical implications of collecting this data? And ultimately, can we measure what a considerable group of the software engineering community considers unmeasurable? In this report I will explore the current possibilities of these measurement tools and where I think we need to harness our efforts in this realm in the future.

Software Analytics Approaches

PERSONAL SOFTWARE PROCESS



Task Summary

ID:	
Name:	
Date:	

Duration	Plan	Actual	To Date	To Date %
Planning				
Design				
Code				
Compile				
Test				
Postmortem				
Total:				

Defects Injected	Actual	To Date	To Date %
Planning			
Design			
Code			
Compile			
Test			
Total:			

Defects Removed	Actual	To Date	To Date %
Planning			
Design			
Code			
Compile			
Test			
Total:			

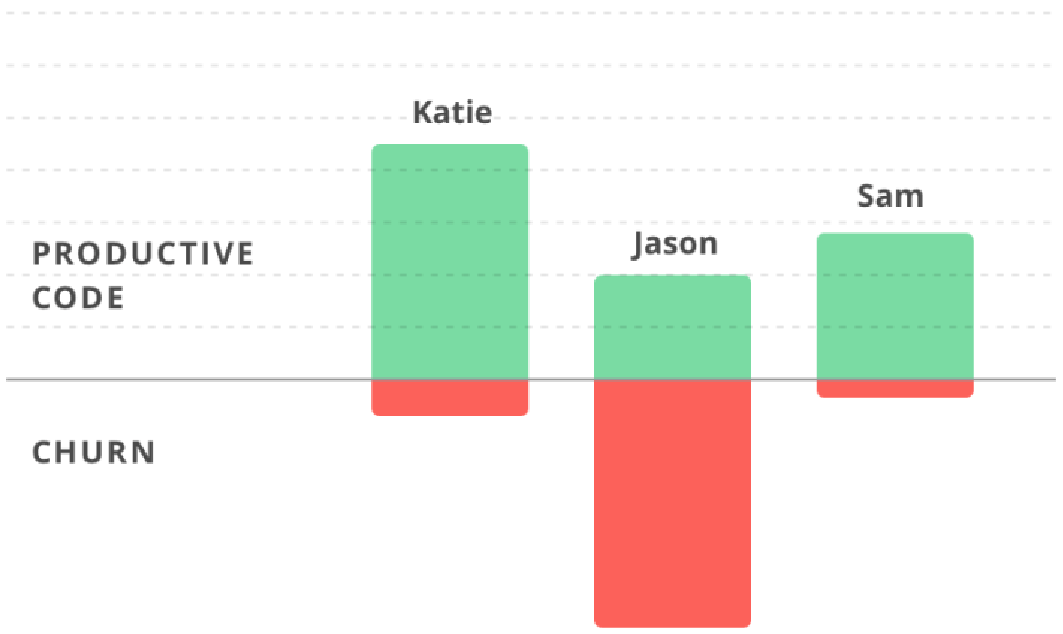
Some past software analytics have been handled using manual forms such the Personal Software Process which was published by Watts Humphrey in 1989. This method requires the developer to manually log a series of twelve forms to give a comprehensive view of the plan for their role in project and the breakthroughs and setbacks along the way. This in collaboration with the Team Software Process would provide a framework of a database like structure where computations such as productivity could be made using proxy-based estimating programs. It's focus leans towards helping the developer to understand how to improve their personal process by giving a framework to input and show their current performance. After a significant amount of data has been gathered a developer can make estimations of the scope and size of new projects and design strategies based on past work. The developers can also see what areas they can improve on themselves via postmortem analysis where comparisons can be made between projections and the actual implementation.

The method can feel cumbersome by today's standards due the manual overhead of inputting values and there is a lot of scope for human error. But it's manual nature also holds benefits in terms of flexibility. The documentation actively encourages personalization to the users' needs and project in hand. But the main issue with this format is that the implementation is dependent on the competency of the developer with the toolset. This system relies heavily on the developer being responsible in logging the data each day, to be honest in his assessments of his own work and have the knowledge and motivation to use these methods to further improve himself and the quality of his work. There is a clean tradeoff of flexibility at the expense of automation and a guarantee of accurate data. This could lead to problems when attempting to handle a system like this for hundreds or thousands of employees.

