Janiform Intra-Document Analytics for Reproducible Research

Jens Dittrich Patrick Bender

Saarland University
infosys.cs.uni-saarland.de
github.com/uds-datalab/PDBF

ABSTRACT

Peer-reviewed publication of research papers is a cornerstone of science. However, one of the many issues of our publication culture is that our publications only publish a summary of the final result of a long project. This means that we put well-polished graphs describing (some) of our experimental results into our publications. However, the algorithms, input datasets, benchmarks, raw result datasets, as well as scripts that were used to produce the graphs in the first place are rarely published and typically not available to other researchers. Often they are only available when personally asking the authors. In many cases, however, they are not available at all. This means from a long workflow that led to producing a graph for a research paper, we only publish the final result rather than the entire workflow. This is unfortunate and has been criticized in various scientific communities. In this demo we argue that one part of the problem is our dated view on what a "document" and hence "a publication" *is*, *should*, and *can be*. As a remedy, we introduce portable database files (PDbF). These files are janiform, i.e. they are at the same time a standard static pdf as well as a highly dynamic (offline) HTML-document. PDbFs allow you to access the raw data behind a graph, perform OLAP-style analysis, and reproduce your own graphs from the raw data — all of this *within* a portable document. We demo a tool allowing you to create PDbFs smoothly from within LaTeX. This tool allows you to preserve the workflow of raw measurement data to its final graphical output through all processing steps. Notice that this pdf already showcases our technology: rename this file to ".html" and see what happens (currently we support the desktop versions of Firefox, Chrome, and Safari). But please: do *not* try to rename this file to ".ova" and mount it in VirtualBox.

1. INTRODUCTION

Irreproducibility is a problem frequently lamented upon in various scientific communities [12, 15]. In the context of computer science it has recently been coined "The Real Software Crisis" [10]. The database community has identified it more than ten years ago and is attacking it through repeatability committees, e.g. [11]. These committees rerun the experiments of accepted papers using the datasets and code provided by the authors. Obviously, given the sheer size and complexity of some projects, in many cases this boils down to black box testing, i.e. it can neither be tested if the code actually implements the algorithms presented in the paper nor whether the code measures and reports results in a proper way. Another problem of repeatability committees is that they cannot remedy inherent publication bias: "reviewers don't like negative results". Hence, for an experimental evaluation you need the "right" queries, the "right" datasets and the "right" baselines. The results then need to be visualized, presented, and interpreted in the "right" way (e.g. logarithmic vs linear scale, offset on y-axis). This naturally leads to a flood of papers with seemingly positive results. And to papers where the "improvements don't add up" [3]. Publication bias was attacked by the inauguration of Experiments&Analysis papers at (P)VLDB. These kind of papers reevaluate existing work in a uniform setting and may also publish negative results. These experimental evaluations may then serve as landmarks in the flood of papers with (overly) positive results giving clear advice on the strengths and weaknesses of a particular method.

This small demo is neither the place to even summarize nor defend the different arguments in the debate on repeatability and our experimental culture. It is an emotional topic where the esteemed reader of these lines probably has a strong opinion in one way or the other. This is just fine. In the following, we will simply accept that there is a problem [15, 10][1]. And that this problem is calling for "a new model for the way how we publish our results" [12]. Handling this problem can be regarded as an instance of "small data" [7, 8][2].

Obviously, we cannot solve all of the world's problems with reproducible research like baseline, dataset, query, and presentation bias. In this demo we will attack the latter and show how to preserve the connectivity from raw data to graphical display in the research workflow. We believe that our demo is an important step towards making access and analysis of raw data easier and more transparent.

2. SIGNIFICANCE OF THE CONTRIBUTION

We believe that our contribution is significant for the following reasons:

(1.) Portable DataBase Files. We provide Portable DataBase Files (PDbF), a general model to combine a static pdf document with additional highly dynamic HTML-content. In order to specify

Proceedings of the VLDB Endowment, Vol. 8, No. 12
Copyright 2015 VLDB Endowment 2150-8097/15/08.

[1]Just read the "ten simple rules for reproducible results" [15] and then ask yourself how little of this we implement in our papers.

[2]Also notice an upcoming Dagstuhl perspectives workshop on Artifact Evaluation for Publications [6] where the first author participates.

a PDbF, we simply require access to a static document S, dynamic content D, and a PDbF-configuration file C defining where to place the dynamic content. An example of D could be an embedded relational database and an appropriate visualization of some of its content, e.g. a bar chart visualizing measurements collected in a table.

