Friedrich-Alexander-Universität Erlangen-Nürnberg

Technische Fakultät, Department Informatik

Ahmet Sitti

MASTER THESIS

Black-box investigation of the Ohloh data source for OSS research

Submitted on 03.06.2014

Supervisor: Prof. Dr. Dirk Riehle, M.B.A.

Professur für Open-Source-Software

Department Informatik, Technische Fakultät

Friedrich-Alexander University Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Nürnberg, 03.06.2014

License

Nürnberg, 03.06.2014

Abstract

Thanks to the collected data by providers like Ohloh, research on open source software has become both easier and popular. Ohloh is offering access to its database containing most of the statistically relevant information collected by Ohloh from publicly available version control systems. There have been many studies in open source software using the data provided by Ohloh. In this study we wanted to investigate the reliability and validity of the data from Ohloh by doing a black box investigation. We tested the reliability of the Ohloh data and found out that the collected data was not reflecting the number of all open source projects due to method changes by Ohloh. Therefore we decided to change the research question to "How to reduce redundancy in R scripting by creating an interactive toolkit?". For that purpose we develop a generic, independent and extendable toolkit, that allows the user not to waste time on repetitive tasks but rather concentrate himself on his research.

Summary - Zusammenfassung

Dank der von Ohloh gesammelten Daten über Open Source Projekte wird die Forschung in diesem Bereich vereinfacht. Ohloh bietet Zugriff auf eine Datenbank, die einen Großteil der von Ohloh gesammelten Informationen von öffentlich zugänglichen Versionskontrollsystemen bereitstellt. Die Daten von Ohloh wurden in zahlreichen Studien über Open Source Software eingesetzt. In dieser Studie wollten wir die Zuverlässigkeit und Gültigkeit der Daten von Ohloh in einer Black-Box-Untersuchung testen. In unseren Tests fanden wir heraus, dass die Daten nicht die Zahl aller Open Source-Projekte abbilden aufgrund von Änderungen der Sammel-Methode von Ohloh. Daher entschieden wir uns, die Forschungs-Frage zu ändern in „Wie lässt sich Redundanz beim Skripten in R durch die Erstellung eines interaktiven Toolkits reduzieren?" Dafür entwickeln wir ein generisches, interaktives, unabhängiges und erweiterbares Toolkit das dem Nutzer erlaubt, keine Zeit auf wiederkehrende Arbeiten zu verschwenden sondern stattdessen sich auf die Forschung zu konzentrieren.

Keywords

- R,
- split,
- apply,
- combine,
- plyr,
- data processing,
- graphics,
- toolkit

List of Illustrations

Glossary of Terms and Abbreviations

SAC : Split-Apply-Combine

SLoC: Source Lines of Code

OLAP: On Line Analytical Processing

R-OLAP: Relational OLAP

M-OLAP: Multidimensional OLAP

H-OLAP: Hybrid OLAP

API: Application Programming Interface

S-Curve: Sigmoidal curve

Goodness-of-fit: Wellness of a statistical model to the set of observations.

p-Value: Probability value for statistical tests.

Map-Reduce: Created by google it performs filtering and sorting in map operation and summarizes the output of the map operation in reduce operation.

Black-box Testing: Testing without knowing the inner workings of an item.

Acknowledgements

I am particularly grateful to Prof.Dr.Dirk Riehle, Dr.Wolfgang Mauerer and Gottfried Hofmann for their support in preparing this thesis.

Table of Contents

Contents

1 Introduction

In the last decade the number of open source projects has drastically increased and has become a data set with huge amounts of information available for academic research concerning software development. Although the amount of data has grown large, its storage is distributed across many different repositories. In order to use this data in research for software development, recently some companies such as Ohloh began collecting the distributed data from all over the web and stored it in their data store in a structured manner so that the data could be used for research purposes.
This thesis is about creating a black-box test for Ohloh data and verifying its reliability. Black-box testing focuses on the external behavior of the system that is being tested. We started by adjusting the existing code created by Carsten Kolassa to use the new database structure provided by Ohloh.net. After the adjustments had been made and we were able to analyze the data, we found out that Ohloh had changed their crawling method and the interval for project changes in the beginning of 2010. That meant data collected after the end of 2009 was not continuous and it showed us that the data was not reliable enough to make predictions. Thus using that data we could not validate the hypotheses by Riehle [5]. He built a mathematical model that describes the total growth of open source software development from January 1995 to December 2006. His model grows exponentially but should become at some point an S-curve.

Having determined in a timely manner that the data was not reliable, we decided to change the research question and thus created a generic object oriented toolkit using R classes, which makes it simple to create graphics based on parameters regardless of where the data comes from and then embed those graphics into other projects.

Empirical studies use quantitative data and are interpreted by creating graphs. The goal of this thesis is to provide a toolkit which reduces the overhead in statistical research and wraps repeating tasks so that researchers can save precious time and can focus on their research and avoid getting lost in various statistical function implementation details.

None of the current tools can both efficiently manipulate data and generate graphics regardless of the data source. Section 2 elaborates the currently available tools.

Data analysis requires the careful grouping and cleaning of data. It could need to be grouped, aggregated, sorted etc. The common strategy for doing this is called split-apply-combine [9]. We are going to introduce this concept in later chapters in greater detail.

After the introduction section we will continue with related works in Section 2. We will go into the details of our research method in Section 3. In Section 4 we will describe future plans and present the conclusion. Section 5 contains references and appendixes.

1.1 Research question

During our research we found out that Ohloh had changed their crawling methods. Now they do not crawl the entire internet for all open source projects anymore but rather add selected projects into the Ohloh database.

Therefore there was no reason to further research the black-box testing of Ohloh data. Instead, we decided to work on a toolkit that simplifies splitting, aggregating, applying statistical functions to the data and visualizing the result similar to OLAP but going beyond it.

For that reason we changed our research question to: as “How to reduce redundancy in R scripting by creating an interactive toolkit?”

The toolkit is intended to be used as a part of an application and not alone. Using just the toolkit will process the data and generate graphs based on the dimensions which can be done also using straight forward R scripting. The toolkit unfolds its benefit if it is being used as a part of an application. For example we created a web based application using the toolkit. You can see the examples in Section 3. The user can select the parameters and the rest is taken care of by the toolkit. The user does not have to write a new script in order to process and create graphs based on the selected parameter combinations.

To demonstrate it we created a scenario that compares the traditional ways of scripting to using the toolkit and shows the advantages of the toolkit.

1.2 Delimitations

Due to the endless number of functions and their various implementation details we decided to implement commonly used regression functions such as lm [1], glm [2], nls [3], arima [4] etc. Although we implemented a limited number of functions the toolkit can be extended by adding other functions to it. Although the toolkit supports creating confidence and prediction intervals, not all functions support these features. Those ones which are not supported are omitted when creating graphs and a warning message is shown to the user.

1.3 Importance and Contributions of this Theses

This work provides a toolkit which can be extended and reused in other statistical projects. The toolkit makes it possible for them to generate graphs easily by wrapping existing functions such as lm , glm , nls , arima etc. so that researchers save time in their research by not getting involved with the implementation details.

2 Related Works

Our original research question was to do black-box testing on the current Ohloh data and test and validate the integrity of the latest data from ohloh.net in order to update the results found by Riehle.

In [5] it is shown, by processing the data using statistical methods that the total number of open source projects and the total number of open source code was growing at an exponential rate. He used a database snapshot from Ohloh containing data from roughly 1995 to December 2006.

Nothing in nature grows exponentially forever [6]. At the time when Riehle conducted his research and published his paper the rate of growth was best described using an exponential model. Using the latest snapshot of the data, researchers at the OSR group wanted to test whether the model introduced by Riehle was becoming a sigmoid curve (S-Curve).

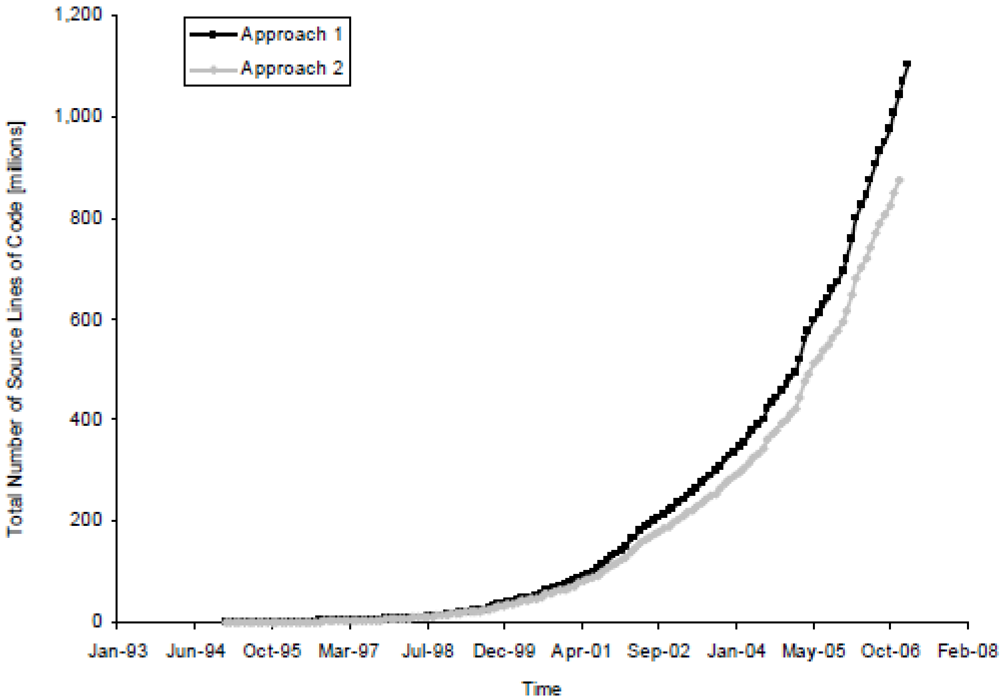


Figure 1 Graph of total source lines of code [millions] (both approaches)[1]

In "The Open Source Big Bang" [7] Riehle shows the relationship between the number of open source projects and the number of committers using data served by Ohloh between 1995

[1] http://dirkriehle.com/publications/2008-2/the-total-growth-of-open-source/

and 2008.

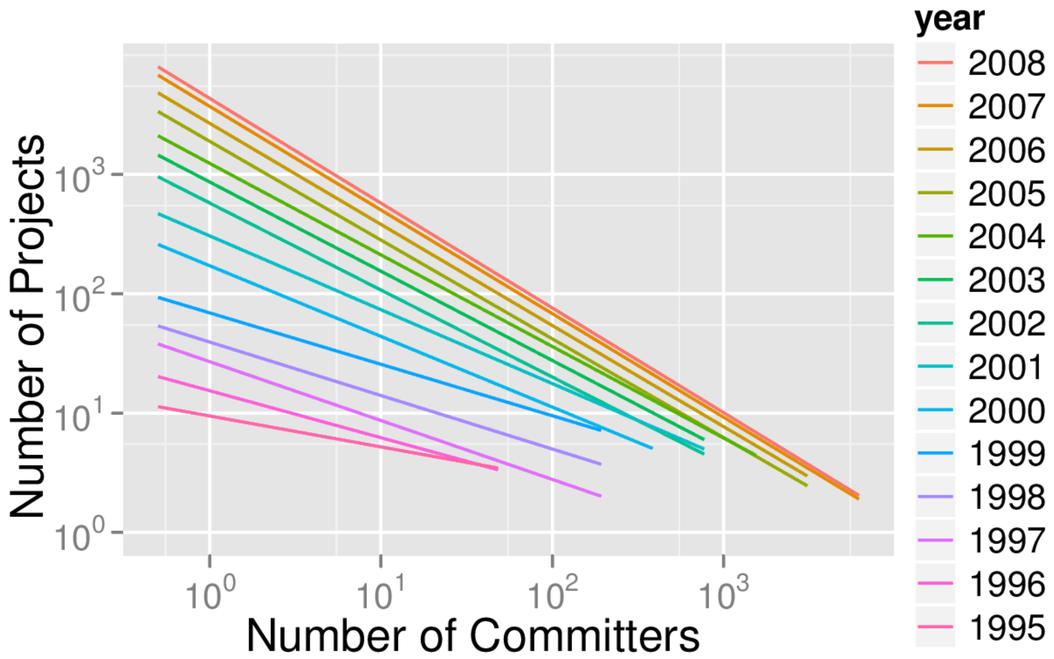


Figure 2 the Open Source Big Bang[2]

Gottfried Hofmann presents in his work [8] the total growth of source code licensed under two distinct types of licenses. He also uses Ohloh data from 1995 to the end of 2007.

All of the previously mentioned research still require validation using the latest data provided by Ohloh.

As we mentioned in the introduction section, we found out in a short time that the data served by Ohloh was not continuously crawled and fell down abruptly by the end of 2009. Since our goal was to test the Ohloh data we not only wanted to adjust the existing code by Carsten Kolassa but also create a toolkit that combines processing data and generating graphics by reducing the complexity so the researcher can focus on his research and not on the details of manipulating data and generating graphics.

In the following we briefly describe the packages and tools we examined throughout our research that are related to our work.

There are numerous libraries and functions developed by researchers to perform data analysis and plot the resulting data. The following section contains a list of tools and libraries we examined.

[2] http://dirkriehle.com/2011/06/21/the-open-source-big-bang/

Data Manipulation Packages

plyr[9] package eliminates the use of loops to focus concentration on key components of the computation. It operates similar to the map-reduce technique recently introduced by Google. Each split piece of data is processed independently of each other and finally put together. Another important aspect of plyr is that it can run its operations in parallel on multiprocessor environments. This package provides the most efficient and simple implementation of the split-apply-combine strategy.

reshape2 [10] package allows the flexible rearrangement and modification of data. Basically it is the predecessor of plyr package. Plyr is more memory-efficient and faster than reshape2.

doBy [11] package essentially does the same as plyr package, though it lacks support for parallel processing.

sqldf [12] package makes it possible to manipulate data frames using SQL statements within the R environment.

Graphics Packages

There are a number of packages for generating graphics using R data manipulated by the aforementioned R packages. ggplot2[13], rCharts[14], googleVis[15]. We will explain them briefly as follows:

rCharts package is an object oriented toolkit which wraps around existing popular web based charting libraries such as highcharts, nvd3, polychart, xCharts, Morris etc. It doesn't handle data processing.

ggplot2 package provides functions to create professional looking graphics in a few lines of code. Its layered design allows us to add layers iteratively. It is composed of independent components which can be combined as desired.

googleVis package is an interface between R environment and the google chart tools. The google chart tools is a web library that can create graphics using html and java script.

None of the tools above can both manipulate data and generate graphics at the same time.

After evaluating the aforementioned packages we decided to use the plyr package for the data manipulation and the ggplot2 package for generating graphics in our toolkit.

Developed by Hadley Wickham plyr is a data manipulation package that implements the split-apply-combine strategy to process the data. Everything plyr does can also be done using R functions but the good thing about the plyr package is that it simplifies the work. Usually you need to write quite a few lines of code to achieve the same result which you can do using plyr in just a couple of lines. It is able to deal with various types of data such as data frames, lists, matrices, and arrays. plyr supports all the functions in base R and other R packages. Although we use the plyr package in our toolkit, the toolkit can also handle data without grouping it.

ggplot2 is a generic graphics package again developed by Hadley Wickham that creates graphics using layers. A layer consists of data, statistical transformations, mappings, and position adjustment information. ggplot2 can create layer based graphics. Each layer may have its own data, statistical transformations, geometric objects and position adjustments.

Both plyr and ggplot2 will be explained in greater detail in Section 3.

3 Research

3.1 Introduction

Ohloh collects statistical information about open source projects from open source repositories and stores it in its database. Using the latest data provided by Ohloh we plotted the graphs below showing the behavior of monthly number of added lines of code, monthly number of contributors and monthly number of commits between 1995 and 2013.

Summing the total added lines of code for all projects in the new database we get the following graph:

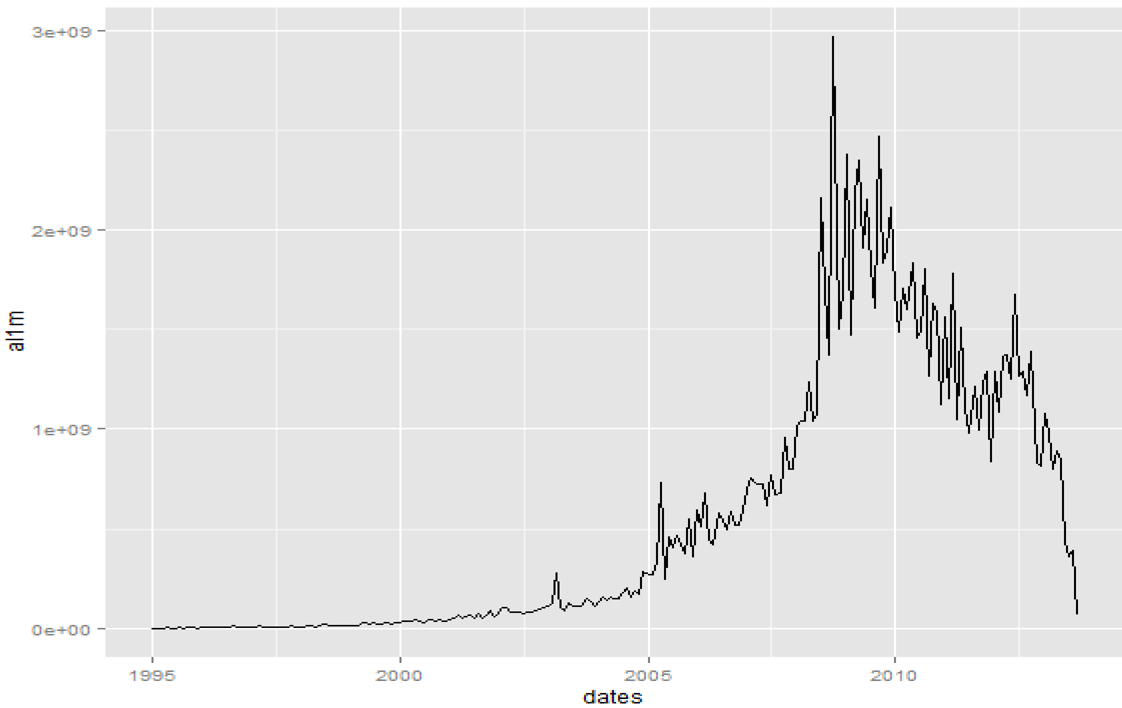


Figure 3 Number of added lines of code (al1m)

By the end of 2009 the monthly activities fall while they have grown roughly exponentially up until then.

The situation is similar with total number of commits and active contributors:

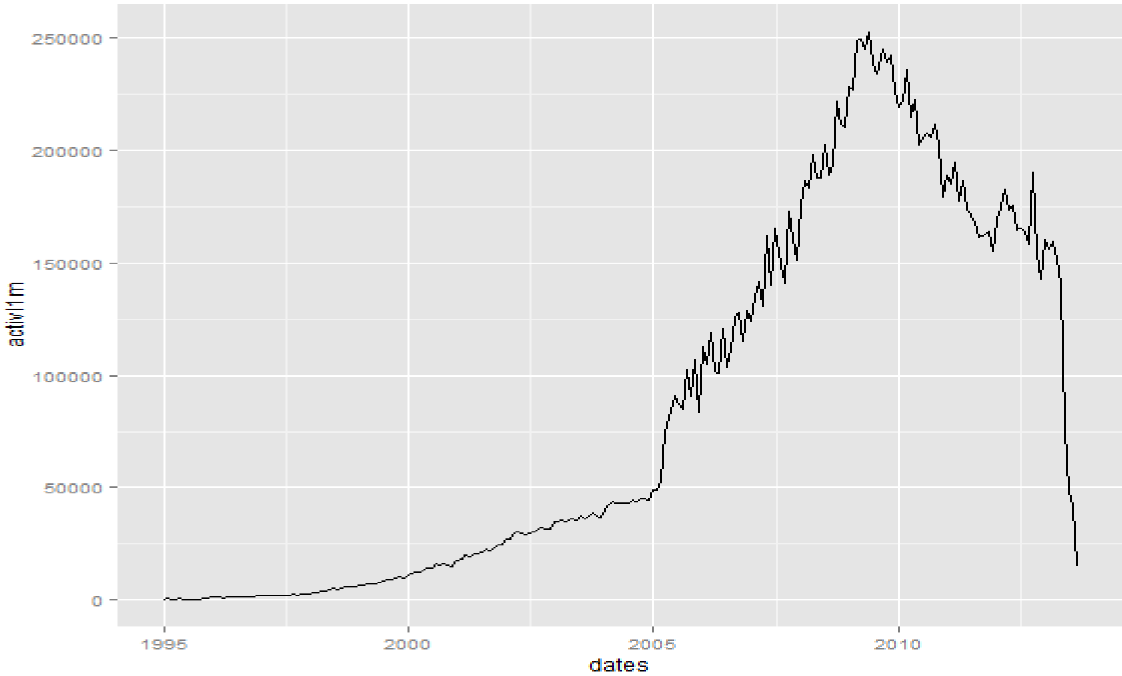


Figure 4 Number of active contributors (activl1m)

