# Automated Checking of Web Application Invocations

William G.J. Halfond

University of Southern California

# Traditional Invocation Verification

```
public void write(File outfile, String buffer, int length)
```

```
write(file, string, int)
```

```
write(file, string, string)
```

In contrast, web applications:

1. Invocations generated by string messages
2. Interfaces defined implicitly

# Example Invoking Component

```
void _jspService(Request req)
 1. print("<html><body>");
 2. print("<h1>Confirm Order</h1>");
 3. String oid = req.getParam("oid");
 4. int quant = getQuantity(oid);
 5. print("<form method=POST action='ProcessOrder'>");
 6. print("<input type=hidden value=" + oid + " name=oid>");
 7. print("<select name=shipto>");
 8. print("<option value=0>Billing Addr.</option>");
 9. print("<option value=1>Home Address</option>");
10. print("<option value=other>Alt.</option>");
11. print("</select>");
12. print("If other: <input type=text name=other>");
13. if  (canModify(oid))
14.             print("<p>Enter new quantity: </p>");
15.             print("<input type=text name=quant value="+ quant + ">");
16.             print("<input type=hidden value=modify " + "name=task>");
17.             print("<input type=submit value='Change" + " Quantity'>");
18. else
19.             print("<input type=hidden value=confirm " + "name=task>");
20.             print("<input type=submit value='Purchase'>");
21. print("</form></body></html>");
```

Takeaway points

1. Two paths in component
2. Six invocations
3. No explicit domain info

# Example Invoked Component

```
void doPost(Request req)
1. String oid = req.getParam("oid");
2. String task = req.getParam("task");
3. int shipOption = Integer.parse(req.getParam("shipto"));
4. String address=req.getParam("other");
5. switch (shipOption)
6.      case 1:
7.              address = getHomeAddress(oid);
8.              break;
9.      case 2:
10.             saveOtherAddress(oid, address);
11.             break;
12. if (task.equals("purchase"))
13.             submitOrder(oid, address);
14. if (task.equals("modify"))
15.             int quant = Integer.parse(req.getParam("quant"));
16.             modifyOrder(oid, quant);
17.             submitOrder(oid, address);
```

Takeaway points:

1. Two distinct interfaces
2. Implicit definitions
    1. Parameter names
    2. Parameters domains
    3. Groupings of parameters

# Invocation Errors

1. Unn
   - Pr … d for
2. Nu
   - …
3. …

```
 7.  print("<select name=shipto>");
 8.  print("<option value=0>Billing Addr.</option>");
 9.  print("<option value=1>Home Address</option>");
10.  print("<option value=other>Alt.</option>");
11.  print("</select>");
```

---

```
5. switch (shipOption)
```

```
19.  print("<input type=hidden value=confirm name=task>");
```

---

```
12. if (task.equals("purchase"))
      …
14. if (task.equals("modify"))
```

```
 7.  print("<select name=shipto>");
 8.  print("<option value=0>Billing Addr.</option>");
 9.  print("<option value=1>Home Address</option>");
10.  print("<option value=other>Alt.</option>");
11.  print("</select>");
```

---

```
3. int shipOption = Integer.parse(req.getParam("shipto"));
```

# The Approach

1. Compute Invocations
2. Identify Interfaces
3. Verify Invocations

# Step 1: Compute Invocations

**Input:** web application implementation

**Output:** set of invocations

- Argument {<name, type, value>+}
- Request method {GET|POST}
- Target

**How:**

a) Identify sets of HTML generating nodes

b) Extract and combine string values/domains from node sets

c) Parse extracted string content for syntax and domain information

# Step 1a – Group HTML Generating Nodes

$$\mathtt{Gen}[n] = \begin{cases} \{\{\}\} & \text{if n is method entry} \\ \{n\} & \text{if n generates output} \\ \{n\} & \text{if n is a callsite and target(n) has a summary} \\ \{\} & \text{otherwise} \end{cases}$$

$$\mathtt{In}[n] = \bigcup_{p \in pred(n)} \mathtt{Out}[p]$$

$$\mathtt{Out}[n] = \{p | \forall i \in \mathtt{In}[n], p \leftarrow \mathtt{append}(i, \mathtt{Gen}[n])\}$$

$$\mathtt{summary}(m) = \left\{ \forall s \in \mathtt{Out}[exitNode(m)] \prod_{n \in s} \mathtt{resolve}(n) \right\}$$

