User-Generated Variables

Streamlined Interaction Design for Feature Requests and Implementations

Jun Kato
National Institute of Advanced Industrial Science and Technology (AIST)
Tsukuba, Ibaraki, Japan
jun.kato@aist.go.jp

Masataka Goto
National Institute of Advanced Industrial Science and Technology (AIST)
Tsukuba, Ibaraki, Japan
m.goto@aist.go.jp

ABSTRACT

Programmers write source code that compiles to programs, and users execute the programs to benefit from their features. While issue-tracking systems help communication between these two groups of people, feature requests have usually been written in text with optional figures that follows community guidelines and needs human interpretation to understand what to implement in which part of the source code. To make this process more direct, intuitive, and efficient, a streamlined interaction design called "User-Generated Variables (UGV)" is proposed. First, the users can declare parameters that they want to tweak in existing programs without reading or understanding the source code. Then, the system turns the proposal into variable declarations in the relevant part of the source code. Finally, the programmers are notified of the proposal and can implement the actual features to reflect changes in the variable value. The proposed interaction is implemented in two existing Web-based Integrated Development Environments, and its user experience is briefly tested with eight users and programmers. Its technical requirements, limitations, and potentials are discussed. The content of this paper with live examples is available at http://junkato.jp/ugv.

CCS CONCEPTS

• **Human-centered computing** → **Web-based interaction**; **User interface programming**; *Interface design prototyping*; • **Software and its engineering** → **Integrated and visual development environments**; *Collaboration in software development*; *Graphical user interface languages*;

KEYWORDS

User-generated content (UGC), live programming, integrated development environment, programming experience

ACM Reference format:
Jun Kato and Masataka Goto. 2017. User-Generated Variables. In *Proceedings of Programming Experience Workshop, Brussels, Belgium, April 4, 2017 (PX/17)*, 7 pages.
DOI: 10.1145/3079368.3079403

1 INTRODUCTION

Programs have increasingly been dealing with more data as in the sensor, robot, and animation applications, and programming is becoming an even more involving activity that requires not only writing code but also managing data and tuning parameter values [20]. More contribution from users is required since it not only helps programmers but also allows the users to customize programs that then fully match their needs. For such successful collaborations, the program development should be considered as an open-ended process and needs tool support that create the technical and social conditions for broad participation in the process which are as important as support for writing code [10].

The open-ended development process has been realized in such systems that allow programmers to create templates and users to fill in the blanks. Example systems include IFTTT [14], Microsoft on{X} [26], and Web-based Integrated Development Environments (WIDE) for interactive content such as TextAlive [21] and f3.js [19]. In these Web-based systems, User-Generated Content (UGC) reflects the design space which is defined by the templates and is difficult to be explored solely by programmers. Meanwhile, this development process is uni-directional in that the users cannot provide feedback on whether the templates are well-designed or not. To provide such feedback, the users need to rely on external

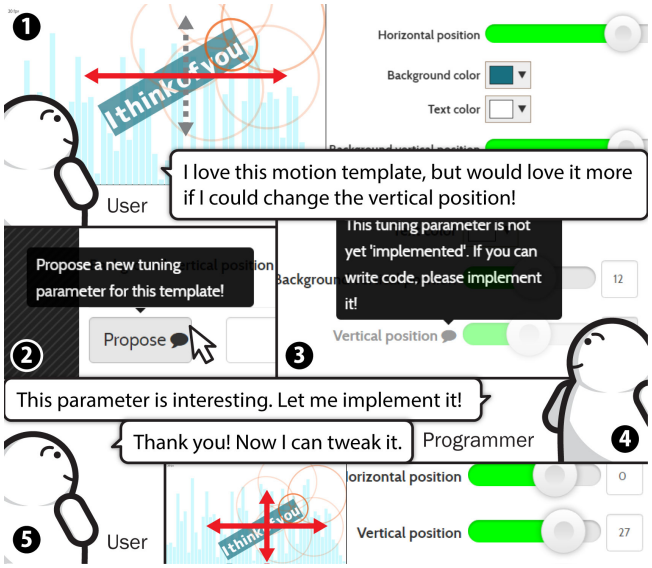

Figure 1: Overview of User-Generated Variables (UGV).

communication tools such as emails or issue tracking systems and write free-form text with optional figures.

