A Model of Programming Errors

Andrew J. Ko

Brad A. Myers
Natural Programming

- Working on how HCI can help design programming tools to prevent, detect, and help repair programming errors.

- We lacked some knowledge about errors that would help us design:
  - What kinds of errors do programmers make?
  - Which errors are most common?
  - What causes these common errors?
Some Frightening Facts

Study commissioned by NIST (2002) of 14 software vendors:

- Software errors cost ~$60 billion annually
- Software engineers spend 70-80% of development time testing and debugging
- Time for 1 developer to fix 1 bug was ~17.4 hours
- Software engineers believed that ‘more useful’ tools for finding and fixing errors would be most helpful
How can we reduce the cost of programming errors?

Build better tools!
How can we reduce the cost of programming errors?

Design

Build better tools!
A survey of taxonomies of programming errors

- From 1975 to the present, more than 35 taxonomies of programming errors have been published.
- That’s more than 500 categories of errors!
A survey of taxonomies of programming errors

- From 1975 to the present, more than 35 taxonomies of programming errors have been published.
- That’s more than 500 categories of errors!

Lots of design knowledge

Difficult to use
What would a good model of programming errors offer?

- Characterization of programming activities
- Characterization of particular errors
- Support for...
  - Characterizing the distribution of errors in an activity
  - Discriminating between particular errors
  - Predicting the causes of specific errors
Blackwell’s Attention Investment Model

Attentional strain, due to a lack of direct manipulation, leads to programming errors.
Blackwell’s Attention Investment Model

Attentional strain, due to a lack of direct manipulation, leads to programming errors

+ **Predictive**: attentional problems → errors
Blackwell’s Attention Investment Model

Attentional strain, due to a lack of direct manipulation, leads to programming errors

+ **Predictive**: attentional problems $\rightarrow$ errors

+ Helps characterize programming strategies in terms of cost, risk, pay-off, investment
Blackwell’s Attention Investment Model

Attentional strain, due to a lack of direct manipulation, leads to programming errors

**Predictive**: attentional problems → errors

+ Helps characterize programming strategies in terms of cost, risk, pay-off, investment

– Doesn’t **characterize** or **discriminate** between errors
Green’s Cognitive Dimensions

Usability problems cause programming errors.
Green’s Cognitive Dimensions

Usability problems cause programming errors.

+ **Predictive**: usability problems $\rightarrow$ errors
Green’s Cognitive Dimensions

Usability problems cause programming errors.

+ **Predictive**: usability problems $\rightarrow$ errors

+ Helps characterize information artifacts in terms of interactive dimensions
Green’s Cognitive Dimensions

Usability problems cause programming errors.

+ **Predictive**: usability problems $\rightarrow$ errors

+ Helps characterize information artifacts in terms of interactive dimensions
  – Doesn’t help **characterize** or **discriminate** between errors
Reason’s Latent Failure Model of Human Error

Knowledge, strategic, and attentional breakdowns predispose error.
Reason’s Latent Failure Model of Human Error

Knowledge, strategic, and attentional breakdowns predispose error.
Reason’s Latent Failure Model of Human Error

Knowledge, strategic, and attentional breakdowns predispose error.

+ **Predictive**: breakdowns $\rightarrow$ errors
Reason’s Latent Failure Model of Human Error

Knowledge, strategic, and attentional breakdowns predispose error.

+ **Predictive**: breakdowns → errors

+ Helps characterize cognitive breakdowns in terms of human’s innate cognitive limitations
What’s missing?

The ability to characterize and discriminate between programming errors by interface, cognitive problem, and strategy.
What’s missing?

The ability to characterize and discriminate between programming errors by **interface**, **cognitive problem**, and **strategy**.
What’s missing?

The ability to characterize and discriminate between programming errors by **interface**, **cognitive problem**, and **strategy**.

Green
Blackwell
Reason

Survey of error categories
What’s missing?

