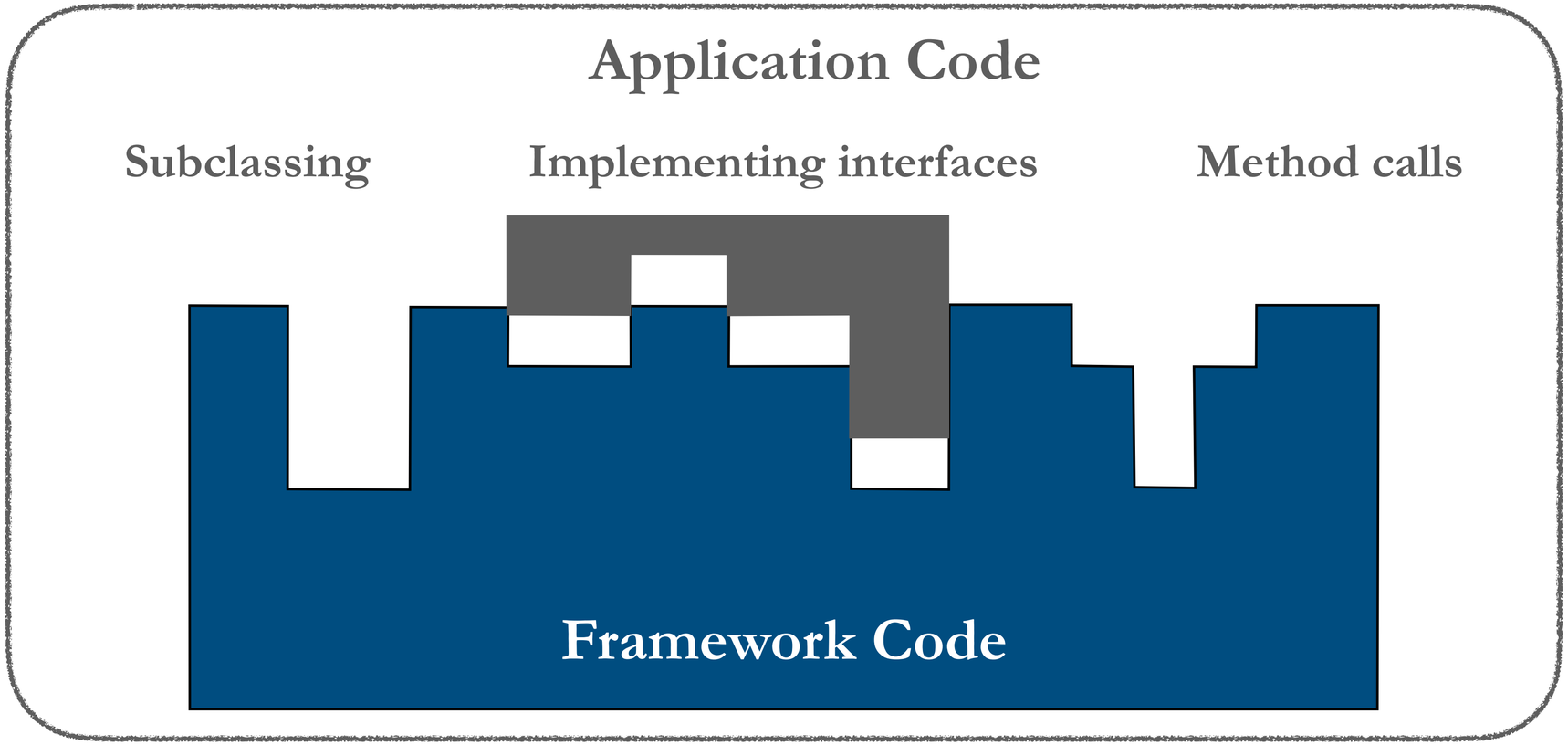
# BENEVOL 2020

## Mining and Recommending Instantiation Patterns for Java Framework Applications

Yunior Pacheco
Jonas De Bleser
Coen De Roover


# Framework Instantiation



# Framework Instantiation

**createMenuManager**

```
protected MenuManager createMenuManager()
```

Returns a new menu manager for the window.

Subclasses may override this method to customize the menu manager.

