How do we know if we designed the “right” tools?
How do we know if our tools have practical benefits for their users?
Purpose of Evaluation

• **Formative:** to inform the design

• **Summative:** to characterize a design, once it is complete
Genres of Evaluation

**Automated** Usability measures computed by software

**Inspection** Based on skills, and experience of evaluators

**Formal** Models and formulas to calculate measures

**Empirical** Usability assessed by testing with real users
Genres of Evaluation

**Automated** Usability measures computed by software

**Inspection** Based on skills, and experience of evaluators

**Formal** Models and formulas to calculate measures

**Empirical** Usability assessed by testing with real users
Formative Inspection: Heuristic Evaluation

- *Rules of thumb* describing features of usable systems
- Small set (3-5) of evaluators (experts) examine interface (or proposal) for compliance with heuristics
Genres of Evaluation

**Automated** Usability measures computed by software

**Inspection** Based on skills, and experience of evaluators

**Formal** Models and formulas to calculate measures

**Empirical** Usability assessed by testing with real users
Genres of Evaluation

**Automated** Usability measures computed by software

**Inspection** Based on skills, and experience of evaluators

**Formal** Models and formulas to calculate measures

**Empirical** Usability assessed by testing with real users
Empirical: Quantitative Studies

• Goal
  • Use to reliably measure some aspect of interface
  • Compare two or more designs on a measurable aspect

• Approaches
  • Collect and analyze user events that occur in natural use
  • Controlled experiments

• Examples of measures
  • Time to complete a task, Average number of errors on a task, Users’ ratings of an interface
Empirical: Qualitative Data

• Example: observe users, use talk-aloud protocol, surveys

• Qualitative methods help:
  • Understand what is going on
  • Look for problems
  • Roughly evaluate usability of the interface

• Can be very valuable to convey experience!
Practical Challenges

• Order-of-magnitude differences in programming productivity

• Difficult comparisons:
  • Lopsided comparison of research code to tools with decades of development
  • What if your tool enables entirely new tasks?
Practical Challenges

• Most valuable results (e.g., learning, productivity) are hard to operationalize

• Recruiting can be difficult:
  Participating in studies not a “community value” for many programmers
Addressing the Challenges

• Triangulation of different methods to get over limitations of any one evaluation:
  • Sharpen the focus of quantification: ask more specific questions
  • Combine with qualitative observations on open-ended tasks
  • Collect self-report data (weaker support)
Case Studies
Proton: Declarative Multitouch Gestures

Kenrick Kin$^{1,2}$ Björn Hartmann$^1$ Tony DeRose$^2$ Maneesh Agrawala$^1$

$^1$University of California, Berkeley $^2$Pixar Animation Studios
The Gesture Recognition Problem

[Image of various hand gestures with corresponding actions]

[References: Wobbrock '09]
Current Gesture Implementation

```c
鸱esDown(Array *touches, Event *event)
  if(event->allTouches()->count() == 1)
    if(touches[0]->target() != 'n')
      _state = GestureFailed;
  else if(event->allTouches()->count() == 2)
    if(touches[0]->target() != 'm')
      _state = GestureFailed;
  else
    _state = GestureFailed;

.touchesMove(Array *touches, Event *event)
  for(i = 0; i < touches->count(); i++)
    if(touches[i]->touchId() == 0 && touches[i]->target() != 'n')
      _state = GestureFailed;
    else if(touches[i]->touchId() == 1 && touches[i]->target() != 'm')
      _state = GestureFailed;

.touchesUp(Array *touches, Event *event)
  if(touches[0]->touchId() == 0)
    if(event->allTouches()->count() == 1 && touches[0]->target() == 'n')
      connectNodes();
    else
      _state = GestureFailed;
  else
    if(event->allTouches()->count() == 1 || touches[0]->target() != 'm')
      _state = GestureFailed;
```
\[ \text{gesture} = \mathcal{D}_1^n \mathcal{M}_1^* \mathcal{D}_2^m (\mathcal{M}_1^n \mathcal{|} \mathcal{M}_2^m) * \mathcal{U}_2^m \mathcal{M}_1^* \mathcal{U}_1^n \]

\text{gesture.finalTrigger(connectNodes());}

\text{gestureMatcher.add(gesture);}
Declarative Specification

gesture = D^n_1 M^n_1*D^m_2(M^n_1||M^m_2)*U^m_2 M^n_1*U^n_1

gesture.finalTrigger(connectNodes());
gestureMatcher.add(gesture);

Declarative Specification
gesture = \( D_1^n M_1^* D_2^m (M_1^n | M_2^m) * U_2^m M_1^* U_1^n \)
gesture.finalTrigger(connectNodes());
gestureMatcher.add(gesture);

Declarative Specification

Recognition Code Generation
gesture = $D^n_1 M^n_1 \ast D^m_2 (M^n_1 \mid M^m_2) \ast U^m_2 M^n_1 \ast U^n_1$

gesture.finalTrigger(connectNodes());
gestureMatcher.add(gesture);

Declarative Specification

Recognition Code Generation

Conflict Detection
Touch Event Symbol

$E \in \{D, M, U\}$
Gesture Tablature

pan Camera()

zoomCamera()
Gesture Tablature

object type → s

touch down

touch move

touch up
Gesture Tablature

\[D_1^s M_1^s D_2^b (M_1^s | M_2^b) (U_1^s M_2^b U_2^b | U_2^b M_1^s U_1^s)\]
User Study:
Build applications with/without Proton
User Study:
Build applications with/without Proton
User Study:
Readability of Notation
Method

• Question: How quickly and accurately can programmers comprehend gestures expressed as: [iOS code, regular expressions, tablature]?

