International Journal of
Information Systems and Project Management
ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X
Available online at www.sciencesphere.org/ijispm

MIDST: an enhanced development environment that improves the maintainability of a data science analysis

Jeffrey S. Saltz
Syracuse University
343 Hinds Hall, Syracuse, NY 13210
USA
jsaltz@syr.edu

Robert Heckman
Syracuse University
343 Hinds Hall, Syracuse, NY 13210
USA
rheckman@syr.edu

Kevin Crowston
Syracuse University
343 Hinds Hall, Syracuse, NY 13210
USA
crowston@syr.edu

Yatish Hegde
Tulane University
7 McAlister Drive New Orleans, LA 70118
USA
yhegde@tulane.edu

Abstract:
With the increasing ability to generate actionable insight from data, the field of data science has seen significant growth. As more teams develop data science solutions, the analytical code they develop will need to be enhanced in the future, by an existing or a new team member. Thus, the importance of being able to easily maintain and enhance the code required for an analysis will increase. However, to date, there has been minimal research on the maintainability of an analysis done by a data science team. To help address this gap, data science maintainability was explored by (1) creating a data science maintainability model, (2) creating a new tool, called MIDST (Modular Interactive Data Science Tool), that aims to improve data science maintainability, and then (3) conducting a mixed method experiment to evaluate MIDST. The new tool aims to improve the ability of a team member to update and rerun an existing data science analysis by providing a visual data flow view of the analysis within an integrated code and computational environment. Via an analysis of the quantitative and qualitative survey results, the experiment found that MIDST does help improve the maintainability of an analysis. Thus, this research demonstrates the importance of enhanced tools to help improve the maintainability of data science projects.

Keywords:
project management; data science; maintainability; visual programming; data science development environment.

DOI: 10.12821/ijispm080301

Manuscript received: 20 May 2019
Manuscript accepted: 5 April 2020

1. Introduction

Data Science is an emerging discipline that combines expertise across a range of domains, including software development, data management and statistics. Data science projects typically have a goal to identify correlations and causal relationships, classify and predict events, identify patterns and anomalies, and infer probabilities, interest and sentiment [1]. As a new field, much has been written about the development of data science algorithms. Unfortunately, less has been written about other challenges that might be encountered when working as a data scientist [2].

One such challenge that has not yet been extensively explored, but that will grow in importance, is the ability to create an analysis that is easy to maintain and enhance. Maintainability, which is the ease with which a solution can be modified to correct faults, improve performance, enhance capabilities, or adapt to a changed environment [3], is of growing importance due to the fact that not every data science analysis is a "one-off" task, and, as more organizations start to leverage data science, there will be a growing need to update previously developed insights. For example, an analysis might need to be updated due to a change in a data source or to evaluate a new machine-learning algorithm. Moreover, this update might need to be done by a new team member, thereby increasing the importance of having code that is easy to understand by someone that was not part of the original team that developed the initial analysis.

In general, maintenance (or actions taken after the first implementation) is an important aspect of a project's lifecycle [4] and as well as an important aspect of project management [5]. While project documentation can represent a valuable source of knowledge, it has been noted that the knowledge codified in project documents is typically not re-used in future projects [6]. Hence, project documentation, while helpful, is not sufficient for a team to maintain an application.

With respect to data science, Jagadish et al. [7] describe a workflow where data flows from acquisition, to information extraction and cleaning, then to data integration, and then modeling and analysis. Furthermore, due to the exploratory nature of data science, this workflow typically involves iterative cycles of obtaining, cleaning, profiling, analyzing, and interpreting data [8, 9]. However, RStudio, which is the most commonly used tool for creating an analysis when using the R programming language [10, 11], uses a functional code-based framework, which is similar to many software development interactive development environments (IDEs), in that there are multiple views within the environment, including views of actual source code, the output of recently run commands, and of the variables defined in the current scope of execution.

Due to this non-linear flow within data science, Rule et al. [12] observed that an analysis, written using a linear construct, which is how one creates R code within RStudio, often becomes difficult to navigate and understand. This difficulty in navigation and understandability discourages sharing and reuse. One approach that has been explored to address this challenge is the use of a visual data flow metaphor [13], and in fact, a data flow construct has been suggested as the conceptual mental model used by a data scientist [14]. This mental model is very different from the text-based coding environment used within tools such as RStudio. Hence, using tools such as RStudio might create a representation mismatch between the code being written and the abstracted data-driven mental model used by the data scientist [15]. This conceptual mismatch could make it difficult for a data scientist to understand and update code that was previously developed, especially if that code was developed by a different person.

Currently, there is no commonly used general purpose R development environment that is based on the concept of a visual data flow metaphor that defines and links R modules. Thus, one potential step forward to improve the maintainability of an analysis could be via a tool that creates such a visual data flow paradigm, integrated within a code-based programming environment. With this background in mind, our research explored if the maintainability of an analysis could be improved when data scientists use an enhanced development environment. Specifically, we focused on the following research question:

> *Does using visual data flow tool, within an integrated data science development environment, improve the ability of a new team member to reuse and update an existing analysis?*

The next section provides some additional background context. In Section 3, we describe the new tool that was developed to explore our research question, and the methodology used for an experiment to assess the impact of the tool on the maintainability of an analysis. Section 4 presents findings from the study. Section 5 presents a synthesis of our research results, and Section 6 summarizes the research, discusses limitations, and describes possible next steps.

2. Background

2.1 Data Science Project Maintainability

Pimentel et al. [16] noted that data science analyses often use poor coding practices, which means that their results can be hard to reproduce and maintain. For example, prior research has revealed that many analyses were difficult to understand, and that even the original analyst can struggle to understand their prior effort [12, 17]. Furthermore, while describing the challenge of creating maintainable R applications, Malviya et al. [18] state that a well-defined workflow (i.e., where to put/get data and how to produce, collect and report the results) can help improve maintainability. They hypothesize that this workflow should be created within a tool, but only describe the tool at a high level. In support of this view, Simmons et al. [19] analyzed code from 1048 data science projects and observed that data science projects suffer from a high rate of functions that use an excessive number of parameters and local variables, and thus, would be difficult to maintain. It was also noted that data science projects do not follow traditional software engineering conventions because traditional software engineering conventions are inappropriate in the context of data science projects.

To help try and address this maintainability challenge, one recent study explored the ability to fold (i.e., hide) logical chunks of code, which had mixed results due to, for example, new data scientists overlooking folded sections [12]. Others have explored how to best capture previous exploration within an analysis by improving tools to understand the history of the analysis [20].