Nodes on path 1:
[1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 21]

print("<input type=submit value='Purchase'>");

print("<input type=text name=quant value="+ quant + ">");

Nodes on path 2:
[1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 19, 20, 21]

# Step 1b – Identify HTML Strings

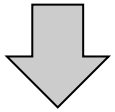
Resolve each node to FSA representing possible string values.

| Node | Statement | Possible Values |
|---|---|---|
| 1 | print("<html><body>"); | <html><body> |
| 5 | print("<form method=POST action='ProcessOrder'>"); | <form method=POST action='ProcessOrder'> |
| 6 | print("<input type=hidden value=" + oid + " name=oid>"); | <input type=hidden value=* name=oid> |
| 7 | print("<select name=shipto>"); | <select name=shipto> |
| … | …. | … |
| 21 | print("</form></body></html>"); | </form></body></html> |

# Step 1b – Identify Domain

*Key insight: certain nodes allow us to infer domain information about invocation values.*

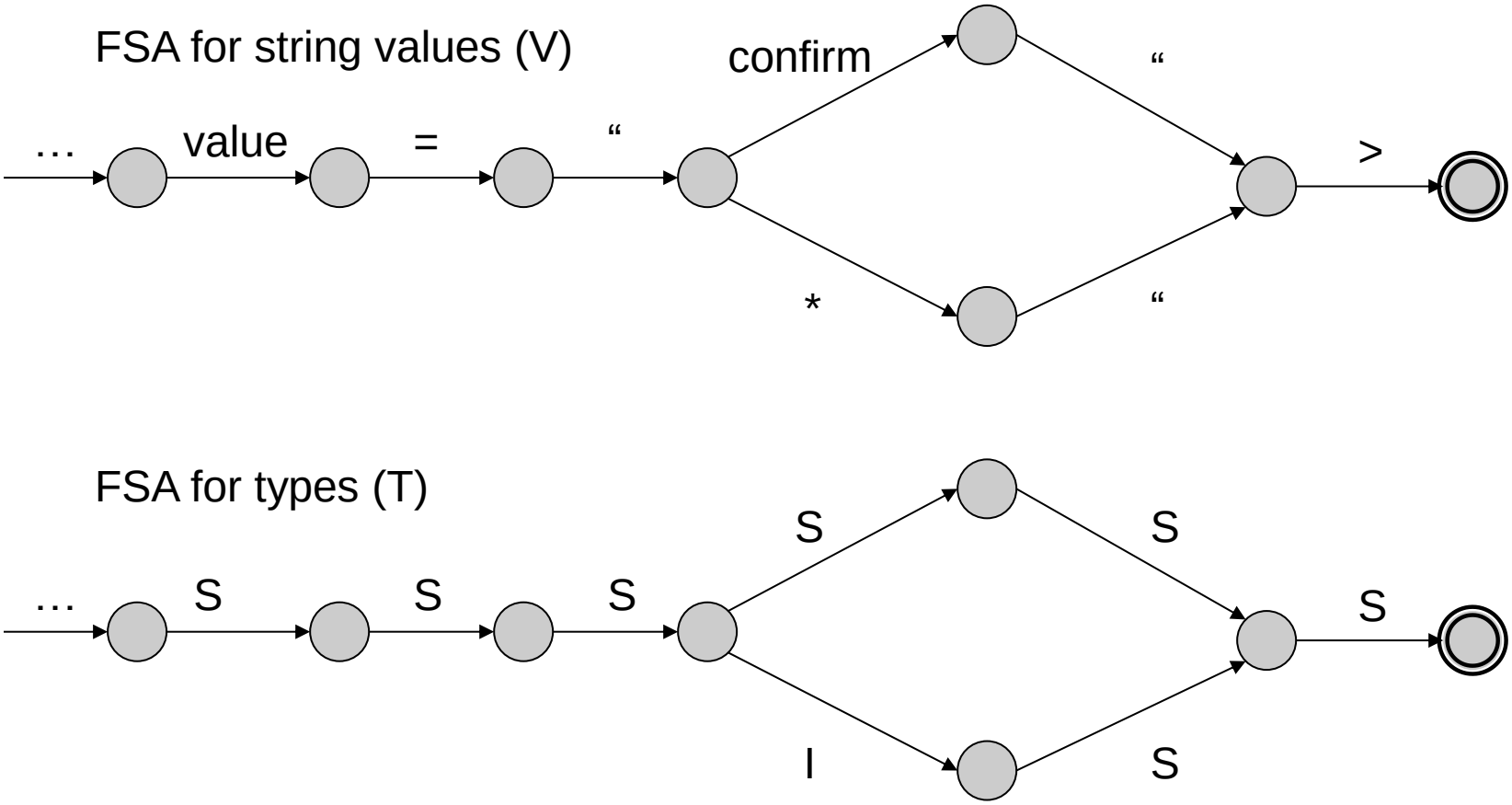
```
3.  String oid = req.getParam("oid");
4.  int quant = getQuantity(oid);
    …
15. print("<input type=text name=quant value="+ quant + ">");
```




