Wildcard: Spreadsheet-Driven Customization of Web Applications

Geoffrey Litt[a] and Daniel Jackson[a]

a Massachusetts Institute of Technology

Abstract Many Web applications do not meet the particular needs of their users. Browser extensions and user scripts offer a way to customize web applications, but most people do not have the programming skills to implement their own extensions.

We present the idea of *spreadsheet-driven customization*: enabling end users to customize existing applications using a live spreadsheet view of the data inside the application. By manipulating the spreadsheet, users can implement a wide variety of customizations, ranging from sorting lists of search results to displaying related data from other web services, without doing any traditional programming.

We built a prototype system called Wildcard that implements spreadsheet-driven customization as a web browser extension. Through concrete examples, we demonstrate that Wildcard has both a low barrier to entry for beginners and enough flexibility to solve many useful problems. We also show that Wildcard can work with real existing websites, by extracting structured data using web scraping techniques.

For an online version of this paper with videos demonstrating the interface, see https://www.geoffreylitt.com/wildcard/

ACM CCS 2012

- *General and reference → Computing standards, RFCs and guidelines;*
- **Applied computing → Publishing;**

Keywords end-user programming, software customization, web browser extensions

The Art, Science, and Engineering of Programming

Perspective The Art of Programming

Area of Submission Programming environments, Visual and live programming

1 Introduction

Web applications often don't match the particular needs of their users. Sometimes there is a browser extension available to patch an issue, and if the user is a programmer they might be able to fix it themselves. But for most people, the only recourse is to complain to the developers, or more likely, to simply give up. Back in 1977, in *Personal Dynamic Media* [16], Alan Kay envisioned personal computing as a medium that let a user "mold and channel its power to his own needs," but today, software feels more like concrete than clay.

In this paper, we present *spreadsheet-driven customization*, a technique that enables end users to customize software. The idea is to augment an application's UI with a spreadsheet that is synchronized with the application's data. When the user manipulates the spreadsheet, the underlying data is modified and the changes are propagated to the UI, and vice versa. We have implemented this technique in a prototype browser extension called Wildcard. Section 2 presents demos of using Wildcard to augment real websites in useful ways, and Section 3 describes the implementation of the extension.

Spreadsheet-driven customization provides an easy entry point for end users, since small tweaks can be performed with a single click. At the same time, it also supports a variety of richer customizations, like adding private annotations to a webpage or joining in related data from a web API. In Section 4, we elaborate on this principle of "low floor, high ceiling," as well as other design principles guiding our work.

Prior work [4, 6, 21] has enabled end users to create "spreadsheet-driven applications" which use spreadsheets as a backing data layer. Spreadsheet-driven *customization* applies this idea in a different context: customizing existing software, rather than building new software from scratch. Our technique does not require that the application actually be backed by a spreadsheet; it merely uses the spreadsheet as an interface for viewing and modifying the internal state of the application.

This approach requires extracting structured data from the user interfaces of existing applications, but we hide the complexity of data extraction from end users. Programmers write *site adapters* which use web scraping techniques to extract structured data from existing applications and map them to the spreadsheet table. End users only interact with the structured spreadsheet, providing a straightforward customization experience. In Section 5, we explain this architecture in more depth, and describe how Wildcard relates to existing work on malleable software, web customization, spreadsheet-driven applications, and web scraping.

The Wildcard extension is currently an early research prototype. We plan to continue testing the system with our own use cases to understand better how well the spreadsheet abstraction maps to real websites and customization needs. Eventually we plan to release the tool publicly, in order to study how end users choose to customize applications, discover usability challenges, and to test the feasibility of programmers building and maintaining site adapters.

2 Demo: booking a trip with Wildcard

To get a sense of the user experience of using Wildcard, let's see an example of someone using it to help with booking a trip. These demos are best viewed as videos in the online version of this paper (https://www.geoffreylitt.com/wildcard).

In 2012, the travel site Airbnb removed the ability to sort accommodation listings by price. Users could still filter by price range, but could no longer view the cheapest listings first. Many users complained on online message boards that the change seemed hostile to users. "It's so frustrating!..What is the logic behind not having this function?" said one user on the Airbnb support forum. Alas, the feature remains missing to this day.

Using Wildcard, the user can fix this omission, while leaving the page's design and the rest of its functionality unchanged. fig. 1 shows an overview of augmenting the Airbnb site. First, the user opens up the Wildcard panel, which shows a table corresponding to the search results in the page. As they click around in the table, the corresponding row in the page is highlighted so they can see the connection between the views.

Then, the user clicks on the price column header to sort the spreadsheet, and the Airbnb UI, by in ascending order by price (fig. 1B). They can similarly use the column headers to filter to listings with a user rating above 4.5, which the original Airbnb UI also doesn't allow.

After manipulating the data, the user closes the table view and continues using the website with its original visual design. The table view offers a way to change the data backing a page, but does not need to replace the original interface entirely.

