Experimental Program Analysis

Andreas Zeller
Saarland University
This is precisely what our proposed approach produces: Given a program, we automatically produce a high-level specification. In the Z specification language the mined specification for `removeChild()` is shown in Figure:

```
removeChild : XMLElement
\Delta XMLElement
child? : XML_ELEMENT

child? ∈ enumerateChildren
child? ≠ null
enumerateChildren' = enumerateChildren \ child?
getChildrenCount' = getChildrenCount - 1
```

Note how the specification captures two important preconditions not stated in the documentation—that `child` be a child of the target node and that `child` be non-null. Both properties are essential for generating test cases. The postconditions precisely describe the effect of `removeChild()` and could be used as test oracles or as a base for program synthesis.

1d.3 State of the Art
1d.3.1 Static Analysis
How does one obtain a specification like this? Static analysis takes the program code and infers properties. The `removeChild()` code indeed reveals some insights: From this code, any static analysis can easily deduce precondition:

```
child? ∈ enumerateChildren
child? ≠ null
enumerateChildren' = enumerateChildren \ child?
getChildrenCount' = getChildrenCount - 1
```
This is precisely what our proposed approach produces: Given a program, we automatically produce a high-level specification. In the Z specification language, the mined specification for `removeChild()` is shown in Figure:

```
removeChild
XMLElement child?
\Delta XMLElement
child? : XML_ELEMENT

child? ∈ enumerateChildren
child? ≠ null
enumerateChildren' = enumerateChildren \ child?
getChildrenCount' = getChildrenCount - 1
```

Note how the specification captures two important preconditions not stated in the documentation— that `child` be a child of the target node and that `child` be non-null. Both properties are essential for generating test cases. The postconditions precisely describe the effect of `removeChild()`, and could be used as test oracles or as a base for program synthesis.
Specifications

Figure 1: The XMLElement class from the NanoXML parser

This is precisely what our proposed approach produces:

Given a program, we automatically produce a high-level specification. In the Z specification language, the mined specification for removeChild() is shown in Figure:

```plaintext
removeChild
XMLElement
child? : XML_ELEMENT

child? ∈ enumerateChildren
child? ≠ null
enumerateChildren' = enumerateChildren \ child?
getChildrenCount' = getChildrenCount - 1
```

Figure 2: Mined specification for removeChild as set forth in this proposal

Note how the specification captures two important preconditions not stated in the documentation—

- child be a child of the target node
- child be non-null

Both properties are essential for generating test cases. The postconditions precisely describe the effect of removeChild() and could be used as test oracles or as a base for program synthesis.

1d.3 State of the Art

1d.3.1 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties. The removeChild() code indeed reveals some insights:

From this code, any static analysis can easily deduce precondition:

child? = null

But how would fully automated testing...
This is precisely what our proposed approach produces:

Given a program, we automatically produce a high-level specification. In the Z specification language, the mined specification for `removeChild()` is shown in Figure:

```latex
\text{removeChild} : \text{XMLElement} \quad \text{XMLElement} \quad \text{XMLElement}

\text{child} \in \text{enumerateChildren}

\text{child} \neq \text{null}

\text{enumerateChildren}' = \text{enumerateChildren} \setminus \text{child}

\text{getChildrenCount}' = \text{getChildrenCount} - 1
```

Note how the specification captures two important preconditions not stated in the documentation—

- That \text{child} be a child of the target node.
- That \text{child} be non-\text{null}.

Both properties are essential for generating test cases. The postconditions precisely describe the effect of `removeChild()` and could be used as test oracles or as a base for program synthesis.

**1d.3 State of the Art**

**1d.3.1 Static Analysis**

How does one obtain a specification like this?

Static analysis takes the program code and infers properties. The `removeChild()` code indeed reveals some insights:

From this code, any static analysis can easily deduce the precondition:

$$\text{child} \in \text{enumerateChildren}$$

$$\text{child} \neq \text{null}$$

But how would...
Specifications

fully automated testing

fully automated debugging

widely automated verification

```java
public class XMLElement implements IXMLElement, Serializable {
    // The name.
    private String name;

    // The child elements.
    private Vector children;

    // Returns an enumeration of all child elements.
    public Enumeration enumerateChildren() {
        return children.enumerate();
    }

    // Returns the number of children.
    public int getChildrenCount() {
        return children.size();
    }

    // Removes a child element.
    public void removeChild(XMLElement child) {
        children.remove(child);
    }
}
```

