2014 IEEE 10th International Conference on eScience

Experiences with User-Centered Design for the Tigres Workflow API

Lavanya Ramakrishnan, Sarah Poon, Valerie Hendrix,
Daniel Gunter, Gilberto Z. Pastorello, Deborah Agarwal
Lawrence Berkeley National Lab
Email:{lramakrishnan, sspoon, gzpastorello, dkgunter, vchendrix, daagarwal}@lbl.gov

***Abstract*—Scientific data volumes have been growing exponentially. This has resulted in the need for new tools that enable users to operate on and analyze data. Cyberinfrastructure tools, including workflow tools, that have been developed in the last few years has often fallen short of user needs and suffered from lack of wider adoption. User-centered Design (UCD) process has been used as an effective approach to develop usable software with high adoption rates. However, UCD has largely been applied for user-interfaces and there has been limited work in applying UCD to application program interfaces and cyberinfrastructure tools. We use an adapted version of UCD that we refer to as Scientist-Centered Design (SCD) to engage with users in the design and development of Tigres, a workflow application programming interface. Tigres provides a simple set of programming templates (e.g., sequence, parallel, split, merge) that can be can used to compose and execute computational and data transformation pipelines. In this paper, we describe Tigres and discuss our experiences with the use of UCD for the inital development of Tigres. Our experience-to-date is that the UCD process not only resulted in better requirements gathering but also heavily influenced the architecture design and implementation details. User engagement during the development of tools such as Tigres is critical to ensure usability and increase adoption.**

I. Introduction

The data being produced at experiment facilities and within other large science collaborations is at a size and complexity that requires analyses to occur near the data. This change threatens to make data analysis at these high-performance computing (HPC) facilities the domain of a few experts. Cyberinfrastructure tools and more specifically Workflow tools enable easy access to analysis for all collaborators.

Many excellent workflow and scripting systems have been built in the past to enable capture of high-level scientific computing flows. However, the adoption of these tools has been significantly less wide-spread than hoped and has resulted in a large plethora of tools available online. Workflow tools adoption have suffered from complex interfaces, the need to learn new languages, the difficulty with running persistent services (i.e., workflow engine) at HPC centers. Thus, we believe that we need a different approach to addressing the usability and technical challenges of developing a workflow tool.

The Tigres (Template Interfaces for Agile Parallel Data-Intensive Science) project attempts to address these problems with two innovative methods in workflow design. First, we believe that a user-centered design (UCD) process is essential to actively engage the user and incorporate their feedback in the workflow tool development. Second, Tigres provides a programming library for workflow tools that can easily integrate with current scientist tools and does not run or rely on persistent services for execution. In recent years, the need for active engagement of users in the development of eScience tools has been recognized [4]. User-centered design processes have been heavily used in eScience projects for user-interfaces [5]. However, there has been no prescribed methodology on the user engagement process for e-Science application programming interface design and development.

In this paper, we describe the use of user-centered design process in the initial design and development of the Tigres project. Specifically, in this paper, we make three contributions.

- We describe the process that we used in Tigres for user-centered design for APIs
- We describe the impact of the process on the design and implementation choices in Tigres.
- We outline guidelines for conducting the initial user-centered design process for similar software tools and APIs.

The rest of the paper is organized as follows. We discuss related work in Section II. We describe the UCD process we use in Tigres in Section III. We describe the results and impact on the Tigres architecture and implementation in Section IV. In section V, we discuss the implications of using user-centered design process and present our conclusions and future work in Section VI.

II. Related Work

In this section, we review related work in user-centered design for APIs, workflow tools, programming models and tools.

User Centered Design for APIs. Little research has been done on the process of API design itself. In 2006, Joshua Bloch proselytized the importance of good APIs [8], providing general guidelines for API design. Since that time, others have written about the application of API design to UI components [20] and Web interfaces [24], which generally agree with Bloch's guidelines. In addition, there have been a few studies that focused on testing and evaluating the usability of APIs [31], [32]. Steven Clarke advocated the employment of User-Centered Design techniques and introduced a "cognitive dimensions framework" to make quantitative comparisons between alternative designs [13]. In addition to laboratory usability tests, Farooq and Zirkler employed the use of API peer reviews, which are light weight, group based usability

978-1-4799-4287-9 2014
U.S. Government Work Not Protected by U.S. Copyright
DOI 10.1109/eScience.2014.56

evaluations performed by other API developers [19]. Although we appreciate and use Bloch's guidelines, we are focused, like Clarke, on evaluating the API directly with the users. Our study is distinct from previous research in that it applies User-Centered Design techniques to the design of Tigres API.

