

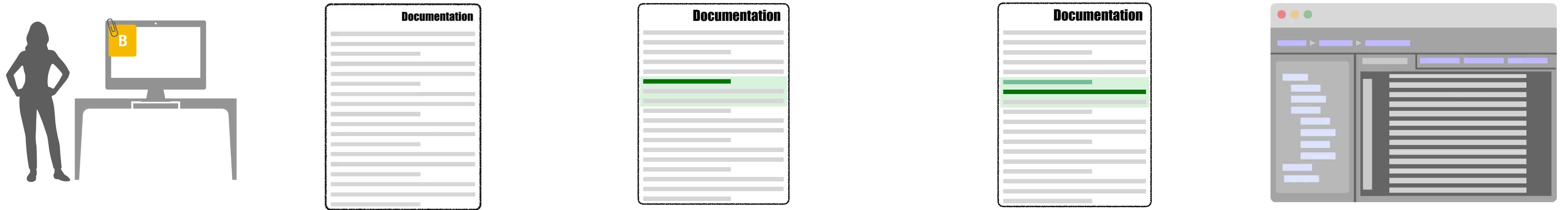
# Active Documentation: Helping Developers Follow Design Decisions

**Sahar Mehrpour** George Mason University

**Thomas D. LaToza** George Mason University

**Rahul K. Kindi** Cornell University

# Scenario: Using Documentation Today

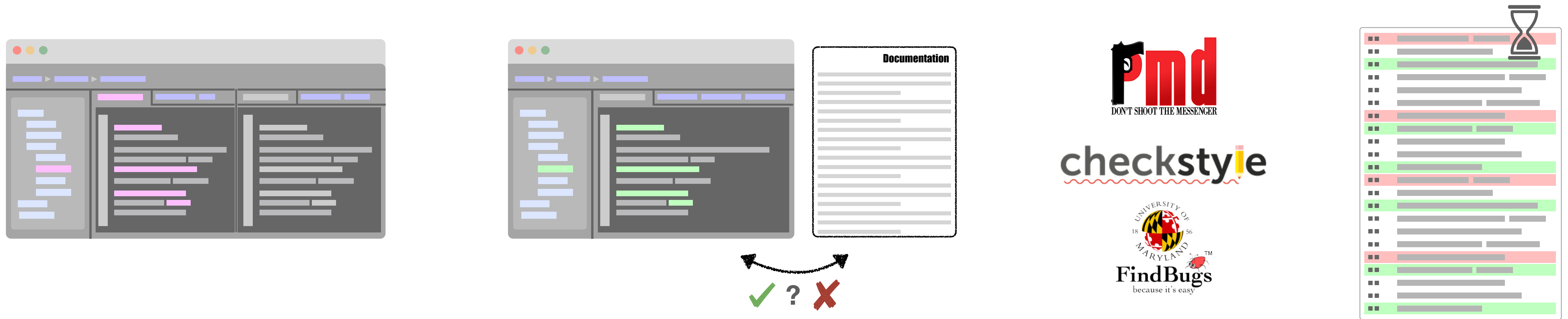


All interaction among *Artifact* classes must be done through *Command* classes to apply *sharding*.

Each *Artifact* must have a *Command* class.

- ▶ Alice is a developer in a Company.
- ▶ She is working to implement of a small feature (a new *Artifact B*) in the codebase.
- ▶ Alice starts reading the *documentation* ... But the documentation is **too long**.
- ▶ Alice reads one of the **Design Decisions** describing what alternative was chosen and why.
- ▶ Looking at the description of the design decision, she reads one of the **Design Rules** describing how to implement the design decision.
- ▶ She tries to **connect** the design rule to the code ... But the documentation and the source code are large and hard to connect.

# Scenario: Using Documentation Today



- ▶ After some time, Alice finds that she believes to be an **Example** illustrating how to implement an Artifact. Following this example, She tries to re-implement her new class.
- ▶ She writes some code and wants to know if it follows the design rules. But she is not sure that she is following the examples correctly, and that there aren't other rules she missed.
- ▶ She looks at the **rule checkers** the company is using, but they only report defects about her use of Java and do not help with understanding these design decisions.
- ▶ Frustrated, she commits her code and waits for **code reviews** from other developers.

# Active Documentation

Our solution: ***active*** documentation



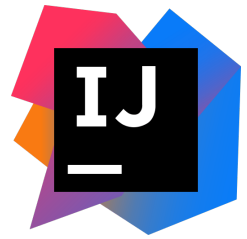
- ✓ Design rules are translated into constraints and actively checked against code.

- ✓ Wherever a design rule applies to code, an active link between the documentation and code is generated.



- ✓ Developers can actively update the documentation.

# ACTIVE DOCUMENTATION



## IntelliJ IDE plugin

```
1 package com.crowdcoding.commands;
2
3 import com.crowdcoding.entities.artifacts.DesignDoc;
4 import com.crowdcoding.servlets.ThreadContext;
5
6
7 public abstract class DesignDocCommand extends Command {
8     protected long DesignDocId;
9
10    // This function is called when a new DesignDoc must be created.
11    @ public static DesignDocCommand create(String title, String description, boolean isApiArtifact, boolean isReadOnly) {
12        return null;
13    }
14
15    private DesignDocCommand(Long DesignDocId) {
16        this.DesignDocId = DesignDocId;
17        queueCommand(this);
18    }
19
20    // All constructors for DesignDocCommand MUST call queueCommand and the end of
21    // the constructor to add the
22    // command to the queue.
23    private static void queueCommand(Command command) {
24        ThreadContext threadContext = ThreadContext.get();
25        threadContext.addCommand(command);
26    }
27
28    public void execute(final String projectId) {
29        if (DesignDocId != 0) {
30            DesignDoc designDoc = DesignDoc.find(DesignDocId);
31
32            if (designDoc == null)
33                System.out
34                    .println("error Cannot execute DesignDocCommand. Could not find DesignDoc with id: "
35                        + DesignDocId);
36            else {
37                execute(designDoc, projectId);
38            }
39        } else
40            execute(DesignDoc: null, projectId);
41    }
42
43
44    public abstract void execute(DesignDoc DesignDoc, String projectId);
45
46
```

← → Table of Content All Rules Violated Rules Generate Rules

Rules applicable for File:

### CrowdCode-master/CrowdCoding/src/com/crowdcoding/commands/DesignDocCommand.java

All Microtask commands must be handled by Command subclasses (view the rule and all snippets) ▲ ▼

IF a method is a static method on Command THEN it should implement its behavior by constructing a new Command subclass instance. The Command class contains a number of static methods. Each method creates a specific type of Command by invoking the constructor of the corresponding subclass.