Most websites that show lists of data also offer actions on rows in the table, like adding an item to a shopping cart. Wildcard has the ability to make these "row actions" available in the data table through the site adapter.

In the Airbnb UI, saving multiple listings to a Favorites list requires tediously clicking through them one by one. Using Wildcard, the user can select multiple rows and favorite all of them with a single click (fig. 1D). Similarly, the user can also open the detailed pages for many listings at once.

Now the user wants to jot down some notes on the pros and cons of each listing. To do this, they type some notes into an additional column in each listing row, and the notes appear inside the listings in the original UI (fig. 1A). The annotations are saved in the browser for future sessions.

Wildcard also includes a formula language, which enables more sophisticated customizations that fetch external data and perform computations.

When traveling without a car, it's useful to evaluate potential places to stay based on how walkable the surroundings are. Using Wildcard formulas, the user can integrate Airbnb with Walkscore, an API that rates the walkability of any location on a 1-100 scale. When they type in a `WALKSCORE` formula with the latitude and longitude of the listing, it returns the walk score as the cell's value. Because the cell's contents are injected into the page, the score also shows up in the page body (fig. 1C).

It might seem that Wildcard is only useful on websites that display lists of tabular data like search results. But in fact, the table metaphor is flexible enough to represent

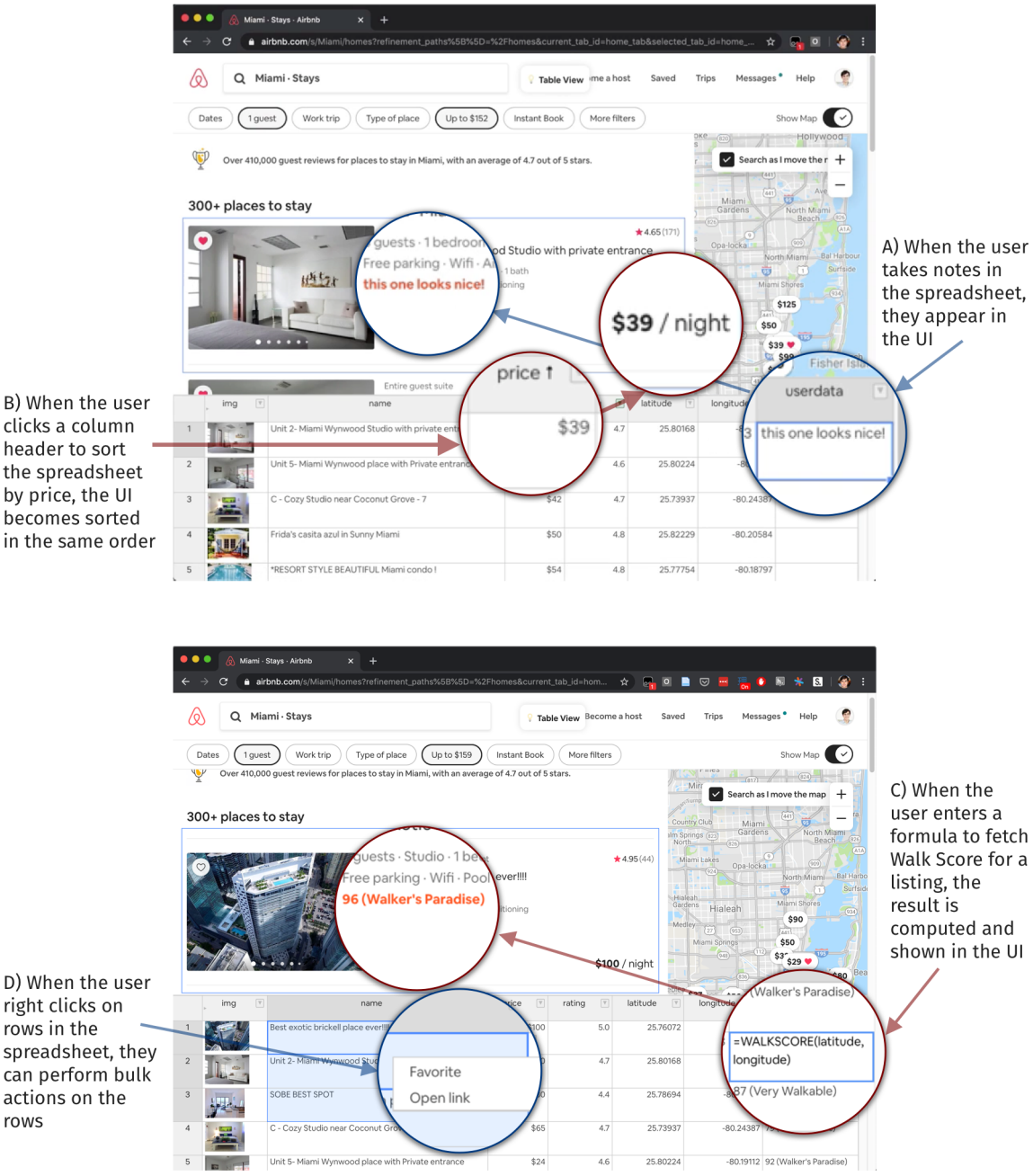

Figure 1 Using Wildcard to augment the Airbnb search page for booking accommodations

