

Let linguistics guide software analysis

*Software Analysis using Natural Language
Queries*

Pooja Rani
Ph.D Student
Software Composition Group
University of Bern, Switzerland

Inspiration

Natural languages like English, German, French etc.

Programming languages like R, Java, Python, C++ etc

Do programming languages shape our mind as the natural languages do?

Software Analysis

Code snippet of a function

distanceBetweenTwoQuestions

```
|sentenceOne sentenceTwo questionDistanceCalculator sentenceDistance|
sentenceOne := 'The hunter killed the lion'.
sentenceTwo := 'The lion was killed by the hunter'.

questionDistanceCalculator := McQuestionsDistance new questionOne: sentenceOne; questionTwo:sentenceTwo.

sentenceDistance := questionDistanceCalculator calculateDistance.

self
assert: [ sentenceDistance isNumber ]
description: [ 'Distance must only contain digits' ].

^ sentenceDistance
```

Software Analysis

Code snippet of a function

I, a function, represent an example.

I have four temporary variables named <sentenceOne,sentenceTwo, questionDistanceCalculator, sentenceDistance>

I takes two setnence named <sentenceOne>,<sentenceTwo>.

I use McQuestionsDistance class to calculate distance between two sentences.

I verify the value <sentenceDistance> given by McQuestionsDistance.

I makes sure the distance returned is in the form of digits.

Software Analysis

Code snippet of a function that calculates the distance between two sentences.

```
distanceBetweenTwoQuestions
```

```
|sentenceOne sentenceTwo questionDistanceCalculator sentenceDistance|
sentenceOne := 'The hunter killed the lion'.
sentenceTwo := 'The lion was killed by the hunter'.

questionDistanceCalculator := McQuestionsDistance new questionOne: sentenceOne; questionTwo:sentenceTwo.

sentenceDistance := questionDistanceCalculator calculateDistance.

self
assert: [ sentenceDistance isNumber ]
description: [ 'Distance must only contain digits' ].

^ sentenceDistance
```

Software Analysis

Code snippet of a function that calculates the distance between two sentences.

I, a function, represent an example.

I have four temporary variables named <sentenceOne,sentenceTwo, questionDistanceCalculator, sentenceDistance>

I takes two setnence named <sentenceOne>,<sentenceTwo>.

I use `McQuestionsDistance` class to calculate distance between two sentences.

I verify the value <sentenceDistance> given by `McQuestionsDistance`.

I makes sure the distance returned is in the form of digits.

Software Analysis

Code snippet of a function that calculates the distance between two sentences.

distanceBetweenTwoQuestions

```
|sentenceOne sentenceTwo questionDistanceCalculator sentenceDistance|
sentenceOne := 'The hunter killed the lion'.
sentenceTwo := 'The lion was killed by the hunter'.

questionDistanceCalculator := McQuestionsDistance new questionOne: sentenceOne; questionTwo:sentenceTwo.

sentenceDistance := questionDistanceCalculator calculateDistance.

self
assert: [ sentenceDistance isNumber ]
description: [ 'Distance must only contain digits' ].

^ sentenceDistance
```

I, a function, represent an example.

I have four temporary variables named <sentenceOne,sentenceTwo, questionDistanceCalculator, sentenceDistance>

I takes two setnence named <sentenceOne>,<sentenceTwo>.

I use **McQuestionsDistance** class to calculate distance between two sentences.