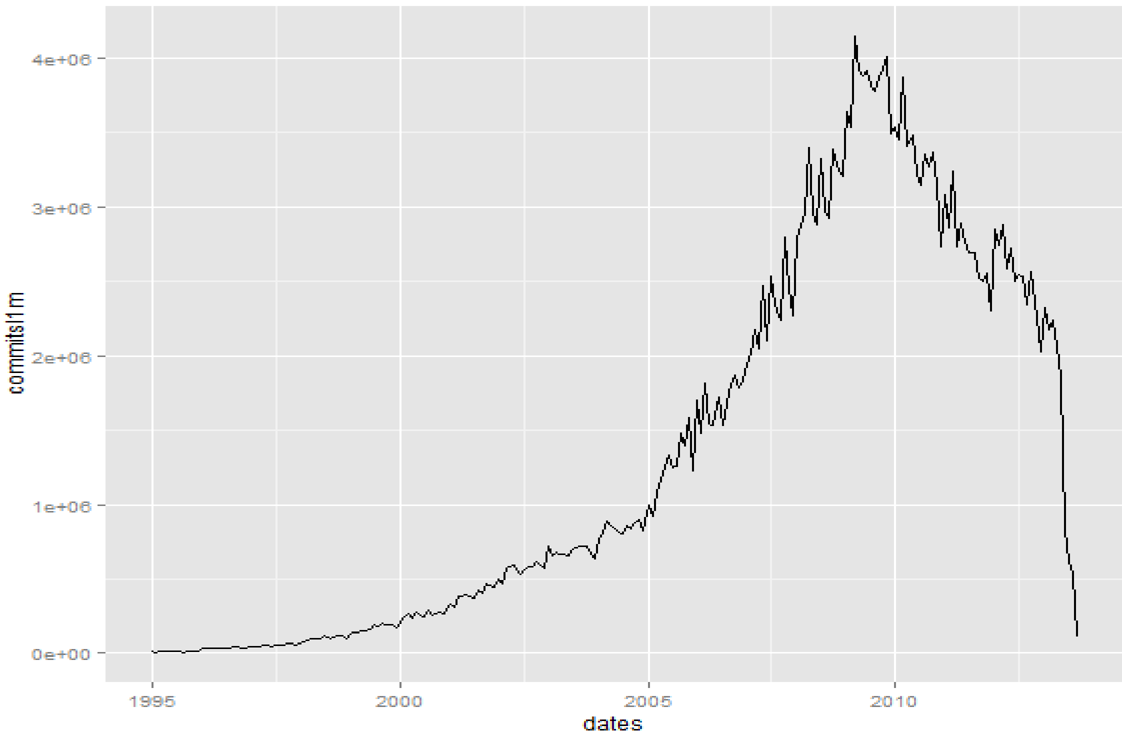


Figure 5 Number of commits of all projects over time (commitsl1m)

When we apply our new toolkit to the total lines of code (and Active contributors or commits) for each month we see an S-curve is emerging:

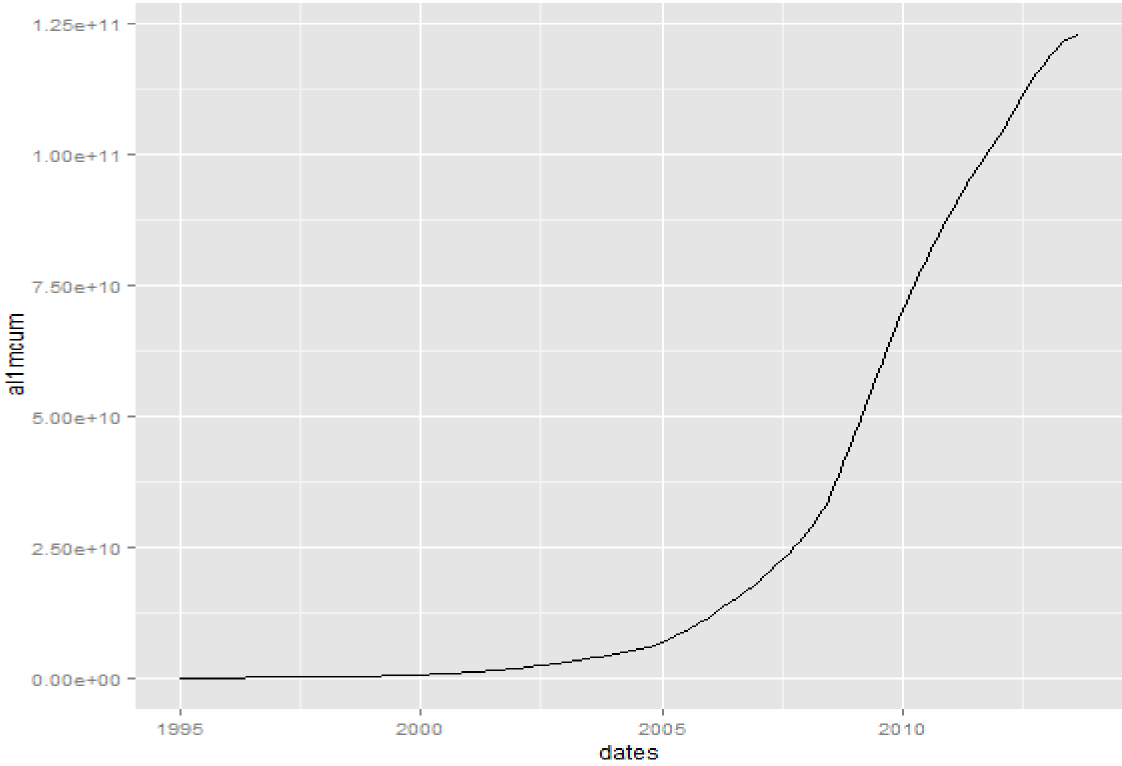


Figure 6 Cumulative sum of added lines of code between 1995 and 2013

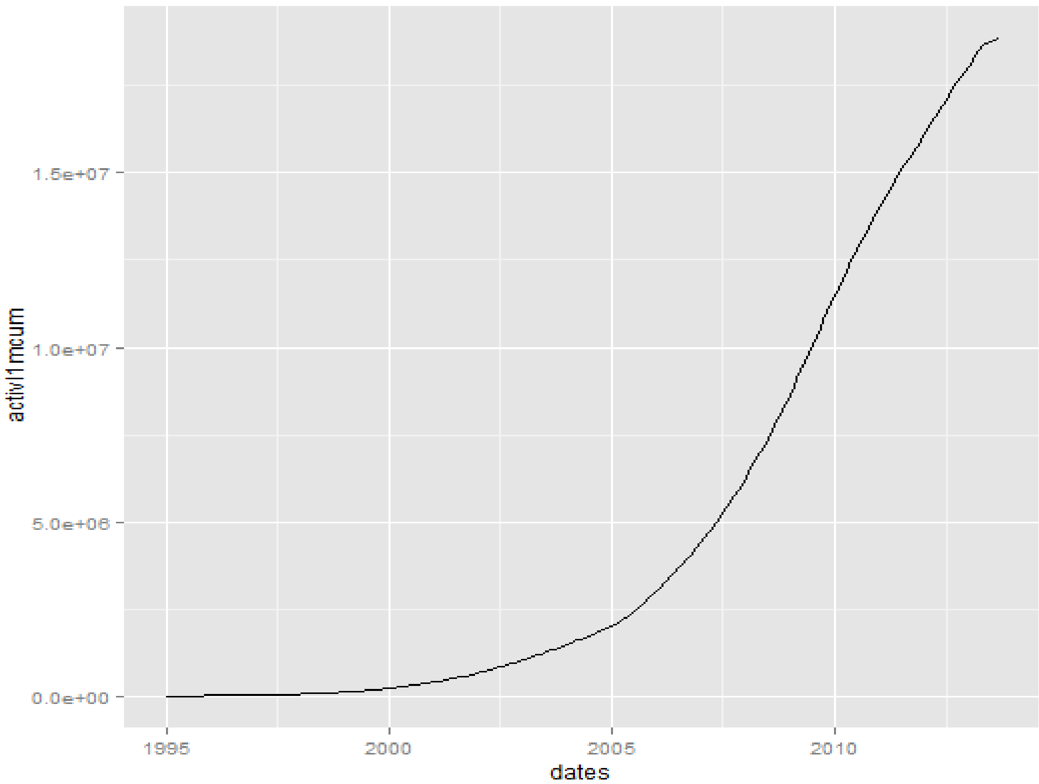


Figure 7 Cumulative sum of active contributors between 2000 and 2013

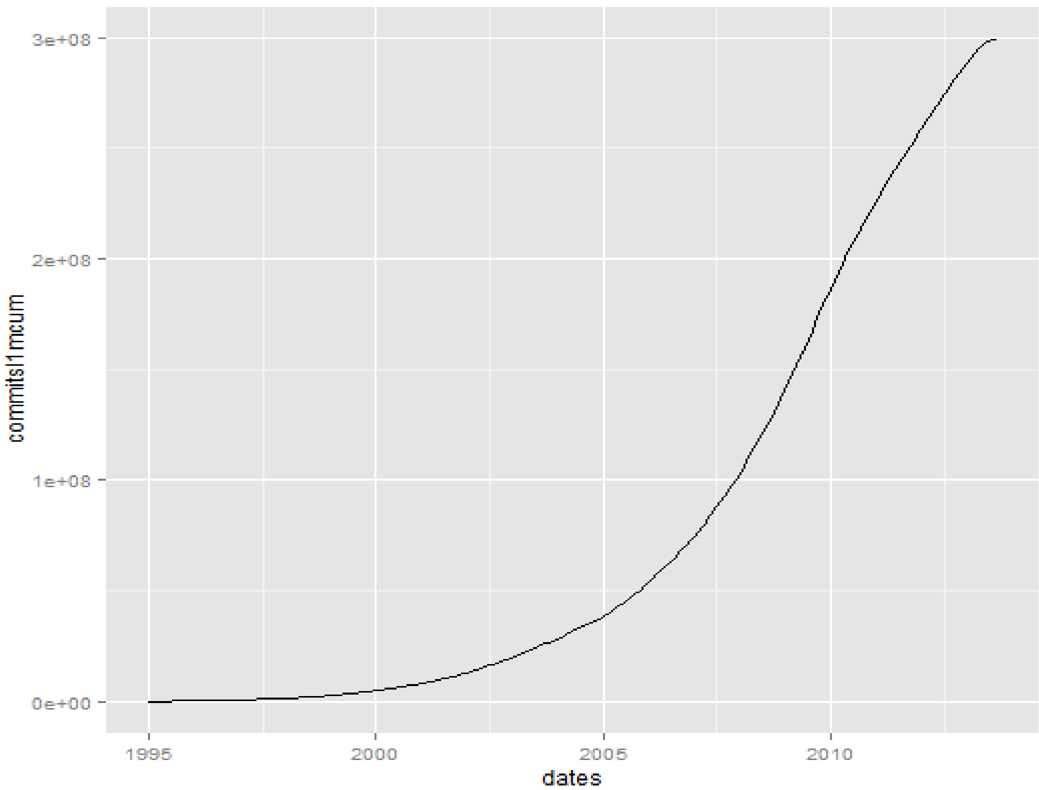


Figure 8 Cumulative sum of commits between 2000 and 2013

If the latest Ohloh data was correct, then we would have found out that the growth of open source software development has reached a maximum in 2009 and the total amount of open source code developed would have been an S-Curve.

Yet the sudden fall in Figure 4 and Figure 5 at the end of 2009 looked suspicious so that we decided to do a black-box testing using the latest Ohloh data.

When we started working on this thesis, we wanted to use the code developed by Carsten Kolassa for our testing. He was using data from the old Ohloh database which only had data until 2008.

Since the existing code developed by Carsten Kolassa used the Ohloh data we had to extract the data from Ohloh database again using ohloh-api and adjust his code to the new database structure.

We stored the extracted data in a local postgre database. Figure 9 in section "Obtaining the data" shows the schema of the local database. The reason for getting the data into the local data storage was, that although we could query Ohloh database we couldn't optimize it or modify the structure of the data at will. Additionally, Ohloh restricts the maximum number of API request per day so the local storage in the database also works as an offline cache.

To extract the data from Ohloh and store the results in the database, we used the R-script “Coche” developed by Gottfried Hofmann [16].

The script developed by Carsten Kolassa was just a straight forward R script which had to be adjusted every time if the data was changed. Instead of using one long script, we decided to create a toolkit, which is modular and generic so that it would need only minimal adjustments if the data was changed.

While adjusting Carsten Kolassa’s code we noticed that the sudden fall was too extreme and therefore we decided to get in touch with Ohloh and investigate the reason for this steep fall. We were told by Ohloh that they had changed their crawling methods. Now they are not crawling the entire internet for all open source projects but rather add selected projects into the database.

Having gotten this information from Ohloh, there was no reason to conduct further research on the black-box testing of Ohloh data. Instead, we decided to work on a toolkit that simplifies splitting, aggregating, applying statistical functions to the data and visualizing the result similar to OLAP but going beyond it.

3.2 Ohloh

We use the database of the open source analytics firm Ohloh Inc. [17]. Ohloh.net is a free and open projects directory that provides various software metrics about both active and idle projects such as commits per month, added lines of code, removed lines of code, language etc. The collected data is stored in relational databases and is publicly available through an API provided by Ohloh[18]. The collected data contains not only high-level information such as project name and author name but also low-level information such as every change to a project.

Based on the aforementioned metrics, Ohloh creates statistics about a project, which provides a basis to do research with the data as well as allowing companies to make informed decisions concerning whether to invest in free software or not.

Ohloh.net grew rapidly. It consisted of 99,977 people with 9,824 projects in 32 languages and 58 documented open source licenses in December 2007. In October 2009 there were already 399,334 people with 413,261 projects in 77 languages and 253 open-source licenses.

As of 6 July 2013, the site lists 590,310 projects, and 538,806 source control repositories[3]. Ohloh is owned and operated by Black Duck Software.

Ohloh created an API and has made it public since 2007. Using the API we created a local database containing a snapshot of the Ohloh database filled with the relevant data required in our work.

Ohloh-api is a webservice which allows you to query the Ohloh database by using an API key. The API key is provided by Ohloh upon registration at their website. The number of daily transactions is limited to 100K maximum. There is a comprehensive documentation at the ohloh-api website explaining all the details of the API usage and database structure of the Ohloh database.

3.3 Obtaining the data

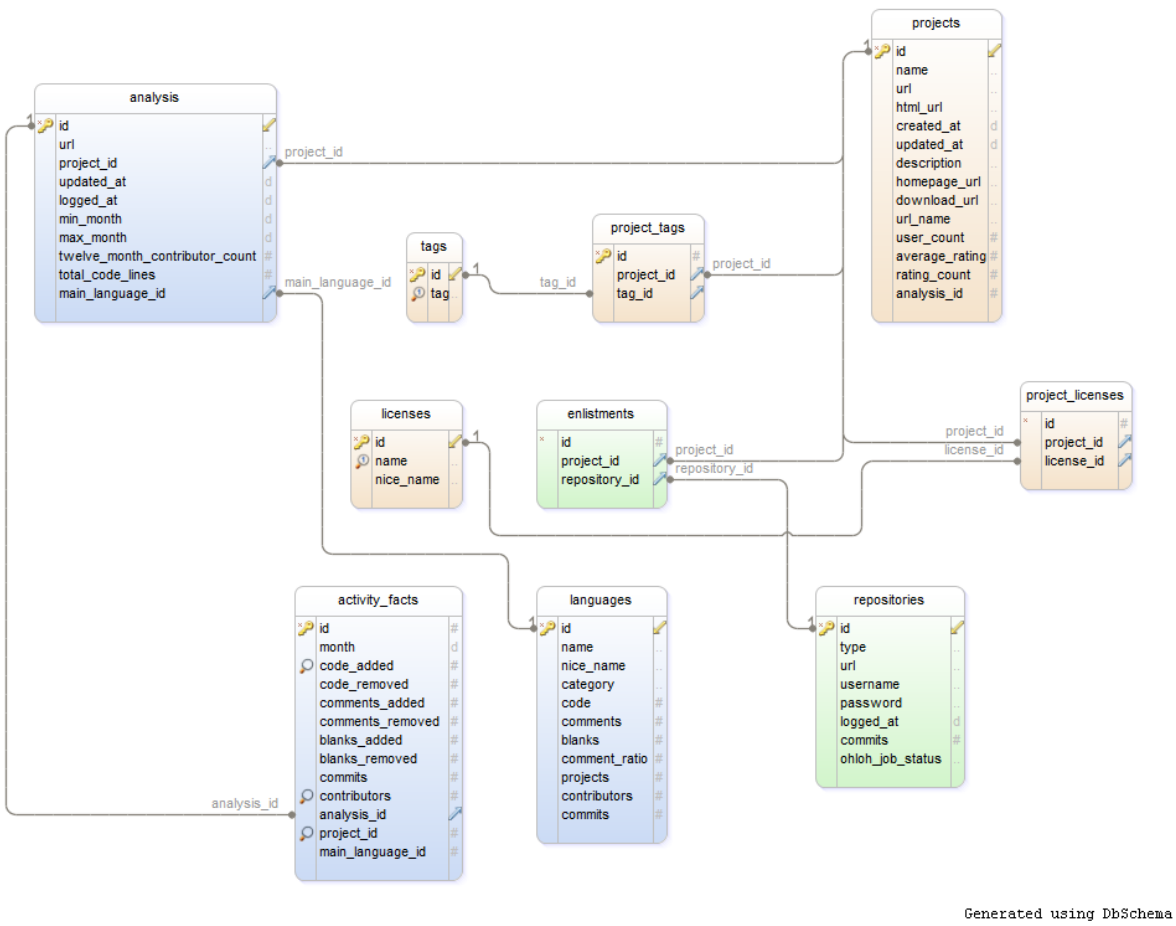


Figure 9 Ohloh Database Schema

[3] http://en.wikipedia.org/wiki/Ohloh

De-normalization

Before we could start using the snapshot of the Ohloh database, we de-normalized the data for performance reasons. A project may have more than one language used in it. Since we were interested in the main programming language used in the projects, we created an additional field called main_language_id in the activity_facts table and populated it with the main_language_id from analysis table. The scripts used to create this new field in the activity_facts table and populate it with the main_language_id from the analysis table can be found in the appendixes.

Creating ohlohdata table and Metrics Employed

The data we used contains the following metrics:

Column	Description
dates:	Activity month
projected:	Id of the project
teamsizes:	Twelve month contributor count
al1m:	SLoC of added lines in the past month
rl1m:	SLoC of removed lines in the past month
commitsl1m:	Number of commits in the past month
activl1m:	Number of active commiters in the past month
vcs:	Version control system
language:	Name of the main programming language
license:	Name of the license
projectname:	Name of the project
id:	Auto incrementing primary key

We created a script called ohlohdata.sql which can be found in the appendixes. The script gathers relevant information from the snapshot ohloh-api database and merges them into the ohlohdata table.

3.4 Repetition of Experiments

As mentioned before we started working on this thesis by adjusting the code written by Carsten Kolassa. We were able to generate a few plots using Carsten Kolassa's code. At some point we couldn't move forward since the old data used by the code employed many supporting tables which were extracted and created by Carsten Kolassa and were not documented well enough, so we could not recreate them. Figure 10 shows an example we created using Carsten Kolassa's code.

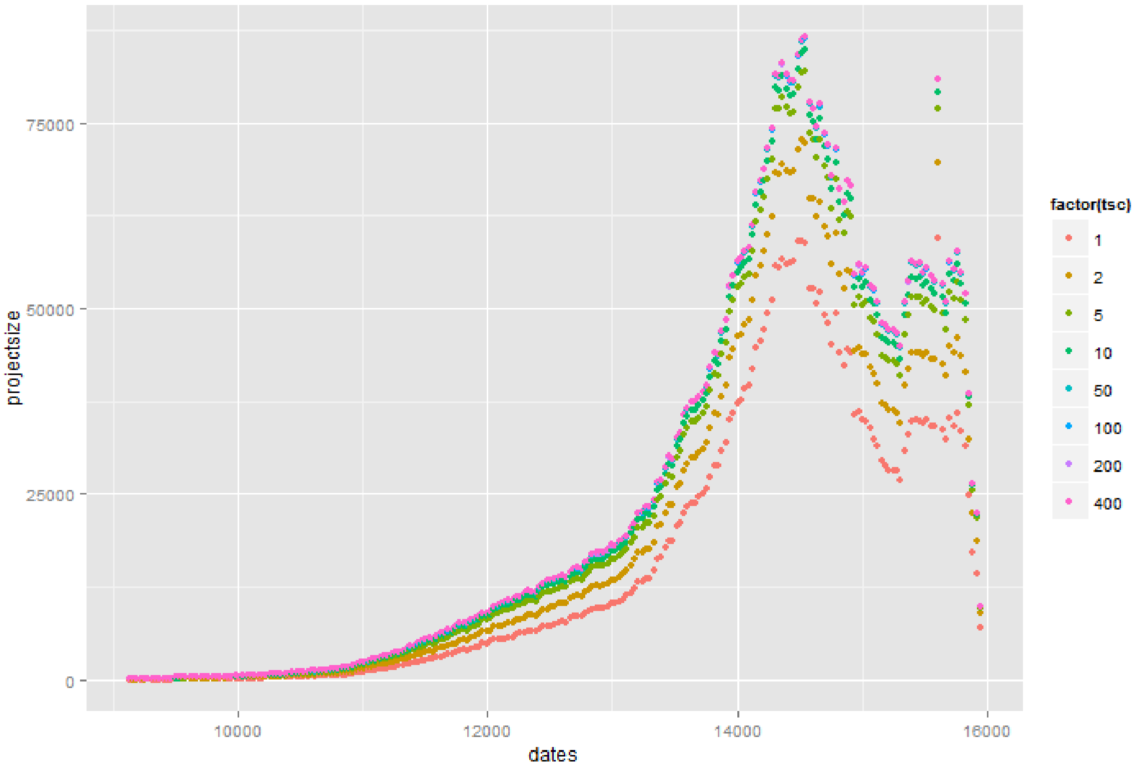


Figure 10 Teamsize vs Number of Projects with up to factor(s) contributors during a given month.

The steep decline in Figure 10 looked even more suspicious than in the graphs shown in section xx. Having noticed this we contacted Ohloh for more information and clarification. Ohloh told us that by the end of 2009 they had changed their parsing and crawling strategy. Before, they used to crawl the web and add all new repositories they could find into their database. Now they are parsing and crawling rather selected projects of interest. Therefore we see a sudden fall in Figure 10.

3.5 The Toolkit

Empirical research uses collected data or observations in order to answer research questions. Our toolkit helps researchers to search and extract patterns within empirical data by processing and visualizing it.

In the following section we describe the requirements for the toolkit and clarify concepts such as split-apply-combine and OLAP. Finally we will go into the details of the toolkit development and toolkit architecture.

3.5.1 Toolkit Requirements

The toolkit should be developed using object oriented techniques so that it can be reusable and generic. By generic, we mean no code repetition and independent from the passed parameters. Instructions are passed into the toolkit using parameters. Data is passed into the toolkit as R data frame.

Requirements:

- Apply pre-defined statistical functions to data
  it should be able to apply statistical functions such as regression functions, aggregation functions.
- Possibility to extend the toolkit with more statistical functions
  it should allow the user to add more functions without modifying the whole toolkit.
- Plotting the pure data and / or the results of the statistical methods
  it should be able to use not only aggregated but also data without aggregation.
- Unified data structure for plotting
  it should produce a common data structure after processing data so that the plots look consistent.
- Optional aggregation of data
  it should allow the user whether to aggregate the data or use it without aggregation.
- Independently assign dimensions to axis for plotting

  The explanatory and response columns should be exchangeable and be propagated throughout the process, for both statistical analysis and plotting.
- Object oriented parameter handling for simple integration into GUIs/web frontends
- Possibility to split the data by given dimensions and apply the statistics to the resulting chunks of data
  it should be able to apply statistical functions such as sum, mean, avg, lm, glm , nlm to grouped data
- Independent of data source
  it should allow to use any type of data source provided that the data is in a data frame.
- Generic
  it should be internally generic so that it would execute for any type of data and dimension.

