Foundations and Strategies

Surprise-Explain-Reward

CS352
Announcements

• Notice upcoming due dates (web page).
• Where we are in PRICPE:
  – Predispositions: Did this in Project Proposal.
  – RI: Research was studying users. Hopefully led to Insights.
  – CP: Concept and initial (very low-fi) Prototypes.
  – Evaluate throughout, repeat iteratively!!
End-User Software Engineering: “Surprise-Explain-Reward”

(I gave this talk at Google in 2007)
End-User Software Engineering: What’s the Problem

• There is a lot of end-user-created software in the real world (mostly spreadsheets):
  – Errors exist in up to 90% of “production” spreadsheets.

• Overconfidence of end users creating and modifying their programs.
End-User Software Engineering

Goal

• Goal: Reduce errors in end-user programs.
  – Without requiring training or interest in traditional software engineering.
• Context: EUSES Consortium
The Setting:
A Research Prototype in Forms/3

- Spreadsheet paradigm.
- Examples/screenshots use this prototype.

EUSE: Surprise-Explain-Reward
“If We Build It, Will They Come”: What we built
Testing for End-User Programmers

• For end users & spreadsheets, what is testing?
  – A Test: A decision if some output is right given the input.
  – Test Adequacy: Have “enough” tests been performed?
WYSIWYG: The Features We Want Them to Use Will...  

• Incrementally update “testedness” (as per a formal criterion), and...  
• ...allow the user to incrementally inform system of decisions, and...  
• ...immediately reflect this information in border colors.
Initially the Spreadsheet is Untested

EUSE  Surprise-Explain-Reward
Explaining   Surprise   Rewards
The User Notices a Correct Value

EUSE Surprise-Explain-Reward Explaining Surprise Rewards
Hmm.. How Can I Turn the Purple Cell Blue?
Example: New Input

EUSE  Surprise-Explain-Reward  Explaining  Surprise  Rewards
Example: Re-Decision

```
if Weekly_BasePay > 1500
then Weekly_BasePay * 0.10
else Weekly_BasePay * 0.15
```
Many Empirical Studies Regarding WYSIWYT

• WYSIWYT participants always:
  – Have higher test coverage.
  – Show a functional understanding.
  – Appropriate confidence/judgment of “testedness”

• In some ways:
  – More effective and faster at debugging (some problems, some bugs, ...)
  – Less overconfidence about correctness.
Assertions: What and Why

• Supplemental information about a program’s properties.

• Add checks and balances that continually “guard” correctness...
  – which can’t be accomplished via testing.

• Need not be all or none:
  – Even one or two assertions provide some benefit.
  – Can add incrementally to refine specifications.
Integration of Assertions

There's got to be something wrong with the formula
How Can We Get End Users Interested?

Surprise-Explain-Reward
Attention Investment

• Usual CS view:
  – “If we build it, they will come”

• But, why should they?
  – Cost of new feature: learning it + ongoing cost of interacting with it.
  – Benefit of new feature: not clear without incurring the cost.
  – Risks: wasted cost (time), getting environment into an odd state from which can’t easily recover.
How to Interest Them: Arouse Curiosity

• Psychology researchers tell us (and empirical studies of programming confirm):
  – Users/programmers believe their programs work.
    • Thus, they have all the information they think they require.

• Research in curiosity also suggests:
  – Showing them the presence of an “information gap” makes them curious.
Our Strategy: Surprise-Explain-Reward

• **Surprise**: shows them the presence of information gap (to arouse curiosity).

• **Explain**: users seek explanation to close the information gap.
  – Self-directed learning (a key attribute).
  – Suggests the actions we want them to do.

• **Reward**: make clear the benefits of taking those actions early.
The Setting for Surprises

• WYSIWYT testing: accessible, and subjects find it easy at first.
  – When “stuck”, they can ask for help conjuring up test cases via Help-Me-Test (HMT).

• Empirical:
  – Users turn to HMT when they get stuck.
  – They like it, and use it more than once.

• Opportunity for surprises: HMT at the same time suggests assertions.
Surprises

• **Surprise 1:** HMT’s assertions
  – Reveals an information gap.
  – These are deliberately bad guesses.
Surprises (cont.)

