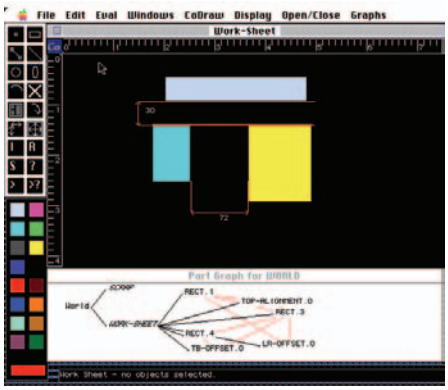
Designers Need End User Software Engineering
Mark D Gross, School of Architecture, Carnegie Mellon University

For many years I have been interested in how design works, and in how to support designers doing design. Inspired by tools like Macsyma (and later Mathematica, etc.), I became interested in building end-user languages and tools that are in a sense general-purpose systems and yet that end users can tailor or customize to correspond to their own needs. Whether there are specific structures and operations that are unique to design (across the disciplines, but distinct perhaps from other kinds of problem solving) I do not yet know, but it seems at least plausible that programming languages for design may have a special character. That question that has framed much of my work.

My earlier work was on design as exploring constraints, and computational support for end user designers to set up systems of constraints to describe, and then solve, problems couched in this framework. Inspired by Sketchpad, CoDraw [1] was a end-user extensible constraint based graphical editor (2D CAD system) in which every object was described as a collection of relationships among its parameters. Although CoDraw provided a few primitive objects to seed the system, end users could define new objects either by subclassing previously defined ones, or by identifying variables and their relationships. A CoDraw end user could extend the system on the fly, using a set of cards (each similar to a small spreadsheet) to define objects in terms of their variables, and relationships. Various graph display and editing features allowed end users to specify part-whole and sub/class relationships among objects in the system, to trace dependencies among constraints and inspect justification paths for derived values. As end users added new objects and relationships to the system they could also add items to tool palettes. In this system, end users could extend and build up the underlying language (a Lisp embedded constraint management system) as well as the graphical user interface that was used to build the CoDraw application.



Through experience with the CoDraw constraint based CAD system and its language, I learned that the designers I worked with were unsatisfied with describing their knowledge in terms of objects, variables, and constraints. They found even the menu and tool palette way of making drawings stilted and difficult to use. They wanted to draw. This led me to begin work on the Electronic Cocktail Napkin, a pen based freehand drawing system that is designed as an interface for knowledge-based systems [2]. The Napkin uses a symbol (glyph) recognizer to identify the freehand strokes that the designer draws. An end user trains the glyph recognizer on the fly, adding new symbols to the program's repertoire or showing the program new ways to draw previously defined symbols. The same is also true for more complex drawing configurations: an end user builds up a grammar (composed of previously defined glyphs and configurations) that corresponds to a specific diagrams in a specific domain. The end user builds the grammar by demonstrating examples of configurations; the Napkin program constructs a description of the relationships that

the user's configuration contains and the user then adjusts this description by making it more or less specific. These visual grammars may correspond to a specific knowledge domain (e.g., analog or digital electronics) or to a highly idiosyncratic way of using diagrams that is specific to the end user (as, for example, graphic or architectural designers may be wont to do).

I have recently become interested in what kinds of tools and environments might be useful for users who lack technical knowledge in programming or electronics to build working prototypes of embedded (tangible, pervasive, ubiquitous) computing systems. To build even a prototype of an embedded system may require expertise in programming, electronics (sensors and actuators), as well as mechanical, physical, and materials design. Each of these domains can alone be daunting and there are few designers who would be capable of managing the ensemble together. End user programming environments for microcontrollers include traditional coding (e.g. in C or Java) or visual languages (e.g., Scratch or Max/MSP). The electronics design similarly can require sophisticated knowledge of components, their interactions; and finally the physical, mechanical design demands that the end user master a 3D modeler, perhaps a kinematics simulator, and so on. We would like to build a "design fusion" environment where a novice programmer (a designer of embedded computing artifacts) can describe the set of desired behaviors and functions, and construct and debug the software, electronics, mechanical, and physical systems to implement these behaviors and functions. This design fusion environment should be powerful (not limiting the designer to a trivial subset of possibilities) yet simple so as to enable the designer to express the desired behaviors and functions without attending to irrelevant low-level language details.

One point in this space is roBlocks [3], a 'computationally enhanced construction kit' that is intended for young people to build simple robots out of blocks, without first demanding that they learn electronics, programming, and the associated manual skills. RoBlocks consists of small (40mm) cubes that snap together magnetically. Each block contains a microcontroller and provides either a sensor, an actuator, or some arithmetic or logic. The blocks snap together to make a robot, transmitting power and data from face to face. In this way a novice user can assemble (in one move) both the physical construction as well as the mechanics and programming necessary to make the robot behave. For example, snapping a photosensor block on a motor block would make a phototropic robot (that moves toward light). Adding an inverter block between the two would make a photophobic robot. We have built a small working set of roBlocks and are currently considering how to build a next-level screen-based language for users who have mastered the physical programming level and would like to change the behaviors of specific blocks.

1. Gross, M.D. Graphical Constraints in CoDraw. in Tanimoto, S. ed. IEEE Workshop on Visual Languages, IEEE Press, Seattle, 1992, 81-87.

2. Gross, M.D. and Do, E.Y.-L. Drawing on the Back of an Envelope: a framework for interacting with application programs by freehand drawing. Computers and Graphics, 24 (6). 835-849.

3. Schweikardt, E. and Gross, M.D., roBlocks: A Robotic Construction Kit for Mathematics and Science Education. in International Conference on Multimodal Interaction, (Banff, Alberta, 2006), ACM.