A) When the user selects a field in the spreadsheet, the corresponding form field is highlighted in the UI

B) The user selects a date using a datepicker widget with access to their private calendar. The value is set in the spreadsheet and the Expedia UI

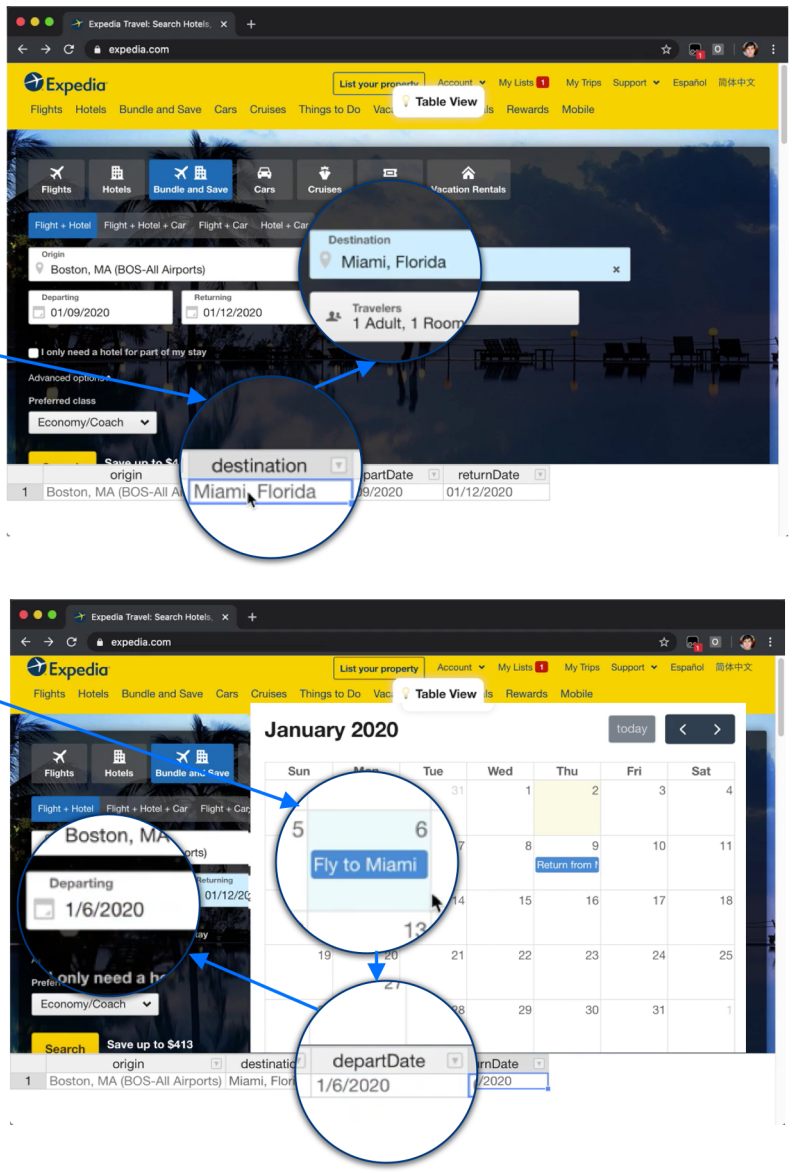

Figure 2 Using Wildcard to augment the Expedia page for booking a flight

many types of data. For example, a flight search form on Expedia can be represented as a single row, with a column corresponding to each input. fig. 2 shows an overview of augmenting the Expedia site.

In previous examples the table cells were read-only; users cannot change the name or price of an Airbnb listing. In this next case, the cells are writable, so that changes in the table are reflected in the form inputs. This becomes especially useful when combined with GUI widgets that can edit the value of a table cell.

Filling in dates for a flight search typically requires opening up a separate calendar app to find the right dates and then manually copying them into the form. In Wildcard, the user can avoid this cumbersome workflow by using a datepicker widget that includes the user's personal calendar information (fig. 2B). The user can directly click on the right date, and it gets inserted into the spreadsheet and the original form.

Here we've presented just a few use cases for spreadsheet-driven customization, but we think the interactive data table is flexible enough to support a wide range of other useful modifications, all while remaining familiar and easy to use.

3 System Architecture

Wildcard is written in Typescript. It is currently injected into pages using the Tampermonkey userscript manager, but in the future we plan to deploy it as a standalone browser extension to make it easier to install.

In order to promote extensibility, Wildcard is implemented as a small core program along with several types of plugins: site adapters, formulas, and cell renderers/editors. The core contains functionality for displaying the data table and handling user interactions, and the table implementation is built using the Handsontable Javascript library.