• **Surprise 2**: red circles around values while HMT is ruminating.
  – HMT’s “thinking” behavior is transparent.

• **Note**: all feedback is **passive**.
  – Attempts to win user’s attention but does not require it.
  – Empirical: users go several minutes before acting on this feedback.
  – Will return to this issue later.
Explanation System Principles

- Semantics, reward, suggested action.
  - With enough info to succeed at the action.

The computer's testing caused it to wonder if this would be a good guard. Fix the guard to protect against bad values, by typing a range or double-clicking.
Explanation System (cont.)

- Why explanations’ viewport is via tool tips.
  - Psychology: users seek explanation from the surprising object.
- Why suggested action.
  - Minimalist learning: get users to take action.
- Why reason/reward.
  - Attention investment: must make rewards clear.
Rewards for Entering Assertions

- Red circles around values indicate either bugs or incorrect assertions.
  - Same long term as in learning phase.
- Improved HMT behavior on input
  - Always occurs, but harder to notice.
- HMT challenges assertions on non-input cells, aggressively seeking bugs.
Rewards (cont.)

• Computer-generated assertions might “look wrong”.
• Red circles around conflicting assertions.
• Are first surprises, then rewards.
A Behavior Study

• 16 participants (business majors).
  – Familiar with spreadsheets, no assertions experience.

• Assertions treatment: none at all.
  – No assertions provided.
  – No mention of assertions at all.

• Research Question: Does Surprise-Explain-Reward entice and enable users to use assertions?
Results: Enticing Users

• Surprises got their attention (eventually):
  – 15 (94%) did use assertions at least one task.
  – Task 1 time of 1st assertion entry: 13 minutes.

• Once they discovered assertions, they kept using them:
  – 14 (87%) used them on both tasks.
  – 18 assertions/user (mean).
  – Task 2 time of 1st assertion entry: 4 minutes.
Results: Enticing Users (cont.)

• Was HMT the entry mechanism?
  – At first: In task 1, 74% were entered via HMT.
  – By task 2, only 33% were entered via HMT (but still as many assertions entered).

• => HMT introduced/trained them, but they didn’t need that support for later assertions.
Results: Sufficiency of Rewards

• Were rewards sufficient?
  – After users used assertions once, used them again.
  – In fact, 56% used them within the 1st minute of Task 2.
  – “… I found them reassuring because I think they help with accuracy.”
  – “I did so to try to get the help feature to stay within my limits.”
Activity

• Can Surprise-Explain-Reward help your project (or, the online grocery)?
  – Is there a feature they might not use that would help them?
  – What is the circumstance in which they should use it?
  – Can you arouse their curiosity about it at a time consistent with this circumstance?
  – Can you find a way to make your surprise passive?
A Closer Look at “Explain”:

What do end-user debuggers want to know?
Experiment

– Pair think-aloud.
– Gave them almost nothing except each other.
  • No tool tips, little instruction, instructor available only via IM, …
– Content analysis of their words in context to see what they wanted to know.
Oracle / Specification

“Divided by 10? I don’t know...I guess it should be times 10.”

• Gaps not centered on features!
• 40% - Users focused on the task, not the environment and its features.

**Implications:**

(1) Need more research into supporting this information gap.

(2) Consistent with Carroll/Rosson’s “active user”.

---

EUSE  Surprise-Explain-Reward  Explaining  Surprise  Rewards
Strategy

“What should we do now?”
“Let’s type it in, see what happens.”

• 30% about strategy
  – mostly (22%/30%) strategy hypotheses.

Implication: Many were global in nature, with no central feature/object to tie to.
Features

“So with the border, does purple mean its straight-up right and blue means it’s not right?”

• Type of information that can work well with tool tips, feature-centric devices.
• Only accounted for 16%

Implication: Focusing on this type of information would address only a fraction of what our participants wanted to know.
Self-Judgment

“I’m not sure if we’re qualified to do this problem.”

• These metacognitive instances are an important factor in learning
• Also ties to self-efficacy, which ties to debugging persistence
• Made up 9%(!) of all information gaps

Implication: Improving accuracy of self-judgments may in turn increase debugging effectiveness.
Big (!) Gaps

“Whoa!” “Help!”

