Future of End-User Software Engineering: Beyond the Silos

Margaret Burnett
Oregon State University

ACM SIGSOFT Webinar
June 18, 2015

Why is there an “EUSE” area?

• It started with End-User Programming (creating new programs).
• Today, end users write lots of programs [Scaffidi et al.]
  – Using IDEs like...

Settings for EUSE: End Users’ “IDEs”

• CoScripter [Cypher et al.]
• Labview [National Instruments]
• Spreadsheets

Settings for EUSE: End Users’ “IDEs”

• CoScripter [Cypher et al.]
• Labview [National Instruments]
• Spreadsheets
But: Programs are not dependable!
- End-user programming empowers to...
  - ...use bad programs to make bad decisions?
  - Compounded by overconfidence.
- Hurricane Katrina [Scaffidi et al.]
- West Baraboo, Wisc: spreadsheet: underestimated borrowing cost by $400,000 [EUSPRIG].
- Faulty business web applications ➔ lost revenue, credibility [Bulldog Reporter].

But: Programs are not dependable!
- End-user programming empowers to...
  - ...use bad programs to make bad decisions?
  - Compounded by overconfidence.
- Hurricane Katrina [Scaffidi et al.]
- West Baraboo, Wisc: spreadsheet: underestimated borrowing cost by $400,000 [EUSPRIG].
- Faulty business web applications ➔ lost revenue, credibility [Bulldog Reporter].


First(?) EUSE example: WYSIWYT
- What You See Is What You Test [Rothermel/Burnett]
- For end users & spreadsheets, what is testing?
  - Test: A decision if some output is right given the input.
First EUSE example: WYSIWYT

• What You See Is What You Test (Rothermel/Burnett)
• For end users & spreadsheets, what is testing?
  – Test: A decision if some output is right given the input.

√: User Notices a Correct Value...

If this value is right, √ it; if it’s wrong, X it. This testing helps you find errors.

At any time, user can check off correct value.
√: User Notices a Correct Value...

- Cell turns more blue (more "tested").
- At any time, user can check off correct value.

Empirical Results (WYSIWYT + derivatives)

- Over 30 empirical studies:
  - Useful, usable, effective!

But... A Future Beyond the Silos?

- Silo: “a system, process, ... that operates in *isolation* from others”
- Silo risk:
  - tunnel vision
  - lost potential for big, outside-the-box gains.
- Challenges/benefits for a de-siloed future EUSE:
  - Who challenges
  - When challenges
  - Why & How challenges

But... A Future Beyond the Silos?

- Silo: “a system, process, ... that operates in *isolation* from others”
- Silo risk:
  - tunnel vision
  - lost potential for big, outside-the-box gains.
- Challenges/benefits for a de-siloed future EUSE:
  - Who challenges
  - When challenges
  - Why & How challenges

De-Siloing the “Who”

Who: The Basics

- EUSE draws from both SE and HCI
  - From HCI, it inherits the importance of “Who”
- 2 Key Requirements of EUSE:
  - 1. Cannot assume SE skills
  - 2. EUSE tools/techniques must fit:
    - interests,
    - motivations,
    - practices
Who: Today in EUSE

• Who, exactly, are end-user developers?
  – Assumption: People who wish they could program (more/better)?
    • Often not the case.
  – Assumption: People who don’t program?
    • No (counter examples)
• If don’t know who they are…
  – …can’t build tools for them.

Empirical answer: Targeted studies \(\rightarrow\) tools

• Study target audience. eg:
  – Teachers (Wiedenbeck), designers (Rosson, Myers), children (Petre), …
• Reveals important information!
  – resulting tools fit audience’s motivations & practices.
• Good, valid, but not enough …
  – Each population research separate.
  – Siloes knowledge by population.

De-siloing “who”: Roles

• Defn: Professional devs vs. EUDs:
  – Different intent, not experience (Nardi, Ko).
  – EUD as a role not an identity (Ko).
• Insight: Role/intent \(\Rightarrow\) nuances “who”.
  – Expands “who”: even people who program regularly
    • if/when little interest in SE.
  – Example: scientist programming “experiment”
    • for one-shot results.