Figure 1: The XMLElement class from the NanoXML parser

This is precisely what our proposed approach produces:

Given a program, we automatically produce a high-level specification. In the Z specification language, the mined specification for `removeChild()` is shown in Figure:

```
removeChild
XMLElement child
\n�� enumerateChildren
�� null = enumerateChildren
�� 0 = enumerateChildren \ child
�� getChildrenCount 0 = getChildrenCount 1
```

Figure 2: Mined specification for `removeChild` as set forth in this proposal

Note how the specification captures two important preconditions not stated in the documentation—

- that `child` be a child of the target node,
- that `child` be non-null.

Both properties are essential for generating test cases. The postconditions precisely describe the effect of `removeChild()` and could be used as test oracles or as a base for program synthesis.

1d.3 State of the Art
1d.3.1 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties. The `removeChild` code indeed reveals some insights:

From this code, any static analysis can easily deduce precondition:

- `child != null`

But how would
"Without specification, there are no bugs—only surprises"

Brian Kernighan
SPECMATE: Specification Mining and Testing

Principal Investigator (PI): Andreas Zeller
PI’s host institution: Saarland University
Project duration: 60 months

SPECMATE Project Summary

In the past decade, automated validation of software systems has made spectacular progresses. On the testing side, it is now possible to automatically generate test cases that effectively explore the entire program structure; on the verification side, we can now formally prove properties for software as complex as operating systems. To push validation further, however, we need specifications of what the software actually should do. But writing such specifications has always been hard—and so far significantly inhibited the deployment of rigorous development methods.

The SPECMATE methodology automatically extracts such specifications from existing systems, effectively leveraging the knowledge encoded into billions of code lines. SPECMATE starts with just an executable program and automatically produces an incremental specification, starting with the most relevant properties; and a set of test cases fully covering the specification.

SPECMATE will boost quality and productivity in all software development activities, in particular:

Verification and modeling as the specifications mined are high-level and incremental, and thus form an ideal starting point for compositional modeling and verification—enabling the rigorous construction and derivation of new, safe, dependable software systems;

Testing as SPECMATE produces a full-fledged test suite for free: rather than manually exploring the system and its concrete executions, the programmer only needs to validate the mined high-level specifications against the (implicitly) intended behavior;

Defect detection since the mined specifications also reveal undesired properties: every such property comes with a test case demonstrating it;

Program maintenance as it eases program understanding and change impact assessment: every aspect of the program behavior will be described in a high-level, abstract specification.
ERC Advanced Grant — Research Proposal

SPECMATE: Specification Mining and Testing

Principal Investigator (PI): Andreas Zeller
PI's host institution: Saarland University
Project duration: 60 months

SPECMATE Project Summary

In the past decade, automated validation of software systems has made spectacular progresses. On the testing side, it is now possible to automatically generate test cases that effectively explore the entire program structure; on the verification side, we can now formally prove properties for software as complex as operating systems. To push validation further, however, we need specifications of what the software actually should do. But writing such specifications has always been hard—and so far significantly inhibited the deployment of rigorous development methods.

The SPECMATE methodology automatically extracts such specifications from existing systems, effectively leveraging the knowledge encoded into billions of code lines. SPECMATE starts with just an executable program and automatically produces an incremental specification, starting with the most relevant properties; and a set of test cases fully covering the specification.

SPECMATE will boost quality and productivity in all software development activities, in particular:

- **Verification and modeling** as the specifications mined are high-level and incremental, and thus form an ideal starting point for compositional modeling and verification—enabling the rigorous construction and derivation of new, safe, dependable software systems;
- **Testing** as SPECMATE produces a full-fledged test suite for free: rather than manually exploring the system and its concrete executions, the programmer only needs to validate the mined high-level specifications against the (implicitly) intended behavior;
- **Defect detection** since the mined specifications also reveal undesired properties: every such property comes with a test case demonstrating it;
- **Program maintenance** as it eases program understanding and change impact assessment: every aspect of the program behavior will be described in a high-level, abstract specification.

