# Visualizing the Evolution of Module Workflows

Marcel Hlawatsch*, Michael Burch*, Fabian Beck*, Juliana Freire†, Claudio Silva† and Daniel Weiskopf*
*University of Stuttgart, Germany
Email: firstname.lastname@visus.uni-stuttgart.de
†New York University, United States
Email: juliana.freire@nyu.edu, csilva@nyu.edu

***Abstract*—Module workflows are used to generate custom applications with modular software frameworks. They describe data flow between the modular components and their execution under certain parameter configurations. In many cases, module workflows are modeled in a graphical way by the user. To come up with the final result or to explore multiple solutions, they often undergo many iterations of adaptation. Furthermore, existing workflows may be reused for new applications. We visualize the evolution of module workflows with a focus-and-context approach and visualization techniques for time-dependent data. Our approach provides insight into user behavior and the characteristics of the underlying systems. As our examples show, this can help identify usability issues and indicate options to improve the effectiveness of the system. We demonstrate our approach for module workflows in VisTrails, a modular visualization system that allows building custom visualizations by combining different modules for processing and visualizing data.**

***Index Terms*—Workflows, information visualization, software visualization.**

## I. Introduction

Modular software frameworks allow users to create custom software for their specific needs by combining a set of software modules. For example, creating effective visualizations strongly depends on the application and typically requires the combination of different visualization techniques. A modular visualization framework enables the user to combine different visualization modules for a specific application.

Using such frameworks often requires one to reference the respective libraries in the source code and call their functions, usually by writing code. However, there are also frameworks that allow combining software modules on a more abstract level. Typically, a graph for the data flow between modules and parameters for executing them are specified. For example, a module for loading the dataset is connected to a filter module removing noise from the data. This module is again connected to a module computing a histogram, which is the result desired by the user. This can often be done in a graphical way, e.g., by drawing the respective connections between the modules.

There seems to be no common term for the involved data flow graph. In the context of scientific data processing, e.g., in bioinformatics and other life sciences, the term *scientific workflow* [1] is used. *Business workflows* [2] usually include the execution of software components as well, but are not restricted to software only. This holds even more for the term *workflow*, which describes the usage and combination of resources and tasks in general. The term *(software) pipeline* is also related, but it normally describes the more restrictive case of a linear sequence of processing steps. We therefore decided to use the term *module workflow* for the usage and combination of software components.

The evolution of a module workflow can provide information about the characteristics of underlying systems, related bottlenecks, and usability issues. Furthermore, it shows the steps users made to generate a final workflow and result. This can help recapitulate finished work and reuse parts of previously created workflows. In this paper, we present a visualization that eases accessing the information contained in the evolution of module workflows. Our work includes the following contributions:

- A visualization of module lifetimes and events. Grouping and filtering methods additionally improve its scalability.
- A visual representation of branches in the workflow evolution. This representation requires only a small spatial footprint and scales to large histories with many branches.
- A focus-and-context approach with multiple coordinated views that combines these visualizations and provides good scalability with respect to large workflow histories.

We demonstrate the utility in a case study related to the change history of module workflows created by users of the visualization framework VisTrails [3]. The case study shows that our technique helps reconstruct the individual steps of a user session. This provides not only an indication of the user's intention but also information about general usage patterns.

## II. Related Work

Visualization in the context of software evolution typically focuses on modifications of the source code, e.g., the submission history of version control systems [4], [5], [6], the changing interaction of developers with code [7], [8], or the evolution of software metrics [9], [10]. An overview of these visualization techniques can be found, e.g., in [11], [12]. In this paper, we visualize the evolution of module workflows: code is not changed but the configuration of the system represented by the workflow, such as the modules used and their parameters. This aspect of software evolution has not yet been studied from a visualization perspective.

There is a close relation to *provenance*. The term describes all information required to reproduce a result. Hence, in the context of workflows, provenance information describes also the evolution of the workflow. Research on provenance [13], [14] mainly focuses on capturing provenance and accessing it

with query languages. However, little attention has yet been devoted to the problem of visualizing provenance information. Macko and Seltzer [15] developed a tool for exploring large data provenance graphs. They use graph summarization and apply semantic zooming to enable users to incrementally explore the details of the graph. Rio and Silva [16] and Anand et al. [17] use visualization to support querying the provenance information. In contrast, our approach visualizes the evolution of workflows and focuses on the temporal aspects of the modules used and related events.

Target applications for our visualization approach are all systems that allow a workflow-like combination of modules or components. Examples are systems for scientific workflows like Kepler [18], Taverna [19], or Galaxy [20]. There are also many visualization applications and frameworks available that allow for building the designated visualization by connecting and combining different modules: AVS [21], VTK [22], MeVisLab [23], the InfoVis Toolkit [24], prefuse [25], etc. VisTrails [3] plays a special role in this list: besides allowing a workflow-based combination of visualization modules, it has integrated support for provenance.

Our approach is also related to techniques for visualizing time-dependent or time-oriented data [26]. Using bars for visualizing time, like applied in our visualization, is a common approach, sometimes called *timelines* [27] and variations of it can be found in different works [28], [29], [30], [31]. A well-known example of this type of visualization, often applied in project management, are Gantt charts [32], which represent tasks and their temporal dependencies on a timeline. Related to our scenario studying evolution of modules, Viégas et al. [33] use bar-like diagrams to visualize the change history of Wikipedia. Waser et al. [34] apply enhanced bar charts with branching and symbols to control simulations in a computational steering environment. Using bars is also a common approach in trace visualization of parallel systems [35], [36]. Similar as applied in our visualization, Burch et al. [37] combine timelines with a tree visualization grouping the individual lines. With respect to indicating events on the timeline, Wang et al. [38] use symbols and histograms to display events. A combination of bars and symbols are used by Cousins and Kahn [39] to visualize different types of events in patient data. Whereas they employ symbols for events without temporal extent and bars for intervals, we represent events always with symbols and use bars to depict the domain events operate on, in our case visualization modules.

Although some of our used visualization techniques exhibit similarities to the ones mentioned above, we present a new approach for visualizing the temporal and causal relations of branches in the workflow evolution. Furthermore, the overall visual design and combination of techniques together with the application to module workflow histories is novel.

## III. Module Workflows

Before we describe our visualization approach in the next section, we discuss certain characteristics of module workflows that are considered in our visual design.

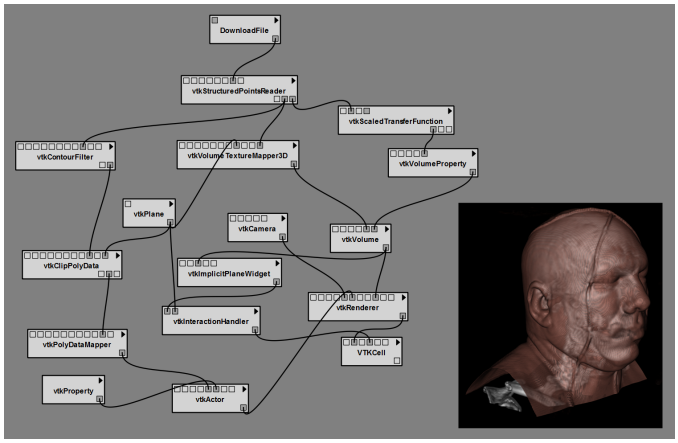



Fig. 1. Example of a module workflow. Different modules are connected to process data and generate the desired result. This example shows a module workflow from VisTrails that generates a volume rendering of a human head (small window, bottom right).

