the state of the art in
end-user software engineering

Andrew J. Ko

based on an article in review for
ACM Computing Surveys

Robin Abraham and Laura Beckwith | Microsoft Corporation
Alan Blackwell | The Computer Laboratory | University of Cambridge
Margaret Burnett, Martin Erwig, and Joseph Lawrance | School of EECS, Oregon State University
Henry Lieberman | MIT Media Laboratory
Brad Myers | Human-Computer Interaction Institute | Carnegie Mellon University
Mary Beth Rosson | Information Sciences and Technology | Penn State University
Gregg Rothermel | Dept. of Computer Science and Engineering | University of Nebraska at Lincoln
Chris Scaffidi and Mary Shaw | Institute for Software Research | Carnegie Mellon University
Susan Wiedenbeck | College of Information Science and Technology | Drexel University

Tuesday, May 26, 2009
achieve the **program** for **other’s** use

achieve the program **behavior** for **personal** use
professional programming
achieve the program for other’s use

end-user programming
achieve the program behavior for personal use
end-user programming

Jack “I want a different picture to show on my web page every time someone visits.

I think I’ll write some JavaScript to show one of a few images.”
"I think I’ll add some more pictures.

Oh, I have to change all of my if statements.

Jack
end-user programming

Jack  “I think I’ll add some more pictures.

Oh, I have to change all of my if statements.

REQUIREMENTS (change)
How’d you do the random photos on your web site?

Oh, just look at my code...
Jane  “How’d you do the random photos on your web site?

Jack  “Oh, just look at my code...

REUSE  (requires planning)
end-user programming

Jane  “Oh, but I wanted it to use photos from my Flickr site.

Jack  “Oh, I didn’t write it for that. You’d have to do it a different way.
end-user programming

Jane  “Oh, but I wanted it to use photos from my Flickr site.

Jack  “Oh, I didn’t write it for that. You’d have to do it a different way.

SPECIFICATIONS (aren’t specified)
end-user programming

Jane  “It’s not ever choosing this particular picture...

Jack  “Really? It works for me.
end-user programming

Jane  “It’s not ever choosing this particular picture...

Jack  “Really? It works for me.

TESTING  (doesn’t happen)
Jack “Why won’t it choose this picture?
end-user programming

Jack

“Why won’t it choose this picture?

DEBUGGING (is still hard)
end-user software engineering

incorporating activities that improve software quality into people’s existing practices
outline

what have we learned about ...

- **requirements** what should my program do?
- **design+specs** how should my program work?
- **reuse** what can I use to write my program?
- **testing+verification** is my program working correctly?
- **debugging** why is my program not working?

cross-cutting issues
what should my program do?

(REQUIREMENTS)

what a program should do in the world ...

... but not how it should do it
( REQUIREMENTS )

are often emergent

because requirements can change without asking
( REQUIREMENTS )

are often emergent

because requirements can change without asking

tend to involve automation

programming is often optional
( REQUIREMENTS )

are often emergent
because requirements can change without asking

tend to involve automation
programming is often optional

come from the user themselves
but this often changes if the code is more broadly useful
more intended uses

more attention to systematic quality assurance
more intended uses

more attention to systematic quality assurance

end-user SE

professional SE
how should my program work?

(DESIGN & SPECIFICATIONS)

how a program achieves its requirements
(DESIGN & SPECIFICATIONS) are often emergent because constraints and dependencies on design are uncovered over time.
(DESIGN & SPECIFICATIONS) are often emergent because constraints and dependencies on design are uncovered over time.

are not obviously valuable in explicit forms

no immediate payoff
what specs are good for

three ways that specs can help

- support a particular design **process**
- make the spec the **language**
- as secondary notations for **validation**
support a design process

support design exploration and iteration

Newman’s DENIM
make the spec the language
design a domain-specific language to replace a lower-level language

Yahoo Pipes
intermediate validation language

a ViTSL template that helps generate provably correct grading spreadsheets

Abraham & Erwig’s ViTSL
what can I use to write my program?

(REUSE)

composition or modification of APIs and existing code
(REUSE) is difficult because...

there are so many things to reuse

... and in many places
REUSE is difficult because...

there are so many things to reuse

... and in many places

reused things change

APIs change internally
input changes
output requirements change
finding code on the web

example code in tutorial or in “view source”

This page contains JavaScript examples - examples of JavaScript tricks that you can use for your own website.

You can use any of these examples simply by copying and pasting the code straight into your website.

We’ve got plenty of HTML examples too!

<table>
<thead>
<tr>
<th>Description</th>
<th>Code</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Popup Window</td>
<td>&lt;!-- JavaScript examples by Quackit.com --&gt;</td>
<td>Get your HTML codes in a popup!</td>
</tr>
<tr>
<td></td>
<td>&lt;script type=&quot;text/javascript&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>// Popup window code</td>
<td></td>
</tr>
<tr>
<td></td>
<td>function newPopup(url) {</td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>Jump Menu</td>
<td>&lt;!-- JavaScript examples by Quackit.com --&gt;</td>
<td>Jump to...</td>
</tr>
<tr>
<td></td>
<td>&lt;script language=&quot;javascript&quot; type=&quot;text/javascript&quot; &gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;!-- hide</td>
<td></td>
</tr>
<tr>
<td>Automatically launch the &quot;print&quot;</td>
<td>&lt;!-- JavaScript examples by Quackit.com --&gt;</td>
<td>Print this page</td>
</tr>
<tr>
<td>dialog</td>
<td>&lt;a href=&quot;JavaScript:window.print();&quot; &gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&gt;Print this page&lt;/a&gt;</td>
<td></td>
</tr>
<tr>
<td>&quot;Print&quot; dialog with printer icon</td>
<td>&lt;!-- JavaScript examples by Quackit.com --&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
finding code in repositories

searching a repository designed for sharing

Scaffidi’s Topes
changing parameters
modifying values in an existing structure
customizing templates

parameterizing and modifying structure
invoking APIs

finding the right primitive for the desired behavior is a search problem

Apple’s Automator API
modifying examples

(code for making an Automator action)

