# An analysis of the programmable Culturally Situated Design Tools from an HCI perspective.

William Babbitt

Rensselaer Polytechnic Institute

Presented at the Human Computer Interaction Symposium

Rensselaer Polytechnic Institute

Wednesday, December 5th 2012

## Abstract

The Culturally Situated Design Tools (CSDTs) are software programs deployed online at http://csdt.rpi.edu. The purpose of these tools is to teach mathematics and computer science concepts through simulation of cultural artifacts. For example, the Adinkra programmable CSDT seeks to teach mathematics concepts through the Akan (Ghana, West Africa) practice of Adinkra stamping. In this paper, I will analyze the programmable CSDTs (pCSDTs) from a Human-Computer Interaction (HCI) point of view. I will examine how the pCSDTs incorporate ethnographic design strategies from the initial software concept to the user experience. I will consider the role of cognitive task loading in the overall software design for the pCSDTs. In addition, I will look at the role that mental models play in the use of the pCSDTs and how understanding these mental models can leverage our improvement of the development process.

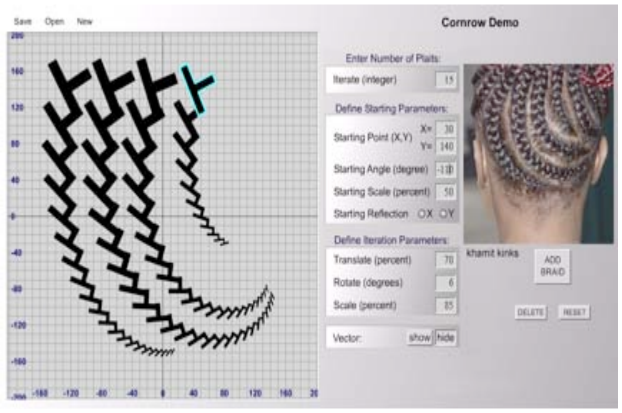
Table of Contents

## Introduction

The Culturally Situated Design Tools (CSDTs) are web-based software applications available at http://csdt.rpi.edu. They are the brainchild of Dr. Ron Eglash, professor of Science and Technology Studies at Rensselaer Polytechnic Institute. The purpose of the CSDT tools is to teach mathematics concepts through simulation of cultural arts. The tools teach a variety of mathematical concepts that range from transformational geometry, Cartesian and polar coordinate systems, to pre-algebra. For example, the Cornrow Curves CSDT tool uses the basic shape of a cornrows plait to teach transformational geometry through simulation of the cornrow braids in that hairstyle. The Adinkra CSDT uses the Akan (Ghana, West Africa) practice of Adinkra stamping to teach the concepts of Cartesian and polar coordinates, as well as linear and logarithmic spirals through simulation of the Adinkra stamp shapes.

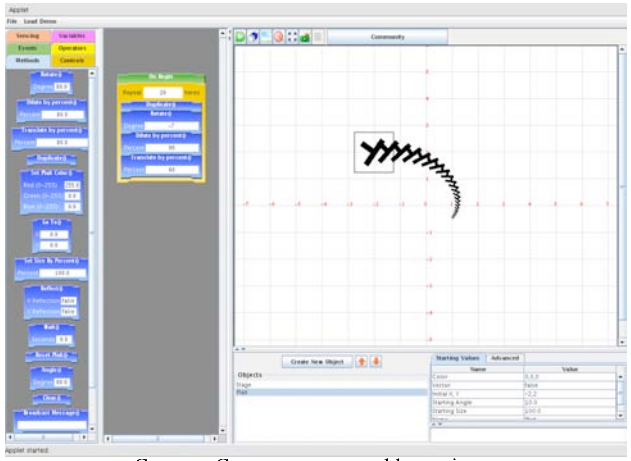
The CSDTs mission of teaching mathematical concepts has been extended to teaching programming and computer science concepts. The programmable CSDTs (pCSDTs) build upon their CSDT predecessors by adding the teaching of such concepts as sequential code execution, looping and conditional branching, along with the use of variables in programming through simulation of these same cultural artifacts.

In this paper, I will analyze the pCSDT tools from an HCI point of view. We will look at how the pCSDTs incorporate ethnographic design strategies from the initial concept of a tool, to the user experience of working with the tool. We will consider the role that cognitive task loading in the overall software design for the pCSDTs affects user experience and user learning outcomes. In addition, we will look at the role mental models play in use of the tools and how understanding those mental models can help to inform and leverage our improvements through the development process.