De-siloing “who”: Roles

• Defn: Professional devs vs. EUDs:
  – Different intent, not experience (Nardi, Ko).
  – EUD as a role not an identity (Ko).
• Insight: Role/intent \(\Rightarrow\) nuances “who”.
  – Expands “who”: even people who program regularly
    • if/when little interest in SE.
  – Example: scientist programming “experiment”
    • for one-shot results.
De-siloing “who”: Roles

- Defn: Professional devs vs. EUDs:
  - Different intent, not experience [Nardi, Ko].
  - EUD as a role not an identity [Ko].
- Insight: Role/intent → nuances “who”.
  - Expands “who”: even people who program regularly
    • if/when little interest in SE.
  - Example: scientist programming “experiment”
    • for one-shot results.

Intent as a basis of de-siloing

- Challenge: Make EUSE tools about an end-user developer’s actual intent.
- E.g: medical scientist’s lab application to collect sensor data.

Attention Investment (Blackwell)

- A common SE assumption:
  - “If we build it, they will come”
- Attn. Investment: why should they?
- PERCEIVED Cost (time):
  - Learn, use.
- PERCEIVED Benefit?
  - Clear without paying the costs?
- PERCEIVED Risks:
  - Eg: probability wasted cost (time) ...
Intent and Attention Investment

- Lab example (intent=fix bug quickly):
- Suppose scientist wants more powerful problem-solving tool — and spots a tool that might help.
- Ventures to touch one:
  - First impression: perceived cost to learn+use.
  - Also perceived risk (probability): tool won’t help.
  - If perceived costs & risks > benefits, unlikely to use the tool (Blackwell).

So, what does that tell us?

- For EUSE tools to target intent-based “who”:
  - (1) must connect to EUD’s intent whenever arises
  - (2) must entice EUD to take appropriate SE steps
    - by making explicit at that moment likely costs, benefits, risks.

How: 3 intent-based strategies

- 1. Research-then-serve existing intent.
  E.g. EUSE debugging: E.g. “why” debugging (Ko, Myers)
How: 3 intent-based strategies

1. Research-then-serve existing intent.
   E.g. EUSE debugging: E.g. “why” debugging [Ko, Myers]

   Question: Why didn’t Pac resize 0.57?
   Answer: One or more of these actions prevented Pac resize 0.57 from happening.
   I.e. checking each action to find out what went wrong.

2. Infer-then-serve intent.
   E.g. EUSE debugging/design: IdeaGarden [Cao, Kwan, Fleming, Scaffidi, Burnett, ...]

   You’ve copied the cell in the “addresses” column of row the and pasted it into a webpage.
   To make your script work for all rows, try enhancing your script with repeat. repeat can help you copy.
   row by row, a column and paste it into a webpage. Do it like this:
   1. Find the lines in your script that make it work for raw

3. Try to generate intent.
   E.g. Pex4Fun (gamify programming) [Tillman et al.]

   Welcome new user! Edit Settings: My Courses | My Quizzes | Settings | Sign Out
   Share your name:

   public class Program {
   public static int Puzzle(int x) {
   // Can you figure out values for m, n in this linear formula?
   int m = ?;
   int n = ?;
   return m * n;
   }
}
Intent-based strategies (cont.)

• These examples have in common:
  – (1) target EUD intent
    • as of “right now”
  – (2) draw EUD into doing just a little
    • when can benefit them directly
    • make clear potential costs, benefits, risks
  – (3) give EUD supports just in time, in their own environment
    • allows them to succeed at SE behaviors
  • These strategies work! (empirical)
  – seem to point ways forward.