---

(a) Executable Program

```java
public class XMLElement implements IXMLElement, Serializable {
    // The name.
    private String name;

    // The child elements.
    private Vector children;

    // Returns an enumeration of all child elements.
    public Enumeration enumerateChildren() { ... }

    // Returns the number of children.
    public int getChildrenCount() { ... }

    // Removes a child element.
    public void removeChild(IXMLElement child) { ... }

    // More methods and attributes...
}
```

(b) Specification

```plaintext
public void testRemoveChild()
{
    child = element.getChildAtIndex(0);
    element.removeChild(child);
    assertEquals(element.getChildrenCount(),
                 getChildrenCount - 1);
}
```

---

(c) Test

Defect detection since the mined specifications also reveal undesired properties: every such property comes with a test case demonstrating it;

Program maintenance as it eases program understanding and change impact assessment: every aspect of the program behavior will be described in a high-level, abstract specification.
Static Analysis

- Originates from *compiler optimization*
- Considers *all possible* executions
- Can prove *universal properties*
- Tied to *symbolic verification* techniques

Dynamic Analysis

- Originates from *execution monitoring*
- Considers (only) *actual* executions
- Covers all abstraction layers
- Tied to *run-time verification* techniques
Static Analysis
requires perfect knowledge

- Originates from compiler optimization
- Considers all possible executions
- Can prove universal properties
- Tied to symbolic verification techniques

Dynamic Analysis

- Originates from execution monitoring
- Considers (only) actual executions
- Covers all abstraction layers
- Tied to run-time verification techniques
Static Analysis
requires perfect knowledge

- Originates from *compiler optimization*
- Considers *all possible* executions
- Can prove *universal properties*
- Tied to *symbolic verification* techniques

Dynamic Analysis
limited to observed runs

- Originates from *execution monitoring*
- Considers (only) *actual* executions
- Covers all abstraction layers
- Tied to *run-time verification* techniques
Dynamic Analysis
limited to observed runs

- Originates from execution monitoring
- Considers (only) actual executions
- Covers all abstraction layers
- Tied to run-time verification techniques
limited to observed runs
need more runs
Generate test cases to systematically explore behavior
Generate test cases to systematically explore behavior

Assess executions to learn about software behavior
Generate test cases to systematically explore behavior

Assess executions to learn about software behavior
Generate test cases to systematically explore behavior

Assess executions to learn about software behavior

specifications
Enriching specifications
Enriching specifications

```java
void ProtocolTest() {
    Protocol p = new ...
    p.conn();
    p.send(x);
    p.quit();
}
```
Enriching specifications

Execute and extract initial spec
Enriching specifications

Execute and extract initial spec

Generate test mutants and enrich specs

Dallmeier et al: “Generating Test Cases for Specification Mining”, ISSTA 2010
void ProtocolTest() {
  Protocol p = new ...
  p.conn();
  p.send(x);
  p.quit();
}

Dallmeier et al: “Generating Test Cases for Specification Mining”, ISSTA 2010
void ProtocolTest() {
    Protocol p = new ...
    p.conn();
    p.send(x);
    p.quit();
}
void ProtocolTest() {
    Protocol p = new ...
    p.conn();
    p.send(x);
    p.quit();
}

Uncovered

0: send(x)
   quit()

1: conn()
void ProtocolTest() {
    Protocol p = new ...
    p.conn();
    p.send(x);
    p.quit();
}

Uncovered
0: send(x)
   quit()
1: conn()

Dallmeier et al: “Generating Test Cases for Specification Mining”, ISSTA 2010
void TestMutant1() {
    Protocol p = new ...
    p.conn();
    p.send(x);
    p.conn();
    p.quit();
}

Dallmeier et al: “Generating Test Cases for Specification Mining”, ISSTA 2010
void ProtocolTest() {
    Protocol p = new ...
    p.conn();
    p.send(x);
    p.quit();
}

void TestMutant1() {
    Protocol p = new ...
    p.conn();
    p.send(x);
    p.conn();
    p.quit();
}

Dallmeier et al: “Generating Test Cases for Specification Mining”, ISSTA 2010
void ProtocolTest() {
    Protocol p = new ...
p.conn();
p.send(x);
p.quit();
}

Uncovered
0:  send(x)
    quit()
1:  conn()

Dallmeier et al: “Generating Test Cases for Specification Mining”, ISSTA 2010
Dallmeier et al: “Generating Test Cases for Specification Mining”, ISSTA 2010
void ProtocolTest() {
    Protocol p = new ...
    p.conn();
    p.send(x);
    p.quit();
}

void TestMutant2() {
    Protocol p = new ...
    //p.conn();
    p.send(x);
    p.quit();
}
void ProtocolTest() {
    Protocol p = new ...
    p.conn();
    p.send(x);
    p.quit();
}

void TestMutant2() {
    Protocol p = new ...
    //p.conn();
    p.send(x);
    p.quit();
}

