Visualizing Work Process in Software Engineering with Developer Rivers

Michael Burch, Tanja Munz, Fabian Beck, and Daniel Weiskopf
VISUS, University of Stuttgart, Germany

Presenter: Arthur Sun
Outline

• What’s the current problem for large software projects
• What’s the previous solution for large projects InfoVis
• What the paper presents
  • DataSet
  • InfoVis Encoding Technique
  • Visualization Method
  • Sample Usage
• Future Improvement
Large open-source projects: 560,519 commits
Large open-source projects: 5659 contributors
Large open-source projects: 441 releases
## Mirror of Apache Storm

- 5,320 commits
- 59 branches
- 18 releases
- 187 contributors

**Branch:** master

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>Merge branch 'STORM-1155' of <a href="https://github.com/tgravescs/storm">https://github.com/tgravescs/storm</a> into ...</td>
<td>5 days ago</td>
</tr>
<tr>
<td>conf</td>
<td>Merge branch 'STORM-1155' of <a href="https://github.com/tgravescs/storm">https://github.com/tgravescs/storm</a> into ...</td>
<td>5 days ago</td>
</tr>
<tr>
<td>dev-tools</td>
<td>bump timeout by 50% due to intermittent travis build failures</td>
<td>5 days ago</td>
</tr>
<tr>
<td>docs</td>
<td>Merge branch 'STORM-1155' of <a href="https://github.com/tgravescs/storm">https://github.com/tgravescs/storm</a> into ...</td>
<td>5 days ago</td>
</tr>
<tr>
<td>examples/storm-starter</td>
<td>STORM-1161: Add License headers and add rat checks to builds</td>
<td>11 days ago</td>
</tr>
<tr>
<td>external</td>
<td>Merge branch 'patch-6' of <a href="https://github.com/vesense/storm">https://github.com/vesense/storm</a> into STORM...</td>
<td>10 days ago</td>
</tr>
</tbody>
</table>

Latest commit 5a79ba5 3 days ago

User: revans2 Added STORM-1190 to Changelog
So many changes in Commit, Releases, Contributors! How can I see them in a whole picture?
What’s do we want

A whole picture of the overall progress of extreme large software engineering project proceeding with time frame in detailed visualization for major participants, their contribution to respective work, how much amount of work they did and their work change
Sankey diagrams are a specific type of flow diagram, in which the width of the arrows is shown proportionally to the flow quantity.

Problem: No Time Frame
Previous work – Gantt Chart

A Gantt chart is a type of bar chart that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the involved tasks. Modern Gantt charts also show the dependency.

Problem: 1. Doesn’t show how many people/resources involved in project
2. Don’t have a whole picture about the project
What’s the author propose

• A graph flow which can not only show the interconnection of different modules of development along with the timeframe but also the programmer who took part in the whole project with vivid color to show difference
Visualization Technique

DataSet: Developer Activity Model

Encoding: Develop River for Time-Varying Developer Activities
Dataset: Developer Activity Model

1. Abstract commit as c, time as t, developer name as d, files as f, all files as F, file modules hierarchy as H

Partition sequence of commits into equally-sized intervals for each interval of commits and every module

Calculate individual developer activity of their files for each module and reach module specific developer activity

Calculate weighted transition matrix for each developer $M_i = \text{Mat}((l+1)*(l+1))$

Calculate the weighted transition matrix for all developers by summing up $M_i$

Paper didn’t show how to map real data into Activity Model Matrix
Mapping Activity Model Matrix into Develop Rivers without intersection
Developer Rivers

Mapping Activity Model Matrix into Develop Rivers with intersection
Developer Rivers Curves

1. Transition: how developers change their behavior between different module groups using cubic Bezier Curves
2. Transition color is a linear gradient from color of start module to target module

Paper didn’t show how to link Matrix Data with Bezier curve creation

Influents
Developer join current step

Effluents
Level the main river
Developer Rivers Curves

Effluents
Leave the main river

Effluents
Leave the main river
Inflow/Outflow: A transition from or to the outside of the diagram identifies a developer entering or leaving the project.

Constant Flow: An intra-transition with a constant width indicating a group of developers constantly working on the same module.

Growth/Decline: An intra-transition with an increasing or decreasing strength hints at a group of developers that keep working on a module but with changing total effort.

Split/Merge: A module that is split into or merged from multiple flows shows a qualitative change of developer activity (i.e., developers’ relative focus switches between modules). While at least one inter-transition is required for this pattern, one of the flows can be an intra-transition.