Beyond this, research on data science maintainability is rare [21]. Others have, however, noted the importance in maintaining an analysis. For example, Dhar & Mazumdar [22] identify maintainability as a key challenge in using big data science within an enterprise context and note that the availability of new tools and IDEs could minimize this challenge and make maintenance less of an issue. In addition, Sachdeva & Chung [23] also observe that maintainability is a key challenge that is typically ignored, but which is becoming increasingly important and needs to be handled in a well-defined manner.

Finally, one key aspect of maintainability is reproducibility, which Tatman et al. [24] define as “recreating the exact results” (pg. 2). In other words, it is the extent to which consistent results are obtained when an analysis is repeated. This is an important first step with respect to maintainability because if one cannot reproduce an analysis, then one cannot refine/enhance that analysis. Tatman at al. [24] describe three levels of reproducibility. Low reproducibility studies are those which merely describe algorithms that were used within an analysis, medium reproducibility studies are those which provide the code and data but not the computational environment in which the code can be run, and high reproducibility studies are those which provide the code, data, and full computational environment necessary to reproduce the results of the study.

With respect to reproducibility, it has been noted that high reproducibility is a significant challenge when developing R-based analyses [25]. In fact, Beaulieu-Jones & Greene [26] observed that even an analysis that is “scriptable and should be easy to reproduce (...) remains difficult and time consuming to reproduce computational results because analyses are designed and run in a specific computing environment, which may be difficult or impossible to match from written instructions” (pg. 342).

2.2 Data Science Development Environments

Beyond RStudio, there are other development environments that focus on high reproducibility, such as container-based environments. Specifically, high reproducibility can be achieved via the use of a container tool such as docker, which allows an application to be packaged with all its parts, such as libraries and other dependencies, and act like a light

weight virtual machine for creating a complete computing environment [25, 26]. This includes tools such as Kaggle Kernels [27] and Jupyter notebooks [28]. However, none of these environments address the non-linear nature of data science analysis [12].

As previously noted, a visual data flow metaphor might be an interesting alternative to existing data science IDEs. At its core, a visual data flow program represents code using graphical box-and-wire diagrams, where boxes denote modules (or functions) and wires denote the passing of values between functions [27]. Leveraging this visual data flow paradigm has been used for decades to analyze data [29, 30]. For example, the Application Visualization System (AVS) was created nearly thirty years ago [31] and was designed around the concept of software building blocks, or modules, which could be interconnected to form a visual data flow program. In fact, it has been noted that visual data flow programming is most successful where data manipulation is the foremost important task [32].

This visual metaphor is now receiving renewed attention across a range of applications [33, 34]. However, within a data science context, these visual data science data flow tools do not focus on the development of R code. Rather, these are higher-level tools that promote the idea of "machine learning for everyone" by enabling people who do not code to be able to create a data analysis. For example, Weka [35], a popular machine learning development environment, provides both a command line interface and a graphical interface that is focused on higher-level graphical programming, as opposed to providing a graphical user interface to help data scientists develop their R code. Another visual data science tool, KNIME, does make it possible for programmers to create nodes using a node wrapper that understands/parses R code, but like Weka, developing new R code is not core to the thought process of the users of KNIME [36]. Other examples of these higher-level visual programming environments include ViSta for statistical analysis [37], and Orange for machine learning [38]. In other words, while there are visual data flow programming tools for data science, these tools are focused on higher level constructs to create an analysis by reusing existing nodes or whole workflows (i.e., not focused on programming). Due to this difference in focus, it has been observed that when trying to develop R code within tools such as KNIME, users had issues due to the lack of integration between visual data flow editor and the textual environment, which led to user frustration [39].

In short, there is currently no commonly used general purpose development environment for developing R code that is based on the concept of data flow, where the user creates code modules and links those custom nodes [39]. However, a small 4-person case study of a data flow prototype system found that integrating a code-based environment with a visual data flow view was useful in creating a more understandable analysis [40], which suggests that a visual data flow view might improve maintainability.

2.3 Software Development Project Maintainability Models

Even in software development, the concept of maintainability is often overlooked. For example, in a systematic literature review that focused on compiling and synthesizing project success factors in Information Technology (IT) projects, the concept of maintainability was not explicitly noted [41]. Furthermore, the need for maintainability was only implicitly noted in a systematic literature review that identified factors of IT project complexity, where reusability was noted as a key complexity factor [42].

However, due to the importance of software maintainability, during the past 20+ years, a wide range of maintainability models have been proposed. In one model, Muthanna et al. [43] focused on design level metrics that characterize the overall data and control flow between modules. In a different example, [44] describes a higher-level quality model, which includes self-descriptiveness, modifiability and testability as the sub characteristics for maintainability. In yet a different example, Rizvi et al. [45] describe a model for maintainability that has two key factors: understandability and modifiability. More generally, Dubey & Rana [46] analyzed 17 maintainability models and identified 37 different maintainability attributes. The most frequently mentioned attribute was testability (noted in 47% of the models), and other frequently mentioned attributes included self-descriptiveness, modifiability and understandability. In fact, understandability has been identified as playing a pivotal role in software maintenance [47].

With respect to evaluating the maintainability of a software system, Roehm et al. [48] explored maintainability via an observational study of 28 developers to identify the steps developers perform when understanding software and the

artifacts they investigate. A different approach was used by Shima et al. [49], where they assessed the ability of students, acting as a proxy for developers, to correctly reconstruct a system from its components, which they termed "software overhaul".

2.4 Deriving a Data Science Maintainability Model

Our data science maintainability model is shown in Figure 1. Since many, such as Schneberger [50], noted the importance of a development environment, this is the initial factor within the model. Furthermore, while testability was a key concept within a software development context, we leveraged Tatman et al.'s [24] concept of reproducibility as a related, but more relevant factor for data science analysis. So, in parallel with reproducibility, we also introduced understandability as a key factor to enable maintainability. Thus, within our model, the maintainability of an analysis is driven by being able to understand and recreate the analysis, both of which could be facilitated via the use of an enhanced development environment. Note that we broadly define a development environment to include the tools used to create, update, run and share an analysis.

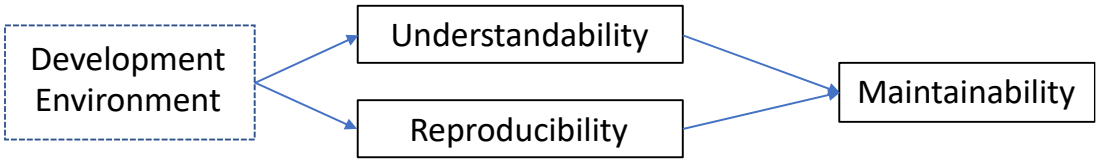

Fig. 1. Data Science Maintainability Model

3. Methodology

3.1 Research Method

To explore how to improve data science analysis maintainability, we used our data science maintainability model, which was previously shown in Figure 1. First, we developed a new visual data flow data science development environment, called MIDST (Modular Interactive Data Science Tool), that we hypothesized would improve understandability and reproducibility. Then, to compare the maintainability of an analysis done using RStudio to an analysis done using the new visual data flow tool, we measured the reproducibility and understandability of an analysis done using RStudio with an analysis done using MIDST.

