

Usability of a Domain-Specific Language for a Gesture-Driven IDE

Michaela Bačíková, Martin Maričák, Matej Vančík

Technical university of Košice

Letná 9, 042-00 Košice, Slovakia

Email: michaela.bacikova@tuke.sk, {martin.maricak, matej.vancik}@student.tuke.sk

Abstract—User interfaces (UIs) are advancing in every direction. The usage of touch screen devices and adaptation their UIs lives its boom. However integrated development environments (IDEs) that are used to develop the same UIs are oversleeping the time. They are directed to developing usable software, but forgot to be usable by themselves. Our goal is to design a new way of user interaction for common IDEs with the help of touch. The target group are hybrid devices formed by a physical keyboard and either an integrated, or separate, touch screen display. In this paper we describe a set of general purpose and domain-specific gestures which represents a language for working with a touch-driven IDE and provide a method their design. We performed two studies with developers from industry and university and developed a prototype of a gesture-driven IDE to evaluate the usability of the presented approach.

I. INTRODUCTION

The accessibility of alternative devices such as wearable gadgets, smartphones, tablets, tablet PCs, wall-sized displays, touch-enabled displays, kinect and others, has given a rise to different design guidelines for application user interfaces (UIs) which, though still young and imperfect, become more and more dominant. According to the world statistics a large part of interaction has moved to touch devices and the vast majority of mobile devices now supports touch control. Touch interfaces are natural and easy to get used to and they also affect the way people try to control other devices such as common PCs or notebooks. Therefore, *hybrid devices*¹ have emerged and operation systems try to adapt their UIs and functionality to touch.

Although touch control support has moved a large leap forward, it still has its disadvantages. Touch gestures are not entirely uniform on all platforms and in many cases, a touch display alone is not sufficient to fully handle all desired functionalities. A typical example is *writing on a virtual keyboard*. Writing on a virtual keyboard is slow, imprecise and the keyboard occupies a large part of the screen [1]. Still, the situation is much better than before and the rise of platform guidelines for touch devices with different screen sizes helps the situation.

Despite the disadvantages, the situation is much better than in case of integrated development environments (IDEs). They are made to improve developer productivity as much as

possible [2] but it is as if developers were primarily focusing on making UIs and interaction better just for their users while forgetting about themselves. The support of touch for the most common IDEs literally overslept the time. Not only the current IDEs are not ready for advanced work on touch displays, they are not even prepared for everyday touch use stereotypes and are largely limited to interaction via traditional hardware devices (e.g., the ubiquitous keyboard and mouse) [3]. It is not possible to use swiping to scroll, pinch to zoom or tapping the same way as in every other touch-enabled UI. From the IDEs we have analyzed, the only IDE ready touch is Eclipse, statistically the most used IDE for Java language. However, it still offers just fundamental features. As for the second in line, IntelliJ IDEA, despite marked innovative and usable, does not even support the most fundamental ones. Instead of browsing or scrolling through code, the swipe gesture selects text, infringing the common interaction stereotypes and adjustment of panel sizes or zooming by using pinch gesture is not possible at all.

Programming requires the speed of interaction and creativity, touch support is unambiguously a step forward because it brings a different form of user experience and according to Greene [4], supports creative thinking. Our goal is not to force the work exclusively via touch gestures but to create a symbiosis of keyboard, touch and, alternatively, mouse. Begel [5] experimentally verified that programmers are so used to physical keyboard that it would be disadvantageous or at most impossible for them to explicitly use virtual keyboard, which, in addition, has a low precision and does not enable writing by ten fingers.

Supporting common general gestures in IDE may not be enough to speed up the developer work. A specific solution is necessary to not just support the basic gestures, but enable advanced interaction adjusted for a specific IDE platform and programming language. In this paper, we show a method of designing a set of gestures for such advanced interaction as a way of designing the lexical part of a visual language. The focus is mainly on usability [6], [7], [8], domain usability and specificity [9], simplicity of use, learnability and compliance with standard design patterns and guidelines.

The main contributions of this paper are:

- Definition of more complex general-purpose gestures in the context of IDEs.

¹In this paper, the term *hybrid devices* will be used to refer to devices with a keyboard and a touch display such as tablet PCs or combinations of desktop PCs/notebooks with an external touch display.