De-Siloing the “When”

Today’s EUSE: Beyond Coding

• Adds support for lifecycle stages to EUP:
  – Testing (WYSIWYT) [Rothermel/Burnett]
  – Design [Rosson]
  – Assertions [Shaw]
  – Reverse engineering [Cunha]
  – Refactoring [Dig, Elbaum, Stolee]
  – Fault/smell/error prevent/detect [Dou, Erwig, Hermans, Scaffidi]
Today’s EUSE: Beyond Coding

• Adds support for lifecycle stages to EUP:
  – Testing (WYSIWYT) [Rothermel/Burnett]
  – Design [Rosson]
  – Assertions [Shaw]
  – Reverse engineering [Cunha]
  – Refactoring [Dig, Elbaum, Stolee]
  – Fault/smell/error prevent/detect [Dou, Erwig, Hermans, Scaffidi]

Challenge: Not Just “Beyond Coding”

• Step 1 (some of this already here):
  – interactive
  – incremental
  – don’t trap me in a “mode”
• may require novel reasoning algorithms (e.g. [Dou, Groce, Hermans, Ko, Rothermel]).

Challenge: Beyond “Beyond Coding”

• Step 2: Beyond the Silos
  – these approaches silo tools and research into lifecycle phases.
• Challenge:
  – transcend one-lifecycle-phase-at-a-time research silos in favor of bigger visions
  – frame EUSE problem in terms of human activity (not SE lifecycle phase)

Challenge: Beyond “Beyond Coding”

• Step 2: Beyond the Silos
  – these approaches silo tools and research into lifecycle phases.
• Challenge:
  – transcend one-lifecycle-phase-at-a-time research silos in favor of bigger visions
  – frame EUSE problem in terms of human activity (not SE lifecycle phase)

Frieda’s story

• What Frieda does:
  – 1. Company issues this year’s annual budget-tracking update.
  – 2. At first, Frieda has 4 variants:
    • last-year company budget-tracking spreadsheet.
    • her last-year dept budget-tracking spreadsheet.
    • this-year company budget-tracking spreadsheet.
    • her emerging this-year dept budget-tracking spreadsheet.
Frieda’s story
• What Frieda does:
  – 1. Company issues this year’s annual budget-tracking update.
  – 2. At first, Frieda has 4 variants:
      • last-year company budget-tracking spreadsheet.
      • her last-year dept budget-tracking spreadsheet.
      • this-year company budget-tracking spreadsheet.
      • her emerging this-year dept budget-tracking spreadsheet.

Frieda’s story (cont.)
  – 3. proceed “almost” like last year:
      • try to copy many of last yr’s changes to this year.
      • problems: copy-paste adjusting itself wrong.
      • test (eyeball).
      • fix (if/when notice errors).
      • save intermediate versions (just in case).
      • by now, Frieda has a lot of variants!
Frieda’s story (cont.)

3. proceed “almost” like last year:
   • try to copy many of last yr’s changes to this year.
   • problems: copy-paste adjusting itself wrong.
   • test (eyeball).
   • fix (if/when notice errors).
   • save intermediate versions (just in case).
   • by now, Frieda has a lot of variants!

Frieda’s story (cont.)

3. proceed “almost” like last year:
   • try to copy many of last yr’s changes to this year.
   • problems: copy-paste adjusting itself wrong.
   • test (eyeball).
   • fix (if/when notice errors).
   • save intermediate versions (just in case).
   • by now, Frieda has a lot of variants!

Frieda’s story (cont.)

3. proceed “almost” like last year:
   • try to copy many of last yr’s changes to this year.
   • problems: copy-paste adjusting itself wrong.
   • test (eyeball).
   • fix (if/when notice errors).
   • save intermediate versions (just in case).
   • by now, Frieda has a lot of variants!

   • = These lifecycle steps:
     – Reverse engr, reuse, implementation, testing, debugging
     – All intermingled!

Insight

• Human activity:
  – Exploration: more holistic than 5 lifecycle phases.
  – Need to also capture:
    • Frieda backs out of partial solutions,
    • Frieda puzzles about odd behaviors,
    • Frieda wants to mix the best bits of partial solutions, ...

• Possibility:
  – First-class EUD intents (“explore”)  approaches beyond SE lifecycle-phase silos.

• Working on this… (Myers/Erwig/Burnett/Sarma/Rothermel)

Challenge: Theory to De-Silo

• Problem:
  – De-siloing through unifying principles:
    how gains/tools/techniques fit together?
Challenge: Theory to De-Silo

• Problem:
  – De-siloing through unifying principles: how gains/tools/techniques fit together?
• Opportunity: Harness theories
  – on how humans reason & problem-solve.