In short, to evaluate reproducibility and understandability, we leveraged the previously discussed methodological concept of a software overhaul [49], but adapted the software overhaul concept to a data science context by having the data scientist share an analysis with a person not involved with its development, and then having that other person update and re-run that existing analysis. Specifically, we conducted an experiment where, in the baseline condition, the analysis and updates were done using RStudio, and in the other, experimental condition, teams used MIDST's new visual development environment. All teams, across both conditions, analyzed the same data, and the new person was given the same task in terms of updating the analysis.

The rest of this section describes our research method in more depth.

3.2 New Visual Data Flow Environment - MIDST

MIDST is a web-based data science IDE that was developed for this project. Due to MIDST's web-based architecture, all the R code runs on a common compute server. This shared execution environment greatly facilitates team collaboration since issues such as what libraries and what versions are installed as well as other details such as location of data files are eliminated. As well, it makes it easy to share code between team members.

When using MIDST, all members of a team can share their code via an easy to use graphical code management system that lets each team member easily "push" their updates and "pull" updates from the team's shared repository, which integrates all modules used in the team's analysis. In fact, MIDST has reminders when a team member needs to pull the updates from their shared repository. In addition, MIDST provides a common computing infrastructure, where each user has a clone of the same computing environment. MIDST provides several different views to enable data scientists to leverage visual data flow programming to construct an analysis.

MIDST Network View: MIDST's network view provides the high-level data flow view of the modules within the application. This network view, shown in Figure 2, enables users to create nodes, define the input and output for each node, and then connect those nodes together, via the concept of data flowing between the nodes. In addition to code modules (the R code is within each of the code modules of the network), there are data nodes and output visualization nodes. Similar to other data flow systems, each code module node within MIDST has inputs and outputs, and MIDST users connect the nodes, where the output of one node is the input to another node. For example, Figure 2 shows a simple application that reads in a raw data file, cleans the data file, stores that data file and then generate a histogram.

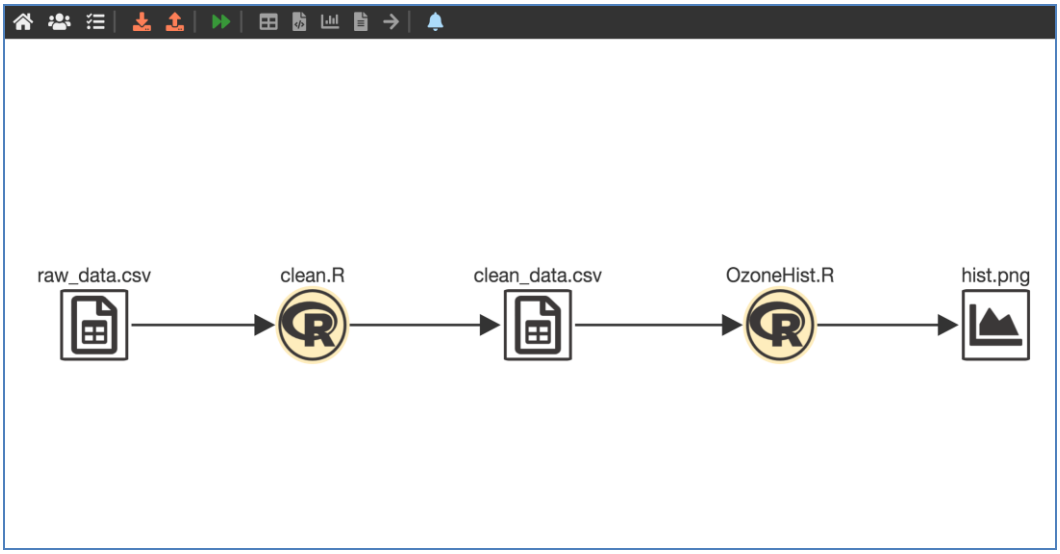

Fig. 2. MIDST Network View

As with other data flow tools, the network editor helps users break tasks into chunks, and to visualize the flow of the project. Of course, since the user has control over how to define the network and the inputs and outputs of each node, the simple analysis shown in Figure 2 could be implemented via a different data flow diagram, as shown in Figure 3.

Within this network view, MIDST users can easily execute the entire network (within their cloned virtual environment) by pressing the 'run' button at the top of their network view window, and any errors that occur during execution of the network are clearly visible as failed nodes, as shown via the exclamation mark in Figure 4. The current version of MIDST does not provide a library of previously created nodes/modules that were used in previous projects or created as utility modules by others. A more production-oriented version of MIDST could easily provide such a library of modules.

MIDST Source Code Editor: MIDST also allows users to drill down to the actual R code within a module, by clicking on a node to view/edit the actual R code within the MIDST source code editor, which is similar to the editor in RStudio. The code view is more of a typical source code editor, with the ability to view, edit and run the actual R code for each

of the modules within the application. As shown in Figure 5, the editor enables the user to write and debug the R code for a module. For example, the R code for the clean module from Figure 3 is shown in Figure 5. In this view, similar to how one uses RStudio, the user writes R code to implement the required functionality for that module. Also, within the source code editor, one can either run the entire module or execute a line of code in the module. Output for the module is always shown in the bottom window pane of the source code editor. This output is for the most recent run of the module (whether that was due to the full network being run in the network view, the full module being run in the source code editor view, or a specific line being executed within the source code editor).

