Six Learning Barriers in
End-User Programming Systems

Andrew J. Ko, Brad A. Myers, and Htet Htet Aung

Project Marmalade
Human-Computer Interaction Institute
Carnegie Mellon University

My name is Andrew Ko, and today I'll be talking about some work that I did with Htet Htet Aung and my advisor Brad Myers.

Sarah, Bob, John on content:

Bob: Worse than last year, too superficial, not useful. Why categorize? Was the number 6 useful, or was it what I found? Implications for design: copy and paste, then build a tool to help it.

Sarah: Needs to be more systematic to come up with categories. Factor analysis! What about independence? Objectify the categories.

Both: Too high level.

Wasn’t clear what the process was that produced the design recommendations. Hunch? Strategy? Design principle? Where did the solutions come from?

Sarah: no links to literature. To what extent have these barriers been seen before? Not very scholarly.

Mark: Really liked it. Mark sees ways to apply it.

Brad: what’s missing? Process timeline. Motivation for environments, study of what people are doing now, a bag of results. How do we turn those into design guidelines? What are the problems? Come up with categories. Just to make sure the categories are reasonable, we had a colleague do it. This inspired new ideas. Now, the audience can use these categories for their design tool. Helps you come up with solutions that wouldn’t have been though of otherwise.

Brad: people liked the context slide in the beginning. The future work slide was good too.
Project Marmalade

- Trying to make programming easier to do by designing new programming systems.
- Designed the Whyline to help with debugging difficulties.
- What other barriers programming systems pose?
- What should be designed to remove these barriers?
Rising Needs and Rising Barriers

- By 2012, 30% of new jobs may require programming skills! (US Department of Labor).
- But learning to program has only become more complex.

In 1991...
1. Type qbasic
2. Type PRINT “Hello World!”
3. Type F5.

In 2004...
1. Choose File > New > Project...
2. Select Visual Basic Project.
4. Drag a button onto the form.
5. Name the button.
6. Double-click on the button.
7. Choose Debug > Start.

And most aren’t professionals.

When I started in 1991...

In 2004... every single one of these is a barrier to learning to program.
And we haven’t even started to program yet!

What are these barriers and how do they impact learners’ progress?

And how can they help us design better programming systems?
Introduction

First, I'll briefly discuss what I mean by “Learning Barriers” and describe the design of our study.

The rest of the talk will be focused on the six types of barriers that we found, and some ideas on how we might remove them.
What is Known About Barriers?

- Mostly about barriers in programming languages.
  - “Syntax” is an awkward concept and difficult to learn.

- Very little about the rest of a programming system:
  - Editor, debugger, help system, error messages, etc.

- Do these tools pose any barriers?

Do these tools pose any barriers? And if so, what are they, and how do they influence learner’s success?
When approaching barriers, what do learners do?

Blackwell has characterized learners reasoning in terms of cost, risk and reward.
When approaching barriers, what do learners do?

Blackwell has characterized learners reasoning in terms of cost, risk and reward.
When approaching barriers, what do learners do?

Blackwell has characterized learners reasoning in terms of cost, risk and reward.
Learning Barriers

Give up and look for something else...

In this case, the system may be abandoned.

What we want to happen is that the learner tries to overcome the barrier. If they do, one of three things may happen.
In this case, the system may be abandoned.

What we want to happen is that the learner tries to overcome the barrier. If they do, one of three things may happen.
What we want to happen is that learners have the confidence to overcome the barrier.

But this isn’t realistic. What usually happens is that

1) Learners make some invalid assumption and aren’t able to overcome the barrier.

2) Or, they make some invalid assumption which helps them overcome one barrier, but leads to another.

So the focus of this study was ...
What we want to happen is that learners have the confidence to overcome the barrier. But this isn’t realistic. What usually happens is that

1) Learners make some invalid assumption and aren’t able to overcome the barrier.