This paper presents an extension to IDEs called "User-Generated Variables (UGV)" to make the feedback process more direct, intuitive, and efficient. It streamlines feature requests from users and implementations by programmers as shown in Figure 1. It provides a graphical user interface (GUI) that allows the users to propose a new "blank" variable to be filled out in the "template," which instantly appears as a user-interface mockup without the actual feature implementation. The authors of the programs are notified of this proposal, and when they navigate to the source code, there is a new code snippet that declares the variable and displays the mockup. Then, the programmers can reject the proposal by removing the variable declaration or implement the features and notify users of that accomplishment.

The rest of this paper discusses the design goal, introduces relevant work, details the example implementations, and reports preliminary feedback from a group of four programmers and four users who used the UGV features for three days, followed by the discussion on the limitations and potentials of UGV.

2 USER-GENERATED VARIABLES

To clarify the scope of this paper, this section describes the design goal and explains the interaction design of the newly-introduced "User-Generated Variables (UGV)."

2.1 Design Goal

The aim of UGV is to streamline the process of feature requests and implementations through sharing a programming concept between users and programmers. User involvement during program development life cycle has been studied well and revealed to have a positive impact, but are challenging as well [1]. We designed UGV so that the challenges can be addressed as shown in Table 1.

Challenges	Solutions
Power asymmetry between programmers and users	UGV provides users partial ownership on the source code by allowing them to declare variables.
Efforts required by users and lack of the motivation	UGV embeds interactive user interfaces for sending feature requests efficiently without leaving the programs.
Lack of users' expertise and communication skills in the development	UGV does not assume user training such as reading rule documents for feature requests. (e.g. `CONTRIBUTIONS.md`)
Misunderstandings (users)	Showing UI mockup makes the goal clear and understood intuitively by stakeholders.
Misunderstandings (programmers)	There is no need to parse free-form text or figures; the request is directly translated to variable declarations in the code.

Table 1: Challenges of user involvement in software development and UGV features to address them.

While end-user programming, in general, provides the full-stack programming experience to users, this paper only deals with variables. While there are various roles in variables [9], UGV are those that store user preferences and are exposed to users through GUI so that they can tweak the values and customize the program behaviors. Live Tuning [18] is an interaction design that eases defining such variables on the programmers' side, and UGV enables their generation by users. The variables are usually declared by programmers by writing a code snippet (e.g. `var motionSpeed = 100`), which requires a prior knowledge on programming. In contrast, UGV can be declared with the user interfaces embedded in the program without expertise on programming.

As the name suggests, UGV are generated by users and implemented by programmers. UGV are useful for enabling end-user customizations and are useless if the programs do not take any user input. In particular, we consider UGV useful for creativity support tools such as software for composing music, graphics, videos, and programs.

2.2 Interaction Design

UGV-enabled GUI applications show a "propose" button in their user interface that allows users to propose new variables for tuning program behaviors (Figure 1 ②). Clicking the button displays a list of available variable types (Figure 2). Once the required information is provided, a GUI widget to tune the behaviors appears in the