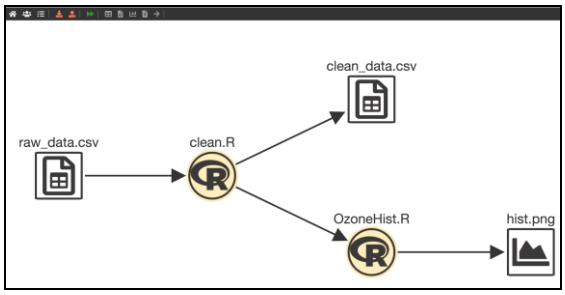

Fig. 3. Alternative MIDST Network

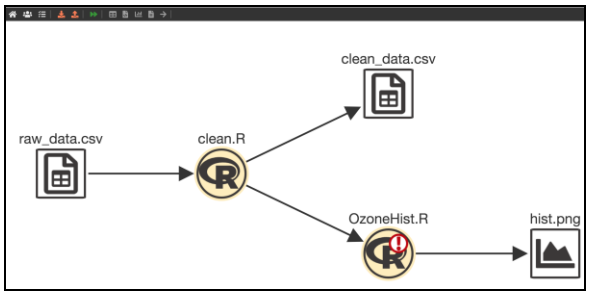

Fig 4. MIDST Network View – with an error

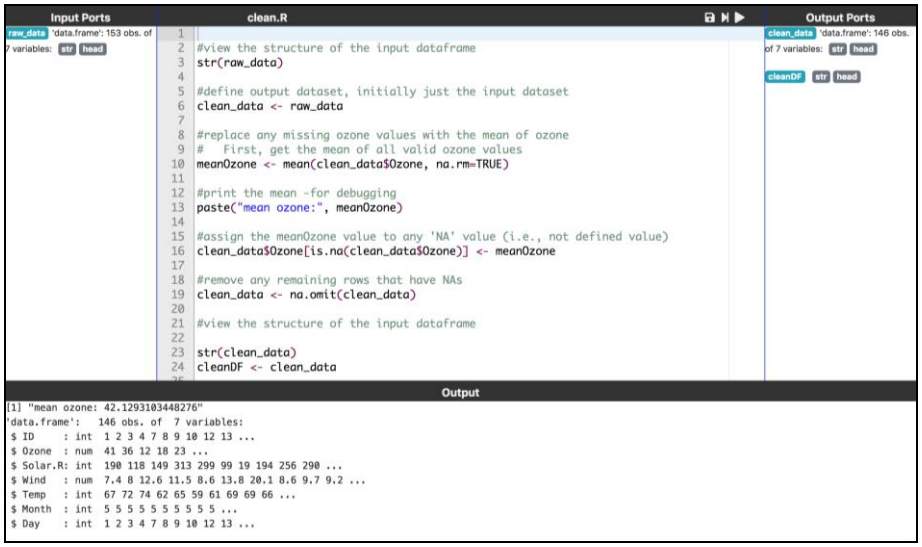

Fig. 5. MIDST Source Code Editor

3.3 Maintainability Experiment

To evaluate the impact of MIDST, a mixed method experiment was conducted where the experimental condition, of participants using MIDST, was compared to the baseline condition, of participants using RStudio.

Participants in both conditions were assigned the same initial task, which was an eight-week long group project, where teams of 4-6 people analyzed a customer survey dataset. The analysis included typical data science tasks such as using visualization, mapping techniques, and machine learning to predict unhappy customers. After the analysis was completed (and submitted for evaluation), there was an optional extra task. For this voluntary task, participants shared

their project with a person in a different team (but within the same experimental condition), and then that new person updated the analysis. The update consisted of the simple task of sampling 75% of the data, after cleaning, and then comparing the results the analysis using the sampled dataset with the evaluation that was conducted when using the entire dataset.

During the experiment, for both the baseline (RStudio) and treatment (MIDST) conditions, quantitative and qualitative information was collected. Table 1 shows the quantitative data collected, which was used to measure reproducibility and understandability. Specifically, we considered two sharing metrics to be indicators of reproducibility: the time it took to locate and assemble the original code, and the percentage of the original code that was shared. The percentage of the original analysis that was able to be run correctly is a third indicator of reproducibility. The time necessary to update the analysis is an indicator of understandability. The time metrics are similar to those used in a software development context, where the maintainability of a system was explored by measuring the total effort, in person-minutes, to comprehend, modify, and test the artifacts related to the system [51].

Table 1. Metrics collected and map from actions to testability and modifiability

Action	**Metrics**	**How Measured**	**Relevance:** Reproducibility	**Relevance:** Understandability
Share Analysis	Time to Share Analysis	Self-reported by person sharing the analysis	X	
	Percentage of Analysis Shared	Objective assessment	X	
Run Analysis	Percentage of Analysis run correctly	Objective assessment	X	
Update Analysis	Time to modify analysis	Self-reported by person doing the updates		X

With respect to the qualitative data, to better understand the thoughts and perceptions of doing this assignment, participants were asked, via an online survey, the following open-ended questions:

- "What were the key challenges to get the code running?"
- "What challenges did you encounter when you tried to update the other team's code?"
- "Please provide some general thoughts/impressions of doing this assignment."

3.4 Data Evaluation

For the quantitative data, the percentage of code shared as well as the percentage of the analysis that was able to be run correctly was determined via an analysis of the code performed by a teaching assistant for the class, who was not part of this research effort. The participants also reported on the time it took them to share their team's analysis, as well as the time it took them to modify the analysis the analysis that they were given to update.

The qualitative data was analyzed to identify the key themes with respect to sharing, updating and running the analysis. This thematic analysis was performed for the comments in the MIDST condition and another thematic analysis was performed for students in the baseline condition. Specifically, for each condition, the Miles and Huberman approach to analytic induction was used [52], with a goal of identifying common themes via an iterative process of item surfacing, refinement and regrouping. The coding started by reviewing each response and defining a high-level item description. If needed, one response was broken into multiple items. These item descriptions were then grouped and re-grouped into higher level conceptual themes. To increase reliability, the coding was done independently by two researchers [53], and ambiguities in coding were discussed and resolved amongst the researchers.

3.5 Experimental Conditions

The participants in this study were graduate students. In terms of the appropriateness of using graduate students within the study, while there has been little written about using students to gain insight into industry teams within a data science context, student experiments within the software development domain have been taking place for decades. In fact, students were used as subjects in 87% of the experiments analyzed over a representative ten-year period [54]. It is also important to note that when using students as subjects, several factors are typically considered. First, "students vs. professionals" is actually a misrepresentation of the confounding effect of proficiency, and in fact differences in performance are much more important than differences in status [55]. Hence, using master level students, many of whom have several years of industry experience can often be a more appropriate choice than undergraduate students with minimal experience. Second, comparing across experimental conditions, using students may actually reduce variability because all students have about the same level of education, leading to better statistical characteristics [56]. Finally, a third consideration is that while students might not be as experienced as practicing professionals, they can be viewed as the next generation of professionals and hence many believe they are suitable subjects for these types of studies [57, 58].

In total, ninety-six graduate students participated in our study. While most of the students in each section were graduate information system students, approximately fifteen percent of the students in each section were in other graduate programs, mainly business administration or public policy. Sixty-five percent of the students had previous work experience. In addition, forty percent of the participants were female. Thirty five percent of the participants were from North America, fifty-five percent from Asia, and ten percent from other locations (all students were co-located during the actual experiment).