The ability to characterize and discriminate between programming errors by interface, cognitive problem, and strategy.

\[ \text{Green Blackwell Reason} + \text{Survey of error categories} = \text{Model of programming errors} \]
Sources of Error

– Reason's cognitive breakdowns (1990)
Sources of Error

- Reason’s cognitive breakdowns (1990)

Knowledge Breakdowns

i.e., Heuristic or oversimplified knowledge
Sources of Error

– Reason’s cognitive breakdowns (1990)

Knowledge Breakdowns

i.e., Heuristic or oversimplified knowledge

Attentional Breakdowns

i.e., fixation or working memory strain
Sources of Error

- Reason’s cognitive breakdowns (1990)

Knowledge Breakdowns

i.e., Heuristic or oversimplified knowledge

Attentional Breakdowns

i.e., fixation or working memory strain

Strategic Breakdowns

i.e., unforeseen interactions or goal conflicts
Errors, Faults and Failures
Errors, Faults and Failures

A programmer’s error

```java
if( a && b || c ) {
    // Code here
}
```
Errors, Faults and Failures

A runtime fault

A programmer’s error

if( a && b || c ) {
    // Code block
}
Errors, Faults and Failures

A program’s failure

A runtime fault

A programmer’s error

if( a && b || c ) {

}
Representing a Breakdown

Activity
Representing a Breakdown

**Activity**

- **Problem**
  - *from Reason*

- **Action**
  - *from survey*

- **Artifact**
  - *from survey*
Breakdowns in Specification Activities

- Creating
- Understanding
- Modifying
Breakdowns in Specification Activities

- Creating
- Understanding
- Modifying

○ Documented Specification
○ Mental Specification
Breakdowns in Specification Activities

- Knowledge Problem
- Attentional Problem
- Strategic Problem

Creating
Understanding
Modifying

Documented Specification
Mental Specification
Breakdowns in Implementation Activities

- Perceiving
- Understanding
- Implementing
- Modifying
- Reusing
Breakdowns in Implementation Activities

- Perceiving
- Understanding
- Implementing
- Modifying
- Reusing

- Environment
- Artifact
- Library
- Algorithm
- Data Structure
- Language
- Construct
- Style-Specific
- Artifact
Breakdowns in Implementation Activities

- Knowledge Problem: Perceiving, Understanding, Implementing, Reusing
- Attentional Problem: Environment, Artifact, Library, Algorithm, Data Structure, Language, Construct, Style-Specific Artifact
- Strategic Problem: Environment, Artifact, Library, Algorithm, Data Structure, Language, Construct, Style-Specific Artifact
Breakdowns in Debugging Activities

- Perceiving
- Understanding
- Determining
Breakdowns in Debugging Activities

- Perceiving
- Understanding
- Determining

Failure
Fault
Error

Andrew Ko

Introduction • Related Work • The Model • Study • Summary and Future Work
Breakdowns in Debugging Activities

- Knowledge Problem
- Attentional Problem
- Strategic Problem

- Perceiving
- Understanding
- Determining

- Failure
- Fault
- Error
Integrating prediction

Breakdowns in Specification Activities
- Knowledge Problem
  - Creating
- Attentional Problem
  - Understanding
  - Modifying
- Strategic Problem

Breakdowns in Implementation Activities
- Knowledge Problem
  - Perceiving
  - Understanding
- Attentional Problem
  - Implementing
- Strategic Problem
  - Modifying
  - Reusing
Integrating prediction
Integrating prediction

Breakdowns in Specification Activities
- Knowledge Problem
  - Creating
- Attentional Problem
  - Understanding
  - Modifying
- Strategic Problem

Breakdowns in Implementation Activities
- Knowledge Problem
  - Perceiving
  - Understanding
- Attentional Problem
  - Implementing
  - Modifying
  - Reusing
- Strategic Problem
  - Environment Artifact
  - Library
  - Algorithm
  - Data Structure
  - Language Construct
  - Style-Specific Artifact
Integrating prediction

Breakdowns in Specification Activities

- Knowledge Problem: Creating
- Attentional Problem: Understanding, Modifying
- Strategic Problem: Documented Specification, Mental Specification

Breakdowns in Implementation Activities

- Knowledge Problem: Perceiving, Understanding
- Attentional Problem: Implementing
- Strategic Problem: Environment Artifact, Library, Algorithm, Data Structure, Language Construct, Style-Specific Artifact
Integrating prediction
Breakdowns in Implementation Activities