2) Or, they make some invalid assumption which helps them overcome one barrier, but leads to another.

So the focus of this study was ...
What we want to happen is that learners have the confidence to overcome the barrier. But this isn’t realistic. What usually happens is that

1) Learners make some invalid assumption and aren’t able to overcome the barrier.

2) Or, they make some invalid assumption which helps them overcome one barrier, but leads to another.

So the focus of this study was ...
What we want to happen is that learners have the confidence to overcome the barrier.

But this isn’t realistic. What usually happens is that

1) Learners make some invalid assumption and aren’t able to overcome the barrier.

2) Or, they make some invalid assumption which helps them overcome one barrier, but leads to another.

So the focus of this study was ...
What we want to happen is that learners have the confidence to overcome the barrier.

But this isn’t realistic. What usually happens is that

1) Learners make some invalid assumption and aren’t able to overcome the barrier.

2) Or, they make some invalid assumption which helps them overcome one barrier, but leads to another.

So the focus of this study was ...
What are the Barriers?

- What are the significant barriers in programming systems?
- What types of invalid assumptions do these barriers predispose?
- How can we remove these barriers from programming systems?

To answer these questions, we designed a study of Visual Basic.NET.
A Study of Visual Basic .NET

- Students in a course called Programming Usable Interfaces
- Looking for insurmountable barriers.

Taught here at Carnegie Mellon.

Barriers that learners could not be overcome without help
A Study of Visual Basic.NET


Students in a course called Programming Usable Interfaces

Looking for insurmountable barriers.

Taught here at Carnegie Mellon.

Barriers that learners could not be overcome without help
A Study of Visual Basic.NET

- Students in a course called Programming Usable Interfaces
- Looking for insurmountable barriers.

Taught here at Carnegie Mellon.

Barriers that learners could not be overcome without help
The Method

- Observed learners working on assignments from class.
- If they got “stuck,” they could consult an oracle for guidance.
- When learners sought advice, they were asked to report:
  1) What they were stuck on.
  2) The series of events led to becoming stuck.
  3) What they had tried to “unstick” themselves.
- After soliciting this information, the oracles helped learners overcome the barrier.

(1) making a form that computes an average, (2) designing an alarm clock interface, and (3) designing a copy machine interface.

After 4 weeks of observations...
The Data

- Overall, we sampled 130 barriers from 22 students.
  - 74 were insurmountable.
  - 56 seemed to be overcome, but because learners made invalid assumptions, these led insurmountable barriers of a different type.

- The barriers were analyzed for similarities and common properties, leading to 6 distinct categories of barriers.

- Two individuals independently categorized each barrier, attaining 94% agreement.

...were insurmountable, and could not overcome them without our help.

For the other 56, learners thought they had overcome them, but had actually made invalid assumptions, leading them to insurmountable barriers of a different type.
Six Learning Barriers

- Design
- Use
- Coordination
- Selection
- Understanding
- Information

The order in which learners typically encountered them.

We'll discuss these in the order that learners typically encountered them.
Design Barriers

- Inherently difficult computational problems, independent of the programming interfaces used to solve it.

- Learners could easily conceive of a solution, but their solutions were wrong.

This wrong solutions led to other barriers.

For example,
Design Barriers

- Learners had to sort a list of names alphabetically.

<table>
<thead>
<tr>
<th>How do I alphabetize a list of names?</th>
<th>...maybe I can just keep moving the names until it looks right....</th>
</tr>
</thead>
<tbody>
<tr>
<td>David</td>
<td>David</td>
</tr>
<tr>
<td>Andy</td>
<td>Bryan</td>
</tr>
<tr>
<td>Bryan</td>
<td>Andy</td>
</tr>
<tr>
<td>Chris</td>
<td>Chris</td>
</tr>
</tbody>
</table>

- Also: event concurrency, Boolean logic, and data flow.

Other design barriers we observed involved...