Site adapters are a key part of Wildcard, since they specify the bidirectional connection between the web page and its structured data representation.

Wildcard provides an interface for concisely expressing how the DOM should be mapped to a data table. For example, here is a code snippet for extracting the name of an Airbnb listing:

```
{
  fieldName: "name", // The name of the field
  type: "text",      // The type of the field
  readOnly: true,    // Whether the user can edit the field
  el: (row) => row.querySelector(.${titleClass}), // Extract DOM element
}
```

Sometimes important data is not shown in the UI, making it impossible to scrape from the DOM. To address this, we have also prototyped mechanisms for site adapters to observe AJAX requests made by the browser and extract data directly from JSON responses. This mechanism was used to implement the Airbnb Walkscore example, since latitude and longitude aren't shown in the Airbnb UI, but they are available in AJAX responses. In the future we also plan to add the ability for site adapters to

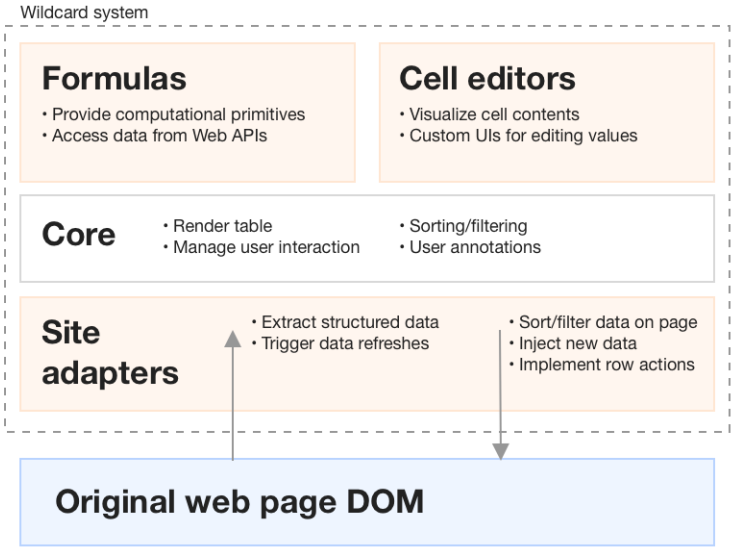

Figure 3 The architecture of the Wildcard system

scrape data across multiple pages in paginated lists of results (as explored in prior work [12]).

The site adapter also needs to support the reverse direction: sending updates from the table to the original page. Most DOM manipulation is not performed directly by the site adapter. Instead, Wildcard automatically mutates the DOM to reflect the table state, using the same declarative spec shown above. The only exception is row actions (like favoriting an Airbnb listing), which are implemented as imperative Javascript functions that can can mutate the DOM, simulate clicks on buttons, etc.

4 Design principles

The idea of spreadsheet-driven customization is guided by several design principles, inspired by prior work and our own experimentation. We think these principles can also broadly inform the design of tools for end user software customization.

4.1 Expose a universal data structure

Today, most personal computing consists of using applications, which bundle together behavior and data to provide some set of functionality. While there are some limited points of interoperability, applications generally are designed to operate independently of one another.

Computing does not need to be organized this way. For example, UNIX offers a compelling alternative design: many small single-purpose utilities, all of which manipulate a universal format of text streams. The universal format creates a high

degree of leverage from tools: users can get a lot of utility from deeply mastering a text editor and some text manipulation utilities, because these tools can be applied to nearly any task. As just one example, a user's preferred text editor can even serve as an interactive input mechanism in shell programs, e.g. for editing git commit messages.

Spreadsheet-driven customization aims to bring some of this UNIX ethos to the world of isolated Web applications, by creating a consistent data structure to represent the data inside many applications. In UNIX, the universal format is a text stream; in Wildcard, it is a relational table. The table is a simple abstraction which is also generic enough to describe the data used in many different applications. Because Wildcard maps the data from all applications to the table format, users can invest in mastering the Wildcard table editor, the formula language, and cell editor UIs, and reuse those same tools to customize many different applications.

This idea relates to Beaudouin-Lafon and Mackay's notion of *polymorphic interface instruments* [3]: UI elements that can be used in different contexts, like a color picker that can be used in many different drawing applications. diSessa has also noted the connection between literacy and the genericness of a medium. Textual literacy rests on the fact that writing can be adapted to many different genres and uses [10]; if people needed to relearn reading and writing from scratch when switching from essays to emails, the medium would lose most of its potency. We think providing generic tools is especially important for software customization, because the most common barrier to customizing software is not having enough time [20]—it's more likely that people will customize software regularly if they can reuse the same tool across many applications.

This design principle leads to several challenges. First, any universal abstraction has its constraints, and can't necessarily naturally express the data in every application. We plan to explore the limits of the table abstraction further, by trying to build adapters for more sites with varied data formats. We expect that many types of data can fit fairly naturally into tables: lists of search results, news articles, and messages can all be seen as relations. On sites that use document structures (e.g. Google Docs) or graph structures (e.g. social friend graphs), it may prove more challenging to map internal data to this abstraction.

