# TrueGrid: Code the Table, Tabulate the Data

Felienne Hermans and Tijs van der Storm

TU Delft and CWI
f.f.j.hermans@tudelft.nl
storm@cwi.nl

**Abstract.** Spreadsheet systems are live programming environments. Both the data and the code are right in front you, and if you edit either of them, the effects are immediately visible. Unfortunately, spreadsheets lack mechanisms for abstraction, such as classes, function definitions etc. Programming languages excel at abstraction, but most mainstream languages or integrated development environments (IDEs) do not support the interactive, live feedback loop of spreadsheets. As a result, exploring and testing of code is cumbersome and indirect.
In this paper we propose a method to bring both worlds closer together, by juxtaposing ordinary code and spreadsheet-like grids in the IDE, called TrueGrid. Using TrueGrid spreadsheet cells can be programmed with a fully featured programming language. Spreadsheet users then may enjoy benefits of source code, including added abstractions, syntax highlighting, version control, etc. On the other hand, programmers may leverage the grid for interactive exploring and testing of code. We illustrate these benefits using a prototype implementation of TrueGrid that runs in the browser and uses Javascript as a programming language.

## 1 Introduction

Spreadsheets are very popular tools for end-user programming. Their formula language is easy to learn and their grid interface is inviting. Apart from this, spreadsheets are *live*: the user interface reacts immediately to changes in input or code. Live programming helps bridging the gulf between code and behavior because the user receives immediate feedback on their actions [8]. More recently, live programming has found its way to a wider audience, for instance, by Bret Victor's influential talk *Inventing on Principle* [11]. Figure 1, taken from Victor's talk, illustrates the idea of live programming: on the right, we have source code and on the left, we have the result of that code, in this case, a tree drawing. Modifying the code will immediately affect the visual representation of the tree.

This liveness is one of the core features of most spreadsheet systems. When a users enters a formula and presses enter, they see the result, without a lengthy edit-compile-run cycle. This live characteristic of spreadsheets is often praised as their key success factor [3]. However, spreadsheets have a number of downsides too. For instance, the lack of abstraction mechanisms forces spreadsheet users to use copy-paste for reusing code. Although solutions to this problem have been researched [4, 6], they do not provide the power of full programming languages

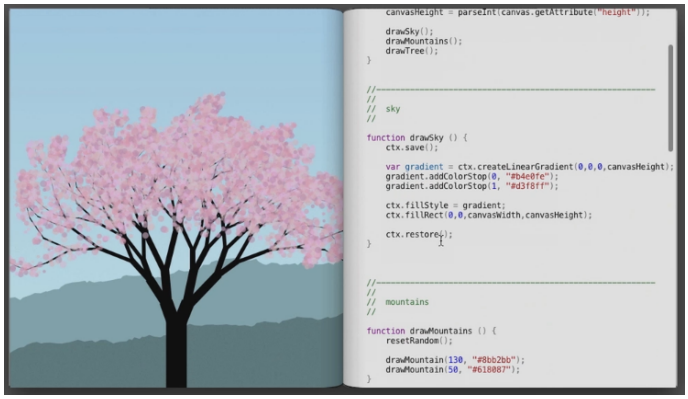


**Fig. 1.** Live programming: on the right the source code and on the left its instantiation of the code which changes immediately when the code is updated, screenshot from [11].

to spreadsheet users. Secondly, the way formulas are edited in a spreadsheet system like Excel has important drawbacks from the programming perspective. For instance, Excel's formula editor lacks even the basic editor services, such as syntax highlighting and reference resolution. In general, the interface is not inviting to apply proper coding styles and practices. Another disadvantage of embedding the code within the sheet itself is that it prohibits versioning and sharing of the formulas separately from the data.

On the other hand, there is source code. All existing programming languages support abstraction, and modern editors help developers understand and structure their code. However, mainstream integrated development environments (IDE) lack the live and interactive style of interaction so coveted by spreadsheet users.

In this paper, we will describe *TrueGrid*, a light-weight approach to bridging the gap between programming and spreadsheets and describe ts implications in both worlds. The basic concept of TrueGrid is to allow a spreadsheet like grid to be programmed using a full featured programming language, in a consistent user interface. Figure 2 shows this idea as implemented in our prototype. The key charactertistic is that developers see the code and data at the same time. Furthermore, like a spreadsheet, TrueGrid is live, i.e. on a change of data or code, the grid is updated. In our example, we use JavaScript as a language, however, the idea itself is not limited to a single programming language.

