Tool Support for Working With Large Systems

ECOOP PPPL Workshop
Jonathan Sillito, Kris De Volder

Tool support for working with large systems

Presented at Workshop on Practical Problems of Programming in the Large at ECOOP 2004.

Performing a change task on a large system can be very difficult, simply because of the volume of data. A task will often involve only a small subset of that data, but the size of the system can still make performing the task difficult.
Managing Complexity

• Careful modularization considered key to managing complexity (*Divide and Conquer*)
• Localizes concerns
• Parnas claims that the expected benefits are greater **flexibility** and **comprehensibility**
• However, only one decomposition is possible!

A standard practice for managing this complexity is to carefully decompose the system, producing modules that are cohesive and loosely coupled with other modules.

Parnas claims that the benefits of an effective decomposition are greater **flexibility** and greater **comprehensibility**. Flexibility comes as modules can be modified independently. Comprehensibility comes because modules provide an abstract view of the system and each module can be understood more or less in isolation.

So given a change task, ideally, you would only need to look at one or two modules and you would easily understand what you needed to understand and easily make the appropriate change.

However, only one decomposition is possible and so some concerns are well modularized at the expense of leaving the rest scattered and tangled. Tasks based on these are much more difficult. With respect to those tasks, the system just is not very flexible or comprehensible.
Non Localized Tasks are Hard

• Hard to identify the relevant subset of the system
• Hard to build and maintain a mental model of that subset
• Hard to work with that subset

*Given an enormous amount of stuff, and some task to be done using some of the stuff, what is the relevant stuff for the task? [C. Glymour 1987]*

For a given task there is some subset of the entire system that needs to be understood and worked with. Even for a well modularized system, for many tasks this will not be a localized portion of the system. So these tasks are difficult because, when faced with a large sea of objects or classes, it is:

• hard to identify the correct subset,
• hard to build and maintain a mental model of that subset, and
• hard to focus on or work with that subset (significant overhead caused by navigation and context switching).
Is the User Interface Part of the Problem?

How much of the problem is simply an artifact of the very limited presentation and interaction provided by the editing environment?

Our Focus

• Supporting non-localized tasks
• Editing environment

Is the interface part of the problem? Or more precisely, how much of the problem is simply an artifact of the very limited \textit{presentation} and \textit{interaction} provided by the editing environment?

So the problem we are looking at is the difficulty of change tasks based on non-localized concerns in large systems. And in particular the lack of support for this kind of work in current editing environments, because no matter what other tools and architectural diagrams are available, the developer eventually finds himself or herself in an editing environment to understand many types of concerns and to perform most change tasks. Here is a screen shot of an editing environment.
What are the problems here? First, the interface gives strong notion of locality. A developer can't simultaneously see very much source code (60 lines maybe) and only those things that are close together in a file.

The developer can't see what is related or how things are related. Picking out things that participate in the concern is a tedious and error prone process. During the discovery process all of the details and relationships need to be stored in the developer's head.

Even once the participants of the concern have been identified, there is no way to tell the development environment what parts are of interest. No way to focus on the appropriate subset. Instead, there is a significant navigation burden as the developer switches (using tabs or hyperlinks or searches) between the various files and parts of files needed to do the task.

A small and an inflexible window on a large space.
As an analogy, suppose you are asked to figure out what this is that we are looking at. You only get to look at small pieces of it like this. You can see a certain amount of information clearly, but this is only local, nothing else is visible.
Here is another small piece. Of course now you can see nothing of the previous stuff and nothing about how what you are looking at now relates to the previous information.
Here is another piece. So what is it?
Its Claude Monet’s Water Lilies. When you only see pieces its difficult to understand the whole.

Similarly editing environments, don’t make it easy to work with and understand a non-localized concern in the source code. They don’t provide the big picture.

It is not my intention to provide a solution with you, instead I want to talk about some principles or ideas that I believe could contribute to a solution.
Thoughts on a Solution

• Use larger displays or more displays
• Show more information, context, etc

To clarify, add detail. [E. Tufte, 1990]

How can the interface help? One simple thing is to utilize larger displays. This multi-display setup consists of 3 18-inch LCD monitors for $3,000 USD.