The students initially registered to be in one of four sections, without any knowledge of which sections would use MIDST, or which sections would not use MIDST. Two of the sections (with a total of 44 students) were selected to be in the baseline (i.e., RStudio) condition, and two of the sections (with a total of 51 students) were in the treatment (i.e., MIDST) condition, where the students used MIDST to do the analysis.

4. Findings

In the Baseline (RStudio) sections, 31 students out of 44 chose to do the extra-credit assignment. In the Treatment (MIDST) sections, 13 students out of 52 chose to do the extra credit assignment using MIDST.

We note that there were 19 other students in the MIDST condition who chose to do the voluntary extra credit assignment using RStudio, which was permitted for the voluntary assignment. This was likely due to the fact that MIDST was an early prototype system, and as will be noted in the qualitative findings, there were some issues that students would occasionally encounter. To explore the possibility of selection bias, we observe that the 19 students in the MIDST condition that used RStudio had a final grade average of 86.1% (with a standard deviation of 4.7), and the 13 students that used MIDST for the maintainability task had an average of 87.0% (with a standard deviation of 3.4). Thus, since the focus of our analysis was on comparing the "No MIDST" condition with the "MIDST" condition, and since there was no selection bias (as noted via the final grade averages), we excluded the 19 students that used RStudio but were in the MIDST condition from the analysis. Also note that, during the sharing of the code, one MIDST user found a bug in MIDST and the time reported for this user was heavily influenced by this bug, so this user's data was not used during the analysis (hence we only evaluated the 12 students that used MIDST and did not encounter this bug).

4.1 Sharing the project

As can be seen in Table 2, students in the MIDST condition shared more of the project (100% vs 84%, on average), and did so much more quickly (30 minutes vs 80 minutes, on average). Using a t-test, which is a statistical test that can be used to determine if there is a significant difference between the means of two groups (in other words, a t-test lets one know if the difference in the mean could have happened by chance), the difference in both what was shared, and how long it took to share the code, was significant.

Table 2. Results for Sharing the Code

	How the code was shared		**Significant Results**
	R Files(s) *(N=31)*	**MIDST Network** *(N=12)*	*(at p=0.5 level)*
Time required to share the code *(average # of minutes)*	80	30	Yes
Percentage of project that was shared (*average*)	84%	100%	Yes

4.2 Updating & Running the project

As can be seen in Table 3, students in the MIDST condition got more of the project code analysis to run correctly (91% vs 78%, on average). It also took them significantly less time to update the code. MIDST users took an average of 3 hours to complete the task, versus close to 5 hours for students working within RStudio. Using a t-test, the differences in how much of the project was able to be successfully recreated, as well as how long it took to do the project update, were both significant. Note that 3 people in the "no MIDST" (RStudio) condition did not finish the project enhancement, and hence, were removed from this part of the analysis.

Table 3. Results for updating the code criteria

	Updates Done Using		**Significant Results**
	RStudio	**MIDST**	*(at p=0.5 level)*
Percentage of code that was working correctly *(percentage of total analysis)*	78%	91%	Yes
Time needed to make changes (*average # of minutes*)	299	180	Yes

4.3 Qualitative Feedback

Our thematic analysis of the qualitative data (i.e., the free form participant survey questions), provides some context as to the challenges RStudio students encountered when sharing and updating the project. Specifically, three key themes were identified for the baseline (RStudio) condition, all related to how difficult the task was to do (i.e., sharing, updating and running the code). In contrast, in the MIDST condition, three different themes emerged (sharing and running the code was easy, and updating the code was easier due to visual data flow network that made the code easier to understand). Below, these themes are explored in more depth.

4.3.1 Baseline Users Qualitative Feedback (RStudio)

As demonstrated by the example comments below, for each of the identified themes, participants felt that sharing the analysis was difficult because after they divided the work across the team members, the team members did not share their code with the other team members. In other words, each team member typically worked in isolation from each other. Furthermore, many in the baseline condition thought that updating the code was difficult, mainly due to the fact that it was a challenge to understand the flow of the analysis. Finally, running the code in the baseline condition was also difficult, mainly due to not all the code being integrated within a single environment.

Sharing the code – hard to do

> "For our project, each group member was responsible for using their own code in creating allocated visualizations, models, etc. So, when attempting to combine it as one code, some things didn't match up. Naming of vectors was different, subsets we defined and used then interfered with later code, there was just a few things that weren't included. For some reason I wasn't able to include the association rules code."
>
> "Not all of our code was shared with me so I shared as much as I could."
>
> "Text mining- I did not have the function that was created and used … Hence, instead of sharing an incomplete code, I chose not to include that [part of the analysis]."
>
> "I did not share a particular code of my group project … the person who did his part … was travelling [and] I did not want the person waiting."
>
> "Shared the R script file via Email. Sent it as an attachment."

Updating the code – hard to do

> "It is too hard to find which part should be modified."
>
> "The flow of the analysis was a little haywire, with no clear distinctions between linear modelling, descriptive statistics, a-rules and SVM. They were all correct, but it took me a while to understand the flow of the code."
>
> "What made the project relatively hard to follow and replicate is the fact that there was no cohesion in the work. Given that the folder that I received contained different files representing the various sections of the project, it was not obvious to figure out objects that may change from one section to the other. As such, there was no structure to follow in re-running and updating the project."
>
> "One of the key challenges was to understand the code. Although there were comments … the code as a whole was little haphazard."

Running the code – hard to do

> "Different files used different datasets so it was really difficult to change it."
>
> "With respect to the R code, I had to update the variables and adjust some codes w.r.t. the sample dataset."
>
> "I had to change … the variable names so that I could get the code running."
>
> "About 30% of the code [had to be] changed. I had to correct the syntax errors, changed the map visuals to make them work."
>
> "[it was difficult] going over the errors and understanding why the errors were happening and also looking at different functions. Every group had their own functions so understanding what different functions do [was challenging]."

> "Three of the substantial algorithms failed to run due to missing objects that were not defined and that I couldn't figure out their origin/source."

4.3.2 MIDST Users Qualitative Feedback

Unlike the baseline users, almost all people using MIDST noted that sharing was easy, as was updating the code. However, due to the fact that MIDST was a prototype system, some also noted the MIDST bugs and performance issues.

Sharing the code – was easy

> "I shared the entire code with the other person through MIDST."

> "I shared every single part of our project with the other member."

> "Easiest way was to provide the Midst link by adding them as temporary collaborator."

Updating the code – leveraged the visual data flow network to understand the code

> "It was important to understand the other team's project and the flow of data between all the nodes in order to update the code so as to get the code running without any errors."