Enter Number of Plaits:
Iterate (integer) 15
Define Starting Parameters:
Starting Point (X,Y) X= 30 Y= 140
Starting Angle (degree) -110
Starting Scale (percent) 50
Starting Reflection ○X ○Y
Define Iteration Parameters:
Translate (percent) 70
Rotate (degrees) 6
Scale (percent) 85
Vector: show hide

Cornrow Curves, mathematics version



Cornrow Curves, programmable version

## Ethnographic Design

The use of ethnography in human computer interaction can mean different things to different people, and the term can vary both in scope and breadth in its use when we talk about HCI relationships. Using an ethnographic approach in HCI design can mean to gather qualitative information about the tasks that users perform with a computer system (Blomberg et. al, 2003). Using an ethnographic approach in HCI design can be about the way in which researchers approach the gathering of this user information, or it can be about how individual users perform specific tasks. In general, we can consider ethnographies to be 'user stories' that recount the experiences of the users interaction with a computer system that can be used to understand and improve the use of that system.

In the case of the programmable CSDTs, I will use the term ethnography both in its traditional anthropologic sense, which is to gather stories about how people live their everyday lives, which includes how people produce craft items of cultural significance. I will also use ethnography in its HCI sense to refer to stories of student interaction experiences with pCSDT simulation software. In the first case, I would not be able to construct software that seeks to simulate culturally significant crafts without the invaluable perspective of the craft artisans. In the second case, ethnographic studies serve as the primary tool in software design that communicates to us as developers whether or not the software is succeeding at its intended purpose. In this case the primary goal is to teach mathematics and computer science concepts. Both sets of ethnographic stories play an important role in informing the design of the computer simulation, which results in an accurate simulation design, as well as a competent pedagogic approach for teaching mathematics and computer science concepts.

In approaching ethnography for both the use and design of the pCSDTs, keen observation has been absolutely critical. Verbal statements may not always match what the artisan or the user is actually feeling or thinking (Blomberg et. al, 2008). The study participant may not be able to adequately communicate his or her feelings or thoughts to the ethnographer in a way that will be understood. In this ethnography work, I have attempted to adopt the participant-observer role (Blomberg et. al, 2008) which made me an active participant while recording observations.

## Ethnographic development stories

The programmable Adinkra stamping simulation design has relied heavily on the ethnography's provided by Gabriel Boakye, an Adinkra artisan in Ghana, West Africa. I first met Gabriel in the summer of 2011 and then again in 2012, at his Adinkra shop in the little town of Ntonso, Ghana where I was able to interview him about his Adinkra stamping craft. There are many Adinkra artisans in that area of Ghana, but Gabriel is arguably the Adinkra master craftsman of the Ntonso village. In my discussions with Gabriel, he has generously provided invaluable information concerning the symbolic meaning of Adinkra, all of the different steps in creating Adinkra, from ink production and stamp carving, to the actual stamping of the Adinkra shapes on to woven fabric. It was from Gabriel that I learned that each of the Adinkra stamp shapes possess a culturally important meaning, and a crafts person will carefully choose a combination of Adinkra images to place on a fabric, resulting in an Adinkra message. Gabriel also taught me that a fabric does not become 'Adinkra' cloth until it has been stamped with an Adinkra symbol.

Gabriel Boakye and the author discussing Adinkra stamping, Ntonso Ghana in the summer of 2011.

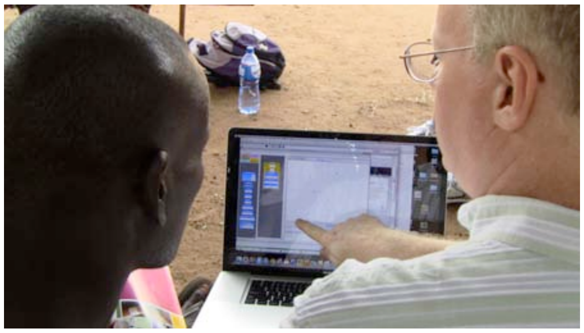
From my experience with Gabriel in 2011, I began to craft our Adinkra Culturally Situated Design Tool. What would be needed was a way to programmatically stamp on a computer screen the different geometric shapes that are found within the Adinkra stamps that Gabriel had shown us. It became clear that the complicated Adinkra symbols could be broken into smaller more manageable shapes such as lines, linear spirals, and logarithmic spirals. These three design elements could be assembled in building block fashion with programmatic elements that determined length, orientation, and constant or varying thickness to reproduce the Adinkra symbols. Actually, this part of the development proved to be relatively easy, deciding how to represent the programmatic elements would prove to be much more of a challenge.