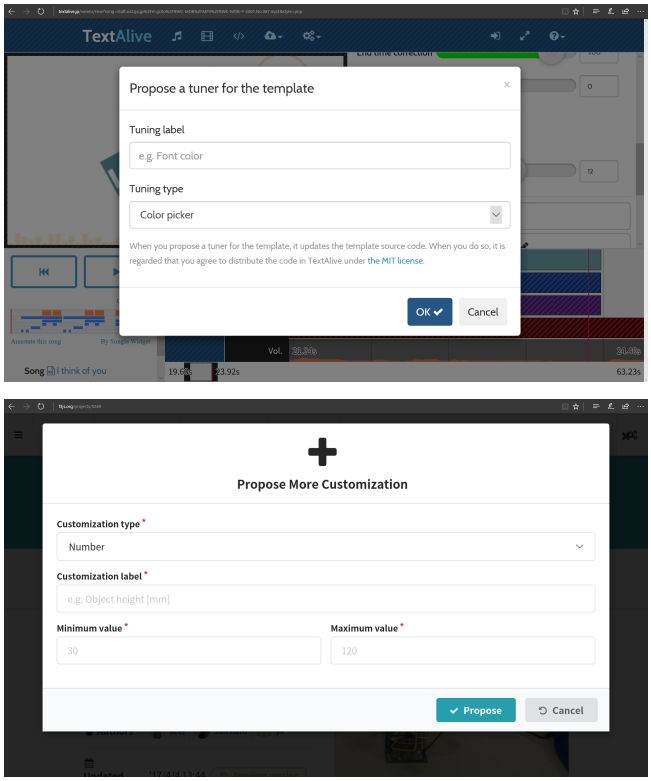

Figure 2: TextAlive [21] and f3.js [19] "Propose" dialogs for generating a User-Generated Variable.

applications as a mockup user interface (Figure 1 ③). We expected that showing the mockup user interface rewards the users with increased feeling of their ownership of the programs.

The system runs static analysis on the source code of the application behind the scenes (detailed in Section 4) and searches for an appropriate location to declare the variables and insert a code snippet to display the mockup user interface. This insertion is a semi-automatic process in that programmers can optionally specify preferred locations for variable declarations in the source code by adding special comments to the locations. At this moment, there is no actual implementation that affects the program behavior.

The programmers are then notified of the variable declaration through social networking services such as Twitter and GitHub (Figure 1 ④). When the programmers navigate to the source code, they can see the declaration and the mockup code and choose whether to reject or accept the proposal. To reject it, they simply remove the declaration and relevant code snippet. To accept it, they start the implementation of the GUI widget that changes the program behavior in response to the parameter tuning events firing from the widget. After the implementation, users of the program can benefit from the parameter tuning widget (Figure 1 ⑤).

3 RELATED WORK

This section introduces prior work on enabling collaborations between programmers, collecting user feedbacks, and leveraging the users' capabilities for program specifications.

3.1 Social Coding

There has been much work on enhancing the collaborations between programmers. Prior work on direct and asynchronous collaborations between them includes social coding platforms such as GitHub [11], which has been studied well in the mining software repositories (MSR) field [16]. It has also been studied from the perspective of computer-supported cooperative work (CSCW); the user activity information shared in such platforms is found to help the users coordinate their work, advance technical skills, and manage their reputations [7].

While such systems assume asynchronous communication, there has been proposed support for real-time collaborations, such as that for pair-programming (e.g. Collabode [12] and Codeshare [27]), and live coding environments for collaborative art performances [29]. There is also a prior study that suggests on-demand expert support to be integrated into the IDEs [6].

Prior work on indirect collaborations (without the need for direct communications) include enhancements to runtime error messages by sharing explanations of their typical workarounds [13]. Code-completion recommendations can be enhanced as well by collecting the usage statistics of APIs [3]. UGV falls into this indirect and asynchronous support for collaborative code edits but differs from prior work since it allows end-users to get involved in the process without requiring any prior knowledge.

3.2 User Feedback

Most of the social coding features are used by programmers, but bug and issue reports and feature requests can be made by users. These are usually text-based and occasionally accompanied by images, videos, and other files that provide context information, which helps programmers to understand issues. However, users first need to learn how to submit such reports, and the resulting reports often lack context information. An emerging source of user feedback is the app reviews in app stores and the feature request management systems such as UserVoice [32], both of which require no user training and allow freeform text input, making it even more challenging to understand issues.

Automatic processing techniques of these raw feedbacks have been proposed to help programmers. For instance, mining the code repositories infers relevant programmers [5] and impacted source code files [15]. The app reviews can be automatically classified into bug reports, feature requests, and others [24].

Another way (and the way UGV takes) to improve the user involvement is to provide tool support to make user feedbacks more informative and structured. Prior efforts include tools that inject GUIs into existing Web-based applications to enable the issues to be easily reported with detailed context information such as client event logs [31]. While our research was aimed at similar goals to the prior tools, ours has a more refined goal of allowing users to propose new variables to customize program behaviors. As a result, UGV can provide programmers more specific tasks of whether to remove the variable declarations or implement the parameter tweaking feature, which is unlike the open-ended goals in the issue-reporting tools.

3.3 End-user Programming and Customization

Prior work has investigated programming environments that can be used without substantial knowledge of programming to allow users to modify program behaviors (end-user programming) or has provided domain-specific tools to enable behaviors to be customized (end-user customization).