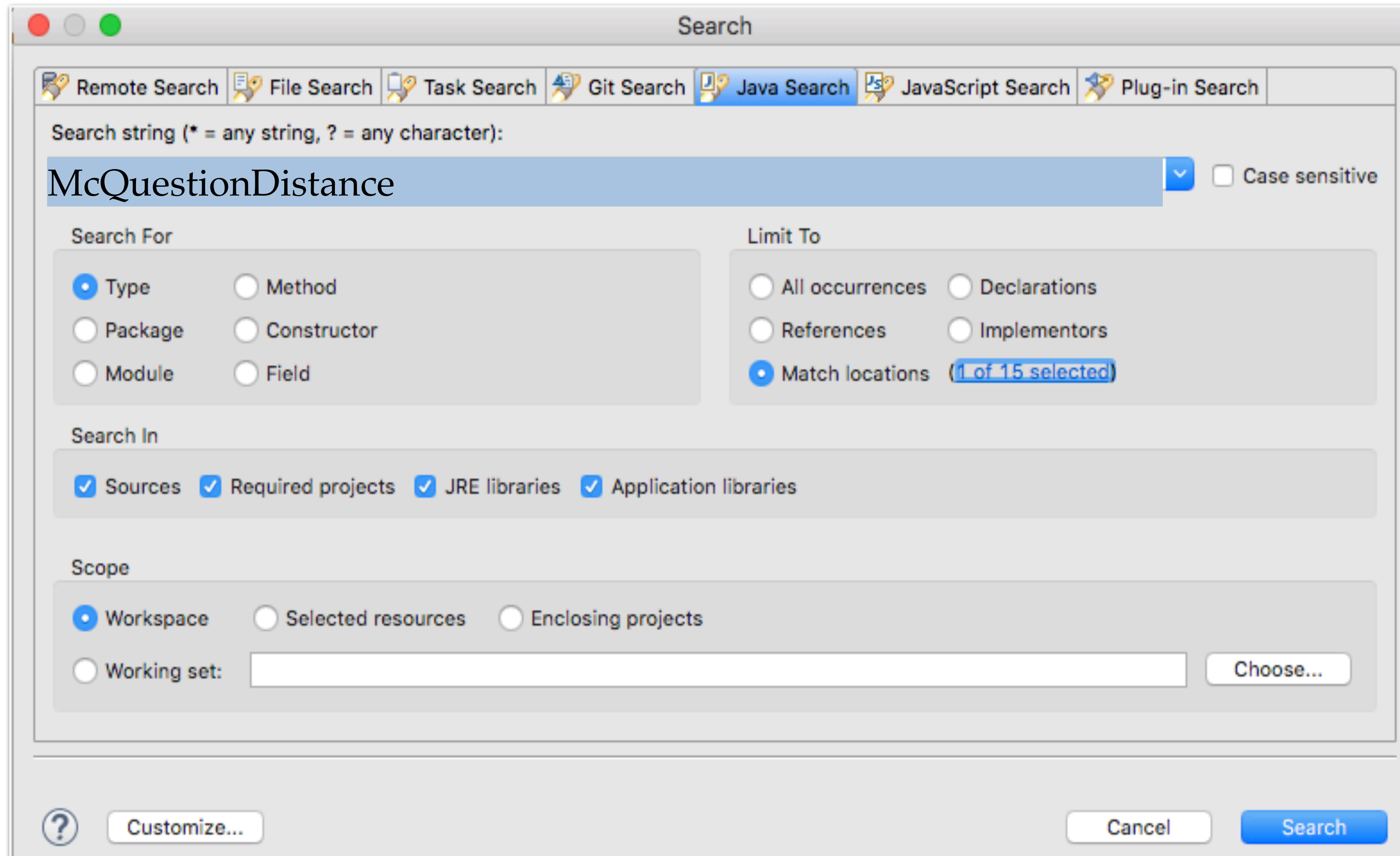
I verify the value <sentenceDistance> given by **McQuestionsDistance**.

I makes sure the distance returned is in the form of digits.

The Interesting part is developers most of the time do not write `comments` for their code.

How we are suppose to understand `code elements` without description?

We **search** for code elements in tools. For their examples, occurrences in function or comments.



Not everyone can use these tools.

