Improving the productivity of software developers

Lecture 5 - How to deliver recommendations?

Gail C. Murphy
University of British Columbia
Tasktop Technologies

@gail_murphy
recommendation delivery

• a recommender that generates useful recommendations but doesn’t get them to a developer isn’t very useful

• recommendation delivery is about the user interface that enables a developer to
  • be aware of the recommendations
  • determine if the recommendations have value
  • act upon a recommendation

information in this lecture is from joint work with Emerson Murphy-Hill
[Murphy-Hill & Murphy 2014]
why delivery matters...

You appear to be stating the obvious. Would you like help?
SE recommendation delivery that seems to work but not for every kind of recommendation…

```cpp
#include <iostream>
#include <string>
#include <list>
#include <vector>

namespace std;

int main(int argc, char *argv[]) {
    list<string> strings;
    string s;

    strings.push_back
    return 0

    void list::assign(size_type _n, const _Tp &_val)
    if void list::assign(_InputIterator __first, _InputIterator
    for reference list::back()
    _const_reference list::back()
    typed _Base::allocator_type _List_base::allocator_type
    _M void list::assign(_InputIterator __first, _InputIterator
    for reference list::back()
    _M iterator list::begin()
    _M const_iterator list::begin()
    _M void list::clear()
    _M void _List_base::clear()
    _M void _List_base::clear()
    typedef _List_iterator<_Tp, const _Tp&, const _Tp*> list::c
    typedef const value typedef list::const_pointer
```
#include <iostream>
#include <string>
#include <list>
#include <vector>

using namespace std;

int main(int argc, char *argv[])
{
    list<string> strings;
    string s;

    strings.

    return 0
        typedef _Base::allocator_type _List_base::allocator_type
        void list::assign(size_type __n, const _Tp &__val)
        f void list::assign(_InputIterator __first, _InputIterator
        v reference list::back()
        f const_reference list::back()
        _typedef _List_base<_Tp, _Alloc> list::_Base
        _iterator list::begin()
        v const_iterator list::begin()
        v void list::clear()
        v void _List_base::clear()
        v void _List_base::clear()
        _typedef _List_iterator<_Tp, const _Tp&, const _Tp*> list::c
        v typedef const value_type* list::const_pointer
five factors to consider

1. understandability
2. transparency
3. accessibility
4. trust
5. distraction

• let’s take a look at each factor and then consider some strategies for user interfaces for SE recommenders
understandability

• **obviousness**
  • how easy is it to recognize what the recommendation is?

• cognitive effort

Potential duplicates:
21556: applet viewing problems
19999: panel overlays are not displaying

[Murphy-Hill & Black 2010]
obviousness

Summary: KDE - The applet panel should not overlap applets

Summary

In the anarok2-svn contextview the bottom bar sometimes obscure applets
[1.4,11.1,11.2,11.3] Applets should not be larger than the viewable area
be given an appropriate sizehint.[11.2] This bug was fixed in 2.1.1[2]

Description
From mangus
1.1 version: svn (using Devel)
1.2 OS: Linux
1.3 Installed from: Compiled sources
1.4 In amarok2-svn I like the the new contextview, but I found the new bottom bar for managing
as it covers parts of other applets sometimes, like lyrics one, so that you miss a part of it.
1.5 Could be handy to have it appear and desappear onmouseover.
1.6 thanks

Potential duplicates:
21556: applet viewing problems
19999: panel overlays are not displaying
obviousness
understandability

• obviousness

• **cognitive effort**
  • how easy/hard is it to understand the meaning of a recommendation?

[Murphy-Hill & Black 2010]

[Zimmermann et al 2004]
understandability: how to achieve?

• use UI design heuristics from Nielsen
  • “match between system and real world”
    • use words, phrases and concepts a user has likely encountered previously
  • “consistency and standards”
    • use the same words, phrases and concepts to mean the same thing
  • “help and documentation”
    • make the system usable without help but provide it when needed

[Nielsen 2005]
transparency

• why is a recommendation being provided?

Potential duplicates:

[0.78] 21556: applet viewing problems

[0.43] 19999: panel overlays are not displaying

[Viriyakattiyaporn & Murphy 2010]
transparency: high

[Viriyakattiaporn & Murphy 2010]
transparency: low?
transparency: how to achieve?