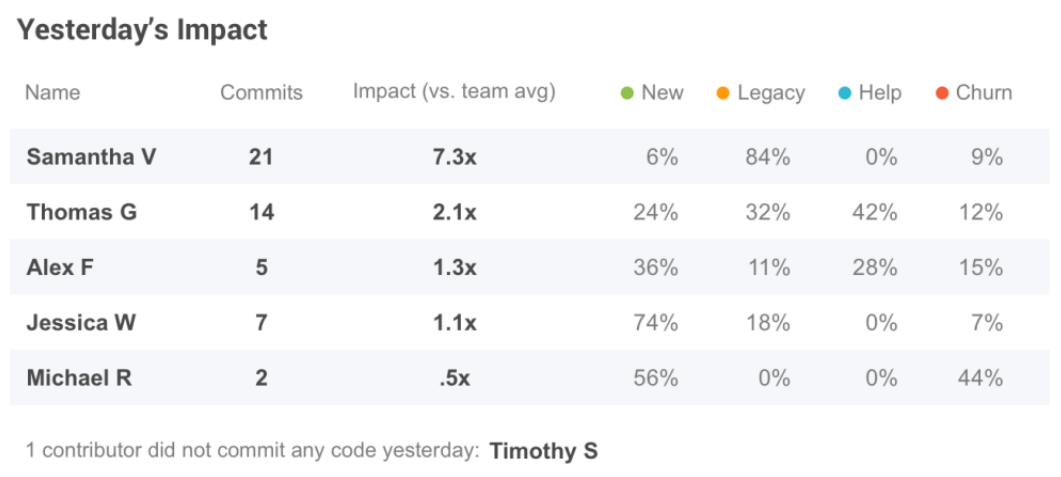
MODERN METRIC BASED ANALYTICS

More recent attempts of software analytics have moved towards the approach of automatically collecting data relating the individual's contribution to the project in the background. This collected data can then be used to represent various metrics. This automated approach greatly reduces the overhead of the manual Personal Software Process and thus increases the efficacy of scaling the system to hundreds or thousands of employees albeit with less flexibility. Also, the current providers of these implementations such as SonarQube and GitPrime provide many options to display graphical representations of the data collected over time. It's obvious that simple metrics such as lines of code written or hours active on the machine don't make for compelling insight into the software engineering process on their own.

But taking further measurements such as number of commits, the amount of code being rewritten or changed (churn) and their levels of coverage of the problem in hand we start to see a picture developing. The issue is like analyzing a piece of modern art, this picture is up to interpretation.



These metrics are telling you something but what that is exactly can be ambiguous. But what they are useful for is pushing you towards making an informed estimation of possible problems in the software engineering process. Leading software analytics service provider GitPrime gives several interpretations of what a developer with high churn could mean for the project. This could be anything from having a particularly difficult problem that requires a lot of reworking to achieve a high-performance goal, a developer trying to feel his way through a series of unclear requirements or perhaps it is simply an issue of just being incompetent of completing the task given. These statistics in combination can point in a direction to assess a developer’s productivity but there is still an array of possibilities under the metric magnifying glass. The conclusion of these analytics still rests on the beholder’s judgement to deduce the information. Human error cannot be completely removed from the equation.