Solution: generate two FSA, one for string values, one for inferred types

# Step 1b – FSA Example

```
<input type=text name=quant value="*">
<input type=text name=quant value="confirm">
```



# Step 1b – Domain Categories

```
print("<tag>" + expr + "</tag>");
```

- String constants

  expr ≡ "s"
- Member of a collection

  expr ≡ collection<t>[x]
- Functions that return a string

  expr ≡ object.toString()
- Convert basic type

  expr ≡ Type.toString()
- Append basic type

  expr ≡ append(Str, Type)

# Step 1b - Example

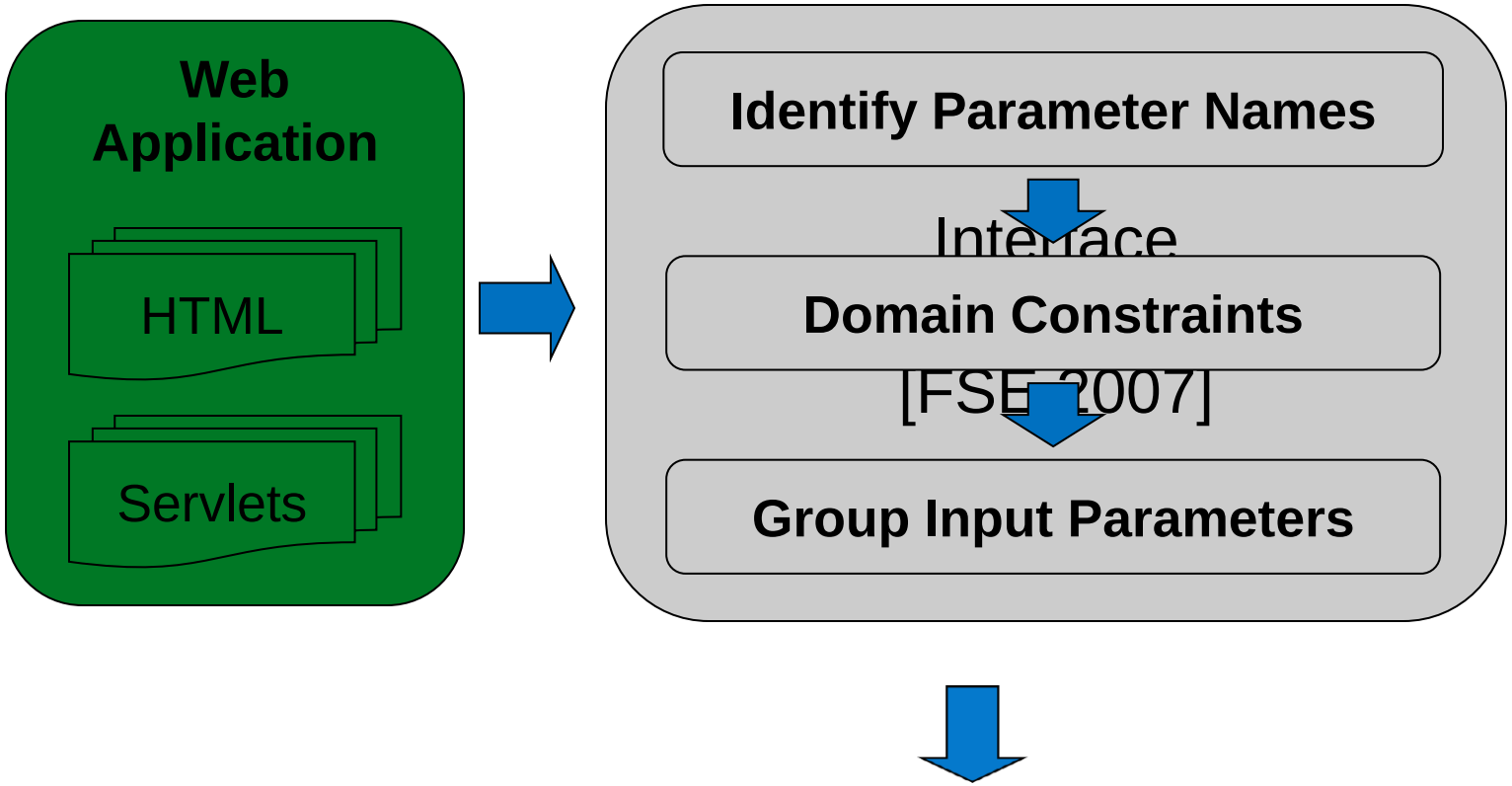
```
<html><body>
        <h1>Confirm Order</h1>
        <form method=POST action='ProcessOrder'>
                <input type=hidden value= * name=oid>
                <select name=shipto>
                        <option value= 0 >Billing Addr.</option>
                        <option value= 1 >Home Address</option>
                        <option value= other >Alt.</option>
                </select>
                If other: <input type=text name=other>
                <p>Enter new quantity: </p>
                <input type=text name=quant value= * >
                <input type=hidden value= modify name=task>
                <input type=submit value='Change Quantity'>
        </form>
</body></html>
```

# Step 1c: Parse HTML

- Identify syntactic elements that define invocations
- Extract substrings' corresponding domain info