An important consideration in the design of the Adinkra stamping simulation included determining the degree of complexity exposed to the student through the scripting interface. We have always thought that this aspect of the simulation development would be an exceptionally fine line to navigate, but this struggle clearly had become the greatest challenge. Being too cautious in exposing complexity, which would in effect be making the software 'easier' to use could result in user boredom; however exposing too much complexity could overwhelm users leading quickly to user frustration. Gabriel provided us with guidance through the ethnographic stories that he shared.

Adinkra is created through the act of stamping, and as Gabriel stamps the images he is using already carved stamps where such decisions concerning angles and curves have already been fixed in the act of carving the stamp. Likewise, in our simulation, many of the representation particulars have already been determined behind the scenes from the scripting interface, in the code that each codelet represents. The user does not need to be concerned about how to get the image on to the screen; they just need be concerned about the adjustments that result in the output that the user desired. This reduction in complexity is especially crucial to 'get right' because it can be the determining factor of tool success or failure with users.

I worked on the Adinkra stamping tool from the summer of 2011 to the summer of 2012, when I was able to present the software to Gabriel at his shop in Ntonso. In our second encounter, I was able to gather additional feedback that indicated that I did not have everything exactly correct in the simulation. For example, I had placed the Cartesian grid that serves as scaffolding to the user, to aid in design element placement, such that all four quadrants were available to the user. Gabriel immediately determined that from a craft point of view, an Adinkra artisan would never look at stamp placement in that way. This was a particularly interesting discussion because I had made that design decision based on my desire to include the four quadrants from a mathematics pedagogic point of view. In the end, I deferred to Gabriel and placed the entire simulation output screen into the first quadrant with coordinates running from 0 on the left increasing to the right and vertically.

The Adinkra stamping software seeks to reproduce the components of Adinkra stamp shapes in simulation. These components include the use of logarithmic spirals, linear spirals, as well as straight lines of varying widths at varying angles. The simulation itself takes place within the confines of a Java applet that includes a scripting interface, and an output window where the result of the script that has been built by the user is displayed. The output window also provides the graphical scaffolding of the Cartesian grid to aid the user in composing their Adinkra design. It is from the use of these software components that we gather ethnographic user stories.


The author demonstrating the Adinkra stamping simulation to Gabriel Boakye in the summer of 2012, Ntonso Ghana.

## Ethnographic user stories

The ethnographic stories that can be gathered from the users of the simulation software can focus on specific portions of the software interface and can tell us about these different software elements depending on what we are interested in. Ethnographies that focus on the scripting interface will provide us with valuable information concerning the efficacy of the tool in teaching computer science topics such as iteration, conditional program flow and algorithmic thinking. Ethnographies that concentrate on the output window, will allow us to infer if the tool is effective in knowledge transfer answering such questions as 'did the user produce an artifact that demonstrates the use and understanding of the mathematics or computer science concepts?'

Stories gathered from user experiences provided crucial information in determining the appropriate level of complexity for the scripting interface. For example, an early question that arose in the development process was how much of the calculation for generating a logarithmic or linear curve on the screen should be shown to the user, and how much should just simply be handled behind the scenes. One choice was to put a point on the screen and then leave it to the user to take it from there. Had we chosen that path, the script to create the curve would have needed to involve the use of variables and complex calculations. It quickly became clear the complexity of the 'dot' approach was not desirable from a user point of view, and we opted for a codelet that placed a spiral on the screen with 'in-place' values that could be easily adjusted to create the desired image. This example demonstrates the types of decisions made during development of the interface and how difficult and important it is to get it right, as the software either succeeds or fails based on these decisions.

From a development point of view, these user stories were instrumental in both helping to reduce user frustration and fix garden-variety software 'bugs'. The narrow line of frustration versus challenge is not always easy to see, and frequently can only be seen through the eyes of a user. Users typically don't have the familiarity with software that a developer does, and this developer familiarity often leads to developer 'blindness'. Examples of software 'bugs' that user experiences reveal can be unexpected, from 'the spiral generates in the wrong direction' to 'the starting angle is off by 90 degrees'. Theses types of things of course, are all a matter of perspective and are likely to come to light through the observations of someone that hasn't been looking at the software for many hours.