Dallmeier et al: “Generating Test Cases for Specification Mining”, ISSTA 2010
Dallmeier et al: “Generating Test Cases for Specification Mining”, ISSTA 2010
Dallmeier et al: “Generating Test Cases for Specification Mining”, ISSTA 2010
A new kind of Analysis
A new kind of Analysis

- Static analysis
A new kind of Analysis

- Static analysis
- 0 runs
A new kind of Analysis

- Static analysis
- Dynamic analysis
A new kind of Analysis

- Static analysis
- Dynamic analysis

- 0 runs
- n given runs
A new kind of Analysis

- Static analysis
- Dynamic analysis
- Experimental analysis

- 0 runs
- n given runs
A new kind of Analysis

- Static analysis
- Dynamic analysis
- Experimental analysis
- 0 runs
- \( n \) given runs
- \( n \) generated runs
A new kind of Analysis

- Static analysis
- Dynamic analysis
- Experimental analysis

- 0 runs
- \( n \) given runs
- \( n \) generated runs
Generate test cases to systematically explore behavior

Assess executions to learn about software behavior
Generate test cases to systematically explore behavior. Assess executions to learn about software behavior. Are these real executions?
public class RandoopTest0 extends TestCase {
    ...
    public void test8() throws Throwable {
        if (debug) System.out.printf("%nRandoopTest0.test8");

        AddressBook var0 = new AddressBook();
        EventHandler var1 = var0.getEventHandler();
        Category var2 = var0.getRootCategory();
        Contact var3 = new Contact();
        AddressBook var4 = new AddressBook();
        EventHandler var5 = var4.getEventHandler();
        Category var6 = var4.getRootCategory();
        String var7 = var6.getName();
        var0.addCategory(var3, var6);
        SelectionHandler var9 = new SelectionHandler();
        AddressBook var10 = new AddressBook();
        EventHandler var11 = var10.getEventHandler();
        Category var12 = var10.getRootCategory();
        String var20 = var19.getName();
        var15.categorySelected(var19);
        var9.addCategorySelectionListener((CategorySelectionListener)var15);
        ContactTablePanel var23 = new ContactTablePanel(var0, var9);
        AddressBook var24 = new AddressBook();
        EventHandler var25 = var24.getEventHandler();
        Category var26 = var24.getRootCategory();
        AddressBook var26 = new AddressBook();
        EventHandler var27 = var26.getEventHandler();
        Category var28 = var26.getRootCategory();
        Contact var29 = new Contact();
        AddressBook var30 = new AddressBook();
        EventHandler var31 = var30.getEventHandler();
        Category var32 = var30.getRootCategory();
        String var33 = var32.getName();
        var26.addCategory(var29, var32);
        SelectionHandler var35 = new SelectionHandler();
        AddressBook var36 = new AddressBook();
        EventHandler var37 = var36.getEventHandler();
        Category var38 = var36.getRootCategory();
        EventHandler var39 = var36.getEventHandler();
        SelectionHandler var40 = new SelectionHandler();
        actions.CreateContactAction var41 = new actions.CreateContactAction(var36, var40);
        boolean var42 = var41.isEnabled();
        AddressBook var43 = new AddressBook();
        EventHandler var44 = var43.getEventHandler();
        Category var45 = var43.getRootCategory();
        String var46 = var45.getName();
        var41.categorySelected(var45);
        var35.addCategorySelectionListener((CategorySelectionListener)var41);
        ContactTablePanel var49 = new ContactTablePanel(var26, var35);
        CategoryTreePanel var50 = new CategoryTreePanel(var24, var35);
        actions.CreateCategoryAction var51 = new actions.CreateCategoryAction(var0, var35);
        AddressBook var52 = new AddressBook();
        Category var53 = var52.getRootCategory();
        AddressBook var54 = new AddressBook();
        EventHandler var55 = var54.getEventHandler();
        Category var56 = var54.getRootCategory();
        EventHandler var57 = var54.getEventHandler();
        SelectionHandler var58 = new SelectionHandler();
        ContactEditionPanel var59 = new ContactEditionPanel(var54, var58);
        JPanel var60 = var59.getPanel();
        JFrame var61 = samples.utils.SampleUtils.createFrame((JComponent)var60);
        CategorySelectionDialog var62 = new CategorySelectionDialog(var52, (java.awt.Frame)var61);
        CategorySelectionDialog var63 = new CategorySelectionDialog(var0, (java.awt.Frame)var61);
        MainWindow var64 = new MainWindow(var0);
        AddressBook var65 = new AddressBook();
        EventHandler var66 = var65.getEventHandler();
        Category var67 = var65.getRootCategory();
        Contact var68 = new Contact();
        Category[] var69 = var68.getCategories();
        var65.removeContact(var68);
        java.util.List var71 = var65.getContacts();
        AddressBook var72 = new AddressBook();
        EventHandler var73 = var72.getEventHandler();
        Category var74 = var72.getRootCategory();
        EventHandler var75 = var72.getEventHandler();
        SelectionHandler var76 = new SelectionHandler();
        actions.CreateContactAction var77 = new actions.CreateContactAction(var72, var76);
        boolean var78 = var77.isEnabled();
        AddressBook var79 = new AddressBook();
        EventHandler var80 = var79.getEventHandler();
        Category var81 = var79.getRootCategory();
        String var82 = var81.getName();
        var77.categorySelected(var81);
        Category var85 = var65.createCategory(var81, "hi!");
        String var86 = var85.toString();
        Category var88 = var0.createCategory(var85, "exceptions.NameAlreadyInUseException");
    }
}
public class RandoopTest0 extends TestCase {