Another challenge is ensuring a clear mapping in the user's mind between the spreadsheet and the original page. Wildcard provides live visual cues as the user navigates the data table (similar to the highlighting provided by DOM inspectors in browser developer tools). In our own usage, we have found that this live highlighting makes it very clear how the two representations map to each other.

4.2 Low floor, high ceiling

Seymour Papert advocated for programming systems to have a "low floor," making it easy for novices to get started, and a "high ceiling," providing a large range of possibilities for more sophisticated users [24]. Our goal is for spreadsheet-driven customization to meet both of these criteria.

One of the most interesting properties of spreadsheets is that users who are only know a tiny sliver of their functionality (e.g., storing tables of numbers or computing

simple sums) can still use them in genuinely valuable ways. The fact that useful tasks can be performed early on supports the user's natural motivation to continue using the tool, and to eventually learn its more powerful features if needed [23]. In contrast, many traditional programming systems require an enormous upfront investment of time and practice before someone is able to write a program that helps them achieve a useful task in their life.

As part of ensuring a low floor, we have focused on including genuinely valuable features for novices. For example, a user can sort a table with a single click, or simply type in some annotations. We would expect many Wildcard users to start by using these simpler features before potentially moving on to more sophisticated features like formulas.

Another aspect of providing a low floor is providing an "in-place toolchain" [15]: minimizing the effort of moving from using to customizing, by making customization tools available in the same environment where the user is already using the software. This quality is distinct from the level of technical skill needed to use the tool. For example, setting up a workflow trigger in an end user programming system like IFTTT does not require much technical skill, but does require leaving the user's normal software and entering a separate environment; conversely, running a Javascript snippet in the browser console requires programming skills, but can be done immediately and casually in the flow of using a website.

	In-place	*Not in-place*
End user friendly	**Wildcard**	IFTTT
Requires programming	browser JS console	forking open source software

Wildcard provides an in-place toolchain because the spreadsheet can be instantly opened in the browser window while using any supported website. Once the user starts editing, Wildcard also provides live feedback, so that even if a user isn't yet totally familiar with Wildcard, they can learn to use the system through experimentation.

Since we have still only built several site adapters and demos, it is still too early to tell exactly how high the ceiling is for the customizations that can be achieved with Wildcard. But we think that with enough operators, the formula language could support a wide variety of customizations—people have managed to use simple spreadsheet formula languages to solve a surprisingly large range of problems. We plan to explore this aspect further by trying to solve more real problems with the system and observing where limitations emerge in practice.

4.3 Build for multiple tiers of users

Real-world spreadsheet usage in offices is highly collaborative: most users just perform simple changes, while a few coworkers help with writing more complex formulas or even programming macros [22]. Inspired by this collaborative approach, we aim to make spreadsheet-driven customization a collaborative activity that combines the different abilities of many users.

The main way we do this is by separating website customization into two separate stages: structured data extraction, and using the resulting spreadsheet. The data extraction stage is currently only available to programmers who can code site adapters in Javascript, whereas the customization stage is available to any non-programmer end user. This architecture frees end users from needing to think about data extraction, and enables a community of end users to reuse the efforts of programmers building site adapters.

The group of users building adapters does not necessarily need to be limited only to programmers. In the future, we plan to explore enabling end users to also create site adapters, drawing on related work in this area [7, 12]. But even in that case, we still envision a separation between highly motivated, tech-savvy end users building adapters, and more casual end users just using the spreadsheet view.

Another stakeholder to consider is the first party developers of the original software. Spreadsheet-driven customization does not depend on cooperation from first-party website developers, but if they were to expose structured data in their web application clients, it would make site adapters unnecessary, producing more robust results with much less work.

We think there are compelling reasons for first parties to consider doing this. Providing Wildcard support would allow users to build extensions to fulfill their own feature requests. It also would not necessarily require much effort: adding Wildcard support could be made fairly straightforward for a first-party that already has direct access to the structured data in the page. There is also precedent for first parties implementing an official client extension API in response to user demand: for several years, Google maintained an official extension API in Gmail for Greasemonkey scripts to use.[1]

5 Related work

5.1 Malleable software

In the broadest sense, Wildcard is inspired by systems aiming to make software into a dynamic medium where end users frequently create and modify software to meet their own needs, rather than only consuming applications built by programmers. These systems include Smalltalk [16], Hypercard [14] , Boxer [9], Webstrates [17], and Dynamicland [27].[2]

While similar in broad goals, Wildcard employs a different solution strategy. These projects generally require building software from scratch in a new environment,

[1] Incidentally, since then, third parties have continued to maintain stable Gmail extension APIs used by many browser extensions [25, 26], illustrating the potential of collaboratively maintaining third party adapters.

