Expressing, Exploring, Communicating, Revising, and Refining Ideas

Robert Hirschfeld
Software Architecture Group, Hasso Plattner Institute, University of Potsdam, Germany

We use computers to understand phenomena. They are our tools to express, explore, communicate, revise, and refine our ideas. And they can help us simulate possible and impossible futures.

For that, we like to alternate between concrete examples and abstract representations to advance our understanding of complex domains of interest to us. Working out meaningful examples and finding powerful abstractions in their support are not only part of a process but also a means to an end since those examples and abstractions are a manifestation of our knowledge about an area of interest.

We welcome notations that allow us to express our intents directly to help improve our understanding and to communicate and collaborate with others. And while doing so, we refine not only our knowledge about the world but also the ways we would like to talk about it.

Our tools help maintain our knowledge and offer ways to interact with it. They support our explorations of the concrete and abstract, provide access to anything of interest, and offer the right feedback at the right time. They allow us to express the same ideas using different notations for different points of view [16]. And they allow us to express different ideas using different notations that can coexist next to each other in complex, multifaceted domains.

By allowing for imperfection and incompleteness, our tools are robust and malleable to help us make mistakes and learn from them. And as both our knowledge and the way we choose to represent it evolve, we also evolve our notations and tools.

Substrates are the technology that enables and helps us achieve that.

Substrates encourage Computational Thinking [4] and Programming as Theory Building [10]. They allow for the creation of Personal Dynamic Media [8] and can be a step toward the vision of the Dynabook [9].

Substrates accommodate the co-existence and symbiosis of concrete data, abstract code, and evolving processes and simulations around them. State, behavior, execution, and tools and interfaces become one (objects, actors).

Substrates offer liveness and immediacy.

Curated examples can help express the intent of abstractions, allowing us to approach them from different levels of confidence, depending on the context of our discourse.

Substrates consist of small, simple kernels that together contribute to the experiences built upon them, adequately representing the intrinsic complexity of the domains of interest they support while reducing accidental complexity imposed by the infrastructure they provide.

To remain useful, substrates need to evolve with our understanding of the world.

The Software Architecture Group [15] is working toward such substrates and, more importantly, the possibilities they will enable for us.

We are striving for better and different approaches to authoring and authoring environments, with *live and exploratory programming* [13] being just one such approach, but the most promising and rewarding to us.


© Robert Hirschfeld;
licensed under Creative Commons License CC-BY 4.0
1st Workshop on Substrates 2025.
Editors: Jonathan Edwards and Tomas Petricek and Robert Hirschfeld; Article No. 23; pp. 23:1–23:3

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Among our systems and research prototypes that contribute to our goals are *Sandblocks*, *Bablylonian Programming*, and *Vivide*.

Sandblocks is a system that allows its users to automatically generate structured editors for every language with a formal grammar available [1]. It allows a direct interaction with the deep structure of programs and their run-time representations and to some degree a freedom of notation [14, 2].

Babylonian Programming is an approach to example-based live programming environments that enables programmers to use examples in larger systems, which span multiple modules [11]. It unites the world of static and dynamic tools into one combined view that provides immediate access to all information of interest and available [12].

Vivide is a platform for seamless integration of tools into one environment that provides views on static and dynamic program-related information and presents a system as a single unit [17]. It allows for low-effort, high-quality tool building and maintenance and treats dedicated, customized development tool support as part of the application domain.

Most of our work is built on or extends *Squeak/Smalltalk* [5, 3] and *Lively Kernel* [7, 6], two of the substrates we are most interested in from a technological point of view, the programming culture they encourage, and the visions of the people that created them.

References

1 Tom Beckmann, Patrick Rein, Stefan Ramson, Joana Bergsiek, and Robert Hirschfeld. Structured Editing for All: Deriving Usable Structured Editors from Grammars. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–16, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3544548.3580785.

2 Tom Beckmann, Jan Reppien, Jens Lincke, and Robert Hirschfeld. Supporting Construction of Domain-Specific Representations in Textual Source Code. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Programming Abstractions and Interactive Notations, Tools, and Environments (PAINT)*, pages 17–28, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3689488.3689990.

3 Adele Goldberg and David Robson. *Smalltalk-80: The Language and Its Implementation*. Addison-Wesley, 1983. URL: https://dl.acm.org/doi/book/10.5555/273.

