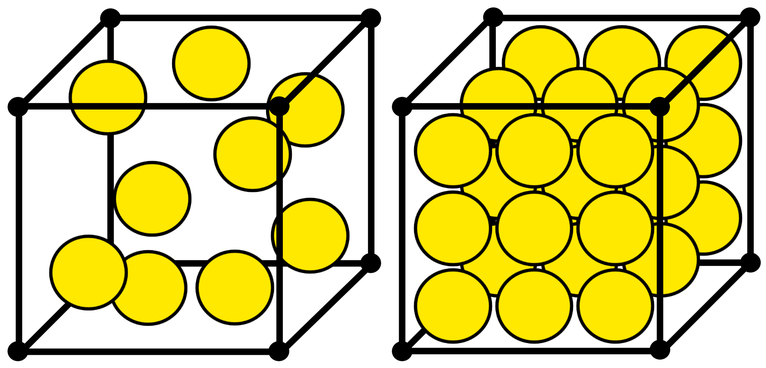
A changeset-based approach to assess source code density and developer efficacy

A step towards data-driven software cost- and effort estimation models

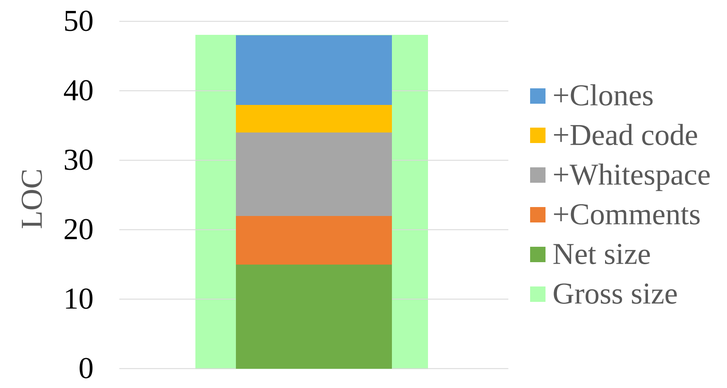
Sebastian Hönel, *Morgan Ericsson, Welf Löwe, Anna Wingkvist*
Data-driven Software and Information Quality Group – Centre for Data Intensive Sciences and Applications
Linnaeus University, 351 95 Växjö, Sweden
{sebastian.honel, morgan.ericsson, welf.lowe, anna.wingkvist}@lnu.se

Context



```
- private Semaphore startLock;
+ private Semaphore lockStart;

- private Semaphore shutdownLock;
+ private Semaphore lockShutdown;

-       this.startLock = new Semaphore(1, 1);
-       this.shutdownLock = new Semaphore(0, 1);
+       this.lockStart = new Semaphore(0, 5);
+       this.lockShutdown = new Semaphore(0, 2);

-           DesiredLatency = 80
+           DesiredLatency = 50

-           var localEp = new IPEndPoint(IPAddress.Any, this.multicastPort);
+           using (var ms = new MemoryStream())
+           {
+               var localEp = new IPEndPoint(IPAddress.Any, this.multicastPo

-           this.udpClient.Client.SetSocketOption(SocketOptionLevel.Socket,
-           this.udpClient.Client.Bind(localEp);
-           this.udpClient.JoinMulticastGroup(this.multicastAddress);
+               this.udpClient.Client.SetSocketOption(SocketOptionLevel.Sock
+               this.udpClient.Client.Bind(localEp);
+               this.udpClient.JoinMulticastGroup(this.multicastAddress);

-           var rsws = new BufferedWaveProvider(new WaveFormat())
-           {
-               BufferDuration = TimeSpan.FromMilliseconds(500)
-           };
+               var rsws = new BufferedWaveProvider(new WaveFormat())
+               {
+                   BufferDuration = TimeSpan.FromMilliseconds(500)
+               };
+           }

-           this.shutdownLock.WaitOne();
+           this.lockShutdown.WaitOne();
```

```
-   /// <summary>
-   /// Initialize the client with the specific address, port and format.
-   /// </summary>
+   /// Initialize the client with the specific address, port and format.
+   /// </summary>
+   /// <param name="multicastAddress"></param>
+   /// <param name="multicastPort"></param>                          lcastPor
+   /// <param name="waveFormat"></param>
+   public UdpWaveStreamPlayer(IPAddress multicastAddress, int multicastPor
+   {
+       this.multicastAddress = multicastAddress;
+       this.multicastPort = multicastPort;
+       this.waveFormat = waveFormat;
+
+       this.startLock = new Semaphore(1, 1);
+       this.shutdownLock = new Semaphore(0, 1);
+       this.output = new WaveOut()
+       {
+           DesiredLatency = 80
+       };
+
+       this.udpClient = new UdpClient
+       {
+           ExclusiveAddressUse = false
+       };
+
+       this.Volume = 100;
+   }
+
+   /// <summary>
```

