Project Citadel: Methods, Data, and Moving Forward

Behram Mistree
Project Citadel: Goal

Goal:

to discover patterns in code and coding behavior that motivate features in a new virtual world scripting language
Outline

- Citadel Methods
- Preliminary Data
- Moving Forward
- Lessons Learning
Methodology: Keylogger

- Cross-platform addons to Emacs (E-Lisp) and Eclipse (Java)

- Logs:
  1. Any change to a document
  2. (Most) Causes of changes (paste/yank, undo, keypress, auto-complete (eclipse), etc.)
  3. When change occurred
Methodology: Subjects

- Logging running on multiple subjects
Methodology: Subjects

- Logging running on multiple subjects
Ewen's Data

- Spans 1.5 Weeks:
  - 451 files opened/modified
  - .html, .h, .hpp, .txt, .cpp, .py, .sh, .tex, etc.
Num open/modifies per file over 1.5 weeks

num open/modifies
File Focus

- /home/ewencp/libprox.git/proxsim/src/Simulator.cpp
  19,736 opens/modifies

- /home/ewencp/sirikata.sirikata/space/src/Proximity.cpp
  17,739 opens/modifies
Methodology: Function Finding

- To get beginning of function, lex for function definition*

- Count number of open and closed braces until all braces match or see beginning of new function.
Methodology: Function Finding (ctd)

- Works for a subset of languages (curly brace dependent).
- Currently only works for class definitions
- Includes commented/stringed code
- Ignores macros
- May have a function definition inside a string
- Unterminated/incorrectly terminated functions incorrectly padded
- Loses if changes happen to function signature
Function Finding: How Well Did it Do?

"void Proximity::handleCheckObjectClass(const UUID) {"  
"void Proximity::handleCheckObjectClass(con) {"  
"void Proximity::proxSubstreamCallback(int x, ProxStreamPtr stream, ProxStreamPtr substream, ProxStreamInfo* prox_stream_info) {"  
"void Proximity::tickQueryHandler(ProxQueryHandler* qh[NUM_OBJECT_CLASSES) {"  
"void Proximity::proxSubstreamCallback(int x, ProxStreamPtr stream, ProxStreamPtr substream, ProxStreamInfo* prox_stream_info) {"  
"void Proximity::handleCheckObjectClass(bool is_) {"  
"std::string Proximity::migrationClientTag() {"  
"void Proximity::proxSubstreamCallback(int x, ProxStreamPtr stream, ProxStreamInfo* prox_stream_info) {"  
"void Proximity::proxSubstreamCallback(int x, ProxStreamPtr stream, ProxStreamPtr substream, ProxStreamInfo* prox_stream_info) {"  
"void Proximity::proxSubstreamCallback(int x, ProxStreamPtr stream, ProxStreamPtr substream, ProxStreamInfo* prox_stream_info) {"  
"void Proximity::removeQuery(UUID obj) {"  

...
Hotspots

Open/mods per func in Simulator.cpp in 1.5 weeks
Why Not Static Analysis?

Selected functions from Proximity.cpp and Simulator.cpp:

<table>
<thead>
<tr>
<th>ID</th>
<th># characters of code</th>
<th># “edits” to function body</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>315</td>
<td>7632</td>
</tr>
<tr>
<td>2</td>
<td>82</td>
<td>8149</td>
</tr>
<tr>
<td>3</td>
<td>793</td>
<td>7397</td>
</tr>
<tr>
<td>4</td>
<td>359</td>
<td>4708</td>
</tr>
</tbody>
</table>

1: Proximity.cpp: "void Proximity::initialize(CoordinateSegmentation* cseg) {
2: Simulator.cpp: "Simulator::QueryIterator Simulator::queriesBegin() {
3: Simulator.cpp  "void Simulator::shutdown() {
4: Simulator.cpp  "void Simulator::createMotionCSVObjects(const std::string csvfile, int nobjects) {"
Write and Return

<Thinking of a way of presenting data>

Ewen Subject often writes and returns

- Writes/modifies function, A, once
- Performs task in other section of code base
- Returns to A, and modifies it.

Studies often do not consider this behavior when analyzing programming.
Moving Forward: Deployment

- 4 emacs keyloggers deployed and ~10 subjects in introductory CS Courses.

- Another round of deployment in Winter Quarter.
Moving Forward: Visualization

histogram of yank sizes

character count
Moving Forward: Visualization

yank events: orange (yank); black (other).
Moving Forward: Analysis

- What features/type of code is in the “difficult” functions? (Eg. asynchronous/recursive/?)
- Sub-function resolution?
- What features/type of code is in the “easy” functions?
- What type of feedback could we give in real-time to programmer?
- What type of feedback could we give in non-real time to programmer?
Lessons Learning: Subject Recruitment

- Have something potential subjects can **do** there
  - Signup sheet >> “You should visit this website, and sign up there!”
Lessons Learning: Subject Recruitment

• Have something potential subjects can do there
  • Signup sheet >> “You should visit this website, and sign up there!”

• Have a shill
Lessons Learning: Subject Recruitment

- Have something potential subjects can do there
  - Signup sheet >> “You should visit this website, and sign up there!”
- Have a shill
  - More potential subjects will approach you if they see someone else at your table
  - More potential subjects sign up if there are already names on signup sheet
Lessons Learning: Subject Recruitment

• Have something potential subjects can do there
  • Signup sheet >> “You should visit this website, and sign up there!”

• Have a shill
  • More potential subjects will approach you if they see someone else at your table (it's okay to give away candy to non-subjects)
  • More potential subjects sign up if there are already names on signup sheet
Lessons Learning: Subject Recruitment

- Make them come to you
  - Too easy to ignore if directly on path
Lessons Learning: Subject Recruitment

- Make them come to you
  - Too easy to ignore if directly on path
- Don't get discouraged