In the next section we explore the implication of the TrueGrid user interface for spreadsheet users. Section 3 explores TrueGrid from the perspective of programmers. In particular we discuss conventions for relating code to the grid view.

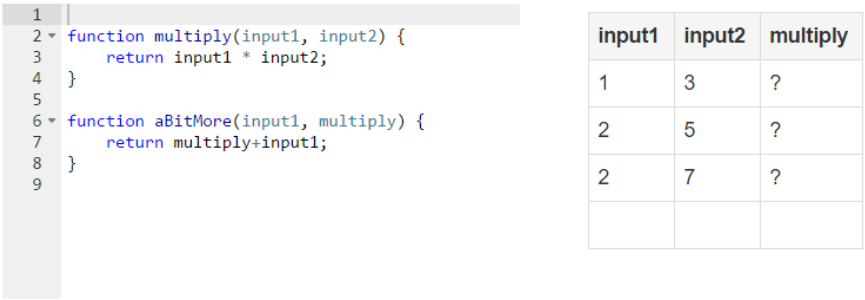


**Fig. 2.** A True Grid implementation, with the code editor on the left hand side and the grid on the righ, available via http://www.felienne.com/TrueGrid/

```
function column_avg(row) {
  return (row.lab + row.exam) / 2;
}

function cell_classAvg(col, row) {
  return sum(col) / col.length;
}
```

| student | lab | exam | avg |
|---|---|---|---|
| Rich | 7 | 8 | 7.5 |
| Jacome | 8 | 9 | 8.5 |
| Birgit | 9 | 9 | 9 |
| | | | *classAvg:* 8.3 |

**Fig. 3.** Mockup of TrueGrid for spreadsheet use

## 2 TrueGrid for Spreadsheet Users

As an implementation of TrueGrid for spreadsheet users, we hypothesize that existing spreadsheet system could be extended with a source code editor view, next to the grid-based view of the data. This allows users to use advanced features like PivotTables and charts on top of their TrueGrid, but also use professional IDE services for expressing computations. For spreadsheet users, using TrueGrid over spreadsheets presents several benefits. For example, the use of a professional editor supports the developer with editor services like syntax highlighting and error marking. Furthermore, the textual form of the code allows for easy diffing and merging, enabling more mature version control on spreadsheets. While learning a new programming language can be challenging, there are spreadsheet developers working with VBA now, which is a fully featured language. Unfortunately the integration between the code and a spreadsheet is low level and cumbersome. We envision TrueGrid having a less steep learning curve, because code and data are juxtaposed.

As an example consider the simple grade book sheet shown in Fig. 3. It shows the actual data (both provided and computed) in the grid. The average and class average are expressed using ordinary Javascript functions. The TrueGrid environment links functions or methods to the cells in the grid using a naming

```
function leftpad(str, len, ch) {
  str = String(str);
  var i = -1;
  if (!ch && ch !== 0) ch = ' ';
  len = len - str.length;
  while (++i < len) {
    str = ch + str;
  }
  return str;
}
```

| function | 0 | 1 | 2 | Result |
|---|---|---|---|---|
| leftpad | "foo" | 5 | | " foo" |
| leftpad | "foobar" | 6 | | "foobar" |
| leftpad | 1 | 2 | 0 | "01" |

**Fig. 4.** Exploring function behavior using TrueGrid

convention. For instance, a function starting with the `column_` prefix computes a complete column, given a particular row (an ordinary Javascript object). This is used in the computation of the `avg` column, where each cell contains the average of the lab and exam cells. The model also allows naming individual cells. This is illustrated in the function `cell_classAvg`. The function receives the current column (`col`, a Javascript array), and the current `row`. Based on the elements in the column the class average is computed.

One could imagine linking the code to the grid using row and column coordinates, just like ordinary spreadsheet formulas refer to (ranges of) rows and columns. However, this would lead to code that is not very intuitive if read separately. Furthermore, insertion of rows and columns in the grid would require updating the source code itself, similar to how spreadsheet systems realign cell coordinates.

## 3 TrueGrid for Developers

Like spreadsheet users, developers often work with (example) data when programming. For instance, developers use read-eval-print-loops (REPLs) to explore the behavior of a function. Another use case is developer testing which requires setting up test fixtures and inspecting the results. Both these use cases, however, suffer from a lack of immediacy. After a change to the code, the developer needs to reenter expressions in the REPL to observe expected changes in behavior. Similar for testing: reexecuting the test is an explicit step after every change to the code. TrueGrid eliminates these hickups and promises a more fluent, live experience.