Though honestly, current editing environments don’t even make effective use of standard displays, I believe they should be designed to aggressively show more information, context etc, as a way to help the developer understand what is need for the task.

What information should be displayed on that screen real estate? There is lots of information that from lots of source, including static analysis and dynamic trace information, and of course there is the source code itself.
This is about 400 lines of code shown with a 4pt font. This is using a uniform scaling to show lots of information, however it isn’t particularly useful since it is all too small to read easily and definitely to small to edit. Also it is not enough of the right stuff since a I claim that a particular task will be concerned with one or two methods here and one or two methods there ...
protected TableIdentifier[] toArray(TableIdentifier table)
{
    TableIdentifier[] tables = null;
    if(null != table) {
        tables = new TableIdentifier[1];
        tables[0] = table;
    } else {
        tables = new TableIdentifier[0];
    }
    return tables;
}

Instead a less uniform scaling could be used where, for example the body of a method could be scaled smaller than its signature. In some cases the body could be elided all together.
package org.axiondb;
import ...

public interface Constraint extends Serializable {
    void resolve(Database db, TableIdentifier table) throws AxionException;
    boolean evaluate(RowEvent event) throws AxionException;
    boolean isDeferred();
    void setDeferred(boolean deferred) throws AxionException;
    boolean isDeferrable();
    void setDeferrable(boolean deferrable);
    String getName();
    void setName(String name);
    String getType();
}

Something similar can be done with classes and interfaces and even whole packages.

Now ideally the right stuff is clearly visible while the rest is smaller or elided completely, for example a task may involve ten different methods in 10 different classes. Ideally the display of source code should prefer to give screen real estate to these ten methods and possibly the fields they use, rather than the other methods in those same classes.

Of course, the editing environment can’t with perfection know exactly what is important …
A space allocation scheme based on degree of interest function may help. The function could consider various factors such as the currently selected entity, relationships between entities and the developer’s navigation or editing history.
More Thoughts on a Solution

• Scaling will draw attention to different entities
• Relationship highlighting or other mechanisms can communicate why something is important and help form mental model

So scaling will draw attention to different entities and hopefully the right entities. A small scale example could be if I am editing a method f() which calls a method g(), possibly both f() and g() could be displayed more prominently.

Other mechanisms could be used to communicate relationships between these entities to help the developer build and maintain a mental model of the concern.
One simple example of this could be background highlighting as in this picture here. Suppose the entity in blue is the currently selected entity, those in highlighted in green are callers of the currently selected entity, and those highlighted in yellow are callees off the selected entity.

As selection changes, so will the highlighting but I suspect that if the highlights change or fade gradually rather than abruptly the developer will be better able to maintain a mental model of the concern.
More Thoughts on a Solution

- Task based focus
- Auto recommendations

_The greatest value of a picture is when it forces us to notice what we never expected to see._

[J. Turkey 1977]

A couple quick last thoughts on a solution to this problem. I believe that as systems get larger, a task based focus is more and more important for developers. A developer should be able to iteratively build up a view that directly supports the current task. However, focusing too narrowly can be dangerous. One danger is that the developer may miss something that is important, another is that the developer may not correctly understand how the subset currently in focus relates to the rest of the system.

I wonder if perhaps something like automatic recommendations could help avoid this problems.
Conclusion

- Many change tasks on large systems are hard because editing environments make it difficult to understand and work with non-localized subsets of the system.
- Ideas for a solution: show more information, emphasize relationships, allow a task based focus, provide important context information in the form of automatic recommendations.

In conclusion, many change tasks on large systems are hard because editing environments make it difficult to understand and work with non-localized subsets of the system.

I have presented a few ideas that could contribute to a more effective editing environment, including: show more information, emphasize relationships, allow a task based focus, provide important context information in the form of automatic recommendations.