The author observing students working with the pCSDTs at the Ayeduasi School, Kumasi Ghana.

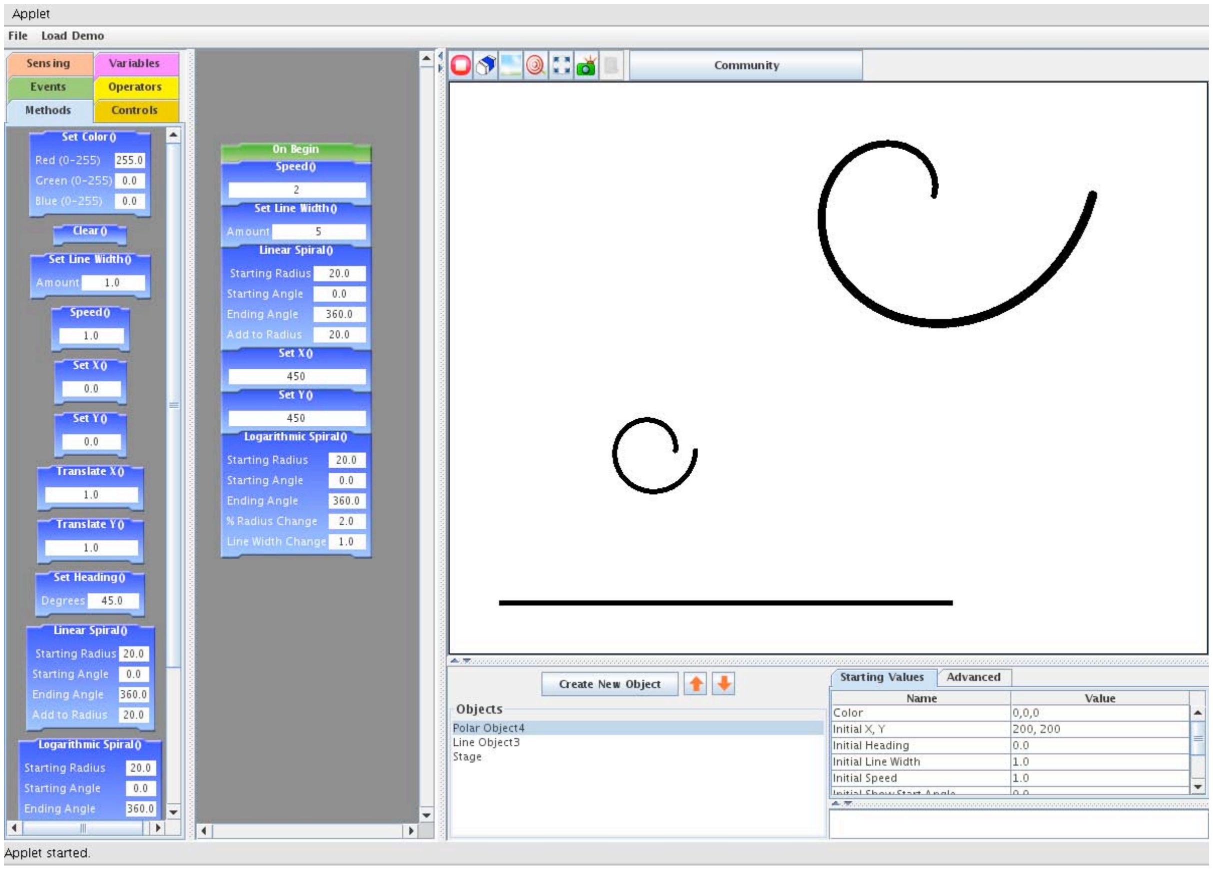
## Mental Models

The programmable Culturally Situated Design Tools were born out of a desire to teach programming skills to pre-college age students. Students would interact with culturally situated objects that are objects with cultural significance, in simulation. This simulation environment would allow for the combination of these basic cultural objects into more complicated designs, in building block fashion. This interaction would be within in a constructivist environment (Mayer, 1996) that would have some desirable artifact as an outcome.

The mental model that the student brings to the pCSDT tool is their idea and understanding about how building blocks should fit together to construct larger objects. They also bring to the tool a preconceived notion about how tools fit in a toolbox and how they can be organized by function. Where most graphical user interfaces in operating systems have their design based on the desktop metaphor, the pCSDTs have as their design base the 'Toolbox' metaphor.

