A Study of Structure in Programmers’ Text Editing

The Natural Programming group at CMU is designing several novel tools to help programmers of a variety of expertise. Today I’ll be talking about efforts to help design new interaction techniques for editing code.
A Study of Structure in Programmers’ Text Editing

Andrew Ko
Htet Htet Aung
Brad Myers

Human-Computer Interaction Institute
Carnegie Mellon University

The Natural Programming group at CMU is designing several novel tools to help programmers of a variety of expertise. Today I’ll be talking about efforts to help design new interaction techniques for editing code.
Two ways to edit code...

As a sequence of characters

As structures

There are basically two interaction techniques for editing code. The first is as a sequence of characters. For example, here you see a programmer using the Eclipse editor for Java. Text is has a lot of advantages: you can type anything in any order, using a variety of interaction techniques. Learning syntax is still a big problem for novices, and many tools lose their helpfulness in the presence of syntax errors.

Structured editors help avoid these syntactic problems, but they are particularly inflexible when it comes to modifying code after you’ve written it.

These basically represent two ends of a continuum. So the question is, how much structure is there to the way that programmers edit code? If it’s somewhere in the middle, we may be able to design editors that have the benefits of both of these two extremes. To find out, we performed a feasibility study.
There are basically two interaction techniques for editing code. The first is as a sequence of characters. For example, here you see a programmer using the Eclipse editor for Java. Text is has a lot of advantages: you can type anything in any order, using a variety of interaction techniques. Learning syntax is still a big problem for novices, and many tools lose their helpfulness in the presence of syntax errors.

Structured editors help avoid these syntactic problems, but they are particularly inflexible when it comes to modifying code after you’ve written it.

These basically represent two ends of a continuum. So the question is, how much structure is there to the way that programmers edit code? If it’s somewhere in the middle, we may be able to design editors that have the benefits of both of these two extremes. To find out, we performed a feasibility study.
There are basically two interaction techniques for editing code. The first is as a sequence of characters. For example, here you see a programmer using the Eclipse editor for Java. Text is has a lot of advantages: you can type anything in any order, using a variety of interaction techniques. Learning syntax is still a big problem for novices, and many tools lose their helpfulness in the presence of syntax errors.

Structured editors help avoid these syntactic problems, but they are particularly inflexible when it comes to modifying code after you’ve written it.

These basically represent two ends of a continuum. So the question is, how much structure is there to the way that programmers edit code? If it’s somewhere in the middle, we may be able to design editors that have the benefits of both of these two extremes. To find out, we performed a feasibility study.
Two ways to edit code...

How much structure is there to the way programmers edit code?

There are basically two interaction techniques for editing code. The first is as a sequence of characters. For example, here you see a programmer using the Eclipse editor for Java. Text is has a lot of advantages: you can type anything in any order, using a variety of interaction techniques. Learning syntax is still a big problem for novices, and many tools lose their helpfulness in the presence of syntax errors.

Structured editors help avoid these syntactic problems, but they are particularly inflexible when it comes to modifying code after you’ve written it.

These basically represent two ends of a continuum. So the question is, how much structure is there to the way that programmers edit code? If it’s somewhere in the middle, we may be able to design editors that have the benefits of both of these two extremes. To find out, we performed a feasibility study.
The Study

• Analyzed 700 minutes of programmers of varying expertise using Java in Eclipse

• Programmers worked on several debugging and maintenance tasks

• Identified 2770 individual text edits for analysis
  insert, backspace, cut, paste, undo, select, code completion, etc.

We studied 700 minutes of Java programmers’ character-by-character edits in the Eclipse programming environment.

Eclipse is a widely used Java editor that supports auto-formatting, indenting, and several other text-editing features.

We found that contrary to most assumptions, that there’s a lot of structure to the way people edit code. We’ll discuss a few of our findings in this talk.
“Keyword” Structures

One of the first things that comes to mind when talking about structured editors is the example of changing a while loop to an if statement. With plain text, this just requires a single keyword change. With structured editors, the whole while statement has to be removed and replaced with a new if statement.