In 2009, several researchers in API usability organized a workshop at CHI (ACM Conference on Human Factors in Computing Systems), the summary of which showed that much work is still needed in understanding the types of APIs and the use cases involved [16]. The (relatively rare) software engineering books on API design focus on general and language-specific advice for the design process and not on evaluating usability [29], [33].

Workflow tools. Workflow tools have been developed to represent and run scientific processes in a distributed environment. In the last decade, various workflow tools such as Galaxy[22], Kepler [3], Taverna [26], Pegasus [18], Triana [12], Swift [36], Trident [7], Makeflow [10] have been developed, that allow users to compose their applications and services into a logical sequence. The Condor DAGMan [14] uses the graph representation to manage dependencies between jobs and hence acts as a meta-scheduler for jobs submitted to a Condor system.

CloudWF [38] and Oozie [1] are workflow systems for cloud that are built on top of Hadoop. These tools enable users to chain multiple MapReduce jobs but do not support any additional patterns. Spark [37] framework introduces a data abstraction called resilient distributed datasets to address a large set of programming patterns while maintaining the scalability and fault-tolerance properties of MapReduce.

Tigres differs from these tools as it is a workflow library rather than a workflow system. Tigres provides a template abstraction in existing programming languages and uses a scientist-centered design process to track the design of the API.

Programming models and tools. The idea of capturing common programming patterns itself is not new to scientific and HPC environments. Various works have proposed patterns [35], [6], [11], programming models [23], shared-memory model standards [15]. However, the idea of using templates in a programming API or tool has not been explored before.

MapReduce [21] was introduced by Google to handle the terabytes of data-per-day generated by Internet applications. The MapReduce programming model is based on the idea of achieving linear scalability by using multiple computers. Our earlier evaluation shows that while MapReduce might be a useful model to handle large-scale parallel data analysis of Internet data, it has significant gaps for general scientific workflow [28], [17]. Tigres builds on the ideas provided by the MapReduce programming interface to capture a wider range of scientific workflows.

III. DESIGN PROCESS FOR TIGRES API

Figure 1 illustrates the design process used in Tigres. The process is divided into four stages a) API design b) API implementation c) advanced execution semantics and, d) optimizations. At each stage in the process, we engage with the users to validate and refine the design and implementation as well as set project priorities. In this paper, we describe our

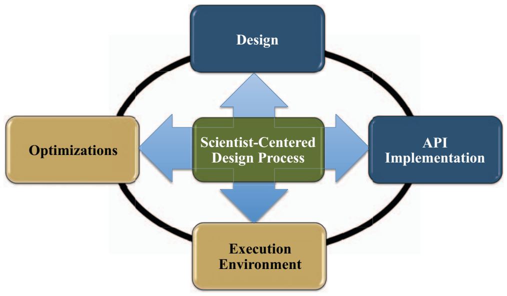


Fig. 1: Tigres Scientist Centered Design Process. In Tigres, we engage with the user at each stage of the tool development.

first usability study and how it influenced the the design and implementation of the API and Execution stages.

The User-Centered Design (UCD) process has emerged as a successful approach to user interface and software development. It is based on a recognition that the usability of the end product is critical to the success of software. UCD accomplishes this by focusing on the needs and goals of the intended users and defining a process by which this can be studied [30]. The formalized UCD process can be summarized in three steps: (a) user needs analysis; (b) design prototyping; and (c) user testing.

The nature of eScience projects (e.g., agile requirements, smaller user teams) makes it difficult to apply the classical UCD approach. We have noted several considerations and challenges in employing the classical UCD approach for scientific applications. For example, the user population for a science application is typically relatively small, often tens and at most thousands, compared to a commercial application. In addition, design of scientific interfaces often requires an immense amount of domain knowledge and must encapsulate goals and work practices that are in flux. As a result, trying to "understand the users" requires more time and resources than is typically practical for science projects, leading to potentially faulty and unreliable user models. We have found that identifying a few key representative users from the scientific community to serve as archetypal users, in lieu of developing classic personas, is the most effective way to inform the design. We work very closely with these domain experts to understand the needs and environment, and develop tight feedback loops with them.

For the last six years, we have been employing the condensed User-Centered Design approach to software development for eScience projects [27]. We refer to our User-Centered Design approach as *Scientist-Centered Design* (SCD). The SCD approach does not use formal user models but key scientific users are identified and actively engaged through a formalized methodology at every stage of the software design and development life cycle. The project has been used in the following projects: the Supernova Factory, Particle Data Group particle information system, Energy Tracker, Knowledge Base (KBase), Carbon Capture Simulation Initiative, Advanced Light Source user portal, and AmeriFlux.