So how can we remove design barriers from programming systems?
Removing Design Barriers

- Learners did not have the time or interest in solving these conceptually difficult problems.
- Instead, provide solutions to these hard problems.
- **The challenge**: make solutions *discoverable, usable* and *customizable*.

For example, Visual Basic was very helpful when learners found prepackaged sorting algorithms that they could use.

The next type of barrier that learners typically encountered is what we call selection barriers.
Selection Barriers

- Aspects of a programming system’s support for discovering
  (1) What programming interfaces are available, and
  (2) Which can be used to achieve a particular behavior.

  *What programming interfaces can I choose from?*

  *“Which ones can do X . . .?”*

- Learners typically overcame selection barriers by finding example code.
Selection Barriers

- Keeping time for an alarm clock.

Also: text formatting, arithmetic, image drawing

What can help me keep time?

When learners asked Visual Basic about time, it answered: “You can keep time with one of these 500 useful tools!”

We also observed selection barriers with text formatting, arithmetic, and image drawing.

So how can we remove selection barriers from programming systems?
Selection Barriers

- Keeping time for an alarm clock.

- Also: text formatting, arithmetic, image drawing

When learners asked Visual Basic about time, it answered:
“You can keep time with one of these 500 useful tools!”

We also observed selection barriers with text formatting, arithmetic, and image drawing.

So how can we remove selection barriers from programming systems?
Selection Barriers

- Keeping time for an alarm clock.

  Which one should I use???

- Also: text formatting, arithmetic, image drawing

Introduction • The Study • Six Types of Barriers • Conclusions

When learners asked Visual Basic about time, it answered:
“You can keep time with one of these 500 useful tools!”

We also observed selection barriers with text formatting, arithmetic, and image drawing.

So how can we remove selection barriers from programming systems?
Removing Selection Barriers

- It should not be easier to go to Google and search 4.5 billion pages than to search the *MSDN library* (but it was).

- Group programming interfaces by what they do, what they *depend on*, and what they can *interact with*.
  - Provide examples that learners can use directly in their code
  - Provide support for adapting the examples to their code

- **The challenge:** do the above for potentially thousands of programming interfaces in a programming system.

Not by how they’re named or implemented.

The next barrier that learners typically encountered were use barriers.
Use Barriers 28% of barriers observed

- Properties of a programming interface that obscure
  - What actions can be performed on it,
  - How to perform these actions, and
  - What effect these actions they will have.

- Direct manipulation interfaces tended to be surmountable

Programming interfaces are things like for loops and conditional statements, as well buttons, menus, and other programmable objects.
Array declarations did not support direct manipulation.

Can I put numbers in an array declaration?

Can I put them in like this?

If I only put in four numbers, what are the other 16 going to be?

Dim numbers(20) as Integer = { 5, 10, count, 20 }

Also: using widgets, methods, and data structures.

So how can we remove use barriers from programming systems?
Removing Use Barriers

- Design *direct manipulation* programming interfaces.
  - What actions are possible on an array declaration? *Provide a menu.*
  - How can those actions be performed? *Click on the menu.*
  - What happens to the array if only four are initialized? *Show its contents.*

- **The challenge**: design a flexible and supportive direct manipulation code editor.

---

I’ll be talking about some work we’ve done on this at my student talk at 11 tomorrow.
The invisible rules that govern how programming interfaces can be combined.

Learners could guess how to coordinate the pieces, but their guesses were typically wrong.

In these situations, programming interfaces were like jigsaw puzzles. Learners could see the pieces, but didn’t know how to put them together.

For example,
Coordination Barriers

- “Pulling” data from a window that has not been shown causes null pointer exceptions.
- Also: language constructs, using data of different types, etc.

So how can we remove coordination barriers from programming systems?
Removing Coordination Barriers

- Make the *invisible* rules *visible*:
  - Provide *examples* to illustrate the rules while learners are selecting the programming interfaces to use.
  - *Enforce* and *explain* the rules through direct manipulation.

