Structuring Documentation to Support State Search

A Laboratory Experiment about Protocol Programming

Joshua Sunshine
Jim Herbsleb
Jonathan Aldrich

Example URLConnection task

```
/*
 * Precondition: `conn` is connected to a valid URL.
 * Postcondition: Return up to the first 1000 characters of the data
 * at the URL to which `conn` is connected
 */
public static String readInput(URLConnection conn) throws IOException {
        InputStreamReader reader = new InputStreamReader(conn.getInputStream());
        char[] urlChars = new char[1000];
        int numBytes = reader.read(urlChars, 0, 1000);
        return new String(urlChars, 0, numBytes);
}
```

PLAID

Example URLConnection task

```
/*
 * Precondition: `conn` is connected to a valid URL.
 * Postcondition: Return up to the first 1000 characters of the data
 * at the URL to which `conn` is connected
 */
public static String readInput(URLConnection conn) throws IOException {
        InputStreamReader reader = new InputStreamReader(conn.getInputStream());
        char[] urlChars = new char[1000];
        int numBytes = reader.read(urlChars, 0, 1000);
        return new String(urlChars, 0, numBytes);
}
```

✗ Cannot read from connection, doInput is false.

Programmer given test to reveal this bug

PLAID

Example URLConnection task

```
public static String readInput(URLConnection conn) {

   InputStreamReader reader = new . . .
   char[] urlChars = new char[1000];
   int numBytes = reader.read(urlChars, 0, 1000);
   return new String(urlChars, 0, numBytes);
}
```

PLAID

Example URLConnection task

```
public static String readInput(URLConnection conn) {
➔ conn.setDoInput(true);
  InputStreamReader reader = new . . .
  char[] urlChars = new char[1000];
  int numBytes = reader.read(urlChars, 0, 1000);
  return new String(urlChars, 0, numBytes);
}
```

PLAID

Example URLConnection task

```
public static String readInput(URLConnection conn) {
   conn.setDoInput(true);✗
   InputStreamReader reader = new . . .
   char[] urlChars = new char[1000];
   int numBytes = reader.read(urlChars, 0, 1000);
   return new String(urlChars, 0, numBytes);
}
```

✗ IllegalStateException:
Already connected

PLAID

Example URLConnection task

```
public static String readInput(URLConnection conn) {
   conn.setDoInput(true);✗
   InputStreamReader reader = new . . .
   char[] urlChars = new char[1000];
   int numBytes = reader.read(urlChars, 0, 1000);
   return new String(urlChars, 0, numBytes);
}
```

✗ IllegalStateException: Already connected

Method: setDoInput

Throws:
IllegalStateException - if already connected

PLAID

Example URLConnection task

```
public static String readInput(URLConnection conn) {
   conn.setDoInput(true);✗
   InputStreamReader reader = new . . .
   char[] urlChars = new char[1000];
   int numBytes = reader.read(urlChars, 0, 1000);
   return new String(urlChars, 0, numBytes);
}
```

Method: connect

URLConnection objects go through two phases: first they are created, then they are connected. After being created, and before being connected, various options can be specified (e.g., doInput and UseCaches). After connecting, it is an error to try to set them.

PLAID

Example URLConnection task

```
public static String readInput(URLConnection conn) {
  UrlConnection c2 = conn.getURL().openConnection();
  c2.setDoInput(true);
  c2.connect();
  InputStreamReader reader = new
      InputStreamReader(c2.getInputStream());
  char[] urlChars = new char[1000];
  int numBytes = reader.read(urlChars, 0, 1000);
  return new String(urlChars, 0, numBytes);
}
```