Natural language interface for analysis tools

Communication barrier for new developers

Search in English



NlpGrammarFormExamples>>#grammarForm

Scoped Variables History Navigator

- ! NlpAstExamples
- ! NlpCoreResponseDepend
- ! NlpCoreResponseExempl
- ! NlpCoreResponseTokenE
- ! NlpDeveloperQuestionEx
- NlpExamplesTextCase
- ! NlpGrammarFormExempl
- ! NlpInputProcessorExempl
- NlpQueryContextExempl

gt-examples

grammarForm

```
grammarForm
<gtExample>
<description:'grammar form of a developers question'>
|
  ^ NlpGrammarForm fromAst: (NlpInputProcessor new process:
(NlpCoreResponseExamples new developerQuestion))
```

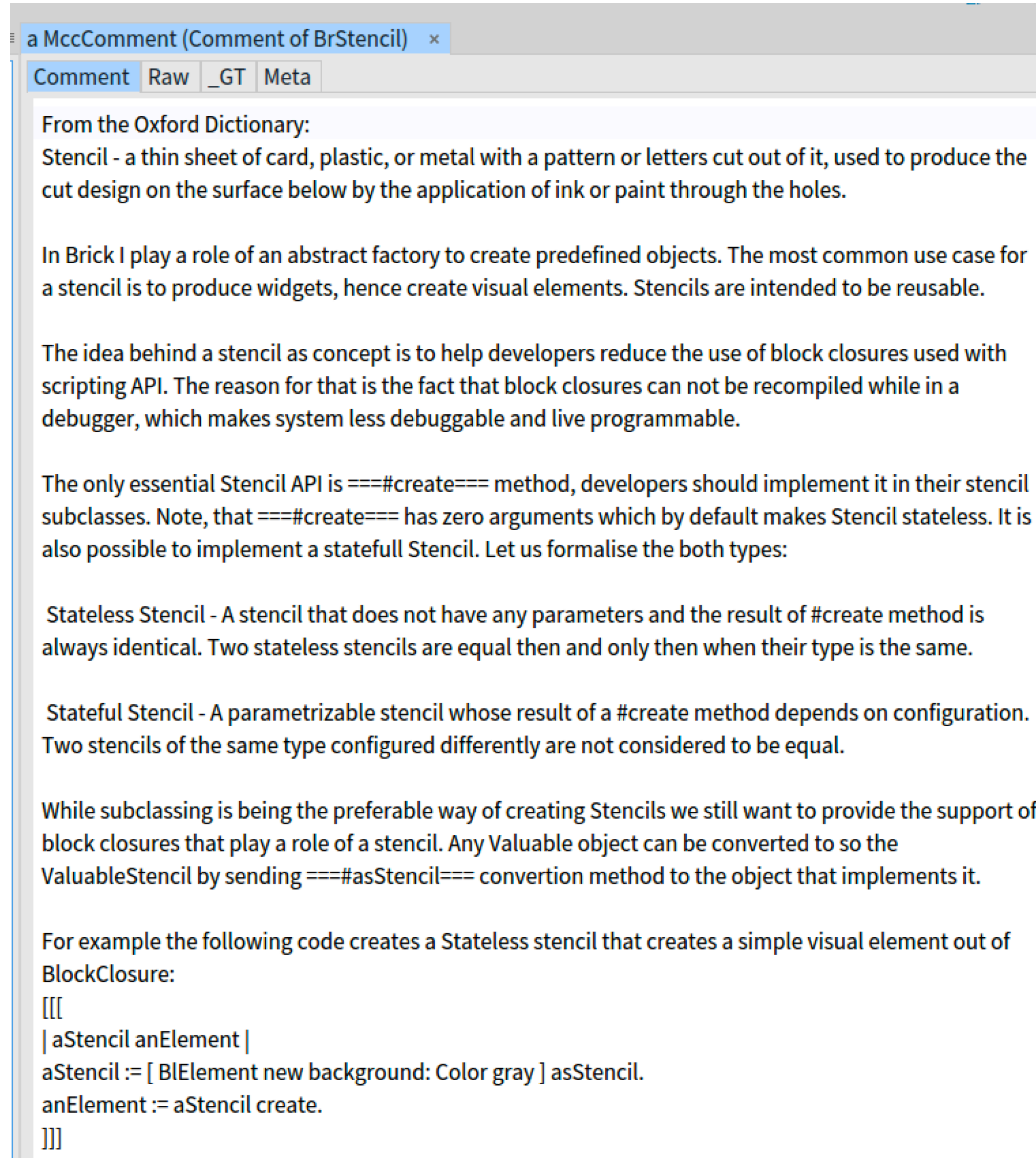
4/6 [1] Format as you read W +L

The Interesting part is developers most of the time do not write `comments` for their code.