4 Mark Guzdial, Alan C. Kay, Cathie Norris, and Elliot Soloway. Computational Thinking Should Just be Good Thinking. *Communications of the ACM (CACM)*, 62(11):28–30, 2019. doi:10.1145/3363181.

5 Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. Back to the Future: The Story of Squeak, A Practical Smalltalk Written in Itself. In *Proceedings of the 12th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 318–326. Association for Computing Machinery, 1997. doi:10.1145/263698.263754.

6 Daniel Ingalls, Tim Felgentreff, Robert Hirschfeld, Robert Krahn, Jens Lincke, Marko Röder, Antero Taivalsaari, and Tommi Mikkonen. A World of Active Objects for Work and Play: The First Ten Years of Lively. In *Proceedings of the ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*, pages 238–249. Association for Computing Machinery, 2016. doi:10.1145/2986012.2986029.

7 Daniel Ingalls, Krzysztof Palacz, Stephen Uhler, Antero Taivalsaari, and Tommi Mikkonen. The Lively Kernel: A Self–supporting System on a Web Page. In *Self-Sustaining Systems (S3)*, pages 31–50, Berlin, Heidelberg, 2008. Springer-Verlag. doi:10.1007/978-3-540-89275-5_2.

8 Alan Kay and Adele Goldberg. Personal Dynamic Media. *IEEE Computer*, 10(3):31–41, 1977. doi:10.1109/C-M.1977.217672.

9 Alan C. Kay. A Personal Computer for Children of All Ages. In *Proceedings of the ACM annual conference, Volume 1*. Association for Computing Machinery, 1972. URL: https://dl.acm.org/doi/10.1145/800193.1971922.

10 Peter Naur. Programming as Theory Building. *Microprocessing and Microprogramming*, 15(5):253–261, 1985. doi:10.1016/0165-6074(85)90032-8.

11 David Rauch, Patrick Rein, Stefan Ramson, Jens Lincke, and Robert Hirschfeld. Babylonian-style Programming: Design and Implementation of an Integration of Live Examples Into General-purpose Source Code. *The Art, Science, and Engineering of Programming*, 3(3):39, 2019. doi:10.22152/programming-journal.org/2019/3/9.

12 Patrick Rein, Christian Flach, Stefan Ramson, Eva Krebs, and Robert Hirschfeld. Broadening the View of Live Programmers. *The Art, Science, and Engineering of Programming*, 8(3):38, 2024. doi:10.22152/programming-journal.org/2024/8/13.

13 Patrick Rein, Stefan Ramson, Jens Lincke, Robert Hirschfeld, and Tobias Pape. Exploratory and Live, Programming and Coding – A Literature Study Comparing Perspectives on Liveness. *The Art, Science, and Engineering of Programming*, 3(1):33, 2019. doi:10.22152/programming-journal.org/2019/3/1.

14 Charles Simonyi, Magnus Christerson, and Shane Clifford. Intentional Software. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 451–464. Association for Computing Machinery, 2006. doi:10.1145/1167473.1167511.

15 Software Architecture Group, Hasso Plattner Institute, University of Potsdam, Germany. URL: https://www.hpi.uni-potsdam.de/swa/people/.

16 Susan Leigh Star and James R. Griesemer. Institutional Ecology, "Translations," and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907–1939. *Social Studies of Science*, 19(3):387–420, 1989. URL: http://www.jstor.org/stable/285080.

17 Marcel Taeumel, Bastian Steinert, and Robert Hirschfeld. The VIVIDE Programming Environment: Connecting Run-time Information With Programmers' System Knowledge. In *Proceedings of the ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*, pages 117–126, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2384592.2384604.

********** PAPER 16 **********
AUTHORS: Robert Hirschfeld
TITLE: Expressing, Exploring, Communicating, Revising, and Refining Ideas

++++++++++ REVIEW 1 (Gilad Bracha) +++++++++
"We use computers to understand phenomena." Well put. Very much in line with the Scandinavian school of OO - Beta etc., "concepts and phenomena". But we use computers for more than that: we use them to control things. Examples include setting the temperature in our home, or guiding a car, drone or rocket to its destination; there are many others.

That's my quibble. Keep up the good work.