• Recruited n=12 experienced programmers

• Task: given a definition, identify a video of a gesture matching that definition
_state = GesturePossible;

touchesBegan(Array *touches)
    if(touches->allTouches()->count() > 2)
        _state = GestureFailed;
    if(touches[0]->target() != 'a')
        _state = GestureFailed;

touchesMoved(Array *touches)
    for(i = 0; i < touches->count(); i++)
        if(touches[i]->target() != 'a')
            _state = GestureFailed;

touchesEnded(Array *touches)
    if(touches[0]->target() != 'a')
        _state = GestureFailed;
    return;
    if(touches->allTouches()->count() == 2)
        if(touches[0]->touchId() == 0)
            execute();
        else
            _state = GestureFailed;
    else
        _state = GestureRecognized;
\[ D_0^a M_0^a (D_1^b (M_0^a | M_1^b))^b U_1^b M_0^a )^a U_0^a \]

execute()
Counterbalancing

Participant 1

iOS  RegEx  Tablature
Counterbalancing

Participant 1
- iOS
- RegEx
- Tablature

Participant 2
- Tablature
- iOS
- RegEx
Counterbalancing

Participant 1

Participant 2

Participant 3

...
Results

Time (s) to Understand Gesture

- Tablature: 1x
- Expression: 4.7x
- iOS: 4.7x
Other Perspectives

- **Debugging**: Present gestures with bugs, ask developers to fix them

- **Maintenance, extension**: Give an existing program, ask developers to extend it

- **Creation**: Assign open-ended programming tasks, record cognitive process through thinkaloud protocol
Reflections on picking tasks

• Make sure you’re not just measuring the learning curve of your tool (tutorials, warmup tasks, discard early data)

• Some narrow tasks that exercise the features of the tool

• Some open-ended tasks that don’t prescribe how to use the tool (external validity)
Quicksilver: Automatic Synthesis of Relational Queries
(Lu, Bodik; Hartmann)

- Synthesis techniques create queries over data tables using programming-by-demonstration
- Example: Given two tables of student data and signup times, generate a third table that lists names, times, and codes for students who have codes.

<table>
<thead>
<tr>
<th>Student</th>
<th>Email</th>
<th>Class</th>
<th>GPA</th>
<th>Adv. Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Doe</td>
<td><a href="mailto:jd@b.edu">jd@b.edu</a></td>
<td>Senior</td>
<td>4.0</td>
<td>2315</td>
</tr>
<tr>
<td>Anne Foe</td>
<td><a href="mailto:af@b.edu">af@b.edu</a></td>
<td>Soph.</td>
<td>4.0</td>
<td>8112</td>
</tr>
<tr>
<td>Rachel Foe</td>
<td><a href="mailto:rf@b.edu">rf@b.edu</a></td>
<td>Fresh.</td>
<td>3.3</td>
<td>2948</td>
</tr>
<tr>
<td>Lue Goe</td>
<td><a href="mailto:lg@b.edu">lg@b.edu</a></td>
<td>Senior</td>
<td>3.2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:00pm</td>
<td>John Doe</td>
</tr>
<tr>
<td>1:20pm</td>
<td>Anne Foe</td>
</tr>
<tr>
<td>1:30pm</td>
<td>Jimmy Toe</td>
</tr>
</tbody>
</table>

Advisees | Signups
Quicksilver:
Automatic Synthesis of Relational Queries
(System: Lu, Bodik; Study consulting: Hartmann)

• Within-subject comparison of users’ performance (task success, completion time) with Quicksilver vs. commercial spreadsheet without synthesis

• High-level result: Quicksilver enables end-users to successfully generate queries that they cannot otherwise achieve within a 10 minute time limit.
Quicksilver: Open Questions

• What are the boundaries of what users can accomplish with the tool?
• What queries are easy/hard to express?
• What kind of errors would users commit?
Automata Tutor

Rajeev Alur (Penn), Loris D’Antoni (Penn), Sumit Gulwani (MSR), Bjoern Hartmann (Berkeley), Dileep Kini (UIUC), Mahesh Viswanathan (UIUC)
**Problem:**
Draw a DFA that accepts all strings over \{a,b\} in which ‘ab’ appears exactly twice.
Using Synthesis for Feedback

- Grade: 4/10
- Feedback: You are computing the language \{ s | ‘ab’ appears in s at least 2 times \}
How good are the grades?

• Collected 800 UIUC student attempts for 6 different assignments

• Graded each assignment using 3 methods:
  • two instructors
  • automatic grading by the tool
  • a naïve grader
    (gives grade proportional to # of states)

• Result: Tool performs at level of instructors [IJCAI 13]
How useful is feedback?

• **Upcoming:** Field experiment through in-class deployment in courses at UIUC and UPenn

• Random assignment of students to three experimental groups:
  No feedback, Counterexample only, Full feedback

• Measures:
  • Number of attempts to correct submission
  • Improvement over time
  • Score on a final single-submission problem
Methodology

Reflections

• Initial evaluation done offline - good idea for “fragile” code to get started - but no feedback from users.

• Online tool has to robust enough to run without hand-holding. Benefit: scale allows for clean experiment for hypothesis testing!

• Online deployments can scale - but we also know less about users’ experience. Will develop survey, and may want to supplement with lab observations.
Other Evaluations

• CodeHint: synthesizes code from user-provided partial dynamic specifications of the desired behavior
  (Galenson, Reames, Bodik, Sen - [LIVE’13])

• Autograding and feedback for Embedded Systems lab assignments
  (Seshia, Alur, Hartmann)
Parting Thoughts

All studies have limitations.

Despite limitations - all of these examples give us much valuable information and insight.

Triangulation between quantiative and qualitative metrics will give most complete picture.