getting to flow in software development

Gail C. Murphy

University of British Columbia
Tasktop Technologies Incorporated
“software is eating the world”

Marc Andreessen
premise

... lots of software to build
should be fun to build it
hypothesis

... tools and interfaces that support flow enable building more software with more fun
what is **flow**?
Mihaly Csikszentmihalyi
single-minded immersion

clear set of goals and progress
clear and immediate feedback
balance of challenges to skills
top 3 reasons for productive day (379 devs)

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>53.2%</td>
<td>complete tasks or goals</td>
</tr>
<tr>
<td>50.4%</td>
<td>getting into a &quot;flow&quot; without many &quot;context switches&quot; and no or few interruptions and distractions</td>
</tr>
<tr>
<td>21.9%</td>
<td>no meetings</td>
</tr>
</tbody>
</table>

[Meyer, Fritz, Murphy & Zimmermann, FSE 2014]
constant switching

[Meyer, Fritz, Murphy & Zimmermann, FSE 2014]
ok?

development tools and interfaces already do a great job at supporting flow
jHotDraw
**set the fill colour of a figure**

**Details**
- **Type:** Story
- **Priority:** Major
- **Status:** TO DO
- **Resolution:** Unresolved
- **Labels:** None

**Description**
As a drawing creator, I need to set the fill colour of a figure on a canvas.

A variety of fill colours must be provided. It should be simple for the drawing creator to select a figure and from the selected figure, see and choose an appropriate colour.

**Sub-Tasks**

<table>
<thead>
<tr>
<th>No.</th>
<th>Task Description</th>
<th>Status</th>
<th>Assignee</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>allow selection of a figure</td>
<td>TO DO</td>
<td>Gail Murphy</td>
</tr>
<tr>
<td>2.</td>
<td>add a menu for setting fill colour of a figure</td>
<td>TO DO</td>
<td>Gail Murphy</td>
</tr>
<tr>
<td>3.</td>
<td>draw the fill colour of a figure</td>
<td>DONE</td>
<td>Gail Murphy</td>
</tr>
</tbody>
</table>
package org.jhotdraw.samples.javadraw;

import org.jhotdraw.framework.*;

/** @version <$CURRENT_VERSION$>
 *
 * public class JavaDrawApp extends MDI_DrawApplication {
 *
 * private Animator fAnimator;
 * private static String fgSampleImagesPath = '/org/jhotdraw/samples/javadraw/sam'
 * private static String fgSampleImagesResourcePath = fgSampleImagesPath + '/';
 *
 * JavaDrawApp() {
 *   super("JHotDraw");
 * }
 *
 * /**
 * * Expose constructor for benefit of subclasses.
 * */
 * @param title The window title for this application's frame.
 * */
 * public JavaDrawApp(String title) {
 *   super(title);
 * }
 * ...
/**
 * Draws the figure in the given graphics. Draw is a template
 * method calling drawBackground followed by drawFrame.
 */
public void draw(Graphics g) {
    Color frame = getFrameColor();
    if (!ColorMap.isTransparent(frame)) {
        g.setColor(frame);
        drawFrame(g);
    }
}

/**
 * Draws the background of the figure.
 * @see #draw
 */
protected void drawBackground(Graphics g) {
add code in several places
bug
```java
protected AttributeFigure() {

    // Draws the figure in the given graphics. Draw is a template method calling drawBackground followed by drawFrame.

    public void draw(Graphics g) {
    }
}
```
mismatch #1

flat information space
mismatch #2
multiple disconnected information
mismatch #3: information overload
mismatch #4: clickitis over cognition
mismatches

#1: flat information space

#2: multiple disconnected information spaces

#3: information overload

#4: clickitis over cognition

goal
an example of how it could be

approaches and tools that address the mismatches
As a drawing creator, I need to set the fill colour of a figure on a canvas.

A variety of fill colours must be provided. It should be simple for the drawing creator and from the selected figure, see and choose an appropriate colour.
This completed task is similar and might be helpful.
<table>
<thead>
<tr>
<th>Description</th>
<th>Category</th>
<th>#E</th>
</tr>
</thead>
<tbody>
<tr>
<td>declared fields</td>
<td>declaration</td>
<td>10</td>
</tr>
<tr>
<td>subclasses</td>
<td>inheritance</td>
<td>9</td>
</tr>
<tr>
<td>superinterfaces</td>
<td>inheritance</td>
<td>3</td>
</tr>
<tr>
<td>casts to type</td>
<td>inter-class</td>
<td>1</td>
</tr>
<tr>
<td>fields of type</td>
<td>inter-class</td>
<td>1</td>
</tr>
<tr>
<td>references to type</td>
<td>inter-class</td>
<td>13</td>
</tr>
<tr>
<td>fields used</td>
<td>intra-class</td>
<td>14</td>
</tr>
<tr>
<td>methods called</td>
<td>intra-class</td>
<td>50</td>
</tr>
<tr>
<td>types referenced</td>
<td>intra-class</td>
<td>25</td>
</tr>
</tbody>
</table>
/**
 * Draws the figure in the given graphics. Draw drawBackground followed by drawFrame.
 */