> "I just needed to add a node with the sampling coded … and then supply this sampling dataset to the models."

> "I updated code in a single node…I was able to understand the code due to the flow in MIDST."

> "I added a node in the MIDST tool to sample the data. After cleaning, I added the flow to the new node sampling.R. After creating the sampled data set, the data was supplied to the models for analysis."

> "The code provided on MIDST was put in an easy to understand fashion. I went through various nodes to understand their function."

Running the code - easy except for MIDST bugs

> "Overall, I did not face any major challenge while running the code."

> "The entire project was running properly. The only issue was … a bug in MIDST, which resulted in errors but after running the network again, the entire network ran without any errors."

> "There were some errors shown on MIDST, but after running the code node by node, the errors were fixed."

> "My code was in MIDST and I was able to easily run all my modules at one time. The only problem I had was while editing the code, there was an error message popping and it was taking too long to execute whereas the same code was running fast in R."

5. Discussion

By leveraging our maintainability model, it can be noted that MIDST improved the both the reproducibility and understandability of an analysis, and thus improved the maintainability of that analysis. Specifically, as shown in Figure 6, there are two key features, which are discussed below, that enabled this improvement. The two features are that (1) MIDST provides a complete, integrated code and execution environment, and that (2) MIDST provides a visual data flow view of the analysis that augments the code-based view of the analysis.

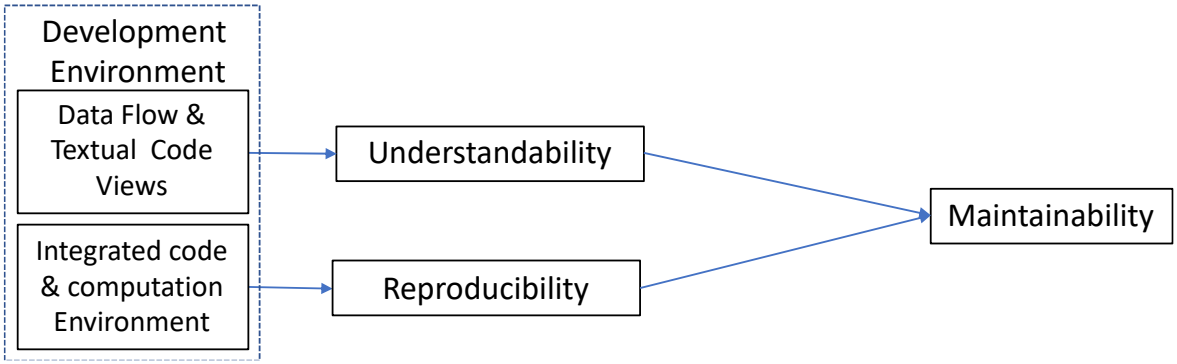

Fig. 6. Key Features of a development environment that leads to improved maintainability

5.1 MIDST's Integrated code and execution environment

One key feature of MIDST is that it provides the code, data, and full computational environment within an integrated environment. Our results show that this integrated environment improved the reproducibility of an analysis by enabling more of the code to be shared, in less time. It also helped enable more of the code (i.e., the analysis) to be executed by a new person.

Some teams in the "No MIDST" condition tried to mitigate this issue (of not having an integrated environment) by having each team member work on a specific analysis. Then, the results of the different analyses were "merged" via a high-level report (i.e., the final report was a collection of non-integrated efforts and the code was not merged into a shared repository/code base). Unfortunately, this approach created maintainability problems because others on the team could not easily understand or run the full analysis. An integrated code repository, such as Github [59], only partially solves this issue in that different team members might still have different execution environments – for example, different libraries installed or access to different data repositories. Hence, an integrated environment is necessary "in order to avoid the 'it runs on my machine' problem, where a project can only be reproduced by running it on the same computer it was originally written on" [24]. This problem is why it has been noted that sharing code and data is an important first step to improve reproducibility, but having a cloned computational environment "increases the ease and longevity of reproducible research projects" [24].

5.2 MIDST's visual data flow dataflow metaphor

MIDST's visual data flow metaphor might have improved the understandability of an analysis due to the fact that the visual data flow view mapped to a data scientists' mental model. With this in mind, we believe that MIDST creates an easier to understand analysis. Several participants noticed MIDST's ability to help improve the understanding the entire project, including comments such as:

> "It was important to understand the other team's project and the flow of data between all the nodes in order to update the code so as to get the code running without any errors."
>
> "I was able to understand the code due to the flow in MIDST."

> "The code provided on MIDST was put in an easy to understand fashion. I went through various nodes to understand their function. Overall, I did not face any major challenge while running the code."

6. Conclusion

6.1 Summary

This study defined and then used a new data science maintainability model, which is based on understandability and reproducibility. The study then used the model to evaluate a new data science interactive development environment (MIDST), which provides an integrated code and execution environment as well as a visual data flow of the analysis, was described and analyzed.

Specifically, to address our research question (*Does using visual data flow tool, within an integrated data science development environment, improve the ability of a new team member to reuse and update an existing analysis?),* a mixed method research study comparing the maintainability of an analysis developed in MIDST compared to an analysis developed within RStudio was performed.

The results of the study did answer the research question, in that the study demonstrated that using visual data flow tool does improve the ability of a new team member to reuse and update an existing analysis. In short, this research shows that a visual data flow tool, such as MIDST, improves the reproducibility and understandability of a data science analysis. Hence, the study highlights the need to explore the development of additional code-level visual data flow IDEs, to complement existing higher-level visual data flow IDEs that currently target higher-level end users who do not need to develop code.

6.2 Limitations and Next Steps

Based on our newly defined data science maintainability model, this study suggests that an analysis is easier to maintain when using MIDST. Future research should leverage these results to continue to explore how tools and processes could improve the maintainability of a data science analysis. Future research could also explore MIDST within an industry context.

One limitation of this research was that some of the findings, with respect to how long tasks took, were self-reported. Future research could explore different approaches to measure how long a task takes to complete, such as providing a fixed amount of time to update an analysis within a controlled environment, similar to what was done by Rule et al. [12]. Further refinements of the Maintainability Model could also be explored, such as whether there are alternative ways to operationalize the reproducibility and understandability variables. In addition, the bugs and performance issues within MIDST might have reduced the impact of the visual data flow approach. Hence, work will continue on improving the robustness and usability of MIDST.

Another possible next step is to note that another alternative to code-based programming environments, such as notebooks, which as previously noted, enables one to integrate R code, R output and formatted textual explanations of the analysis within one document. However, since notebooks do not naturally enable one to easily keep track of the output from one step (such as data cleaning) that is to be used across multiple next steps (such as different data analytical algorithms or visualizations), future research could explore the use of notebooks within a data flow context.