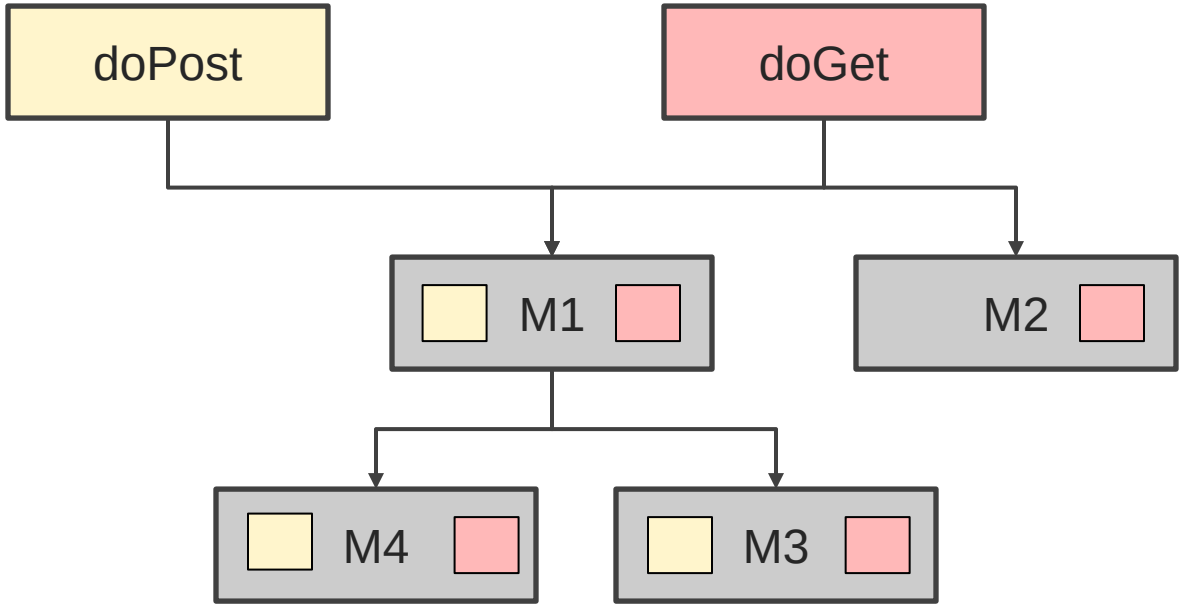
| # | Invocation Arguments |
|---|---|
| 1 | <oid, *, “”> <task, *, “modify”> <shipto, *, 0> <other, *, “”> <quant, INT, “”> |
| 2 | <oid, *, “”> <task, *, “modify”> <shipto, *, 1> <other, *, “”> <quant, INT, “”> |
| 3 | <oid, *, “”> <task, *, “modify”> <shipto, *, “other”> <other, *, “”> <quant, INT, “”> |
| 4 | <oid, *, “”> <task, *, “confirm”> <shipto, *, 0> <other, *, “”> |
| 5 | <oid, *, “”> <task, *, “confirm”> <shipto, *, 1> <other, *, “”> |
| 6 | <oid, *, “”> <task, *, “confirm”> <shipto, *,“other”> <other, *, “”> |

# Step 2: Identify Interface Information




| # | Interface Domain Constraints |
|---|---|
| 1 | int(shipto) && (shipto=1 \|\| shipto=2) && task=”purchase” |
| 2 | int(shipto) && (shipto=1 \|\| shipto=2) && task=”modify” && int(quant) |

# Interfaces: Identify Request Method




Mark interface elements with request methods that can reach them

# Step 3: Verification

Compare each invocation against its target's interfaces.

| | # | Invocation Arguments |
|---|---|---|
| ✗ | 1 | <oid, *, “”> <task, *, “modify”> <shipto, *, 0> <other, *, “”> <quant, INT, “”> |
| ✓ | 2 | <oid, *, “”> <task, *, “modify”> <shipto, *, 1> <other, *, “”> <quant, INT, “”> |
| ✗ | 3 | <oid, *, “”> <task, *, “modify”> <shipto, *, “other”> <other, *, “”> <quant, INT, “”> |
| ✗ | 4 | <oid, *, “”> <task, *, “confirm”> <shipto, *, 0> <other, *, “”> |
| ✗ | 5 | <oid, *, “”> <task, *, “confirm”> <shipto, *, 1> <other, *, “”> |
| ✗ | 6 | <oid, *, “”> <task, *, “confirm”> <shipto, *,“other”> <other, *, “”> |

| # | Interface Domain Constraints |
|---|---|
| 1 | int(shipto) && (shipto=1 \|\| shipto=2) && task=”purchase” |
| 2 | int(shipto) && (shipto=1 \|\| shipto=2) && task=”modify” && int(quant) |