PLAID

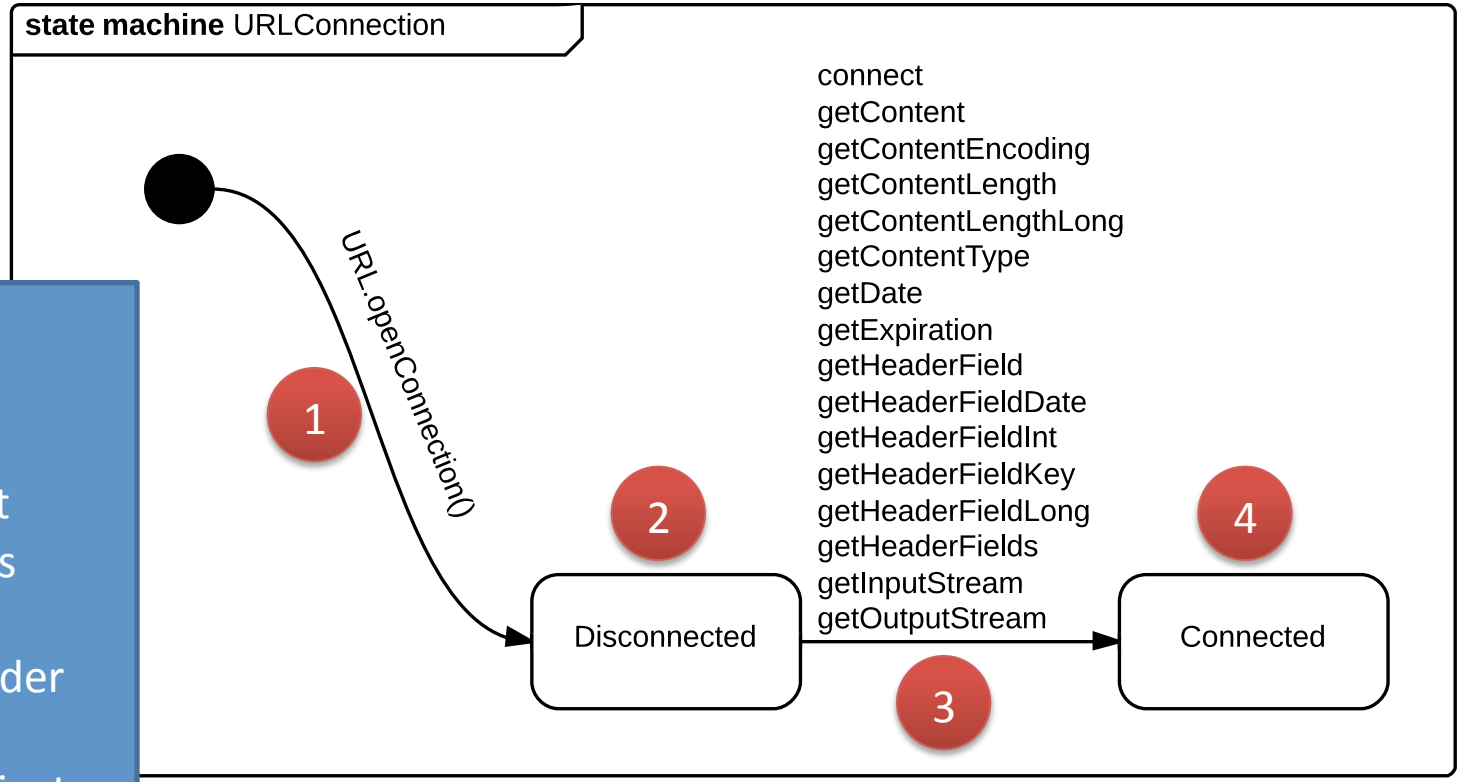
Example URLConnection task

```
public static String readInput(URLConnection conn) {
   UrlConnection c2 = conn.getURL().openConnection();
   c2.setDoInput(true);
   c2.connect();
   InputStreamReader reader = new
        InputStreamReader(c2.getInputStream());
   char[] urlChars = new char[1000];
   int numBytes = reader.read(urlChars, 0, 1000);
   return new String(urlChars, 0, numBytes);
}
```

test passes

PLAID

URLConnection has a protocol



To use:

1) create an instance
2) set request parameters
3) connect
4) access header fields and remote object content

PLAID

Protocols are common and hard

- Protocols are common [Beckman 2011]:
  - 3X as many Java stdlib. classes define protocols as use type parameters
- Libraries with protocols can be difficult to use [Jaspan 2011]
  - ¾ of ASP.NET forum posts involved temporal constraints
  - StackOverflow search results (November 2013):