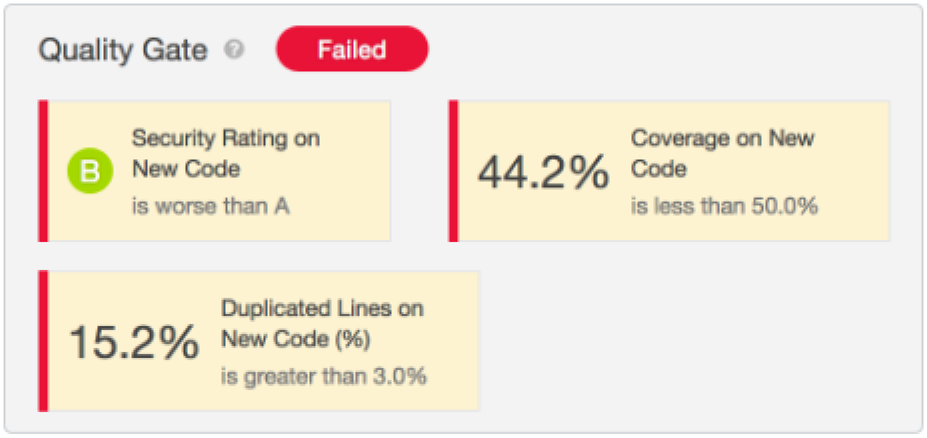


Yesterday's Impact

Name	Commits	Impact (vs. team avg)	● New	● Legacy	● Help	● Churn
Samantha V	**21**	**7.3x**	6%	84%	0%	9%
Thomas G	**14**	**2.1x**	24%	32%	42%	12%
Alex F	**5**	**1.3x**	36%	11%	28%	15%
Jessica W	**7**	**1.1x**	74%	18%	0%	7%
Michael R	**2**	**.5x**	56%	0%	0%	44%

1 contributor did not commit any code yesterday: **Timothy S**

But increasingly complex measures are being brought in by software analytics providers to represent more abstract concepts of productivity using more tangible terms. These efforts look to further shift the weight of deductive reasoning onto the analytics software. GitPrime's toolset claims to measure 'Impact', an algorithmic measure of "Roughly how much cognitive load did the engineer carry when implementing these changes". SonarQube's analytical software has an emphasis on 'Fixing Leaks' in software such as bugs and security weakness and has a 'Quality Gate' which the committed build can pass or fail depending on a series of these factors.



Quality Gate Failed

B Security Rating on New Code is worse than A

44.2% Coverage on New Code is less than 50.0%

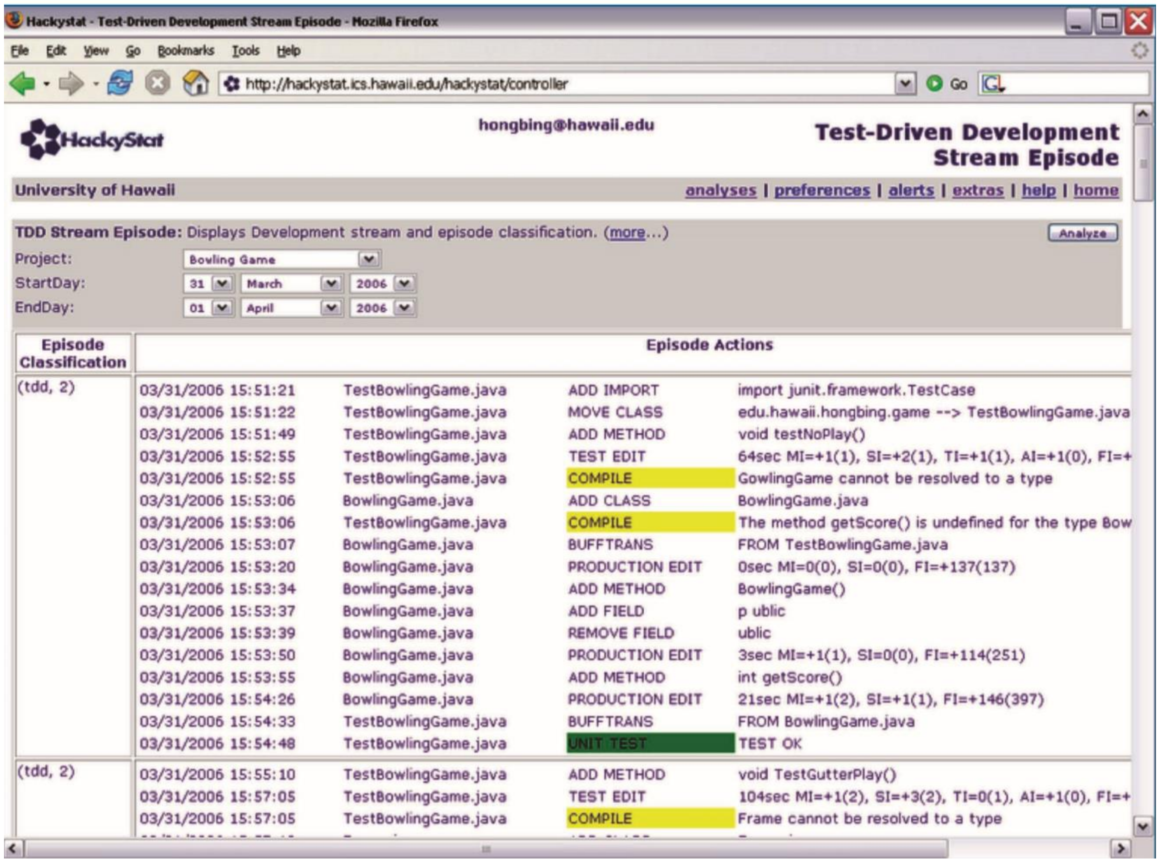
15.2% Duplicated Lines on New Code (%) is greater than 3.0%