3.5.2 Split- Apply-Combine Strategy and Plyr Package

Most of the time R is used to modify data frames, matrixes, and data tables with grouped data. The "apply" functions (apply, sapply, lapply) are a very powerful suite of tools for looping through data and returning the combined results. Although these functions are very useful, they can be difficult to work with. Returned data must be adjusted and processed again most of the time to have it in the expected format.

Hadley Wickham developed a package called plyr which makes the aforementioned tasks easy to code and fast performing by implementing a SAC (Split Apply Combine) strategy. The SAC approach breaks up the data into smaller pieces, processes those pieces individually and finally puts them back together. This strategy is well known and there are functions within R you can implement this strategy with. What Hadley Wickham did is unifying the abstraction to the existing functions.

From the documentation:

"plyr is a set of tools that solves a common set of problems: you need to break a big problem down into manageable pieces, operate on each piece and then put all the pieces back together. It's already possible to do this with split and the apply functions, but plyr just makes it all a bit easier..."

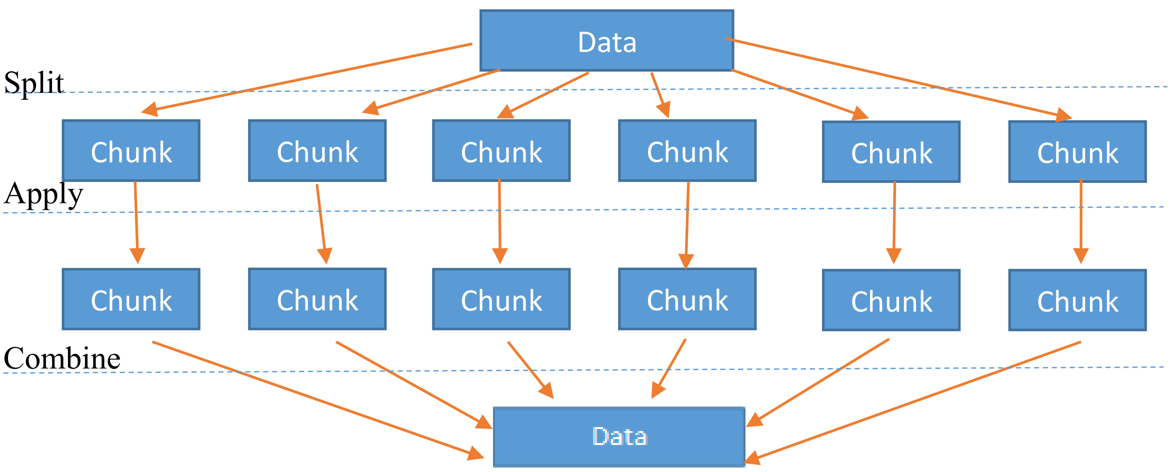


Figure 11 Split-Apply-Combine work principle

The “plyr” package eliminates the use of loops and concentrates on key components of the computation. It operates similar to the map-reduce technique recently introduced by Google. Each split piece of data is being processed independently of each other. Therefore it cannot be used with overlapping data.

The plyr package greatly simplifies many lines of code for a computation into one line of code where the details are taken care of.

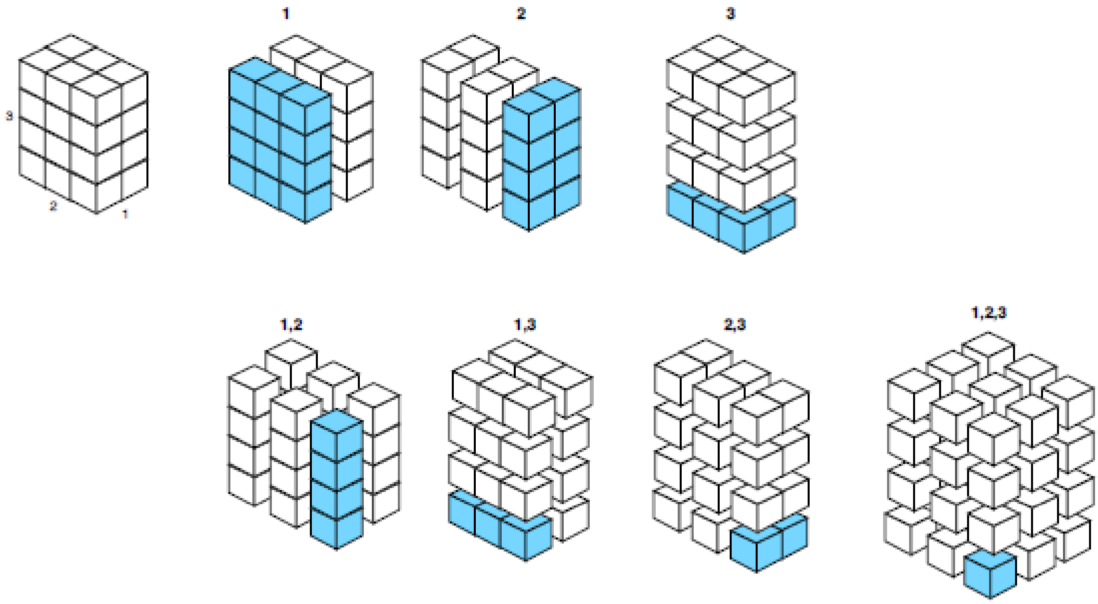


Figure 12 Possibilities of splitting a 3D array[4]

3.5.3 OLAP

Today's businesses are faced with a flood of data of various types, from many different sources. These huge databases consist of valuable data for the company. This data can be used to better position the company in the market to uncover problem areas or to optimize its production processes. Information is becoming an increasingly important factor for companies. Only those who will have current, detailed and meaningful information can create a better long term position for their company.

In particular, management can perform better analysis and provide better predictions for the future, and thus also make better business decisions with accurate data. Therefore, it is crucial for the success of the company to have an efficient data analysis system and to integrate it into the company's IT structure. However, the task to filter out useful information from the huge amount of data is very complex.

[4] Figure from [11], page 7

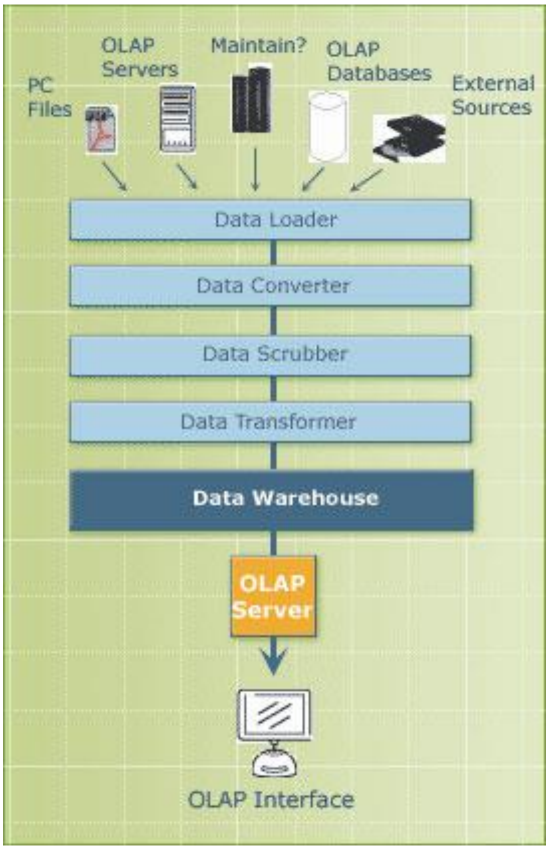


Figure 13 Data Warehouse and OLAP[5]

OLAP (OnLine Analytic Processing) helps us to explore, visualize and model large amounts of complex data using a multidimensional data model. It simply deals with mountains of data by splitting, slicing, and aggregating the data into manageable results, where you can see a tendency or a pattern and make your decision based on that summarized information. OLAP's data model is usually described as a cube since it consists of multidimensional data.

OLAP technologies are divided into three categories on the basis of the used server-side data management:

1. **R-OLAP (Relational OLAP)**

Relational data is stored in the core data warehouse, using either star or snowflake schemas. R-OLAP accesses this relational data, and prepares the requested multidimensional results. ROLAP is sometimes referred to as a focused version of MOLAP.

2. **M-OLAP (Multidimensional OLAP)**

Multidimensional On-Line Analytical Processing (MOLAP) uses proprietary database systems, which are optimized for multidimensional data. The data is stored in data marts and

[5] http://www.elml.uzh.ch/preview/fois/DSSII/en/html/lu2_learningObject2.html

by doing so the flexibility increases and the response time gets faster.

3. **H-OLAP (Hybrid OLAP)**

Hybrid OLAP combines the advantages of both R-OLAP and M-OLAP. When the user pushes forward through drill-down navigation into more detailed data areas, relational data is used.

Due to ever increasing data volumes, the queries can lead to bad performance. For this reason, the data is distributed company-wide into smaller data pools, containing data for a specific department or subject (data marts).

OLAP Operations:

1. **Pivoting/Rotating**

   this operation rotates the cube by swapping the dimensions on its own axis and allows grouping data with different dimensions. The data can be analyzed from different an-gle.

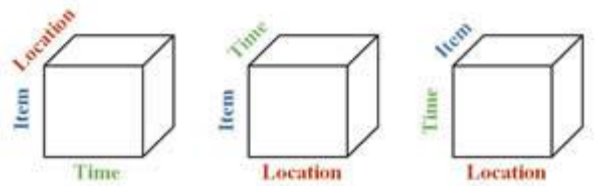


Figure 14 Pivot/Rotate[6]

[6] http://www.cis.drexel.edu/faculty/song/courses/info%20607/tutorial_OLAP/operations.htm

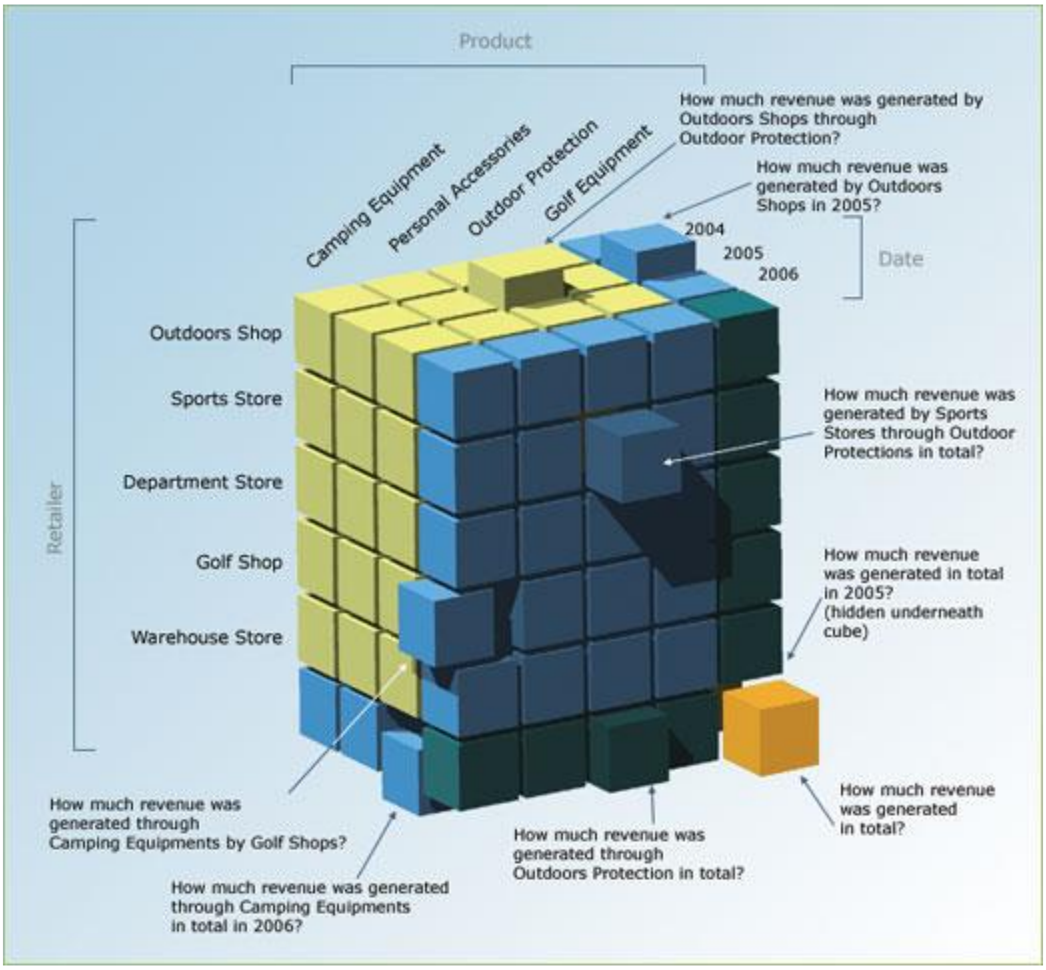


Figure 15 OLAP Cube[7]

2. **Roll-up , drill-down and drill-across**

   Roll-up aggregates the current aggregation one level upwards on one or more dimensions. Drill-down is the opposite of the roll-up operation. Drill-across combines OLAP cubes on a common dimensions in both OLAP cubes.

[7] http://www.elml.uzh.ch/preview/fois/DSSII/en/html/lu2_learningObject3.html

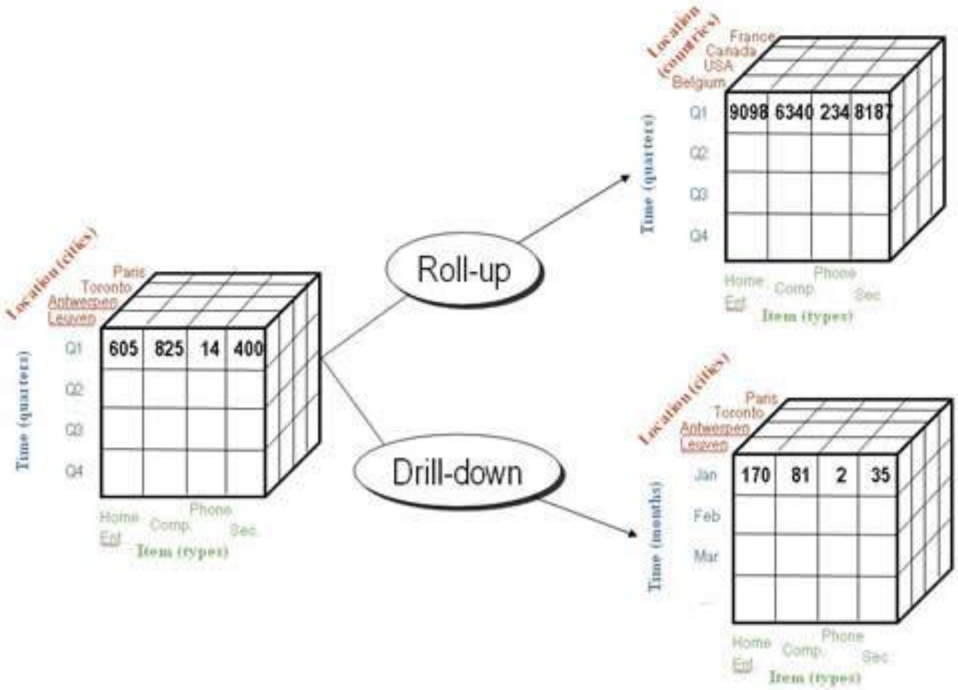


Figure 16 Roll-up, Drill-down[8]

3. **Slice and dice**

   Individual views on multidimensional data cubes can be achieved by Slice and Dice. A slice is the view of a part of the cube achieved by holding one dimension, resulting in a sub-cube.

   Dice are considered partial cubes for specific combinations and thus corresponds to ad-hoc requests. The Dice operation results in a sub-cube selecting one or more dimensions.

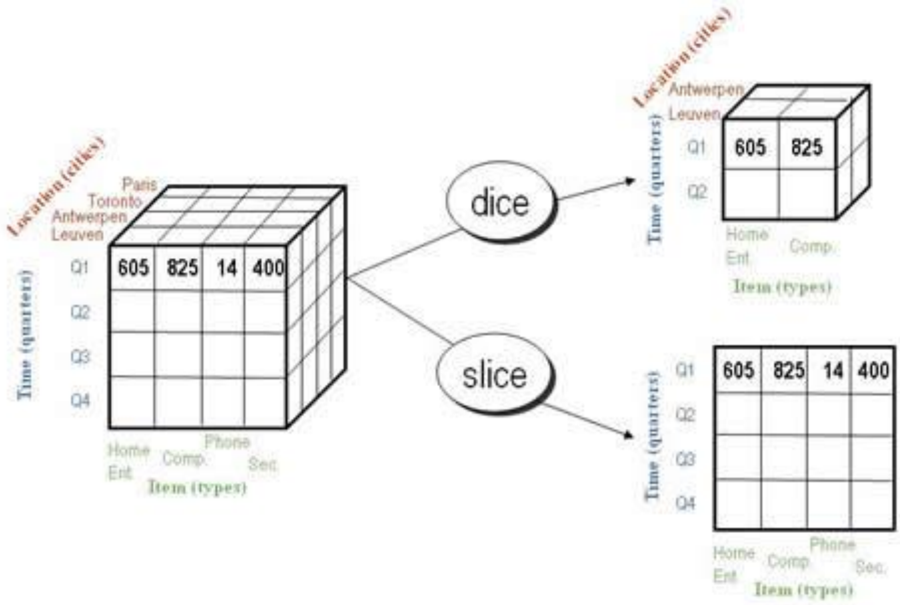


Figure 17 Slice, Dice[8]

[8] http://www.cis.drexel.edu/faculty/song/courses/info%20607/tutorial_OLAP/operations.htm

Data mining is a further analysis approach and aims to use, relationship patterns, such as regularities and irregularities in the underlying data to identify models and map them by logical or functional relationship contexts.
Data mining allows you to carry out analysis and correlations of the data. The analysis in data mining is determined by questions such as ' What is the trend in sales of groups of countries?' It leads to the identification and mapping of relationship contexts in the form of a model. The analysis is being done using the following methods:

1. **Clustering**
   aims to collect the underlying data into groups based on their characteristic values.
2. **Classification**
   assigns the underlying data set to classes.
3. **Regression**
   is used to determine the relationship between individual characteristics of the underlying data set.
4. **Dependency**
   identifies relationships between different forms of characteristics of the underlying dataset.
5. **Deviation**
   determines whether the characteristics in the underlying data differ distinctively from other values of the characteristics.

The main difference between OLAP and data mining is that OLAP is an analysis method, which allows the user to summaries data and generate rich calculations, where data mining is a method that discovers hidden patterns in data and it operates at the detail level and not at a summary level.
Data mining and OLAP can complement each other. Where OLAP reveals the problems with sales of products, data mining could be used to find out the customers' habits in a particular region.

3.5.4 OLAP vs Split-Apply-Combine

Although both OLAP and SAC have similar operations, SAC is better if you want not only to group, aggregate or select the data but also apply statistical functions to the data. In OLAP you can split and aggregate the data but the result will be still a multidimensional dataset which in turn has to be processed again in R using SAC techniques in order to apply statistical functions.

Our toolkit uses the plyr package and also provides the capabilities of an OLAP tool[9] in the R environment. The toolkit processes data, runs statistics on the data and returns the plots in one step directly in the R environment.

OLAP uses pre-calculated cubes and it needs a special environment with specialized data structures to store that data. The data has to be prepared before it can be analyzed. For example using R-OLAP you need to change your database schema to either the snowflake or star schema[10].

Using S-A-C in R we can run statistics not just on the whole dataset but on subsets based on dimensions. We can easily select, aggregate, put data into relation, and apply statistics on chunks of data. This is all achieved simultaneously, interactively, and on-the-fly.

---

[9] Although not all of the OLAP operations can be found in the toolkit they can be implemented with little effort.

[10] Using the new Ohloh data we had de-normalized the data a little due to better performance.

3.6 Toolkit Architecture

The toolkit is composed of parameter classes, the gfw.chart class and the gfw.chart.model.processor class.

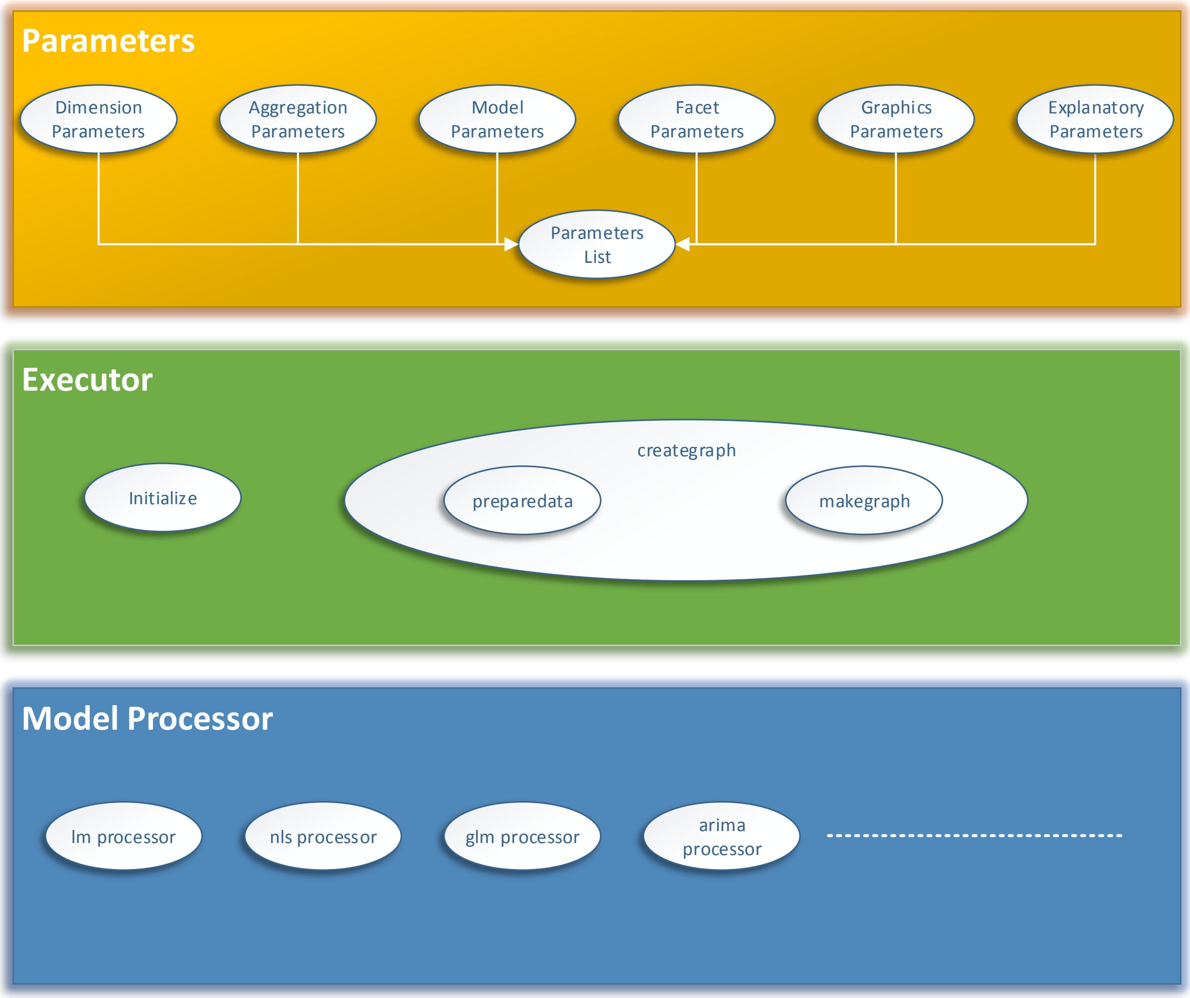


Figure 18 Toolkit Components

The execution process is being illustrated as a sequence diagram in Figure 25.