...
public class RandoopTest0 extends TestCase {
    
    public void test8() throws Throwable {
        if (debug) System.out.printf("%nRandoopTest0.test8");
        AddressBook var0 = new AddressBook();
        EventHandler var1 = var0.getEventHandler();
        Category var2 = var0.getRootCategory();
        Contact var3 = new Contact();
        AddressBook var4 = new AddressBook();
        EventHandler var5 = var4.getEventHandler();
        Category var6 = var4.getRootCategory();
        String var7 = var6.getName();
        var0.addCategory(var3, var6);
        SelectionHandler var9 = new SelectionHandler();
        AddressBook var10 = new AddressBook();
        EventHandler var11 = var10.getEventHandler();
        Category var12 = var10.getRootCategory();
        EventHandler var13 = var10.getEventHandler();
        SelectionHandler var14 = new SelectionHandler();
        actions.CreateContactAction var15 = new actions.CreateContactAction(var10, var14);
        boolean var16 = var15.isEnabled();
        AddressBook var17 = new AddressBook();
        EventHandler var18 = var17.getEventHandler();
        Category var19 = var17.getRootCategory();
        String var20 = var19.getName();
        var15.categorySelected(var19);
        var9.addCategorySelectionListener((CategorySelectionListener)var15);
        ContactTablePanel var23 = new ContactTablePanel(var0, var9);
        AddressBook var24 = new AddressBook();
        Category var25 = var24.getRootCategory();
        AddressBook var26 = new AddressBook();
        EventHandler var27 = var26.getEventHandler();
        Category var28 = var26.getRootCategory();
        Contact var29 = new Contact();
        AddressBook var30 = new AddressBook();
        EventHandler var31 = var30.getEventHandler();
        Category var32 = var30.getRootCategory();
        String var33 = var32.getName();
        var26.addCategory(var29, var32);
        SelectionHandler var35 = new SelectionHandler();
        AddressBook var36 = new AddressBook();
        EventHandler var37 = var36.getEventHandler();
        Category var38 = var36.getRootCategory();
        EventHandler var39 = var36.getEventHandler();
        SelectionHandler var40 = new SelectionHandler();
        actions.CreateContactAction var41 = new actions.CreateContactAction(var36, var40);
        boolean var42 = var41.isEnabled();
        AddressBook var43 = new AddressBook();
        EventHandler var44 = var43.getEventHandler();
        Category var45 = var43.getRootCategory();
        String var46 = var45.getName();
        var41.categorySelected(var45);
        var35.addCategorySelectionListener((CategorySelectionListener)var41);
        ContactTablePanel var49 = new ContactTablePanel(var26, var35);
        CategoryTreePanel var50 = new CategoryTreePanel(var24, var35);
        actions.CreateCategoryAction var51 = new actions.CreateCategoryAction(var0, var35);
        AddressBook var52 = new AddressBook();
        Category var53 = var52.getRootCategory();
        AddressBook var54 = new AddressBook();
        EventHandler var55 = var54.getEventHandler();
        Category var56 = var54.getRootCategory();
        EventHandler var57 = var54.getEventHandler();
        SelectionHandler var58 = new SelectionHandler();
        ContactEditionPanel var59 = new ContactEditionPanel(var54, var58);
        JPanel var60 = var59.getPanel();
        JFrame var61 = samples.utils.SampleUtils.createFrame((JComponent)var60);
        CategorySelectionDialog var62 = new CategorySelectionDialog(var52, (java.awt.Frame)var61);
        CategorySelectionDialog var63 = new CategorySelectionDialog(var0, (java.awt.Frame)var61);
        MainWindow var64 = new MainWindow(var0);
        AddressBook var65 = new AddressBook();
        EventHandler var66 = var65.getEventHandler();
        Category var67 = var65.getRootCategory();
        Contact var68 = new Contact();
        Category[] var69 = var68.getCategories();
        var65.removeContact(var68);
        java.util.List var71 = var65.getContacts();
        AddressBook var72 = new AddressBook();
        EventHandler var73 = var72.getEventHandler();
        Category var74 = var72.getRootCategory();
        EventHandler var75 = var72.getEventHandler();
        SelectionHandler var76 = new SelectionHandler();
        actions.CreateContactAction var77 = new actions.CreateContactAction(var72, var76);
        boolean var78 = var77.isEnabled();
        AddressBook var79 = new AddressBook();
        EventHandler var80 = var79.getEventHandler();
        Category var81 = var79.getRootCategory();
        String var82 = var81.getName();
        var77.categorySelected(var81);
        Category var85 = var65.createCategory(var81, "hi!");
        String var86 = var85.toString();
        Category var88 = var0.createCategory(var85, "exceptions.NameAlreadyInUseException");
    }
}
public class RandoopTest0 extends TestCase {
    public void test8() throws Throwable {
        if (debug) System.out.printf("%nRandoopTest0.test8");