Microtask Command Sharding

Examples 0 out of 54

Violated 1 out of 1

Violated snippet for this file

```
public static DesignDocCommand create(String title, String description, boolean isApiArtifact, boolean isReadOnly) {
    return null;
}
```

Violated snippet for other files

No snippet

Commands must implement execute (view the rule and all snippets) ▲ ▼

IF a class is a subclass of Command THEN it must implement execute. Commands represent an action that will be taken on an Artifact. In order for this action to be invoked, each subclass of Command must implement an execute method. This method should not be directly invoked by clients, but should be used by the Command execution engine.

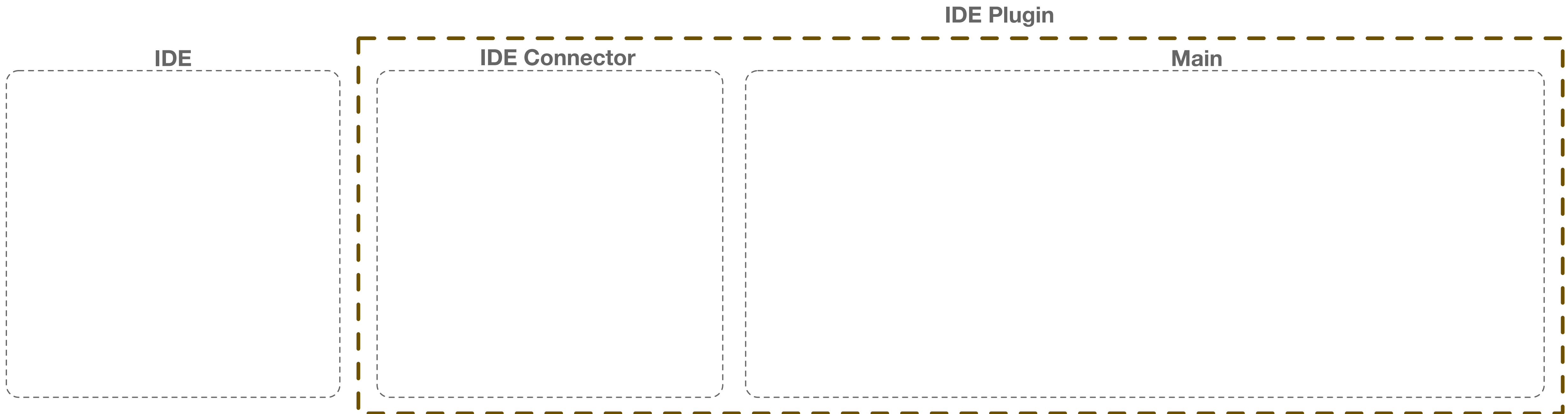
Microtask Command Sharding

Examples 0 out of 53

Violated 0 out of 0

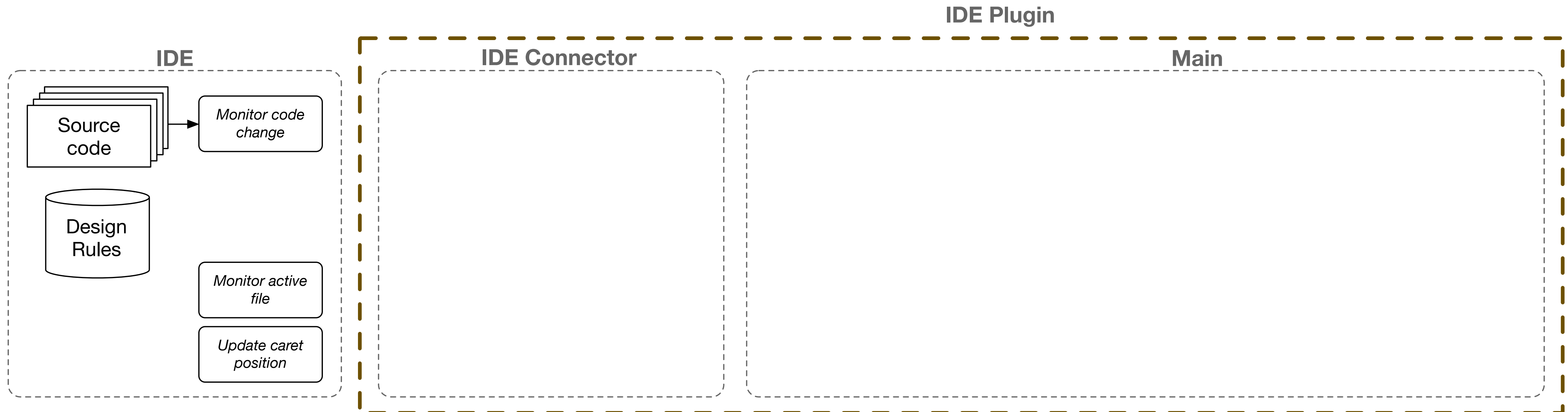
# ACTIVE DOCUMENTATION **System Architecture**