Parameter classes hold the instructions about what the toolkit should do with the data during execution. They derive from the gfw.chart.param base class. There are five parameter classes which derive from the gfw.chart.param class and are as follows:

- gfw.chart.param.dimensions

  Holds dimensions and Boolean values that determine whether confidence intervals and prediction intervals should be used or not.

- gfw.chart.param.aggregation

  Holds the aggregation columns and aggregation function which are implemented in the first pass of data preparation within preparedata method in gfw.chart class.

- gfw.chart.param.model

  Hold the model function and model formula. Additional parameters can be used due to (…) in class initialization.

- gfw.chart.param.facet

  Holds the facet function and facet scale values. The Facet function splits up the data by one or more variables in subsets and plots them together. By default the axis scales are fixed but by using the facet scale value it can be changed to one of the following values: free, free_x, free_y. free indicates that each plot will have its own scales.

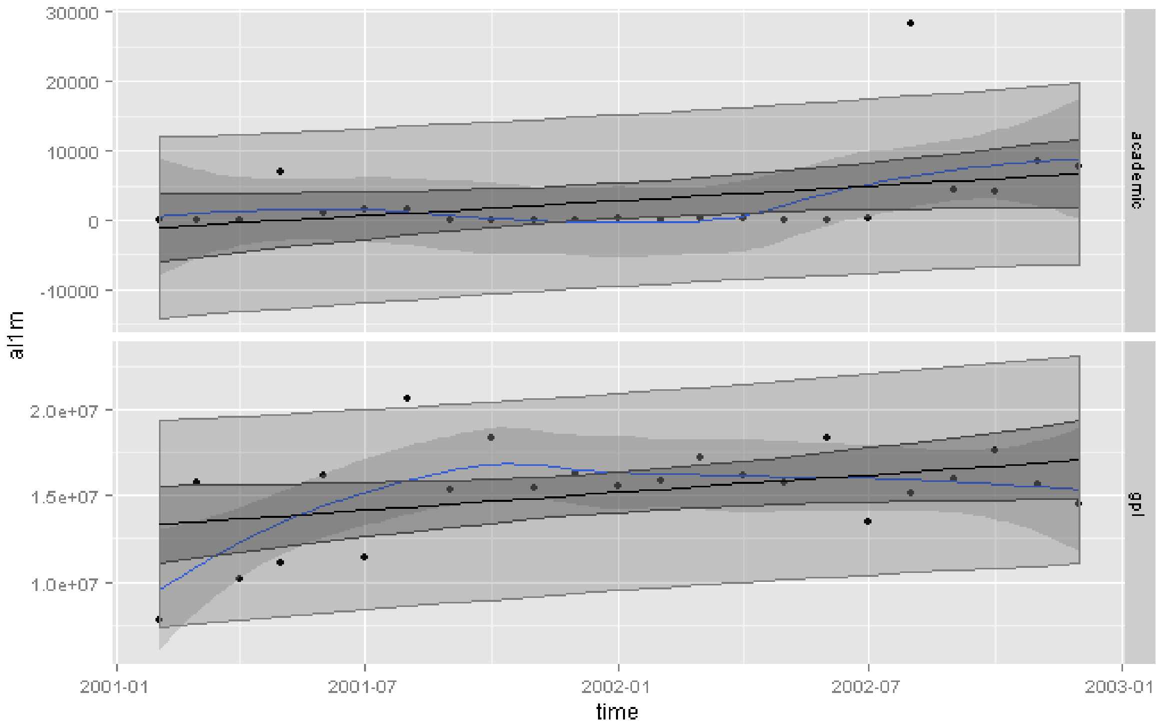


Figure 19 fixed scales facet[11]

[11] http://www.cookbook-r.com/Graphs/Facets_(ggplot2)

free_x allows each plot to have its own x-axis values and free_y does the same for the y-axis.

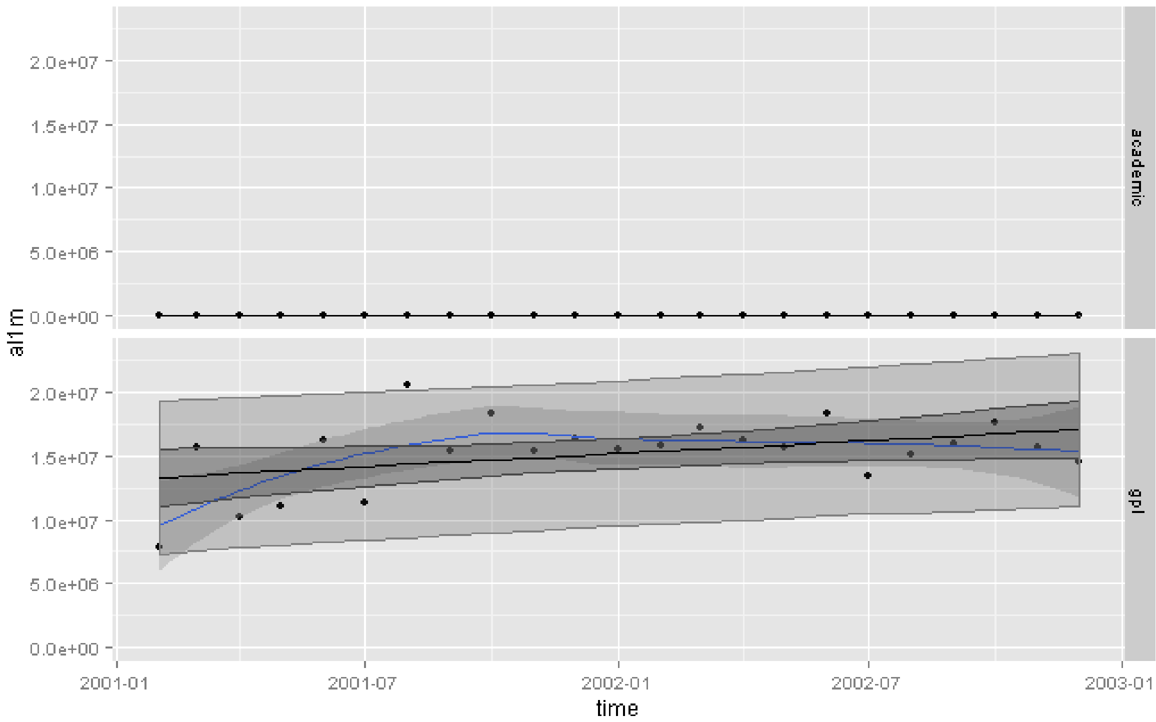


Figure 20 free_x scales facet[11]

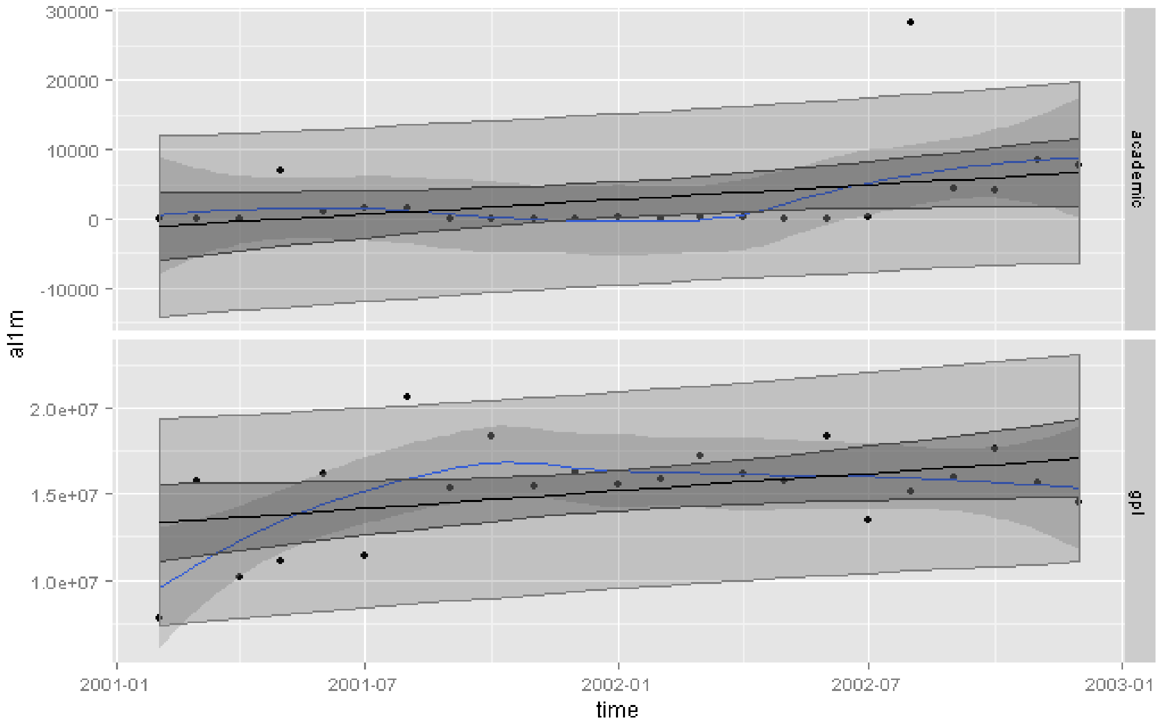


Figure 21 free_y scales facet[11]

- gfw.chart.param.explanatory

  Holds the column name which will be used on x-axis if there is no aggregating column.

- gfw.chart.param.graph

  Holds list of gfw.chart.param.graphic objects which define the layers of ggplot function. First layer has parameters for ggplot function itself. Additional parameters are the layers of the plot and each layer may have its own “aes”, “data” and “legend” parameters.

The “Executor “ gfw.chart class uses the parameters from “Parameters” in order to extract the needed data from the data frame, aggregate it, pass it to the gfw.chart.model.processor class for statistical processing and finally pass it to the graphing function for graph generation. When all the data for all dimensions is processed, the result is a list of graphs ready for plotting.

As mentioned before the gfw.chart.model.processor class in “Model Processor” gets the pre-processed data from gfw.chart class and applies statistical functions to the data based on the parameters and returns it back to the gfw.chart class for graphing.

The object oriented manner of the toolkit is limited to the capacity of object oriented development in the R environment. It does not support the full range of OOP features and therefore it does not support the full strength of OOP-based design patterns.

As an example limitation of R’s object oriented programming capabilities let us consider interfaces. In R you do not have interfaces which makes it difficult to implement most of the OOP-Design Patterns.

In R a class can be declared an abstract class by using the keyword “VIRTUAL” in the definition of the class. This keyword makes the class abstract and it cannot be instantiated.

The toolkit builds on top of two packages plyr and ggplot are discussed further in the following section.

For aforementioned reasons we could not utilize design patterns as they were supposed to be but rather we ended up with simplified pseudo patterns.

One of the most important requirements of the toolkit is that it should be independent of data source and generic so that the toolkit need not be changed when the data structure, column names, and etc. change. The data might come from a relational database server or from a text file or OLAP database etc. As long as the data is in data frame format the toolkit will be able to handle the data. The toolkit is generic since the only thing that needs to be done to adjust the parameters. Unless you want to add more functions to the toolkit and need to make modifications in the graphing function, there is no need to modify the source code. The toolkit kit converts the parameters internally to generic names and uses them.

We also utilized a concept called split-apply-combine by using the plyr package. The Split Apply Combine Strategy is a well-known strategy to process statistical data. SAC allows you to analyze data based on groups defined by dimensions. Instead of working with a huge chunk of data at once, the data is being split into smaller chunks and is processed independently from other chunks. Finally the processed chunks are recombined and returned. This way you can process multidimensional data very efficiently.

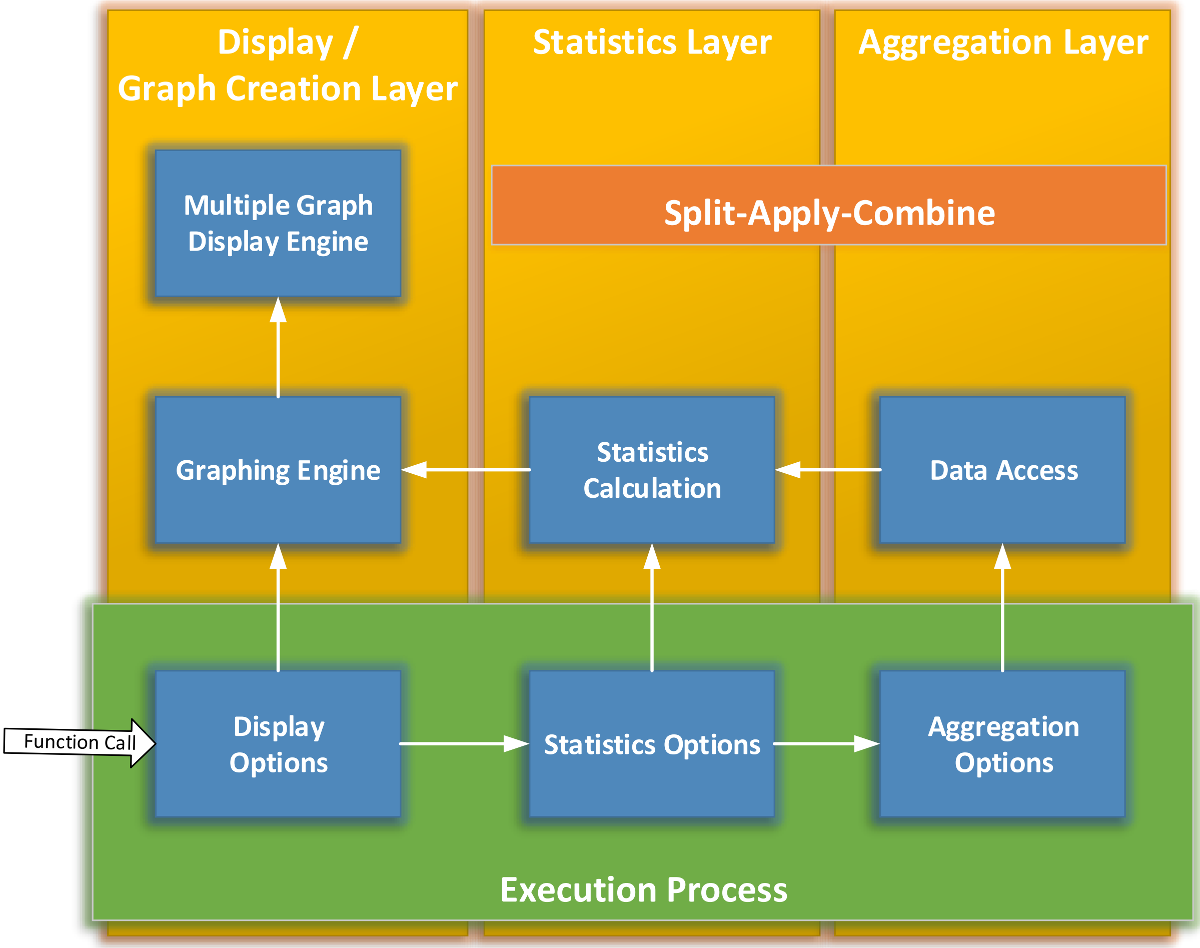


Figure 22 Toolkit Architecture

Figure 22 shows the conceptual architecture of the toolkit. The green box shows the execution of the toolkit using parameters. The yellow boxes are the individual layers where the data gets selected, aggregated and statistically analyzed. The blue boxes represent the methods and their parameters of the toolkit.

The toolkit takes list of parameters that contain the instructions about what the toolkit should perform with the data. The execution process goes through the layers and applies the instructions using the corresponding parameters. The toolkit eliminates unnecessary data by selecting data based on dimensions and aggregation parameters from the input data. The selected data is aggregated in the aggregation layer. The data can also be used without aggregating it. Then it is passed to the statistics layer where statistical functions are applied to the data. Finally the graph creation layer gets the processed data and creates a list of graphs. The resulting list of graphs are displayed by the multiple graph display engine.

3.7 Toolkit Development

In this section we will explain our methodology and go into the implementation details of this work. Before we started the development of the toolkit we decided to develop using object oriented design for the sake of maintainability and simplicity.

The toolkit consist of three main components: gfw.chart, gfw.chart.datasplitter and gfw.testdata.

As mentioned earlier in Related Works the toolkit uses two packages, plyr and ggplot. Both plyr and ggplot2 packages were developed by Hadley Wickham and will be explained as follows.

3.7.1 Plyr Package

plyr simplifies modifying data in R. The following shows an example of how plyr makes the same task much simpler than using the traditional way in R. Below is an example from the plyr paper, which shows how plyr eliminates the complexity of the code and neatly uses all the data types and corresponding labels of the data in just two lines of code.

Complex code[12]

```
# SPLIT/APPLY
agg.cor<-tapply(1:nrow(orangedf), orangedf$Tree, FUN=function(x) cor(or-
angedf$age[x], orangedf$circumference[x]))
agg.cov<-tapply(1:nrow(orangedf), orangedf$Tree, FUN=function(x) cov(or-
angedf$age[x], orangedf$circumference[x]))
# Convert arrays to tables, then dfs
agg.cor<-(as.data.frame(as.table(agg.cor)))
agg.cov<-(as.data.frame(as.table(agg.cov)))
# COMBINE
chart <-merge(agg.cov, agg.cor, by=c("Var1"))
colnames(chart) <-c("TREE", "COVARIANCE", "CORRELATION")
```

Plyr code

```
library(plyr)
# Define 2 new functions to be used with ddply
ncor<-function(newdf) {chart <-cor(newdf[, 2],newdf[, 3]); return(chart)}
ncov<-function(newdf) {chart <-cov(newdf[, 2],newdf[, 3]); return(chart)}
chart <-ddply(odf, .(Tree), c("ncov", "ncor"))
colnames(chart) <-c("TREE", "COVARIANCE", "CORRELATION")
```

plyr function names are not trivial names and have a meaning. For example dlply means the functions takes a data frame as parameter, processes it and returns data in a list. ldply does the opposite, it takes a list as parameter and returns a data frame.

All together there are 12 functions in plyr package as shown below[13]:

[12] [9], page 10,13
[13] [9] , page 5

- a*ply(.data, .margins, .fun, ..., .progress = "none")
- d*ply(.data, .variables, .fun, ..., .progress = "none")
- l*ply(.data, .fun, ..., .progress = "none")
- m*ply(.data, .fun = NULL, ..., .expand = TRUE, .progress = "none", .inform = FALSE, .print = FALSE, .parallel = FALSE, .paropts = NULL)

"*" can be an a (array), l (list) or d (data frame)

Last but not least plyr functions are able to run in parallel mode. The setting `.parallel = TRUE` runs any plyr function in parallel.

	data frame	**list**	**array**	**nothing**
data frame	ddply	ldply	adply	d_ply
list	dlply	llply	alply	l_ply
array	daply	laply	aaply	a_ply

Table 1 plyr naming conventions

For plotting, the underscore (_) (a_ply, d_ply etc.) option is useful. It will process the data and then throw away the output instead of keeping it in the R environment.

3.7.2 ggplot2 Package

Developed by Hadley Wickham, ggplot2's popularity has grown in a very short time among users because of its simple layer based usage. A ggplot2 graph can be stored in a variable and additional features can be added to it just by using the (+) operator. That makes it very flexible and comfortable for the user. Below are a couple of examples created using the ggplot2 package.

Example.

```
p<-ggplot(subset(subset(alldatwithLicenseLanguage,dates%in%unique(
alldatwithLicenseLanguage$dates)[c(20,40,60,80,100,120,140)]), teamsizes%in%
c(1,5,20,50,100,200,400)),aes(x=dates, y=al1m))
p<-p + facet_wrap(~teamsizes, scales="free_y")
p<-p + geom_point(colour="red",size=2)
p<-p + geom_smooth(aes(x=dates, y=al1m), method="loess",colour="blue")
p<-p + geom_point(aes(x=dates, y=al1m), size=1.1)
show(p)
```

Creates the following graph:

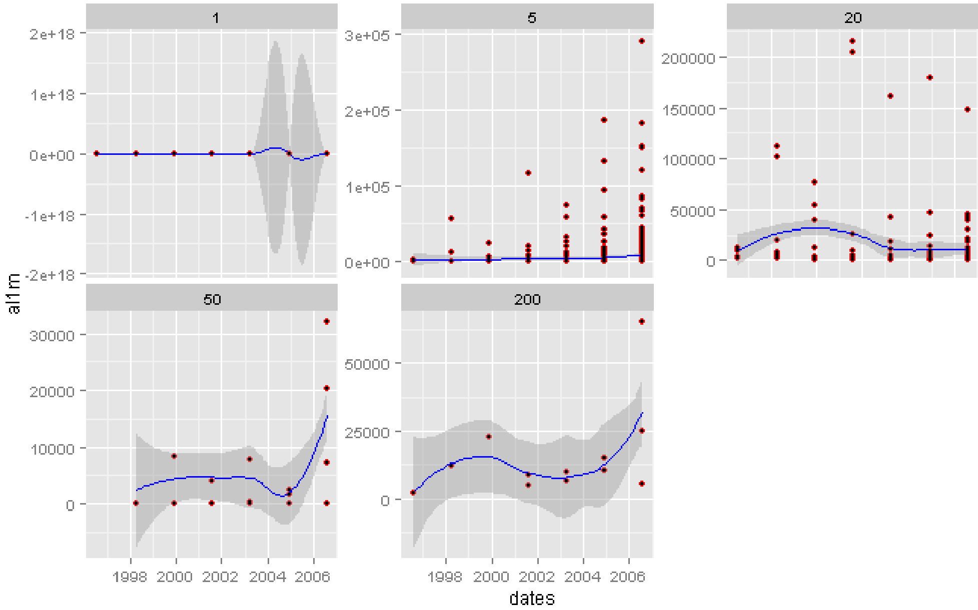


Figure 23 ggplot2 example

The data must be in data frame format in order to use it with ggplot2. Graphs are created by layers. By using layers it is possible to put different types of graphs on top of each other. Each layer may have a different data source, different type of geometric object, different mappings and different positioning.