The Interesting part is developers
write **comments** for their code.

When they write, how will you take maximum information out of those comments?

How developers write comments



a MccComment (Comment of BrStencil) x

Comment Raw _GT Meta

From the Oxford Dictionary:
Stencil - a thin sheet of card, plastic, or metal with a pattern or letters cut out of it, used to produce the cut design on the surface below by the application of ink or paint through the holes.

In Brick I play a role of an abstract factory to create predefined objects. The most common use case for a stencil is to produce widgets, hence create visual elements. Stencils are intended to be reusable.

The idea behind a stencil as concept is to help developers reduce the use of block closures used with scripting API. The reason for that is the fact that block closures can not be recompiled while in a debugger, which makes system less debuggable and live programmable.

The only essential Stencil API is `===#create===` method, developers should implement it in their stencil subclasses. Note, that `===#create===` has zero arguments which by default makes Stencil stateless. It is also possible to implement a statefull Stencil. Let us formalise the both types:

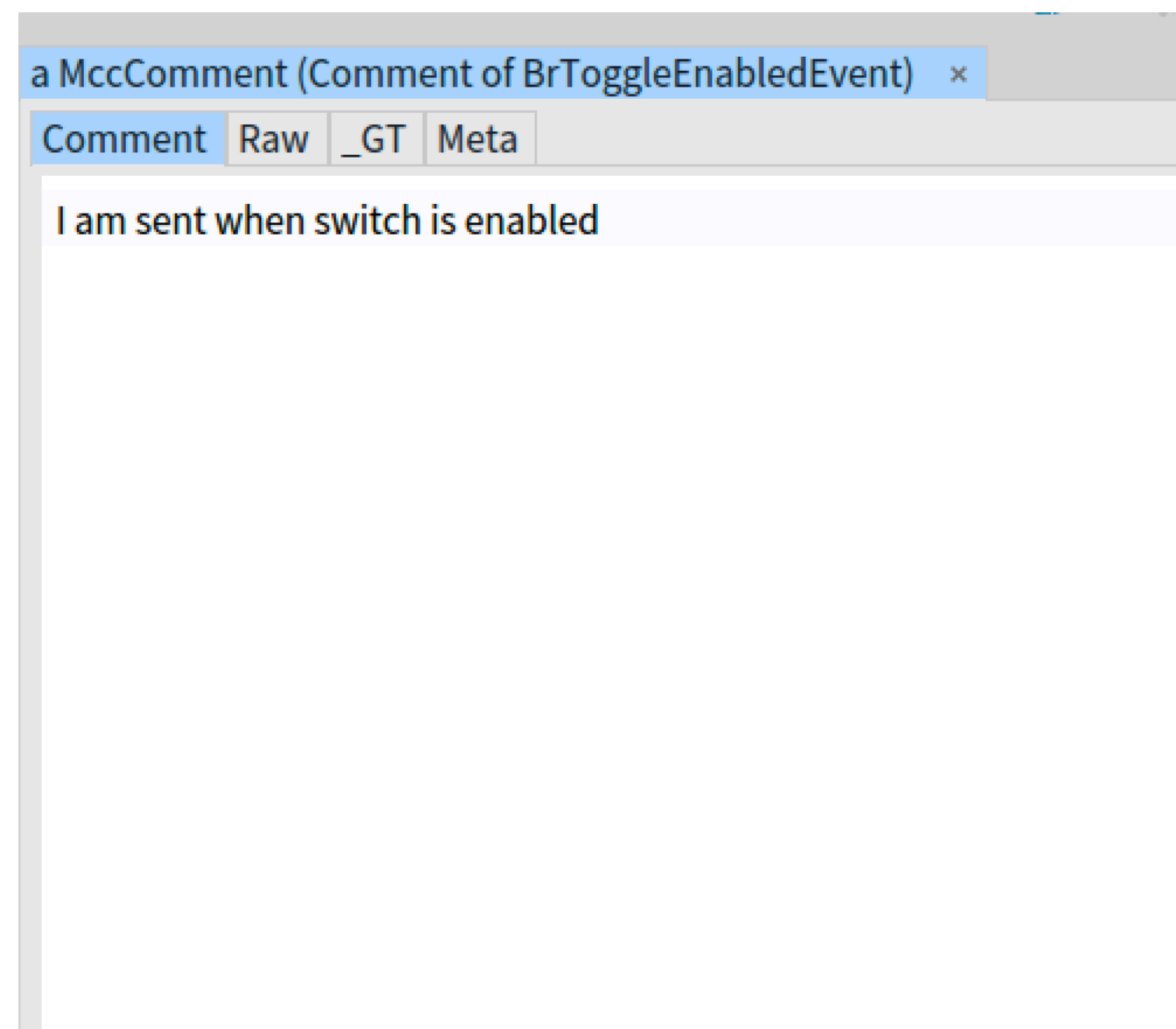
Stateless Stencil - A stencil that does not have any parameters and the result of `#create` method is always identical. Two stateless stencils are equal then and only then when their type is the same.