        AddressBook a1 = new AddressBook();
        AddressBook a2 = new AddressBook();
        Category a1c = a1.createCategory(a1.getRootCategory(), "a1c");
        Category a2c = a2.createCategory(a1c, "a2c");
    }
}

<table>
<thead>
<tr>
<th>First name</th>
<th>Last name</th>
<th>E-mail</th>
<th>Phone</th>
<th>Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>James S.</td>
<td>Roebeck</td>
<td>JamesS Roebeck</td>
<td>561-888-1</td>
<td>561-888-1</td>
</tr>
<tr>
<td>Naomi D.</td>
<td>Long</td>
<td>NaomiD Long</td>
<td>390-12-5</td>
<td>390-12-1</td>
</tr>
<tr>
<td>Karen L.</td>
<td>Lloyd</td>
<td>KarenLLloyd</td>
<td>228-76-1</td>
<td>228-76-1</td>
</tr>
<tr>
<td>Jean R.</td>
<td>Voigt</td>
<td>JeanRVoigt</td>
<td>610-344-1</td>
<td>610-344-1</td>
</tr>
<tr>
<td>Douglas L.</td>
<td>Green</td>
<td>DouglasLG</td>
<td>612-615-1</td>
<td>612-615-1</td>
</tr>
</tbody>
</table>

**New Contact**

- **First name**: Karen L.
- **E-Mail**: KarenLLloyd@ex
- **Last name**: Lloyd
- **Second e-mail**: Karen@CreditCa
- **Phone**: 228-76-1230
- **URL**: http://www.crec
- **Mobile**: 228-76-8710
- **Notes**: 1673 Jehovah Drive, Fredericksburg, VA 22408
how many addressbooks?
112 failures
112 failures
0 problems
<table>
<thead>
<tr>
<th>First name</th>
<th>Last name</th>
<th>E-mail</th>
<th>Phone</th>
<th>Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>James S.</td>
<td>Roebuck</td>
<td>JamesSRoe...</td>
<td>561-888...</td>
<td>561-888...</td>
</tr>
<tr>
<td>Naomi D.</td>
<td>Long</td>
<td>NaomiDLo...</td>
<td>390-12-5...</td>
<td>390-12-1...</td>
</tr>
<tr>
<td>Karen L.</td>
<td>Lloyd</td>
<td>KarenLLlloyd...</td>
<td>228-76-1...</td>
<td>228-76-1...</td>
</tr>
<tr>
<td>Jean R.</td>
<td>Voigt</td>
<td>JeanRVoigt...</td>
<td>610-344...</td>
<td>610-344...</td>
</tr>
<tr>
<td>Douglas L.</td>
<td>Green</td>
<td>DouglasLG...</td>
<td>612-615...</td>
<td>612-615...</td>
</tr>
</tbody>
</table>