The usage of module workflows (e.g., the visualization workflow shown in Figure 1) exhibits some conceptual differences to classic software development, where source code is written and then compiled to create an application. The applications from classic software development typically offer some flexibility and a set of features that can be used to solve a variety of different problems with similar context. The work with such applications does not require one to modify code. The offered features are used to generate the desired result from the input data during runtime. Since code and execution are conceptually separated, analyzing the evolution of software usually does not involve information about the runtime configuration and usage of the application.

With module workflows, parts of the development process are shifted to the runtime of the software and the boundary between developing and using the software is blurred. There are similarities to working with interpreted or scripting programming languages. Module workflows are typically created for a very specific and narrowed purpose and their output directly contains the requested information. Hence, analyzing the evolution of module workflows provides also information about the usage phase of the software. One has to discern, however, between the execution of the framework providing the module workflow and executing the workflow itself. Our approach can be considered as a visualization of the evolution of the workflow or as a visualization of the user behavior during the execution of the module framework.

We designed our visualization under the assumption of certain characteristics of module workflows and their usage, which are described in the following. Module workflows combine modules that typically provide quite complex functionality, e.g., filtering or transforming data. Therefore, this corresponds to software development on a high level of abstraction. The number of used modules is rather low compared to the number of used functions in a typical software project. Additionally, there are in many cases no complex dependen-

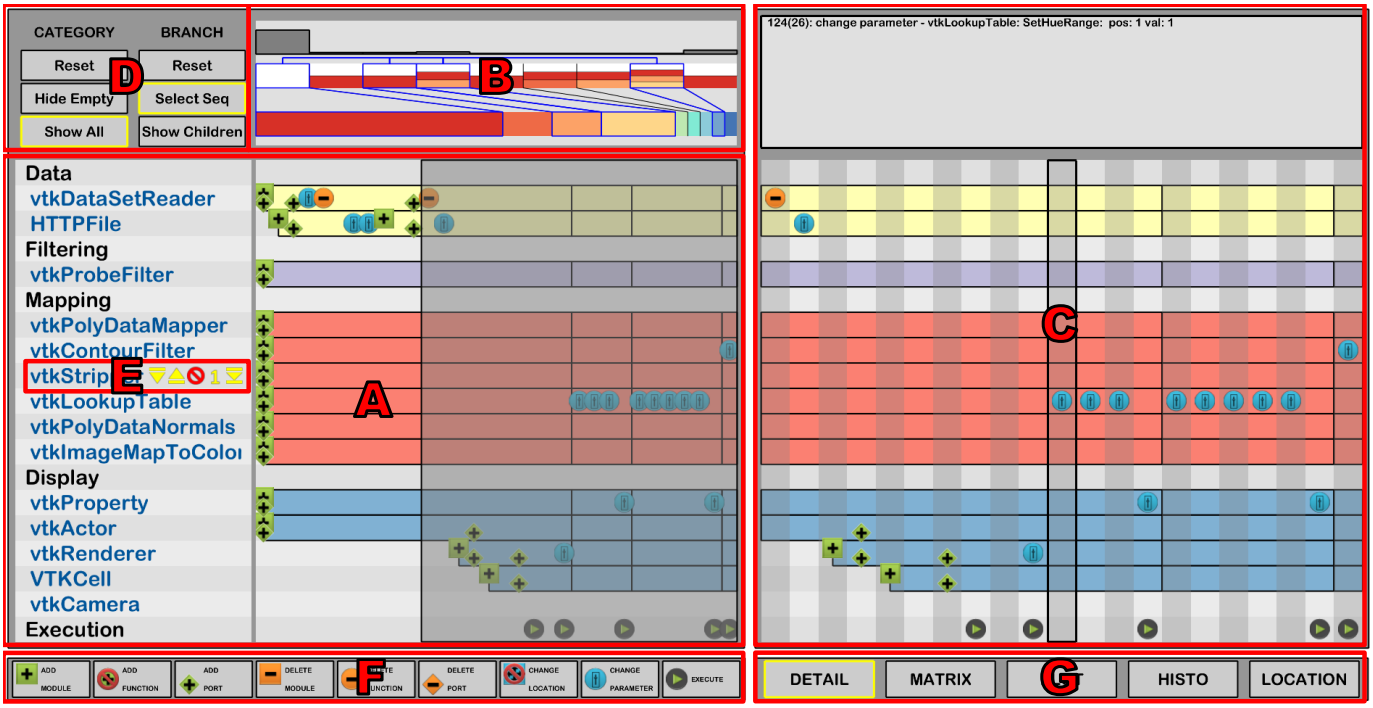



Fig. 2. The visualization is split in three main parts: (A) the module and event view (Section IV-A), (B) the branch view (Section IV-B), and (C) different detail views (Section IV-C). The module and event view and the branch view on the left side provide an overview of the workflow evolution and allow users to select and filter parts of it. The selected time span (gray area in the module and event view) can then be analyzed in detail on the right side with different detail views. Global settings and operations can be applied with the buttons in (D). The context menu in (E) appears when the mouse pointer hovers the category list and is used to (un)group modules and functions for the module and event view. (F) is the legend for the event symbols and allows us to filter them. The different views for (C) are selected with the buttons in (G).

cies between such modules. Even if there are branches in the workflow and modules are connected to several other modules, the overall structure is usually quite sequential and similar to software pipelines. Furthermore, every module is employed for a single operation only. In the case that a module provides different operations, multiple instances have to be created in order to apply the different operations.

In typical applications, the developer and the user of the module workflow is the same person. Since the aim of this person is to generate results and not software, we use the term *user* in the remainder of this paper. Furthermore, many systems allow the user to create the module workflow with a graphical tool, e.g., modules can be placed on the work space and connections can be drawn between them (see Figure 1).

Another important aspect is that all relevant parameters are set before the workflow is executed, i.e., setting parameters is also part of modifying the workflow. This leads in many cases to repeated modifications of the workflow, e.g., when the parameter space of an operation is explored.

Finally, the use of undo operations and the reuse of previously created workflows lead to branches in the workflow evolution.