[2] The project's name Wildcard comes from the internal pre-release name for Hypercard, which doubly inspired our work by promoting both software modification by end users and the ideas behind the Web.

whereas Wildcard aims to maximize the malleability of software built with existing tools. This approach has the pragmatic benefit of enabling customization for much more software, although it also imposes more rigid constraints.

With substantial future work, we think Wildcard could become more similar to these other projects, growing from a platform for tweaking existing software into a platform for building new software from scratch. This would likely end up resembling existing tools for building spreadsheet-driven applications (discussed more below), but with an additional focus on customizability by end users of the software.

5.2 Web customization

Wildcard's goals are closely shared with other systems that provide interfaces in the browser for end users to augment and customize websites while using them.

5.2.1 Structured augmentation

Wildcard's solution approach is most similar to other tools that identify structured data in a web page, and then use that structure to support end user customization of the page.

Sifter [12] enables users to sort and filter lists of data on web pages, which resembles Wildcard's sort and filter functionality. The underlying mechanism is also similar, since Sifter extracts structured data from the page to enable its user-facing functionality. The systems also have major differences. Wildcard aims to extend this approach to support much broader functionality, and shows the structured data table directly to the user. In contrast, Sifter only supports sorting and filtering, and does not reveal the underlying data table. The extraction mechanism is also different: Sifter uses a combination of automated heuristics and interactive user feedback to enable end users to extract data, whereas Wildcard hides data extraction from end users.

Thresher [11] enables users to create wrappers which map unstructured website content to Semantic Web content. Like Wildcard and Sifter, Thresher augments the experience of original page based on identifying structure; once semantic content has been identified, it creates context menus in the original website which allow users to take actions based on that content. Wildcard and Thresher focus on complementary parts of the problem. Thresher aims to enable end users to create content wrappers, but the actions available on the structured data are created by programmers; conversely, Wildcard delegates wrapper creation to programmers but gives end users more flexibility to use the structured data in an open-ended way.

5.2.2 Sloppy augmentation

"Sloppy programming" [19] tools like Chickenfoot [5] and Coscripter [18] enable users to create scripts that perform actions like filling in text boxes and clicking buttons, without directly interacting with the DOM. Users express the desired page elements in natural, informal terms (e.g. writing "the username box" to represent the textbook closest to the label "username"), and then using heuristics to determine which elements most likely match the user's intent. This approach allows for expressing a wide variety of commands with minimal training, but it also has downsides [19]:

- Reliability: It can be difficult to know whether a command will consistently work over time. In addition to changes to the website, changes to the heuristics can also cause problems.
- Discoverability: it can be difficult for users to discover the space of possible commands.

Wildcard offers a sharp contrast to sloppy programming, instead choosing to expose a high degree of structure through the familiar spreadsheet table. Wildcard offers more consistency; for example, clicking a sort header will always work correctly as long as the site adapter is maintained. Wildcard also offers clearer affordances for what types of actions are possible, or, crucially, what actions are *not* possible, which is useful to know. On the other hand, Wildcard's explicit site adapter approach will result in coverage of many fewer websites, and also a narrower range of possible actions than "sloppy" tools.

5.3 Spreadsheet-based app builders

Many others have made the powerful realization that a spreadsheet can serve as an backing data store and computation layer for an interactive web application, enabling end users to create such applications more easily. Research projects like Object Spreadsheets [21], Quilt [4], Gneiss [6], and Marmite [28], as well as commercial tools like Airtable Blocks [1] and Glide [8] allow users to view data in a spreadsheet table, compute over the data using formulas, and then connect the table to a GUI. Because many users are already familiar with using spreadsheets, this way of creating applications tends to be much easier than traditional software methods; for example, in a user study of Quilt, many users were able to create applications in under 10 minutes, even if they expected it would take them many hours.

Wildcard builds on this idea, but applies it to modifying existing applications, rather than building new applications from scratch. For many people, we suspect that tweaking existing applications provides more motivation as a starting point for programming than creating a new application from scratch.

An important design decision for tools in this space is how to deviate from traditional spreadsheets like Microsoft Excel or Google Sheets. Quilt and Glide use existing spreadsheet software as a backend, providing maximum familiarity for users, and even compatibility with existing spreadsheets. Gneiss has its own spreadsheet implementation with additional features useful for building GUIs. Marmite provides a live data view that resembles a spreadsheet, but programming is actually done using a separate data flow pane rather than spreadsheet formulas. (Marmite's approach led to some confusion in a user study, because users expected behavior more similar to spreadsheets [28].) Airtable deviates the furthest: although the user interface resembles a spreadsheet, the underlying structure is a relational database with typed columns. Wildcard's table is most similar to Airtable; the structure of a relational table is most appropriate for most data in websites, and we have not yet found a need for arbitrary untyped cells.

5.4 Web scraping / data extraction

Web scraping tools focus on extracting structured data out of unstructured web pages. Web scraping is closely related to the implementation of Wildcard, but has different end goals: web scraping generally extracts static data for processing in another environment, whereas Wildcard customizes the application's UI by maintaining a bidirectional connection between the extracted data and the page.