- User may not be able to voice a specific question
- 5% of total

Implication: The timing and context of a big information gap may reveal the type & cause of confusion.
Implications & Opportunities

• Information gaps:
  – Do not primarily focus explanations on Features
  – Users’ greatest need: Oracle/Specification
  – Strategy outnumbered Features 2:1, needs local and global support

• Accuracy of users’ Self-Judgments may impact effectiveness
Toward Answering What Users Asked

• Drew from:
  – Above results.
  – Various education theories.
  – Self-efficacy theory.
  – Shneiderman et al.’s and Baecker’s research into how to do video demonstrations.
Current Prototype

• A trio:
  – Tool tips.
    • Features/Rewards + links to strategy videos.
  – Video explanations of strategy.
  – Side panel:
    • Links to text-based and video versions.
Current Prototype Screenshots

In general what should I try to do?

* Check off right values: First, try to find cells whose values are right and place a checkmark. *(This is how you test.)*

* X out wrong values: If you *find wrong values*, place an X mark. Making decisions (placing checks or Xs) on the cells and the interior cell colors help to narrow your search for the error.
Summary of Empirical Findings

• Closed ~half these gaps:
  – Strategy, oracle/specification, self-judgment.

• Video vs. text form:
  – Different tasks caused different ones to rise slightly above the other (eg, learning at first, vs. later refresh, clarification, or digging).
  – Males & females responded very differently to videos.

• Having both really matters!
A Closer Look at “Surprise”: Surprises as Interruptions
2 Types of Interruptions (McFarlane)

• Negotiated.
  – System announces need to interrupt,
    • but user controls when/whether to deal with.
  – Example: underlines under *mispelled* words.

• Immediate.
  – Insists that user immediately interact.
  – Example: pop-up messages.
Experiment

- 2 groups of business students.
- Debugging 2 spreadsheets, varying order.
- A device: assertions.
  - Prior work: they help end users’ debugging.
- Tutorial: no mention of assertions.
- Research question:
  - negotiated or immediate: which better in end-user debugging?
To Interest Users: Surprises (Communicated Negotiated Style)

- “Help Me Test”: can be invoked to suggest new sample values.

The computer’s testing caused it to wonder if this would be a good guard. Fix the guard to protect against bad values, by typing a range or double-clicking.
More Surprises/Rewards (Communicated Negotiated Style)

All guards for a cell must agree.
Surprises
(Communicated Immediate Style)

Instances when users are interrupted:
1. when users indicate interest in assertions
2. when there are value violations
3. when HMT generates assertion(s)
4. when there is an assertion conflict
5. when system-generated assertion(s) are created

Figure 2: Instances of immediate-style interruptions in the experiment.
Results: Will they come?

- Time until enticed to enter assertions.

<table>
<thead>
<tr>
<th>Interruption style</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; Task</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negotiated (n=16)</td>
<td>13:26</td>
<td>3:40</td>
</tr>
<tr>
<td>Immediate (n=22)</td>
<td>8:56</td>
<td>4:49</td>
</tr>
</tbody>
</table>

- Accuracy of assertions: exactly same.
Results: Do they learn it?

- Comprehension ($p=0.0153$).

- Legend: dark=negotiated.
Results: Am I done debugging yet?

• If “yes” too early, relying on wrong answers!

• Results:
  – Negotiated: Reasonable predictors ($p = .04$, $p = .02$).
  – Immediate: Not ($p = .95$, $p = .17$).
Results: Debugging Productivity

• Bugs fixed:
  – Negotiated won (Task 2: p<0.0001).

• Why?
Results Debugging (cont): How They Spent Their Time

• Negotiated did significantly more:
  – Editing formulas.

• Immediate did significantly more:
  – Help-me-test.

• Both groups about the same:
  – Creating/editing assertions.
Discussion: Why?

Debugging Strategies

• Consider:
  – Working memory & immed. interruptions (Bailey et al., Burmistrov/Leonova).
  – Debugging breakdowns & attentional problems (Ko/Myers).
  – Participant activity differences:
    • Immed. stayed with local activities not needing much memory.

• Strong suggestion of local, shallow debugging strategies.
Conclusion

• **Expected:**
  – Immediate: better learning of debugging devices;
  negotiated: better productivity.