(2.) Alternative Dynamic Views on the Data. Our technology allows you to perform *offline* OLAP-style analytics on the *data shipped within the document*. All you need is a Web Browser (currently Firefox, Chrome or Safari on a desktop machine). We support a rich feature list. See Section 4 for our currently supported features (as of June 2015).

(3.) PDbF-Compiler. We provide a compiler, called *janic*, taking as its input the triple (S, D, C). Our compiler outputs a janiform document. That document is *at the same time* a valid pdf *and* a valid HTML-document. Thus, if you open the file with a pdf-viewer, you will see the static content, i.e. only the S-part. However, if you rename the file to ".html" and open it with a Web browser, you will be able to inspect the dynamic part, i.e. D and S.

(4.) Full LaTeX integration. We instrument LaTeX to output not only the static pdf-file S, but also the necessary data to create a valid PDbF-configuration file C. In addition, we provide an extension to LaTeX allowing users to create graphs directly from within LaTeX — without requiring the user to invoke multiple tools manually. In addition, this process creates dynamic variants of the graphs as a side-effect, i.e. the D-part. This means that the user simply defines the initial display of the graph (or table). Everything else is generated automatically. The result of this instrumentation is again a triple (S, D, C) which can be fed into our PDbF-Compiler (see Contribution 3) to create a janiform PDbF-document.

(5.) Preservation of Raw Data and Graph-Connectivity. Our compiler preserves the connection between raw data and the graphs or tables produced from that raw data *through the LaTeX compiler*. Thus, we are able to ship that part of the workflow, i.e. data, graphs, and the code producing the graphs within a single "document".

(6.) Natural Longterm Preservation of Raw Data. As our tools embed the raw data within the "pdf publication", PDbF-documents naturally archive the raw data with the document. Therefore, the raw data may be "downloaded" directly from within the PDbF-document. In addition, there is no need anymore to only publish a subset of the measurements (as graphs). Embedding all raw data does not require much space and allows other researchers to see the whole picture. And all of this works by simply shipping a single "pdf" to the publisher of the research. There is no need to provide an extra archive for data and/or code.

(7.) Impact On Research in General. We believe that our technology may not only be interesting to the database community. Our tool may be interesting for all research communities working with experimental data. Therefore, we open-sourced it: github.com/uds-datalab/PDBF.

3. THE PDBF FRAMEWORK

3.1 Janiform File Format

How is it possible to create a single file that may both be interpreted as a valid pdf document and a valid HTML document? The core idea is to create a document where complementary parts of the file are ignored by the different applications. The core structure of a PDbF is shown in Figure 1. Its core structure is inspired by "funky files" [2].

A **PDF reader** has the following *perspective* on this file: It reads the PDF marker (aka *magic number*) "%PDF-1.5". Then it locates

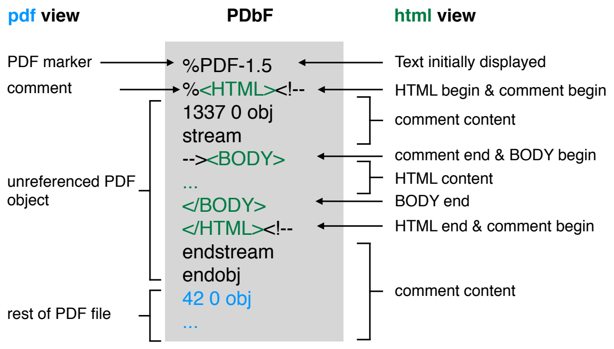


Figure 1: Two different views on the same janiform PDbF

the Xref table at the end of the file. That Xref contains a dictionary of all objects in this PDF file except for the unreferenced PDF object ("1337" until "endobj" in the example). As the unreferenced object is neither referenced by another object nor the xref table, it is never read by the PDF viewer.

An **HTML browser** has the following *perspective* on this file: It reads first lines. Hence, the text "%PDF-1.5\n%" is very briefly displayed at this point. After that text, an HTML comment begins which is only closed right before the BODY begin tag (we leave away HEAD for simplicity here). After BODY end there is an HTML end followed by an HTML comment begin. As this comment is never closed, everything afterwards will be ignored. The html body displays all its content on an additional layer hiding the initially displayed text "%PDF-1.5\n%".