## IV. Visualization Technique

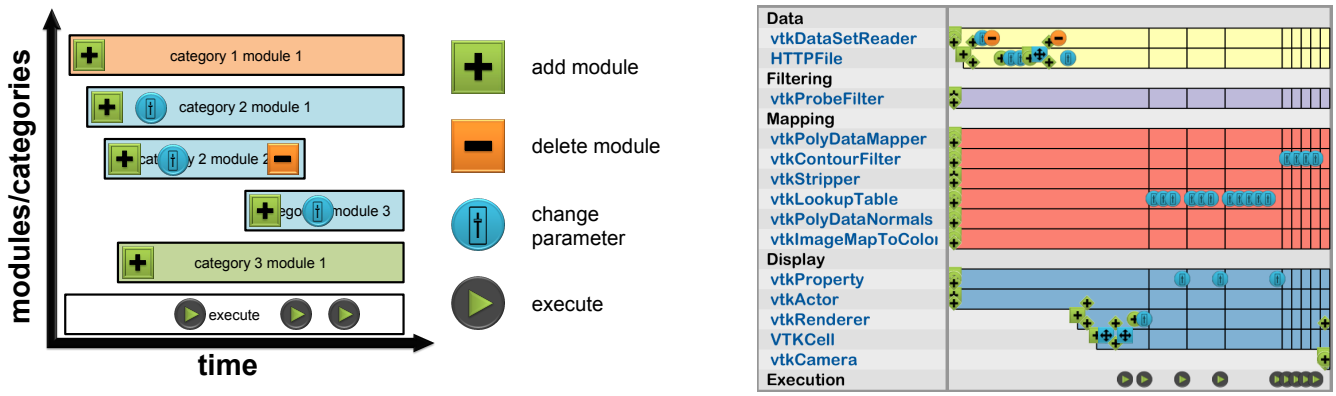
The visualization approach we introduce in this paper depicts the evolution of workflows recorded for individual usage sessions of a modular software framework. Our visualization setup uses multiple coordinated views and consists of three main elements as shown in Figure 2:

1) Visualization of the module lifetime and module-related events on a timeline (Section IV-A)
2) Visualization of branches in the workflow evolution and their relationship (Section IV-B)
3) Different views that allow a detailed analysis of certain aspects of the workflow (Section IV-C)

The combination of these views is important to support the analysis of large workflow histories. The branch view provides an overview of all branches in the workflow history and allows for selecting single branches or a sequence of connected branches (see Section IV-B). The module and event view shows then only the selected part of the workflow history (see Section IV-A). Since individual branches can still cover a large time range, the detail view is used to further drill down to a finer scale (see Section IV-C). The detail views additionally help analyze sequences of events, module parameter changes, and changes of the workflow layout.

### A. Module and Event View

For the visualization of module lifetimes and events, central aspects are the representation of time and time spans. While events, e.g., adding modules or changing parameters, occur at specific points in time, the existence of modules during workflow evolution is associated with time spans. In our case,




(a) (b)

Fig. 3. The concept behind our module and event view. We visualize the evolution of workflows by displaying the time spans of modules used and events related to them. The basic scheme is illustrated in (a); (b) shows how we realized it in our implementation. The time spans of modules are shown as bars, where time is on the x-axis and the different modules are on the y-axis. Vertical lines on the bars indicate the beginning of a new branch. Events are represented by symbols and are superimposed on the bars—the position of a symbol marks the time of the corresponding event and the related module. Categories are used to group modules, allowing users to drill down to an appropriate level of granularity. Color coding and the positions of the modules indicate to which category they belong. Since the execute event operates always on all modules, it has its own category.

the time span of a module corresponds to the period between adding and deleting it. To account for these aspects, our visualization consists of two key elements: representation of time spans and representation of events (Figure 3).

First, bars are used to represent the time spans of the modules in the workflow (Figure 3(a)). This choice is inspired by Gantt charts [32], which are effective for visualizing the time spans of tasks. They allow us to easily assess temporal extents and to compare them.

Since module workflows may contain a large number of modules, we cluster them into a set of categories to improve scalability. The time spans of the categories correspond to the aggregated time spans of the modules assigned to the categories. The categories can be expanded to display the modules represented by them (in our implementation with a context menu, see Figure 2). Furthermore, modules can also be displayed isolated or hidden to focus on a specific subset. In our case study (Section V), we analyze workflows from the visualization framework VisTrails. We therefore use a simple classification scheme for the modules with four categories that reflect high-level tasks and are inspired by the visualization pipeline proposed by Haber and McNabb [40] : "data", "filtering", "mapping", and "display" (Figure 3(b)).

Second, since events are associated with a specific point in time, we represent them with symbols (Figure 3(a)). By overlaying the bars of the module chart with these symbols, we maintain their context both with respect to time and related modules—it is easy to see when events occur and to which modules they are related.

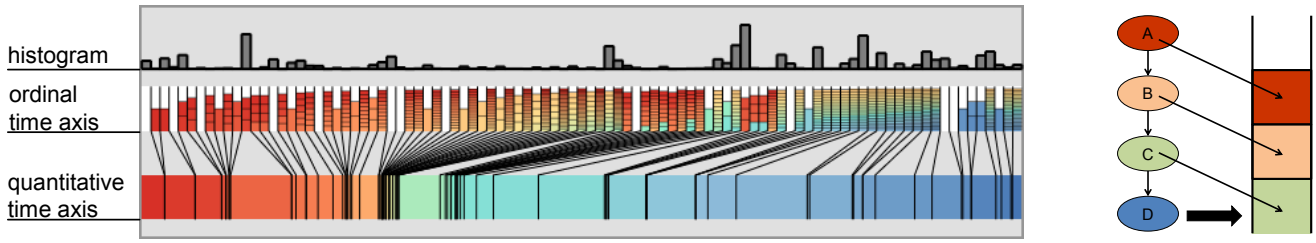
To ease the interpretation, a legend for the event symbols is shown below the module and event view (see Figure 2). There are four types of events: add, delete, change, and execution events. The execution event is special because it represents the execution of the workflow and therefore operates on all modules and functions. Thus, we use a separate category for it; otherwise, the execution symbol would appear on all rows.

Displaying simultaneously a large number of events is problematic. Therefore, we integrated the option to filter events and show only the ones of interest, e.g., parameter change events. Furthermore, the symbols for different types of events were designed with different shape, color, and slight changes in position. For example, events affecting directly a module (add, delete, change position) are squares and slightly shifted to the top. Add events are green, delete events orange, and change events blue. The goal is that they can be still differentiated even when they overlap to a certain degree (see Figure 3(b)).

The combination of the visualization techniques for time spans and events is important. Displaying only the time span of modules would neglect events that do not have a direct influence on the time span of modules, e.g., changing parameters. In contrast, the symbol representation fully represents the evolution of the workflow because it provides a direct visualization of all user actions. However, deducing the set of active modules would require us to follow and memorize all previous events. The combined visualizations reveal the currently active modules and events.