While most of the work in these categories has focused on empowering individual novices to benefit from creating or customizing programs, our work aims at collecting feedbacks from the novices and improving the global "template" programs shared among the users. This fundamental difference results in the varied levels of required expertise to use these systems.

If users want to add a new feature (as in our goal of adding a variable for parameter tuning) in an end-user programming environment, they need to read the source code and understand program behaviors. Some GUI frameworks for live programming such as "Halo" of the Morphic framework [25] and "It's Alive!" feature of the TouchDevelop [4] can reduce the required effort by allowing the GUI elements to be selected and edited without terminating the program. However, appropriately editing the code without breaking existing features still requires program understanding.

Macros and scripting engines can prevent from destroying the core features of the programs and yet allowing them to be extensively customized. Example systems include VisualBasic for Application (VBA) scripts for Microsoft Office applications and Chickenfoot for Web browsers [2].

In contrast to these systems, our interaction design requires no user training and targets systems that allow end-users to "fill-in-the-blank templates" as was explained in Section 1. Please note that

ordinary GUI applications also provide interactive GUI widgets to customize program behaviors. What differentiates tools in the "fill-in-the-blank template" paradigm from these standard applications is that the tools are editing source code elements that define program behaviors, rather than data entries in the configuration files. The most prominent example is the Live Tuning interaction [18] that allows end-users to edit variable values defined in the source code. UGV improves on this work and allows the end-users to propose new variables.

4 IMPLEMENTATION

UGV can be implemented on top of any IDEs in GUI applications connected to the Internet. We implemented UGV in two existing Web-based IDEs, i.e., TextAlive [21] and f3.js [19].

4.1 Example Systems

TextAlive [21] is available at http://textalive.jp and allows users to create videos with music by composing multiple visual effects, each of which is called a "template." A template has multiple parameters that define the details of the effects, by which the users create visual effects in their styles. They can propose new parameters to tweak the visual effects by introducing UGV. For instance, a user can propose a new variable "stroke width" to the visual effect of dancing lines.

f3.js [19] is available at http://f3js.org and allows users to create physical computing devices by writing code that specifies both their behaviors and enclosure layout. It enables parametric generation of the entire device design, allowing the users to customize the devices with GUI widgets. They can propose new options for customization by introducing UGV. For instance, a user can propose a new option as to whether to add a flash LED and feature to enlighten it to a camera device or not.

4.2 GUI Toolkits for User-Generated Widgets

Compared to the ordinary GUI toolkits, UGV implementation automatically appends the "propose" button to the container instances that host widgets for parameter tuning. In addition, the containers show mockup user interfaces generated by users along with the widgets with the actual implementations.

The example systems supported Live Tuning [18] that allows programmers to easily expose GUI widgets to end-users for parameter tuning, which are bound to the variable declarations in the source code. This support eases the implementation of UGV – instead of adding a full-specification code snippet that creates the GUI widget in the source code (e.g., `var cbox = new Checkbox(); panel.add(cbox); cbox.addEventListener(···)`), our system parses the code and inserts a special comment next to the variable declaration (`// @ui Check()`) that is handled by the IDE to populate the widget. For more details on this static analysis process, please refer to the online version of this paper. We extended the special comment format of the systems to check if the comment was inserted as part of the proposal or not. If the comment was part of the proposal (`// @ui @proposed Check()`), the widgets were disabled so that the users could not edit the values. In addition, the text encouraging implementation was displayed next to the mockup widgets.

Please note that there is a trade-off between the flexibility of layout and the ease of widget generations. While the full-spec code snippet could potentially allow the user to position the mockup widgets freely, our current comment format is not capable of precise layout control and shows the list of widgets in the order of the comments.

Table 2 shows variable types and widgets that are supported in the example systems. The last entry (`Undefined`) is provided to allow feature requests that do not fit in any types.

Variable type	GUI widget
`Number`	Slider with the specified range
`String`	Text box with the specified default value
`Boolean`	Checkbox with the specified default value
`Color`	Color palette with the specified default value
`Undefined`	Free-form text message to programmers

Table 2: Types of currently supported UGV.