public void draw(Graphics g) {
    Color frame = getFrameColor();
    if (!ColorMap.isTransparent(frame)) {
        g.setColor(frame);
        drawFrame(g);
    }
}
/**
 * Draws the figure in the given graphics. Draw
 * drawBackground followed by drawFrame.
 */

public void draw(Graphics g) {
    Color frame = getFrameColor();
    if (!ColorMap.isTransparent(frame)) {
        g.setColor(frame);
        drawFrame(g);
    }
}
/**
 * Draws the figure in the given graphics. Draw
drawBackground followed by drawFrame.
*/

public void draw(Graphics g) {
    Color fill = getFillColor();
    if (ColorMap.isTransparent(fill)) {
        g.setColor(fill);
        drawBackground(g);
    }
    Color frame = getFrameColor();
    if (!ColorMap.isTransparent(frame)) {
        g.setColor(frame);
        drawFrame(g);
    }
}
[Ko and Myers, 2008]
techniques for improved flow

- task context
- recommendations
- dialog
- summarization
technique #1 of 4: task context

- parts and relationships of artifacts relevant to developer as they work on a task
- different approximations
  - e.g., commits
  - e.g., interaction history (Eclipse Mylyn)

[Kersten & Murphy, 2006]
task context in Mylyn

[Kersten & Murphy, 2006]
if we have task context...

- have a mapping between “intent” and parts of artifacts
- can use as a concept mapping
- can use as basis for recommending artifacts to look at
- could use to analyze defects based on ‘intent”
- reduce information space shown to the developer
- share work
achieving flow with task context...

• capture without requiring work on the part of the developer

• challenge: how to get broader intent?

• challenge: how to get all parts of artefacts worked on or looked at?

• challenge: how to infer task switches automatically?
technique #2 of 4: recommendations

- suggest parts of current or past information spaces that might be helpful for current work

- e.g., text similarity on previously solved bugs [Čubranić & Murphy 2003]

- e.g., where others have navigated from a given point [DeLine et al. 2005]
This completed task is similar and might be helpful.
recommendations in Hipikat

[Čubranić & Murphy, 2003]
if we have recommendations...

- can eliminate low-level searching and navigating
- can reduce incorrect paths followed
- can help if developer is stuck
- can provide information a developer might miss
achieving flow with recommendations...

- challenge: appropriate effective user interface delivery
- challenge: what are sufficient precision and recall for various tasks
- challenge: explaining recommendations
technique #3 of 4: dialog

- make units of discourse between developer and interface dialog
- task-specific
  - e.g., Whyline for debugging [Ko & Myers 2004 & 2008]
  - e.g., Ferret for code navigation [de Alwis & Murphy 2008]
Ferret
navigation

Whyline
debugging
if we have dialog...

- can eliminate low-level searching and navigating
- can reduce incorrect paths followed
- may be able to make it easier for broader range of people to program effectively
achieving flow with dialog...

- challenge: can we make general approaches for broad task categories?
- challenge: what are appropriate user interaction flows for dialog?
- challenge: what if a question isn’t presented that needs to be asked?
technique #4 of 4: summarization

- present abstracted form of information
- extractive or abstractive
- e.g., bugs [Rastkar & Murphy, 2010, 2013]
- e.g., code [Moreno 2013]
/**
 * Draws the figure in the given graphics. Draw
 * drawBackground followed by drawFrame.
 */

public void draw(Graphics g) {
    Color frame = getFrameColor();
    if (!ColorMap.isTransparent(frame)) {
        g.setColor(frame);
        drawFrame(g);
    }
}
bug summarization

Epic
Story
Task

use machine learning to learn and extract appropriate sentences

extracted summary

[Rastkar & Murphy, 2010 & 2012]
if we have summarization...

- developer can make more informed decisions as timely effective access to information
- can look for similarities and trends at different level of abstraction
- keep developer focused on task by eliminating cognitive switches
achieving flow with summarization

- challenge: what to summarize when?
- challenge: how much intent needs to be known to summarize effectively?
- challenge: how accurate must a summary be not to be mis-leading?
interaction for flow

<table>
<thead>
<tr>
<th>mismatches</th>
<th>interaction types</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. flat information space</td>
<td>task context</td>
</tr>
<tr>
<td>2. multiple disconnected information spaces</td>
<td>recommendation</td>
</tr>
<tr>
<td>3. information overload</td>
<td>dialog</td>
</tr>
<tr>
<td>4. clickitis over cognition</td>
<td>summarization</td>
</tr>
</tbody>
</table>
what should the programming tools of the future look like?

is it enough in research to produce approaches that fill information spaces?
john anvik
elisa baniassad
wesley coelho
davor cubranic
brian de alwis
rob elves
thomas fritz
reid holmes
rahul jiresal
mik kersten
shawn minto
e murphy-hill
jingwen ou
martin robillard
sarah rastkar
nick sawadsky
david shepherd
ducky sherwood
annie ying
robert walker
and many others!
software is eating the world

interaction styles that might improve flow
- task context
- recommendation
- dialog
- summarization

what might programming environments look like to make programming more fun and more inclusive

Gail C. Murphy
@gail_murphy