To provide more information about the workflow structure, we considered visually connecting the modules, e.g., by drawing lines between the modules. However, visual connections clutter the visualization and increase the visual load. We therefore decided to omit module connections at the cost of losing information about data flow. We think that module categories and names already provide a good-enough impression of the purpose of the workflow. Furthermore, the creation and deletion of connections is displayed as events. However, there might be applications where the order of module connections has a huge influence on the outcome of the workflow; in the future, we intend to explore alternatives for displaying connectivity information.

In the case of large workflow histories, our proposed module




(a) (b)

Fig. 4. Visualizing branches in the evolution of a workflow. (a) Our branch representation consists of multiple components: On the ordinal time axis in the center, the branches are shown as equally sized columns, neglecting their unequal time spans. The blocks inside these columns represent all parents of the branch. The concept for this is illustrated in (b): All parent branches (A–C) of branch (D) are shown as small blocks inside the column of branch (D). The color of the blocks represents the time stamp of the parents. The respective color map is shown on the quantitative time axis at the bottom; white indicates that there are no further parents. The quantitative time axis additionally shows the actual time spans of the branches with the connected lines. The histogram at the top indicates the number of events in the branches shown on the ordinal time axis below.

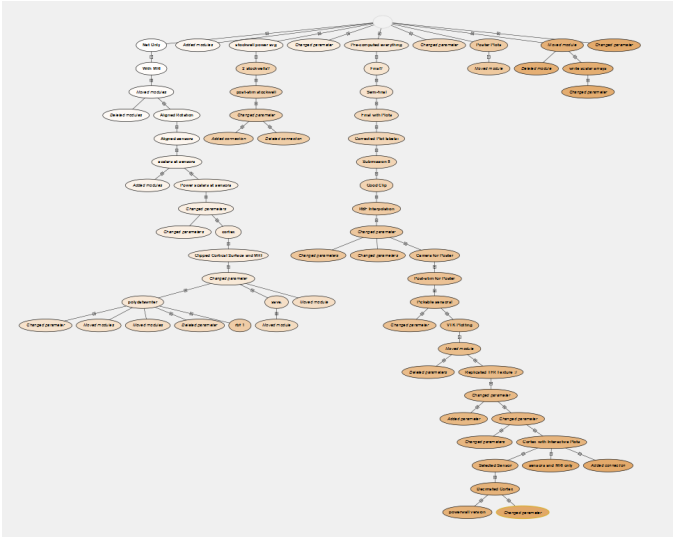


Fig. 5. The history view of VisTrails is an example of representing the branches of a workflow as a tree. Here, the saturation of the nodes denotes time and the edges can be expanded to show all events between two nodes.

and event view can exhibit visual clutter due to displaying a large number of overlapping symbols. Furthermore, displaying events in a linear fashion neglects the occurrence of branches, which appear when users continue from a previous state, e.g., by using undo functionality. If branching occurs, two events that follow each other in a temporal sense can operate on totally different states of the workflow. To address this problem, we developed an additional visual representation of branches that can also be used for selecting them.

### *B. Visualization of Branches*

The second element of our visualization is a visual representation of branches in the workflow evolution (Figure 4). The term branches is used here with respect to the causality of events, i.e., they occur when users return to previous states of the workflow and continue their work from there. Hence, while all events in a workflow history form a linear sequence with respect to the time they appeared, the causal relationship between them forms a tree structure because multiple sequences of events can evolve from a single state.

In the design of the branch view, we considered two key issues: conveying the temporal order of the branches and scalability. Since the branch view is displayed together with the module and event view (see Figure 2), it should exhibit a small spatial footprint.

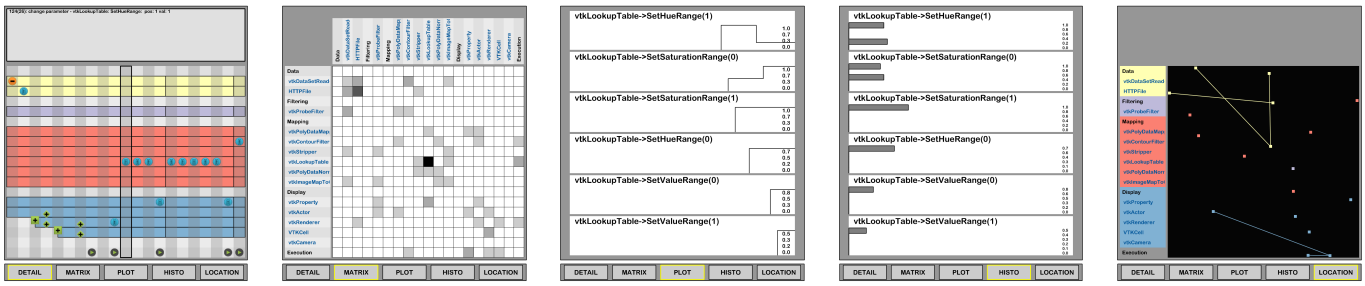
A common approach to visualize branches—e.g., implemented in VisTrails—is a node-link representation of the branch tree (Figure 5). Although this representation well conveys the relation between branches, it typically requires some space and the temporal order of branches is difficult to obtain. We therefore decided to use a different approach.

While we are primarily interested in the temporal order of branches, it should still be possible to retrieve the relationship of branches. Our branch visualization (Figure 4) is therefore divided into three parts: an ordinal time axis, a quantitative time axis, and a histogram.

The ordinal time axis shows the branches with equally sized columns to provide good scalability with respect to the dynamic range of different time spans. These columns are further divided to represent causal relationships to previous branches (Figure 4(b)). We use blocks inside the columns to represent these relationships. Each block represents a parent branch, i.e., the stacking of the blocks represents the causal relationship of the branches. The color of the blocks indicate the time stamps of the parent branches. This approach provides the user with information about the number of parent branches and their temporal occurrence.

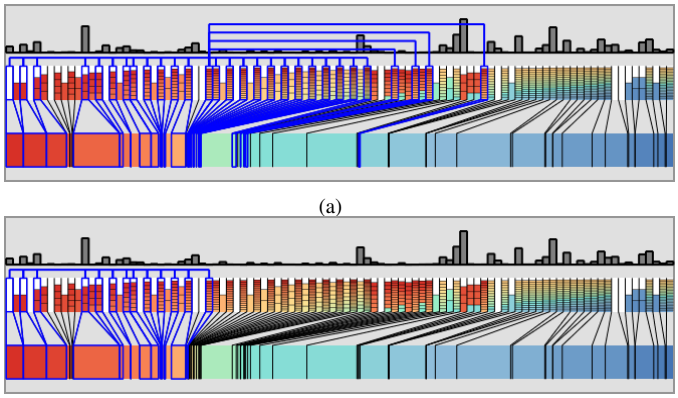
To also provide a quantitative representation, we additionally show the time spans of the branches on a quantitative time axis at the bottom. This axis is also used to display the color map for the time stamps of the parent branches in the ordinal axis. The combination of both axes provides good scalability with respect to branches of very different length in time without discarding their quantitative relation.