**New Contact: Karen L. Lloyd**
- **E-Mail:** KarenLLlloyd@ex
- **Phone:** 228–76–1230
- **URL:** http://www.crec
- **Notes:** 1673 Jehovah Drive
  Fredericksburg, VA 22408
<table>
<thead>
<tr>
<th>First name</th>
<th>Last name</th>
<th>E-mail</th>
<th>Phone</th>
<th>Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>James S.</td>
<td>Roebuck</td>
<td>JamesSRoe...</td>
<td>561-888---</td>
<td>561-888---</td>
</tr>
<tr>
<td>Naomi D.</td>
<td>Long</td>
<td>NaomiDLo...</td>
<td>390-12-5---</td>
<td>390-12-1---</td>
</tr>
<tr>
<td>Karen L.</td>
<td>Lloyd</td>
<td>KarenLLlo...</td>
<td>228-76-1---</td>
<td>228-76-7---</td>
</tr>
<tr>
<td>Jean R.</td>
<td>Voigt</td>
<td>JeanRVoigt...</td>
<td>610-344---</td>
<td>610-344---</td>
</tr>
<tr>
<td>Douglas L.</td>
<td>Green</td>
<td>DouglasLG...</td>
<td>612-615---</td>
<td>612-615---</td>
</tr>
</tbody>
</table>

- Contractors
- Customers
- Employees
- Suppliers
  - Europe
  - U.S.

**New contact**

- **First name:** Karen L.
- **E-mail:** KarenLLloyd@ex
- **Phone:** 228–76–1230
- **Second e-mail:** Karen@CreditCa
- **URL:** http://www.crec
- **Mobile:** 228–76–8710
- **Notes:** 1673 Jehovah Drive
  Fredericksburg, VA 22408
Quotes

My method is to take the utmost trouble to find the right thing to say, and then to say it without levity. Answers to Nine Questions (September 1896), answers to nine questions submitted by M. D. H. who had interviewed him in 1895.

We have no more right to consume happiness without producing it than to consume wealth without producing it. Candida, Act I (1898)

I'm only a beer teetotaler, not a champagne teetotaler. I don't like beer. Candida, Act III

We don't bother much about dress and manners in England, because as a nation we don't dress well and have no manners. You Never Can Tell, Act I (1898)

The great advantage of a hotel is that it's a refuge from home life. You Never Can Tell, Act I

My specialty is being right when other people are wrong. You Never Can Tell, Act IV

There is only one religion, though there are a hundred versions of it. Plays Pleasant and Other Plays, preface (1898)
Quotes

- My method is to take the utmost levity. Answers to Nine Questions which I had interviewed him in 1895.
- We have no more right to control producing it. Candida, Act I
- I'm only a beer teetotaler, not a teetotaler
- We don't bother much about dining manners. You Never Can Tell
- The great advantage of a hotel
- My specialty is being right
- There is only one religion, though there are a hundred versions of it. Plays Pleasant and the Preface (1898)
Today's Keynote: Search-Based Program Analysis
Search-based System Testing

Joint work with Florian Gross and Gordon Fraser
Search-based System Testing

• Generate tests at the user interface level

Joint work with Florian Gross and Gordon Fraser
Search-based System Testing

- Generate tests at the user interface level
- Aim for code coverage and GUI coverage

Joint work with Florian Gross and Gordon Fraser
Search-based System Testing

- Generate tests at the user interface level
- Aim for code coverage and GUI coverage
- Synthesize artificial input events

Joint work with Florian Gross and Gordon Fraser
Search-based System Testing

- Generate tests at the user interface level
- Aim for code coverage and GUI coverage
- Synthesize artificial input events
- Any test generated is a valid input

Joint work with Florian Gross and Gordon Fraser
Expanding this to Android, Metro, Web