- A method of collecting, identifying and designing a set of domain-specific gestures via multiple user surveys and interviews including industry programmers and gesture recorder.
- Verbal definition of a domain-specific language of gesture patterns to support highly specialized work with code and consequently a design and realization of drawn gesture input and recognition in a form of a gesture-driven IDE for Android. When compared to a physical keyboard, we hypothesize that touch gestures are *more learnable and memorable than key shortcuts*.
- Evaluation of the proposed approach by means of usability evaluation.

Although our prototype is designed for Android devices, we do not focus on touch devices explicitly (although they may be a possible target), our target group are hybrid devices.

II. A TOUCH-ENABLED IDE

Here, we will describe a design of fundamental gesture interaction for touch devices related to IDEs. Elementary gestures are based on usability guidelines [10], [11], analysis of existing desktop and touch-based IDEs and discussions with industrial developers. Primarily, we followed the Google Material design guidelines [12] that also define the most common gesture patterns (Fig. 1).

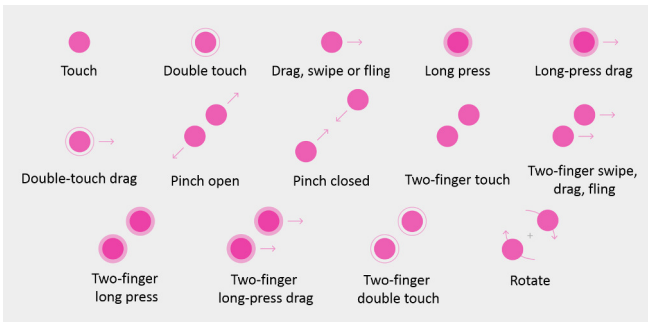


Fig. 1. The mechanics of touch gestures according to the Google Material Guidelines

We identified the following two fundamental categories of gestures related to their design:

- 1) *General-purpose gestures (GPGs)* - used for standard interaction. They can be used in different context, however their meaning is always the same or at least very similar. The most common GPGs are touch, double-touch, swipe or pinch. This category can be considered platform-independent although according to Wroblewski [13], not all gestures are applied on all platforms consistently to the same functionalities.
- 2) *Drawn gestures* - a specific type of gestures that have no standard or specification indicating their form, purpose or semantics. For example, drawing a circle can be used for restoration, rotation or selecting multiple items in a list. The design of semantics of drawn gestures is often intuitive and closely linked to the context of their use.

Breaking the stereotypes of *GPGs* can cause serious usability issues, mainly in the case of advanced or experienced touch device users. Therefore it is good to abide the standard interaction in case of this category. In case of an IDE, it is advantageous to design additional gestures similar to general-purpose ones by using multiple fingers to support more action shortcuts. When designing the set of GPGs, we primarily followed the guidelines of the Android platform and we were also inspired by the iOS platform guidelines for multi-touch gestures [14].

Based on the simple gestures displayed in Fig. 1, we further designed the following additional multi-touch gestures for common IDEs:

- 1) Double-touch to *select words*.
- 2) Tripple-touch to *select lines*.
- 3) Two-finger horizontal swipe for *undo/redo* inspired by the usage on Apple trackpads [14].

III. DOMAIN-SPECIFIC LANGUAGE FOR A GESTURE-DRIVEN IDE

IDEs are used in the domain of programming and to be able to support at least the standard interaction with an IDE, we need to design *domain-specific gestures*.

The set of gestures supported by an application can be perceived as a *sign language*, which the user has to learn to be able to communicate effectively. A correct conceptual model helps users to better understand actions performed upon a UI, thus speeds up the learning process.

From this point of view, the list of domain-specific gestures (both general-purpose and drawn) described further in this paper can be perceived as a *domain-specific language* of the touch-enabled IDE:

- 1) Concrete syntax is defined by the *gestures* themselves: touch or movement, finger count, touch count and movement directions.
- 2) Abstract syntax is defined by the *actions* that the gestures *cause*.
- 3) Semantics is defined by their *type and implementation*.

Because of the limited space and for we design a simple visual language (the set of gestures is kept small), we will not formally describe the semantics of the gesture DSL in this paper. We expect that common language constructs, such as loops, conditions, comments or class and method declarations are known to common programming language designers. Therefore we expect the semantics to be clear from the purpose of the gesture. For each gesture, the semantics is a generated programming language construct. We use templates to generate the constructs similar to the ones used in common IDEs using the Velocity template engine. Concrete and abstract syntaxes are described verbally.

