SEEUP 2009

Workshop Review and Next Steps

Carnegie Mellon
Human-Computer Interaction Institute

Carnegie Mellon
Software Engineering Institute

Organizers:

Len Bass
Grace A. Lewis
Brad Myers
Dennis B. Smith

Goal

To discuss the importance of end-user software engineering as a way to improve the practice of end-user programming

- Reduce errors and consequences of errors
- Add formalism without disrupting the end user experience

"Look outside of our group instead of just talking to each other"

The State of the Art in End-User Software Engineering [1]

Andy Ko, University of Washington

Multiple reasons for writing code

- Achieve the program for other's use (professional programming)
- Achieve the program behavior for personal use (end-user programming)

EUP, although more informal, does face requirements, reuse, specifications, testing, and debugging problems

The State of the Art in End-User Software Engineering 2

Andy Ko, University of Washington

End-User Software Engineering is incorporating activities that improve software quality into people's existing practices

- Requirements—what should my program do
  - Emergent, tend to involve automation and come from the users themselves
- Design + Specs—how will the program achieves its requirements
  - Emergent, no value in making them explicit
- Reuse—what can I use to write my program
  - Composition and modification, many things to reuse, things change
- Testing + Verification—is my program working correctly
  - Don't know correct behavior, overconfident, immediate feedback
- Debugging—why isn't my program working
  - Guess-based, understand dependencies

The State of the Art in End-User Software Engineering 3

Andy Ko, University of Washington

Cross-Cutting Issues

- Risk + reward—what is the payoff
- Self-efficacy and strategy—levels of confidence

What is the interesting difference between EUSE and professional SE?

- Requirements—implicit vs. explicit
- Specifications—implicit vs. explicit
- Reuse—unplannned vs. planned
- Testing—overconfident vs. cautious
- Debugging—opportunistic vs. systematic
- **Difference is the attention you pay to systematic quality issues**

A challenge

- Build tools that can be generalized across many domains

End-User Software Engineering and Distributed Cognition [1]

Margaret Burnett, Oregon State University

Premise: EUP won't become more disciplined unless

- Perceived payoff
- Discipline is so low cost they don't have to worry about it

Distributed Cognition

- Cognition beyond the individual to encompass interactions between people, resources, materials and environment

Instead of what can we figure out automatically … how can we help the end user think keeping in mind perceived costs/benefits

End-User Software Engineering and Distributed Cognition 2

Margaret Burnett, Oregon State University

WYSIWIT and Surprise-Explain-Reward

- Helps end users perform testing in spreadsheets
- System tracks progress of “things-to-test”, finds possible culprits for errors, and prioritizes them, and user makes value correctness judgments

Debugging Learned Programs

- Allows users to ask questions of machine-learned programs (e-mail program) and explanations represent “code” that users can correct

Debugging Via Information Foraging

- Empirical study of how programmers navigate information when they are debugging—information foraging stands out as a major activity
- System can provide clues about where to look for information

Extending the Boundary of Spreadsheet Programming: Lessons Learned from Chinese Governmental Projects

Xingliang Yu, Chinese Academy of Sciences

Multiple levels of information sharing in Chinese government in form of spreadsheets (mostly Excel)

- Long-tail effect in number of different reports and lifetime
- Different types of data and report formats with no standardization in many cases

Challenges with spreadsheet use

- Expressiveness of programming language
- Usability
- Effectiveness

Solution

- ESL Language (EUD-enabled Spreadsheet Language)

Looking at the concept of a Release Waiting Farm (RWF) for testing plug-ins

End-User Software Development in a Scientific Organization [1]

Mark Vigder, NRC Canada

Characterization

- End users program for themselves but also for teams
- Most of the work is small programs or tinkering with existing programs—limited software knowledge
- No software engineering support or software development processes, e.g. testing, CM, versioning
  - Multiple variants due to cloning exacerbate the problem

Focus: Workflow Implementation

- Even though the workflow tasks are similar/constant, the tools used are very different
- Challenge: How to keep the end user at the workflow level?

End-User Software Development in a Scientific Organization 2

Mark Vigder, NRC Canada

Tool: WF Developer

- Python-based
- Workflow described in terms of activities rather than software apps
- Maps software to tools
- Generates GUI based on workflow script

Results

- Many programming operations done at the GUI level
- Minimization of cloning and modifying (parameterized workflows)
- Easier to automate workflows
- Integration of off-the-shelf tools into workflows

Using Crystal Reports: Examples of Richly-Formatted Report Creation by Non-Developers [1]

Harold Schellekens, SAP BusinessObjects

Rich designer and simple viewers

Extensive formula language

Opportunity is to make the designer easier and more accessible for people not formally trained as developers

- Buttons for some formulas
- Better helpers
- Use reports published on the Web to find the most created
- Managing data model complexity with semantic models
- Templates

Using Crystal Reports: Examples of Richly-Formatted Report Creation by Non-Developers 2