- ▶ Independent from IDEs
- ▶ Two main components: IDE Connector and Main
- ▶ IDE Connector transfers data to/from the IDE




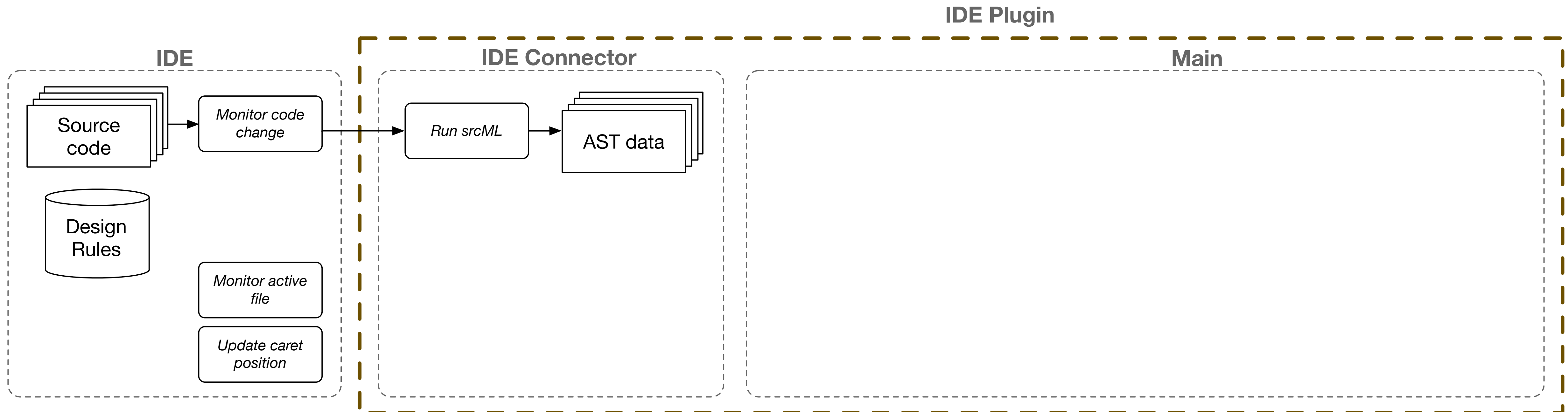
# ACTIVE DOCUMENTATION **System Architecture**

- ▶ IDE is responsible for reporting code change, active file in the editor, and updating the caret position
- ▶ Stored design rules (stored as .json) are accessible in the IDE



# ACTIVE DOCUMENTATION **System Architecture**

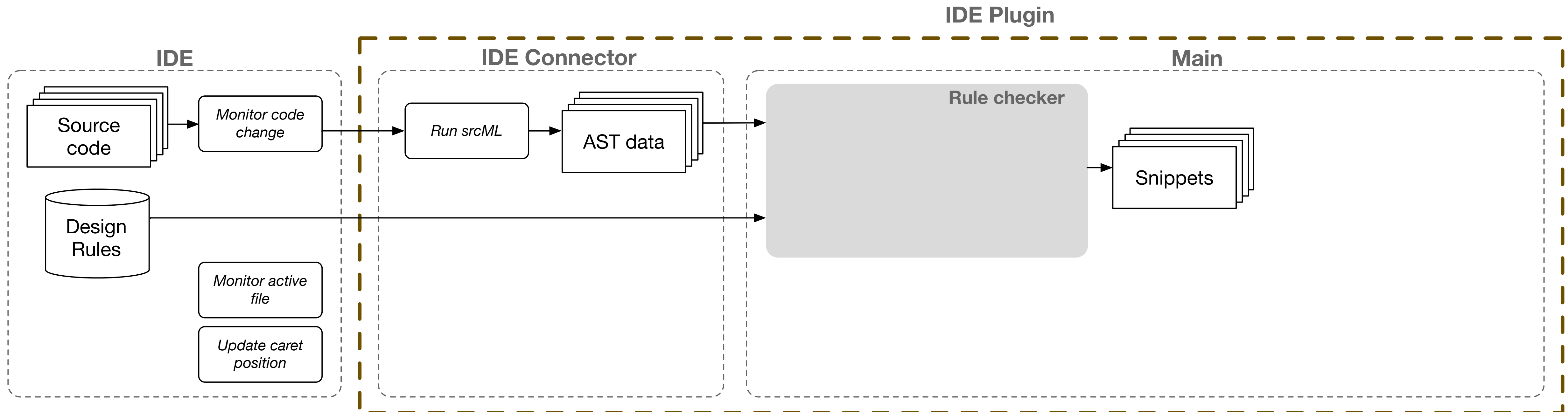
- ▶ IDE connector creates the AST of the source code.
- ▶ XML representation of the ASTs are easier to work with.
- ▶ We used srcML to create the AST.  [Maletic et al. 2002]





# ACTIVE DOCUMENTATION **System Architecture**

- ▶ The rule checker uses the design rules and the AST of code to extract snippets from code.
- ▶ ACTIVE DOCUMENTATION is agnostic to the underlying rule checker.



# Rule Checker In ACTIVE DOCUMENTATION

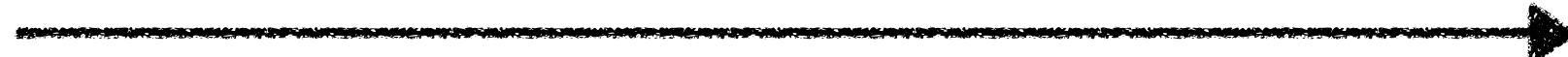
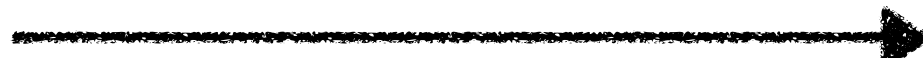
- ▶ Existing rule checkers *only* find **violations** of rules.
- ▶ Developers need to search code to know *how* a rule is followed.
- ▶ ACTIVE DOCUMENTATION shows snippets from code that **satisfy** or **violate** the rule.