References

[1] M. Das, R. Cui, D. Campbell, G. Agrawal, and R. Ramnath. "Towards methods for systematic research on big data," IEEE International Conference on Big Data (Big Data), 2015.

[2] J. Saltz, and I. Shamshurin. "Big Data Team Process Methodology: A Literature Review and the Identification of Critical Factors for a Project's Success," IEEE International Conference on Big Data (Big Data), 2016.

[3] IEEE. Standard Glossary of Software Engineering Terminology, IEEE Computer Society Press, 1990.

[4] K. Henttonen, J. Kääriäinen, and J. Kylmäaho. “Lifecycle management in government-driven open source projects–practical framework,” International Journal of Information Systems and Project Management, vol. 5, no.3, 2017.

[5] R.Teubner. “IT program management challenges: insights from programs that ran into difficulties,” International Journal of Information Systems and Project Management, vol. 6, no. 2, 2018.

[6] A. Coners, and B. Matthies. “Perspectives on reusing codified project knowledge: a structured literature review,” International Journal of Information Systems and Project Management, vol. 6, no. 2, 2018.

[7] H. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. “Big data and its technical challenges,” Communications of the ACM, vol. 57, no. 7. 2014.

[8] R. Guo. Software tools to facilitate research programming, Ph.D. Dissertation, Stanford University, 2012.

[9] R. Muenchen. The Popularity of Data Analysis Software, http://r4stats.com/articles/popularity/, 2017.

[10] J. Racine. “RStudio: A Platform-Independent IDE for R and Sweave,” Journal of Applied Econometrics, vol. 27, no. 1, 2012.

[11] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. “Enterprise data analysis and visualization: An interview study,” IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 12, 2012.

[12] A. Rule, I. Drosos, A. Tabard, and J. Hollan. “Aiding Collaborative Reuse of Computational Notebooks with Annotated Cell Folding,” Proceedings of the ACM on Human-Computer Interaction, 2(CSCW), 2018.

[13] K. Subramanian, J. Maas, M. Ellers, C. Wacharamanotham, S. Voelker, and J. Borchers. “StatWire: Visual Flow-based Statistical Programming,” In Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems, ACM, 2018.

[14] J. Saltz. “The Need for New Processes, Methodologies and Tools to Support Big Data Teams and Improve Big Data Project Effectiveness,” IEEE International Conference on Big Data (Big Data), 2015.

[15] L. Thamsen, T. Renner, M. Byfeld, M. Paeschke, D. Schroder, and F. Böhm. “Visually programming dataflows for distributed data analytics,” IEEE International Conference on Big Data (Big Data), 2016.

[16] J. Pimentel, L. Murta, V. Braganholo, and J. Freire. “A large-scale study about quality and reproducibility of jupyter notebooks.,” In 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR) IEEE, 2019, pp. 507-517.

[17] M. Kery, B. John, R. O’Flaherty, A. Horvath, and B. Myers. “Towards Effective Foraging by Data Scientists to Find Past Analysis Choices,” In Proceedings of CHI Conference on Human Factors in Computing Systems Proceedings, 2019.

[18] A. Malviya, A. Udhani, and S. Soni. “R-tool: Data analytic framework for big data,” In 2016 Symposium on Colossal Data Analysis and Networking (CDAN) IEEE, 2016, pp. 1-5.

[19] A. Simmons, S.. Barnett, J. Rivera-Villicana, A. Bajaj and R. Vasa. “A large-scale comparative analysis of Coding Standard conformance in Open-Source Data Science projects,” In Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2020.

[20] M. Kery, M. Radensky, M. Arya, B. John, and B. Myers. “The story in the notebook: Exploratory data science using a literate programming tool,” In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, ACM, 2018.

[21] N. Davies, and S. Clinch. “Pervasive data science,” IEEE Pervasive Computing, vol. 16, no. 3, 2017.

[22] S. Dhar, and S. Mazumdar. “Challenges and best practices for enterprise adoption of big data technologies,” 2014 IEEE International Technology Management Conference, IEEE, 2014, pp. 1-4.

[23] V. Sachdeva, and L. Chung. “Handling non-functional requirements for big data and IOT projects in scrum, “In 2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence, IEEE, 2017, pp. 216-221.

[24] R. Tatman, J. VanderPlas, and S. Dane. “A Practical Taxonomy of Reproducibility for Machine Learning Research,“ In Reproducibility in Machine Learning Workshop at ICML, 2018.

[25] C. Boettiger. “An introduction to Docker for reproducible research,“ ACM SIGOPS Operating Systems Review, vol. 49, no. 1, 2015, pp. 71-79.

[26] B. Beaulieu-Jones, C. Greene. “Reproducibility of computational workflows is automated using continuous analysis,” Nature biotechnology, vol. 35, no. 4, 2017.

[27] Z. Usmani. https://www.kaggle.com/getting-started/44916, accessed March 1, 2019.

[28] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, ... & R. Ivanov. Jupyter Notebooks-a publishing format for reproducible computational workflows. In ELPUB, 2016, pp. 87-90.

[29] A. Culler and D. Culler. “Dataflow architectures,“ Ann. Rev. Comput. Sci., vol. 1, 1986.

[30] A. Davis, and R. Keller. “Data flow program graphs,” Computer, vol 2, 1982.

[31] C. Upson, T. Faulhaber D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, ... and A. Van Dam. “The application visualization system: A computational environment for scientific visualization, “ IEEE Computer Graphics and Applications, vol. 9, no. 4, 1989.

[32] D. Hils. “Visual languages and computing survey: Data flow visual programming languages,” Journal of Visual Languages & Computing, vol. 3, no. 1, 1992.

[33] A. Henley, and S. Fleming. “Yestercode: Improving code-change support in visual dataflow programming environments,” In 2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC. IEEE. 2016, pp. 106-114.

[34] L. Melander, K. Orsborn, T. Risch, and D. Wedlund. “VisDM–A Data Stream Visualization Platform,” In International Conference on Database Systems for Advanced Applications, 2017, pp. 677-680.

[35] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. “The weka data mining software: An update,” ACM SIGKDD Explorations Newsletter, ACM, vol. 11, no. 1, 2009, pp. 10–18.

[36] S. O’Hagan & D. Kell. “Software review: the KNIME workflow environment and its applications in Genetic Programming and machine learning,” Genetic Programming and Evolvable Machines, vol. 16, no. 3, 2015, pp. 387-391.