UCD process. The UCD process starts out similar to the traditional software development approach where *user requirements* are gathered, to better understand the typical use scenarios. In the traditional software design process, the requirements drive the design and implementation of a completely functional prototype. Instead, using this methodology, the user requirements drive a conceptual design phase which results in a "non-functional" low-fidelity *design prototype*. The design prototype is used to engage users in a *usability study* for feedback. In designing this low-fidelity prototype, designers may ask scientists to help brainstorm about design considerations and ideas, in a form of participatory design. The prototype is evaluated using usability studies, a methodology where users are asked to engage with the system through a well-defined set of tasks. This activity allows the design team to get valuable *feedback* about usage models and interpretation of functionality without investing too much time in actual development. The feedback from the initial User-Centered Design process is distilled into a list of usability issues and can be prioritized based on Jakob Nielsen's usability severity ratings [25]. This process leads to development of the first functional prototype. The functional prototype is used to engage users to get additional feedback and the process continues. The user engagement in every stage helps the design and development team to translate user requirements to concrete design decisions. The process also helps identify project priorities at every stage in the design and life cycle of the project.

Tigres first usability test. We first piloted the API test using friendly users, two programmers and one web developer. The process of testing the API went through several refinement iterations (discussed in Section **??**). Subsequently, we ran the API tests with three scientists.

The artifacts used for the Scientist-Centered Design process were API documentation and example code snippets. The prototype API was written in a pseudo-code format to enable feedback on style of the interface. Each test participant was given a small workflow problem to solve. The participant was given 15 minutes to review the documents. The workflow problem was broken down into smaller chunks. Each participant was asked to write pseudo-code for the chunk from the workflow problem. We used Google docs, a collaborative document writing tools that allowed us to interact with the participants actively. We asked the participants to "think out loud" while performing the test, offering no guidance to the participants unless they were unable to complete the task. The participants could at any time "compile" and run the code, and the test givers would verbally return back any compilation or run-time errors.

We used two criteria during the tests to measure the experience a) we noted places where the user had difficulty with the tasks and/or understanding the documentation b) we also tracked the total time of the test. Between each test session, we used the results of the test to iterate on the prototype API and refine our testing process. As noted earlier, very little guidance is given in the literature on how to conduct a usability study for APIs. So, we adapted the methods used to test graphical user interfaces to test the API.

At the conclusion of the test, we held follow-up interviews to discuss the API. In all of the usability studies, the test participants felt that they had a good sense for how Tigres works, how the API would be used, and how the Tigres templates might be used in their own work. More importantly, the usability tests provided the right amount of context for the test participants to have more detailed discussions about how they would like to use Tigres in their work.

IV. Tigres

The Tigres project [2] is developing reusable *templates* that enable scientists to easily compose, run, and manage collaborative computational tasks. Tigres provides a simple set of template abstractions for describing scientific analysis pipelines. Designed as a library in existing programming languages, Tigres allows users to easily include the workflow concepts within their programs. The templates present the scientists with constructs that provide easy interfaces to common workflow patterns. Underneath the scripting interfaces Tigres has an ecosystem which transparently invokes needed provenance collection, data access, and data movement optimizations.

Computational pipelines have become an integral part of the scientific discovery process in science domains. We are working with pipelines for image processing (Advanced Light Source experiments and cosmology telescope observations), DNA sequence assembly (Joint Genome Institute), uncertainty quantification (Carbon Capture Simulation Initiative), multi-scale modeling (carbon flux monitoring), dynamic computations (materials discovery and gamma ray background capture).

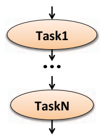
We strongly believe that the key to the broad adoption of Tigres will be design of an API that naturally integrates into the scientific computational pipeline development process and does not pose a high cost of adoption. The Scientist-Centered Design process, is enabling us to work closely with a variety of scientists from different domains to prototype and test our templates and their specification before committing to an implementation. These use cases each pose unique challenges in the area of parallel processing, data management, and provenance and our design process enables us to productively engage with the user groups in the design of the interface.

The user-centered design process has provided substantial advantage to the Tigres project development path. Our first usability studies have significantly impacted the design of Tigres including fundamental decisions such as implementation language, form of the API, priorities, and understanding of the features needed. The process has been vital to deciding project priorities. Additionally, it provided key insights on the process as it needs to be applied to eScience tools. In this section, we summarize in detail the Tigres design and implementation and its context describe our experience with using UCD.