Exchange: A pair of intra-transitions connecting two modules in opposite directions at the same time is a specific qualitative change of activity: some developers move between the two modules in both directions.
In contrast to Python, the Linux kernel did not undergo an extensive growth of changes in the studied period (however, we also study a shorter period), but just a slight growth. A minor exception is the year 2013, where development activity slightly declined with respect to the previous year. It is further interesting to note that the overall pattern created by the flows is quite similar across all periods: a small inflow distributed among all selected modules (relative to their size in the period), a similar but even smaller outflow, large constant flows for all modules, and only small inter-transitions; only between drivers and arch, there are considerable inter-transitions showing an exchange pattern—this might be partly explained through their size, but could also mean that these two modules are related so that developers naturally switch between them. In general, the development of the Linux kernel seems to be an established work process with low amount of variance in the developer activity.

2) Developers:
Although the overall development activity is quite stable, the most active developers within the modules selected in Figure 8 change quite often over the full time period. To further investigate the stability of developer roles, Figure 9 shows developer sparklines for the top 5 most active ones. We find, for instance, that some developers have clear responsibilities that stay constant over time (e.g., Greg Kroah-Hartman).
Visual Patterns

• Main Module Overview
  • Consists main directories, developers and their contributors

• File Type Overview
  • Automatic definition of modules by file types

• Developer Sparklines
  • Highlight top 5 star developer contributing most to the whole project

• Subsystem Details
  • Modules in a subdirectory of the system shows details of a specific system
Fig. 5. Python main module overview, 1991–2000, 2-years interval.

van Rossum editing more documentation files (as confirmed by using details on demand). These transitions fall together with Fred Drake entering development and changing Doc files. For the period of 1996–1997, the documentation effort was considerably intensified, which can be traced back largely to Fred Drake, who then became the most active developer of Doc. After that, during 1999–2000, Fred Drake started also to working in Lib and Modules and Guido van Rossum changed his focus back to these parts—this causes the strong transitions from Doc to the blue and green river between the last two periods.

Generating developer sparklines for the top 5 developers of Python allows to confirm most of these observations. The Developer Rivers as shown in Figure 6 encode again the full period of 1991–2011; since a stretched aspect ratio matches the character of a sparkline, the temporal resolution can be increased to one year per period. Studying the sparklines of Guido van Rossum and Fred Drake, we see the increasing effort in documentation and the subsequent switch of focus towards implementation. But much more insight can be gained based on these diagrams; just to name a few examples: Guido van Rossum steadily worked less on the project during 1998–2004 (with respect to changed files) followed by a sudden peak in 2007. Georg Brandl joined the project late (2005); his efforts switch back and forth between Lib and Doc. Benjamin Peterson, in contrast to the others, started to contribute heavily already in his first year. Jack Jansen had two unconnected periods of high activity, while Fred Drake particularly focused on Doc.

3) Subsystems: With these observations as a background, it is now interesting to go into the details of the module structure. As an example, we chose the Tools directory and mark some of its main subdirectories as new modules. Figure 7 depicts the subsystem details; due to limited space, we here restricted the studied period to 1997–2011, divided into 5-years intervals. The resulting diagram is quite different from those discussed before: the overall number of changes is not increasing, but roughly stays constant; there is a large variety of dominance comparing two time periods; and there are many transition connections between the different modules.

In the first period (1997–2001), scripts (a collection of scripts for various purposes), freeze (a Python compiler for Unix), pynche (a color editor), idle (a Python code editor), and compiler (a Python bytecode compiler) assembled the main activity. In the transition to the next period (2002–2006), there is no considerable outflow—nearly all Tools developers continued to work on the project, however, with overall less activity while some other developers joined. Within the period, scripts (see above), bgen (a source code generator), idle (see above), and pybench (a benchmark suite) were most active, following the strongest transitions between the two first periods, considerable developer activity moved from idle and compiler to scripts (split in idle and compiler, merge in scripts). In the last period (2007–2011), we again find a

Python main module overview
van Rossum editing more documentation files (as confirmed by using details on demand). These transitions fall together with Fred Drake entering development and changing Doc files. For the period of 1996–1997, the documentation effort was considerably intensified, which can be traced back largely to Fred Drake, who then became the most active developer of Doc. After that, during 1999–2000, Fred Drake started also to working in Lib and Modules and Guido van Rossum changed his focus back to these parts—this causes the strong transitions from Doc to the blue and green river between the last two periods.

Generating developer sparklines for the top 5 developers of Python allows to confirm most of these observations. The Developer Rivers as shown in Figure 6 encode again the full period of 1991–2011; since a stretched aspect ratio matches the character of a sparkline, the temporal resolution can be increased to one year per period. Studying the sparklines of Guido van Rossum and Fred Drake, we see the increasing effort in documentation and the subsequent switch of focus towards implementation. But much more insight can be gained based on these diagrams; just to name a few examples: Guido van Rossum steadily worked less on the project during 1998–2004 (with respect to changed files) followed by a sudden peak in 2007. Georg Brandl joined the project late (2005); his efforts switch back and forth between Lib and Doc. Benjamin Peterson, in contrast to the others, started to contribute heavily already in his first year. Jack Jansen had two unconnected periods of high activity, while Fred Drake particularly focused on Doc.