What we found, however, is that the programmers we observed never performed this edits. When they did edit “keyword” structures, they removed them and replaced with something else rather than modify them. This was usually because the change they were making required an entirely different solution entirely different solution.

So what did programmers actually do?
Programmers didn’t edit this way.

When “keyword” structures were edited, they were always removed and replaced with something else.

One of the first things that comes to mind when talking about structured editors is the example of changing a while loop to an if statement. With plain text, this just requires a single keyword change. With structured editors, the whole while statement has to be removed and replaced with a new if statement.

What we found, however, is that the programmers we observed never performed this edits. When they did edit “keyword” structures, they removed them and replaced with something else rather than modify them. This was usually because the change they were making required an entirely different solution entirely different solution.

So what did programmers actually do?
Names

43% of all edits

Many name edits changed internal structure in a name.

Almost half of programmers edits involved creating, modifying and removing names, such as variable references or declarations.

13% of these were edits to some internal structure of the name, as part of copy and paste or renaming. This is something that all code editors could take advantage of, but none currently do.

We also found that 25% of all name edits resulted in references to undeclared names. Structured editors typically require something to exist before it can be referred to, but this data suggests that this flexibility is a important requirement for structured editors.

The screen shots and videos in the talk come directly from the study.
Names

43% of all edits

Many name edits changed internal structure in a name.

Almost half of programmers edits involved creating, modifying and removing names, such as variable references or declarations.

13% of these were edits to some internal structure of the name, as part of copy and paste or renaming. This is something that all code editors could take advantage of, but none currently do.

We also found that 25% of all name edits resulted in references to undeclared names. Structured editors typically require something to exist before it can be referred to, but this data suggests that this flexibility is a important requirement for structured editors.

The screen shots and videos in the talk come directly from the study.
Four basic operations

Inserting and removing items from lists.

Wrapping and unwrapping expressions.

There were other four major types of edits. Many were simple insert and remove operations on lists. For example, here’s a programmer inserting variable names into a list of variable declarations.

The other two major types of edits were wrapping and unwrapping expressions. For example, here a programmer unwraps two expressions by removing the parentheses and division operator and then wraps them with a different operator.

There were several variations in the particular way that these edits were performed, but they were largely based on the language syntax. However, because the basic operations were so universal, structured editors could potentially provide generic interaction techniques for these four operations, and in many cases, avoid a lot of the overhead.
Four basic operations

Inserting and removing items from lists.

Wrapping and unwrapping expressions.

There were other four major types of edits. Many were simple insert and remove operations on lists. For example, here’s a programmer inserting variable names into a list of variable declarations.

The other two major types of edits were wrapping and unwrapping expressions. For example, here a programmer unwraps two expressions by removing the parentheses and division operator and then wraps them with a different operator.

There were several variations in the particular way that these edits were performed, but they were largely based on the language syntax. However, because the basic operations were so universal, structured editors could potentially provide generic interaction techniques for these four operations, and in many cases, avoid a lot of the overhead.
Four basic operations

Inserting and removing items from lists.

Wrapping and unwrapping expressions.

There were other four major types of edits. Many were simple insert and remove operations on lists. For example, here’s a programmer inserting variable names into a list of variable declarations.

The other two major types of edits were wrapping and unwrapping expressions. For example, here a programmer unwraps two expressions by removing the parentheses and division operator and then wraps them with a different operator.

There were several variations in the particular way that these edits were performed, but they were largely based on the language syntax. However, because the basic operations were so universal, structured editors could potentially provide generic interaction techniques for these four operations, and in many cases, avoid a lot of the overhead.
Comments

37% of comments were **annotations**

63% of comments were **exclusions**

Although not very common, programmers used comments in interesting ways.

A third of these were annotations. They were inserted in arbitrary places: above, below, left and right of statements, but typically they were added so the layout of the code wouldn’t be disturbed.