Web scraping tools differ in how much structure they attempt to map onto the data. Some tools like Rousillon [7] extract data in a minimally structured relational format; other tools like Piggy Bank [13] more ambitiously map the data to a rich semantic schema. In Wildcard, we chose to avoid schemas, in order to minimize the work associated with creating a site adapter.

In the future, we might be able to integrate web scraping tools to help create more reliable site adapters for Wildcard with less work, and to open up adapter creation to end users. Sifter was built on top of the Piggy Bank scraping library, suggesting precisely this type of architecture where web scraping tools are used to support interactive page modification.

6 Future work

There are still many open questions about spreadsheet-driven customization which we hope to answer through targeted development and usage of the Wildcard prototype.

The most important question is: where are the limits of this computational model? What types of useful customizations can it support or not support? While initial demos suggest a variety of use cases, we plan to develop more site adapters and demos to explore this question further. We will start with privately testing the system with our own needs, and then eventually deploy the tool publicly, once the API is stable enough and can support a critical mass of sites and use cases. We also plan to run usability studies to evaluate and improve the design of the tool.

We suspect that two areas of the current model may prove limiting. First, Wildcard's data model currently shows a single table at a time, without any notion of relationships between tables. A richer data model with foreign keys might help support certain use cases. For designing a spreadsheet interface on top of a richer relational model, we could learn from other systems with this design [2, 21].

A second potential limitation is that there is currently no mechanism for end users to express imperative workflows with sequences of actions, which related work has shown to be useful [5, 18]. It's not clear how such workflows could fit into Wildcard, since spreadsheets have a fundamentally different computation model, and site adapters created by programmers cannot easily account for every possible action a user would want to take in a page.

Another open question is how efficiently site adapters can be created for real websites, which often include complex markup. We plan to explore this question by creating adapters for sites with different data structures and in different domains, and perhaps running user tests with programmers. Possible future improvements we've

considered include developing automated heuristics to assist with the adapter creation process, and developing new abstractions that make it easier for programmers to efficiently create new robust adapters.

7 Conclusion

In this paper, we have presented *spreadsheet-driven customization*, a technique that enables end users to customize software by augmenting an application's UI with a spreadsheet. We hope that this technique contributes to making the Web into a more dynamic medium that users can mold to their own needs.

We plan to continue developing the Wildcard prototype and to eventually deploy it as an open-source tool. To receive future updates on Wildcard and notifications about a public release, sign up for the email newsletter.

We are also looking for private beta testers. If you have an idea for how you might want to use Wildcard, please get in touch. We would love to hear about your needs and help find ways to use Wildcard to solve them.

Acknowledgements Thank you to Glen Chiacchieri, Steve Krouse, Daniel Windham, and Maggie Yellen for providing helpful feedback on this work.

References

[1] *Airtable: Organize Anything You Can Imagine*. https://airtable.com.

[2] Eirik Bakke and David R. Karger. "Expressive Query Construction through Direct Manipulation of Nested Relational Results". en. In: *Proceedings of the 2016 International Conference on Management of Data - SIGMOD '16*. San Francisco, California, USA: ACM Press, 2016, pages 1377–1392. ISBN: 978-1-4503-3531-7. DOI: 10.1145/2882903.2915210.

[3] Michel Beaudouin-Lafon and Wendy E. Mackay. "Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces". In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '00. Palermo, Italy: ACM, 2000, pages 102–109. ISBN: 978-1-58113-252-6. DOI: 10.1145/345513.345267.

[4] Edward Benson, Amy X. Zhang, and David R. Karger. "Spreadsheet Driven Web Applications". en. In: *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology - UIST '14*. Honolulu, Hawaii, USA: ACM Press, 2014, pages 97–106. ISBN: 978-1-4503-3069-5. DOI: 10.1145/2642918.2647387.

[5] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. "Automation and Customization of Rendered Web Pages". en. In: *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology - UIST '05*. Seattle, WA, USA: ACM Press, 2005, page 163. ISBN: 978-1-59593-271-6. DOI: 10.1145/1095034.1095062.

[6] Kerry Shih-Ping Chang and Brad A. Myers. "Creating Interactive Web Data Applications with Spreadsheets". en. In: *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology - UIST '14*. Honolulu, Hawaii, USA: ACM Press, 2014, pages 87–96. ISBN: 978-1-4503-3069-5. DOI: 10.1145/2642918.2647371.

[7] Sarah E. Chasins, Maria Mueller, and Rastislav Bodik. "Rousillon: Scraping Distributed Hierarchical Web Data". en. In: *The 31st Annual ACM Symposium on User Interface Software and Technology - UIST '18*. Berlin, Germany: ACM Press, 2018, pages 963–975. ISBN: 978-1-4503-5948-1. DOI: 10.1145/3242587.3242661.