A. General-Purpose Gestures

We designed the following GPGs for touch-enabled IDEs:

- 1) Swiping:
 - a) Two-finger swipe for "*stuck shift*" - gradual shift from method to method by swiping two fingers.

- b) Three-finger swipe to the *end/beginning of the file*.
- 2) Code folding:
- a) Three-finger pinch for *method folding*.
 - b) Four-finger pinch for *"semantic zoom"*.
 - c) Five-finger pinch to *fold all class members*.

"*Semantic zooming*"² is a way of grouping content into specific predefined categories. In case of an IDE it would be a combination of related methods or blocks of code such as getters, setters, void methods, public methods, imports, nested classes, etc., into a single group. After performing the gesture, the user is prompted to select a semantic group. After selection, everything else not belonging to the selected group is folded. This enables to focus only on the important parts of the code.

In case of pinch gestures, the action of pinching-in standardly folds class members and the opposite direction of pinching-out causes unfolding action.

B. Drawn Gestures

As we stated in section II, drawn gestures are not subject to any specifications or conventions. Therefore their design needs to be approached with caution. They are specific for the target domain of use, therefore *domain experts should be included* in the design and development process. A domain analysis was performed to determine the way how programmers interpret different language constructs visually. We interviewed multiple developers from industry and their claims were supplemented by a survey. In the next subsections we will describe the results of the survey that was performed in two iterations.

1) *First Survey - Collection of Patterns*: In order to elicit the correct conceptual model most effectively, a support application for Android touch devices called "*Gesture Recorder*" (Fig. 2) was created. The whole survey process was performed in the application, using a Google Nexus 7 tablet, which asked users to interpret given programming language constructions by directly inducing gestures on touch screen. We used a standard Android tablet for the survey.

We wanted to cover a wide spectrum of programmer experience, therefore we included 68 participants, from which 8 considered themselves advanced programmers, 38 intermediate and 22 were beginner programmers. During the survey, each participant performed the survey independently to prevent influence and was assisted by one member of our research team in order to be able to correctly work with the survey application.

The survey application prompted the participants to interpret the following constructs by a drawn gesture: class declaration, comment, variable declaration, loop statements (both for, while and do-while), condition statement, method declaration and deleting a line.

For each prompt, an example of a language construct was displayed in the application to help the user to better connect his/her mental model with a new gesture. After all language

²More information about semantic zooming can be found in Microsoft official guidelines: <https://msdn.microsoft.com/en-us/library/windows/apps/hh465319.aspx>

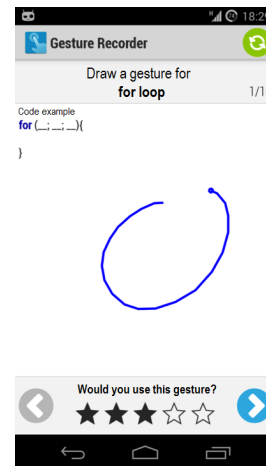


Fig. 2. The GestureRecorder survey application

constructs, participants were given a possibility to add their own idea for a new gesture and language construct. Then they were to fill a questionnaire about their programming experience. The results for each participant were sent to a distant server, manually checked and analyzed.

a) *Evaluation Method*: Similarity of gesture shapes was taken into account via their abstractions into patterns. Differences in size, shape and format were abstracted into a single unified pattern while the direction of drawing was preserved. Differences were neglected only if the conceptual model of the gesture interpretation remained unchanged. The patterns with low repetition rate were omitted. Fig. 3 shows the simplified survey results. Each group represents one language construct and indicates the most common gesture pattern and the number of participants that used the pattern. The direction of drawing is indicated by the small circle representing the final point of drawing.

b) *Results*: The results in the case of a class were influenced by the fact that the respondents come from Slovakia, where class is called "trieda" and begins with the letter "T". Many of the participants also included "N" as in "new", but the number was not as significant. Both were mainly beginners or at least advanced programmers and from our experience, sooner or later older and more experienced programmers begin to in English language explicitly and do not perceive the class declaration and instantiation as the same concept anymore. In case of variable declaration, the results were unclear, therefore we decided not to include the gesture without further research. In case of cycles, as for the shape, the results jointly indicate a circle, as for the direction, the results are unclear.