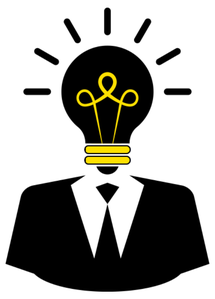
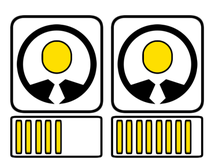
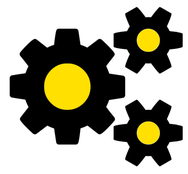


Typical segmentation of lines of code (LOC) between two versions of the same file

$D = \frac{F}{S}$ The Density D is the ratio between the Functionality F and the Size S. A density of 1 means that all code contributes to the functionality. Therefore, the notions of D_{max} and D_{min} relate to the net- and gross-size of the software (S_n, S_g).

$E = \frac{D * S}{t}$ The effort, when measured using functionality instead of size, is more meaningful. The equation $E = {}^{F}/_{t}$ is therefore equivalent.

$P = \frac{E}{S}$ The Productivity P for software is defined as *Effort* per *Size*, according to ISBSG[1]. We can hence deduce a direct correlation between productivity and density using the equation $P = {}^{D}/_{t}$.

	Money	Developer Skill	Time	Productivity
				

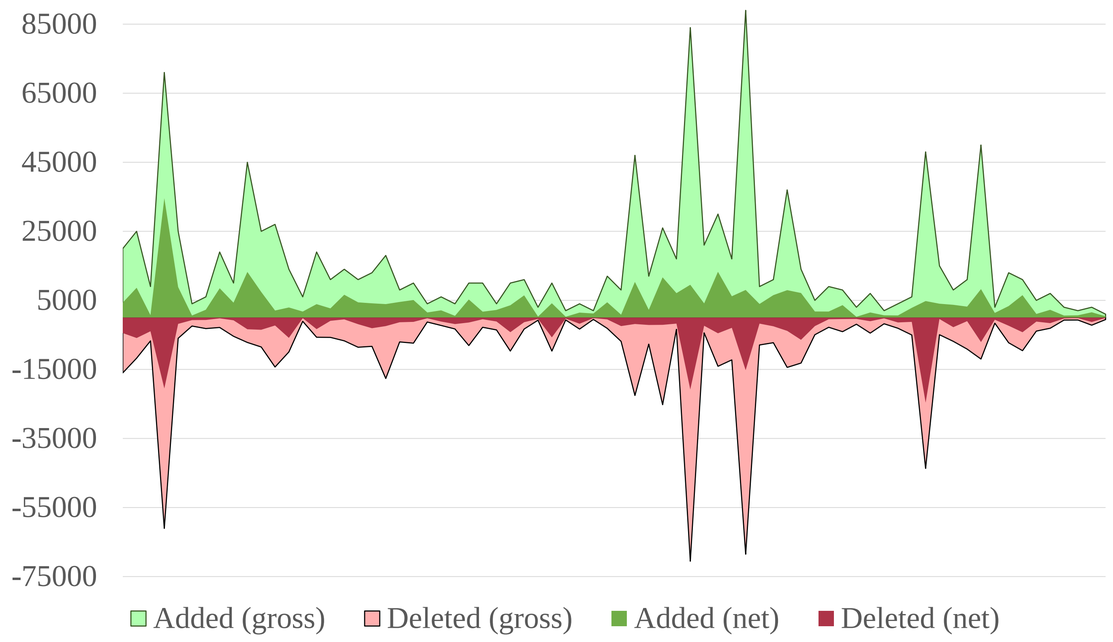
Commits can be associated with one developer. This is important for effort- and cost-estimation models, so that the planner can assess the productivity of their mixed-skill team more accurate. When analyzed as a time series, the density can be forecast, allowing the planner to adjust the (ongoing) project's estimates and to prevent fallouts and plan resources accordingly.