The more common use of comments was to exclude part of the program from execution. For example, here’s a programmer by commenting out a whole for loop. The interesting thing about exclusions is that in many cases, programmers still needed support, such as code completion, navigating to the definition of a variable, but Eclipse couldn’t provide it, because comments are ignored by the parser.
Comments

37% of comments were annotations

63% of comments were exclusions

Although not very common, programmers used comments in interesting ways.

A third of these were annotations. They were inserted in arbitrary places: above, below, left and right of statements, but typically they were added so the layout of the code wouldn’t be disturbed.

The more common use of comments was to exclude part of the program from execution. For example, here’s a programmer by commenting out a whole for loop. The interesting thing about exclusions is that in many cases, programmers still needed support, such as code completion, navigating to the definition of a variable, but Eclipse couldn’t provide it, because comments are ignored by the parser.
Comments

37% of comments were **annotations**

63% of comments were **exclusions**

Although not very common, programmers used comments in interesting ways.

A third of these were annotations. They were inserted in arbitrary places: above, below, left and right of statements, but typically they were added so the layout of the code wouldn’t be disturbed.

The more common use of comments was to exclude part of the program from execution. For example, here’s a programmer by commenting out a whole for loop. The interesting thing about exclusions is that in many cases, programmers still needed support, such as code completion, navigating to the definition of a variable, but Eclipse couldn’t provide it, because comments are ignored by the parser.
Undo wasn’t used for undoing; in fact, it was very rare that programmers accidentally edited text and needed to undo it.

Instead, it was used for inspecting recent versions of code, because it was the only tool in the environment keeping a history. For example, this programmer had recently replaced a for loop with this line of code, and needed to refer back to it.

To actually undo mistakes, programmers used the backspace key exclusively, both for typos, and when they changed their mind about some decision. Using backspace as undo was actually quite versatile, since it didn’t depend on the editing history, like the undo command.
Undo

Undo was used to refer to old versions of code.

Backspace was used to “undo.”

Undo wasn’t used for undoing; in fact, it was very rare that programmers accidentally edited text and needed to undo it.

Instead, it was used for inspecting recent versions of code, because it was the only tool in the environment keeping a history. For example, this programmer had recently replaced a for loop with this line of code, and needed to refer back to it.

To actually undo mistakes, programmers used the backspace key exclusively, both for typos, and when they changed their mind about some decision. Using backspace as undo was actually quite versatile, since it didn’t depend on the editing history, like the undo command.
Undo

Undo was used to refer to old versions of code.

Backspace was used to “undo.”

Undo wasn’t used for undoing; in fact, it was very rare that programmers accidentally edited text and needed to undo it.

Instead, it was used for inspecting recent versions of code, because it was the only tool in the environment keeping a history. For example, this programmer had recently replaced a for loop with this line of code, and needed to refer back to it.

To actually undo mistakes, programmers used the backspace key exclusively, both for typos, and when they changed their mind about some decision. Using backspace as undo was actually quite versatile, since it didn’t depend on the editing history, like the undo command.
98% of programmers’ selections obeyed structured boundaries. For example, here a programmer is selecting multiple expressions. Unfortunately, the interaction technique selected on character boundaries, making structural selection difficult.

Many of the selections we observed were of multiple discontiguous fragments. For example, here a programmer is removing a for loop header and closing brace. The interaction technique required the programmer to divide what could have been a single selection of the for loop into two separate selections. Structured editors may be able to offer new types of selection interaction techniques that better match these types of edits.
Text Selection

98% of selections obeyed structural boundaries. But the editor obeyed character boundaries.

Many selections were of multiple discontiguous fragments.

98% of programmers’ selections obeyed structured boundaries. For example, here a programmer is selecting multiple expressions. Unfortunately, the interaction technique selected on character boundaries, making structural selection difficult.