# Empirical Evaluation

**RQ1:** How much time is needed to run the technique?

**RQ2:** What is the approach's precision in identifying domain-related invocation errors?

**RQ3:** How many new errors are identified as compared to previously known errors?

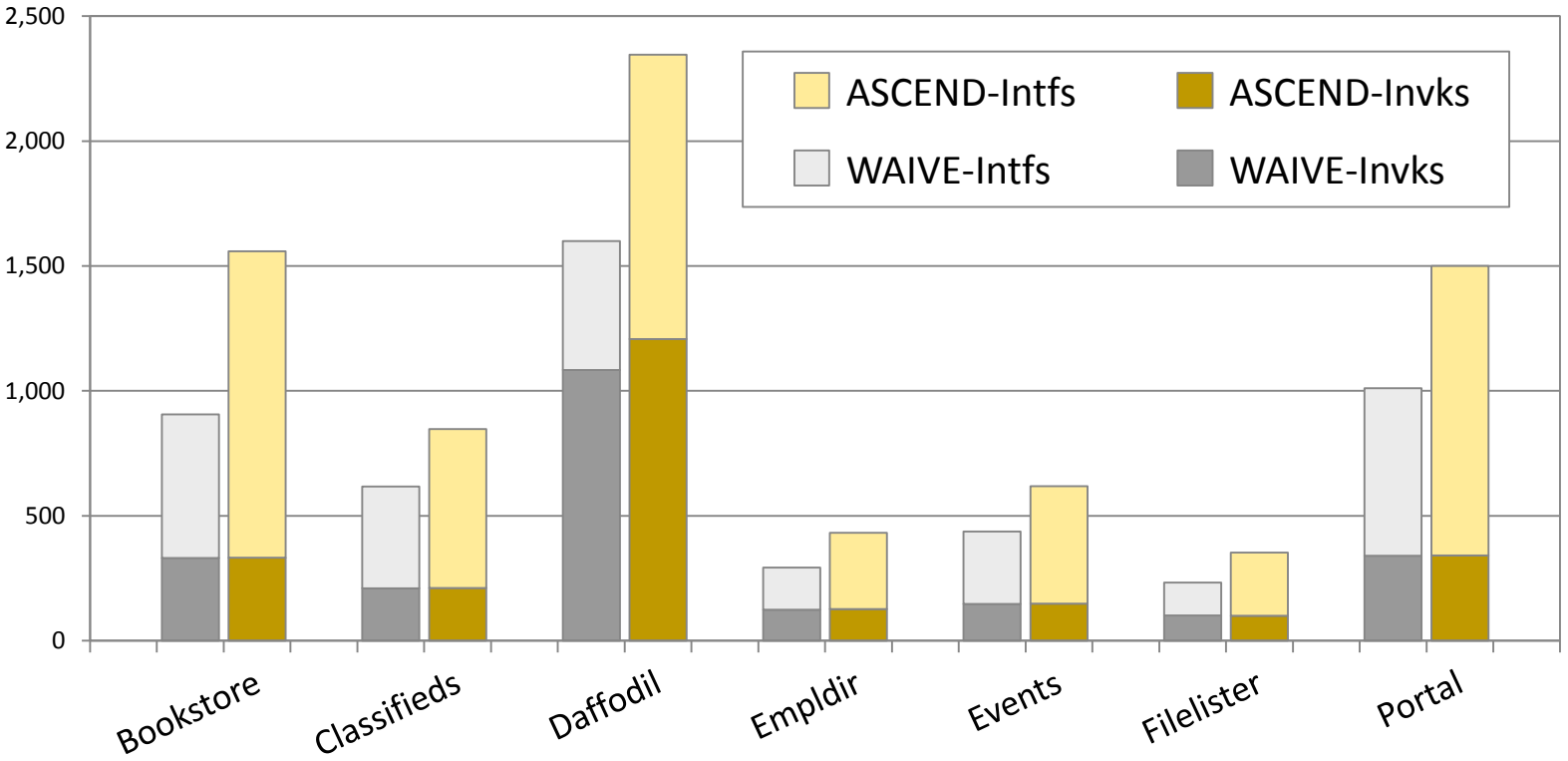
# Ascend Implementation of Approach

- Written in Java
- Analyzes bytecode of JEE web apps
- Interfaces: modified version of WAM [FSE 2007]
- Support libraries
  - Soot: call graphs
  - JSA: string resolution
  - HTML Parser: analyze HTML output
- Compare against WAIVE [FSE 2008]