• **Surprises:**
  – Better learning & effectiveness with negotiated.
  – Immediate seemed to promote shallow, local strategies.
Bottom Line

• For debugging devices:
  – Our recommendation: use negotiated style.
  – No reasons to use immediate-style in debugging,
  – & several reasons not to.
Overall Summary

• Software engineering devices for end users.
  – Cannot be achieved by grafting color graphics onto traditional approaches.

• Talked about end-user support for:
  – Testing, assertions
  – Thru Surprise-Explain-Reward strategy

• Empirical results: how to surprise, explain, and reward end users problem-solving about their programs.
Leftovers start here
A Closer Look at “Rewards”: Functional and Affective Rewards
Research Questions about Affective Rewards

• RQ1. Effectiveness:
  – Affective rewards impact ability to fix faults?

• RQ2. Usage:
  – Affective rewards impact usage of a debugging device?

• RQ3. Understanding:
  – Affective rewards impact end user’s understanding of a debugging device?
WYSIWYT Rewards

• Rewards:
  – Systematic coloring of cell borders
  – Testedness bar (spreadsheet granularity)
  – Data flow arrows (sub expression granularity)

• Previous studies indicate benefits of these rewards
The Gradebook Spreadsheet

- 70 -
The Gradebook Spreadsheet

[Diagram of a spreadsheet with numerical values entered]
Fault Localization Rewards

- **WYSIWYT**: A springboard
- **Place X-mark when cell’s value is incorrect**
  - Suspect cells are colored in shades of yellow-orange continuum

- Darker the cell interior, greater
Investigating Rewards in Fault Localization

- 5 issues, trade off’s
- 2 Groups:
  - “Low-Reward” Group
  - “High-Reward” Group
- Difference: *Quantity* of potentially *affective rewards*
5 Issues: #1

Mixed Message vs. Loss of Testedness

- High-Reward Group

- Low-Reward Group

Fig 1

Fig 2
Arrows’ Mixed Message vs. Loss of Testedness: #2

- High-Reward Group
  - Data flow arrow colors retained
  - Greater rewards

- Low-Reward Group
  - Data flow arrow colors removed
  - Less rewards

Fig 3

Fig 4
5 Issues: #3

• Testedness Progress Bar

• High-Reward Group  • Low-Reward Group
5 Issues: #4

• High-Reward Group  
  – Explanations were kept intact

• Low-Reward Group  
  – No explanations for “decolored” cell borders, arrows
5 Issues: #5

• High-Reward Group: “Bug Likelihood” bar

• Low-Reward Group: No “Bug Likelihood” bar
5 Issues in a Nutshell

• Differences described are not contrived differences
  – Both groups’ implementations had solid reasons

• High-Reward Group:
  – Quantitatively greater perceivable reward, even if disadvantages from other perspectives
Experiment

• Design of the experiment
  – Both groups had environments with same functionality

• Difference: *Quantity of perceivable rewards*

• 54 participants
  – 24 in Low-Reward group
  – 30 in High-Reward group

• Tutorial:
  – Taught use of WYSIWYT
  – Did NOT teach Fault Localization
Results: Effectiveness

- Measured number of faults fixed by each group
- High-Reward group fixed significantly more faults (p=0.025)

- Significant difference suggests users’ perception of rewards has powerful impact on effectiveness
RQ 2: Usage

• Two metrics:
  – (1) "Persistent" X-marks
  – (2) "Total" X-marks

• Effectiveness difference could be contributed to more use of X marks

• Surprise !! No significant differences were found

• High-Reward participants fixed more faults despite no difference in the amount of usage of the fault localization device!
RQ3: Understanding

• Two types of comprehension:
  – Interpreting the feedback received (2 questions)
  – Predicting feedback under various circumstances (6 questions)

• High-Reward group: More understanding of the feedback’s implications
RQ3: Understanding (cont’d)

• High-Reward participants comprehend better than Low-Reward participants
Interpretation of Results

• Interesting findings:
  – High-Reward participants understood the feedback better, despite “mixed messages”
  – Mixed message was confusing (54 % “makes sense” vs. 46 % “does not make sense”)
  – Confusion should hamper understanding
• Curiosity a factor too?
• Other possible factors?
Conclusion

• Improved debugging, but not through functional rewards
• Affective rewards alone significantly improved debugging effectiveness