WHEN and HOW the rule should apply...

**Quantifier** WHEN the rule should apply

**Constraint** HOW the rule should apply

- ▶ Each Artifact must have a Command class.

IF a class is an artifact  **Quantifier**  
THEN it should have a Command class  **Constraint**

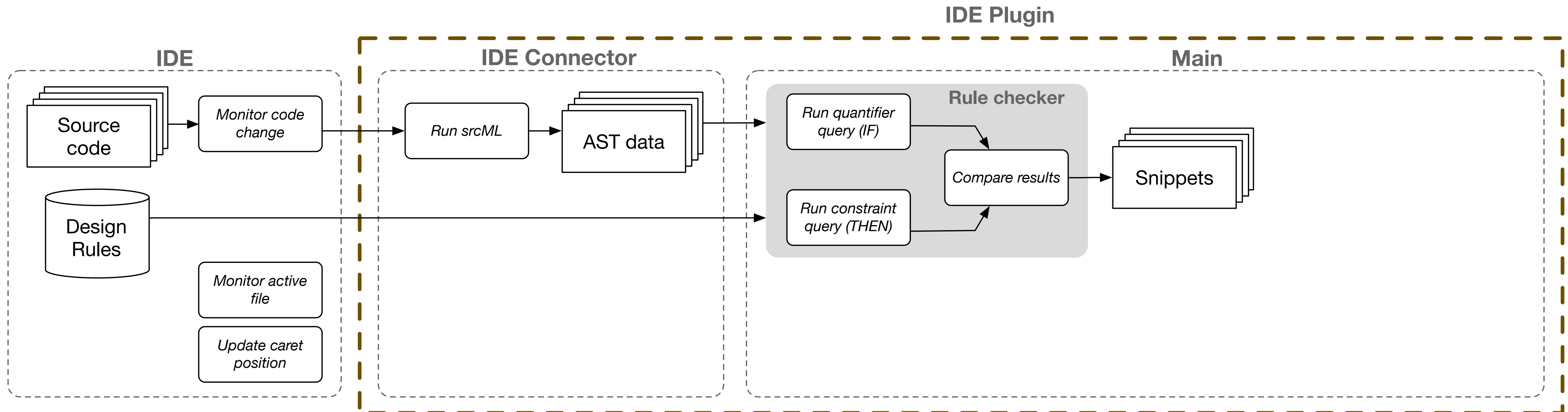
# Rule Checker In ACTIVE DOCUMENTATION

In an IF/THEN structure of a rule:

IF part  $\longrightarrow$  Quantifier Query

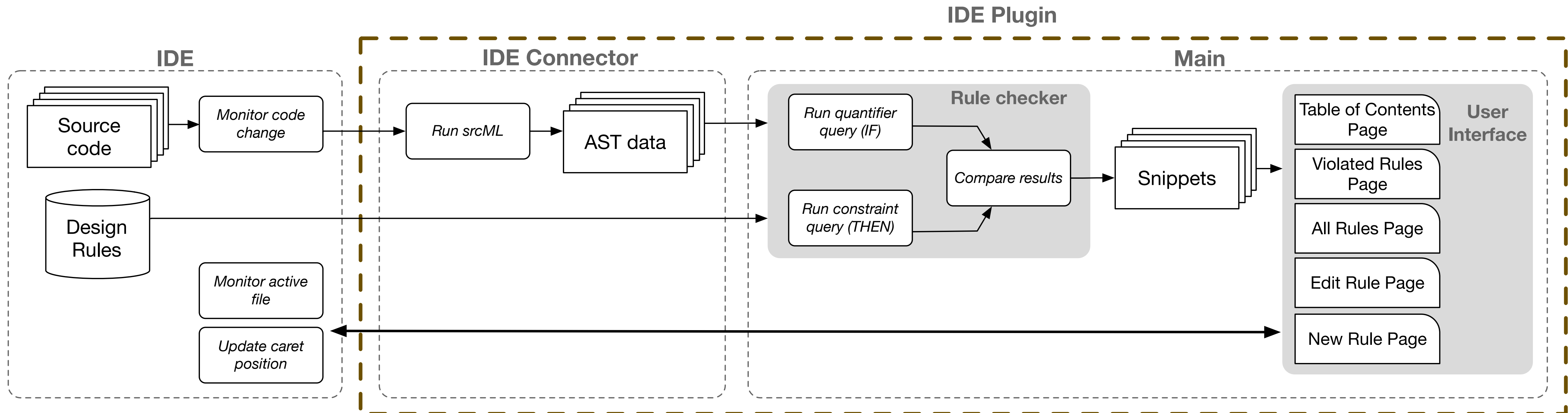
THEN part  $\longrightarrow$  Constraint Query

Compare the results of queries  $\longrightarrow$  Satisfied and Violated Snippets