The pCSDT user interface opens and reveals a series of panels organized from left to right. The left most panels contain tabbed panes that map directly to the toolbox metaphor. Each tab in the panel contains building blocks called codelets that are grouped by function. Immediately to the right of the codelet panel is the scripting pane, where the building block codelets are assembled into small programs. The output window then follows to the right of the scripting panel and occupies the majority of the user interface. The output panel is where all the action happens as the user created script is executed. In addition, there are two smaller panels underneath the output pane, which list the objects that have been created in the interface and the initial values (such as location or color) for those objects.

The codelet panel has tabbed panes that are color-coded and labeled 'Event', 'Method', 'Controls', 'Operators', 'Variables', and 'Sensing'. Each panel contains codelets, the same color as the label on the pane that performs particular functions. The group of codelets within each pane all performs



Set Color()
Red (0-255) 255.0
Green (0-255) 0.0
Blue (0-255) 0.0
Clear()
Set Line Width()
Amount 1.0
Speed()
1.0
Set X()
0.0
Set Y()
0.0
Translate X()
1.0
Translate Y()
1.0
Set Heading()
Degrees 45.0
Linear Spiral()
Starting Radius 20.0
Starting Angle 0.0
Ending Angle 360.0
Add to Radius 20.0
Logarithmic Spiral()
Starting Radius 20.0
Starting Angle 0.0
Ending Angle 360.0

On Begin
Speed()
2
Set Line Width()
Amount 5
Linear Spiral()
Starting Radius 20.0
Starting Angle 0.0
Ending Angle 360.0
Add to Radius 20.0
Set X()
450
Set Y()
450
Logarithmic Spiral()
Starting Radius 20.0
Starting Angle 0.0
Ending Angle 360.0
% Radius Change 2.0
Line Width Change 1.0

Polar Object4
Line Object3
Stage

Starting Values | Advanced

| Name | Value |
|---|---|
| Color | 0,0,0 |
| Initial X, Y | 200, 200 |
| Initial Heading | 0.0 |
| Initial Line Width | 1.0 |
| Initial Speed | 1.0 |
| Initial Show Start Angle | 0.0 |

The programmable Adinkra stamping simulation.
The software is running with three shapes on the screen, logarithmic spiral, linear spiral, and horizontal line.

similar functions within the scripting system. Similar to a toolbox, a user accesses the appropriate panel and drags the desired codelets into the scripting pane and attaches the codelet to those already in the panel. By assembling the codelet building blocks in a particular order, the user writes a small program, the result of which will be displayed on the output panel.

The green Events pane contains codelets that are designed to respond to user interface events. For example, the Events pane has a codelet entitled ‘On Begin’ which responds to the User Interface (UI) event of clicking the run button. Once a user clicks on the run button in the UI, the event stack fires and systematically works its way through all of the objects currently created in the system. For each object, the system finds the event ‘On Begin’ and sequentially executes all of the codelets that are attached to it. This systematic path through the event stack creates a loop and updates the attributes of each object and displays these updates in the output pane.

The blue Methods pane contains codelets that are designed to set the attributes programmatically for each object currently created in the system. The method codelets expose these attributes to the user and allows for those attributes to be altered during script execution. As the event stack is processed after the run button is clicked, and as each ‘On Begin’ codelet is located for each object, the methods attached to the ‘On Begin’ codelet are executed in sequence. This execution updates the system values for each attribute in the codelet and then these changes are reflected in the update to the output window.

The orange Controls pane contains codelets that allow the user to incorporate traditional programming concepts such as looping for a fixed number of times, forever, or while a condition holds true as well as conditional execution. These control codelets allow the user to alter the sequence of script execution just as it is possible with a traditional programming language. Here however, the user is altering the onscreen behavior of the object in the output window.

The yellow Operators pane contains codelets that allow the student to use relational operators to create condition statements for the event stack to test for conditional script execution during the execution loop. These relational statements can incorporate system variables such as object attributes listed in the orange Sensing pane or user-defined variables created in the pink Variables pane. The Sensing pane lists object attributes that have their initial values set through the Starting Values panel, and those initial starting values are programmatically changed during script execution through the use of the method codelets.