3) Subsystems: With these observations as a background, it is now interesting to go into the details of the module structure. As an example, we chose the Tools directory and mark some of its main subdirectories as new modules. Figure 7 depicts the subsystem details; due to limited space, we here restricted the studied period to 1997–2011, divided into 5-years intervals. The resulting diagram is quite different from those discussed before: the overall number of changes is not increasing, but roughly stays constant; there is a large variety of dominance comparing two time periods; and there are many transition connections between the different modules. In the first period (1997–2001), scripts (a collection of scripts for various purposes), freeze (a Python compiler for Unix), pynche (a color editor), idle (a Python code editor), and compiler (a Python bytecode compiler) assembled the main activity. In the transition to the next period (2002–2006), there is no considerable outflow—nearly all Tools developers continued to work on the project, however, with overall less activity while some other developers joined. Within the period, scripts (see above), bgen (a source code generator), idle (see above), and pybench (a benchmark suite) were most active, following the strongest transitions between the two first periods, considerable developer activity moved from idle and compiler (split in idle and compiler, merge in scripts). In the last period (2007–2011), we again find a...
Fig. 5. Python main module overview, 1991–2000, 2-years interval.

Fig. 6. Python developer sparklines of top 5 developers, 1991–2011, 1-year interval.

van Rossum editing more documentation files (as confirmed by using details on demand). These transitions fall together with Fred Drake entering development and changing Doc files. For the period of 1996–1997, the documentation effort was considerably intensified, which can be traced back largely to Fred Drake, who then became the most active developer of Doc. After that, during 1999–2000, Fred Drake started also to working in Lib and Modules and Guido van Rossum changed his focus back to these parts—this causes the strong transitions from Doc to the blue and green river between the last two periods.

Generating developer sparklines for the top 5 developers of Python allows to confirm most of these observations. The Developer Rivers as shown in Figure 6 encode again the full period of 1991–2011; since a stretched aspect ratio matches the character of a sparkline, the temporal resolution can be increased to one year per period. Studying the sparklines of Guido van Rossum and Fred Drake, we see the increasing effort in documentation and the subsequent switch of focus towards implementation. But much more insight can be gained based on these diagrams; just to name a few examples: Guido van Rossum steadily worked less on the project during 1998–2004 (with respect to changed files) followed by a sudden peak in 2007. Georg Brandl joined the project late (2005); his efforts switch back and forth between Lib and Doc. Benjamin Peterson, in contrast to the others, started to contribute heavily already in his first year. Jack Jansen had two unconnected periods of high activity, while Fred Drake particularly focused on Doc.

3) Subsystems: With these observations as a background, it is now interesting to go into the details of the module structure. As an example, we chose the Tools directory and mark some of its main subdirectories as new modules. Figure 7 depicts the subsystem details; due to limited space, we here restricted the studied period to 1997–2011, divided into 5-years intervals. The resulting diagram is quite different from those discussed before: the overall number of changes is not increasing, but roughly stays constant; there is a large variety of dominance comparing two time periods; and there are many transition connections between the different modules.

In the first period (1997–2001), scripts (a collection of scripts for various purposes), freeze (a Python compiler for Unix), pynche (a color editor), idle (a Python code editor), and compiler (a Python bytecode compiler) assembled the main activity. In the transition to the next period (2002–2006), there is no considerable outflow—nearly all Tools developers continued to work on the project, however, with overall less activity while some other developers joined. Within the period, scripts (see above), bgen (a source code generator), idle (see above), and pybench (a benchmark suite) were most active, following the strongest transitions between the two first periods, considerable developer activity moved from idle and compiler (split in idle and compiler, merge in scripts). In the last period (2007–2011), we again find a
In particular, we split the history of the project into seven clean periods covering only full years. This dataset consists of 70,813 commits by 159 developers. This project forms a suitable example for demonstrating our approach as it has reasonable size and history. Since the same dataset has already been visualized and visualized in a different context for the storylines of Python's development history in clean periods covering only full years.
Linux Developer Sparkline of top 5 developers
Linux Subsystem details of Tools Directory
Future Improvements

• Show us how do the author organize the data (Data->Matrix)
• Show how to transfer the data into influents and effluents (Matrix->Influents)
• Provide tool ready for practitioners who can use developer river directly (No description about how to tackle the dataset)
• Distinguished colors may be up to 10 colors, otherwise graph may be hard to see
• Transfer the way to study software engineering research into social-technical aspects of engineering research