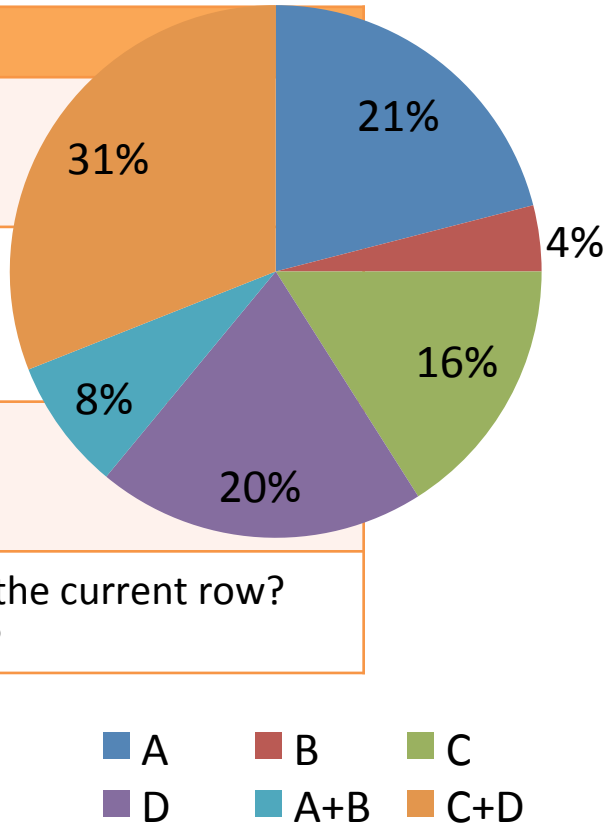
Rank	Exception type	# of results
1	NullPointerException	16,879
2	RuntimeException	13,756
3	ClassNotFoundException	7,654
4	**IllegalStateException**	**5,890**
…		
11	IndexOutOfBoundsException	856
12	UnsupportedOperationException	818
…		
—	Total for all 27 java.lang exception types	63,933

PLAID

Protocol programmer information needs

Programmer time was dominated by four categories of state search:

	Category	Example instances
A	What abstract state is the object in?	Is the TimerTask scheduled? Is [the ResultSet] x scannable?
B	What are the capabilities of object in state X?	Can I schedule a scheduled TimerTask? What can I do on the insert row?
C	In what state(s) can I do operation Z?	When can I call doInput? Which ResultSets can I update?
D	How do I transition from state X to state Y?	How do I move from the insert row to the current row? Which method schedules a TimerTask?



[Sunshine 2013]

PLAID

Plaiddoc

- Documentation
  - Javadoc
    Generated from
    annotated Java source

URLConnection
addRequestProperty
connect
getDoInput
setDoInput
getContent

PLAID

Plaiddoc – methods by state

- Documentation:
  - Javadoc
  - Plaiddoc

URLConnection
- addRequestProperty
- connect
- getDoInput
- setDoInput
- getContent

- Plaiddoc generated from:
  - Lightweight manually-created state specifications
  - Java code

URLConnection
- getDoInput

Disconnected
- addRequestProperty
- connect
- setDoInput

Connect
- getContent

PLAID

Plaiddoc – transitions

URLConnection

Ret Type	*Method Name*
void	addRequestProperty
void	connect
boolean	getDoInput
void	setDoInput
Object	getContent

State Name	*Postcondition*	*Ret Type*	*Method Name*
URLConnection			
	--	boolean	getDoInput
Disconnected			
	--	void	addRequestProperty
	Connected	void	connect
	--	void	setDoInput
Connected			
	--	Object	getContent

PLAID

Plaiddoc – transitions

URLConnection

Ret Type	*Method Name*
void	addRequestProperty
void	
boole	
void	
Objec	

Plaiddoc also generates:
Preconditions
and
ASCII state diagrams

See paper for details

State Name			
URLConnection			
Disconnected			
	Connected	void	connect
	--	void	setDoInput
Connected			
	--	Object	getContent

PLAID

Contribution: laboratory experiment of protocol programming

- State search tasks
  - 3 JSL APIs (URLConnection, ResultSet, Timer)
  - Each task asked participants to answer instances of 4 state search questions
- Primary comparison: Plaiddoc vs. Javadoc
- Primary output variables:
  - Task completion time
  - Correctness

PLAID

Experiment - research questions

- RQ1 Can programmers answer state search questions more efficiently using Plaiddoc than Javadoc?
- RQ2 Are programmers as effective answering non-state questions using Plaiddoc as they are with Javadoc?
- RQ3 Will programmers who use Plaiddoc answer state search questions more correctly than programmers who use Javadoc?
- RQ4 Will programmers get better at answering state search questions as they get more practice?

PLAID

Experiment - background

- Search tasks can be much easier with diagrams than unstructured text [Larkin 1987]
- Demonstrations of API use improvements:
  - eMoose directives significantly improved task completion times [Dekel 2009]
  - MAPO example search significantly reduced errors [Zhong 2009]