Microsoft now provides the framework to analyze your own data and write your own specialized metrics and visualizations via their open source Azure project. These are all valid approaches of abstracting more understanding of the software engineering process, but it shows that metrics-based analytics providers are still exploring different possibilities with different emphasis instead of moving towards a consensus. There may not be a definitive silver bullet method that all providers are moving towards but highlights that there is plenty of potential pathways for innovation in the future.

Ethical Concerns

While I have talked mainly of the positive aspects of software analytics there are some ethical dilemmas that come when collecting such vast quantities of data. A software development organization wishes to use metrics to both motivate a team to strive for better results or to mine information for insight for research or future planning. But if measurement of performance is mishandled this can lead to perversion of the performance results you wish to seek.

An organization can be considered as a network of contracts that are set to complete mutually beneficial goals. This organization wants to exert more performance information from their software development team and implements a system to automatically monitor their every action inputted into their development computer. While this measure could have had the best intentions of collecting solely for informational insight, as the team becomes aware of the system a behavioral change occurs. Robert D. Austin in his paper Measuring and Managing Performance in Organizations states that “Dysfunction occurs when the validity of information delivered by a system of measurement is compromised by the unintended reactions of those being measured.” The Hackystat system that used was used in the University of Hawaii is an example of a negative reaction from developers being monitored in such fine detail. They felt uncomfortable with management having access and while these measures could improve productivity with some individuals, there is no question this system made developers act differently with the knowledge of being under the shadow of analytical surveillance. So, should you forgo transparency in pursuit of more accurate results in type of system? Is it worth the emotional implications for the individuals being monitored?


Hackystat - Test-Driven Development Stream Episode - Mozilla Firefox

http://hackystat.ics.hawaii.edu/hackystat/controller

hongbing@hawaii.edu

Test-Driven Development Stream Episode

University of Hawaii

analyses | preferences | alerts | extras | help | home

TDD Stream Episode: Displays Development stream and episode classification. (more...)

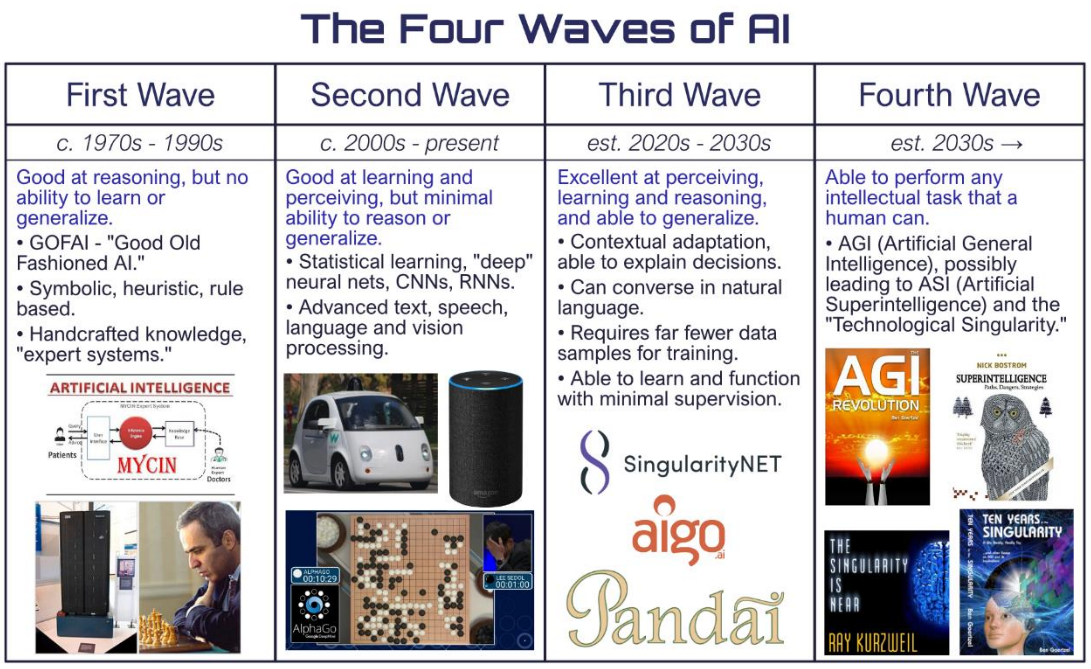
Project: Bowling Game
StartDay: 31 March 2006
EndDay: 01 April 2006