If we wanted to add a line as in *Figure 23* that is specified by slope and intercept, then we can simply add to the previous example the following line:

```
p <- p + geom_abline()
```

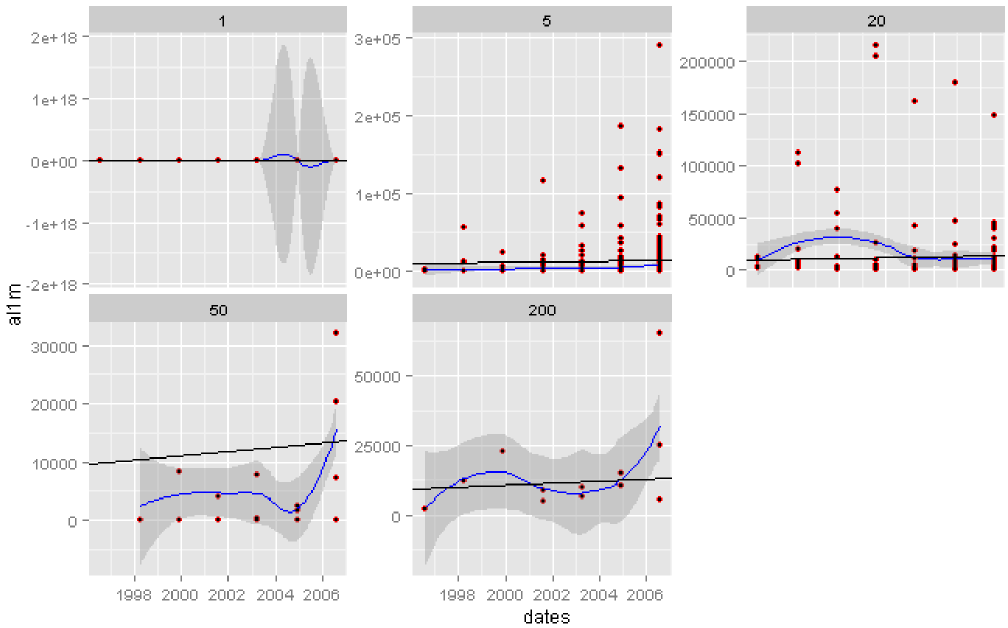


Figure 24 ggplot2 with abline

3.7.3 gfw.chart Component

gfw.chart is essentially an object oriented interpreter which takes class based parameters as instructions and generates graphs based on those parameters.

The toolkit is written in R and is object oriented. We designed the toolkit so that the user can generate graphs just by supplying parameters, without knowing the fine details of the implementation.

gfw.chart class derives from gfw.chart.base class which has two main properties; data, a data frame holding the data being processed and params, an object collection holding the parameter objects needed to process the data. Parameter classes derive from the gfw.chart.param class. There are 9 parameters:

- gfw.chart.param.dimensions: Class for a vector of objects of type gfw.chart.param.dimension.
- gfw.chart.param.dimension: Class containing the dimension and Boolean parameters for confidence/prediction intervals.
- gfw.chart.param.model: Class containing the model function and formula.
- gfw.chart.param.graphics: Class containing a list of gfw.chart.param.graph objects.
- gfw.chart.param.graph: Class containing the information of a layer in the graph.
- gfw.chart.param.aggregation: Class containing the aggregation column, aggregation row and aggregation function.
- gfw.chart.param.facet: Class containing the facet function, facet scale and facet formula.
- gfw.chart.param.exploratory: Class containing the column name for the x-axis in case no aggregation column is defined.

gfw.chart class makes use of these aforementioned parameters in order to decide internally what to do. After supplying the parameters and the data source the user calls the creategraph() method of the gfw.chart class and receives a list of graphs as a result. The user then can plot the graphs in his preferred environment like in shiny server or in R.

Calling the creategraph() method of gfw.chart class triggers a chain reaction. The data frame is prepared per dimension before the graph is even created. First the preparedata() method is called to prepare the data frame and aggregate it depending on the aggregation parameter. The gfw.chart.model.processor class is used to apply the model functions and formula to the data frame and is returned by the preparedata() method. gfw.chart.model.processor class does not return the model itself but it extracts relevant data for plotting.

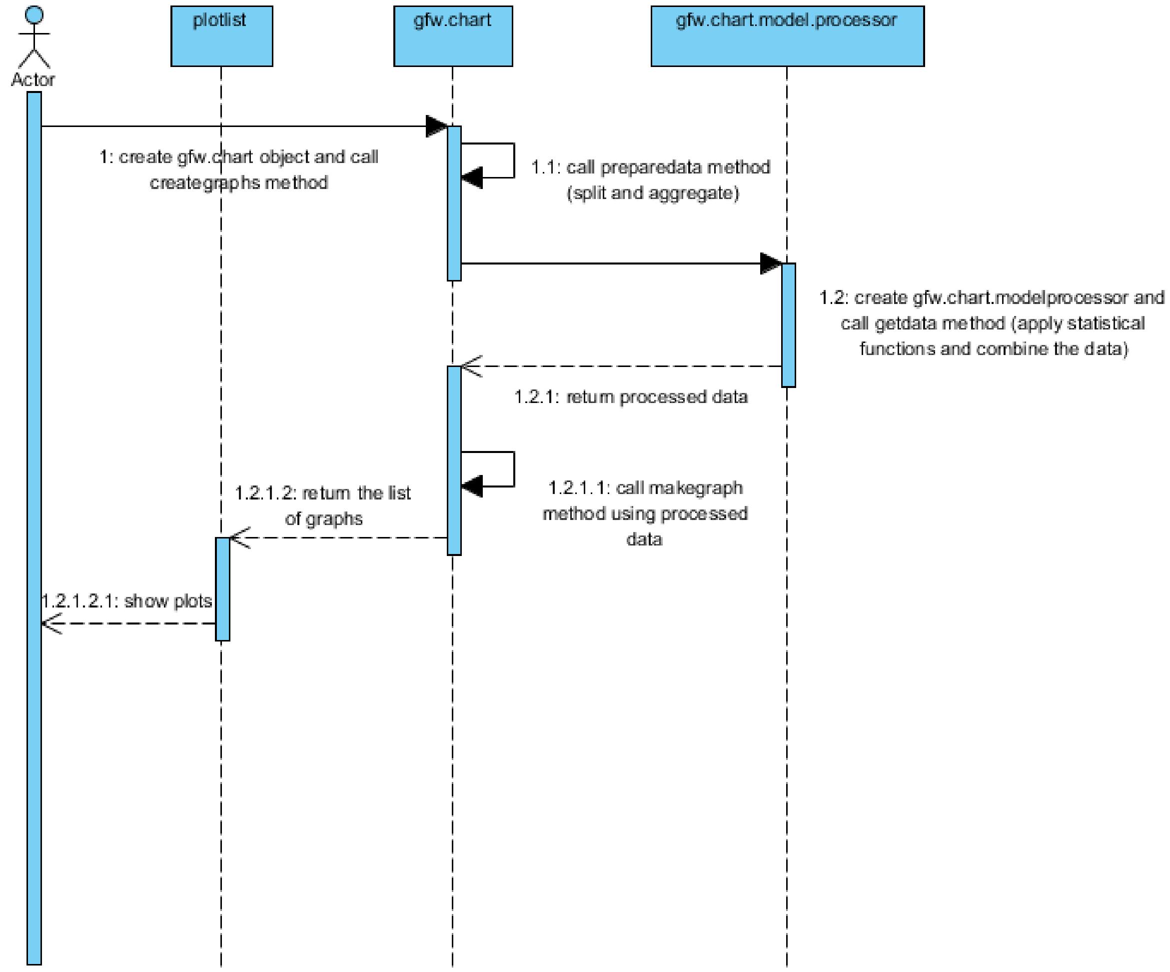


Figure 25 Toolkit sequence diagram

After these two phases of data preparation the makegraph() method is called to create the graph and add it to the "plotlist". "plotlist" is a list which holds the generated graphs and is used as the return value of the creategraph() method.

3.7.4 gfw.chart.datasplitter Component

The gfw.chart component is able to group and process the data but for ease of use it is desirable to have the ability to split the data on the fly by a group of values. For this purpose, the gfw.char.datasplitter class splits a data frame based on groups of grouping values. As an example we can use the data from the Ohloh database. In our data we have version control systems such as "CvsRepository", "SvnRepository", "HgnRepository", "GitRepository", "BzrRepoistory", SvnSyncRepository". Instead of grouping by all of the available repositories, we may want to see the data in two groups such as "vcsgroup1" composed of "CvsRepository", "SvnRepository", and "HgnRepository" and "vcsgroup2" composed of "GitRepository", "BzrRe-poistory", and SvnSyncRepository". We call the new dimension "vcstype". In this case if we generate graphs grouped by "vcstype" the toolkit will generate two plots instead of six.

Below is an example of splitting data described as above.

Example.

```
vcssplit1 <- gfw.chart.split(splitvalue = "vcsgroup1", setvalues =
c("CvsRepository", "SvnRe-pository", "HgRepository"))
vcssplit2 <- gfw.chart.split(splitvalue = "vcsgroup2", setvalues =
c("GitRepository", "BzrRe-pository", "SvnSyncRepository"))

languagesplit1 <- gfw.chart.split(splitvalue = "languagegroup1", setvalues =
c("c", "cpp"))
languagesplit2 <- gfw.chart.split(splitvalue = "languagegroup2", setvalues =
c("java", "python", "php"))

splitset1 <- gfw.chart.splitset(splitcolumn = "vcstype", datacolumn =
"vcs", splitset = list(vcssplit1, vcssplit2))
splitset2 <- gfw.chart.splitset(splitcolumn = "languagetype", datacolumn =
"language", splitset = list(languagesplit1, languagesplit2))

datasplitter <- gfw.chart.datasplitter(splitsets = list(splitset1, splitset2))
df <- datasplitter$getSplitData(alldatwithLicenseLanguage)
```

gfw.chart.datasplitter uses parameters which contain the instructions concerning what it has to do. These parameters are as follows:

- gfw.chart.param.split

  required to define the new column and values to be grouped.

- gfw.chart.param.splitset

  list of splits of type gfw.chart.param.split

3.7.5 gfw.chart.testdata Component

The toolkit has also a data generator class called gfw.chart.testdata. Similar to gfw.chart class parameters, it also uses parameters to generate test data in data frame format. The parameters are composed of a list of options which are as follows:

- "dimensions": List of default columns
- "columns": List of columns which are going to be used to name and populate the column with data.
    - "columnindex": Index of the column in the dimensions list.
- "columntype":Type of the column such as "date", "character".

- “columnname”: Name of the column. Used to rename the column in the dimensions list using “columnindex”.
- “columnfactor”: Boolean value to decide whether the column is a factor or not.
- “values”: list of values to fill the column with.
- “filldata”: fill the corresponding column with “columnindex” in “fillwith” list with data as per the values in the values list.
- “numberofrows”:total number of rows of the new data frame
- “saveas”: optional parameter to save the data frame on hard drive for later use.

The following code shows an example usage of gfw.testdata class:

Example.

```
source("gfw.chart.testdata.R")
paramsSSlogis = list(dimensions = as.list(paste("var", 1:10, sep = "")), columns =
list(list(columnindex = 1, columntype = "date",
                              columnname = "time", columnfactor = F, values =
as.Date(c("2000-01-01"))), list(columnindex = 2,
                              columntype = "character", columnname = "license",
                              columnfactor = T, values = c("academic", "gpl"),
filldata = list(columnindex = 2,
                         fillwith = list(expression(100/(2 + exp((12500
-as.numeric(df$time))/200)) + rnorm(length(df$al1m))),
                                   expression(1/(1002 + exp((3 -
as.numeric(df$time))/4)) + rnorm(length(df$al1m)))))),
                         list(columnindex = 3, columntype = "integer", columnname
= "al1m", columnfactor = F,
                                   values = c(1000:2000))), numberofrows = 100,
          saveas = "df.dat")
generator <- gfw.chart.testdata(params = paramsSSlogis)
df <- generator$getdataframe()
```

The example above generates the following data frame

	time	al1m	license
1	2000-01-01	-0.73541245	academi
2	2000-01-01	0.23681247	gpl
3	2000-02-01	1.06715588	academ
4	2000-02-01	0.01429561	gpl
5	2000-03-01	-0.01934306	academic
6	2000-03-01	0.39079474	gpl
7	2000-04-01	-1.42904095	academic
8	2000-04-01	-0.49644631	gpl
9	2000-05-01	1.87951577	academi
10	2000-05-01	-0.72839644	gpl
11	2000-06-01	0.67255619	academic
12	2000-06-01	-0.40374365	gpl
13	2000-07-01	-0.06626166	academi
14	2000-07-01	0.70575887	gpl
15	2000-08-01	-0.12477639	academi
16	2000-08-01	0.32860457	gpl
17	2000-09-01	1.02139033	academic
18	2000-09-01	-2.45198350	gpl

Figure 26 data frame containing data generated by gfw.chart.testdata.generator class

If we create a graph using the test data it looks like as follows:

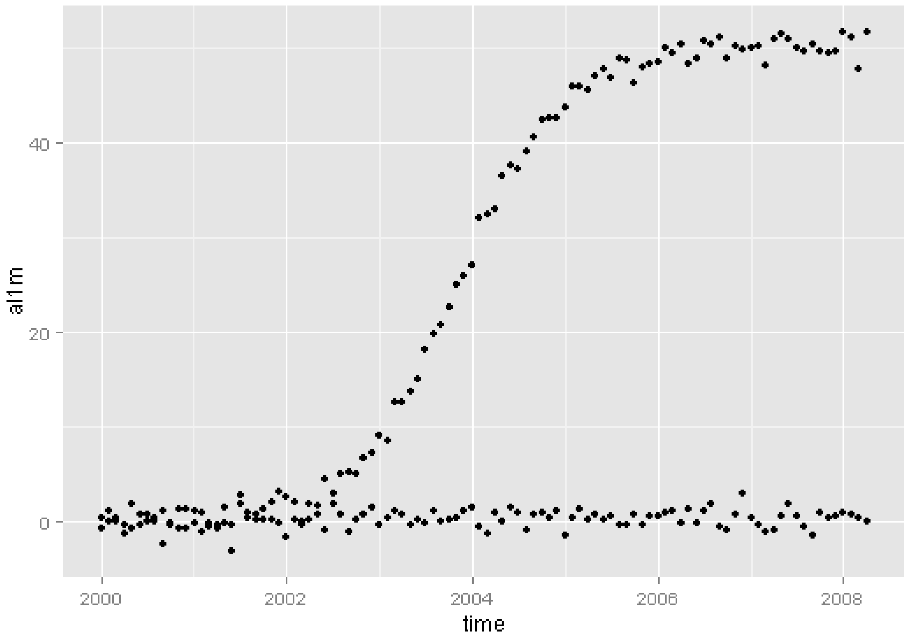


Figure 27 Graph created using test data

3.7.6 Example Usage of the Toolkit: Shiny Integration

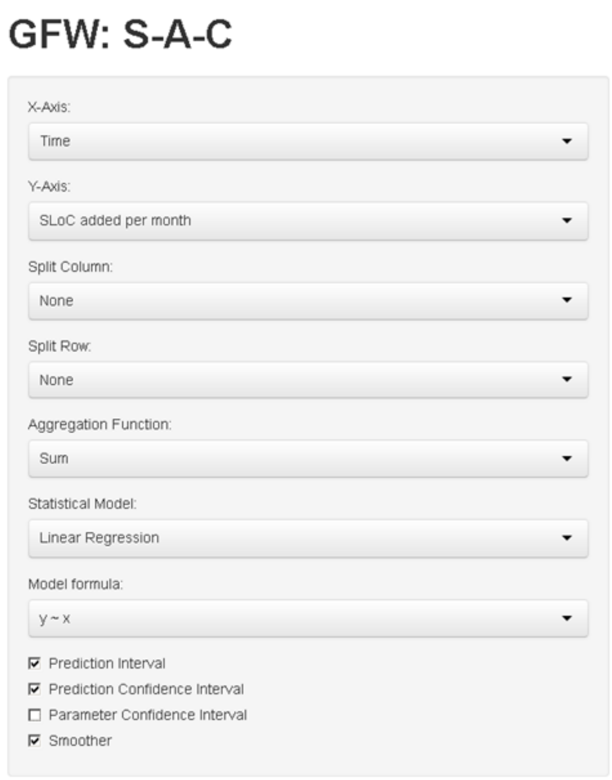


Figure 28 Options for figure 29

The toolkit is designed to be used generically and independently of data sources. To demonstrate this, we developed a shiny application [14] that uses the toolkit to process and visualize the data in a web environment.

Shiny is an R package that allows us to develop web based R applications and makes user interaction simple over the web. We will not go into the details of the shiny package since it is beyond the scope of this work.

The application sources can be found in the appendices. In the following pages we display a couple of screenshots from the application showing different features and use cases of the toolkit.

Figure 28 shows a linear regression graph of the sum of added lines of code between 2001 and 2003 over all projects. The Shiny user interface allows the user to select various options such prediction interval, confidence interval and parameter confidence interval.

If the data has multiple columns they can simply be selected as x- or y-axis and the graph will be shown using the selected options. Even if the database structure or the dimensions change there is no need to change the toolkit itself but just the parameters.

[14] The application was developed by Gottfried Hofmann.

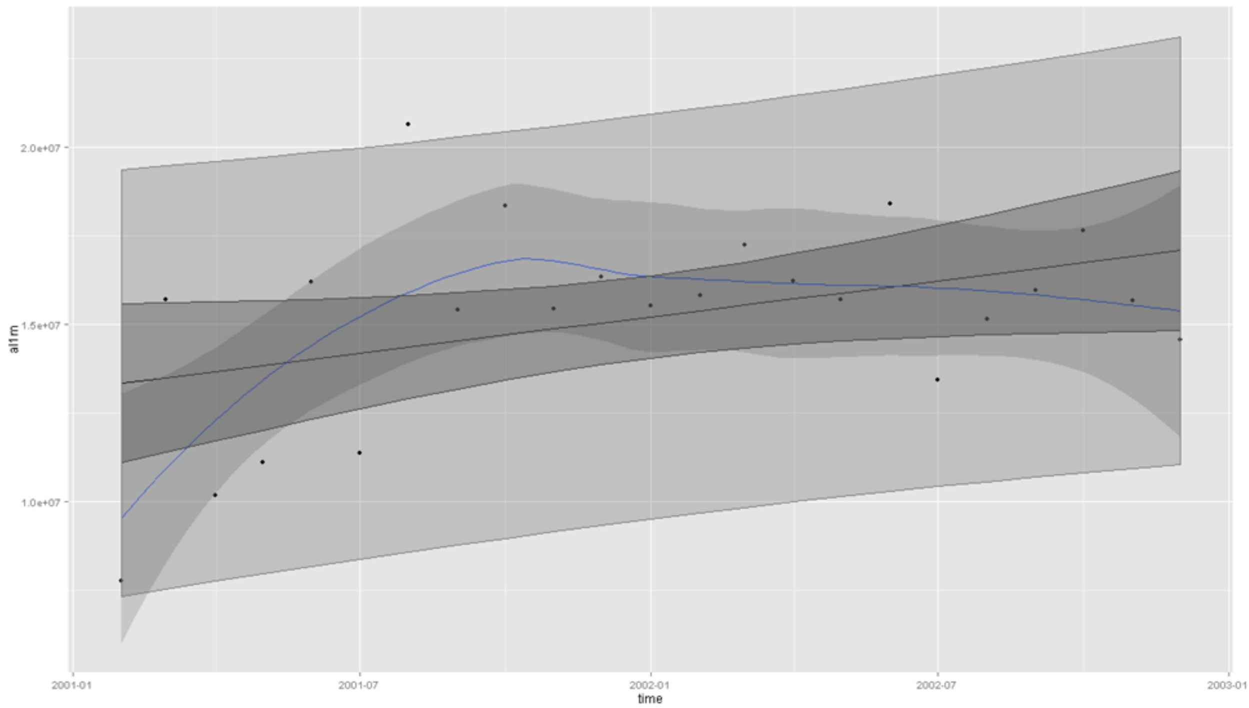


Figure 29 Shiny application using gfw.chart toolkit, linear regression model showing monthly added lines of code with prediction interval and confidence interval and no splitting.

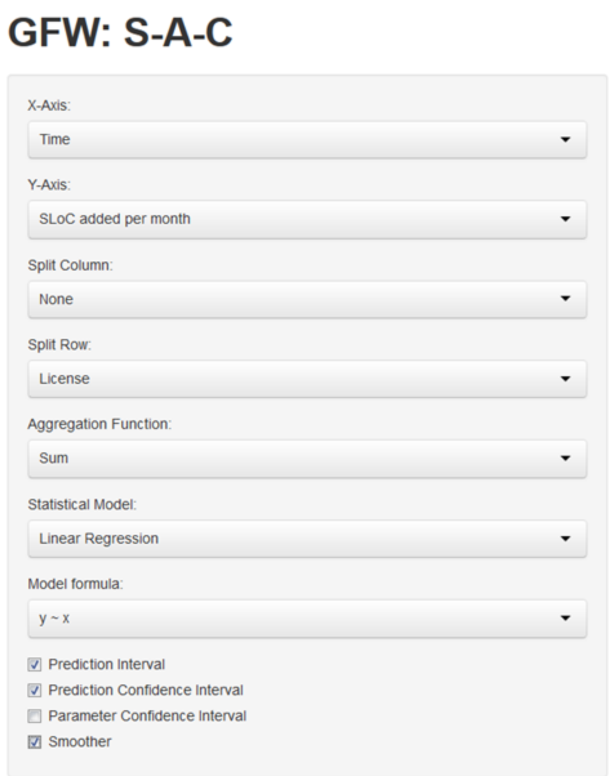


Figure 30 Options for figure 31

The data can be split by dimensions if there is factorial data in one or more of the columns. This way comparison is made easy.