Knowledge Problem
- Perceiving
- Understanding
- Implementing
- Modifying
- Reusing

Attentional Problem
- Implementing

Strategic Problem
- Modifying

Breakdowns in Debugging Activities

Knowledge Problem
- Perceiving

Attentional Problem
- Understanding

Strategic Problem
- Determining

Errors → Faults → Failures
Introduction

Related Work

The Model

Study

Summary and Future Work


Andrew Ko

Introduction ● Related Work ● The Model ● Study ● Summary and Future Work
Introduction  ●  Related Work  ●  The Model  ●  Study  ●  Summary and Future Work
Andrew Ko

Introduction ● Related Work ● **The Model** ● Study ● Summary and Future Work

Data Structure
Language
Construct
Perceiving
Modifying
Reusing
Implementing

Attentional Problem
Knowledge Problem
Strategic Problem

Breakdowns in Implementation Activities

Environment Artifact
Library
Algorithm
Data Structure
Language Construct
Style-Specific Artifact

Errors
Faults
Failures

Breakdowns in Debugging Activities

Knowledge Problem
Attentional Problem
Strategic Problem

Perceiving
Understanding
Implementing
Modifying
Reusing

Failure
Fault
Error

Determining

Faults

Perceiving
Understanding
Determining

Mental Specification
Documented Specification
Creating
Understanding
Strategic Problem
Attentional Problem
Knowledge Problem
Modifying
The Model

Breakdowns in Implementation Activities

Knowledge Problem

Attentional Problem

Strategic Problem

Perceiving

Understanding

Implementing

Modifying

Reusing

Implementing

Breakdowns in Debugging Activities

Knowledge Problem

Attentional Problem

Strategic Problem

Perceiving

Understanding

Determining

Failure

Fault

Error

The Model
Introduction • Related Work • The Model • Study • Summary and Future Work
Introduction ● Related Work ● The Model ● Study ● Summary and Future Work

Andrew Ko
The Model in Action

A breakdown chain from a study.
The Model in Action

A breakdown chain from a study.
The Model in Action

- Attentional Problem
- Creating
- Specifications for the Boolean logic
The Model in Action

if(!(dot1.isShowing && dot2.isShowing && dot3.isShowing && dot4.isShowing))...
The Model in Action

if(!(dot1.isShowing && dot2.isShowing && dot3.isShowing && dot4.isShowing))...
The Model in Action

\[
\text{if}(!(\text{dot1.isShowing} && \text{dot2.isShowing} && \text{dot3.isShowing} && \text{dot4.isShowing}))...\]

Attentional Problem

Creating

Specifications for the Boolean logic

Strategic Problem

Implementing

Boolean logic for seeing if all dots are gone

if not (dot1.isShowing and dot2.isShowing...)

Conditional becomes true after one dot is eaten

Missing reference to BigDot
The Model in Action

if(!(dot1.isShowing && dot2.isShowing && dot3.isShowing && dot4.isShowing))...

Conditional becomes true after one dot is eaten

Pac-Man bounces before all dots are gone

Attentional Problem

Creating Specifications for the Boolean logic

Strategic Problem

Implementing Boolean logic for seeing if all dots are gone

if not (dot1.isShowing and dot2.isShowing...)

Missing reference to BigDot

if not (dot1.isShowing and dot2.isShowing...)

Pac-Man bounces before all dots are gone

if not (dot1.isShowing and dot2.isShowing...)
Claims

- Concisely describes and discriminates between errors
  - 135 breakdown categories
  - Describes all error categories from previous taxonomies

- Helps characterize specific programming activities

- Has a predictive ability, like the other models
Research Questions

- Does model help discriminate between particular types of errors?
- Does it help characterize specific activities’ error-proneness?
Method

- Wanted to find the most common breakdowns in the Alice programming system
- Studied 7 programmers with expertise ranging from Visual Basic to multiple languages
  - (There was no evidence that novices or experts would differ)
  - 4 worked on a simple Pac-Man game, 3 used Alice in a class
- Think-aloud and contextual inquiry used to track goals, strategies, intents
Data Extraction