Episode Classification	Episode Actions			
(tdd, 2)	03/31/2006 15:51:21	TestBowlingGame.java	ADD IMPORT	import junit.framework.TestCase
	03/31/2006 15:51:22	TestBowlingGame.java	MOVE CLASS	edu.hawaii.hongbing.game --> TestBowlingGame.java
	03/31/2006 15:51:49	TestBowlingGame.java	ADD METHOD	void testNoPlay()
	03/31/2006 15:52:55	TestBowlingGame.java	TEST EDIT	64sec MI=+1(1), SI=+2(1), TI=+1(1), AI=+1(0), FI=+
	03/31/2006 15:52:55	TestBowlingGame.java	COMPILE	GowlingGame cannot be resolved to a type
	03/31/2006 15:53:06	BowlingGame.java	ADD CLASS	BowlingGame.java
	03/31/2006 15:53:06	TestBowlingGame.java	COMPILE	The method getScore() is undefined for the type Bow
	03/31/2006 15:53:07	BowlingGame.java	BUFFTRANS	FROM TestBowlingGame.java
	03/31/2006 15:53:20	BowlingGame.java	PRODUCTION EDIT	0sec MI=0(0), SI=0(0), FI=+137(137)
	03/31/2006 15:53:34	BowlingGame.java	ADD METHOD	BowlingGame()
	03/31/2006 15:53:37	BowlingGame.java	ADD FIELD	p ublic
	03/31/2006 15:53:39	BowlingGame.java	REMOVE FIELD	ublic
	03/31/2006 15:53:50	BowlingGame.java	PRODUCTION EDIT	3sec MI=+1(1), SI=0(0), FI=+114(251)
	03/31/2006 15:53:55	BowlingGame.java	ADD METHOD	int getScore()
	03/31/2006 15:54:26	BowlingGame.java	PRODUCTION EDIT	21sec MI=+1(2), SI=+1(1), FI=+146(397)
	03/31/2006 15:54:33	TestBowlingGame.java	BUFFTRANS	FROM BowlingGame.java
	03/31/2006 15:54:48	TestBowlingGame.java	UNIT TEST	TEST OK
(tdd, 2)	03/31/2006 15:55:10	TestBowlingGame.java	ADD METHOD	void TestGutterPlay()
	03/31/2006 15:57:05	TestBowlingGame.java	TEST EDIT	104sec MI=+1(2), SI=+3(2), TI=0(1), AI=+1(0), FI=+
	03/31/2006 15:57:05	TestBowlingGame.java	COMPILE	Frame cannot be resolved to a type

Much of this depends on the law, followed by the moral and ethical standards of the company. If an employee wishes to leave a company, will their analytics be deleted, kept or sold? We are only now seeing the effects big data is having on the world in main stream media and measures to regulate it such the General Data Protection Regulation in EU law. This offers some protection by providing transparency for employees. Although, even if the measures are laid in front of an employee they most likely still have consent to receive the job regardless of their opinion. Under this ruling an employee now has the right to know the purposes of processing data, if this information is being to other parties and the existence of any automated decisions being made. This holds the individuals in charge accountable to use information responsibly and for the good of everyone involved.