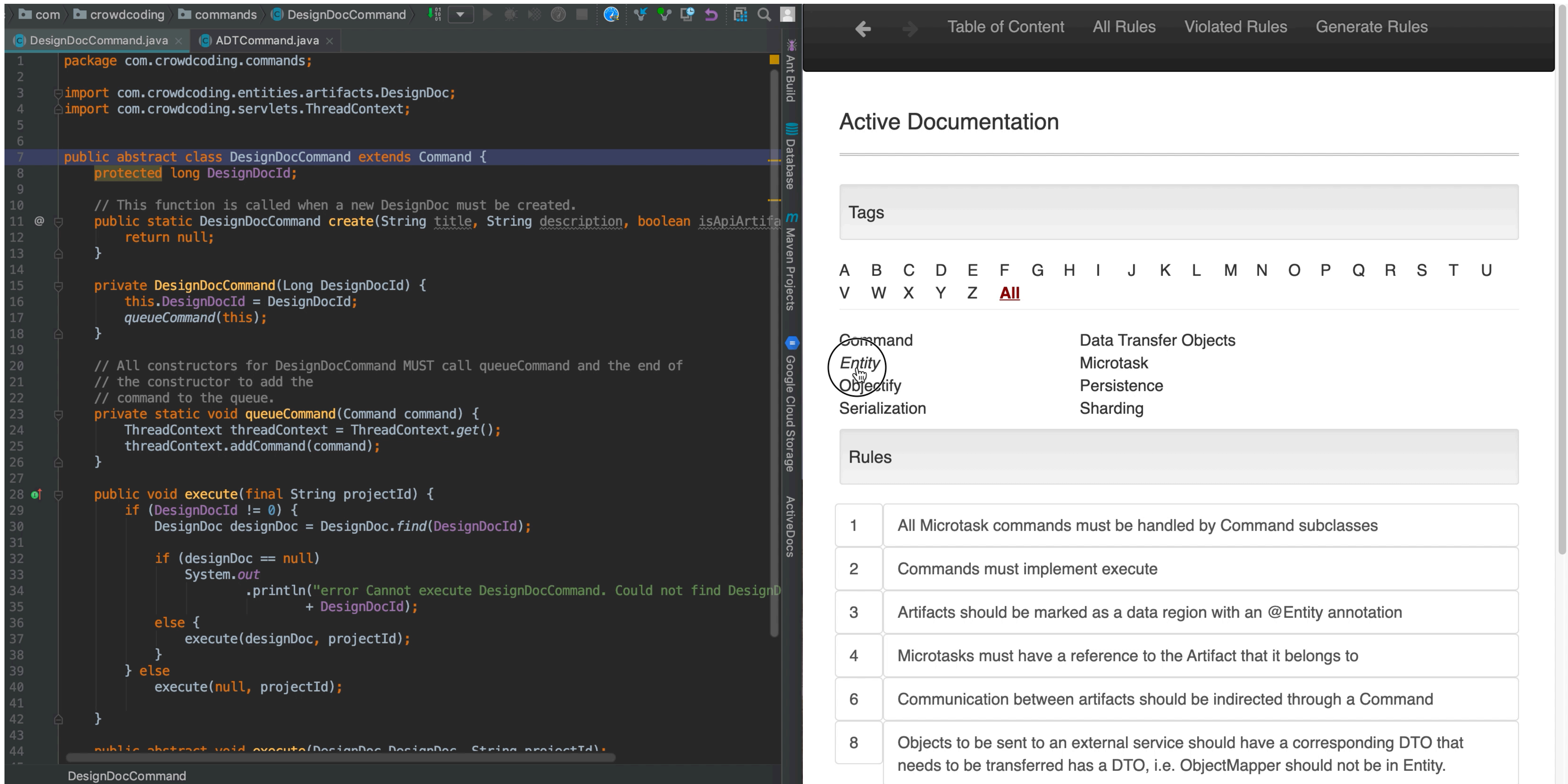


# ACTIVE DOCUMENTATION System Architecture

- ▶ After generating code snippets, they are visualized in the user interface through different pages.
- ▶ The User Interface sends and receives tasks to and from the IDE



# Rule Organization



The image shows a split-screen view. On the left is an IDE window displaying the Java source code for `DesignDocCommand`. The code includes package declarations, imports, a constructor, a `create` method, a `queueCommand` method, and an `execute` method. On the right is a web application interface with a dark header containing navigation links: `Table of Content`, `All Rules`, `Violated Rules`, and `Generate Rules`. Below the header is a section titled `Active Documentation` with a `Tags` input field and a list of letters `A` through `Z` and `All`. A sidebar on the left of the web interface lists categories: `Command`, `Entity`, `Objectify`, `Serialization`, `Data Transfer Objects`, `Microtask`, `Persistence`, and `Sharding`. The `Entity` category is circled. Below this is a `Rules` section containing a table of rules:

Rule ID	Rule Description
1	All Microtask commands must be handled by Command subclasses
2	Commands must implement execute
3	Artifacts should be marked as a data region with an @Entity annotation
4	Microtasks must have a reference to the Artifact that it belongs to
6	Communication between artifacts should be indirected through a Command
8	Objects to be sent to an external service should have a corresponding DTO that needs to be transferred has a DTO, i.e. ObjectMapper should not be in Entity.

# Using Example Code Snippets

2

```
1 package com.crowdcoding.commands;
2
3 import com.crowdcoding.entities.artifacts.DesignDoc;
4 import com.crowdcoding.servlets.ThreadContext;
5
6
7 public abstract class DesignDocCommand extends Command {
8     protected long DesignDocId;
9
10    // This function is called when a new DesignDoc must be created.
11    public static DesignDocCommand create(String title, String description, boolean isApiArtifa
12        return null;
13    }
14
15    private DesignDocCommand(Long DesignDocId) {
16        this.DesignDocId = DesignDocId;
17        queueCommand(this);
18    }
19
20    // All constructors for DesignDocCommand MUST call queueCommand and the end of
21    // the constructor to add the
22    // command to the queue.
23    private static void queueCommand(Command command) {
24        ThreadContext threadContext = ThreadContext.get();
25        threadContext.addCommand(command);
26    }
27
28    public void execute(final String projectId) {
29        if (DesignDocId != 0) {
30            DesignDoc designDoc = DesignDoc.find(DesignDocId);
31
32            if (designDoc == null)
33                System.out
34                    .println("error Cannot execute DesignDocCommand. Could not find DesignD
35                        + DesignDocId);
36            else {
37                execute(designDoc, projectId);
38            }
39        } else
40            execute(null, projectId);
41    }
42
43
44    public abstract void execute(DesignDoc designDoc, String projectId);
45
```

← → Table of Content All Rules Violated Rules Generate Rules

## Violated Rules

[\(view the rule and all snippets\)](#) ▲ ▼

### All Microtask commands must be handled by Command subclasses

IF a method is a static method on Command THEN it should implement its behavior by constructing a new Command subclass instance. The Command class contains a number of static methods. Each method creates a specific type of Command by invoking the constructor of the corresponding subclass.