4.3 Automatic Variable Declarations

The user proposal is immediately passed to the Web server when it is made. The server finds the source code that generates the corresponding user interface container and runs static code analysis to locate appropriate positions where the new variable should be declared, and the GUI widget should be initialized.

In general, the source code of GUI applications could have separate definitions for the model and view and the localization process could be challenging. In contrast, the example systems supported Live Tuning, and each UGV (a variable declaration and GUI widget initialization) could be written in one place.

TextAlive defines a visual effect template as a JavaScript class and each variable for parameter tuning as its property. A new UGV is inserted immediately after the last property declaration. F3.js defines each project as a Node.js JavaScript project, and each variable for parameter tuning is a global variable in the project source code. A new UGV is inserted at the end of the code.

The variable name is randomly generated until it does not conflict with any other variables. The location to which a new UGV is inserted can be changed by manually adding a special comment (`// @variables`) to the preferred location.

4.4 Message Notifications

Unlike conventional IDEs, UGV implementation requires the IDE to have social features that foster communication between programmers and users. At the least, the IDEs should identify each user and be able to send message notifications to them.

Both of the example systems allowed users to log in with a Twitter (social networking service) account. Any actions made during the UGV process were notified through tweets from our accounts (@TextAliveJpn and @F3JsOrgn where the last "n" stands for notifications) that were associated with the systems.

5 PRELIMINARY USER FEEDBACK

This section briefly reports early user feedback collected from an in-house user study.

5.1 Study Setup

To collect preliminary user feedback of the UGV extension to the example systems, we asked four users to use the UGV features in the systems. We also asked four programmers to respond to the proposals.

Since the proposals and implementations (or rejections) took place asynchronously, we did not specify specific working hours but asked them to find some time during a period of three days.

5.2 Results

Within TextAlive, seven variables were declared and five were implemented as requested. The implemented variables include the width of line strokes, rotation of moving boxes, whether to zoom in or out, and background color and transparency of graphic objects. One proposal of adding rotation effect to each of the child graphic objects (e.g. character objects that belong to a parent word object) was rejected because such visual effect could be already achieved by assigning rotation templates to the child objects. The remaining one proposal was implemented but transformed into a different set of UGVs. The original proposal was made to a template named "Rainbow" that change colors of character objects randomly from time to time. It aimed to enable choosing color tones from options such as pop or cool rather than the completely random colors. In response to the proposal, the programmer did not add such options but seven color palettes so that the users will be able to define a set of colors to be randomly assigned to the character objects.

Within f3.js, seven variables were declared and six were implemented. The implemented variables include a number of buzzers in a musical instrument device, whether to eliminate one plane of a box shape or not (so that the user can use the box as an accessory case which is impossible with all the six planes), volume of the audio effect, a message string that congratulates game clears, and a number of players for a game device. The rejected one was a Boolean variable that switches whether to enable voiceover feature or not for a translator device that recognizes text with its camera. The proposal was rejected because the device was not equipped with a speaker module at the time and it seemed to take much time to implement it.

In summary, the users generated fourteen variables, and the programmers implemented eleven of them. We observed that the rest three variables were rejected by the varied reasons as follows:

- The desired feature could be already achieved without the proposed variable.
- The desired feature could be implemented more effectively with a different set of variables.
- The desired feature was considered too difficult to implement.

After the three days, we conducted informal interviews with the participants. Programmers welcomed the UGV's simple workflow, especially that *"the variable to be implemented was already there and what they had to do was crystal clear."* The fact that the users were encouraged to send a limited kind of requests made the users feel relaxed as one of them stated: *"I did not need to worry if I was making nonsense proposals."* It also accelerated the understanding of general programming as another stated: *"I could learn types of parameters that could be defined in the programs."* This implies the potential of UGV-enabled systems to motivate people to learn programming.

6 DISCUSSION AND FUTURE WORK

Given our experience of building two example systems and the results of collecting the preliminary user feedback, limitations and potentials of the proposed interaction design is discussed.

6.1 Limitations

6.1.1 Lack of Scalability. UGV in the current implementation can be declared but cannot be removed by the users. If every user requests new features, mockup elements could take up space quite easily. Some programmers and users suggested the feature to vote up and down variables (either mockups or those with actual implementations.) Future work should implement such features and investigate how to help consensus building among such requests.