3.2 janic Compiler Architecture

Our PDbF-compiler coined *janic* is written in Java. The compilation process can be subdivided into multiple steps:

(1.) janic invokes the standard LaTeX Compiler.
(2.) The standard LaTeX compiler outputs the PDbF-configuration file as a side-effect (we achieve this through instrumenting a PDBF.sty file which is referenced by the LaTex document). In addition, LaTeX creates a draft version of the PDF document.
(3.) janic reads that PDbF-configuration file and calculate the relative positions of the dynamic content on each page. We do this in java as it was too slow in LaTeX.
(4.) janic reads all tables from the database. (The database is specified as an inlined SQL-dump or a JDBC-connection in the PDbF-configuration file.)
(5.) janic invokes PhantomJS to create static snapshots of all graphs (as specified in the LaTeX document) automatically. This is required for the pdf view.
(6.) PhantomJS creates a static image for every dynamic object present in the PDbF-configuration file.
(7.) janic invokes the LaTeX compiler a second time to obtain the final placements of all static snapshots.
(8.) The LaTeX compiler outputs the final PDF document with images from Step 6.
(9.) janic reads the final PDF document, the PDbF interpreter (see Section 3.3), and the finalized PDbF-configuration. janic then outputs the resulting janiform PDbF.

Obviously, the entire process adds a few seconds to the standard LaTeX compilation time, however the overhead is not substantial and only required once for the final deployment of the PDbF. For instance, on an Intel Core i5-3230M CPU@2.60 GHz (2 cores) we require 21 seconds total compilation time for this submission. The total file size of the PDbF is roughly 2.7 MB for the embedded tools plus twice the size of the static pdf plus the size of the database. For instance, the file size of this demo submission is only 10.2 MB in total (5 MB of which is the database containing 30,000 tuples).

3.3 PDbF Interpreter

In order to display the static pdf as well as the HTML-overlay in a Web browser we extended PDF.js [13]. We store the static pdf, the database, and the config file as base64-encoded javascript strings inside the HTML. When the PDbF is viewed as an HTML-file, our PDbF interpreter first decodes this data. Then, the queries of all visualizations are executed by alasql [1], an in-memory javascript SQL-engine. After that, all visualizations are rendered and placed on top of their static image counterparts.

4. PDBF FEATURES

In this section we discuss the features currently supported by our framework.

Support for DESKTOP Browsers and PDF Viewers. We have tested our PDbF files with the following browsers: Chrome 41, Opera 28, Firefox 36, and Safari 8. We tested the following PDF-Viewers: Adobe Reader XI, PDF-XChange Viewer, QuickPDF (Android), Google Drive-PDF-Viewer (Android).

Dynamic Graphs. In the *PDF view*, a static snapshot of each graph will be displayed showing the initial configuration of the graph. In the *HTML view*, click the ⛶ symbol in the upper left corner of a graph. A window will appear with additional options. You may change the display in many ways, e.g. from linear scale to logarithmic scale, adjust display ranges, and many other options. Data for dynamic content can be specified in three ways: inline in LaTeX, using an SQL-dump or via JDBC connector that imports arbitrary SQL results. We support several types of graphs including multi-column bar plots (Figure 2) and line plots (Figure 3). In addition (not displayed due to space constraints), we support stacked bar plots, grouped stacked bar plots, multiplots, add/remove filtering functions, change queries generating a graph, outlier computation, and confidence intervals. You may specify a graph directly in LaTeX. For instance, in order to specify a line chart (Figure 3 of this paper), you simply write:

```
\documentclass{vldb}
\usepackage{PDBF}
...
\dbSQLFile{somedb.sql}
\chart[width=\textwidth, height=0.75\textwidth,
      xunit=Row Number, yunit={Runtime [in sec]}]{
   SELECT ROWNUM() as rownum, runtime
   FROM fact_experiment LIMIT 3000;
}
```

In this example, the data is contained in a SQL-dump "somedb.sql". It is also possible to connect to a database, e.g. using a `\dbSQLJDBC{}` command. In either case, all base tables will be replicated into the final PDbF.

The dynamic graphs are built using c3 [4], a javascript chart library. Arbitrary options may be specified in LaTeX that are then passed to c3.

Dynamic Pivot Tables. We also support dynamic pivot tables. See Figure 4 for an example. This visualization is based on HTML pivot tables [14] and jQuery [9]. You may group on arbitrary keys and display grouping keys in rows and columns. This will in turn change the underlying database query on the fly.

Raw Data Access. Raw data may be "downloaded" directly from the dynamic visualizations as CSV. This may also be extended to "download" source code and/or binaries from within the PDbF.