Microtask Command Sharding

Examples 54

Violated 1

```
public static ADTCommand create (String description, String name, HashMap<String,String> s
return new Create( description, name, structure, examples, isApiArtifact,
```

```
public static ADTCommand update(long ADTId, String description, String name, HashMap<Strin
return new Update(ADTId, description, name, structure );
```

```
public static ADTCommand delete(long ADTId) {
return new Delete(ADTId);
```

```
public static TestCommand create(TestDTO test, long functionId, boolean isApiArtifact, boo
return new Create(test, functionId, isApiArtifact, isReadOnly);
```

```
public static TestCommand update(TestDTO test) {
return new Update(test);
```

```
public static TestCommand delete(TestDTO test) {
return new Delete(test);
```

```
public static WorkerCommand awardPoints(String workerID, int points)
{ return new AwardPoints(workerID, points); }
```

# Instant Feedback

```
com crowdcoding commands DesignDocCommand
DesignDocCommand.java x ADTCommand.java x
28 public void execute(String projectId) {
29     if (DesignDocId != 0) {
30         DesignDoc designDoc = DesignDoc.find(DesignDocId);
31
32         if (designDoc == null)
33             System.out
34                 .println("error Cannot execute DesignDocCommand. Could not find DesignDoc
35                     + DesignDocId);
36         else {
37             execute(designDoc, projectId);
38         }
39     } else
40         execute(null, projectId);
41 }
42 }
43
44 public abstract void execute(DesignDoc DesignDoc, String projectId);
45
46 protected static class Create extends DesignDocCommand {
47     private String description;
48     private String title;
49
50     private boolean isApiArtifact;
51     private boolean isReadOnly;
52
53
54     public Create(String title, String description, boolean isApiArtifact, boolean isReadOnly) {
55         super(0L);
56         this.title = title;
57         this.description = description;
58         this.isApiArtifact = isApiArtifact;
59         this.isReadOnly = isReadOnly;
60     }
61 }
62
63 public void execute(DesignDoc designDoc, String projectId) {
64     new DesignDoc(title, description, isApiArtifact, isReadOnly, projectId);
65 }
66 }
67
68 }
```

(view the rule and all snippets) ▲ ▼

**All Microtask commands must be handled by Command subclasses**

IF a method is a static method on Command THEN it should implement its behavior by constructing a new Command subclass instance. The Command class contains a number of static methods. Each method creates a specific type of Command by invoking the constructor of the corresponding subclass.

**Microtask** **Command** **Sharding**

Examples 0 out of 54 Violated 1 out of 1

(view the rule and all snippets) ▲ ▼

**Commands must implement execute**

IF a class is a subclass of Command THEN it must implement execute. Commands represent an action that will be taken on an Artifact. In order for this action to be invoked, each subclass of Command must implement an execute method. This method should not be directly invoked by clients, but should be used by the Command execution engine.

**Microtask** **Command** **Sharding**

Examples 1 out of 54 Violated 0 out of 0

(view the rule and all snippets) ▲ ▼

**Artifacts should be marked as a data region with an @Entity annotation**

IF an object is an artifact subclass THEN it needs to be an entity. To signal that instances of a class constitute a separate data region, the class should have the @Entity annotation. All Artifact subclasses should be marked as a data region.

(view the rule and all snippets) ▲ ▼

**Microtasks must have a reference to the Artifact that it belongs to**

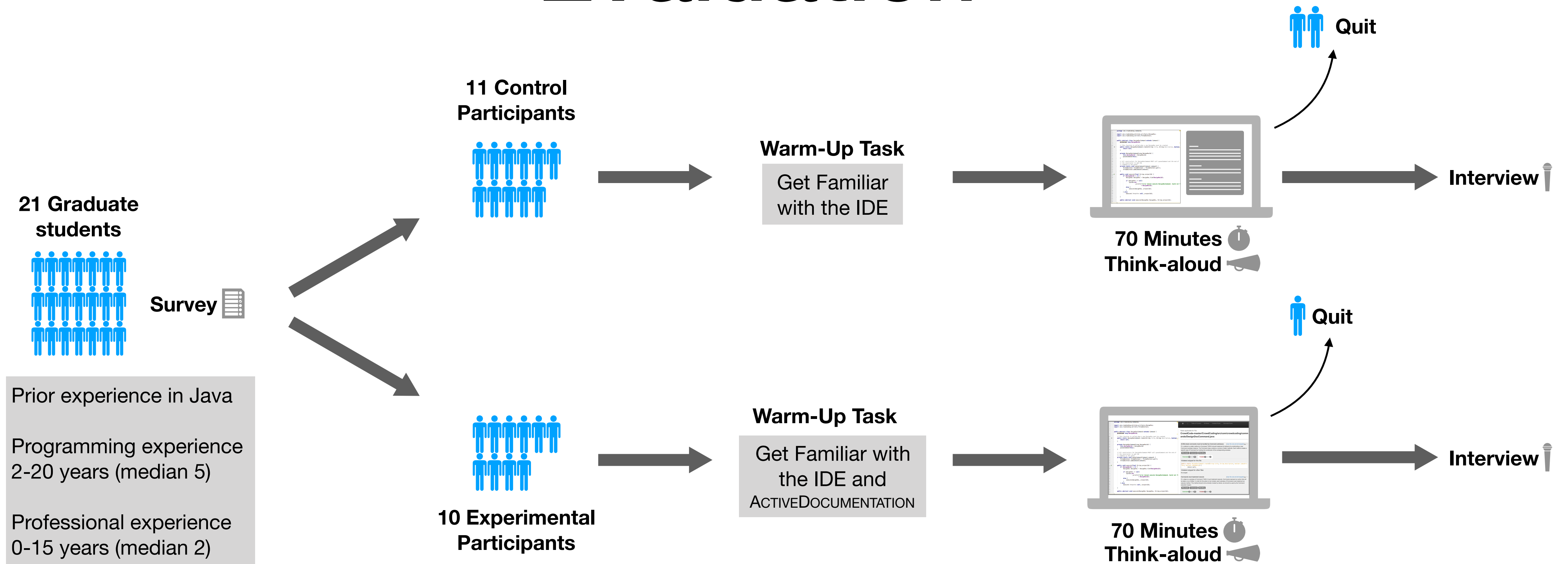
IF a class is a subclass of Microtask THEN it needs a field representing the reference to the associated entity. Each Microtask represents work to be done on an Artifact. As such, it needs to be connected back to its owning artifact through a reference to the Artifact. Without the reference, they need to have an ID of the artifact and for submitting they need to load the data beforehand.