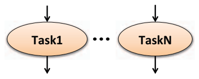
A. Design Goals

The goal of the Tigres project is to develop a light-weight framework that can be used by scientists in their own software implementations. We used the initial interviews in the UCD process to guide our design goals.

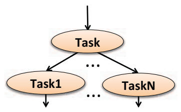
Easy Composition. Typically, workflow tools have specialized languages that often have a learning curve. In contrast, Tigres



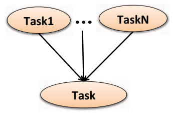
(a) Sequence template



(b) Parallel template



(c) Split template



(d) Merge template

Fig. 2: Execution behavior of the four basic Tigres templates.

is designed to be a library in existing programming languages. The Tigres API is invoked at run-time, similar to MPI (Message Passing Interface). The implementation of the templates needs to handle the work composition and parallelism needs of the scientists, while remaining concise.

Scalability. Tigres needs to handle large-scale data-intensive workflows that are becoming increasingly common in sciences from cosmology to bioinformatics. These workflows have a number of challenges: expensive data movement, extensive monitoring, and fault-tolerance. In addition, the desktop development environment must scale to running on large-scale HPC systems.

Light-weight Execution. Pipelines built using Tigres need to be deployed and executed on shared HPC resources. Therefore, Tigres needs to avoid depending on persistent services, akin to the approach of MPI applications on current HPC resources.

B. Templates

We interviewed representatives from groups to better understand their general programming proficiency, their work culture, and types of workload. Through interviews with scientists we stepped through typical computational pipelines and discussed their development and production environments. As a result of these interviews, we identified that Tigres' API needs to support four templates (*sequence*, *parallel*, *split* and *merge*, shown in Figure 2). Task inputs and outputs can be simple types, complex objects, or references to external data sources such as files or databases. The results from the execution of a template can be used in a subsequent stage of the workflow or directly by the application. New tasks and templates can be defined or executed at virtually any part of the application code.

These four templates, according to our results, cover the basic needs of many scientific computational pipelines. Other workflow patterns can be built using a combination of these patterns [34]. In addition, we learned the user might need programming language constructs (e.g., loops) in their workflows. Since, our approach is a library in an existing language, users are able to directly leverage these constructs in the existing language.

In addition to the templates, the interviews identified three key areas that were important to our users. First, that there was a need to support the Tigres API in both Python and C. Our initial plan was to support the API only in Python but this would have limited its applicability in HPC environments. Second, the ability to seamlessly scale from the desktop to supercomputers was important to our users so the API needed to be usable in those environments.

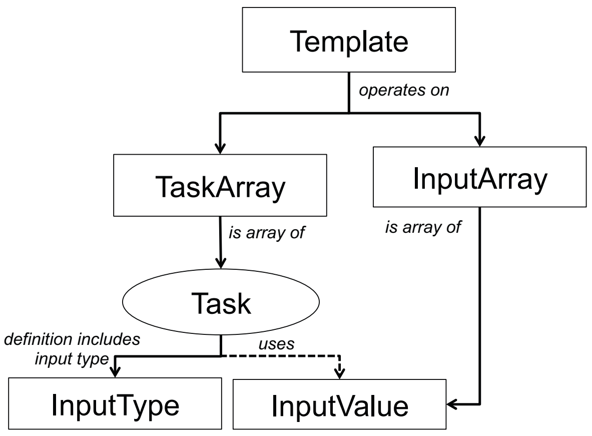


Fig. 3: Concepts used in the Tigres API and their relationships.

C. Template Concepts

Figure 3 shows the concepts upon which the Tigres API is built. A *Template* takes a *TaskArray*, that is a collection of tasks to be executed together, and an *InputArray* with the corresponding inputs to the tasks. The elements in a *TaskArray* form a collection of tasks that need to execute either in a sequence or in parallel.

Each *Task* definition includes the *InputTypes* that defines the type of inputs that the task takes. The *InputArray* is a collection of *InputValues*. *InputValues* are the values that are inputs to the task and its types match what is defined in the *InputTypes*. *InputValues* are passed to the task during execution and not included in the task definition. This allows for reuse of task definitions and late binding of actual data elements to the workflow execution, as desired by our users.

D. Tigres API

Table I summarizes our key results with respect to the API. Our API has three main categories a) Input definitions b) Task definitions and, c) Templates. The results from the usability study roughly fell under two categories i) changes that were made during the initial usability study phase ii) features or changes that affected the first functional prototype (implementation phase).