The Future and Conclusion

As I mentioned earlier, complex automated metrics point us to making certain conclusions but more often than not lack a definitive or in some cases an accurate answer to what it attempts to measure. Campbell's law of corruption states "The more any quantitative social indicator is used for social decision-making, the more subject it will be to corruption pressures and the more apt it will be to distort and corrupt the social processes it is intended to monitor." This is one of the biggest roadblocks for this current generation of software engineering analytics. Instead of pursuing creating the best software possible will we instead just pursue creating the most positive metrics? Is the software engineering process actually improving or are we making metrics to fit a narrative? Can we ever ensure 100% accurate analytics? It's hard to say. After assessing several analytics techniques Philip M. Johnson's paper 'Searching under the Streetlight for Useful Software Analytics' concludes that "The field isn't converging on a single best approach, nor are the latest approaches intrinsically better than earlier ones. Rather, the community has been exploring the space of tradeoffs among expressiveness, simplicity, and social acceptability." There's nothing perfect about each implementation, just different tradeoffs and problems. But does this spell the doomed finality of software analytics? Far from it.



The Four Waves of AI

First Wave	Second Wave	Third Wave	Fourth Wave
c. 1970s - 1990s	c. 2000s - present	est. 2020s - 2030s	est. 2030s →
Good at reasoning, but no ability to learn or generalize. • GOFAI - "Good Old Fashioned AI." • Symbolic, heuristic, rule based. • Handcrafted knowledge, "expert systems."	Good at learning and perceiving, but minimal ability to reason or generalize. • Statistical learning, "deep" neural nets, CNNs, RNNs. • Advanced text, speech, language and vision processing.	Excellent at perceiving, learning and reasoning, and able to generalize. • Contextual adaptation, able to explain decisions. • Can converse in natural language. • Requires far fewer data samples for training. • Able to learn and function with minimal supervision.	Able to perform any intellectual task that a human can. • AGI (Artificial General Intelligence), possibly leading to ASI (Artificial Superintelligence) and the "Technological Singularity."

Six Kin Development (adapted from DARPA's "Three Waves of AI")

With the next wave of artificial intelligence, we are moving future away from the AI fallacy that the only way to develop systems that perform tasks at the level of experts is to replicate the thinking processes of human specialists. It is not impossible to think that one day a computer might able to factor in the unknowable qualities using our current methods. To factor in qualities of empathy to behaviour and thus have human like judgement along with analyzing the cold hard statistics. The social implications of a machine making calls like this is an ethical question in its own right but I believe hybrid approaches like this provide the missing context that I feel evades current systems. More research in the same vein of the how networks of people affect productivity or even the research into measuring happiness through wearable tech could provide further insight in the software engineering process. But until the point where that technology becomes a viable solution I feel it is important not to view current software analytics as a replacement to managing a team but as a useful tool to provide direction to decision making in management.

BIBLOGRAPHY

https://www.epi.org/productivity-pay-gap/

https://resources.sei.cmu.edu/asset_files/SpecialReport/2009_003_001_15029.pdf

https://www.gitprime.com/downloads/resources/GitPrime-Data-Driven-Management.pdf

https://dev9.com/blog-posts/2015/1/the-myth-of-developer-productivity

Searching under the Streetlight for Useful Software Analytics - Philip M. Johnson, University of Hawaii at Manoa

https://blog.gitprime.com/6-causes-of-code-churn-and-what-you-should-do-about-them/

https://blog.gitprime.com/impact-a-better-way-to-measure-codebase-change/

https://www.sonarqube.org/features/clean-code/

https://docs.microsoft.com/en-us/azure/log-analytics/log-analytics-tutorial-viewdata?toc=/azure/azure-monitor/toc.json

http://ptgmedia.pearsoncmg.com/images/9780133492071/samplepages/0133492079.pdf

https://www.siliconrepublic.com/advice/gdpr-employee-effects-data-deloitte

https://www.youtube.com/watch?v=Dp5_1QPLps0

https://medium.com/@scott_jones/third-wave-ai-the-coming-revolution-in-artificial-intelligence-1ffd4784b79e

Measuring Happiness Using Wearable Technology —Technology for Boosting Productivity in Knowledge Work and Service Businesses

Evolution of the networked enterprise: Mckinsey Global Survey results