Stateful Stencil - A parametrizable stencil whose result of a `#create` method depends on configuration. Two stencils of the same type configured differently are not considered to be equal.

While subclassing is being the preferable way of creating Stencils we still want to provide the support of block closures that play a role of a stencil. Any Valuable object can be converted to so the ValuableStencil by sending `===#asStencil===` conversion method to the object that implements it.

For example the following code creates a Stateless stencil that creates a simple visual element out of BlockClosure:

```
[[  
| aStencil anElement |  
aStencil := [ BElement new background: Color gray ] asStencil.  
anElement := aStencil create.  
]]
```



a MccComment (Comment of BrToggleEnabledEvent) x

Comment Raw _GT Meta

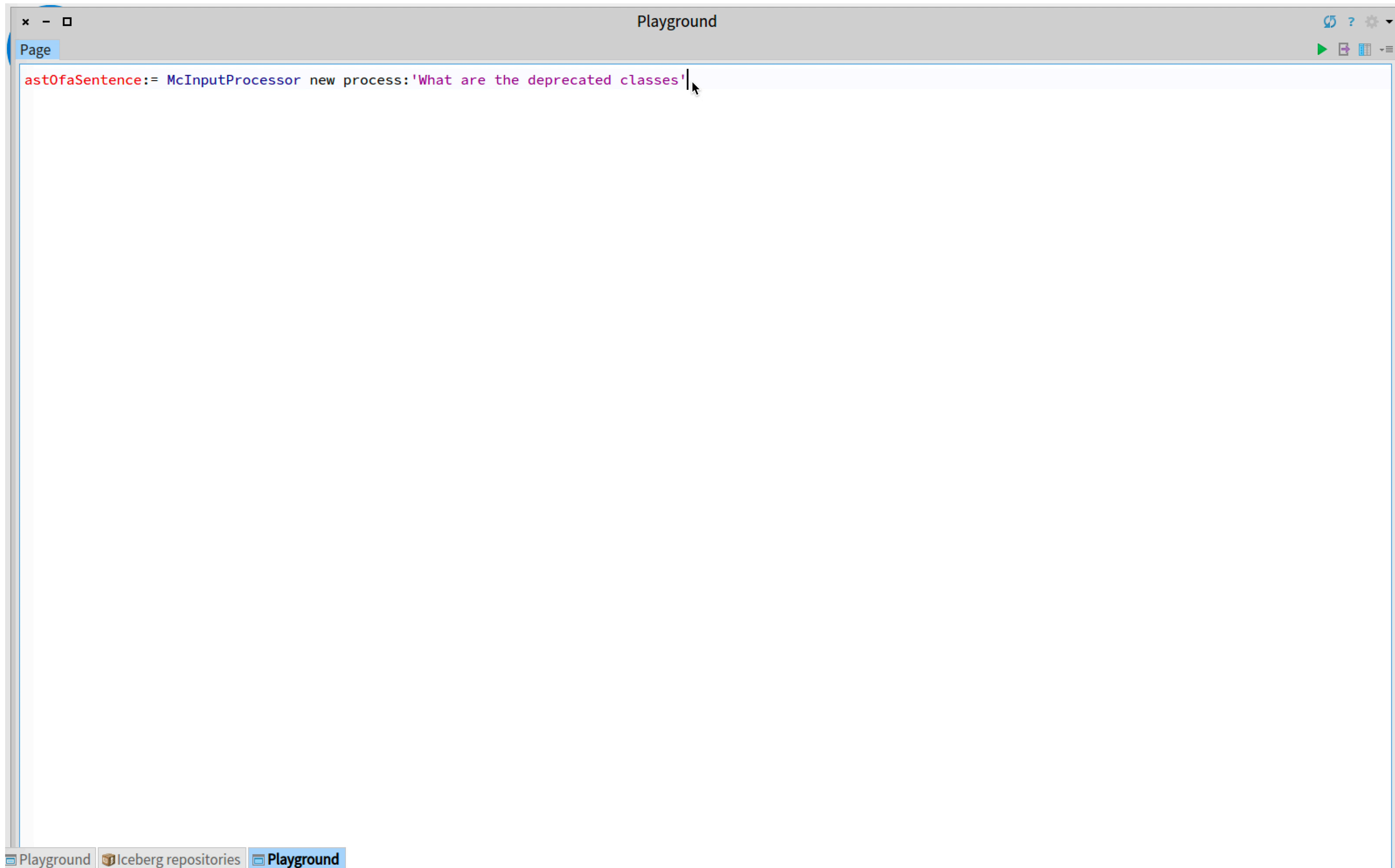
I am sent when switch is enabled

Challenges



- Developers do not write detailed comments
- The text data is mixed of programming and natural language
- Applying linguistics to software artifacts
- Semantic tools are not ready for software engineering domain

Analyze structure of a sentence



The image shows a screenshot of a web-based Scala Playground interface. The window title is "Playground". The code editor contains the following Scala code snippet:

```
astOfaSentence := McInputProcessor new process: 'What are the deprecated classes'
```

The code is color-coded: "astOfaSentence" is in red, "McInputProcessor" is in blue, "new" is in purple, and the string "What are the deprecated classes" is in pink. A mouse cursor is positioned at the end of the string. The interface includes a "Page" tab, a toolbar with icons for refresh, help, settings, run, copy, and paste, and a taskbar at the bottom with three open windows: "Playground", "Iceberg repositories", and "Playground".