Experiment – methodology

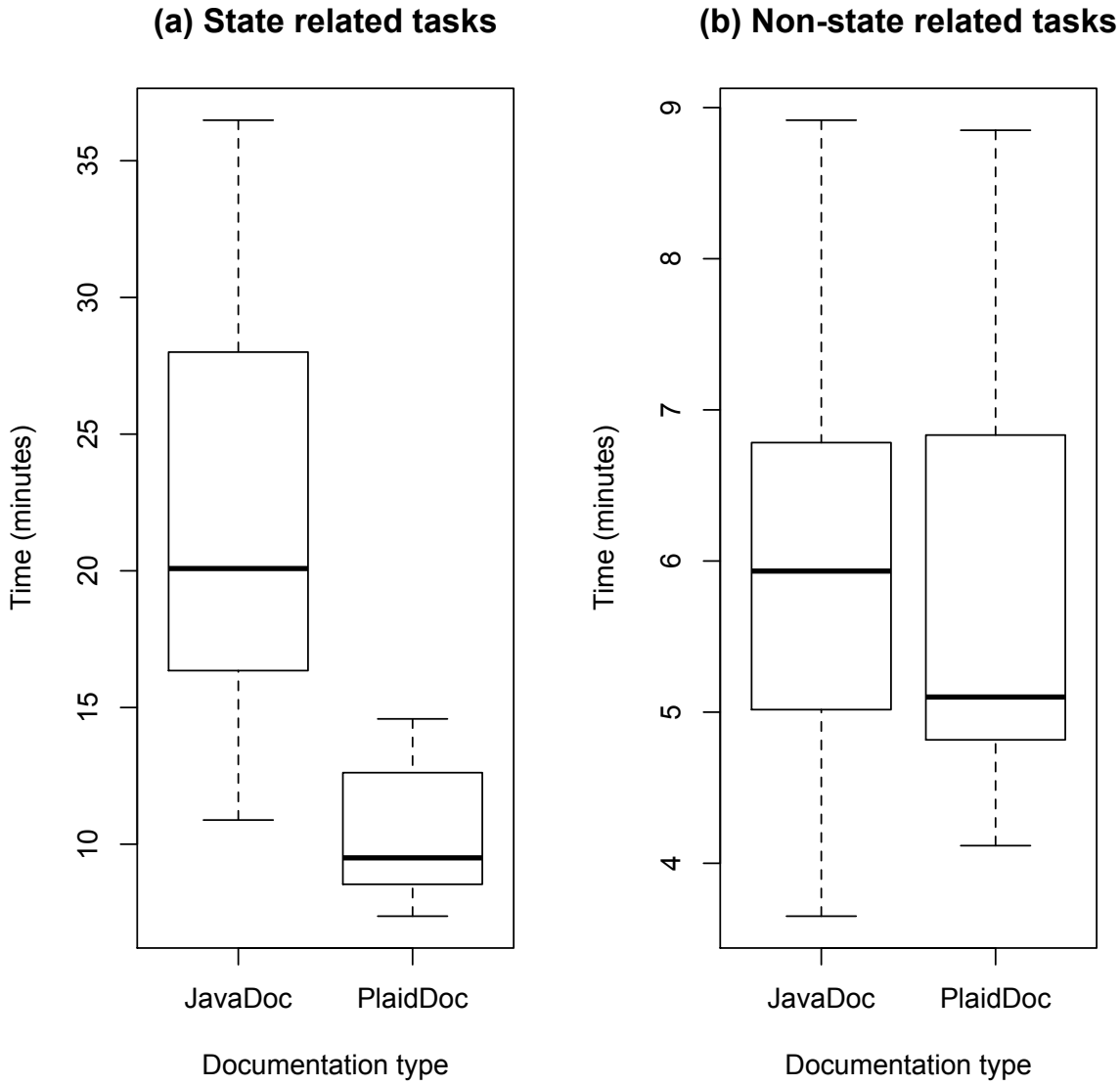
- 20 undergraduate and masters students
- Two by two design:
  - Plaiddoc vs. Javadoc
  - Two task orderings: Timer batch first vs. URLConnection batch first
- Between subjects: five participants in each of the four conditions
- All participants trained in all interventions
- 21 tasks – 16 state-search and and five non-state search tasks

PLAID

Results – time on task

- Time between end of question and final answer

- State task mean:
  - Plaiddoc: 10.3 min
  - Javadoc: 22.4 min
  - p<0.001
- Non-state mean:
  - Plaiddoc: 5.77 min
  - Javadoc: 5.95 min
  - p=0.802



PLAID

Results - correctness

- Participants confirmed “final answers”
- Sometimes wrong
  - Experiment proceeded regardless of correctness
  - Participants were not told if they were correct