Harold Schellekens, SAP BusinessObjects

Other topics

- Formula language
  - Easier or less necessary?
  - Helpers/wizards and re-entrant problem
- How to move/bridge into more advanced workflows—is a smooth continuum possible/feasible?
  - Self-educating?
- Others
  - Persona-based design
  - Much more usability testing

Proposed Topics 1

- Critical steps for instilling SE discipline in the scientific community
- Experiences/examples of multiple user creation of end-user engineered software
- How to effectively advocate for the “goodness” or “first-class citizenship” of EUSE and an ongoing subset of overall SE
- Implications of some end-user programmed software being part of a “human-mediated” service vs. other software being treated more as a product
- Prevalence and implications of “programming by example” in EUP
- How to shape EUP frameworks to produce better software
  - Automated software quality measurement
  - Heuristics to improve user awareness

Proposed Topics 2

- Enabling and motivating better specification of end-user needs in EUP-created software products
- Inherent difficulty in EUP for different products
- Open research questions in EUSE
- The joy of problem solving and its implications for EUSE
- Lessons learned from EUSE for professional SE
- Implications of distributed cognition

Experiences/Examples of Multiple User Creation of End-User Engineered Software [1]

Idea initially triggered by pervasive computing

Basic idea is dealing with multiple people contributing to a program

Examples

- Assisted living: program to be followed by a house in case of emergencies
  - Variations
  - Configuration management
  - Stakeholder management (Who controls? How to deal with conflicts?)
  - Social implications, e.g. privacy
  - Quality attribute issues: safety, security—testing?
  - Can it scale?

Experiences/Examples of Multiple User Creation of End-User Engineered Software 2

Examples …

- Collaborative prototyping in design school with experts for example using Flash code and others providing requirements with feedback in both directions
- Wiki creation
- Cooperative tailoring—way to share experience between more and less experienced programmers
  - Search tool tailoring that started using basic constructs and then more experienced programmers add complexity
  - Telecommunications example where less experienced end users start with the basics and ask for help when needed
- MIT project to share stories

Experiences/Examples of Multiple User Creation of End-User Engineered Software 3

Thoughts and Research Issues

- Being able to show the state of the program is very important
- The nature of the relationship between end users is important
- Traceability—who did what
- Training
- Greater need for validation
- Difference with professional software engineering is not the problems but what the solution has to look like
- Solving conflicts and clashes in multiple user creation environments in a way that can be used by EUPs
- More languages for collaborative environments that could exploit computation

Experiences/Examples of Multiple User Creation of End-User Engineered Software 4

What would be the right tools?

- Versioning tools that indicate who made the change and **why**, e.g. Google spreadsheets (at a higher level of abstraction)—multiple levels of changes
- Collaboration tools for requirements management

How to Shape EUP Frameworks to Produce Better Software 1

- Can we control/constrain development frameworks so that better software is produced
- Papers that talked about automation had to make assumptions—the more real the less likely the assumptions could be met
- Adobe Catalyst captures common interactive behaviors so that they don't have to be coded over and over again
- More knowledge about what people need to do
- If we automate too much, the joy of problem solving goes away
  - Let them think about what really matters (where they can be creative) and take away the need to think about the obvious
- What are the primitives on which you base your solution—depends on the domain
- Reusable components that can be used across domains with simple parameterization

How to Shape EUP Frameworks to Produce Better Software 2

- Three dimensions
  - Domain—study the domain before coming up with a framework
  - Characterization of the end users
  - Social side of the end user context
- How do these dimensions relate to each other?
  - Need more descriptive studies—still in a very exploratory phase
- As researchers, how can we provide a basis (sets of principles) so that other people can build tools for specific domains

How to Shape EUP Frameworks to Produce Better Software 3

Challenges

- General vs. domain specific
- Principles for domain-specific languages
- Right level of expressiveness

General Comments 1

- Adoption: How do we get end users to use tools; motivation; who are the right people to target as EUP adopters? When does it not make sense to introduce EUP?
- Characterization: What tool characteristics appeal to end users in specific domain
  - e.g. LightRoom—user-friendly version of Photoshop that does about 85% is perfectly appropriate for certain users—metaphor is that of a roll of film (the non-computerized version)
  - Are there common aspects of tools in different domains that can be generalized?
- Adaptation: Tools tailored for different levels of expertise—What is the delta between levels of expertise? How do we reduce the delta? How does the delta change by introducing EUP?
- Continuum from professional programming to EUP to pervasive computing

General Comments 2

- Will skills change when the prevalent demographic becomes "digital natives"
- We need to be better at pulling in work from other fields
- "Software is developed by people, for people, and in the case of EUP it's the same people"
- How can we change the way that end users do their activities without requiring a 4-year degree
- Processes for various EUP situations

Next Steps

Workshop Post-Proceedings (SEI Technical Report)

- Extended abstracts from published papers
- Invited talks
- Workshop results

Future SEEUP or WEUSE workshops?

Funding opportunities?

Collaborations?