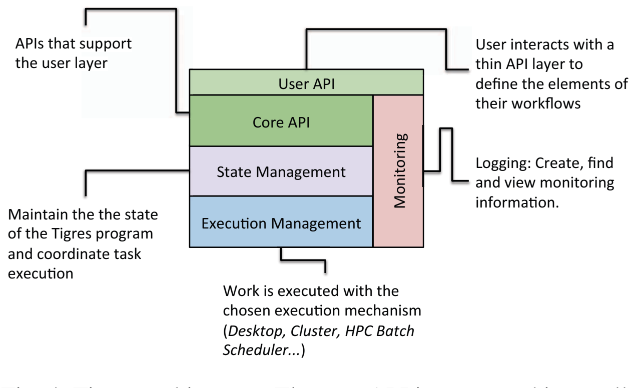
An example of a change resulting from the process is that our initial API started with different names for $InputTypes$, $InputValues$ and $..Array$. These names were confusing for

TABLE I: Impact of the process on the Tigres API with severity ratings [25]. The issues that were fixed during the user-centered design phase are marked as Fixed. The issues are rated as 0 -Don't think it is a usability problem, 1 - Cosmetic usability problem, 2 - Minor usability problem, 3 - Major usability problem,, needs to be fixed, 4-Catastrophic usability problem, needs to be fixed. Other issues were fixed in our first implementation.

Tigres API after usability testing	**Individual changes**	**Group-level changes**
InputTypes (name, types[ ])	Initially was called parameter_list **(3-Fixed)**	Make name optional **(1)**, Support language arrays **(2)**, Unsure how implicit data parallelism will work **(0)**, Unsure if user needs to specify O/P/s **(0)**
InputValues (name, values[ ])	Initially was called data_list **(3-Fixed)**	
InputArray (name, input_values[ ])	Initially started with set and renamed to arrays **(3-Fixed)**	
Task (name, type, impl_name, input_types, env)	Confusion over impl_name **(1)**	Make name optional **(1)**, Use of language-supported arrays rather than a new type**(2)**
TaskArray (name, task[ ])		
Sequence (name, task_array, input_array)	Allow users to not specify dependency when it is a simple sequence **(2)**	Dual syntax for dependency **(3)**
Parallel (name, task_array, input_array)	Was initially called DataParallel and it was not clear if it would handle dissimilar tasks **(1-Fixed)**	
Split (split_task, split_input_values, task_array, task_array_in)	The difference between task and task array was striking here **(1)**	
Merge (task_array, input_array, merge_task, merge_input_values)	Started with calling it Synchronization **(2-Fixed)**	

our study participants. Although, the study participants were able to determine what they referred to with minimal or no help, the early users in our study all expressed confusion over the names. The design team proposed a name change during the usability study phase and the new names were more easily understandable to the study participants and resulted in a better outcome. Additionally, the names of the templates were adjusted to address some of the questions raised by the study participants. For example, $DataParallel$ was renamed to $Parallel$ to reflect that it was able to handle dissimilar tasks.

The pseudo-code language neutral approach in the usability study was a little difficult for our users, who are all programmers. The users requested that the final API should support language-specific optimizations to coding (e.g., array types in Python, optional parameters). Tigres attempts to support an API that can be easily ported to other languages while still allowing language specific optimizations for programming ease.



APIs that support the user layer
User API
Core API
State Management
Execution Management
Monitoring
User interacts with a thin API layer to define the elements of their workflows
Logging: Create, find and view monitoring information.
Maintain the the state of the Tigres program and coordinate task execution
Work is executed with the chosen execution mechanism (Desktop, Cluster, HPC Batch Scheduler...)

Fig. 4: Tigres architecture. The user API is supported internally by the Core API. The core API interacts with the state management layer which in turn interacts with the execution management layer. All components interact with the monitoring component.

E. Dependency Syntax

There were two key issues that were identified as critical to be resolved during the implementation phase. First, the initial prototype had two different ways to specify dependencies - between the tasks in the workflow and and between the templates. The users used the task dependency approach while chaining templates as well, which highlighted that we need to have a single approach to specifying and managing dependencies. Second, users were very interested in simplifying the workflow specification for simple cases. For example, in simple sequence templates where each task depends on the previous task, there was a request to allow users to not explicitly specify the dependency.