Many of the selections we observed were of multiple discontiguous fragments. For example, here a programmer is removing a for loop header and closing brace. The interaction technique required the programmer to divide what could have been a single selection of the for loop into two separate selections. Structured editors may be able to offer new types of selection interaction techniques that better match these types of edits.
98% of programmers’ selections obeyed structured boundaries. For example, here a programmer is selecting multiple expressions. Unfortunately, the interaction technique selected on character boundaries, making structural selection difficult.

Many of the selections we observed were of multiple discontiguous fragments. For example, here a programmer is removing a for loop header and closing brace. The interaction technique required the programmer to divide what could have been a single selection of the for loop into two separate selections. Structured editors may be able to offer new types of selection interaction techniques that better match these types of edits.
Although there we did observe unstructured edits, they were very rare.

The majority were typos, and in many cases worsened by Eclipse’ automatic formatting. For example, here’s a programmer trying to type a name, and accidentally types an open brace.

Many were just the temporary removal of delimiters. For example, here a programmer removes the left part of a new expression, then removes the right. Many of these could be replaced by unwrap operators.

The last example is where two or more structural selections were interleaved. For example, here is a programmer interleaving the creation of a block for an if statement, and pasting a list of statements. In all of these editing situations, the purpose of the interleaving seemed to because of proximity; the right brace needed to be placed at the end, and the insertion point
Unstructured Edits

Typos

Temporary removal of delimiters

Two or more structured edits interleaved.

Although there we did observe unstructured edits, they were very rare.

The majority were typos, and in many cases worsened by Eclipse’ automatic formatting. For example, here’s a programmer trying to type a name, and accidentally types an open brace.

Many were just the temporary removal of delimiters. For example, here a programmer removes the left part of a new expression, then removes the right. Many of these could be replaced by unwrap operators.

The last example is where two or more structural selections were interleaved. For example, here is a programmer interleaving the creation of a block for an if statement, and pasting a list of statements. In all of these editing situations, the purpose of the interleaving seemed to because of proximity; the right brace needed to be placed at the end, and the insertion point
Unstructured Edits

Typos

Temporary removal of delimiters

Two or more structured edits interleaved.

Although there we did observe unstructured edits, they were very rare.

The majority were typos, and in many cases worsened by Eclipse’ automatic formatting. For example, here’s a programmer trying to type a name, and accidentally types an open brace.

Many were just the temporary removal of delimiters. For example, here a programmer removes the left part of a new expression, then removes the right. Many of these could be replaced by unwrap operators.

The last example is where two or more structural selections were interleaved. For example, here is a programmer interleaving the creation of a block for an if statement, and pasting a list of statements. In all of these editing situations, the purpose of the interleaving seemed to because of proximity; the right brace needed to be placed at the end, and the insertion point
Unstructured Edits

Typos

Temporary removal of delimiters

Two or more structured edits interleaved.

Although there we did observe unstructured edits, they were very rare.

The majority were typos, and in many cases worsened by Eclipse’ automatic formatting. For example, here’s a programmer trying to type a name, and accidentally types an open brace.

Many were just the temporary removal of delimiters. For example, here a programmer removes the left part of a new expression, then removes the right. Many of these could be replaced by unwrap operators.

The last example is where two or more structural selections were interleaved. For example, here is a programmer interleaving the creation of a block for an if statement, and pasting a list of statements. In all of these editing situations, the purpose of the interleaving seemed to be because of proximity; the right brace needed to be placed at the end, and the insertion point
Implications

- There is *a lot* of structure to the way people edit (Java) code (and little use of the flexibility of text).
- Including more structure in code editors could make operations like selection, wrapping, and unwrapping easier.
- Code editors could take advantage of the implicit structure in names and comments.
- Next: design new interaction techniques that support the edits we observed, but include the benefits structured editors.

There are several implications to our results.

The biggest is that there is a lot of structure to the way people edit code.

This suggests that we may be able to trade some of the flexibility of unstructured text for more support from the programming environment. For example:

Including more structure in code editor’s interaction techniques could help operations like selection, wrapping, and unwrapping easier.

Code editors could take advantage of the implicit structure in names and comments, to help programmers.