• more information generally the better

• concrete rather than abstract

• “People who like X also use Y” may be better than “You may also like Y”
a user must be able to assess whether or not a recommendation is relevant and whether or not it should be acted upon

web page recommendations:
- if pages visited before might be easy to assess
  [Sawadsky et al 2013]
- if pages never seen, harder to assess
  [Sawadsky & Murphy 2011]
assessability: how to achieve?

• if a recommendation is an alternative (e.g., potential duplicate bug), make it easy to compare
• a user must trust a recommendation to be acted upon

autocomplete is easy to trust
trust: how to achieve it?

• Build it
  • start small and build?
  • allow to rate?

• Borrow it
  • refer to colleagues in recommendation

• Fake it
  • use principles of persuasion?
  • ‘this recommendation is based on the analysis of millions of lines of similar code’?
distraction

• when should a recommender make a recommendation?

all the time (ambient?)

all the time (interruptive)?

at natural break points?

what if the recommendation only applies at the moment?
distraction: how to reduce?

• negotiated interruption

  • inform a user a recommendation is available without forcing user to act immediately (or at all)

• attention sensitive alerting

  • infer when a user is not in the “middle of something”
strategies used in SE recommender user interfaces

1. getting a user’s attention
2. providing more information
getting a user’s attention

A. reactive (ask the recommender)
B. **proactive** (recommender initiates)
proactive:

annotations

mark-ups on program text to associate a recommendation with the segment of text they are displayed on

not necessarily good when recommendations are soft, imprecise or overlapping
proactive:
annotations as document splits
proactive: icons

small graphic images usually displayed on periphery (e.g., markers in the gutter of an editor)
proactive: affordance overlays

annotations that appear on top of user interface affordances (e.g., files in a browser)

[Maalej & Sahm 2010]
proactive: popup

appear in a new user interface layer on top of an existing user interface (e.g., popups)

a popup might force a user to acknowledge or may disappear after some time

[Carret & Dewan 2010]
proactive: dashboard

use of a fixed known location on the screen (typically at periphery of user’s vision) that allow a user to glance frequently and with low commitment.

[Murphy-Hill & Black 2010]
proactive: email notification

delivery of recommendation to a user’s email (e.g., Coverity)

recommendations need to be asynchronous

can involve others with cc

enables advanced workflows to manage notifications
providing more information
providing more information: textual

[ASIDE Explanation]

**ABSTRACT**

The return value of `getParameter()` at line 83 is vulnerable to be manipulated by malicious users.

**EXPLANATION**

When untrusted input gets into the application without proper validation and is used by critical operations, it may subvert the original semantics of that operation. For example, the following code dynamically constructs and executes a SQL query that searches for items matching a specified name based on the value that is passed by the client. The query restricts the items displayed to those where the owner matches the user name of the currently-authenticated user.

**REMEDIATION RECOMMENDATION**

The best practice to avoid introducing the aforementioned vulnerabilities into your code is to validate all the values that are passed into your application. One option is to validate the input against established Regular Expression.

[Xie et al 2012]
providing more information: transformative

show the impact of taking a recommendation

```
public static int fullSize()
{
    int size = 0;
    for(String s : list)
    {
        size = findSize(size, s);
    }
    return size;
}
```

```
public static int findSize(int size, String s)
{
    int string_size = s.length();
    size = string_size;
    return size;
}
```

preview of what will happen if a refactoring is used

[Foster et al 2012]
providing more information: visualization

[Holten 2006]
I need a UI for my recommender, what do I do?

• proof-of-concept stage - maybe no UI

• need some UI to evaluate your recommender, maybe do some mockups and cognitive walkthroughs of a UI that draws on other existing UIs for inspiration

• how do I evaluate my UI?
  
  • wizard-of-oz or heuristic evaluation
  
  • A/B testing
  
  • controlled experiments
  
  • case studies
iteration is key
summary

• the user interface greatly affects use (or non-use) of a recommender

• can be tricky to separate UI from recommender

• discussed various factors that affect a UI and some ways a UI might be realized

• next time we will conclude by talking about evaluation strategies for determining if a recommender is really useful
references

Carter & Dewan. Design, implementation and evaluation of an approach for determining when programmers are having difficulty. CSCW 2010.


Ge, DuBose & Murphy-Hill. Reconciling manual and automatic refactoring. ICSE 2012.


Sawadsky & Murphy. Fishtail: from task context to source code examples. Workshop on Developing Tools as Plug-ins, 2011.

Viriyakattiyaporn & Murphy. Improving program navigation with an active help system. CASCON 2010.

Xie, Lipford & Chu. Evaluating interactive support for secure programming. CHI 2012.