We paid special attention to the dependency syntax in our first implementation which was entirely possible only due to the usability tests. The PREVIOUS syntax was standardized for all types of dependencies. Additionally, we paid special attention to implicit handling of data dependencies. We provide a short hand for splitting iterable output among parallel tasks (i.e., PREVIOUS.i indicates that in a set of parallel tasks the i-th task should get the i-th output from PREVIOUS task). We gave the users extended functionality that allows them to define specific dependencies such as getting the n-th output for the task name X (i.e., PREVIOUS.X.i[n].

F. Architecture and Implementation

Tigres has five major components in a layered architecture: User API, Core API, State Management, Execution Management and Monitoring (Figure 4). The layered architecture was largely influenced by the usability studies. First, without usability studies, the development team would have initially worked on the execution management and then built a user API that focused on the execution semantics. Also, in the execution semantics we would have first focused on the HPC system execution and not focused on the desktop use case. Second, it was possible to build the desktop version of Tigres within a couple of weeks since the user API was very clear.

The user interacts with the thin User API to define the

TABLE II: Monitoring API. The monitoring API was developed based on feedback from the usability studies

init (tigres-destination, user-destination)
set-log-level(level) enumeration FATAL upto TRACE
write(level, name, key-value pairs)
check(type, names)
find(name, key-value-pairs)

elements of their workflows, log messages and monitor their program execution. The core API is a collections of interfaces that support the User API with graphing, creating tasks, managing task dependencies and running templates. Also, since monitoring was mentioned as a key element by our users, we paid special attention to the monitoring API in the system. Monitoring is used by most other components to log system or user-defined events and to monitor the program with a set of entities for querying those events.

State management encompasses different management aspects of the workflow. For instance, it validates the user input to confirm they result in a valid workflow both prior to start as well as during execution. It transforms the Tigres tasks into *Work* units that are then handed off to execution management. The state management layer also provides monitoring and provenance capabilities. It maintains state as each template is invoked and integrity of the running Tigres program.

The execution layer can work with one or more resource management techniques including desktop, cluster and HPC systems. In addition, the separation of the API and the execution layer allows us to leverage different existing workflow execution tools while providing a native programming interface to the user. In the execution layer, a user's configuration is used to determine the execution mechanism. Tigres currently multiple exection mechanisms including local thread and process, cluster and HPC batch queues. Tigres does not start any specific services to manage the execution. The user program launches the specific tasks and once the tasks are completed, the next step in the program is executed.

Our implementation has been heavily influenced by the usability studies. Our implementation faithfully implemented the user API that was tested, only making changes where the usability studies indicated improvements were needed. We wrote tests for user scenarios from the usability studies that has helped test our codebase and ensure consistency.

G. Monitoring API

The usability studies indicated that real-time and user-defined monitoring schemes were very important to the users. The users desired the monitoring capabilities even for simple workflows. Thus, this would increase the use of templates, and help offset the initial programming effort required to use the templates. Table II shows the resulting monitoring API in our implementation.

V. DISCUSSION

We argue that user-centered design process should be an integral part of all eScience research and development tools. The process is very important to enable adoption of eScience tools developed. In this section, we summarize our experiences and provide some guidelines that will allow other projects to apply the techniques.

Design Process. The process of usability testing for the API is challenging. The process went through several refinement iterations. The usability study needs to be comfortable for the user. In our first pilot study, we asked the user to code the problem on a white board. We quickly found that coding on the whiteboard was unnatural and tedious. Moving to a Google doc on the participants own laptop greatly improved their comfort level. Also, for some of the tasks, some initial code was given, to reduce some of the tedium of having to repeat writing the same lines of code. Breaking down the problem into subtasks improved the general understanding of how to approach the API, although it also introduced some constraints in the programming solution that wouldn't normally exist. In terms of incorporating changes, we differentiated between typical user preferences versus general usability issues by using feedback from multiple users into account before making changes.

Impact on development cycle. The scientist-centered design process adds an upfront cost in the development cycle, since the initial usability studies delay the start of the development cycle. However, well-conducted usability studies reduce the development time as well as result in a better product, analogous to how good software engineering practices can impact the outcome of software practices [9]. We observed the following from Tigres: a) the first stub implementation took about two weeks to create rather than an estimated few months b) design and development priorities and decisions were more easily resolved based on results from the studies.

Team. Our design and development team usually consists of at least one usability and user interface design expert and one software engineer who together conduct interviews, run usability studies, and brainstorm about design ideas, with the designer taking the lead on all three areas. We believe that involving the software engineer in the SCD process allows the engineer to begin work on the infrastructure, keeping in mind implications that the design and needs represent, and helps the engineer prioritize which features to implement, based on the strongest user needs.