Summary

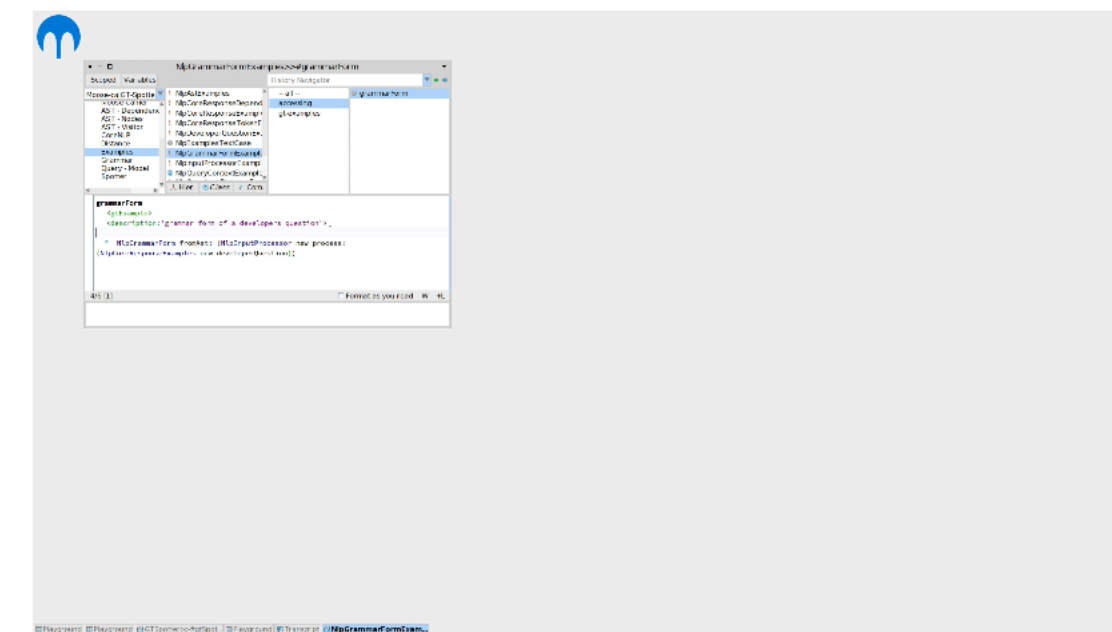
Software Analysis

Code snippet of a function that calculates the distance between two sentences.

```
def calculateDistance(sentence1, sentence2):  
    # I have four temporary variables named 'sentenceOne', 'sentenceTwo', 'questionDistanceCalculator', and 'sentenceDistance'.  
    # I use McQuestionDistance class to calculate distance between two sentences.  
    # I verify the value 'sentenceDistance' given by McQuestionDistance.  
    # I make sure the distance returned is in the form of digits.  
    sentenceOne = 'The cat sat on the mat.'  
    sentenceTwo = 'The dog ran in the park.'  
    questionDistanceCalculator = McQuestionDistance(sentenceOne, sentenceTwo)  
    sentenceDistance = questionDistanceCalculator.calculateDistance()  
    return sentenceDistance
```

The Interesting part is developers most of the time do not write **comments** for their code.

Search in English



How developers write comments

```
/*  
 * From the Oxford Dictionary:  
 * Stencil - a thin sheet of card, plastic, or metal with a pattern or letters cut out of it, used to produce the  
 * cut design on the surface below by the application of ink or paint through the holes.  
 *  
 * In Brick play a role of an abstract factory to create predefined objects. The most common use case for  
 * a stencil is to produce widgets, hence create visual elements. Stencils are intended to be reusable.  
 *  
 * The idea behind a stencil as concept is to help developers reduce the use of block closures used with  
 * scripting API. The reason for that is the fact that block closures can not be recompiled while in a  
 * debugger, which makes system less debuggable and live programmable.  
 *  
 * The only essential Stencil API is ==create== method, developers should implement it in their stencil  