# Research Question

- ▶ Compared to traditional documentation, are developers able to use `ACTIVEDOCUMENTATION` to write code following design rules more quickly and successfully?
- ▶ In what ways does `ACTIVEDOCUMENTATION` support developers in writing code in an unfamiliar codebase?



# Evaluation



**Task:** Add a small feature to an existing code

- ▶ Existing code: web-based IDE, 9K LOC, 107 Java classes, abstraction based on *artifacts* (persisted in a persistence framework)
- ▶ Requested code: add a new *artifact*, add 20 lines of code, edit 2 lines of code

# Result - Quantitative

	Diff		Time (Minutes)		Submitted Lines of Code		
	Added	Removed	First Edit	Task Durat.	Missing	Incorrect	Task Irrlvnt
<b>Control Group</b>							
Mean	45.00	3.56	20.33	68.33	7.44	1.78	20.67
Median	36.00	3.00	12.00	70.00	1.00	1.00	8.00
Std. Dev.	39.64	3.40	19.82	3.39	8.37	2.54	27.76
<b>Experimental Group</b>							
Mean	29.67	4.44	6.33	48.89	1.89	0.11	5.89
Median	29.00	3.00	6.00	47.00	0.00	0.00	2.00
Std. Dev.	6.36	5.50	3.71	17.44	5.30	0.33	9.87
<b>All Participants</b>							
Mean	37.33	4.00	13.33	58.61	4.67	0.94	13.28
Median	29.00	3.00	8.50	70.00	0.00	0.00	3.00
Std. Dev.	28.65	4.46	15.59	15.77	7.37	1.95	21.59
<i>p</i> value	0.142	0.343	<b>0.015</b>	<b>0.038</b>	0.056	<b>0.043</b>	0.082

- ▶ Experimental participants were **3 times** faster in starting editing the code and **28%** faster in finishing the task.
- ▶ Experimental participants added few lines of code and removed more lines of code.
- ▶ Experimental participants submitted **98%** fewer incorrect LOC.

# Result - Qualitative

## Control Group

- ✗ Challenges in finding relevant design decisions within the design documentation
- ✗ Challenges in connecting code with design decisions
- ✗ Challenges in finding relevant pieces of code, scattered in different classes

## Experimental Group





- ✓ Used *Violated Rules* page to find relevant design decisions.
- ✓ Used the violated snippets to identify relevant places to make changes.
- ✓ Used example snippets listed to compare examples of the rule and the faulty lines of code.
- ✓ Used real-time feedback to detect errors and violations early, immediately after changing the code without running the application.

# Active Documentation: Helping Developers Follow Design Decisions

Sahar Mehrpour, Thomas D. LaToza, Rahul K. Kindi



## Scenario







- ▶ After some time, Alice finds a class assuming being an Artifact and considers it as an **Example**. She tries to re-implement them in the new class.
- ▶ She writes some code and wants to know if it conforms with the design rules ... But she is not able to **verify** it herself.
- ▶ She looks at the **rule checkers** the company is using, but they are only reporting **universal defects** and not helpful.
- ▶ Frustrated, she commits her code and waits for **code reviews** from other developers.

## Active Documentation

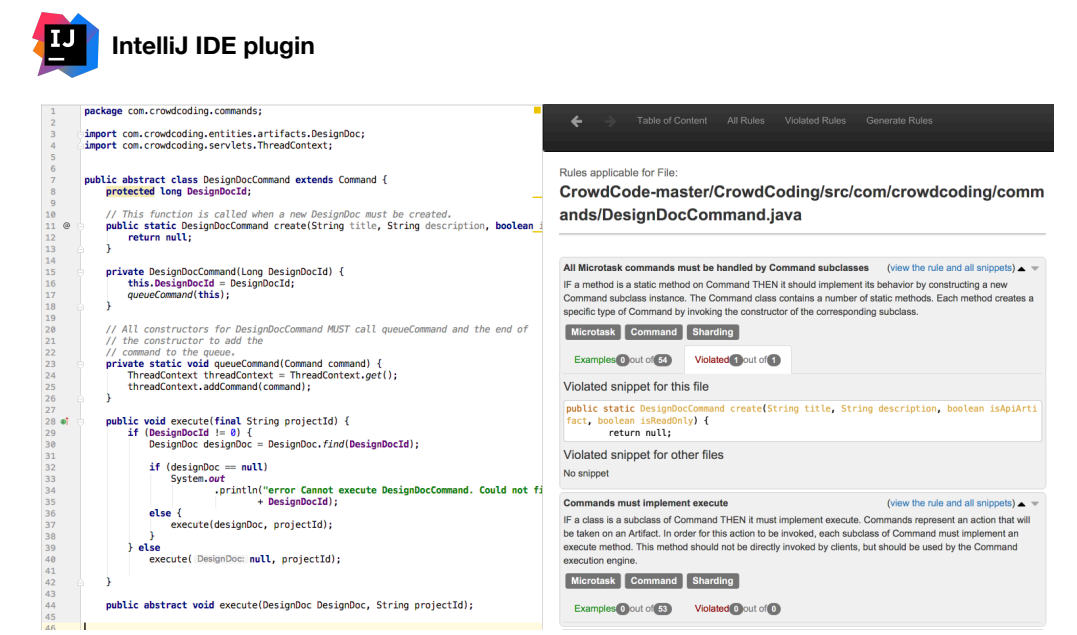
Our solution: **active** documentation

- ✓ Design rules are translated into constraints and **actively checked** against code.
- ✓ Wherever a design rule applies to code, an **active link** between the documentation and code is generated.
- ✓ Developers can **actively update** the documentation.