Adoption. We believe that the active and constant participation of scientists in the process helps ensure adoption in the long run. Our user group is eager to know about the development of Tigres and test future releases. We have found that scientists are not often able to fully describe their needs and goals, and this is because their needs, goals, and even their methods are constantly shifting. Constant communication is the design process is required, and early prototypes of design provide context needed for scientists to verbalize or even formulate their needs. In addition, much of the software needed for scientific work takes the form of expert systems, and it is not often possible to transfer all the domain knowledge needed.

Managing feedback. The user-centered design process is not an exact science. It is critical that the team doesn't offer too much direct unsolicited guidance during the usability study. Additionally, it is important to interpret the results carefully. To help manage this, we propose using schemes like the severity rating to classify and prioritize usability study issues. An important consideration when working directly and closely with scientific users relates to managing expectations. Scientists may expect that all of their suggestions appear in the design, but the designers need to weigh the preferences of any given scientist against the data collected about the user base as a whole. Feedback from scientists should inform the design but not necessarily drive it. Even when scientists contribute design ideas in the form of participatory design, there should be an understanding that the designer is the arbiter regarding what elements appear in the prototype.

Guidelines for Other Projects. We provide high-level steps that other eScience projects can use to engage with scientists during the design phase for eScience projects with similar APIs.

1) It is critical to identify early in the project the scientists who will serve as representatives of the intended audience of the tool.
2) The team must understand the work practices, work goals, what the scientists would like to achieve, and current similar tools. This can be accomplished by interviews and participant observation (i.e., watching them work).
3) Develop high-level usage scenarios that accomplish work goals.
4) Develop a low-cost prototype that is believed to address the work goals identified in step (1). In the case of a graphical user interface, this may take the form of a paper prototype or a quick clickable prototype. API prototypes can take the form of a document describing the function definitions. Documentation and example code aid in early usability tests.
5) Conduct a usability study with a select group of users using the following guidelines. Determine the testing medium that will be comfortable for the users to use. For example, Google docs was commonly used by all our test participants.
   a. *Preparation time:* Allocate a few minutes for the test participant to absorb the material and scenario when testing an API.
   b. *Scenario:* Ask users to work through the scenario, ”thinking out loud”.
   c. *Feedback:* Provide feedback on demand but do not intervene unless a participant cannot continue with the exercise. The feedback provided should be minimal and guide the scientist towards the solution rather than provide the solution. Use creative methods (e.g., a human compiler) to recreate an environment close to what the users will experience when using your tool.
   d. *Follow-up.* The study should be followed up with an open discussion. The discussion should cover aspects of the tests the users felt comfortable with. Additionally, ask followup questions to see whether the material was understandable and if the participants can see themselves using it for their work
   e. *Results.* The usability test results should be evaluated and used to determine if a redesign is required and/or prioritize project priorities appropriately.
6) Repeat the prototyping and testing cycle, increasing in the fidelity/functionality of the prototype at each iteration.

VI. Conclusion and Future Work

This paper presents Tigres and the design process that has influenced the initial design of Tigres. The key benefits of the approach to-date include: a significant reduction in misunderstood requirements, an improved understanding of the software capabilities needed, and less time spent developing software that will never be executed (only develop capabilities that can be used through the interface).

Tigres has greatly benefited from the user-centered design process. The process has enabled us to improve nomenclature of the concepts and API structure early in the design and shortened the initial development cycle.

In the next usability study of Tigres, we plan to include the prototype implementation of Tigres. The goals of the next study would be to evaluate the use of the API, graphing, monitoring and execution semantics.

Acknowledgment

This work was funded by the Office of Science, Office of Advanced Scientific Computing Research (ASCR) of the U.S. Department of Energy under Contract Number DE-AC02-05CH11231.

References

[1] Oozie: Workflow engine for Hadoop. http://yahoo.github.com/oozie/.

[2] Tigres Website. http://tigres.lbl.gov/, 2013.

[3] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, 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*, pages 423–424, 2004.

[4] C. Aragon and S. Poon. The impact of usability on supernova discovery. In *Workshop on Increasing the Impact of Usability Work in Software Development, CHI 2007: ACM Conference on Human Factors in Computing Systems*, 2007.

[5] C. R. Aragon and S. S. Poon. Designing scientific workflow management systems for data-intensive astrophysics projects. In *Designing Cyberinfrastructure to Support Science Workshop, CSCW 2008: ACM Conference on Computer Supported Cooperative Work*, 1998.