Finally, a histogram at the top displays the number of events in the branches shown on the ordinal time axis underneath. Please note that the histogram entries do not necessarily correlate with the time spans of the branches. Since some




(a) (b) (c) (d) (e)

Fig. 6. Different views revealing specific details of the workflow. (a) Detailed view of selected time range, (b) transition matrix for events, (c) plot and (d) histogram for parameter values, and (e) visualization of the module positions in the graphical workflow editor.



(a) (b)

Fig. 7. Enhanced branch view. (a) To provide a clear view on the relationship of branches, all connected branches can be shown with an orthogonal tree visualization on top of the branch representation (blue lines) when the user hovers over the branch with the mouse. (b) To reduce visual clutter, only the parents of a branch can be highlighted.

operations apply to multiple modules (e.g., pasting or moving a group of modules), several events may occur at the same point in time; a branch covering a smaller time span may contain more events than a temporally longer one. When events are filtered out (see Section IV-A), the histogram is adapted accordingly.

This visualization provides only an initial overview of the temporal aspects of the branches. It does not directly show which branches are connected. Therefore, the user can additionally show all connected branches of a single branch in a tree visualization (Figure 7(a)), displayed when the mouse hovers over a branch. To reduce visual load, displaying related child branches can be deactivated (Figure 7(b)). The parent branches are typically more interesting because the current state of the workflow is a result of them.

### C. Detail Views

To support a detailed view without losing context and to further improve the scalability with respect to the length of the workflow history, we developed several detail views placed on the right side of our visualization setup (Figure 2). The default view (Figure 6(a)) acts as a lens of the event view, i.e., it zooms in and magnifies the selected time range. Additionally, the info box at the top provides detailed information about single events. The second view (Figure 6(b)) is a matrix that shows the amount of transitions between two modules with respect to subsequent events. If there are two subsequent events, with the first operating on module A and the second on module B, the respective entry in the matrix for these two modules is increased. Next, the plot view (Figure 6(c)) displays a plot over the selected time range of the parameters for all expanded functions of the modules. As an alternative view, a histogram (Figure 6(d)) can also be shown for these parameters. In contrast to a classic histogram, not the recurrence of values is counted, but the accumulated time range of the respective value range is shown. The last view is a plot of the locations the modules have in a graphical workflow editor (Figure 6(e)); the purpose of this view is mainly to let the user analyze usability issues with respect to placement of modules. All views relate to a user-selected area in the module and event view. Filtering events also affects all views. For instance, the transition matrix considers only parameter change events when only this type of events is selected.

## V. Case Study: Modular Visualization

To demonstrate our approach, we implemented it to visualize and analyze the evolution of module workflows from VisTrails [3]. VisTrails is a visualization framework that allows one to combine different modules, e.g., from the visualization toolkit VTK [22], to create a custom visualization. The module workflow is also referred to as *pipeline* in VisTrails and can be created in a graphical way (see Figure 1). VisTrails was specifically designed to collect and provide provenance information [41], i.e., all user actions are logged. Hence, it is easy to extract the evolution of the module workflow from a VisTrails dataset.

We implemented the prototype tool in C++ with OpenGL for the graphics part. Since the visualization is not computationally intensive, no special optimizations or implementations, e.g., for GPUs, were required to provide interactivity.

In this section, two different application scenarios are discussed. In the first case (Section V-B), the visualization is used to identify potential bottlenecks and barriers for users of VisTrails. The second one (Section V-C) aims at aiding in the retrospective analysis of visualization sessions.

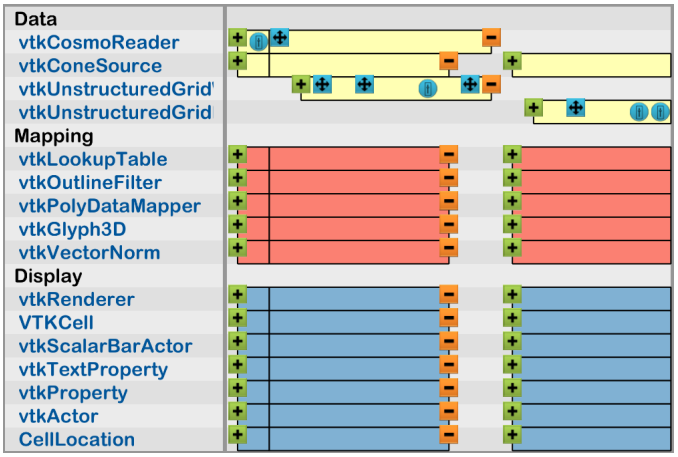



Fig. 8. Two branches of an expert dataset showing redo behavior, i.e., modules are deleted and then inserted again.

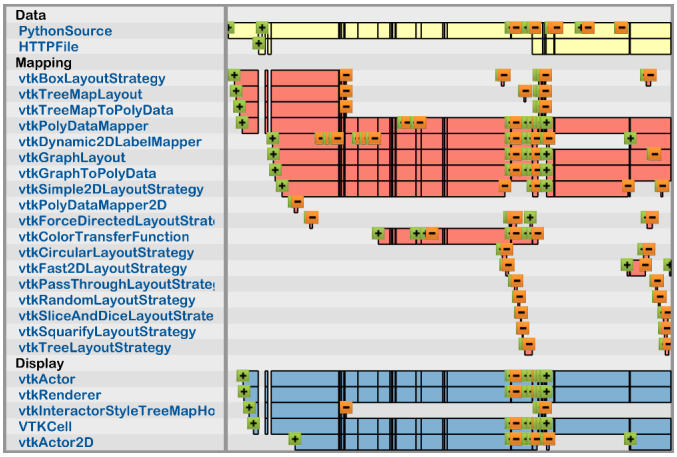



Fig. 9. A student dataset exhibiting many modules with very short lifetime. The module names reveal that they are used to apply different layout strategies.

We used three different types of datasets: First, visualization experts provided us with several VisTrails files with the provenance of their visualization sessions. The second type of dataset contains the provenance of student assignments in a visualization course obtained from Silva et al. [42]. The examples provided with VisTrails are the last type of data; we especially discuss the example visualization of a brain scan[1].

Before we present the application scenarios, we briefly describe in the following a typical example of how our visualization approach can be used for analysis.

### A. Typical Usage Session

After loading the dataset, we initially looked at the event view to obtain a first overview of the data. If there are not too many events in the dataset, we can already recognize interesting details: Since we were mainly interested in the visualization methods used, we expanded the "mapping" category. To reduce visual load, events can be filtered out, e.g., change events related to the module position in the graphical workflow editor are usually less interesting.

To analyze the data in detail, especially if it covered a long time range, looking at the branch view helped us to select interesting parts of it. For example, branches with no or only one parent might be of interest (visible as white columns or columns with a single colored block inside, see Figure 4) because they indicate that the user started something new or continued from an earlier stage. If the number of modules is high, it may help reduce it by collapsing categories or removing uninteresting modules. After selecting a branch, with or without all related parents, we browsed through the data by moving the selection area and followed the individual steps of the visualization session. It is often interesting to look at parameter changes. For this, we expanded the functions of interesting modules. We also used the plot and histogram views to follow the parameter changes.