[8] *Create an App from a Google Sheet in Minutes · Glide*. en. https://www.glideapps.com/.

[9] A. A diSessa and H. Abelson. "Boxer: A Reconstructible Computational Medium". en. In: *Communications of the ACM* 29.9 (Sept. 1986), pages 859–868. ISSN: 00010782. DOI: 10.1145/6592.6595.

[10] Andrea A. diSessa. *Changing Minds: Computers, Learning, and Literacy*. Cambridge, MA, USA: MIT Press, 2000.

[11] Andrew Hogue and David Karger. "Thresher: Automating the Unwrapping of Semantic Content from the World Wide Web". en. In: *Proceedings of the 14th International Conference on World Wide Web - WWW '05*. Chiba, Japan: ACM Press, 2005, page 86. ISBN: 978-1-59593-046-0. DOI: 10.1145/1060745.1060762.

[12] David F. Huynh, Robert C. Miller, and David R. Karger. "Enabling Web Browsers to Augment Web Sites' Filtering and Sorting Functionalities". en. In: *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology - UIST '06*. Montreux, Switzerland: ACM Press, 2006, page 125. ISBN: 978-1-59593-313-3. DOI: 10.1145/1166253.1166274.

[13] David Huynh, Stefano Mazzocchi, and David Karger. "Piggy Bank: Experience the Semantic Web Inside Your Web Browser". en. In: *The Semantic Web – ISWC 2005*. Edited by Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pages 413–430. ISBN: 978-3-540-32082-1. DOI: 10.1007/11574620_31.

[14] Hypercard. "HyperCard". en. In: *Wikipedia* (Dec. 2019). Page Version ID: 931376685.

[15] Ink and Switch. *End-User Programming*. en-US. Mar. 2019.

[16] A. Kay and A. Goldberg. "Personal Dynamic Media". In: *Computer* 10.3 (Mar. 1977), pages 31–41. ISSN: 1558-0814. DOI: 10.1109/C-M.1977.217672.

[17] Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. "Webstrates: Shareable Dynamic Media". en. In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15*. Daegu, Kyungpook, Republic of Korea: ACM Press, 2015, pages 280–290. ISBN: 978-1-4503-3779-3. DOI: 10.1145/2807442.2807446.

[18] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. "CoScripter: Automating & Sharing How-to Knowledge in the Enterprise". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: ACM, 2008, pages 1719–1728. ISBN: 978-1-60558-011-1. DOI: 10.1145/1357054.1357323.

[19] Greg Little, Robert C. Miller, Victoria H. Chou, Michael Bernstein, Tessa Lau, and Allen Cypher. "Sloppy Programming". en. In: *No Code Required*. Elsevier, 2010, pages 289–307. ISBN: 978-0-12-38154I-5. DOI: 10.1016/B978-0-12-381541-5.00015-8.

[20] Wendy E. Mackay. "Triggers and Barriers to Customizing Software". en. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Reaching through Technology - CHI '91*. New Orleans, Louisiana, United States: ACM Press, 1991, pages 153–160. ISBN: 978-0-89791-383-6. DOI: 10.1145/108844.108867.

[21] Matt McCutchen, Shachar Itzhaky, and Daniel Jackson. "Object Spreadsheets: A New Computational Model for End-User Development of Data-Centric Web Applications". en. In: *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software - Onward! 2016*. Amsterdam, Netherlands: ACM Press, 2016, pages 112–127. ISBN: 978-1-4503-4076-2. DOI: 10.1145/2986012.2986018.

[22] Bonnie A. Nardi and James R. Miller. "An Ethnographic Study of Distributed Problem Solving in Spreadsheet Development". In: ACM Press, 1990, pages 197–208.

[23] Bonnie A. Nardi and James R. Miller. "Twinkling Lights and Nested Loops: Distributed Problem Solving and Spreadsheet Development". en. In: *International Journal of Man-Machine Studies* 34.2 (Feb. 1991), pages 161–184. ISSN: 00207373. DOI: 10.1016/0020-7373(91)90040-E.

[24] Mitchel Resnick. *Designing for Wide Walls*. en. Aug. 2016.

[25] Streak. *InboxSDK*. https://www.inboxsdk.com/.

[26] Kartik Talwar. *Gmail.Js*. en. https://github.com/KartikTalwar/gmail.js. 2019.

[27] Bret Victor. *Dynamicland*. https://dynamicland.org/.

[28] Jeffrey Wong and Jason I. Hong. "Making Mashups with Marmite: Towards End-User Programming for the Web". en. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '07*. San Jose, California, USA: ACM Press, 2007, pages 1435–1444. ISBN: 978-1-59593-593-9. DOI: 10.1145/1240624.1240842.

About the authors

Geoffrey Litt is a PhD student at MIT, researching programming tools for end users and developers.

Daniel Jackson is a professor in the Department of Electrical Engineering and Computer Science at MIT, associate director of CSAIL, and a MacVicar Fellow. He leads the Software Design Group.