Public repository hosting services allow us to freely access and analyze millions of commits across a variety of programming languages, project setups and developer capabilities. We are fetching a great number of changesets and try to determine their densities. We expect to unearth correlations between the different aspects of those projects.

[1] ISBSG. "Software size as the main input parameter to cost estimation models." http://isbsg.org/software-size/

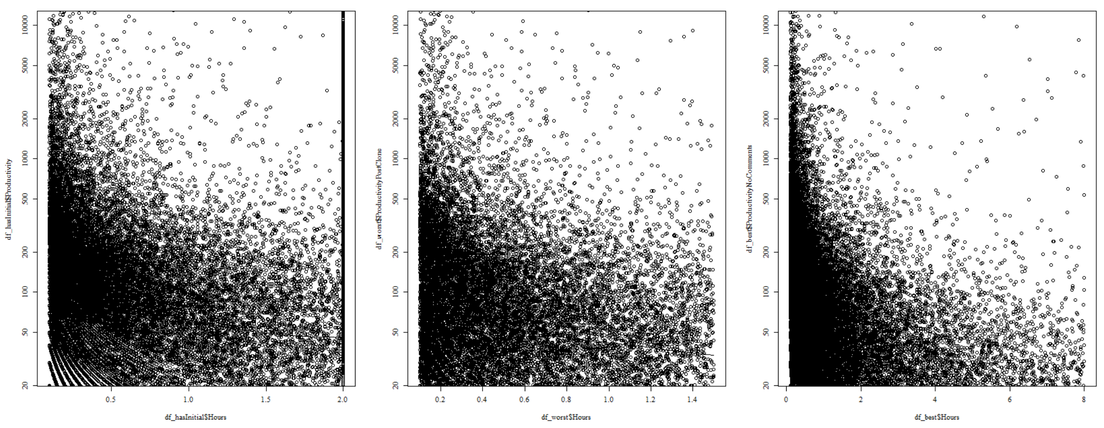
The gross size is determined by the plain count of lines of code. Removing unnecessary comments, space lines, dead and unreachable code as well as any cloned functionality yields the net size of the software.

We are focusing on examining how close the gross size of changes is compared to the net size using various methods, such as clone detection or string similarity measures to find the density of contained functionality. We want do examine which method is the most suitable for determining the density.