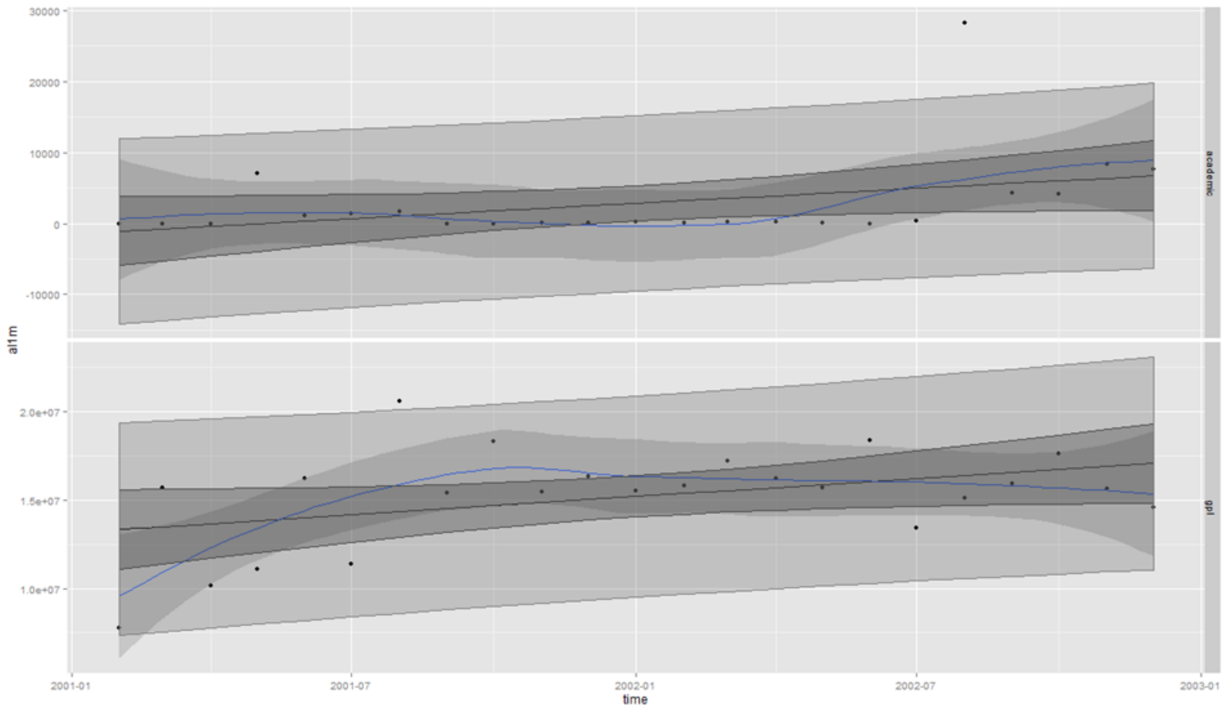
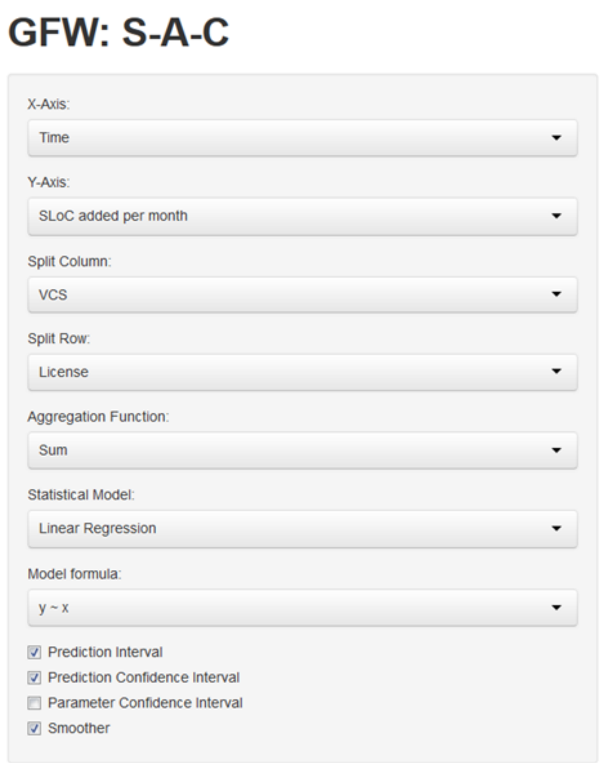


Figure 31 Shiny application using gfw.chart toolkit, linear regression model showing monthly added lines of code with prediction interval and confidence interval, split by licenses.



The shiny demo application uses the facet options provided by ggplot2 which allow for splitting of at most two dimensions. Internally the toolkit provides a plotting function for an arbitrary number of dimensions.

Figure 32 Options for figure 33

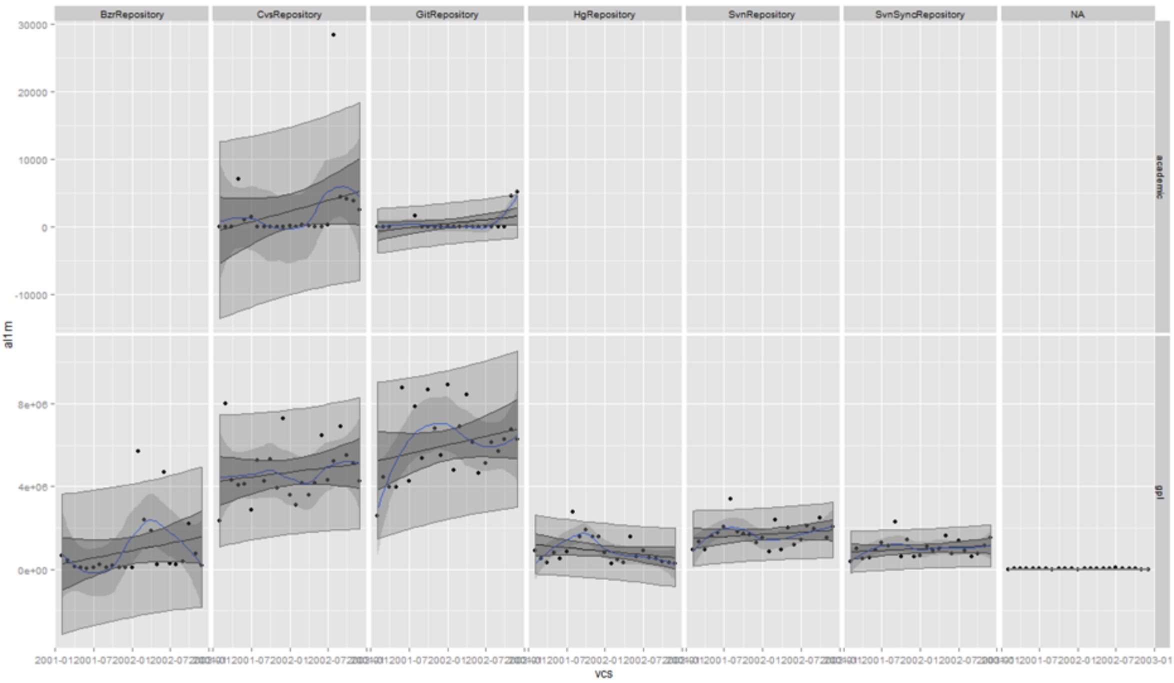


Figure 33 Shiny application using gfw.chart toolkit, linear regression model showing monthly added lines of code split by licenses and version control systems.

No regression curve is shown for "academic" because no model could be fit. Confidence intervals are selected but not shown because they could not be computed, either. Errors are being omitted in the results.

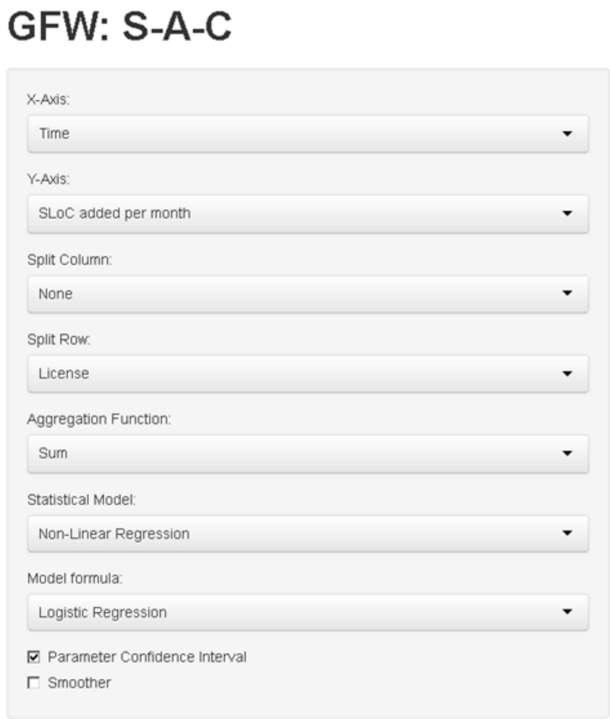


Figure 34 Options for figure 35

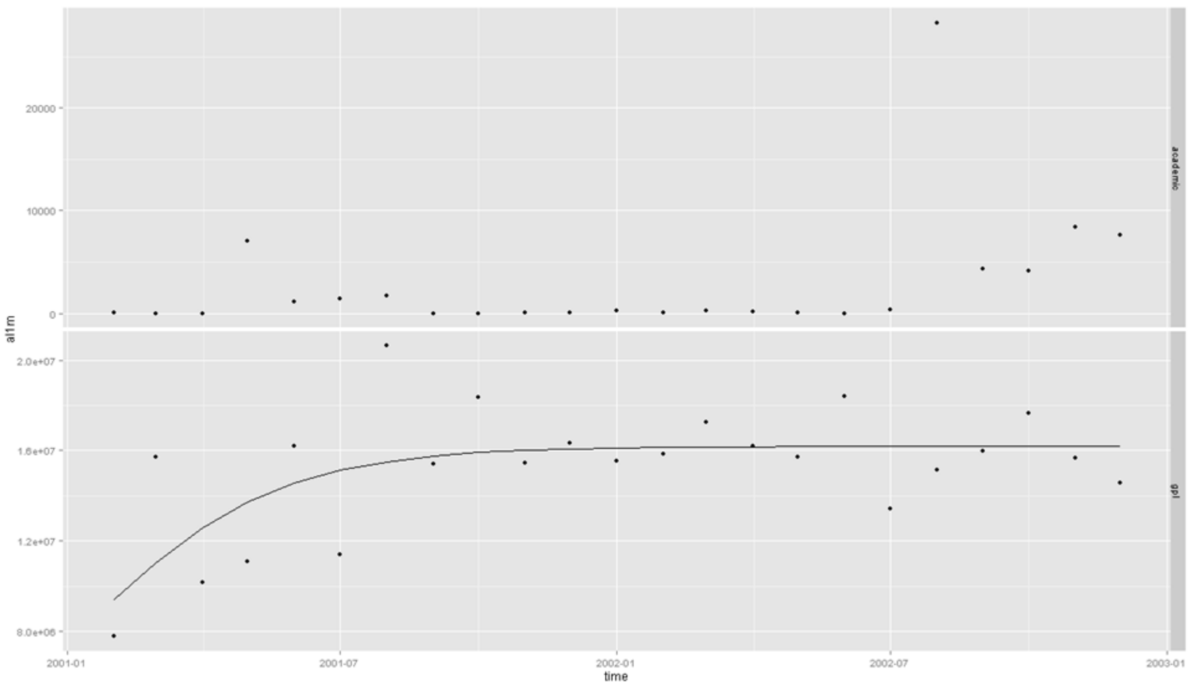


Figure 35 Shiny application using gfw.chart toolkit, logistic regression model showing monthly added lines of code split by licenses

X- and Y-axis can be set arbitrarily. In this example added lines of code are plotted over removed lines of code split by license type.

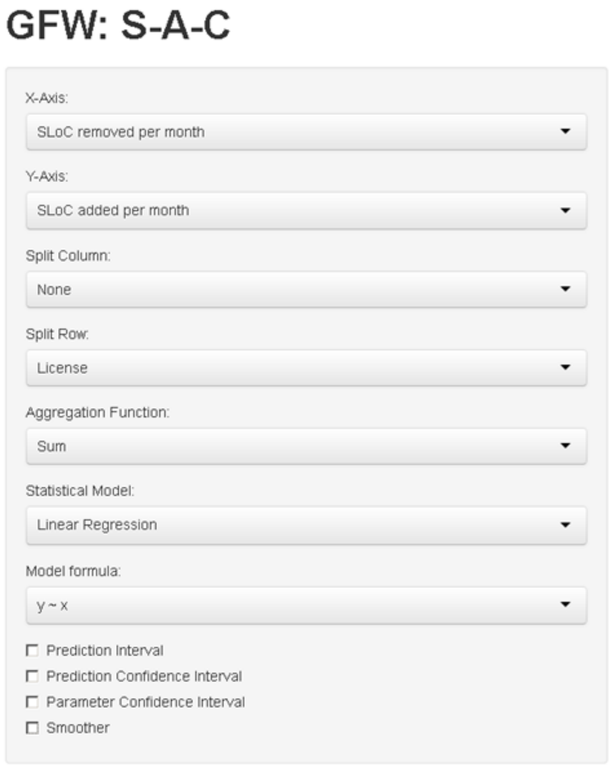


Figure 36 Options for figure 37

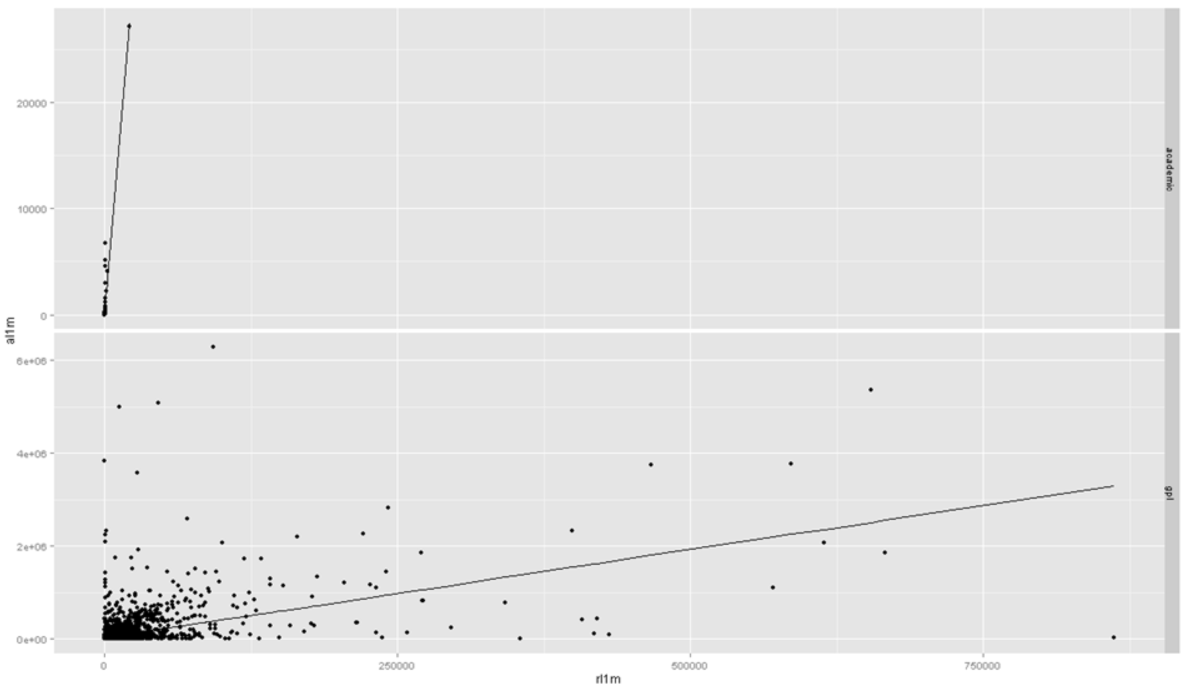


Figure 37 Added lines of code over removed lines of code split by licenses with linear regression line. X-axis and y-axis can be changed at will.

3.8 Traditional R scripting vs Toolkit usage

For the demonstration we used a subset of the Ohloh data snapshot. The structure of the data is as follows:

- dates — month based date of when the project was recorded
- teamsizes — team size measured as the number of committers who have carried out a commit since the Project start to the snapshot date.
- projectid — Project id number
- al1m — Number of added lines of code in the last month.
- rl1m — Number of removed lines of code in the last month.
- commitsl1m — Number of committed lines of code in the last month.
- activl1m — Number of committers who have performed at least one commit in the last month
- license — Project license
- language — Project language
- vcs — Version control system
- name — Project name

The data contains data for a subset of projects between 1995 and 2010. The list of projects selected in the data is as follows:

- Apache HTTP Server
- OpenSSL
- U-Boot
- Linux Kernel
- Mozilla Firefox
- QEMU
- JBoss Tools
- Git
- jQuery
- Samba
- Eclipse Linux Tools
- Project bootstrap

- JDepend plugin for Eclipse
- Filesync plugin for Eclipse

Using the shiny application in conjunction with the toolkit the following options generate the graph in Figure 39.

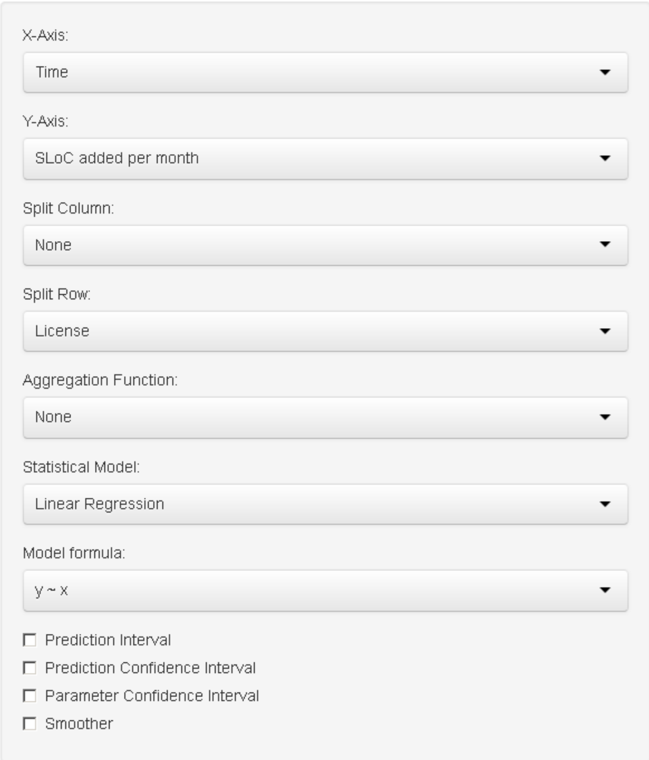


Figure 38 Options for figure 39

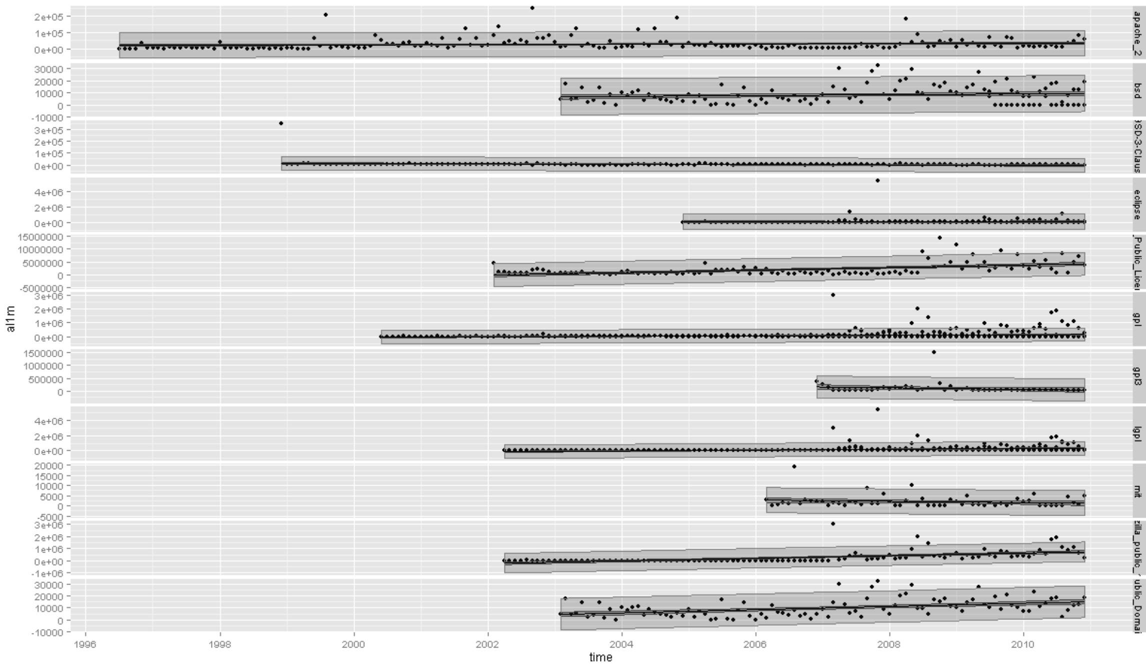


Figure 39 monthly total added lines of code per version control systems

The example above shows the linear regression of version control systems over added lines of code between 1995 and 2010.