Example 1: Theory ➔ Strategic Insights

• Recall: Attention Investment [Blackwell] explains how EUDs decide
  – whether to try a new tool/feature
• Has shaped/predicted acceptance of EUSE tools [Bogart, Burnett, Fleming, Kwan, Ruthruff, ...]
• Also prescriptive, suggesting strategies to improve acceptance/adoptions

Example 2: Theory Aiding Empirical Studies

• The Design:
  – Work design theory:
    • types of questions to ask [Murphy-Hill ICSE'14]
  – Human error theory:
    • an empirical framework used in EUSE [Ko et al.]
• The Interpretation:
  – Why statistical gender differences in feature usage in EUSE debugging tools [Beckwith]
Example 2: Theory Aiding Empirical Studies

• The Design:
  – Work design theory:
    •⇒ types of questions to ask [Murphy-Hill ICSE’14]
  – Human error theory:
    •⇒ an empirical framework used in EUSE [Ko et al.]
• The Interpretation:
  – ⇒ Why statistical gender differences in feature usage in EUSE debugging tools [Beckwith]

Example 3: Connecting the Dots

• Information Foraging Theory:
  – predator/prey theory on how people seek info
  – reveals behaviors in various EUSE settings [Kuttal/Sarma]
  – showed commonalities & design patterns
    • ⇒ 20 different SE tools [Fleming]
    • spanning debugging, refactoring, and reuse

What EUSE has to say to SE

• Makes problems / needs visible.
• New vision: developer-tool partnership:
  – Dev’s motivations ⇒ what tool “wants” of them.
  – First-class focus on intent.
  – Benefits professional devs!
• Examples:
  – Gender differences EUDs ⇒
  Gender differences professional devs [Burnett]
  – Whyline for Alice ⇒ Whyline for Java [Ko/Myers]

Onward...

What EUSE has to say to SE

• Makes problems / needs visible.
• New vision: developer-tool partnership:
  – Dev’s motivations ⇒ what tool “wants” of them.
  – First-class focus on intent.
  – Benefits professional devs!
• Examples:
  – Gender differences EUDs ⇒
  Gender differences professional devs [Burnett]
  – Whyline for Alice ⇒ Whyline for Java [Ko/Myers]
What EUSE has to say to SE

Question: Why didn’t Pac realize 6.57?
Answer: It’s all or none of these...

Why do you think this is the case?

- Dev’s motivations > what tool “wants” of them.
- First-class vision: developer-tool partnership:
  - Dev’s motivations > what tool “wants” of them.
  - Benefits
  - Examples
  - Gender differences
  - Whyline for Alice

Conclusion

- Why de-silo
  - Transformative gains require getting outside the box.

Conclusion (cont.)

- Who challenge: EUD as an intent
  - Opens door to intent-oriented operations
    - Manipulate & reward intent, infer intent, target intent...
- When challenge: Beyond lifecycle silos
  - Opportunistic, incremental, interactive, not “moded in”
- Why & how challenge:
  - Theory: a foundation, a strategic guide, a connector

Conclusion (cont.)

- Who challenge: EUD as an intent
  - Opens door to intent-oriented operations
    - Manipulate & reward intent, infer intent, target intent...
- When challenge: Beyond lifecycle silos
  - Opportunistic, incremental, interactive, not “moded in”
- Why & how challenge:
  - Theory: a foundation, a strategic guide, a connector

Conclusion (cont.)

- Who challenge: EUD as an intent
  - Opens door to intent-oriented operations
    - Manipulate & reward intent, infer intent, target intent...
- When challenge: Beyond lifecycle silos
  - Opportunistic, incremental, interactive, not “moded in”
- Why & how challenge:
  - Theory: a foundation, a strategic guide, a connector
- Not easy, but worth it!
  - Big visions, beyond one silo at a time!
Resource

- Margaret M. Burnett and Brad A. Myers (2014). Future of end-user software engineering: Beyond the silos. Proceedings on Future of Software Engineering (FOSE’14 at ICSE’14), Pages 201-211.

- http://dl.acm.org/citation.cfm?id=2593896