In particular, TrueGrid can be seen as a persistent REPL, where expressions or method invocations are continuously evaluated, after every change to the code or input data. Consider the example shown in Fig. 4. On the left is a simple Javascript function for left-padding values with spaces or other padding characters. Using TrueGrid, the programmer can provide example data in the grid and explore the implementation. This can be especially valuable early in

the development process when you have some data and only a vague idea of how certain functionality should be implemented.

From the testing perspective, TrueGrid provides a kind of "FIT testing on steroids" [9], where the grid functions as a live dashboard of test success and failure. In this case, the grid shown in Fig. 4 could have an additional column indicating the success or failure of the function execution, or use green/red coloring of rows to the same effect. Again, the success indicators would be automatically updated upon changing the source code on left.

## 4 Related Work

This work is positioned at the intersection of end-user programming and professional software development [7]. The particular link between software engineering and spreadsheet has been explored before. For instance, Cunha et al. present model-based programming environments for spreadsheets [2]. The focus of this work is to improve reliability of spreadsheet engineering by applying model-driven techniques. In this work we primarily focus on improving programmer experience.

Integrating spreadsheet-based user interfaces with code editors is a simple form of a heterogeneous programming environment (e.g., [10]), originally pioneered in the work on structure editors, and recently popularized by the Jetbrains Meta Programming System [5]. Although this kind of work aims for a much more invasive integration, it is in line with the goal of having best user interfaces for each aspect of the code. A similar strand of research is explored in the context of DSLs by Adam and Schultz [1].

## 5 Conclusion and Outlook

Spreadsheets are live programming environments. However, their lack of abstraction mechanisms and editor support are impediments to professional spreadsheet development. Conversely, traditional programming environments lack the continuous feedback that makes spreadsheets so attractive. In this paper we have presented TrueGrid, a user interface design combining code editing and grid-based data view, with a live execution model.

TrueGrid presents a promising bridge between the domain of spreadsheets and software development. It has the potential to improve end-user programming experience from two perspectives:

- For spreadsheet users: computations are expressed using program code, instead of formulas in cells, so that end-users may enjoy both abstraction and liveness at the same time.
- For professional developers: spreadsheet-like grids support live exploration and testing of code, without explicitly invoking test scripts of entering expressions in a REPL.

We have implemented a prototype of TrueGrid which runs in the browser, using Javascript as the programming language. Further research is needed to empirically investigate the benefits of TrueGrid from both the programmer and spreadsheet user perspectives. We expect that TrueGrid provides a fruitful vehicle for exploring the middle ground between end-user programming on the one hand side, and professional software development on the other.

## References


1. Adam, S., Schultz, U.P.: Towards tool support for spreadsheet-based domain-specific languages. In: Proceedings of the 2015 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences. pp. 95–98. ACM (2015)
2. Cunha, J., Mendes, J., Saraiva, J., Visser, J.: Model-based programming environments for spreadsheets. Science of Computer Programming 96, 254–275 (2014)
3. Hermans, F.: Analyzing and Visualizing Spreadsheets. Ph.D. thesis, Delft University of Technology (2013)
4. Hermans, F., van der Storm, T.: Copy-paste tracking: Fixing spreadsheets without breaking them. In: Proceedings of the International Conference on Live Coding (2015)
5. Jetbrains: Meta programming system. Online (2016), `https://www.jetbrains.com/mps/`
6. Jones, S.P., Blackwell, A., Burnett, M.: A user-centred approach to functions in excel. ACM SIGPLAN Notices 38(9), 165–176 (2003)
7. Ko, A.J., Abraham, R., Beckwith, L., Blackwell, A.F., Burnett, M.M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B.A., Rosson, M.B., Rothermel, G., Shaw, M., Wiedenbeck, S.: The state of the art in end-user software engineering. ACM Comput. Surv. 43(3), 21 (2011), `http://doi.acm.org/10.1145/1922649.1922658`
8. Lieberman, H., Fry, C.: Bridging the gulf between code and behavior in programming. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 480–486. CHI '95, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1995), `http://dx.doi.org/10.1145/223904.223969`
9. Mugridge, R., Cunningham, W.: Fit for Developing Software: Framework for Integrated Tests. Prentice Hall (2005)
10. Schrage, M.M., Swierstra, S.D.: Beyond ASCII – parsing programs with graphical presentations. J. UCS 14(21), 3414–3430 (2008)
11. Victor, B.: Inventing on principle (2012), `http://vimeo.com/36579366`