[1]This dataset (brain_vistrail.vt) is included in the VisTrails examples. By downloading VisTrails, the reader may follow this example in VisTrails.

### B. Usability Issues

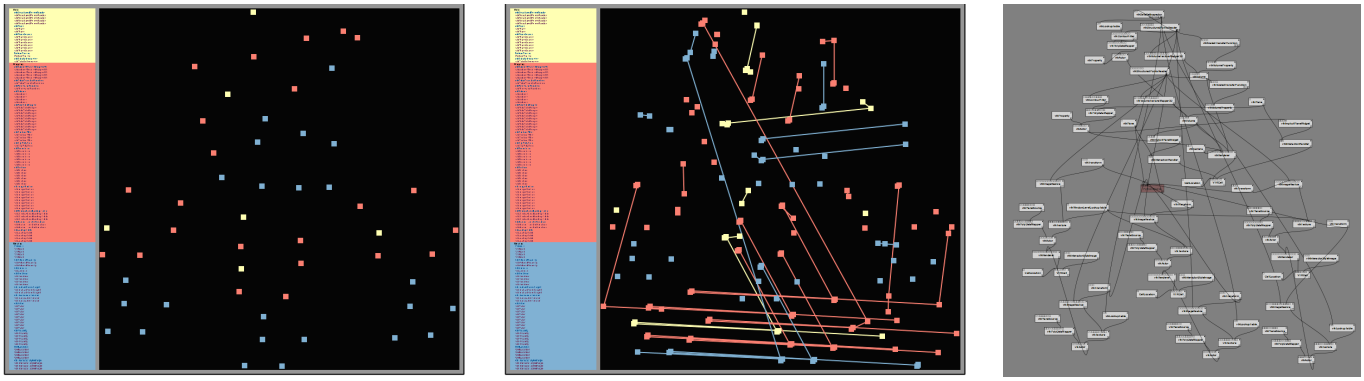
In this scenario, we explored the student and expert datasets at a coarse level. We were mainly interested in anomalies and unexpected patterns. Every dataset was browsed rather fast, for no longer than 5 minutes per dataset. But even this relatively rough exploration yielded some interesting discoveries.

Figure 8 shows a part of an expert dataset. In this example, a large number of modules were pasted, deleted, and afterward pasted again. This is visible through bars starting at the same time and also through the aligned symbols for module creation. It was presumably easier for the user to delete all modules and paste them again instead of using the undo capabilities of VisTrails to return to a previous state.

The second example (Figure 9) shows many modules that exist only for a short duration. An interpretation for this could be that the user tried out different modules and was not able to use them for the task. However, looking at the module names reveals that every module provides a different layout method for graph or tree visualizations. Hence, the user had to repeatedly create and connect modules to test different layouts.

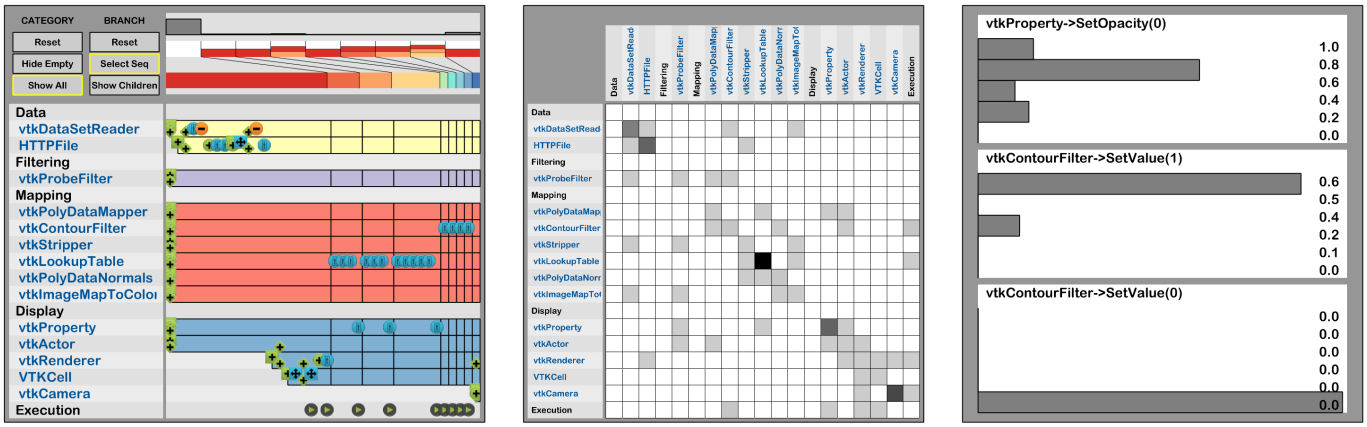
In the third case (Figure 10), we looked at the positions that modules have in the workflow editor of VisTrails. In the beginning, a large group of modules was pasted. Their positions are visible in Figure 10(a). There seems to be no overall structure in their placement, the color coding reveals that modules of all categories are mixed. Looking at the full time range of the visualization session (Figure 10(b)), additional modules appeared and some of them were moved. Still, no clear structure is visible in this workflow with a relatively large number of modules (Figure 10(c)). Hence, finding specific modules for modifications can be difficult. Of course, it is very subjective what is perceived as a good structure. Nevertheless, this result can be an indication that the user should be better supported by VisTrails in keeping the workflow structured.

In addition to the presented observations, we noticed a significant number of paste operations in our explorations. This



(a) (b) (c)

Fig. 10. Analysis of module locations in an expert dataset. Some patterns can be seen in the lower part of the location plot for their initial creation. (a) Three similar groups occur there. However, an overall structure where the modules are grouped according to their category cannot be seen. (b) The location plot for the full time range confirms this. Only some modules were moved in groups, visible in the parallel lines of their movement. (c) The complexity is also visible in the pipeline view of VisTrails.




(a) (b) (c)

Fig. 11. Retracing the visualization of the brain dataset. (a) The initial overview already reveals the basic structure of the way the visualization was created. Because of the small number of different modules and events, many details can be seen without filtering. (b) The transition matrix shows typical sequences of events for this module workflow. (c) The parameter histogram shows the value range of different module parameters. Both additional views were computed for the total time range of the data.