[6] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical report, EECS Department, University of California, Berkeley, Dec 2006.

[7] R. Barga, J. Jackson, N. Araujo, D. Guo, N. Gautam, and Y. Simmhan. The trident scientific workflow workbench. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 317 –318, dec. 2008.

[8] J. Bloch. How to design a good api and why it matters. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 506–507. ACM, 2006.

[9] F. P. Brooks, Jr. *The mythical man-month (anniversary ed.).* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[10] P. Bui, L. Yu, and D. Thain. Weaver: integrating distributed computing abstractions into scientific workflows using python. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 636–643, 2010. http://doi.acm.org/10.1145/1851476.1851570.

[11] N. Cerezo, J. Montagnat, and M. Blay-Fornarino. Computer-assisted scientific workflow design. *Journal of Grid Computing*, 11(3):585–612, 2013.

[12] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang. Programming Scientific and Distributed Workflow with Triana Services. *Concurrency and Computation: Practice and Experience (Special Issue: Workflow in Grid Systems)*, 18(10):1021–1037, 2006.

[13] S. Clarke. Measuring API usability. *Dr. Dobb's Journal*, 29:S6–S9, 2004.

[14] Condor DAGMan. http://www.cs.wisc.edu/condor/dagman/.

[15] L. Dagum and R. Menon. OpenMP: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[16] J. M. Daughtry, U. Farooq, J. Stylos, and B. A. Myers. Api usability: Chi'2009 special interest group meeting. In *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, pages 2771–2774. ACM, 2009.

[17] E. Dede, M. Govindaraju, D. Gunter, and L. Ramakrishnan. Riding the elephant: managing ensembles with hadoop. In *Proceedings of the 2011 ACM international workshop on Many task computing on grids and supercomputers*, pages 49–58, 2011.

[18] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[19] U. Farooq and D. Zirkler. Api peer reviews: A method for evaluating usability of application programming interfaces. 2010.

[20] M. Gemmell. API design. http://mattgemmell.com/2012/05/24/api-design/, 2012.

[21] S. Ghemawat and J. Dean. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI04), San Francisco, CA, USA*, 2004.

[22] J. Goecks, A. Nekrutenko, J. Taylor, and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010.

[23] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, Aug. 1978.

[24] N. Lorang. API design for humans. http://37signals.com/svn/posts/3018-api-design-for-humans, 2011.

[25] J. Nielsen and J. T. Hackos. *Usability engineering*, volume 125184069. Academic press Boston, 1993.

[26] T. Oinn et al. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, 2006.

[27] S. Poon and et. al. Scientist-centered design for escience. Technical report, Lawrence Berkeley National Lab.

[28] L. Ramakrishnan, P. T. Zbiegel, S. Campbell, R. Bradshaw, R. S. Canon, S. Coghlan, I. Sakrejda, N. Dešai, T. Declerck, and A. Liu. Magellan: Experiences from a science cloud. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, ScienceCloud '11, pages 49–58, 2011. http://doi.acm.org/10.1145/1996109.1996119.

[29] M. Reddy. *API Design for C++*. Morgan Kaufmann, march 2011.

[30] D. Saffer. *Designing for interaction: creating smart applications and clever devices*. New Riders Pub, 2010.

[31] J. Stylos and S. Clarke. Usability implications of requiring parameters in objects' constructors. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 529–539, Washington, DC, USA, 2007. IEEE Computer Society.

[32] J. Stylos, B. Graf, D. Busse, C. Ziegler, R. Ehret, and J. Karstens. A case study of api redesign for improved usability. In *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*, pages 189–192, 2008.

[33] J. Tulach. *Practical API Design: Confessions of a Java Framework Architect*. Apress, 2008.

[34] W. M. P. van der Aalst, Ter, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, July 2003.

[35] M. Wei, C. Jiang, and M. Snir. Programming patterns for architecture-level software optimizations on f requent pattern mining. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 336 –345, april 2007.

[36] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652, 2011.

[37] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud'10, pages 10–10, 2010. http://dl.acm.org/citation.cfm?id=1863103.1863113.

[38] C. Zhang, H. De Sterck, M. Jaatun, G. Zhao, and C. Rong. CloudWF: A Computational Workflow System for Clouds Based on Hadoop. In M. G. Jaatun, G. Zhao, and C. Rong, editors, *Cloud Computing*, Lecture Notes in Computer Science, pages 393–404. Springer Berlin Heidelberg, Berlin, Heidelberg. http://www.springerlink.com/content/n426v37875r87150/.