[37] F. Young and C. Bann. “ViSta: The Visual Statistics System, Technical Report,” UNC LL Thurstone Psychometric Laboratory Research Memorandum, 1996.

[38] J. Demšar, B. Zupan, G. Leban, and T. Curk. “Orange: From Experimental Machine Learning to Interactive Data Mining,” In Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD ’04), 2004, pp. 537–539.

[39] K. Subramanian, J. Maas, M. Ellers, C. Wacharamanotham, S. Voelker, and J. Borchers. “StatWire: Visual Flow-based Statistical Programming,” In Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems, ACM, 2018.

[40] T. Kluyver, B. Ragan-Kelley, F. Pérez, B.E. Granger, M. Bussonnier, J. Frederic, J., ... and P. Ivanov. “Jupyter Notebooks-a publishing format for reproducible computational workflows,” In ELPUB, pp. 87-90, 2016.

[41] C. Iriarte, and S. Bayona. “IT projects success factors: a literature review,” International Journal of Information Systems and Project Management, vol 8, no. 2, pp. 49-78. 2020.

[42] S. Morcov, L. Pintelon, and R. Kusters. “Definitions, characteristics and measures of IT project complexity-a systematic literature review,” International Journal of Information Systems and Project Management, vol. 8, no. 2, 2020, pp. 5-21.

[43] S. Muthanna, K. Kontogiannis, K. Ponnambalam, and B. Stacey. “A maintainability model for industrial software systems using design level metrics,” In Proceedings Seventh Working Conference on Reverse Engineering, IEEE, 2000, pp. 248-256.

[44] R. G. Dromey. “A model for software product quality,” IEEE Transactions on Software Engineering, vol. 2, pp. 146-163, 1995.

[45] S. Rizvi, and R. Khan. Maintainability Estimation Model for Object-Oriented Software in Design Phase (MEMOOD) (No. arXiv: 1004.4447), 2010.

[46] S. K. Dubey, and A. Rana. “Assessment of maintainability metrics for object-oriented software system,” ACM SIGSOFT Software Engineering Notes, vol. 36, no. 5, pp. 1-7. 2011.

[47] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk, and R. Oliveto. “Automatically assessing code understandability: How far are we?,” In Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, IEEE Press, 2017, pp. 417-427.

[48] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej. “How do professional developers comprehend software?,” In 34th International Conference on Software Engineering (ICSE), 2012, pp. 255–265.

[49] K. Shima, Y. Takemura, and K. Matsumoto. “An approach to experimental evaluation of software understandability,” In International Symposium on Empirical Software Engineering, 2002, pp. 48–55.

[50] S. Schneberger. “Distributed computing environments: effects on software maintenance difficulty,” Journal of Systems and Software, vol. 37, pp. 101–116, 1997.

[51] J. Lim, S. Jeong, and S. Schach. “An empirical investigation of the impact of the object-oriented paradigm on the maintainability of real-world mission-critical software,” Journal of Systems and Software, vol. 77, no. 2, 2005.

[52] K. Punch. Introduction to Social Research: Quantitative and Qualitative Approaches, 2nd ed., Sage Publications, Newbury Park, California, pp. 197-202, 2005.

[53] D. Silverman. Qualitative data analysis: interpreting talk, text and interaction, 3rd ed., Sage Publications, Newbury Park, California, 2006.

[54] D. Sjøberg, J. Hannay, O. Hansen, V. Kampenes, A. Karahasanovic, N. Liborg, and A. Rekdal. “A survey of controlled experiments in software engineering,” IEEE Transactions on Software Engineering, vol. 31, no. 9, 2005.

[55] Z. Soh, Z. Sharafi, B. Van den Plas, G. Porras, Y. Guéhéneuc, and G. Antoniol. “Professional status and expertise for UML class diagram comprehension: An empirical study,” IEEE 20th International Conference on Program Comprehension, 2012, pp. 163-172.

[56] A. Ko, T. LaToza, and M. Burnett. “A practical guide to controlled experiments of software engineering tools with human participants,” Empirical Software Engineering, vol. 20, no. 1, pp. 110-141, 2015.

[57] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam, and J. Rosenberg. “Preliminary guidelines for empirical research in software engineering,” IEEE Transactions on Software Engineering, vol. 28, no. 8, pp. 721-734, 2002.

[58] I. Salman, A. Misirli, and N. Juristo. “Are students representatives of professionals in software engineering experiments?,” In Proceedings of the 37th International Conference on Software Engineering-Volume 1, 2015, pp. 666-676.

[59] B. Beer, Introducing GitHub: A non-technical guide. 2nd ed., Sebastopol, CA, USA, O'Reilly Media, Inc. 2018.

Biographical notes

Jeffrey Saltz
Jeffrey Saltz is an Associate Professor at Syracuse University, where he leads their graduate applied data science program. His research focuses on agile data science project management. Prior to joining Syracuse, he worked as the head of technology for risk and authorizations for Chase Credit Card, Vice President of computational technology for JP Morgan and Chief Technology Officer at Goldman Sachs/Goldman Sachs Ventures. He started his career as a technology leader with Digital Equipment Corp and holds a B.S. degree in computer science from Cornell University, an M.B.A. from the Wharton School at the University of Pennsylvania and a Ph.D. in information systems from the New Jersey Institute of Technology.

Kevin Crowston
Kevin Crowston is a Distinguished Professor of Information Science at Syracuse University. He received his A.B. in Applied Mathematics (Computer Science) from Harvard University and a Ph.D. in Information Technologies from the Sloan School of Management, Massachusetts Institute of Technology. He currently serves as Associate Dean for Research. His research examines new ways of organizing made possible by the use of information technology. He approaches this issue in several ways: empirical studies of coordination-intensive processes in human organizations (especially virtual organization); theoretical characterizations of coordination problems and alternative methods for managing them; and design and empirical evaluation of systems to support people working together.

Robert Heckman
Robert Heckman is Professor Emeritus at Syracuse University. He has previously served as Graduate Program Director, Associate Dean for Academic Affairs, and Senior Associate Dean in the School of Information Studies at Syracuse. His research interests include design of work-based learning experiences, learning strategies for information professionals, self-directed learning, and collaboration in virtual communities and teams. Previously, he had more than 20 years of experience in the information services industry as a senior manager of data processing operations, systems development, and information systems marketing. He received his Ph.D. in information systems from the Katz Graduate School of Business, University of Pittsburgh.

Yatish Hegde
Yatish hegde is a Lecturer in A. B. Freeman School of Business at Tulane University. He has a wealth of experience leading the system development for research projects in Machine Learning, Natural Language Processing and Information Systems domains. Yatish also has considerable experience teaching data science at both graduate and under-graduate levels.