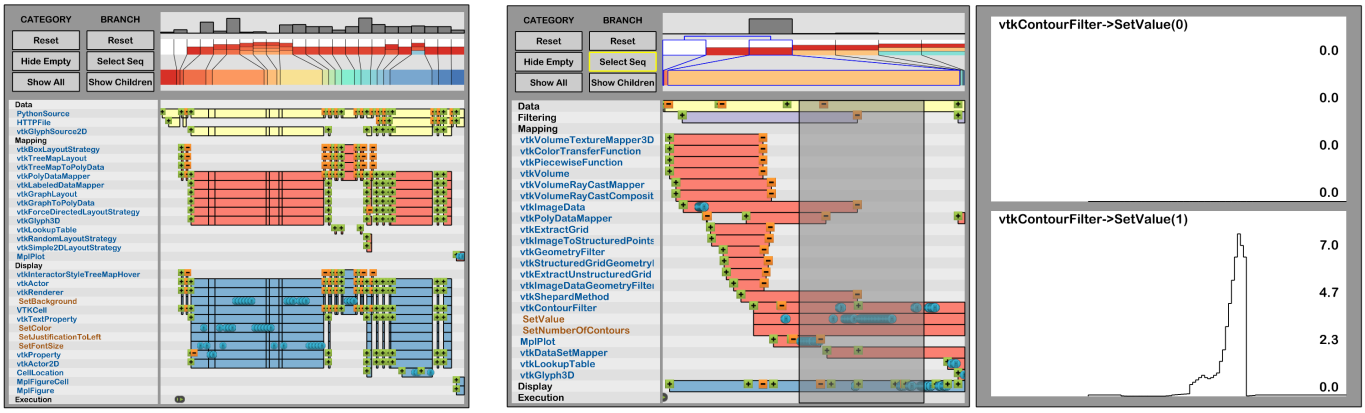
seems to be independent from the experience of the user. Both experts and students pasted groups of modules several times during a typical usage session.

### C. Retrospective Analysis

In this scenario, we conducted a detailed analysis of a single visualization session to understand how the respective visualization workflow was created. Results from the exploration of the brain dataset provided with VisTrails are shown in Figure 11. The overview (Figure 11(a)) shows that the workflow was not built from scratch but most of the modules were initially copied from another workflow. The creation of the visualization seems to be well structured. First, data sources were set up, then, general display settings were applied, visible as parameter changing events. Later occurring branches deal only with parameter changes of the visualization method. Being familiar with VTK, we can see that only isosurface visualizations were used ("vtkContourFilter" module).

Furthermore, patterns in the parameter changes are visible. First, changes of color and opacity were performed ("vtkLookupTable" and "vtkProperty"). This involved similar sequences of events on both modules, which is clearly visible in the transition matrix representation (Figure 11(b)). We can also see in the matrix that the workflow was executed after changes on four different modules. Toward the end, the isovalue was changed several times ("vtkContourFilter"). The value range of the parameter changes are provided in the histogram view (Figure 11(c)), e.g., different opacity values between 0.2 and 1.0 were used. Furthermore, the opacity had a value around 0.8 for most of the time.

The next example (Figure 12(a)) exhibits interesting patterns in the branch view. There are many branches with no or a single parent at the second half of the time range. This means that the work was restarted several times from an early stage of the workflow. Looking at the events reveals additional interesting details, especially when only events for adding and deleting modules, and changing parameters are displayed.




(a) (b)

Fig. 12. Two examples from the student data. (a) This usage session exhibits many branches connected to the two initial branches and without any children. (b) The second student worked mainly in a single branch, in which he tried out several methods.

Beside the fact that the student used many pasting operations, it is interesting that parameter changes occur mainly in the beginning of the visualization session. If we then look at the functions affected by changes, we can see that the student spent most of the time changing the background color or the size and color of the used font. In some cases, this may be enough to create a good visualization. However, together with the repeated pasting of large module groups, this can also be an indication of cheating, i.e., solutions might have been copied from workflows of other students.

The student in the last example (Figure 12(b)) worked mainly in a linear way without branching. He first used volume rendering techniques ("vtkVolume..." modules) and then isosurface visualization ("vtkContourFilter"). Looking at the parameter plot for the "SetValue" function of the isosurface module, we notice that the student put considerable effort into finding a suitable parameter. However, he used the initial value around zero at the end indicating that he possibly failed in finding a suitable parameter for his visualization.

### D. Summary

Some general observations can be derived from our examples. Looking at the way users create the workflows might help improve the usability of the workflow system (Section V-B). For example, we have seen that copying a group of modules is often used. The large number of events that change the module positions in the third example suggests that a useful improvement would be to provide automatic layout mechanisms when users add and connect modules.

There are various applications for a detailed retrospective analysis of a user session (Section V-C). One aspect is to understand or recall previous steps when continuing the work with a module workflow. This could be done for one's own work, but it can also help in collaborative projects, understanding the work of the collaboration partners. The workflow history also exposes what methods were used and which parameters were changed to obtain the final result. It might help characterize sessions (or parts of sessions) with respect to the nature of the actions performed, for example, phases of the session that consisted mostly of parameter tweaking as opposed to workflow construction. Finally, a detailed exploration can point to characteristic usage patterns or commonly used parameters. This information may help improve the handling of the workflow system, e.g., by offering commonly used parameters in a special menu or widget.

## VI. Conclusion and Future Work

We have presented visual representations that help analyze the evolution of module workflows. The proposed combination of visualization techniques focuses on the temporal aspects of the modules and related events. Our branch view provides scalability with respect to the size and dynamic range of the time range. Additional views enable a detailed exploration of events. The hierarchical representation of modules allows the exploration of workflows with a large number of modules. As discussed in the case study, the insights gained with our visualization are useful in different application scenarios: from identifying potential improvements for the underlying system to the retrospective analysis of previous application sessions.

Although we demonstrated our approach for module workflows in VisTrails, the visualized elements—time spans of modules, occurrence of different events, changes of parameters—are part of workflows in general. Hence, besides loading the datasets, there are no elements in our implementation that were specifically created for VisTrails. We therefore think that our approach can be easily transferred to other workflow systems (see Section II).

There are still open problems that we intend to pursue in future work. While our techniques have been designed to analyze individual sessions, it is also of interest to consider multiple sessions simultaneously. Furthermore, the scalability of the approach with respect to very large datasets and big data applications should be evaluated and possibly improved.

In addition, our approach can be enhanced in an application-specific way. Currently, the semantics behind events are visualized only at a very basic level. The visualization does not

directly convey which aspects of the data are affected by a parameter change and what the effect on the result is.

## ACKNOWLEDGMENT

This work was partially supported by the German Research Foundation (DFG) within the Cluster of Excellence in Simulation Technology (EXC 310) at the University of Stuttgart.

## REFERENCES

[1] A. Barker and J. Van Hemert, "Scientific workflow: A survey and research directions," in *Proceedings of the 7th International Conference on Parallel Processing and Applied Mathematics*, 2008, pp. 746–753.

[2] W. M. Aalst, A. H. Hofstede, and M. Weske, "Business process management: A survey," in *Business Process Management*, ser. Lecture Notes in Computer Science, W. Aalst and M. Weske, Eds. Springer, 2003, vol. 2678, pp. 1–12.

[3] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. T. Vo, "Managing the evolution of dataflows with VisTrails," in *Proceedings of 22nd International Conference on Data Engineering Workshops*, 2006.