Easy to use. We believe that the key to wide acceptance of a tool is its ease of use and little overhead. In order to inspect a PDbF no additional software is required for the end-user. And if a user only wants to see the static pdf, he does not need to rename the file to html. In terms of creating a PDbF, our tool integrates well with LaTeX and only adds minor compilation overhead.

There are also some **Limitations.** For instance, a PDbF must not be post-processed with a pdf-tool, as this may remove and/or disable the dynamic features of the PDbF. Another issue is performance. In the general case, we expect a database with experimental data to be small (in the sense of less than 100,000 tuples). Query processing times on such data is not feelable, e.g. filtering and aggregating a column of 100,000 integer values takes about 0.00015 seconds in C++ (2.8 GHz Intel Core i7, no SIMD, no multi-threading). However, in the browser these kind of operations may be about 10,000 times slower. Is this all due to javascript and text parsing? And how can that be fixed? We want to investigate these scalability aspects in a full paper, see Section 6.

5. THE DEMO

A major part of the demo (PDbF file format) is already in your hands, the other parts (PDbF Compiler and LaTeX integration) will be shown at VLDB.

The PDbF File Format. This is contributions 1, 2, 5, and 6. We invite the reader of this pdf (and also the audience at VLDB) to change the suffix of this file from ".pdf" to ".html" (on your desktop computer). Your desktop browser will open and you will be able to see the dynamic features explained in Section 4. Notice that the dynamic content is completely offline and runs entirely in your Web browser's sandbox (Firefox, Chrome, and Safari).

The PDbF Compiler. This is contribution 3. We will invite the audience to create PDbF-files themselves on arbitrary input documents. These documents can then be enriched with dynamic content using our janic compiler. The final result may be viewed with a desktop Web browser.

LaTeX Integration. This is contribution 4, see also Section 3.2. We invite readers to bring their own LaTeX files. We will show them how to prepare their tex-documents in order to be able to define graphs directly on the raw data. We will demonstrate the seamless integration of our technology into LaTeX.

6. CONCLUSIONS AND FUTURE WORK

Summary. This demo opened the book for portable database files (PDbF). PDbFs allow you to embed raw data into a document while preserving the entire data-pipeline from raw data to graphs. This allows readers to perform in-document OLAP-style analytics on the published document. PDbFs are janiform documents that are at the same time a valid pdf *and* a valid HTML document. Thus they can be viewed in either way. This pdf is an example of such a janiform PDbF. In addition, the conference demo show-cases our compiler janic allowing you to seamlessly create PDbFs from within LaTeX.

Future Work. We only started full time development on this project in February 2015. Hence, as of June 2015 there are many things left to do. One idea for future work is to explore the synergies between VisTrails [16] and PDbF. The former tool provides provenance and workflows, however it still relies on external files, applications, and/or web services to display dynamic content: they may, however, eventually become offline. This problem is solved by PDbF which provides a single downward compatible janiform file that is archived just like an old-fashioned pdf with the publisher. In addition, it can be viewed in multiple ways using standard tools like a pdf viewer or a browser. We also want to explore the scalability aspects of PDbF in a full paper. In addition, we would like to integrate additional aspects of the research pipeline into PDbF.