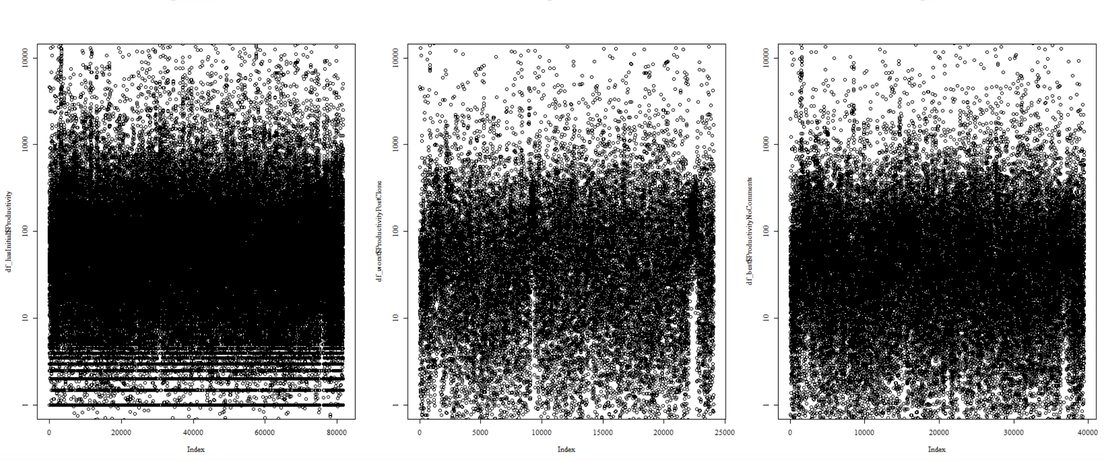


Evaluation

- We learned about the significance of density and how it impacts large and small changesets.
- We were able to identify the most and least suitable methods to determining density.
- We learned which methods can best approximate a certain notion of time using a certain notion of size.
- We have analyzed about 1,650 open-source software repositories with ~80k commits across various programming languages and project setups. Large deviations in gross- and net-size were discovered.

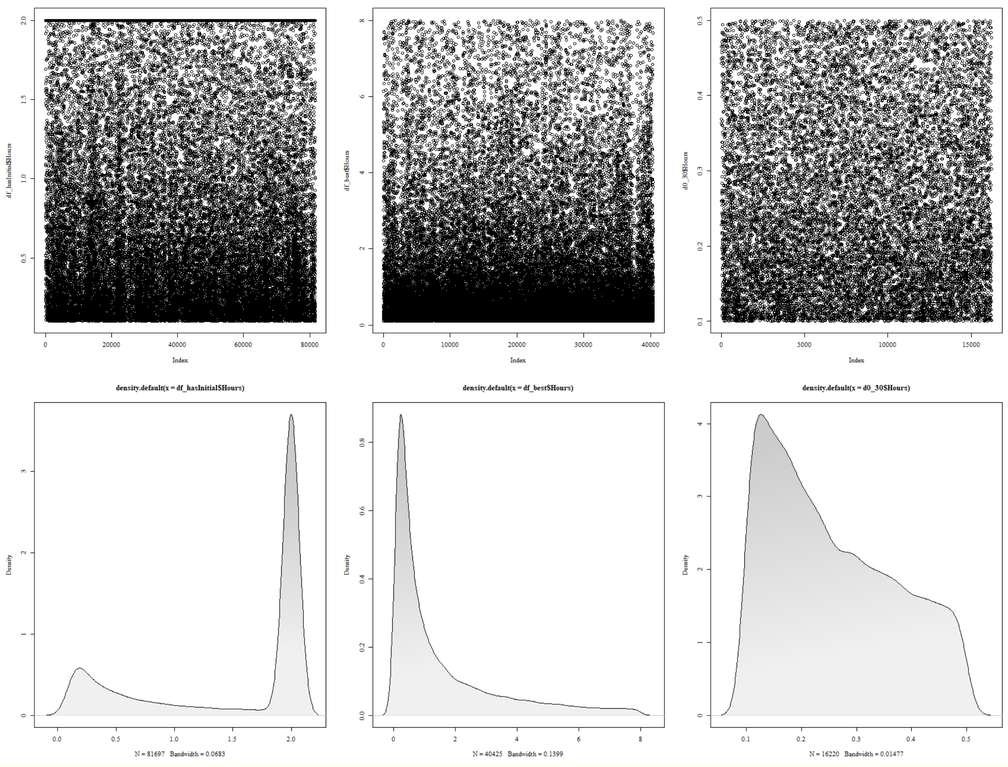


Three different notions productivity, based on gross-size and default settings for time logging with clear irregularities (left), then obtained for the worst-performing dataset (middle) and the best-performing dataset, with clearly improved density and less noise (right).



Three different notions of time, as obtained by git-hours using the default settings and no filtering (left), using the best-working settings of 480 minutes and filtering (middle) and the most normally-distributed notion of time (right) based on 30 minute-sessions and filtering.

- We can detect typical developer commit-behavior and how it negatively impacts time estimates and leads to distortions of measurements of productivity.
- We have unveiled shortcomings in automatically deducing spent time from commit timestamps using git-hours and found better-than-default settings by denoising the data.
- Currently, the distorted measurements of time do not allow to correlate any (specific) notion of size to it.



Linnæus University
Centre for Data Intensive Sciences and Applications
Data-driven Software and Information Quality Group
Lnu.se