6.1.2 Limited Widget Layout. The layout of the GUI widgets generated by the users is not flexible at all. As discussed in Subsection 4.2, there is a trade-off between the flexibility and the ease of proposal feature. Furthermore, fixing the layout could be beneficial since it could have special meanings such as the first UI mockup in the list denoting the proposal with top votes. Future work could investigate these multiple possibilities.

6.1.3 Limited Kinds of Variables/Widgets. A type of a variable is currently limited to fixed sets as shown in Table 2. Future work will add more types of variables. The most interesting type seems to be functions, which will be discussed in detail in the next subsubsubsection. It would also be interesting to allow users to propose a new type of variables, which require programmers to implement new GUI widgets.

6.2 Source Code as a Shared Medium

The role of source code is what differentiates this work from prior efforts on encouraging user involvement in software development. Both users and programmers directly edit it through dialogs and code editors, respectively. The user feedback is embedded in the text-based source code, not in another database such as an issue tracking system.

There has been prior work that makes text-based code more comprehensible for non-programmers. Sikuli [33] and Picode [22] show graphical data as inline images in the source code editor. Picode user study [17] reports that inline photos of human and robots representing their posture data allowed elementary school students without prior programming experience to comprehend what the code surrounding the photos is doing. Furthermore, they could customize the program behavior by replacing the photos.

General text-based code requires prior knowledge of programming to comprehend and edit. Appropriate representations of the source code, however, could make it more accessible for a broader range of people. We foresee much potential in this approach and expect that such effort would lead to better understanding of software development process and adequate acknowledgment to programmers by the general public.

6.3 User-Generated Content vs. Variables

User-Generated Content (UGC) usually denotes content created by users (not professional creators). In the presented systems, the users can author videos (TextAlive) and customized physical computing devices (f3.js), which are examples of UGC.

While UGC is similar to the proposed UGV in that they both enable user involvement in a creative process, UGC is a concrete artifact created with creativity support tools, and UGV expands ways of expressing the creativity by adding new parameters – in other words, UGV improves the tools. UGC is authored by the users themselves, but UGV delegates its actual implementation tasks to programmers and has a collaborative aspect on its own.

Aligned with the discussion in the meta-design (= designing the design process) [10], we consider that tool support for collaborations enables continuous co-development of programs, making all of the participants in the design process express themselves with their personal motivations. During the process, we expect that users will have chances to become programmers – it was a good sign that a user got interested in programming concepts by using the UGV features during the study.

Future work should investigate more variety of variable types. The current implementation allows to add GUI widgets that are suitable for tweaking constant values, and one interesting addition is the "function" type – a simple extension would allow declaring function objects that generate buttons.

6.4 Layered User Interfaces for Diverse People

Visual programming languages for dataflow programming such as PureData [8] might allow end-users to add new nodes. However, the user interface as a whole requires prior knowledge of programming and is not very accessible compared to our research where the end-user applications had dedicated GUIs to propose a new variable. We considered this separation of user interfaces for programming and the use of applications to be important for engaging more end-users. There is a prior work on providing multiple layers of user interfaces for people with different levels of expertise [30], and ours is just another example. The difference is that our layers are designed for collaborations between people while the original idea was for people growing their expertise and switching layers by themselves.

Please note that separation was meant to be on the user interface level and not on the implementation level. The example systems eliminate the gap between programming and runtime environments on the implementation level, as was introduced in the Live Tuning paper [18]. These are unified environments with two different user interfaces for programmers and end-users.

6.5 IDEs that Make Tweets

Some programmers reported that they liked that the IDEs have their Twitter accounts and make tweets to notify them. They seemed to feel less pressure compared to being directly communicated by the users.

Human-Agent Interaction is the most relevant research field, and related projects include Sharedo [23] that provides todo management interfaces shared between software agents and human users. For each todo entry, Sharedo provides a discussion board that can be used as a communication medium between the software agents and users. For instance, an agent asks for clarifying the todo details. In our case, the IDE agent could mediate communications between programmers and users. In our future work, a link to such discussion board and buttons to vote up and down could be added next to each variable. Once IDEs have their representative agents, a potential application domain would be a tutoring system [28].