The desired outcome for the user spending time with a pCSDT is to learn new things in mathematics and computer science. This learning amounts to the extension of their existing mental model to incorporate the tools that they used to construct their onscreen artifact. A successful outcome will be one in which the user extends their mental model of the initial toolbox of the user interface to include the new tools used in building the onscreen output. Based on student interaction with for example, the 'Repeat While' control codelet, they will have developed a mental model that 'If I construct a condition, the codelets placed inside the 'Repeat While' will execute only while that condition remains true, and when that condition becomes false, the changes being executed will stop. I will observe this start and stop on screen in the output window'. The proper graphical feedback then is crucial. It is based on this feedback that the student will succeed in validating their new extended mental model of how these programming concepts work. If the UI responds in an unexpected or incorrect way, the user will fail to validate and thus acquire the new mental model and will not learn the new concepts.

The success or failure of the student in learning programming concepts such as the use of control structures, variables, etc. requires that the designers of the user interface walk a very fine line between design that outright instructs and one that allows for user exploration. This is the creative tension that we as developers need to be careful to balance in this type of educational software. If the tool is too difficult, the student will become frustrated and give up. If the tool is too easy, then the student will become bored and will miss out on potential learning opportunities. An additional strategy in this UI tuning is to find ways to reduce the cognitive load the users face from other interface elements.

## Cognitive task loading

As stated previously, the developer of the tool needs to walk the line between concealing the complexity of the underlying software behind codelets that do more, at the risk of reducing learning opportunities for the user and possibly making the tool boring to users. The alternative is to expose more complexity in the underlying software by designing codelets that do less, at the risk of increasing student frustration with a design tool that has become much more complicated to use. In addition to the level of cognitive complexity that is chosen for the codelets, there are additional ways to reduce cognitive complexity to support user learning.

In the considerations that have been discussed throughout our design efforts for the pCSDTs, the most important has been to support the user in whatever way we can, without diminishing the opportunity for user learning. The user interface design considerations that have helped in this support has been the selective display of information, in particular, the availability of codelets through the use of panes, and by further limiting the display of codelets to those belonging to the currently selected object. This limitation of codelets reduces the cognitive load for the student, which makes available to them, only the current relevant codelets from which they have to chose in constructing their program script.

When the UI starts, it loads a default set of objects and the scripts that have been created for that set of objects. The scripts use codelets that are from the Methods pane for each object. At this point, some small demonstration program is available to be run, getting the users 'feet wet' in how the tool works.

From here, the user can then customize the default script, extending its functionality and thus extending the output results in a desirable fashion in the output pane. With the UI starting with an active object selected, the method codelets available to use for that object, and a small script made from those codelets, the universe of possible scripts has been limited to a much more readily understood starting point. Color-coding for control, method, and event codelets easily allow the user to identify which panel will contain additional similar codelets, for use in extending the script further.

As the user gains familiarity with the scripting panel and the already displayed codelets, the time will come to expand the creation, and the student will create additional objects. Each object is selectable through the object listing and upon selection, relevant codelets will populate the method pane for use in that objects' scripting panel. The user has already gained familiarity with the method codelets and scripting pane of the first object, and so starting work on another object should feel comfortable to the user. This familiarity reduces the cognitive challenge to one of simply understanding the new codelets presented that perhaps were not available in the first object. In addition, familiarity at this level will probably encourage the student to begin exploring the control and operator codelets to create additional, more interesting designs. This progressive disclosure of codelets helps to reduce cognitive load by limiting the interface elements that are visible to the user at any one time, to only those elements that are relevant to the task capable of being performed (Apple, 1992).

Another way the UI assists users through reducing cognitive load is to provide on screen scaffolding in the form of (possibly) familiar tools such as Cartesian grids for object placement in the output screen. When the pCSDT is not in the 'running' mode, a Cartesian grid appears over the output, and all objects are returned to their initial positions. This grid then allows users to more easily see where in the plane their object should be placed based on the ordered pair location possible within the Cartesian grid. When the tool is in 'running' mode, the grid is no longer drawn and thus does not appear in the final output of the artifact.

An interesting way that the UI can be used to create motivation in a student is to provide goal images. True to reducing cognitive load, these images need to be purposely selected before they appear in their own separate window, but once they are available, provide a valuable resource for the student as they create their own design. Sometimes a student does not have sufficient motivation to create a design entirely on his or her own. However, a great deal can be learned through reproducing a design. In addition to the reproduction of the design, there can also be the challenge of reproducing a design efficiently, or more efficiently than a competitor. Team design can result in a deeper understanding of the tool than just a single person struggling through his or her own efforts.