If we want to create this graph using traditional R scripting then we can create a script as follows:

```
library(plyr)
library(ggplot2)
load("databyprojects19952010.Rdata") # load the subsetdat
dat<- subsetdat
selectcolumns <- c()
selectcolumns <- c('al1m','license','time')

## Make a copy of data frame
data <- dat[, selectcolumns]

modeldata <- ddply(data,.(license,time),.parallel=FALSE, function(df)
sum(df$al1m))

names(modeldata)[which(colnames(modeldata) == "V1")] <- c(paste("al1m", sep = ""))

resultdf <- ddply(modeldata,.(license),function(df){
               model<-lm(al1m~time,data=df)
               df$fitted<-fitted(model)
               predicted<-predict(model, interval = "prediction")
               df$plwr<-predicted[,c("lwr")]
               df$pupr<-predicted[,c("upr")]

               confidence<-predict(model,interval = "confidence")
               df$pclwr<-confidence[,c("lwr")]
               df$pcupr<-confidence[,c("upr")]

               df$clwr<-confint(model)[[1]] + as.numeric(df$time)*con-
fint(model)[[2]]
               df$cupr<-confint(model)[[3]] + as.numeric(df$time)*con-
fint(model)[[4]]
               return(df)
          })
p<-ggplot(data=resultdf,aes(x=time,y=al1m))+geom_point(aes(x = time, y =
al1m))+facet_grid(license ~.,scales="free_y")
if (("pclwr" %in% colnames(resultdf)) && ("pcupr" %in% colnames(resultdf))) {
     p <- p + geom_ribbon(aes(x = time, ymin = pclwr, ymax = pcupr), colour =
"gray30", alpha = 0.3)
}
if (("clwr" %in% colnames(resultdf)) && ("cupr" %in% colnames(resultdf))) {
     p <- p + geom_ribbon(aes(x = time, ymin = clwr, ymax = cupr), colour =
"gray20", alpha = 0.2)
}
if (("plwr" %in% colnames(resultdf)) && ("pupr" %in% colnames(resultdf))) {
     p <- p + geom_ribbon(aes(x = time, ymin = plwr, ymax = pupr), colour =
"gray50", alpha = 0.2)
}
if (("fitted" %in% colnames(resultdf))) {
     p <- p + geom_line(aes(x = time, y = fitted))
}
# Show graph
show(p)
```

Example Code 1, Script creating graph for added lines of code per month split by language

The code above generates the following graph:

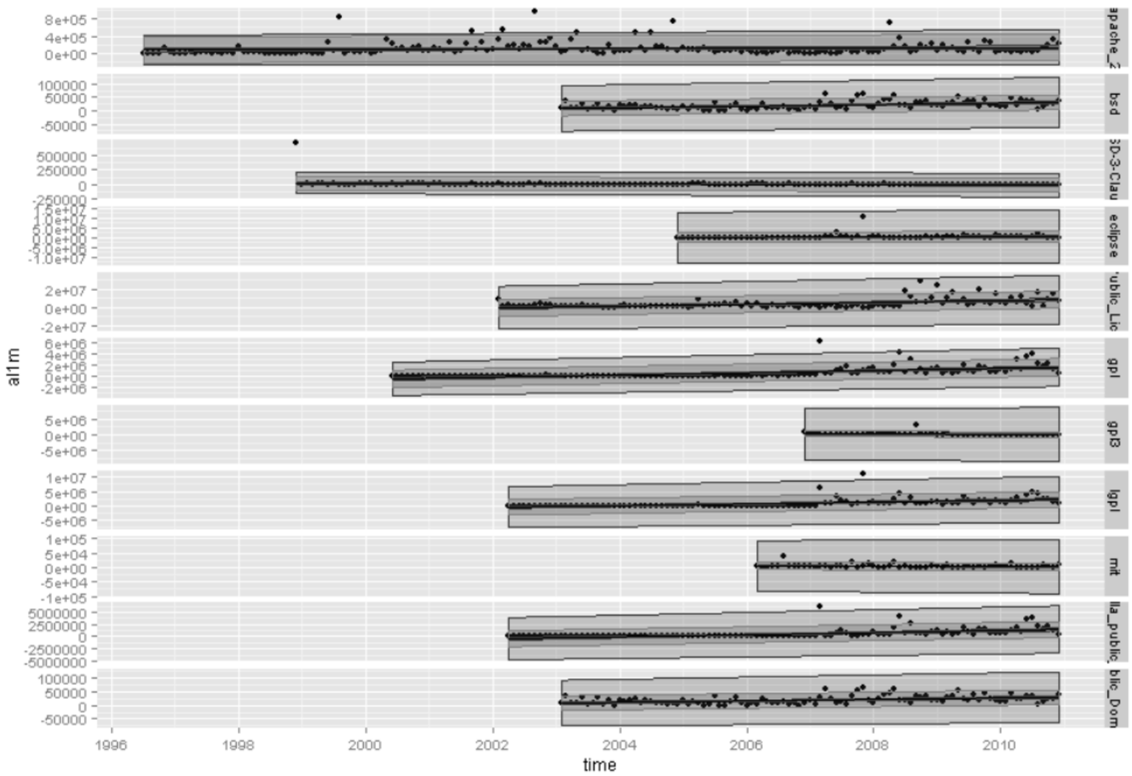


Figure 40 monthly total added lines of code per version control systems (traditional)

Suppoes we wanted to show two groups of grouped licenses as follows:

- Licensegroup1
    - apache_2
    - BSD-3-Clause
    - bsd
    - gpl
    - eclipse
- Licensegroup2
    - gpl
    - GNU_General_Public_License_v2_0_only
    - lgpl
    - mit
    - gpl3
    - mozilla_public_1_1
    - Public_Domain

In case of using the toolkit, only three lines of code have to be added:

Example.
```
licensesplit1<-gfw.chart.param.split(splitvalue="licensegroup1",
setvalues=c('apache_2','BSD-3-Clause','bsd','gpl','eclipse'))
licensesplit2<-gfw.chart.param.split(splitvalue="licensegroup2",
setvalues=c('gpl','GNU_General_Public_License_v2_0_on-
ly','lgpl','mit','gpl3','mozilla_public_1_1','Public_Domain'))
splitset3<-gfw.chart.param.splitset(splitcolumn="licensetype",

datacolumn="license",splitset=list(licensesplit1,licensesplit2))
```

When the UI is used, one additional line of code would be needed to make the new set available from the UI:

```
"License Type" =  c("licensetype"),
```

Note that the process of adding new sets is streamlined and simple. Now the user can select the option and generate the graph. The result can be seen as follows:

Figure 41 Options for figure 42

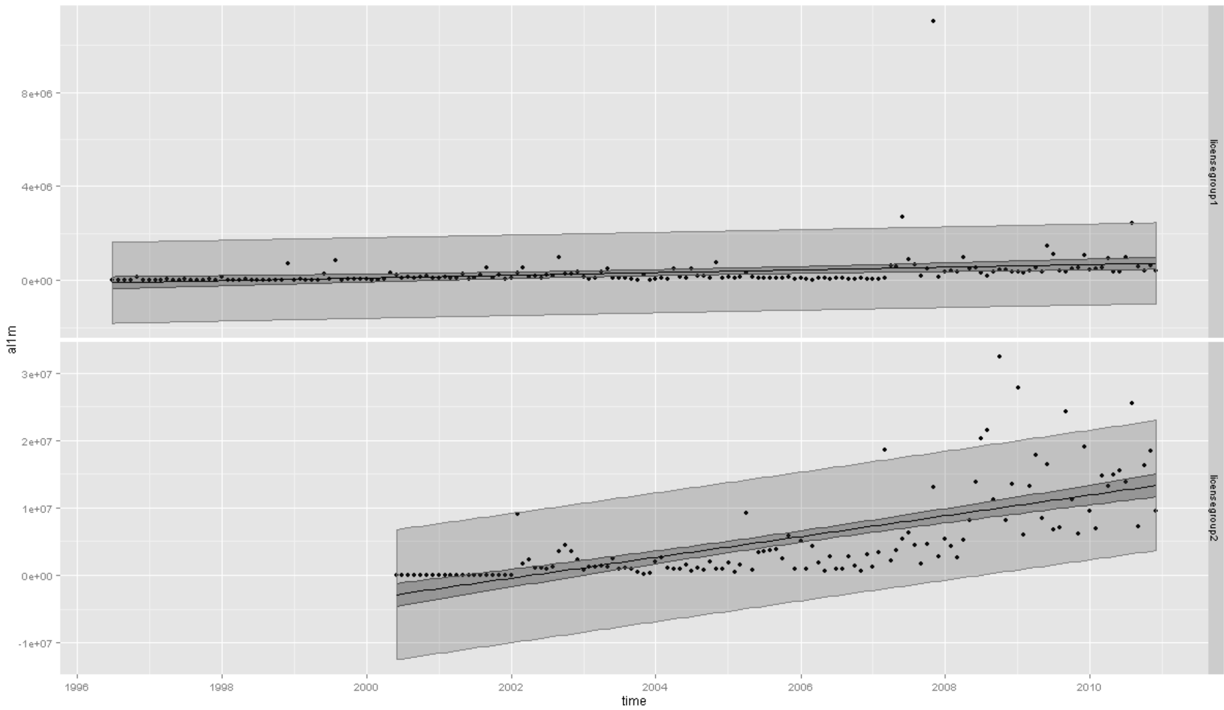


Figure 42 total monthly added lines of code per licensetypes

In case of adding the changes to the custom script, the code had to be changed as follows, the changes are marked in yellow background:

```
library(plyr)
library(ggplot2)
load("databyprojects19952010.Rdata")
dat<- subsetdat
selectcolumns <- c()
selectcolumns <- c('al1m','license','time')

## Make a copy of data frame
data <- dat[, selectcolumns]

#Create a new column for splitting and assign split values
setvalues<-c('apache_2','BSD-3-Clause','bsd','gpl','eclipse')
for(j in 1:length(setvalues))
{
     setvalue<-setvalues[j]
     #print(setvalue)
     data[which(data['license']==setvalue),'licensetype']<-'licensegroup1'
}
setvalues2<-c('gpl','GNU_General_Public_License_v2_0_on-
ly','lgpl','mit','gpl3','mozilla_public_1_1','Public_Domain')
for(j in 1:length(setvalues2))
{
     setvalue<-setvalues2[j]
     #print(setvalue)
     data[which(data['license']==setvalue),'licensetype']<-'licensegroup2'
}
selectcolumns <- c()
selectcolumns <- c('al1m','licensetype','time')
```

```
modeldata <- ddply(data,.(licensetype,time),.parallel=FALSE, function(df)
sum(df$al1m))
names(modeldata)[which(colnames(modeldata) == "V1")] <- c(paste("al1m", sep = ""))
resultdf <- ddply(modeldata,.(licensetype),function(df){
                model<-lm(al1m~time,data=df)
                df$fitted<-fitted(model)
                predicted<-predict(model, interval = "prediction")
                df$plwr<-predicted[,c("lwr")]
                df$pupr<-predicted[,c("upr")]

                confidence<-predict(model,interval = "confidence")
                df$pclwr<-confidence[,c("lwr")]
                df$pcupr<-confidence[,c("upr")]

                df$clwr<-confint(model)[[1]] + as.numeric(df$time)*con-
fint(model)[[2]]
                df$cupr<-confint(model)[[3]] + as.numeric(df$time)*con-
fint(model)[[4]]
                return(df)
          })
p<-ggplot(data=resultdf,aes(x=time,y=al1m))+geom_point(aes(x = time, y =
al1m))+facet_grid(licensetype ~.,scales="free_y")
if (("pclwr" %in% colnames(resultdf)) && ("pcupr" %in% colnames(resultdf))) {
     p <- p + geom_ribbon(aes(x = time, ymin = pclwr, ymax = pcupr), colour =
"gray30", alpha = 0.3)
}
if (("clwr" %in% colnames(resultdf)) && ("cupr" %in% colnames(resultdf))) {
     p <- p + geom_ribbon(aes(x = time, ymin = clwr, ymax = cupr), colour =
"gray20", alpha = 0.2)
}
if (("plwr" %in% colnames(resultdf)) && ("pupr" %in% colnames(resultdf))) {
     p <- p + geom_ribbon(aes(x = time, ymin = plwr, ymax = pupr), colour =
"gray50", alpha = 0.2)
}
if (("fitted" %in% colnames(resultdf))) {
     p <- p + geom_line(aes(x = time, y = fitted))
}
# Show graph
show(p)
```

Example Code 2, Script creating graph for monthly total added lines of code per licensetype

The code above generates the same graph as the previous one from the toolkit

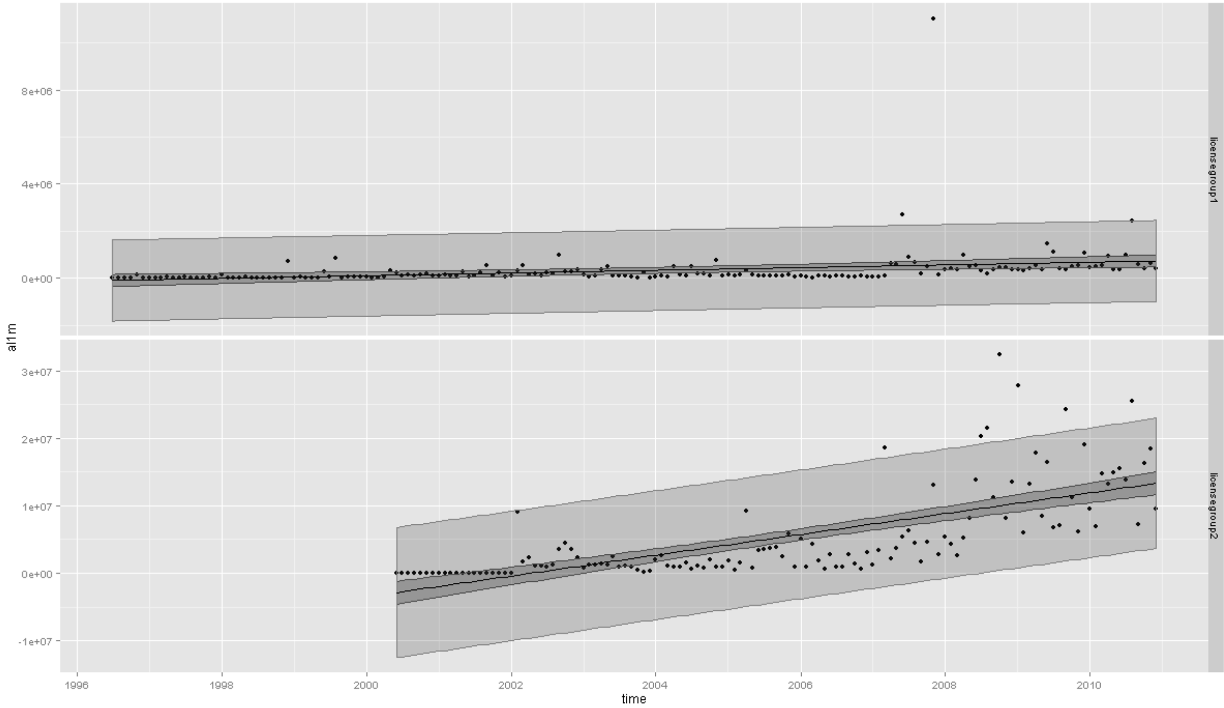


Figure 43 total monthly added lines of code per licensetypes

If we would want to create a graph for number of active committers per month split by language type and run a non-linear regression with logistic model formula, then the following figure shows the settings for the toolkit:

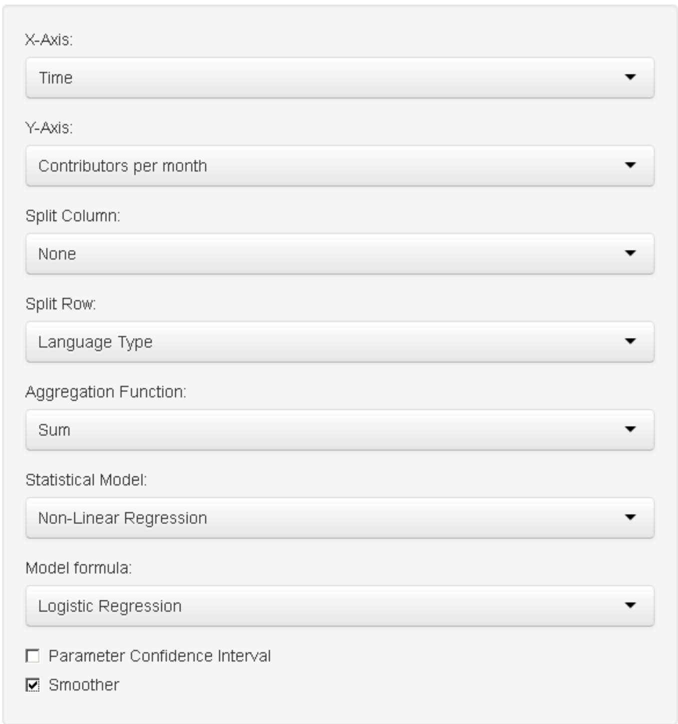


Figure 44 Options for figure 45

Which generates the following graph:

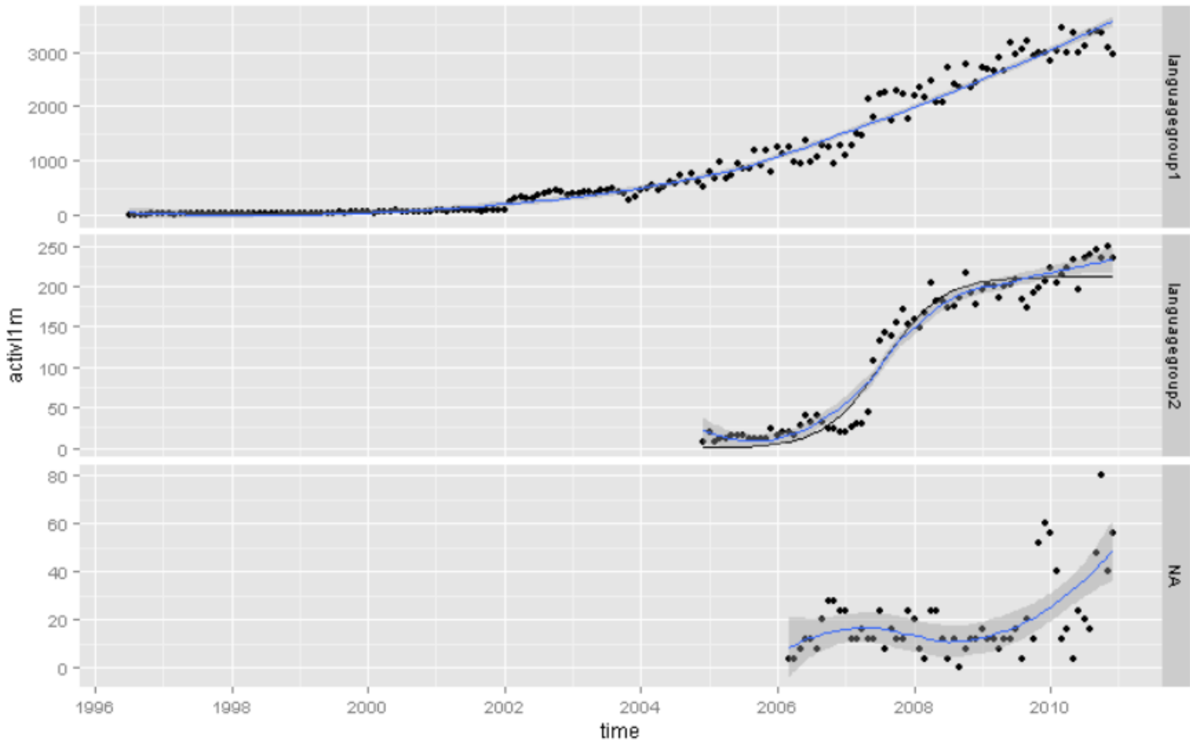


Figure 45 Non-linear regression with logistic formula on number of active committers per month split by language type.

To generate the graph from the example above by scripting it in R, using example code 1 requires many changes. This would happen every time if you want to use different statistical methods or change the split sets:

```
dat<- subsetdat

setvalues<-c("c", "cpp")
for(j in 1:length(setvalues))
{
    setvalue<-setvalues[j]
    #print(setvalue)
    dat[which(dat['language']==setvalue),'languagetype']<-'languagegroup1'
}

setvalues2<-c("java", "python", "php")
for(j in 1:length(setvalues2))
{
    setvalue<-setvalues2[j]
    #print(setvalue)
    dat[which(dat['language']==setvalue),'languagetype']<-'languagegroup2'
}
```

```
selectcolumns <- c()
selectcolumns <- c('activl1m','languagetype','time')

## Make a copy of data frame
data <- dat[, selectcolumns]

modeldata <- ddply(data,.(languagetype,time), function(df) sum(df$activl1m))

names(modeldata)[which(colnames(modeldata) == "V1")] <- c(paste("activl1m", sep =
""))
resultdf <- ddply(modeldata,.(languagetype),function(df){
                df$time<-as.numeric(df$time)
                    try(model<-nls(activl1m~SSlogis(time, Asym, xmid,
              scal),data=df))
                try(df$fitted<-fitted(model))
                    df$time<-as.Date(df$time)
                return(df)
          })
p<-ggplot(data=resultdf,aes(x=time,y=activl1m))+geom_point(aes(x = time,
y = activl1m))+facet_grid(languagetype ~.,scales="free_y")

if (("fitted" %in% colnames(resultdf))) {
     p <- p + geom_line(aes(x = time, y = fitted))
}
p<-p+geom_smooth()
show(p)
```

Example Code 3, Script creating graph for monthly total active number of contributors per languagetype

The examples above show that even simple tasks that require only the setting of a few parameters in the toolkit require a lot of work when scripting them by hand. Changing parameters in hand-written scripts is error-prone. When using the toolkit parameters are changed in one place only.

Another advantage of the toolkit is that if it is being used in an application like a shiny UI the parameters can be interactively changed and the result will be shown immediately.

When working with the traditional R-approach, a lot of code duplication is happening as well while the toolkit automates repetitious parts.

When using new statistical functions, they have to be added to the toolkit only once so when they are needed a second time the code does not need to be written again. Also note that some statistical functions in R require subtle changes in the data structures which is a source of error when scripting the traditional way[15]. By setting up the rules for handling the data structures for a specific function in the toolkit the user has to take care of this only once.

For the exploratory part of the research process the toolkit in combination with an application like a shiny UI simplifies the work of the researcher.

[15] In the example code 3 the data column nls() is working on is coerced to numeric which is a requirement of nls() but not of lm().

3.9 Conclusion and Future Plans

The current implementation of the toolkit uses the plyr package for preparing data and a considerable drawback is that it requires data to be loaded into the R environment before it can be processed. In case of a big amount of data it will be better to use the dplyr package [19] developed by Hadley Wickham.