- **The challenge**: what exactly are the rules in a programming system?

“That can’t go here; it has to go there.”
“You can put X and Y together in these three ways...”

Have to know this in order to make them visible.
Understanding Barriers

- Properties of a program’s behavior that obscure what a program did or did not do.

I thought doing $X$ ... The Program ... would cause $Y$, but it caused $Z$ instead. Why?

- What if $X$ did cause $Y$? What if it caused $Y$ and $Z$? What if $X$ never happened? What if $Z$ is $Y$?

The big problem was that $X$ caused $Z$. But, there are more subtle issues.
Understanding Barriers

- Why isn’t the clock updating?

Is the timer *added*? Is the timer’s *enabled* property set to true? Is anything using the timer’s value to update the label?

Is the enabled property?
This is something they couldn’t possibly know about.

So how can we remove understanding barriers from programming systems?
Removing Understanding Barriers

- **The challenge**: help learners think of questions to ask and reveal any invalid assumptions.

Instead of having users think of these questions themselves, why not provide the questions for them?
Removing Understanding Barriers

- **The challenge**: help learners think of questions to ask and reveal any invalid assumptions.

Instead of having users think of these questions themselves, why not provide the questions for them?
**Information Barriers**

- Obstacles in getting information about how a program works on the inside.
- Learners could not find tools to answer their questions.
- Learners could not use the tools that they did find.

**Six Types of Barriers**

- Obstacles in getting information about how a program works on the inside.
- Learners could not find tools to answer their questions.
- Learners could not use the tools that they did find.
Information Barriers

- Programmers often made infinitely recursive method calls, leading to stack overflow exceptions.
- Most learners pressed help.
- Pressing break pointed learners to the top of their program instead of to the source of the exception.

Programmers had very few options when answering these questions. Here, you can see that the learners’ choices are “break” “continue” and “help”. Little do they know, clicking break will point them directly to where the exception occurred. But which do you think they click?

So how can we remove information barriers from programming systems?
Removing Information Barriers

- Instead of having learners collect the information, have the computer collect it.

- The challenge: design interfaces that automatically provide task-relevant information.

This method calls itself over and over again.

```java
MyMethod(int number1) {
    MyMethod(2)
}
```

That’s what it’s good at.

Already made progress with this with the Whyline. By allowing programmers to ask questions about their program’s output, the system can collect the information it needs to answer their question. This way, learners don’t have to collect it themselves.
Future Work

- This study has revealed several things that we would not have known otherwise.

- Currently using the six barriers to design a new Java programming system.
  - Flexible and supportive direct manipulation programming interfaces.
  - Support for copying and adapting example code.
  - Support for What can? questions to help find programming interfaces.
Questions?

Thanks to Brad Myers, Htet Htet Aung, and the students of PUI for their help with this research.

Who developed categories and how were they coded?
Categories developed, then assigned for each barrier.

But the 4.5 billion weren’t all relevant, and there are communities for these things.
But it takes a long time to find those communities and the examples were often incorrect, or inapplicable.

Aren’t these barriers at the wrong level of abstraction to support design?
For fine-grained design, yes, but these were intended to guide the choice of what tools to design, not how to design them.

How will you know if your environment is better?
Mimic current interfaces and compare errors and time.
Questions?

http://www.cs.cmu.edu/~NatProg/marmalade.html

Thanks to Brad Myers, Htet Htet Aung, and the students of PUI for their help with this research.

Who developed categories and how were they coded?
   Categories developed, then assigned for each barrier.

But the 4.5 billion weren’t all relevant, and there are communities for these things.
   But it takes a long time to find those communities and the examples were often incorrect, or inapplicable.

Aren’t these barriers at the wrong level of abstraction to support design?
   For fine-grained design, yes, but these were intended to guide the choice of what tools to design, not how to design them.

How will you know if your environment is better?
   Mimic current interfaces and compare errors and time.