2) *Second Survey - Suitability of Collected Patterns*: Because the results from the first survey were very variable, we decided to perform a second survey. The goal was to determine which shape of the previously identified ones was the most appropriate for the given language construct.

The second survey was performed on 65 new participants. In the first survey we covered different levels of programmer experience to get universal results but this decision has proven to be wrong. In the second round we rather decided to target






























Class declaration	Variable declaration	Comment	Do-while loop
13 	7 	35 	16 
4 	2 	10 	6 
6 	59  other	23  other	6 
7 			2 
38  other			38  other
While / for loop	Condition	Method declaration	Line deletion
16/12 	9 	17 	10 
9/16 	8 	51  other	9 
43/40  other	3 		49  other
	4 		
	44  other		

Fig. 3. The results of the first survey

the survey explicitly to last-year master students of informatics to achieve more relevant results. We used a questionnaire that contained only questions related to gestures. We included the most frequent answers from the first survey so the users could select from existing choices.

a) *Results:* The results of the second survey were much clearer. In case of class declaration, a significant number of 69% respondents have chosen the shape of the "C" letter which indicates it is an appropriate representation of a the gesture. In case of loops, the results indicate that the direction of drawing is not significant. There are multiple research papers that try to reveal the reason of the direction. A cross-cultural study by Amenomori et al. [15] identified a direct impact of culture and learning methods on the direction. Based on the facts we conclude that the gesture should be unified for all three loop constructions and should support both drawing directions. To unify all loop types, we need to add an additional step after gesture drawing, where the user selects the particular loop construction (while, do-while or for) according to his/her choice.

The results for condition statement were not as definite as in the case of class gesture, however a slightly more significant number of users selected the shape resembling a question mark. Since branching can again be noted by multiple constructs such as if, if-else, switch or shortened if-then-else statement (:?), the gesture can unify all possible constructs and enable a choice after, similarly to the loop gesture.

As for the final question, most of the respondents agreed on a horizontal line for line deletion.

IV. IMPLEMENTATION OF THE GESTURE DSL

At this point we will describe one of the ways of implementing recognition of the described gestures. We chose Android as a model platform mainly because of its accessibility, openness and good support of developing and recognizing custom gestures. However, the mechanics of gestures and the described

procedures are common for any platform. The goal is not to point out to the details but to show the method of developing a gesture recognition engine.

We developed two prototypes of a simple gesture-driven IDE with the goal to design the gesture recognition algorithms and tested it with industrial developers. We focused on maximal usability. We designed and implemented recognition of both complex GPGs and drawn gestures.

Recognition of GPGs is implemented via native platform libraries. Recognition of drawn gestures is more complex and uses machine learning and recognition probabilities. Due to the limited space, we will only describe the implementation of domain-specific *drawn gestures*.

A. Recognition of Drawn Gestures

To recognize drawn gestures we used the Android `com.android.gesture` package containing classes such as `Gesture`, `GestureOverlayView` and `Prediction` enabling possibilities to record, store, read and recognize gestures. The package is undocumented, therefore we will describe the design in more detail. `GestureOverlayView` is a graphical component that enables the actions of gesture recording and displaying it as curves. The recorded and predefined gestures are stored in a binary file located in application memory and contain a list of points and a single name may be assigned to each gesture.

Recognition works on the principle of comparing a drawn gesture with the whole file content and the result is a list of identified patterns with assigned probabilities. However, the default recognition is not very accurate, especially if the stored patterns are slightly similar to each other or if the drawn gesture did not imitate the target pattern too precisely. The more patterns are stored in the database, the less precise is the recognition. In case there are multiple gesture patterns, the recognition rate can be less than 40% which is unacceptable. Taking this fact into account, we needed to adjust the default recognition process so that each pattern could have a *database of multiple possible shapes*. We recommend to create the databases by recording pattern shapes from different users and multiple times and also to record the gestures drawn by the IDE users during their first use by means of a tutorial.