[4] L. Voinea, A. Telea, and J. J. van Wijk, "CVSscan: visualization of code evolution," in *Proceedings of the ACM Symposium on Software Visualization*, 2005, pp. 47–56.

[5] L. Voinea and A. Telea, "CVSgrab: Mining the history of large software projects," in *Proceedings of the Joint Eurographics – IEEE VGTC Symposium on Visualization*, 2006, pp. 187–194.

[6] C. Müller, G. Reina, M. Burch, and D. Weiskopf, "Subversion statistics sifter," in *Proceedings of the International Symposium on Visual Computing*, 2010, pp. 447–457.

[7] M. Ogawa and K.-L. Ma, "StarGate: A unified, interactive visualization of software projects," in *Proceedings of the IEEE Pacific Visualization Symposium*, 2008, pp. 191–198.

[8] M. Ogawa and K. L. Ma, "code_swarm: A design study in organic software visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1097–1104, 2009.

[9] M. Lanza, "The Evolution Matrix: Recovering software evolution using software visualization techniques," in *Proceedings of the 4th International Workshop on Principles of Software Evolution*, 2001, pp. 37–42.

[10] M. Pinzger, H. Gall, M. Fischer, and M. Lanza, "Visualizing multiple evolution metrics," in *Proceedings of the ACM Symposium on Software Visualization*, 2005, pp. 67–75.

[11] S. Diehl, *Software Visualization—Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, 2007.

[12] P. Caserta and O. Zendra, "Visualization of the static aspects of software: A survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 7, pp. 913–933, 2010.

[13] S. da Cruz, M. Campos, and M. Mattoso, "Towards a taxonomy of provenance in scientific workflow management systems," in *Proceedings of World Conference on Services – I*, 2009, pp. 259–266.

[14] S. B. Davidson and J. Freire, "Provenance and scientific workflows: Challenges and opportunities," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2008, pp. 1345–1350.

[15] P. Macko and M. Seltzer, "Provenance Map Orbiter: Interactive exploration of large provenance graphs," in *Proceedings of the Third Workshop on the Theory and Practice of Provenance*, 2011, pp. 292–301.

[16] N. Rio and P. Silva, "Probe-it! Visualization support for provenance," in *Advances in Visual Computing*, ser. Lecture Notes in Computer Science. Springer, 2007, vol. 4842, pp. 732–741.

[17] M. Anand, S. Bowers, and B. Ludascher, "Provenance browser: Displaying and querying scientific workflow provenance graphs," in *Proceedings of the IEEE International Conference on Data Engineering*, 2010, pp. 1201–1204.

[18] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: an extensible system for design and execution of scientific workflows," in *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, 2004, pp. 423–424.

[19] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[20] J. Goecks, A. Nekrutenko, and J. Taylor, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biology*, vol. 11, no. 8, pp. 1–13, 2010.

[21] C. Upson, J. Faulhaber, T.A., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam, "The application visualization system: a computational environment for scientific visualization," *IEEE Computer Graphics and Applications*, vol. 9, no. 4, pp. 30–42, 1989.

[22] W. Schroeder, K. M. Martin, and W. E. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics (2nd ed.)*. Prentice-Hall, Inc., 1998.

[23] F. Ritter, T. Boskamp, A. Homeyer, H. Laue, M. Schwier, F. Link, and H.-O. Peitgen, "Medical image analysis," *IEEE Pulse*, vol. 2, no. 6, pp. 60–70, 2011.

[24] J.-D. Fekete, "The InfoVis Toolkit," in *Proceedings of the IEEE Symposium on Information Visualization*, 2004, pp. 167–174.

[25] J. Heer, S. K. Card, and J. Landay, "Prefuse: A toolkit for interactive information visualization," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2005, pp. 421–430.

[26] W. Aigner, S. Miksch, H. Schumann, and C. Tominski, *Visualization of Time-Oriented Data*. Springer, 2011.

[27] E. R. Tufte, *The Visual Display of Quantitative Information*. Graphics Press, 1986.

[28] L. Chittaro and C. Combi, "Visualizing queries on databases of temporal histories: new metaphors and their evaluation," *Data & Knowledge Engineering*, vol. 44, no. 2, pp. 239–264, 2003.

[29] W. Aigner, S. Miksch, B. Thurnher, and S. Biffl, "PlanningLines: novel glyphs for representing temporal uncertainties and their evaluation," in *Proceedings of the Ninth International Conference on Information Visualisation*, 2005, pp. 457–463.

[30] R. Kosara and S. Miksch, "Metaphors of movement: a visualization and user interface for time-oriented, skeletal plans," *Artificial Intelligence in Medicine*, vol. 22, no. 2, pp. 111–131, 2001.

[31] C. Plaisant, B. Milash, A. Rose, S. Widoff, and B. Shneiderman, "LifeLines: visualizing personal histories," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1996, pp. 221–227.

[32] H. L. Gantt, *Work, Wages, and Profits*. Hive Pub. Co., 1913.

[33] F. B. Viégas, M. Wattenberg, and K. Dave, "Studying cooperation and conflict between authors with history flow visualizations," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2004, pp. 575–582.

[34] J. Waser, R. Fuchs, H. Ribičić, B. Schindler, G. Blöschl, and M. E. Gröller, "World Lines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1458–1467, 2010.

[35] D. Hackenberg, G. Juckeland, and H. Brunst, "High resolution program flow visualization of hardware accelerated hybrid multi-core applications," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 786–791.

[36] J. Trümper, J. Bohnet, and J. Döllner, "Understanding complex multithreaded software systems by using trace visualization," in *Proceedings of the ACM Symposium on Software Visualization*, 2010, pp. 133–142.

[37] M. Burch, F. Beck, and S. Diehl, "Timeline Trees: visualizing sequences of transactions in information hierarchies," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, 2008, pp. 75–82.

[38] T. Wang, C. Plaisant, B. Shneiderman, N. Spring, D. Roseman, G. Marchand, V. Mukherjee, and M. Smith, "Temporal summaries: Supporting temporal categorical searching, aggregation and comparison," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1049–1056, 2009.

[39] S. B. Cousins and M. G. Kahn, "The visual display of temporal information," *Artificial Intelligence in Medicine*, vol. 3, no. 6, pp. 341–357, 1991.

[40] R. B. Haber and D. A. McNabb, "Visualization idioms: A conceptual model for scientific visualization systems," in *Visualization in Scientific Computing*, B. Shriver, G. Nielson, and L. Rosenblum, Eds. IEEE Computer Society Press, 1990, pp. 74–93.

[41] C. Silva, J. Freire, and S. Callahan, "Provenance for visualizations: Reproducibility and beyond," *Computing in Science and Engineering*, vol. 9, no. 5, pp. 82–89, 2007.

[42] C. Silva, E. Anderson, E. Santos, and J. Freire, "Using VisTrails and provenance for teaching scientific visualization," *Computer Graphics Forum*, vol. 30, no. 1, pp. 75–84, 2011.