Apple’s Applescript

on run {input, parameters}
    set whatN to (get n of parameters)
    set L to (get words of input)
    set output to {}
    repeat with ix from 1 to (count L)
        if ix mod whatN is 0 then
            set end of output to item ix of L
        end if
    end repeat
    set text item delimiters to " "
    return (output as string)
end run
is my program working correctly?

( TESTING & VERIFICATION )

gaining an objective and accurate level of confidence about the correctness of a program
( TESTING & VERIFICATION ) is difficult because...

people are **imperfect** oracles

don’t always know the
correct behavior
TESTING & VERIFICATION is difficult because...

people are imperfect oracles

don’t always know the correct behavior

end users are more overconfident

skepticism comes with experience
(TESTING & VERIFICATION) is difficult because...

people are **imperfect** oracles
don’t always know the
correct behavior

end users are more **overconfident**
skepticism comes with experience

**Immediate feedback**
combats overconfidence

increases visibility of correctness
detecting errors with testing
visualize test coverage data in context

Burnett et al’s WYSIWYT
checking against specs

propagate assertions through data flow

Burnett et al’s assertions
consistency checking

exploit program notation to infer types and check for consistent use

Abraham and Erwig’s UCheck
why isn’t my program working?

( DEBUGGING )

finding and removing errors after an error has been detected
(DEBUGGING) is difficult because...

people have to guess what is causing failing output

guesses are usually wrong the first time
(DEBUGGING) is difficult because...

people have to **guess** what is causing failing output

guesses are usually wrong the first time

people have to understand **dependencies** in code and its execution

writing code ≠ understanding execution
reasoning backwards from output

X the bad value and see what’s contributing

Burnett et al’s Forms/3 fault localization
reasoning backwards from output

ask “why” about output, see what caused it

Ko’s Whyline
offering change suggestions

specify expected value, get change suggestions

Abraham and Erwig’s Goal Debug

<table>
<thead>
<tr>
<th>Average</th>
<th>Above/Below Ave</th>
<th>Improvement</th>
<th>LetterGrade</th>
</tr>
</thead>
<tbody>
<tr>
<td>80.0</td>
<td>0</td>
<td>90.0</td>
<td>A</td>
</tr>
<tr>
<td>69.3</td>
<td>0</td>
<td></td>
<td>D</td>
</tr>
<tr>
<td>90.0</td>
<td>1</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>51.7</td>
<td>0</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>85.0</td>
<td>1</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>75.2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
cross-cutting issues
risk + reward

why would someone use these tools?

low cost, low risk, high reward

*Attention Investment* theory argues that tool adoption is influenced by perceptions of these factors [Blackwell 2000]
the design strategy

surprise

explain

reward
self-efficacy and strategy

people with less confidence in their abilities work in different ways

how can tools account for this variation in strategies?
the design strategy

design many obvious paths between points A and B

incremental, immediate feedback make this possible
the design strategy

design many obvious paths between points A and B

incremental, immediate feedback make this possible
what’s the **interesting** difference between **EUSE** and professional **SE**?
<table>
<thead>
<tr>
<th>activity</th>
<th>professional</th>
<th>end-user</th>
</tr>
</thead>
<tbody>
<tr>
<td>requirements</td>
<td>explicit</td>
<td>implicit</td>
</tr>
<tr>
<td>specifications</td>
<td>explicit</td>
<td>implicit</td>
</tr>
<tr>
<td>reuse</td>
<td>planned</td>
<td>unplanned</td>
</tr>
<tr>
<td>testing &amp; verification</td>
<td>cautious</td>
<td>overconfident</td>
</tr>
<tr>
<td>debugging</td>
<td>systematic</td>
<td>opportunistic</td>
</tr>
</tbody>
</table>
EUP $\neq$ certain language or paradigm

physicists use C and Fortran to explore hypotheses in particular accelerator data

software engineers use Yahoo Pipes to complex and sophisticated RSS aggregators
EUP ≠ a certain group of people

EUP is an intent, not an identity

skilled programmers engage in EUP when...

- writing tools for personal, short-term use
- writing a script to rename some files
- creating a batch Photoshop filter to process some images
EUP ≠ experience

- code for others
  - computer science student programming
  - professional programming

- code for one's self
  - novice programming
  - expert programming

- end user programming
  - programming experience
more intended uses

more attention to systematic quality assurance
more intended uses

more attention to systematic quality assurance
implications

designing EUSE tools means designing for a particular situation in a particular domain. The challenge is to invent tools to generalize across situations and domains.

- **incremental, immediate feedback** tools are useful across many domains and strategies.