The *recognition probability* p of match is determined by a positive number. The exact meaning of p was not clear since the package is documented, thus the value was determined empirically for each pattern as stated below. In general, the gesture is successfully recognized $p > 1$ and the highest probability is considered as correctly recognized. This applies only after verifying the number of traces forming the gesture. If a circle is recognized after drawing two mirror-inverted C characters, the result will not be considered correct, even though it has the highest p value.

The database for each pattern was created by drawing multiple slightly deformed shapes and in both drawing directions. Because of significant differences between probabilities for patterns with different complexity, an optimal p interval was determined empirically for each pattern as follows:

- *Loop gesture* - is considered to be recognized if $p > 4$.
- *Condition gesture* - the most complex shape of all because of two changes in the direction. Recognized when $p > 3$.
- *Class gesture* - a higher value has proven to be optimal. This gesture is recognized if $p \geq 5$.
- *Comment gesture* - a simple gesture of two consequent lines. Recognized if $p > 10$
- *Line deletion gesture* - the most simple gesture, almost identical to comment, but with only one line and different direction. Valid recognition if $p \geq 10$.

The values were determined by re-entering gestures multiple times and observing the p value after recognition. We verified both positive and negative cases and with the stated p values, we achieved almost 100% success rate.

Since the recognition database is built using a simple form of machine learning and the recognition thresholds are determined empirically by applying probabilities, it is important to consider the number of available patterns in the database. Low number of predefined gestures that are not very complex and similar to each other is optimal. Also, it is very important *not* to continue the machine learning process during application use, because the user might draw incorrect shapes and hinder the recognition of the predefined gestures. Instead, we recommend to allow the user to create his/her own gestures and teach the correct and incorrect shapes to the recognition algorithm. This is also a problem of many systems for recognizing handwritten text. Each person has his/her own writing style and when supporting as many characters as the whole alphabet, the recognition can be incorrect multiple times and lead to reluctance and rejection of using the system. Moreover, the user is only able to remember a limited number of gesture shortcuts and forcing his/her to remember a whole database of patterns can reduce usability.

V. GESTURE DSL - USABILITY TEST

To evaluate the gesture DSL, we created a prototype of a simple gesture-driven IDE for Android according to the design described in this paper and performed a usability evaluation.

Our target group were Java programmers, experienced in using IDE and minimally in using a touch-enabled smartphone or a tablet. For the reasons indicated by Nielsen [16], [7], we performed the usability evaluation with five participants and we targeted at qualitative evaluation. The participants were given tasks related to tutorial, GPGs and 2 tasks targeted at general work with the IDE.

Each testing was performed in the presence of one member of the research team. The sample was represented by five experienced developers with 1-5 years of practice in industry, using mostly IntelliJ IDEA. To avoid influence, the test was performed separately with each participant. In all cases we used Google Nexus 7 tablet and each time the application was reset to its original settings. Two of the respondents were given an external keyboard.

At the beginning, the participants were briefly familiarized with the research and with the testing process and they were given an in-app introductory tutorial. After finishing the

tutorial, the participants were to perform the tasks on a sample class and to express their opinion.

A. Observations and Conclusions

Gesture discoverability was 100%, each participant was able to perform the tasks successfully. The feedback received from the subjects indicated that almost all participants liked the idea of manipulating with the editor using GPGs and generating parts of code using drawn gestures. However, the idea of using it on a tablet was not accepted, each participant expressed the wish to use the gesture language on a common IDE which indicates the potential of our approach on hybrid devices. We concluded that the negative opinion towards tablets is related to writing code using a virtual keyboard. The results were definitely better in case of participants that used an external keyboard.

The gesture for folding all methods was marked most useful, however the users expressed the idea of subsequent use of the gesture multiple times to fold methods to regions. All of them liked the semantic zoom feature and indicated that it could replace the common IDE feature of code structure preview. They suggested to add a filter for constructors, member variables and a filter for displaying all method names.

The possibility to define custom gestures was described as a feature with high potential. Each developer mentioned an example of use, such as definition of custom predefined code parts and templates or combining with macros.

The design of new gestures is a major challenge. The gestures have to be simple, easy to learn and quick to write. This holds for the set presented in this paper, but with a growing number of gestures, their complexity might rise significantly, which could complicate their use. It might be beneficial to explore multiple combinations of gestures and patterns [17] [18] of their use.