Category	Plaiddoc	Javadoc
Correct	151	143
Incorrect	2	15
Timed-out	7	2

- Contingency table analysis, p=0.002

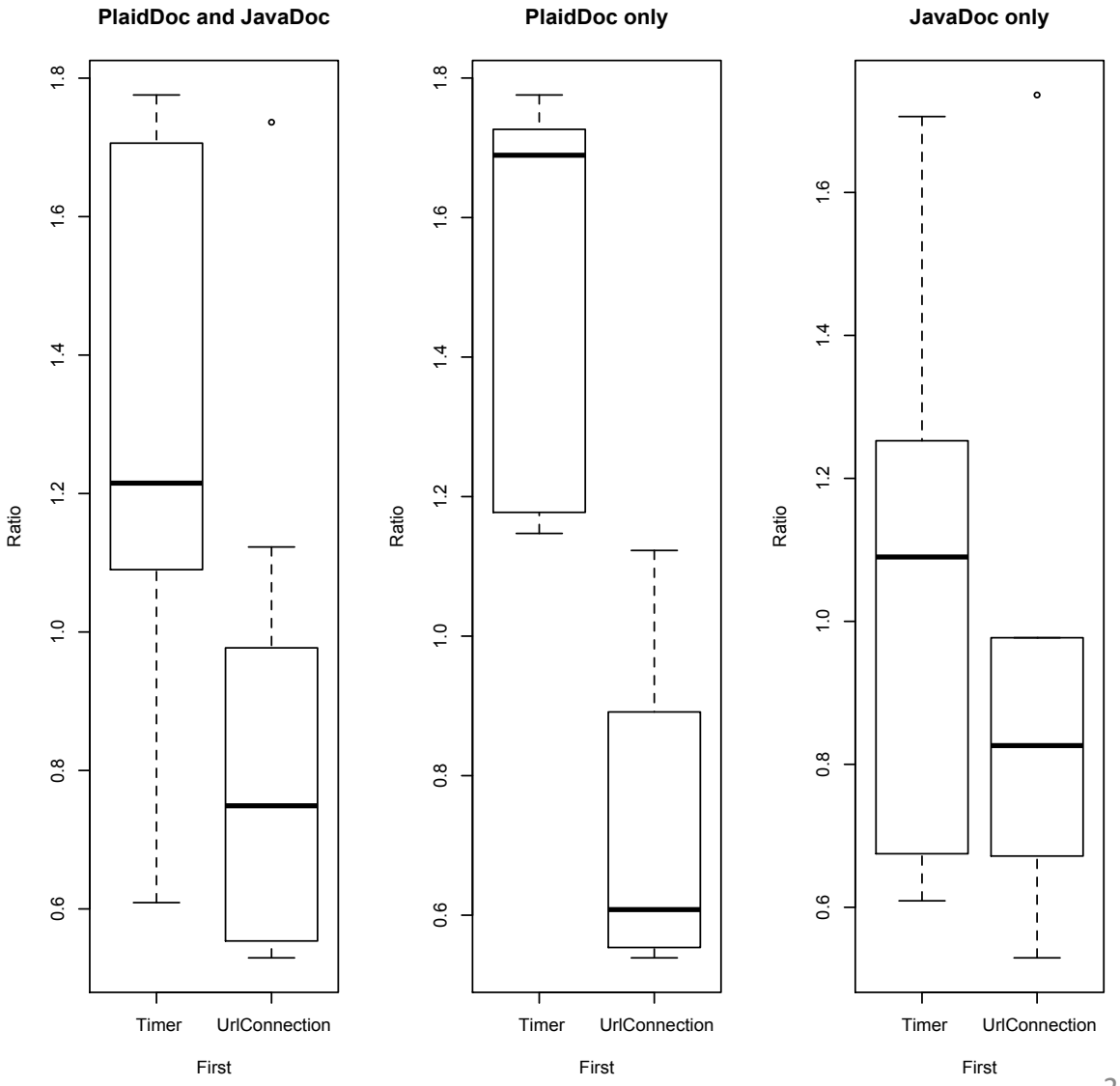
PLAID

Incorrect responses

- Method does not exist
  - How do I transition the ResultSet from the ForwardOnly to the Scrollable state?
  - Question resulted in all 7 timeouts and 6 wrong answers
- Tripped up by abnormal mode of use
- Used heuristics to digest a lot of text, sometimes these heuristics failed

Results – learning

- Ratio of first task batch completion time to last batch
- Plaiddoc means:
  - Timer first: 1.5
  - UrlCon first: 0.74
  - p = 0.03
- Javadoc means:
  - Timer first: 1.07
  - UrlCon first: .95
  - p=0.69
- ANOVA - marginally significant interaction between order and doc type (p = 0.08)



Conclusions

- Plaiddoc is effective and usage barrier is small
  - Training was quick and required no specialized knowledge
  - 1-3 annotations per method
- Lightweight type annotations can
  - provide benefits as documentation alone
  - even for well-documented APIs

The End

Any Questions?

PLAID