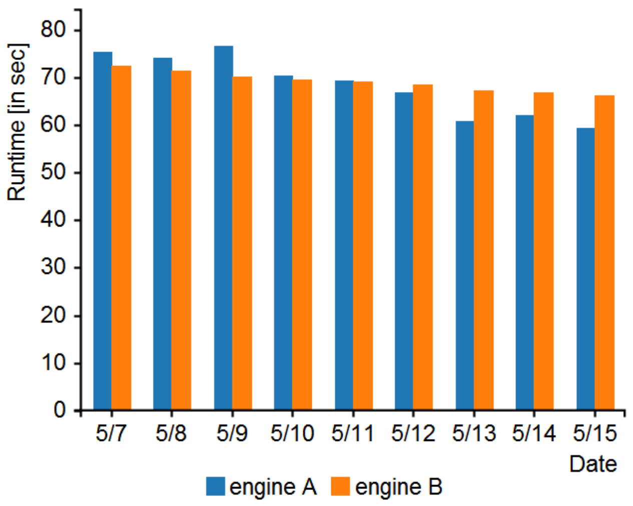


Figure 2: Multi-column Bar Chart

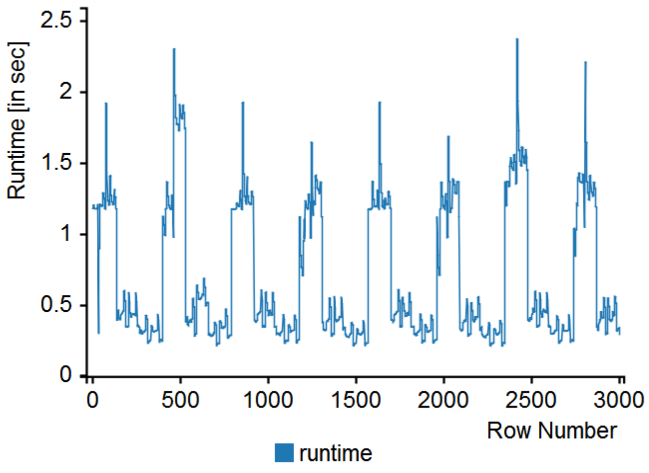


Figure 3: Line chart

For instance, in future, one might consider to embed a virtual machine image including an operating system image, and all software that is required to run the original code *within the PDbF* (this may be a matter of a few megabytes for small operating systems up to ~3GB of data for bigger ones). This may sound infeasible in 2015. But 4K video streaming over the Internet sounded equally silly in 1995. In order to create such virtual image, we want to leverage ReproZip [5]. As an outlook to this feature try renaming this file to ".ova" and open it with a VirtualBox emulator. Enjoy.

Acknowledgments. We would like to thank the anonymous reviewers for their helpful comments.

7. REFERENCES

[1] https://github.com/agershun/alasql.

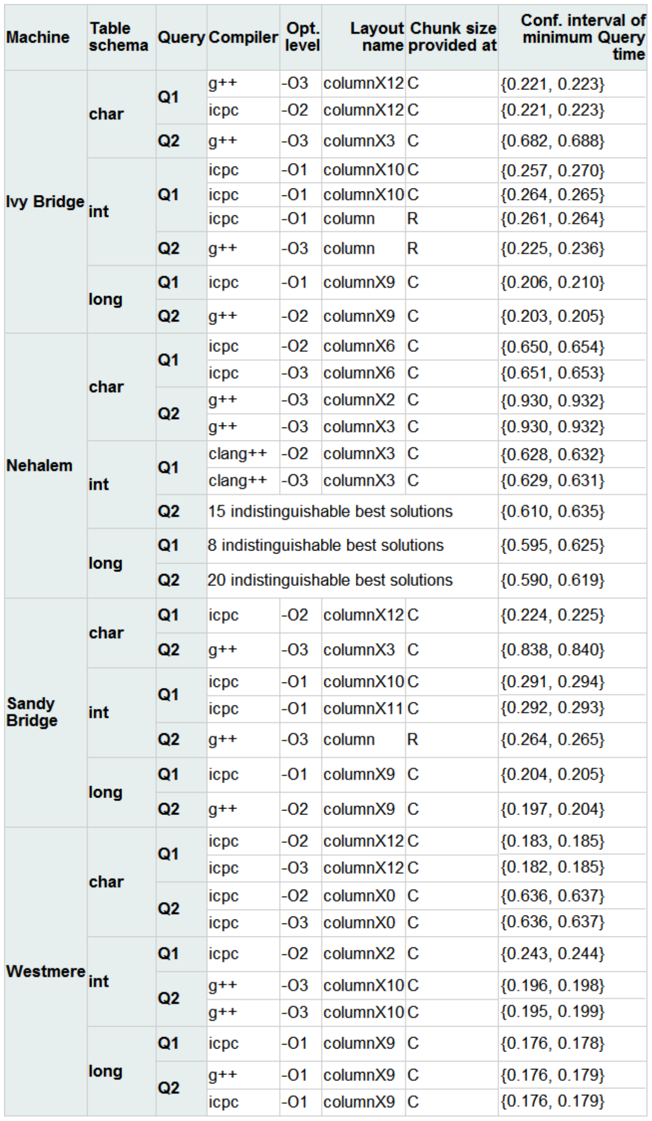
[2] A. Albertini. Funky File Formats. In *CCC*. 2014.

[3] T. G. Armstrong, A. Moffat, W. Webber, and J. Zobel. Improvements That Don't Add Up: Ad-hoc Retrieval Results Since 1998. CIKM, pages 601–610, 2009.

[4] http://c3js.org.

[5] F. Chirigati, D. Shasha, and J. Freire. Packing Experiments for Sharing and Publication. In *SIGMOD*, pages 977–980, 2013.