VI. RELATED WORK

Biegel et al. [19] propose an inspirational approach to "touchifying" an IDE. They highlight multiple issues of common IDEs related to supporting touch handling in general. Authors present a solution on Eclipse IDE. Compared to our gesture set, they used already existing GPGs to support multiple functionalities. In their study, they did not include domain-specific drawn gestures. The fact that programmers rather prefer the combination of a traditional PC, touch display and a mouse shown in their paper *supported our findings* and confirms that this approach has a potential to be used on hybrid devices.

There are multiple works that focus on code refactoring triggered by gestures, such as the works by Murphy-Hill et al. [20] and Raab et al. [21]. Compared to their solution, we use a small set of gestures for commonly used features and for the sake of usability we never use multiple composed gestures to trigger a single operation. Lee et al. [22] propose a solution where refactoring can be triggered via drag-and-drop gestures but in contrast to our work they optimized the gestures for using a mouse.

Hesenius et al. [23] introduced an environment for tablets with advanced features but focused primarily on creating a new environment, not adaptation to hybrid devices. Delimarschi et al. [3] describe a natural UI for graphical IDE using a Kinect voice and physical gesture commands. Although we favour the potential of voice commands as means to speeding-up the developer work, voice recognition is still not mature enough to fully support multiple languages. As for physical gestures, the potential is less definite. Edwards and Barnette [24] unsuccessfully used tablet PCs with a pen in a laboratory programming course without adapting the software to this novel device.

Begel et al. [5] mentioned that it is possible that developers are less efficient by using a natural interface instead of typing. This is related to the fact that typing on a physical keyboard is already ingrained in every developer's habits, while more natural interaction might not be as standard in the development process. We agree that in the case of IDEs, not each natural interaction type is beneficial, however we argue for touch input being advantageous based on our observations and user feedback. Direct manipulation characteristics of touch helps to reduce the cognitive load [19]. Further investigation is needed to support this claim.

VII. CONCLUSION

We introduced the concept of a DSL of gestures in the context of IDEs. We performed multiple studies to explore the visual representations of conceptual models of programming constructs and developed a prototype tool which served to conduct a pilot study to assess the feasibility of our approach. Despite the challenges and current limitations, we believe that it has the potential to improve and speed-up development and ease of use. To make the approach more accessible and verifiable in the context of hybrid devices, we have set forth the following goals: 1. Implement a prototype into a common desktop IDE such as Eclipse, IntelliJ IDEA or NetBeans in form of a plug-in. 2. Integrate learning capabilities that would allow the system to adapt to the user and his/her drawing style and allow the user to add new custom gestures via the existing IDE-native code template engine. 3. Generalize the approach beyond Android and Java, for a variety of operating systems and programming languages. 4. Experimentally identify the optimal number, complexity and combinations of drawn gestures.

ACKNOWLEDGMENT

This work was supported by VEGA Grant No. 1/0341/13 Principles and methods of automated abstraction of computer languages and software development based on the semantic enrichment caused by communication.

