Unifying Artifacts and Activities
in a Visual Tool for Distributed Software Development Teams

Jon Froehlich and Paul Dourish
Interactive & Collaborative Technologies Group
School of Information and Computer Science
University of California, Irvine
Overview

• Software development is complex
  – Involves dealing with complexity of source code (artifact)

• Distributed software development is even more complex
  – Involves dealing with both complexity of artifact and activities around that artifact

• These two sources of complexity are not independent
  – Our approach seeks to combine them in a familiar visual frame
class String implements Comparable
{
    String(char value[]){
    }
    String substring(int index){
    }
    public int compareTo(Object o){
    }
}
String.java

```java
class String implements Comparable
{
    String(char value[]){
        //
    }

    String substring(int index){
        //
    }

    public int compareTo(Object o){
        //
    }
}
```

Artifact-Based Tools

Amy

Frank

Daniel

Jan '00

August '02

March '04

July '02

August '02

74%
Activity-Based Tools

String.java

```java
class String implements Comparable
{
    String(char value[]){
    }

    String substring(int index){
    }

    public int compareTo(Object o){
    }
}
```

March '04

August '02

July '02

March '04
Separation

• This separation makes it difficult to see elements that involve both artifact and activity:
  – Who has been working on new sections of code?
  – Which developers have worked together recently and on what functions/files?
• or more complicated questions like
  – What modules depend on those recently updated
Our Approach

• To address this separation, we’ve developed a visualization tool called Augur.
  – Augur uses the source code itself to reveal information about development activity

• The source code becomes a unifying principle with which to reveal information about its development
Our Approach

- Augur relies on existing configuration management systems for its activity data
  - Currently supports CVS
- Augur performs language/structural analysis to better understand semantics of code
Visualizations

• Visualizations shift load from cognitive system to perceptual system
  – Leverage visual system’s ability to perceive patterns and structures

• Visualizations allow user to process large amounts of information in a consumable way
HelloWorld.java

public class HelloWorld {

    /* Default constructor */

    public HelloWorld() {
        Log.println("Constructor called");
    }

    public void printMessage() {
        Log.println("printMessage called");
        System.out.println("HelloWorld");
    }

}
This is a test class to demo Augur!

@authors
Jon, Sandy, Edward

```
public class HelloWorld {

    // Default constructor

    public HelloWorld() {
        Log.println("Constructor called");
    }

    // Prints HelloWorld msg & logs call

    public void printMessage() {
        Log.println("printMessage called");
        System.out.println("HelloWorld");
    }
}
```
public class HelloWorld {

    public HelloWorld() {
        Log.println("Constructor called");
    }

    public void printMessage() {
        Log.println("printMessage called");
        System.out.println("HelloWorld");
    }

    //end HelloWorld class
}
This is a test class to demo Augur!

@authors
Jon, Sandy, Edward

public class HelloWorld {

/**
 * Default constructor
 */
 public HelloWorld()
 {
  Log.println("Constructor called");
 }

/**
 * Prints HelloWorld msg & logs call
 */
 public void printMessage()
 {
  Log.println("printMessage called");
  System.out.println("HelloWorld");
 }
}

Big Picture: on March 6th, Sandy added logging functionality to HelloWorld. How?
- Adding a constructor
- Adding one line to an existing method

Augur

- Source code is the common artifact around which developer activities take place
  - The code itself provides a common & familiar space in which to reveal information about development activity
  - Code becomes a unifying structure for organizing many different types of information
Augur in Practice

- Three examples follow that demonstrate how relationships between artifact and activity can be interpreted with Augur
Three Examples

1. The activity of commenting
It appears that there are relationships between temporality and artifact!

First the methods were added and then, 6 months later, the comments were added/modified.

Now we see, most recently added lines are comments…

And comments were added by a another author!

Not just detecting comments, but the activity of commenting!

Clearly there is a temporal pattern here, but…

This richer characterization made possible by the combination of information.
Author Activity
Author Activity

- Majority of author’s development activity is commenting
  - This is easily discernible
- Tying together multiple views makes this inference possible
Three Examples

1. The activity of commenting
2. Unification of views reveal richer notions of activity
Unification of Views
Three Examples

1. The activity of commenting
2. Unification of views reveal richer notions of activity
3. Exploring changes in context
Controls and Color Legend

Primary Visualization Window
(Augmented Seesoft View)

Secondary Visualizations
Architecture
Extensibility

- Supports
  1. Multiple Version Control Systems
  2. Multiple analytic tools
  3. Multiple visualizations
Ongoing & Future Work

- Exploring temporal activity patterns in source code
- Exploring social networking patterns in source code
- Applying Augur philosophy in other environments
Temporal Patterns
Social Patterns
Integrating Augur
Philosophy
Conclusion

• Activity information is situated within the source code
  – Source code is an “inhabited space”
• Exposing activity information in the context of code provides a richer understanding of the relationship between them
Thank You

Download Augur:
http://www.isr.uci.edu/projects/augur

Contact :
jfroehli@ics.uci.edu
jpd@ics.uci.edu