Using tutorials is another more typical way that the UI can be used to assist in the reduction of stress in the cognitive load of learning a new tool. Tutorials can provide small toy scripts that behave in typical ways to get the user started in building their own designs, and can be used to communicate both basic functions as well as more complicated scripting techniques. Tutorials can be especially helpful when the tool has exposed more of the complexity of the underlying software to the user, thus helping to build familiarity with the program. Seeing how those codelets' function within a tutorial can reduce user stress and help users learn the codelets abilities more quickly and effectively.

## Future Work

The Culturally Situated Design Tools have come a long way from where they began as small tools for teaching mathematics. However, there are many important areas that can still be explored, and I think that HCI can help to show the way. Some of the most exciting areas for the future development of CSDTs would be the use of participatory design, tutorial automation, and the creation of an online community. In addition to these three areas of development, I think that the tools could also benefit from a more rigorous approach to usability testing, incorporating HCI formalisms in task analysis and user feedback.

A very exciting approach to design and development of future tools could incorporate participatory design, where users interact with developers on a regular basis resulting in the development direction being set by the users (Muller, 2008). User participation could be very helpful in the development process, because it would then have a group of users with a vested interest in the selection and outcome of design decisions. This highly interested and invested user group would bring many assets to the development process. A developer would immediately benefit from understanding these users mental models of the software that would inform the design process. In addition, the tool would have a ready and willing beta testing community.

I have had the good fortune of being able to interact with some wonderful crafts people while developing the Adinkra stamping simulation. I have been able to collect ethnographic information about Adinkra in Ghana, return home to the US and work on the software application, and then present a version of that tool back to the craftsperson whose help was so invaluable. Gabriel (Adinkra stamping) was able to further my understanding by pointing out subtle differences in my simulation to his practice of the craft. However, that back and forth took two years to accomplish and it still strikes me as how much better the process would have been if I could share my development iterations in something closer to real time.

We could also extend this model of participatory design to include student users and interested young people (Bruckman et. al, 2008). In my opinion, incorporating young students in the software design, as design partners would be a complete paradigm shift as far as the current CSDT development process is concerned, but it remains a great source of excitement for possible future directions. Allowing for such partnerships necessarily means surrender of some control over some design decisions, which might be scary at first, but could yield worthwhile results.

Tutorials offer a way to create a sense of familiarity in users that are new to the software. This sense of familiarity equates to the extension of a users mental model, to include whatever new ideas that are involved (Sutcliffe, 2008). Current CSDT tutorials are somewhat static in their design and can be limited to the presentation of a demonstration script for the user to study, or else a goal image for the user to attempt to duplicate. This limited approach could be so much more exciting to the user if it offered automation and interactivity, allowing for user direction of the outcome on the screen.

An exciting alternative might be to automate the 'getting familiar' tutorial experience through the use of JavaScript. For example, a tutorial page might open and the JavaScript could simultaneously access the underlying pCSDT application in the background. As the user explores the tutorial and makes changes in the simulation environment, those changes could respond in real time through the use of the JavaScript code. Rather than simply exploring the pCSDT application through the use of its scripting interface, JavaScript would allow for this exploration to be more guided and smaller in scope. The user could be more closely guided to build on each small success in their understanding of the interface, rather than rely on just the visual feedback produced when clicking the 'run' button.

This new framework could break down the challenge of building a script in to bite sized chunks, with immediate feedback on success or failure. This framework could also be extended to 'game-ify' the learning process by creating challenges through the use of scenarios (Rosson, Carroll, 2008) or time limited tasks. Advances in effective tutorial design could really improve learning outcomes.

Finally, as the user base for the Culturally Situated Design Tools grows, incorporating an online community seems to be the next step in the evolutionary process for the tools. An online community would provide a sense of social presence to software users, allowing them to interact with other users of the tools (Zaphiris et. al, 2008). This community would offer a venue for users to showcase the designs that they have created in the form of both jpeg picture files and user programs created within the pCSDT tools. The picture files present the outcome of running the program that the user has created and the user programs would allow other users to download the programs that created the pictures. Both the jpeg pictures and the programs would offer the opportunity for users to give and receive feedback on their creations.

In addition to social presence and feedback, an online community would allow for the development of a learning community based around the use of the tools. This learning community can become a knowledge base of support to users of the tools as they seek to become more proficient in their programming skills. This learning community could also be a source of ideas for software improvement and extension in the wider development effort.