[6] Dagstuhl Perspectives Workshop 15452: Artifact Evaluation for Publications. 11/2015. http://www.dagstuhl.de/de/programm/kalender/semhp/?semnr=15452.



Machine	Table schema	Query	Compiler	Opt. level	Layout name	Chunk size provided at	Conf. interval of minimum Query time
Ivy Bridge	char	Q1	g++	-O3	columnX12	C	{0.221, 0.223}
			icpc	-O2	columnX12	C	{0.221, 0.223}
		Q2	g++	-O3	columnX3	C	{0.682, 0.688}
	int	Q1	icpc	-O1	columnX10	C	{0.257, 0.270}
			icpc	-O1	columnX10	C	{0.264, 0.265}
			icpc	-O1	column	R	{0.261, 0.264}
		Q2	g++	-O3	column	R	{0.225, 0.236}
	long	Q1	icpc	-O1	columnX9	C	{0.206, 0.210}
		Q2	g++	-O2	columnX9	C	{0.203, 0.205}
Nehalem	char	Q1	icpc	-O2	columnX6	C	{0.650, 0.654}
			icpc	-O3	columnX6	C	{0.651, 0.653}
		Q2	g++	-O3	columnX2	C	{0.930, 0.932}
			g++	-O3	columnX3	C	{0.930, 0.932}
	int	Q1	clang++	-O2	columnX3	C	{0.628, 0.632}
			clang++	-O3	columnX3	C	{0.629, 0.631}
		Q2	15 indistinguishable best solutions				{0.610, 0.635}
	long	Q1	8 indistinguishable best solutions				{0.595, 0.625}
		Q2	20 indistinguishable best solutions				{0.590, 0.619}
Sandy Bridge	char	Q1	icpc	-O2	columnX12	C	{0.224, 0.225}
		Q2	g++	-O3	columnX3	C	{0.838, 0.840}
	int	Q1	icpc	-O1	columnX10	C	{0.291, 0.294}
			icpc	-O1	columnX11	C	{0.292, 0.293}
		Q2	g++	-O3	column	R	{0.264, 0.265}
	long	Q1	icpc	-O1	columnX9	C	{0.204, 0.205}
		Q2	g++	-O2	columnX9	C	{0.197, 0.204}
Westmere	char	Q1	icpc	-O2	columnX12	C	{0.183, 0.185}
			icpc	-O3	columnX12	C	{0.182, 0.185}
		Q2	icpc	-O2	columnX0	C	{0.636, 0.637}
			icpc	-O3	columnX0	C	{0.636, 0.637}
	int	Q1	icpc	-O2	columnX2	C	{0.243, 0.244}
		Q2	g++	-O3	columnX10	C	{0.196, 0.198}
			g++	-O3	columnX10	C	{0.195, 0.199}
	long	Q1	icpc	-O1	columnX9	C	{0.176, 0.178}
		Q2	g++	-O1	columnX9	C	{0.176, 0.179}
			icpc	-O1	columnX9	C	{0.176, 0.179}

Figure 4: Pivot table using confidence intervals to determine sets of indistinguishable best solutions

[7] J. Dittrich. The Case for Small Data Management. In *CIDR*, 2015.

[8] J. Dittrich. The Case for Small Data Management. In *keynote at BTW*, 3/2015. extended version of [7] (youtube).

[9] https://jquery.com.

[10] S. Krishnamurthi and J. Vitek. The Real Software Crisis: Repeatability As a Core Value. *Commun. ACM*, 58(3):34–36, Feb. 2015.

[11] I. Manolescu, L. Afanasiev, A. Arion, J. Dittrich, S. Manegold, N. Polyzotis, K. Schnaitter, P. Senellart, S. Zoupanos, and D. Shasha. The Repeatability Experiment of SIGMOD 2008. *SIGMOD Rec.*, 37(1):39–45, Mar. 2008.

[12] J. P. Mesirov. COMPUTER SCIENCE. Accessible Reproducible Research. *Science (New York, N.Y.)*, 327(5964):10.1126/science.1179653, 01 2010.

[13] https://mozilla.github.io/pdf.js.

[14] https://github.com/nicolaskruchten/pivottable.

[15] G. K. Sandve, A. Nekrutenko, J. Taylor, and E. Hovig. Ten Simple Rules for Reproducible Computational Research. *PLoS Comput Biol*, 9(10):e1003285, 10 2013. http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003285.

[16] http://www.vistrails.org.