End-User Design

Prof. Dr. Alexander Repenning
University of Lugano

Problem

Are UML diagrams a good tool to teach middle school students how to make video games? Probably not, but what kinds *end-user design* aids such as mental models, scaffolding structures, examples or other kinds of objects to think we can we give to end-users in order to gradually introduce them to good programming practice?

In the end-user programming arena the fundamental challenges have gradually shifted from basic syntactic challenges towards semantic challenges including the need to convey an understanding of design and engineering principles relevant to end-users. Visual programming has significantly lowered the threshold of programming [1] mostly by sharply reducing or even completely eliminating syntactic programming challenges. Visual programming languages using drag and drop mechanisms as programming approach make it virtually impossible to create syntactic errors. Even traditional text-based end-user programming approaches nowadays provide useful tools such as syntax coloring, symbol completion and wizards to reduce syntactic problems quite effectively. With the syntactic challenge being – more or less – out of the way we can focus on the semantic level of end-user programming.

At a semantic level the challenges ahead are substantially more complex. How can the user explore what a program does or design a new program in a systematic way? Programming languages including interactive debuggers such as Ruby, Lisp and Smalltalk allow users to comprehend code gradually by allowing them to decompose code into smaller code fragments such as individual methods that they could test in the context. The comprehension problem is a hard one but still relatively simple compared to the composition problem. If we have a specific problem in mind how do we think about the problem? How do we gradually map the problem, using some form of programming, to a solution? The “cursor is blinking in the upper left corner of an empty window” problem is hard because we know next to nothing about the users’ intensions. Are they trying to write a game or trying to solve a bookkeeping problem. How should they think about the problem in general? All of this leads to the challenge of end-user design.

Our Work

From systems perspective we have explored the notion of *tactile programming* [5] as means to make program comprehension and composition more concrete. Any piece of code can be tested and explained through an explanation generator interpreting a program and presenting in to the user. These explanations include specific representations of what function parameters means and even how their actual values should be interpreted. We have also explored knowledge-based approaches to computer supported program synthesis [4]. Programming by Analogous Examples has used small semantic annotation provided by the user to enable very high-level reuse.

Dagstuhl Seminar Proceedings 07081
End-User Software Engineering
http://drops.dagstuhl.de/opus/volltexte/2007/1099

From a pedagogical perspective we have tried to convey design thinking using a design scaffolding processes. Our Gamelet design [2] approach provides a couple of concrete design stepping stones and game design patterns to turn a game idea description into a running game. Using AgentSheets [3] and AgentCubes [1] we have employed variants of the approach in education. At the university level we have graduate and undergraduate computer science students using a version of the Gamelet design process including UML diagrams to build sophisticated games. At the middle school level we successfully experimented with more informal versions of the Gamelet design approach. This work has resulted in the world most compact game design course in which middle school children build for instance a simple Frogger game in very little time.

Challenges

Our initial efforts into end-user design suggest that it is possible to create semi-formal approaches to convey design knowledge to end-users. However, we feel that we are only at an early beginning and wonder if there is a more general *science of end-user design*.

- Can we create useful end-user design methods by scaling down existing design and engineering methods?
- Could – and should – there be something like an end-user UML diagram or and end-user pattern?
- Would we just be dumbing down real design issues and loose all value (e.g., The Idiots Guide to Software Design & Engineering)?
- If we make clever tools such as natural language problem statement parsers that would automatically create finished games would the users still be able to gain design knowledge?

References

[1] Repenning, A. and Ioannidou, A., AgentCubes: Raising the Ceiling of End-User Development in Education through Incremental 3D. in IEEE Symposium on Visual Languages and Human-Centric Computing 2006, (Brighton, United Kingdom, 2006), IEEE Press.

[2] Repenning, A. and Lewis, C., Workshop: Gamelet Design for Education. in Annual Games, Learning & Society Conference (GLS 2006), (Madison, Wisconsin, 2006).

[3] Repenning, A. and A. Ioannidou 2005. What makes End-User Development Tick? 13 design guidelines. End-User Development. F. Paterno and V. Wolf. Dordrecht, Kluwer.

[4] Ioannidou, A., Programmorphosis: a Knowledge-Based Approach to End-User Programming. in Interact 2003: Bringing the Bits together, Ninth IFIP TC13 International Conference on Human-Computer Interaction, (Zürich, Switzerland, 2003).

[5] Repenning, A., and J. Ambach, "Tactile Programming: A Unified Manipulation Paradigm Supporting Program Comprehension, Composition and Sharing," Proceedings of the 1996 IEEE Symposium of Visual Languages, Boulder, CO, Computer Society, 1996, pp. 102-109.