REFERENCES

- [1] A. Schade. Large touchscreens: What is different? [Online]. Available: <http://goo.gl/jCsnDJ>
- [2] I. Zayour and H. Hajjdiab, "How much integrated development environments (ides) improve productivity?" *JSW*, vol. 8, no. 10, pp. 2425–2431, 2013. doi: 10.4304/jsw.8.10.2425-2431
- [3] D. Delimarschi, G. Swartzendruber, and H. Kagdi, "Enabling integrated development environments with natural user interface interactions," in *Proc. 22nd Intern. Conf. on Program Comprehension*, ser. ICPC 2014. New York, NY, USA: ACM, 2014. doi: 10.1145/2597008.2597791. ISBN 978-1-4503-2879-1 pp. 126–129.
- [4] S. L. Greene, "Characteristics of applications that support creativity," *Commun. ACM*, vol. 45, no. 10, pp. 100–104, Oct. 2002. doi: 10.1145/570907.570941
- [5] A. Begel and S. L. Graham, "An assessment of a speech-based programming environment," in *IEEE Symp. on Vis. Languages and Human-Centric Comp., VL/HCC'06*, Sept 2006. doi: 10.1109/VLHCC.2006.9 pp. 116–120.
- [6] Usability partners, "ISO standards in usability and user-centered design". [Online]. Available: <http://usabilitypartners.se/about-usability/iso-standards>
- [7] J. Nielsen, *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN 0125184050
- [8] D. A. Norman, *The Design of Everyday Things*. New York, NY, USA: Basic Books, Inc., 2002. ISBN 9780465067107
- [9] M. Bačiková and J. Porubán, "Ergonomic vs. domain usability of user interfaces," in *2013 The 6th Intern. Conf. on Human System Interaction (HSI)*, June 2013. doi: 10.1109/HSI.2013.6577817. ISSN 2158-2246 pp. 159–166.
- [10] D. Wigdor and D. Wixon, *Brave NUI World: Designing Natural User Interfaces for Touch and Gesture*, 1st ed. San Francisco, CA, USA: Elsevier (Morgan Kaufmann Publishers Inc.), 2011. ISBN 978-0-12-382231-4
- [11] C. Abras, D. Maloney-krichmar, and J. Preece, "User-centered design," in *Encyclopedia of Human-Computer Interaction*, Bainbridge, W. Thousand Oaks: Sage Publications, 2004.
- [12] Google 2014: Material design guidelines. [Online]. Available: <http://www.google.com/design/spec/material-design>
- [13] L. Wroblewski. (2010) Touch gesture reference guide. [Online]. Available: <http://www.lukew.com/ff/entry.asp?1071>
- [14] Apple 2014: Mac basics: Multi-touch gestures, official apple support". [Online]. Available: <https://support.apple.com/en-us/HT4721>
- [15] M. Amenomori, A. Kono, J. S. Fournier, and G. A. Winer, "A cross-cultural developmental study of directional asymmetries in circle drawing," *Journ. of Cross-Cultural Psychology*, vol. 28, no. 6, pp. 730–742, 1997. doi: 10.1177/0022022197286005
- [16] N.-N. Group. How many test users in a usability study? [Online]. Available: <http://www.nngroup.com/articles/how-many-test-users/>
- [17] J. Kollár et al., "Plero: Language for grammar refactoring patterns," *FedCSIS '13*, pp. 1503–1510, 2013.
- [18] M. Nosál and J. Porubán, "Xml to annotations mapping definition with patterns," *Comp. Sci. and Inf. Syst.*, vol. 11, no. 4, pp. 1455–1478, 2014. doi: 10.2298/CSIS130920049N
- [19] B. Biegel et al., "U can touch this: Touchifying an ide," in *Proc. 7th Int. Works. on Coop. and Human Asp. of Soft. Eng.*, ser. CHASE 2014. New York, NY, USA: ACM, 2014. doi: 10.1145/2593702.2593726. ISBN 978-1-4503-2860-9 pp. 8–15.
- [20] E. R. Murphy-Hill, M. Ayazifar, and A. P. Black, "Restructuring software with gestures," in *VL/HCC*. IEEE, 2011. doi: 10.1109/VLHCC.2011.6070394. ISBN 978-1-4577-1246-3 pp. 165–172.
- [21] F. Raab, C. Wolff, and F. Ehtler, "Refactorpad: Editing source code on touchscreens," in *Proc. 5th ACM SIGCHI Symp. on Eng. Interactive Comp. Syst.*, ser. EICS '13. New York, USA: ACM, 2013. doi: 10.1145/2494603.2480317. ISBN 978-1-4503-2138-9 pp. 223–228.
- [22] Y. Y. Lee, N. Chen, and R. E. Johnson, "Drag-and-drop refactoring: Intuitive and efficient program transformation," in *Proc. 2013 Intern. Conf. on Soft. Eng.*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013. ISBN 978-1-4673-3076-3 pp. 23–32.
- [23] M. Hesenius, C. D. O. Medina, and D. Herzberg, "Touching factor: Software development on tablets," in *Soft. Comp.*, ser. Lect. Notes in Comp. Sci., vol. 7306. Springer, 2012. doi: 10.1007/978-3-642-30564-1_10. ISBN 978-3-642-30563-4 pp. 148–161.
- [24] S. H. Edwards and N. D. Barnette, "Experiences using tablet pcs in a programming laboratory," in *Proc 5th Conf. on Inf. Techn. Edu.*, ser. CITC5 '04. New York, USA: ACM, 2004. doi: 10.1145/1029533.1029573. ISBN 1-58113-936-5 pp. 160–164.