7 CONCLUSION

We proposed a novel interaction design called "User-Generated Variables (UGV)" that allows program users to propose a new variable for tuning the program behavior. It is inserted as a new variable declaration in the source code, helping programmers to implement the feature. While the current UGV implementation has a certain limitation such as lack of scalability, it has potential to foster co-development of programs by programmers and users. Please refer to http://junkato.jp/ugv for the live demonstration of UGV and up-to-date content.

ACKNOWLEDGMENTS

This work was supported in part by JST CREST and ACCEL.

REFERENCES

[1] Muneera Bano and Didar Zowghi. 2015. A systematic review on the relationship between user involvement and system success. *Information and Software Technology* 58 (2015), 148 – 169. DOI: https://doi.org/10.1016/j.infsof.2014.06.011

[2] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. 2005. Automation and Customization of Rendered Web Pages. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST '05)*. ACM, New York, NY, USA, 163–172. DOI: https://doi.org/10.1145/1095034.1095062

[3] Marcel Bruch, Eric Bodden, Martin Monperrus, and Mira Mezini. 2010. IDE 2.0: Collective Intelligence in Software Development. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research (FoSER '10)*. ACM, New York, NY, USA, 53–58. DOI: https://doi.org/10.1145/1882362.1882374

[4] Sebastian Burckhardt, Manuel Fahndrich, Peli de Halleux, Sean McDirmid, Michal Moskal, Nikolai Tillmann, and Jun Kato. 2013. It's Alive! Continuous Feedback in UI Programming. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '13)*. ACM, New York, NY, USA, 95–104. DOI: https://doi.org/10.1145/2491956.2462170

[5] Gerardo Canfora and Luigi Cerulo. 2006. Supporting Change Request Assignment in Open Source Development. In *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC '06)*. ACM, New York, NY, USA, 1767–1772. DOI: https://doi.org/10.1145/1141277.1141693

[6] Yan Chen, Steve Oney, and Walter S. Lasecki. 2016. Towards Providing On-Demand Expert Support for Software Developers. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 3192–3203. DOI: https://doi.org/10.1145/2858036.2858512

[7] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW '12)*. ACM, New York, NY, USA, 1277–1286. DOI: https://doi.org/10.1145/2145204.2145396

[8] Pure Data. 2017. Pure Data - Pd Community Site. https://puredata.info/. (2017). Accessed April 1, 2017.

[9] Kelly Blincoe Leif Singer Daniel M. German Eirini Kalliamvakou, Georgios Gousios and Daniela Damian. 2006. Roles of variables in three programming paradigms. *Computer Science Education* 16, 4 (2006), 261–279. DOI: https://doi.org/10.1080/08993400600874584 arXiv:http://dx.doi.org/10.1080/08993400600874584

[10] Elisa Giaccardi and Gerhard Fischer. 2008. Creativity and evolution: a metadesign perspective. *Digital Creativity* 19, 1 (2008), 19–32. DOI: https://doi.org/10.1080/14626260701847456 arXiv:http://dx.doi.org/10.1080/14626260701847456

[11] GitHub. 2017. GitHub. https://github.com/. (2017). Accessed April 1, 2017.

[12] Max Goldman, Greg Little, and Robert C. Miller. 2011. Collabode: Collaborative Coding in the Browser. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '11)*. ACM, New York, NY, USA, 65–68. DOI: https://doi.org/10.1145/1984642.1984658

[13] Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R. Klemmer. 2010. What Would Other Programmers Do: Suggesting Solutions to Error Messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*

(CHI '10). ACM, New York, NY, USA, 1019–1028. DOI: https://doi.org/10.1145/1753326.1753478

[14] IFTTT. 2017. Learn how IFTTT works. http://ifttt.com/. (2017). Accessed April 1, 2017.

[15] Huzefa Kagdi and Denys Poshyvanyk. Who can help me with this change request?. In *Proceedings of IEEE 17th International Conference on Program Comprehension* (2009-05) *(IEEE ICPC)*. 273–277. DOI: https://doi.org/10.1109/ICPC.2009.5090056

[16] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The Promises and Perils of Mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*. ACM, New York, NY, USA, 92–101. DOI: https://doi.org/10.1145/2597073.2597074

[17] Jun Kato. 2013. *Integrated Graphical Representations for Development of Programs with Real-world Input and Output*. Ph.D. Dissertation. The University of Tokyo.