 * subclasses. Note, that ==create== has zero arguments which by default makes Stencil stateless. It is  
 * also possible to implement a stateful Stencil. Let us formalize the both types:  
 *  
 * Stateless Stencil - A stencil that does not have any parameters and the result of create method is  
 * always identical. Two stateless stencils are equal then and only then when their type is the same.  
 *  
 * Stateful Stencil - A parametrizable stencil whose result of a create method depends on configuration.  
 * Two stencils of the same type configured differently are not considered to be equal.  
 *  
 * While subclassing to being the preferable way of creating Stencils we still want to provide the support of  
 * block closures that play a role of a stencil. Any valuable object can be converted to the  
 * ValuableStencil by sending ==asStencil== conversion method to the object that implements it.  
 *  
 * For example the following code creates a Stateless stencil that creates a simple visual element out of  
 * BlockClosure:  
 *  
 * []  
 * | aStencil anElement  
 * | aStencil = [ [BlockClosure new background: Color gray ] asStencil.  
 * | anElement = aStencil create.  
 * | ]
```

```
/*  
 * I am sent when switch is enabled  
 */
```



Feedback

- What kinds of linguistic analysis we can do in a software?
- What is the linguistic pattern of developers in different programming language?
- What are the social aspects of a software?
- Is there any English linguistic expert here, I would like to meet.

Feedback

What kinds of linguistic analysis we can do on a software?

What are the linguistic patterns of developers in different programming language?

What are the social aspects of a natural language text present in a software?

Is there any English linguistic expert here, I would like to meet.