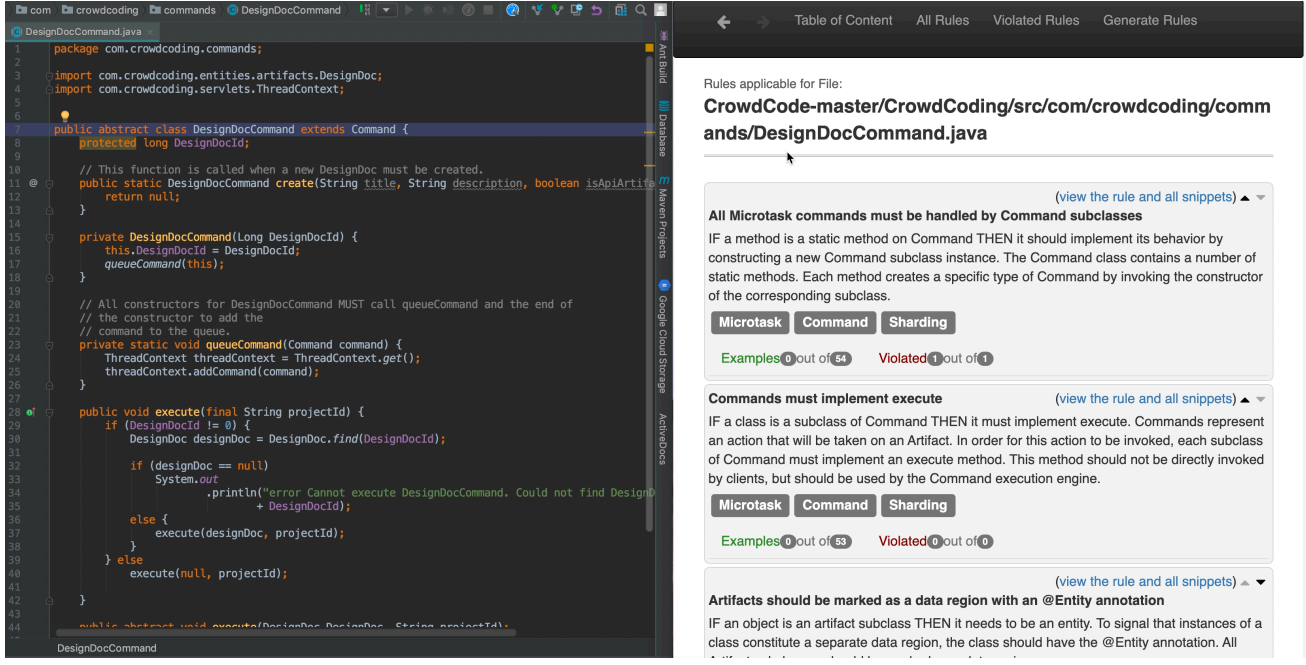


## ACTIVE DOCUMENTATION

IntelliJ IDE plugin



## Instant Feedback



## Result - Quantitative

	Diff		Time (Minutes)		Submitted Lines of Code		
	Added	Removed	Pre Edit	Post Edit	Missing	Incorrect	Total Lines
<b>Control Group</b>							
Mean	45.00	3.56	20.33	68.33	7.44	1.78	20.67
Median	36.00	3.00	12.00	70.00	1.00	1.00	8.00
Std. Dev.	39.64	3.40	19.82	3.39	8.37	2.54	27.76
<b>Experimental Group</b>							
Mean	29.67	4.44	6.33	48.89	1.89	0.11	5.89
Median	29.00	3.00	6.00	47.00	0.00	0.00	2.00
Std. Dev.	6.36	5.50	3.71	17.44	5.30	0.33	9.87
<b>Participants</b>							
Mean	37.33	4.00	13.33	58.61	4.67	0.94	13.28
Median	29.00	3.00	8.50	70.00	0.00	0.00	3.00
Std. Dev.	28.65	4.46	15.59	15.77	7.37	1.95	21.59
p value	0.142	0.343	<b>0.015</b>	<b>0.038</b>	0.056	<b>0.043</b>	0.082

- ▶ Experimental participants were **3 times faster** in starting editing the code and **28% faster** in finishing the task.
- ▶ Experimental participants added few lines of code and removed more lines of code.
- ▶ Experimental participants submitted **98% fewer incorrect LOC**.

## Result - Qualitative

Control Group	Experimental Group
✗ Challenges in finding relevant design decisions within the design document	✓ Used <i>Violated Rules</i> page to find relevant design decisions.
✗ Challenges in connecting code with design decisions	✓ Used the violated snippets to identify relevant places to make changes.
✗ Challenges in finding relevant pieces of code scattered in different classes	Used example snippets listed to compare examples of the rule and the faulty lines of code.
	Used real-time feedback to detect errors and violations early, immediately after changing the code without running the application.

# Thank You!

[bit.ly/ActiveDocumentation](https://bit.ly/ActiveDocumentation)

[smehrpou@gmu.edu](mailto:smehrpou@gmu.edu)