# Evaluation Subjects

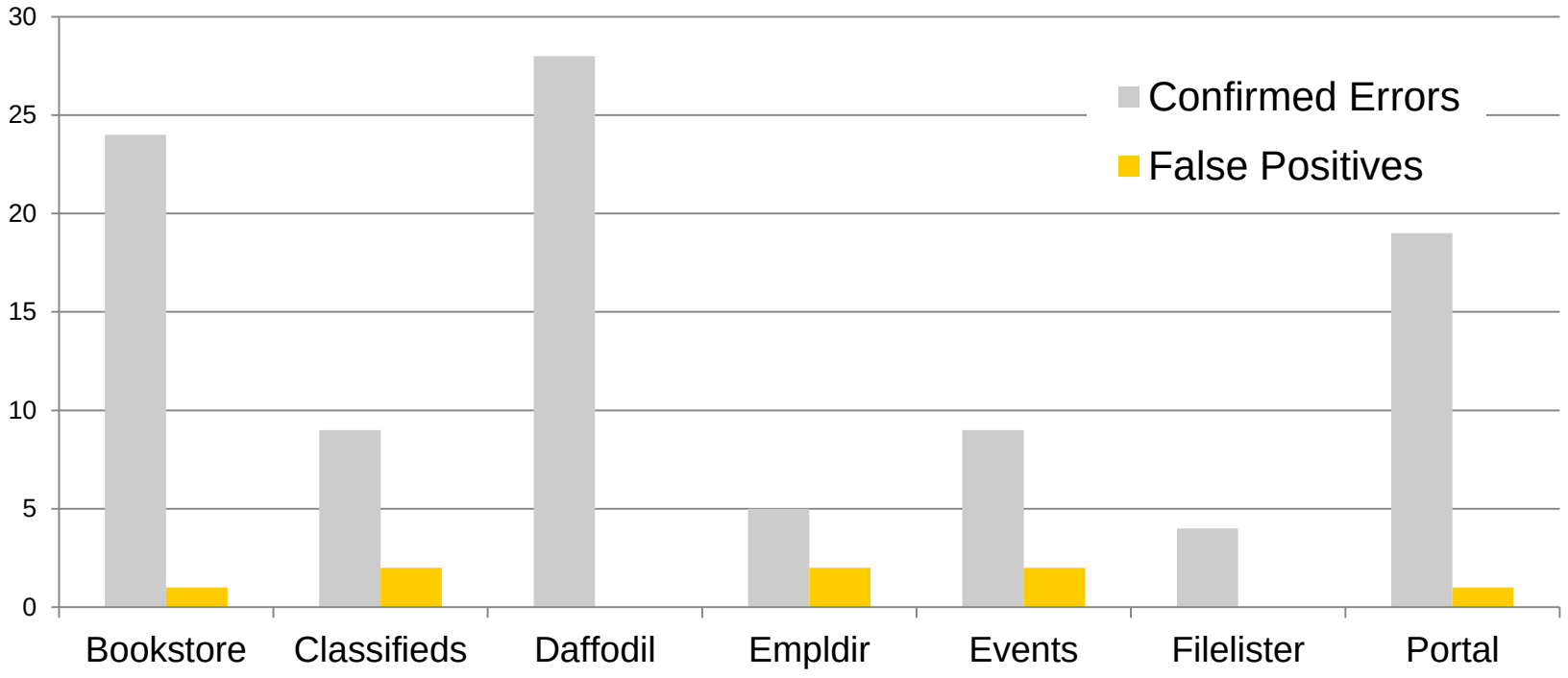
| Subject | Description | LOC | Classes |
|---|---|---|---|
| Bookstore | Online bookstore | 19,218 | 29 |
| Classifieds | Ad management | 11,203 | 19 |
| Daffodil | Customer DBMS | 19,236 | 121 |
| Empldir | Employee directory | 5,823 | 10 |
| Events | Event calendar | 7,327 | 13 |
| Filelister | File browser | 8,773 | 42 |
| Portal | Club management | 16,849 | 28 |