The dplyr package, built on top of the plyr package, allows for the processing of large amounts of data by fast and efficient filtering, selecting, reconstructing, and aggregating. It doesn't load the entire data first from the data source before processing it but rather it builds a query before querying the data source and loads only the required data.

Currently the framework is computing the statistics but the results are only used for visualization purposes. One might need to use more information from the model such as goodness-of-Fit, p-values etc. For this the gfw.chart.model.processor class could be modified to return a model object instead of data extracted from the model.

4 List of References

[1] http://stat.ethz.ch/R-manual/R-patched/library/stats/html/lm.html

[2] http://stat.ethz.ch/R-manual/R-patched/library/stats/html/glm.html

[3] http://stat.ethz.ch/R-manual/R-patched/library/stats/html/nls.html

[4] http://stat.ethz.ch/R-manual/R-patched/library/stats/html/arima.html

[5] Deshpande, Amit; Riehle, Dirk (2008): The Total Growth of Open Source. In Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi (Eds.): Open Source Development, Communities and Quality, vol. 275. Boston, MA: Springer US (IFIP – The International Federation for Information Processing), pp. 197–209.

[6] Wächst Freie Software exponentiell? http://keimform.de/2008/waechst-freie-software-exponentiell/

[7] The Open Source Big Bang. http://dirkriehle.com/2011/06/21/the-open-source-big-bang/

[8] Gottfried Hofmann, Dirk Riehle, Carsten Kolassa, Wolfgang Mauerer. "A Dual Model of Open Source License Growth." In Proceedings of the 9th International Conference on Open Source Systems (OSS 2013). Springer Verlag, 2013.

[9] Hadley Wickham (2011). The Split-Apply-Combine Strategy for Data Analysis. Journal of Statistical Software, 40(1), 1-29. URL http://www.jstatsoft.org/v40/i01/.

[10] Hadley Wickham (2007). Reshaping Data with the reshape Package. Journal of Statistical Software, 21(12), 1-20. URL http://www.jstatsoft.org/v21/i12/.

[11] http://cran.r-project.org/web/packages/doBy/index.html

[12] G. Grothendieck (2014): sqldf: Perform SQL Selects on R Data Frames. Available online at http://CRAN.R-project.org/package=sqldf.

[13] Wickham, Hadley (2009): ggplot2. Elegant graphics for data analysis. Dordrecht [u.a.]: Springer (Use R!).

[14] Vaidyanathan,R. (2014), rCharts: Interactive Charts using Polycharts.js, R package version 0.4.2, github.com/ramnathv/rCharts.

[15] Markus Gesmann and Diego de Castillo. Using the Google Visualisation API with R. The R Journal, 3(2):40-44, December 2011.

[16] "Coche" script http://github.com/GottfriedHofmann/coche

[17] Ohloh, Inc. See http://www.ohloh.net.

[18] Ohloh, Inc. Ohloh API. See http://www.ohloh.net/api.

[19] Hadley Wickham and Romain Francois (2014): dplyr: dplyr: a grammar of data manipulation. Available online at http://CRAN.R-project.org/package=dplyr.

[20] Arafat, O., & Riehle, D. (2009, January). The commit size distribution of open source software. In System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on (pp. 1-8). IEEE.

[21] John Verzani, "simpleR", in PDF, http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf

[22] Thomas Lumley, "R Fundamentals and Programming Techniques",http://faculty.washington.edu/tlumley/Rcourse/R-fundamentals.pdf

[23] Google R Style Guide, http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html

[24] An Introduction to R, http://cran.r-project.org/doc/manuals/R-intro.html

[25] Data Manipulation with R, Phil Spector, Springer

[26] Software for Data Analysis, John Chambers, Springer

[27] R Cookbook, Paul Teetor,O'Reilly

[28] R Graphics Cookbook, Winston Chang, O'Reilly

[29] Learning R A Step-by-Step Function Guide to Data Analysis, Richard Cotton, O'Reilly

[30] R in a Nutshell, Joseph Adler, O'Reilly

[31] The Art of R Programming, Norman Matloff, O'Reilly

[32] Fowler, M. (2004). Inversion of control containers and the dependency injection pattern.

[33] Martin, R. C. (2000). Design principles and design patterns. *Object Mentor*, 1-34.

[34] Colyer, A., Rashid, A., & Blair, G. (2004). On the separation of concerns in program families. Technical report, Computing Department, Lancaster University.

[35] G. Gorjanc. Working with unknown values: the gdata package. R News, 7(1):24–26, 2007. URL http://CRAN.R-project.org/doc/Rnews/Rnews_2007-1.pdf.

[36] Beeley, Chris (2013): Web Application with R using Shiny. Birmingham: Packt Publishing.

[37] Self-Starting Nls Gompertz Growth Mode. http://stat.ethz.ch/R-manual/R-patched/library/stats/html/SSgompertz.html

5 Appendixes

1. Normalize.sql script for normalizing main_language_id field in activitiy_facts table
2. Ohlohdata.sql script for creating ohlohdata table
3. Toolkit Documentation

5.1 normalize.sql

-- Function: normalize()

```
-- DROP FUNCTION normalize();
CREATE OR REPLACE FUNCTION normalize()
  RETURNS void AS
$BODY$
DECLARE
    r RECORD;
BEGIN
    FOR r IN select a.*,f.id afid from analysis a, activity_facts f where
a.id=f.analysis_id LOOP
        update activity_facts
set main_language_id=r.main_language_id
where id=r.afid;
    END LOOP;
    RETURN;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION normalize()
 OWNER TO asitti;
```

5.2 ohlohdata.sql

-- Function: ohlohdata(date, date)

```
-- DROP FUNCTION ohlohdata(date, date);
CREATE OR REPLACE FUNCTION ohlohdata(startdate date, enddate date)
  RETURNS integer AS
$BODY$

DECLARE retval integer;
DECLARE row RECORD;
DECLARE dates text[][];
DECLARE startyear integer;
DECLARE endyear integer;
DECLARE startmonth integer;
DECLARE endmonth integer;
DECLARE startday integer;
```

```
DECLARE endday integer;
DECLARE startperiod text;
DECLARE endperiod text;
DECLARE datestart text;
DECLARE dateend text;
DECLARE counter integer;
BEGIN

--DROP TABLE IF EXISTS ohlohdata;

CREATE TABLE IF NOT EXISTS ohlohdata(
dates date,
projectid integer,
teamsizes integer,
al1m integer,
rl1m integer,
commitsl1m integer,
activl1m integer,
vcs varchar(100),
language varchar(100),
license varchar(100),
projectname varchar(100)
);

startyear:=(SELECT date_part('year',startdate));
startmonth:=(SELECT date_part('month',startdate));
startday:=(SELECT date_part('day',startdate));
endyear:=(SELECT date_part('year',enddate));
endmonth:=(SELECT date_part('month',enddate));
endday:=(SELECT date_part('month',enddate));

FOR i IN startyear..endyear
LOOP

Raise notice 'Year : %',i;

startperiod:=i;
endperiod:=i+1;
IF (i=1) THEN
datestart:=CAST(startperiod as text)|| '-' || CAST(startmonth as text) || '-' ||
CAST(startday as text);
```

```
ELSE
datestart:=CAST(startperiod as text)|| '-01-01';
END IF;

IF (i=endyear) THEN
 dateend:=CAST(endperiod as text)|| '-' || CAST(endmonth as text) || '-' ||
CAST(endday as text);
ELSE
 dateend:=CAST(endperiod as text)|| '-12-31';
END IF;

FOR row IN EXECUTE
'SELECT DISTINCT
  AF.MONTH AS DATES,
AN.twelve_month_contributor_count AS TEAMSIZES,
AF.PROJECT_ID AS PID,
AF.CODE_ADDED AS AL1M,
AF.CODE_REMOVED AS RL1M,
AF.COMMITS AS COMMITSL1M,
AF.CONTRIBUTORS AS ACTIVL1M,
R.TYPE AS VCS,
LN.NAME AS LANGUAGE,
LI.NAME AS LICENSE,
P.NAME as PROJECTNAME
FROM
ACTIVITY_FACTS AS AF
INNER JOIN ANALYSIS AN
ON AF.analysis_id=AN.id
INNER JOIN PROJECTS AS P
ON
AF.PROJECT_ID=P.ID
LEFT OUTER JOIN
ENLISTMENTS EN
ON
AF.PROJECT_ID=EN.PROJECT_ID
LEFT OUTER JOIN REPOSITORIES AS R
ON EN.REPOSITORY_ID=R.ID
LEFT OUTER JOIN
PROJECT_LICENSES AS PL
ON
AF.PROJECT_ID=PL.PROJECT_ID
```

```
LEFT OUTER JOIN
LICENSES AS LI
ON
PL.LICENSE_ID= LI.ID
LEFT OUTER JOIN
LANGUAGES AS LN
ON
AF.MAIN_LANGUAGE_ID=LN.ID
WHERE
AF.MONTH BETWEEN ''' || datestart || ''' AND ''' || dateend || '''
ORDER BY AF.MONTH'

LOOP

counter:=counter+1;

INSERT INTO ohlohdata
     (dates, projectid,teamsizes, al1m, rl1m, commitsl1m, activl1m, vcs, "lan-
guage", license, projectname)
VALUES
     (row.dates,row.pid,row.teamsizes ,row.al1m ,row.rl1m ,row.commitsl1m ,row.ac-
tivl1m , row.vcs,row.language, row.license, row.projectname);

IF (counter=500) THEN
 COMMIT;
 counter:=0;
END IF;

END LOOP;
END LOOP;
RETURN 0;
END
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION ohlohdata(date, date)
  OWNER TO asitti;
```

5.3 gfw.chart Toolkit

Version	1.0
Date	03.06.2014
Title	Toolkit for Graphics Generation
Author	Ahmet Sitti
Maintainer	Ahmet Sitti<ahmet.sitti@studium.uni-erlangen.de>
Description	A toolkit for generating graphics using parameters
Depends	R(>=2.10.0),
Imports	plyr, ggplot2, zoo
System Requirements	
License	GPL-2

5.4 R Topics Documented

5.4.1 gfw.chart

Description

Main class in gfw.chart toolkit to generate the graphics

Usage

gfw.chart(data=df,params=params)

Arguments

data	data frame containing unprocessed raw data
params	list of parameters

Details

Base class used by gfw.chart and gfw.model.factory classes.

Example

```
params<-list(aggregation=aggregation,dimensions=dimensions,
          facet=facet,graphics=graphics,model=model,timeseries=timeseries,
          explanatory=explanatory,coordinatesystem=coordinatesystem)
gc <-gfw.chart(data=df,params=params)
plotlist<-gc$creategraph()
show(plotlist[[1]])
```

5.4.2 gfw.chart.model.processor

Description

Used by gfw.chart class in order to process the data frame per dimension based on model parameter.

Usage gfw.chart.model.processor(data=data,params=params,dindex=dindex)

Arguments

data	preprocessed data
params	list of instruction parameters
dindex	integer index of current dimension

Details

gfw.chart.model. processor class implements the instructions given by gfw.chart.param.model class. The gfw.chart.model.processor class chooses what model was given by gfw.chart.param.model class and implements the instructions on the data frame passed by a reference. The data processed by gfw.chart.model. processor is a per-dimension-preprocessed (can be aggregated) data frame.

Example

```
model._processor <-gfw.chart.model.processor(data=data,params=params,
dindex=1)
modeldata<-factory$getdata()
```

5.5 Parameter Classes:

5.5.1 gfw.chart.param

Description

Base class for parameter classes.

Usage

Arguments

data	preprocessed data
params	list of instruction parameters
dindex	integer index of current dimension

Details

All parameter classes derive from gfw.chart.param class and inherit the "use" property. This property allows the toolkit to determine whether to use the parameter or not.

Example

```
model_processor <-gfw.chart.model.processor(data=data,params=params,
dindex=1)
modeldata<- model_processor$getdata()
```

5.5.2 gfw.chart.param.aggregation

Description

Parameter class for aggregation instructions.

Usage gfw.chart.param.aggregation(useaggregate,aggregatecolumn, aggregaterow,aggegatefunction= aggegatefunction)

Arguments

useaggregate	turn aggregation on / off. Can be either TRUE or FALSE. Default is TRUE.
aggregatecolumn	Character column name for aggregation. It is used both for aggregating the data and in facet formula selection. Default value is NA.
aggregaterow	Character row name for aggregation. It is used both for aggregating the data and in facet formula selection. Default value is NA.

Details

Example

```
aggregation<-gfw.chart.param.aggregation(aggregatecolumn=NA,
aggregaterow="license",aggregatefunction=sum)
```

5.5.3 gfw.chart.param.dimension

Description

Parameter class for a dimension which is used in gfw.chart.param.dimensions class.

Usage gfw.chart.param.dimension(dimension=dimension,confidenceinterval= confidenceinterval,confidencevarinterval=confidencevarinterval,predictioninterval= confidencevarinterval)

Arguments

dimension	Name of the dimension.
confidenceinterval	Boolean confidence interval. Decides whether confidence interval should be shown or not for this dimension.
confidencevarinterval	Boolean confidence interval. Decides whether confidence interval should be shown or not for this dimension.

predictioninterval	Boolean prediction interval. Decides whether prediction interval should be shown or not for this dimension.

Details

Example

```
dimension<-gfw.chart.param.dimension(dimension="xx",confidenceinterval=T,
confidencevarinterval=T,predictioninterval=T)
```

5.5.4 gfw.chart.param.dimensions

Description

Parameter class for dimensions.

Usage gfw.chart.param.dimensions(dimensions=list(dimension))

Arguments

dimensions List of dimension type of gfw.chart.param.dimension.

Details

Example

```
dimension1<-gfw.chart.param.dimension(dimension="xx",confidenceinterval=T,
     confidencevarinterval=T,predictioninterval=T)
dimension2<-gfw.chart.param.dimension(dimension="yy",confidenceinterval=F,
     confidencevarinterval=T,predictioninterval=T)
  dimensions<-
  gfw.chart.param.dimensions(dimensions=list(dimension1,dimension2))
```

5.5.5 gfw.chart.param.explanatory

Description

Parameter class for explanatory column.

Usage gfw.chart.param.explanatory(explanatory= explanatory)

Arguments

explanatoryColumn name system. Default value is cartesian_coord

Details

Example

explanatory<-gfw.chart.param.explanatory(explanatorycolumn="xx")

5.5.6 gfw.chart.param.graph

Description

Parameter class for a graphics layer.

Usage

gfw.chart.param.graph(usegraph=T,geomfunction=geom_point,geomaes=aes(x=x, y=y),
geomdata=NULL,geomcolour=NULL,geomlegend=NULL)

Arguments

usegraph Boolean value. Decides whether the graph shold be shown or not.

geomfunction ggplot function such as geom_point, geom_line etc.

geomaes Aesthetics function and aesthetics parameters.

geomdata Data used by the layer

geomcolor Layer graphics color

geomlegend Legend of the layer

Details

Example

```
# default for ggplot
graphlayer1<-gfw.chart.param.graph(geomfunction=ggplot,geomaes=aes(x=x,y=y))
graphlayer2<-gfw.chart.param.graph(geomfunction=geom_point,
geomaes=aes(x=x,y=y))
graphlayer3<-gfw.chart.param.graph(geomfunction=geom_line,
geomaes=aes(x=x,y=fitted))
graphlayer4<-gfw.chart.param.graph(geomfunction=geom_smooth,
geomaes=aes(x=x,y=y))
```

5.5.7 gfw.chart.param.graphics

Description

Holds and maintains a list of graphs of type gfw.chart.param.garph objects.

Usage gfw.chart.param.graphics(graphs=list(graph))

Arguments

graphs List of graph type of gfw.chart.param.graph.

Details

Example

```
# default for ggplot
graphlayer1<-gfw.chart.param.graph(geomfunction=ggplot,geomaes=aes(x=x,y=y))
```

graphlayer2<-
gfw.chart.param.graph(geomfunction=geom_point,geomaes=aes(x=x,y=y))
graphlayer3<-
gfw.chart.param.graph(geomfunction=geom_line,geomaes=aes(x=x,y=fitted))
graphlayer4<-
gfw.chart.param.graph(geomfunction=geom_smooth,geomaes=aes(x=x,y=y))
graphics<-gfw.chart.param.graphics(graphs=list(graphlayer1,graphlayer2,
graphlayer3,graphlayer4))

5.5.8 gfw.chart.param.facet

Description

Parameter class for facet used for faceting ggplot result.

Usage gfw.chart.param.facet(facetfunction=facetfunction,facetscale=facetscale,
aggregation=aggregation)

Arguments

facetfunction facet function lays out panels in a grid

facetscale scale graphics

aggregationaggregation object type of gfw.chart.param.aggregation class

Details

Example

facet<-gfw.chart.param.facet(facetfunction=facet_grid,facetscale="free_y",
aggregation=aggregation)

5.5.9 gfw.chart.param.model

Description

Parameter class for model definitions.

Usage gfw.chart.param.model(modelfunction="lm",modelformula=x~y,…)

Arguments

modelfunction model function such as glm, lm, arima.

modelformula formula used by the model function.

… variable number of parameters per function.

Details

Example

model<-
gfw.chart.param.model(modelfunction="lm",modelformula=NA,family=gaussian)

5.6 Data Frame Splitter Classes:

Sometimes there is need for pre-processing the data to facilitate grouping. By set rules, the values in one column can be used to place values in an additional column upon which the grouping can then be done.

5.6.1 gfw.chart.split

Description

Used to define a split parameter to be used in gfw.chart.param.split.

Usage gfw.chart.param.split(splitvalue= splitvalue,setvalues= setvalues))

Arguments

splitvalue Character value. Contains the value assigned to the new column per value found in the setvalues vector

setvalues Vector of character search values

Details

This class is a supporting class to gfw.chart.splitset class. It contains the values depending on values of the datacolumn parameter in gfw.chart.splitset class.

Example

vcssplit<-gfw.chart.split(splitvalue="vcsgroup1",setvalues=c("CvsRepository", "SvnRepository","HgRepository"))

languagesplit<-gfw.chart.split(splitvalue="languagegroup1",setvalues=c("c","cpp"))

5.6.2 gfw.chart.splitset

Description

Parameter class for time series.

Usage

gfw.chart.param.splitset(splitcolumn=splitcolumn,datacolumn=datacolumn,splitset=splitset)

Arguments

splitcolumnname of the splitting column.

datacolumnname of the data column to be used for splitting.

splitset list of objects type of gfw.chart.param.split class.

Details

This class encapsulates split column, data column and splitsets in order to use them in gfw.chart.datasplitter class. It basicly contains the instructions for the dataplitter class.

Example

vcssplit1<-gfw.chart.param.split(splitvalue="vcsgroup1",setvalues=c("CvsRepository",
"SvnRepository","HgRepository"))
vcssplit2<-gfw.chart.param.split(splitvalue="vcsgroup2",setvalues=c("GitRepository",
"BzrRepository","SvnSyncRepository"))

languagesplit1<-gfw.chart.param.split(splitvalue="languagegroup1",setvalues=c("c","cpp"))
languagesplit2<-
gfw.chart.param.split(splitvalue="languagegroup2",setvalues=c("java","python","php"))

splitset1<-gfw.chart.param.splitset(splitcolumn="vcstype",datacolumn="vcs",
splitset=list(vcssplit1,vcssplit2))
splitset2<-gfw.chart.param.splitset(splitcolumn="languagetype",datacolumn="language",
splitset=list(languagesplit1,languagesplit2))

5.6.3 gfw.chart.datasplitter

Description

Modifies the data frame and creates a split data frame by adding new columns with grouped values.

Usage gfw.chart.datasplitter (data=data, splitsets=splitsets)

Arguments

Data	data frame to be modified
splitsets	list of splitsets of type gfw.chart.param.splitset

Details

If the data is a time series then it needs to be processed in a special way so the data becomes a time series. This parameter allows the user to create time series graphics.

Example

vcssplit1<-gfw.chart.param.split(splitvalue="vcsgroup1",
setvalues=c("CvsRepository","SvnRepository","HgRepository"))
vcssplit2<-gfw.chart.param.split(splitvalue="vcsgroup2",setvalues=c("GitRepository",
"BzrRepository","SvnSyncRepository"))

languagesplit1<-gfw.chart.param.split(splitvalue="languagegroup1",

```
setvalues=c("c","cpp"))
languagesplit2<-gfw.chart.param.split(splitvalue="languagegroup2",
setvalues=c("java","python","php"))

splitset1<-gfw.chart.param.splitset(splitcolumn="vcstype",datacolumn="vcs",
splitset=list(vcssplit1,vcssplit2))
splitset2<-gfw.chart.param.splitset(splitcolumn="languagetype",
datacolumn="language",splitset=list(languagesplit1,languagesplit2))

datasplitter<-gfw.chart.datasplitter(splitsets=list(splitset1,splitset2))
splittteddata<-datasplitter$getSplitData(subsetdat)
```

5.7 Test Data Generator Classes

5.7.1 gfw.chart.testdata

Description

Creates test data defined by parameters.

Usage gfw.chart.testdata(params=params)

Arguments

params	list of instruction parameters

Details

Testdata class takes a parameter called params which is a list of parameters containing instructions about how testdata should create the data frame, column names and how the columns should be filled. Parameters are as follows:

dimensions	
columns	list of parameters containing column details. A column can be of type date, character, numeric or logical and it is defined with columntype. Columnindex is an integer value to access the column defined by dimensions. columnname is the name of the column. A column can also be a factor. The filldata list fills the column with the data using fillwith list. Fillwith can be a single value but also an expression. numberofrows defines the number of rows in the data frame to be created. The created data frame can also be saved by using the saveas parameter.

Example

```
paramsSSlogis=list(dimensions=as.list(paste("var",     1:10,sep="")),columns=list(list(
columnindex=1,columntype="date",columnname="time",columnfactor=F,values=as.D
```

```
ate(c("2000-01-
01"))),list(columnindex=2,columntype="character",columnname="license",
columnfactor=T,values=c("academic","gpl"),filldata=list(columnindex=2,fillwith=list(
expression(100/(2+exp((12500-as.numeric(df$time))/200)) + rnorm(length(df$al1m))),
expression(1/(1002+exp((3-as.numeric(df$time))/4))    +    rnorm(length(df$al1m)))))),
list(columnindex=3,columntype="integer",columnname="al1m",columnfactor=F,value
s=c(1000:2000))),numberofrows=100,saveas="df.dat")
testdatagenerator<-gfw.chart.testdata(params=paramsSSlogis)
testdataframe<- testdatagenerator $getdataframe()
```