See formal demo at ICSE 2012
Coverage achieved

- Randoop
- Exsyst
Coverage achieved

100 %

75 %

50 %

25 %

0 %

Addressbook  Calculator  TerpSpreadSheet  TerpWord  TerpPresent

Randoop

Exsyst
Coverage achieved

- Addressbook: 100%
- Calculator: 0%
- TerpSpreadSheet: 75%
- TerpWord: 0%
- TerpPresent: 0%

Comparison of coverage between Randoop and Exsyst.
Coverage achieved

- Addressbook: 50% (Randoop)
- Calculator: 25% (Randoop)
- TerpSpreadSheet: 100% (Exsyst)
- TerpWord: 0% (Randoop)
- TerpPresent: 0% (Randoop)
Coverage achieved
Coverage achieved

- **Addressbook**: 75%
- **Calculator**: 50%
- **TerpSpreadSheet**: 25%
- **TerpWord**: 50%
- **TerpPresent**: 75%

**Tools**
- Randoop
- Exsyst
Coverage achieved

Coverage for different applications:
- Addressbook: 100%
- Calculator: 100%
- TerpSpreadSheet: 75%
- TerpWord: 50%
- TerpPresent: 25%
Generate test cases to systematically explore behavior

Assess executions to learn about software behavior

specifications
Generate test cases to systematically explore behavior

Assess executions to learn about software behavior

real executions

specifications
Generate test cases to systematically explore behavior.

Assess executions to learn about software behavior.

real executions

real specifications
Carving Invariants

(a) Executable Program

```java
public class XMLElement implements IXMLElement6 {
    // The name.
    private String name;

    // The child elements.
    private Vector children;

    // Returns an enumeration of all child elements.
    public Enumeration enumerateChildren() { ... }

    // Returns the number of children.
    public int getChildrenCount() { ... }

    // Removes a child element.
    public void removeChild(IXMLElement child) { ... }

    // More methods and attributes...
}
```

(b) Specification

```
XMLElement
 rencont
 child? : XML_ELEMENT

removeChild

child? ∈ enumerateChildren
child? ≠ null
enumerateChildren' = enumerateChildren \ child?
getChildrenCount' = getChildrenCount − 1
```

(c) Test

```
public void testRemoveChild() {
    child = element.getChildAtIndex(0);
    element.removeChild(child);
    assertEquals(element.getChildrenCount(),
                old_getChildrenCount - 1);
}
```
Carving Invariants

(a) Executable Program

(b) Specification

(c) Test

```java
public void testRemoveChild()
{
    child = element.getFirstChild();
    element.removeChild(child);
    assertEquals(element.getChildrenCount(),
                 old_getChildrenCount - 1);
}
```
Challenges
Challenges

- Finding relevant specifications
  Ranking wrt usage, bug-finding capabilities
Challenges

• Finding relevant specifications
  Ranking wrt usage, bug-finding capabilities

• Expressing specifications
  Choosing a generic, domain-specific vocabulary
Challenges

- Finding relevant specifications
  Ranking wrt usage, bug-finding capabilities

- Expressing specifications
  Choosing a generic, domain-specific vocabulary

- Continuous specification
  Abstract feedback while you program
Challenges

- Finding relevant specifications
  Ranking wrt usage, bug-finding capabilities

- Expressing specifications
  Choosing a generic, domain-specific vocabulary

- Continuous specification
  Abstract feedback while you program

- Verified specifications
  Integration with symbolic verification
Compositional Verification

Eiffel Program → C Program → Server
Compositional Verification

enriched spec

C Program

Server
Compositional Verification

enriched spec

enriched spec

Server
Compositional Verification
Compositional Verification

enriched spec

enriched spec

enriched spec
Compositional Verification

Eiffel Program

enriched spec

enriched spec
Compositional Verification

Eiffel Program

⚠️ enriched spec

⚠️ enriched spec
Compositional Verification

Diagram showing the process of enriching specifications (enriched spec) through a C Program, resulting in another enriched specification.
Compositional Verification
Compositional Verification

enriched spec → enriched spec → Server
Compositional Verification

enriched spec

enriched spec

Server

✔ ✔
enriched spec
Experimental analysis

- Generate test cases to systematically explore behavior
- Assess executions to learn about software behavior

Complete behavior

- Execute and extract initial spec
- Generate test mutants and enrich specs

Real behavior

- Coverage progress: 83.64%
- Server

Reliable software

- See formal demo at ICSE 2012

Compositional Verification

- Enriched spec
- Server