Regardless of the specific directions that the development of the CSDTs take in the future, HCI offers a wealth of formalisms in task analysis (Courage et. al, 2008) and user feedback (Kuniavsky, 2008) that would help to inform development and design decisions. Formal task analysis would seek to define user tasks within the software and then seek to determine the best way to perform those tasks. Different strategies for task completion could then be developed and presented to users to determine which strategy was more intuitive or proved less frustrating to users. Formal methods of user feedback could be built around ongoing user surveys, perhaps delivered through the software interface or else the software website. Gathering feedback on an ongoing basis would provide a growing body of user opinions for statistical analysis, rather than gathering information from user around a particular question at a particular point in time.

## Conclusion

My analysis of the programmable Culturally Situated Design Tools from an HCI point of view has focused on ethnographic design, mental models and cognitive task loading. These three areas reflect only a small subset of possible analysis offered by HCI thinking. The ethnographic design principles used with the pCSDTs reflect a two track approach, the first being with ethnographic stories concerning the craft being simulated and the second being user stories with the simulation. Mental models offer a way of understanding how users think of the software being used and how the software can help to extend the users mental models, which we otherwise would describe as learning. Finally, cognitive task loading helps us think about how the complexity of the software presented to a user can either help or hinder learning outcomes.

I look forward to continuing my work with the Culturally Situated Design Tools in the future. I feel privileged to have been involved in this project. The CSDTs offer a rich medium to study both software design and the underlying crafts that the software seeks to simulate. They can focus both on the user and user-learning outcomes as well as crafts people and their crafts. In addition, they offer a rich medium to understand developers and the development process. Thus, the CSDTs are limited only to the extent of our imaginations.

## References

Apple. *Macintosh Human Interface Guidelines*. Addison-Wesley Publishing Company, 1992.

Ashman, A. H., Brailsford, T., Burnett, G., Goulding, J., Moore, A., Stewart, C., & Truran, M. (2008). HCI and the web. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 559-571). New York: Lawrence Erlbaum Assoc.

Blomberg, J., & Burrell, M., Guest, Greg (2003). An ethnographic approach to design. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 964-986). New York: Lawrence Erlbaum Assoc.

Bruckman, A., Bandlow, A., & Forte, A. (2008). HCI for kids. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 793-806). New York: Lawrence Erlbaum Assoc.

Byrne, S. J. (2008). Cognitive architecture. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 94-111). New York: Lawrence Erlbaum Assoc.

Cooper, J., & Kugler, M. B. (2008). The digital divide: The role of gender in human computer interaction. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 763-773). New York: Lawrence Erlbaum Assoc.

Courage, C., Redish, J., Wixon, D. (2008). Task analysis. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 927-934). New York: Lawrence Erlbaum Assoc.

Dumas, J. S., & Fox, J. E. (2008). Usability testing: Current practice and future directions. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 1129-1143). New York: Lawrence Erlbaum Assoc.

Holtzblatt, K. (2008). Contextual design. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 949-963). New York: Lawrence Erlbaum Assoc.

Kuniavsky, M. (2008). User experience and HCI. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 897-915). New York: Lawrence Erlbaum Assoc.

Lazzaro, N. (2008). Why we play: Affect and the fun of games, designing emotions for games, entertainment interfaces and interactive products. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 679-701). New York: Lawrence Erlbaum Assoc.

Muller, M. J. (2008). Participatory design: The third space in HCI. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 1061-1077). New York: Lawrence Erlbaum Assoc.

Marcus, M. A. (2008). Global/intercultural user interface design. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 355-377). New York: Lawrence Erlbaum Assoc.

Pagulayan, R. J., Keeker, K., Fuller, T., Wixon, D., & Romero, R. L. (2008). User centered design in games. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 741-758). New York: Lawrence Erlbaum Assoc.

Payne, S. J. (2008). Mental models in human-computer interaction. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 64-74). New York: Lawrence Erlbaum Assoc.

Rosson, Mary Beth, Carroll, John M. (2008). Scenario Based Design. In A. Sears & J. A.

Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 1041-1057). New York: Lawrence Erlbaum Assoc.

Sutcliffe, A. (2008). Multimedia User Interface Design. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 393-411). New York: Lawrence Erlbaum Assoc.

Watzman, W. J., & Re, M. (2008). Visual design principles for usable interfaces everything is designed: Why we should think before doing. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 329-340). New York: Lawrence Erlbaum Assoc.

Zaphiris, P., Ang, C. S., Laghos, A. (2008). Online communities. In A. Sears & J. A. Jacko (Eds.), *The human-computer interaction handbook : Fundamentals, evolving technologies, and emerging applications* (pp. 603-618). New York: Lawrence Erlbaum Assoc.