[18] Jun Kato and Masataka Goto. 2016. Live Tuning: Expanding Live Programming Benefits to Non-Programmers. In *Proceedings of the Second Workshop on Live Programming Systems (ECOOP LIVE '16)*. 6.

[19] Jun Kato and Masataka Goto. 2017. f3.js: A Parametric Design Tool of Physical Computing Devices for Both Interaction Designers and End-users (to appear). In *Proceedings of the 2017 ACM Conference on Designing Interactive Systems (DIS '17)*. ACM, New York, NY, USA, 10. DOI: https://doi.org/10.1145/3064663.3064681

[20] Jun Kato, Takeo Igarashi, and Masataka Goto. 2016. Programming with Examples to Develop Data-Intensive User Interfaces. *Computer* 49, 7 (Jul 2016), 34–42. DOI: https://doi.org/10.1109/MC.2016.217

[21] Jun Kato, Tomoyasu Nakano, and Masataka Goto. 2015. TextAlive: Integrated Design Environment for Kinetic Typography. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 3403–3412. DOI: https://doi.org/10.1145/2702123.2702140

[22] Jun Kato, Daisuke Sakamoto, and Takeo Igarashi. 2013. Picode: Inline Photos Representing Posture Data in Source Code. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI '13)*. 3097–3100. DOI: https://doi.org/10.1145/2470654.2466422

[23] Jun Kato, Daisuke Sakamoto, Takeo Igarashi, and Masataka Goto. 2014. Sharedo: To-do List Interface for Human-agent Task Sharing. In *Proceedings of the Second International Conference on Human-agent Interaction (HAI '14)*. ACM, New York, NY, USA, 345–351. DOI: https://doi.org/10.1145/2658861.2658894

[24] Walid Maalej and Hadeer Nabil. 2015. Bug report, feature request, or simply praise? On automatically classifying app reviews. In *Proceedings of IEEE 23rd International Requirements Engineering Conference*. 116–125. DOI: https://doi.org/10.1109/RE.2015.7320414

[25] John H. Maloney and Randall B. Smith. 1995. Directness and Liveness in the Morphic User Interface Construction Environment. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology (UIST '95)*. ACM, New York, NY, USA, 21–28. DOI: https://doi.org/10.1145/215585.215636

[26] Microsoft. 2012. on{X}. https://www.youtube.com/watch?v=qfLMTsIJsoo. (2012). Accessed April 1, 2017.

[27] Lee Munroe and Tejesh Mehta. 2017. Codeshare - Share code in real-time in your browser. http://codeshare.io/. (2017). Accessed April 1, 2017.

[28] Yoshiki Ohshima, Alessandro Warth, Bert Freudenberg, Aran Lunzer, and Alan Kay. 2016. Towards Making a Computer Tutor for Children of All Ages: A Memo. In *Proceedings of the Programming Experience 2016 (PX/16) Workshop (PX/16)*. ACM, New York, NY, USA, 21–25. DOI: https://doi.org/10.1145/2984380.2984383

[29] Charlie Roberts, Karl Yerkes, Danny Bazo, Matthew Wright, and JoAnn Kuchera-Morin. 2015. Sharing Time and Code in a Browser-Based Live Coding Environment. In *Proceedings of the First International Conference on Live Coding*. ICSRiM, University of Leeds, Leeds, UK, 179–185. DOI: https://doi.org/10.5281/zenodo.19351

[30] Ben Shneiderman. 2003. Promoting Universal Usability with Multi-layer Interface Design. In *Proceedings of the 2003 Conference on Universal Usability (CUU '03)*. ACM, New York, NY, USA, 1–8. DOI: https://doi.org/10.1145/957205.957206

[31] Macropod Software. 2017. The simplest bug tracker and issue tracker | BugHerd. https://bugherd.com/. (2017). Accessed April 1, 2017.

[32] UserVoice. 2017. Roadmap Prioritization from Product Feedback | UserVoice. https://www.uservoice.com/. (2017). Accessed April 1, 2017.

[33] Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. 2009. Sikuli: Using GUI Screenshots for Search and Automation. In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology (UIST '09)*. ACM, New York, NY, USA, 183–192. DOI: https://doi.org/10.1145/1622176.1622213