# RQ1: Analysis Time



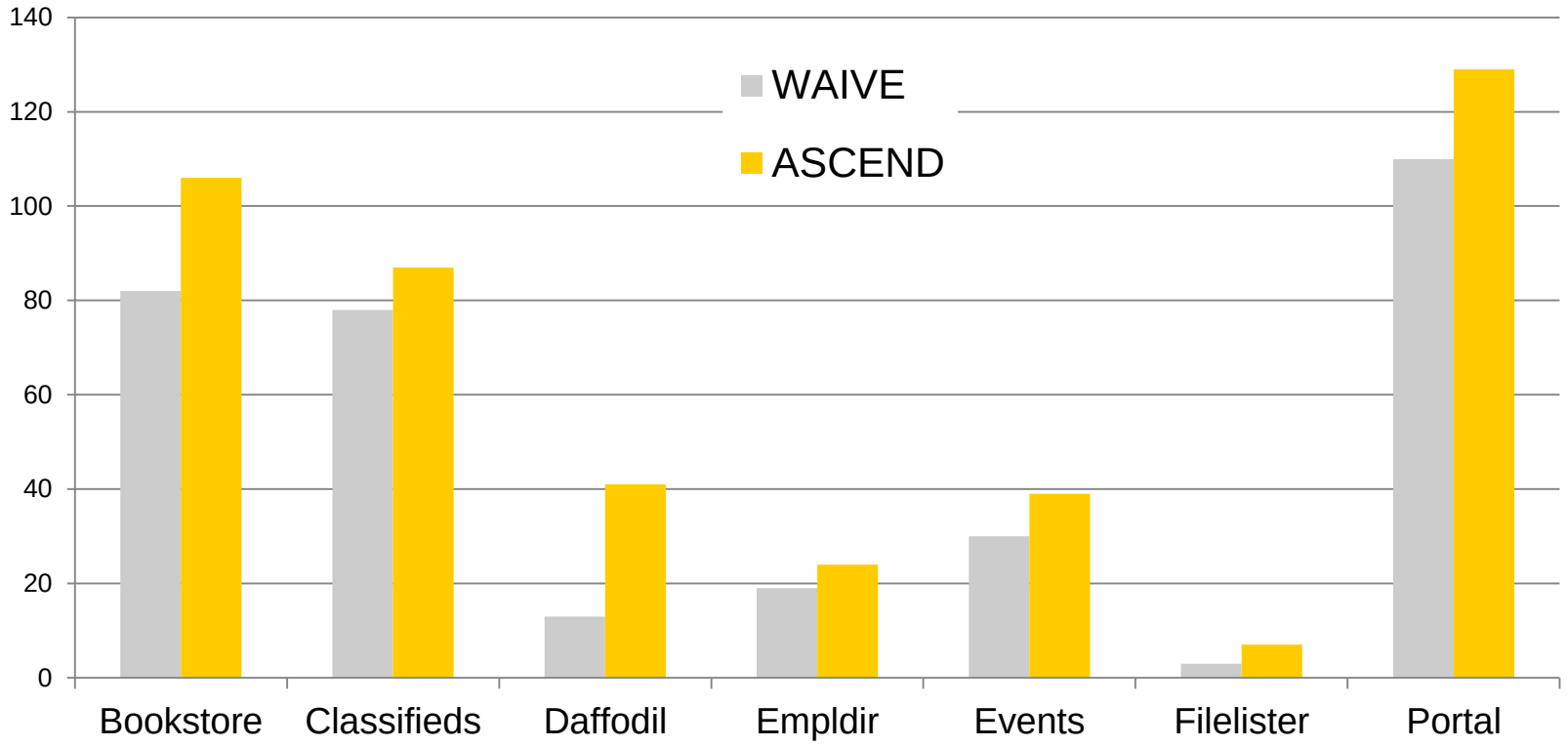

- Time ranged from six to forty minutes
- Primary cost increase over WAIVE was due to interface analysis

# RQ2: Precision




98 confirmed domain-related errors and 8 false positives

# RQ3: Comparison




- ASCEND found 433 errors versus 335 by WAIVE
- An increase of almost 30%

# Summary

- Verify web application invocations for names, type, values, and request methods
- Key insight for static analysis: string source allows us to infer domain information
- Evaluation shows
  - Reasonable time cost
  - Low false positives (7%)
  - More errors  found (30%)

**Thank You!**