- ~900 minutes of videotape were analyzed for
  - Tasks and task goals
  - Strategies for achieving goals
  - Outcome of each strategy used
  - Errors, faults, failures

- Each action was analyzed for breakdowns
  - Cognitive problems determined from think-aloud data
  - Only recorded breakdowns that led to failure

- Chains of breakdowns constructed by building a goal hierarchy, using causal links from model
The Dataset

All subjects’ data pooled together

- 159 breakdowns

- 102 errors

- No differences between experts and novices
Discriminability

- Including only cognitive problems and actions, 33 breakdown categories to observe
  - 17 of 33 were observed
- The task did not involve creating or modifying specifications, removing 6 categories
- Thus, we observed multiple instances of 17 of the 27 breakdowns possible in the task
Discriminability

- Including only cognitive problems and actions, 33 breakdown categories to observe
  - 17 of 33 were observed
- The task did not involve creating or modifying specifications, removing 6 categories
- Thus, we observed multiple instances of 17 of the 27 breakdowns possible in the task
Discriminability

- Including only cognitive problems and actions, 33 breakdown categories to observe
  - 17 of 33 were observed
- The task did not involve creating or modifying specifications, removing 6 categories
- Thus, we observed multiple instances of 17 of the 27 breakdowns possible in the task
Most Common Breakdowns
Introduction • Related Work • The Model • Study • Summary and Future Work
Most Common Breakdowns

- Strategic - Modifying: 11%
- Strategic - Implementing: 15%
- Other: 36%

Andrew Ko

Introduction • Related Work • The Model • Study • Summary and Future Work
Most Common Breakdowns

- Strategic - Implementing: 15%
- Strategic - Modifying: 11%
- Knowledge - Implementing: 11%
- Other: 36%
Most Common Breakdowns

- Knowledge - Determining: 9%
- Knowledge - Understanding: 11%
- Knowledge - Implementing: 11%
- Strategic - Modifying: 11%
- Strategic - Implementing: 15%
- Other: 36%
**Most Common Breakdowns**

- **Debugging Breakdowns**
  - Knowledge - Determining: 7%
  - Knowledge - Understanding: 11%
  - Knowledge - Implementing: 11%
  - Strategic - Modifying: 11%
  - Strategic - Implementing: 15%
- **Implementation Breakdowns**
  - Other: 36%

Andrew Ko

Human-Computer Interaction Institute

Introduction • Related Work • The Model • Study • Summary and Future Work
Andrew Ko

Introduction ● Related Work ● The Model ● Study ● Summary and Future Work

Most Common Breakdowns

- Debugging Breakdowns
  - Knowledge - Determining: 7%
  - Knowledge - Understanding: 11%
  - Strategic - Modifying: 11%
  - Strategic - Implementing: 15%
  - Other: 36%

- Implementation Breakdowns
  - Knowledge - Implementing: 11%

Caused 50% of Study
Study Summary

- Alice is prone to strategic breakdowns in implementing possibly because of its structured editing interface
- Alice is prone to knowledge and attentional breakdowns in determining faults and errors, which leads to strategic breakdowns in repairing code
- These findings will help guide our design of breakdown-preventing programming and debugging tools
Summary

- The model helps discriminate between particular breakdowns and errors, and their causes.
- We can use it with other design tools, such as Cognitive Dimensions and the Attention Investment Model to help design more usable programming systems.
Summary

- The model helps discriminate between particular breakdowns and errors, and their causes.
- We can use it with other design tools, such as Cognitive Dimensions and the Attention Investment Model to help design more usable programming systems.
Summary

- The model helps discriminate between particular breakdowns and errors, and their causes.
- We can use it with other design tools, such as Cognitive Dimensions and the Attention Investment Model to help design more usable programming systems.
The Whyline
A debugging tool to prevent strategic, attentional, knowledge breakdowns in determining faults and errors.
The Whyline
A debugging tool to prevent strategic, attentional, knowledge breakdowns in determining faults and errors.
Thanks!