**Returns:**

a menu manager

# Framework Instantiation

```
Code Writer
import java.io.Console;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Properties;

public class GetSnap {

    public static void main(String[] args) {

    // Arguments: dasName piServerName [trustedConnection useDCA userName password protocol
    // Examples:
    // Trusted connection:              java GetSnap myDas myPi yes Https/Soap:5461 2 sin%
    // DCA or login dialog if DCA is not found: java GetSnap myDas myPi no yes Https/Soap:5
    // User name and Password (not recommended):    java GetSnap myDas myPi no no myUserNam

    int argsLength = args.length;
    int protocolOrderArgPosition = 4;
    Console console = System.console();
    String dasName = "";
    String dataSourceName = "";
    String isTrustedConnection = "";
    String useDCA = "";
    String userName = "";
    String password = "";
    String protocolOrder = "";
    String logLevel = "";
    String tagNamePattern = "";

    /*---------------------- Get supplied arguments ----------------------*/
    if (argsLength >= 1)
        dasName = args[0];
    if (argsLength >= 2)
        dataSourceName = args[1];
    if (argsLength >= 3)
        isTrustedConnection = args[2];
    if (isStringFalseOrNo(isTrustedConnectior
        if (argsLength >= 4) {
        useDCA = args[3];
        protocolOrderArgPosition = 5;
        if (isStringFalseOrNo(useDCA)) {
            protocolOrderArgPosition = 7;
            if (argsLength >= 5)
            userName = args[4];
            if (argsLength >= 6)
            password = args[5];
        }
        }
    }
Ln 1    Col 1
```

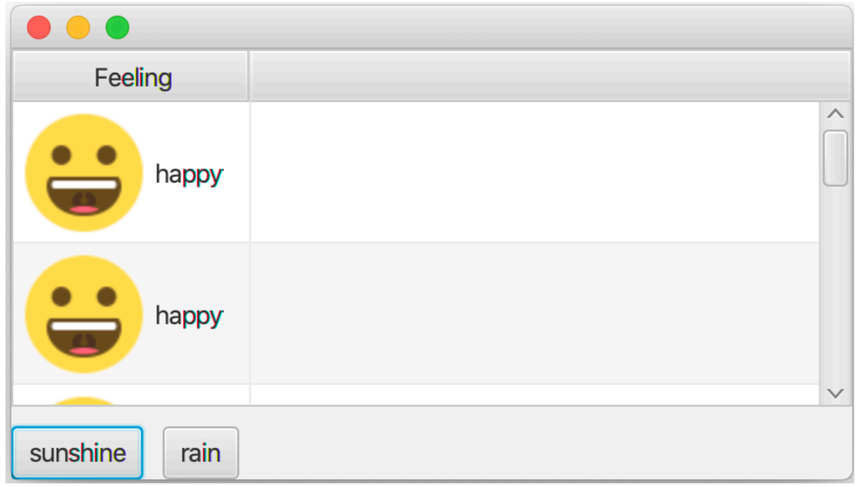
```
1   long int gcd(long int a,long int b)
2   {
3       if (a==0) return b;
4       return gcd(b%a,a);
5   }
6   int main() {
7     int t;
8     cin>>t;
9     while(t--)
10    {
                                    ,g,l;
                                    );
                                    <<l<<endl;
```

```
package rentalStore;
import java.util.Enumeration;
import java.util.Vector;

class Customer {
    private String _name;
    private Vector<Rental> _rentals = new Vector<Rental>();

    public Customer(String name) {
        _name = name;
    }
    public String getMovie(Movie movie) {
        Rental rental = new Rental(new Movie("", Movie.NEW_RELEASE), 10);
        Movie m = rental._movie;
        return movie.getTitle();
    }
    public void addRental(Rental arg) {
        _rentals.addElement(arg);
    }
    public String getName() {
        return _name;
    }
```

Existing applications

# Motivating example: Custom Table

```
TableView<Item> table = new TableView<>(FXCollections.observableArrayList(
        new Item(Feeling.HAPPY),    new Item(Feeling.HAPPY)
          ));


TableColumn<Item, Feeling> feelingColumn = new TableColumn<>("Feeling");

feelingColumn.setCellFactory(new Callback<TableColumn<Item, Feeling>, TableCell<Item, Feeling>>() {

    @Override
    public TableCell<Item, Feeling> call(TableColumn<Item, Feeling> param) {
        return new EmojiCell<>();
    }
});
          table.getColumns().add(feelingColumn);
```




```
public class EmojiCell<T> extends TableCell<T, Feeling> {

    private final ImageView image;

    public EmojiCell() {..

    @Override
    protected void updateItem(Feeling item, boolean empty) {
        super.updateItem(item, empty);

        if (empty || item == null) {
            // set back to look of empty cell
            setText(null);
            image.setImage(null);
        } else {
            // set image and text for non-empty cell
            image.setImage(item.getEmoji());
            setText(item.getValue());
        }
    }
}
```

# Motivating example: Custom Table

```
TableView<Item> table = new TableView<>(FXCollections.observableArrayList(
        new Item(Feeling.HAPPY),    new Item(Feeling.HAPPY)
        ));

TableColumn<Item, Feeling> feelingColumn = new TableColumn<>("Feeling");

feelingColumn.setCellFactory(new Callback<TableColumn<Item, Feeling>, TableCell<Item, Feeling>>() {

    @Override
    public TableCell<Item, Feeling> call(TableColumn<Item, Feeling> param) {
        return new EmojiCell<>();
    }
});
        table.getColumns().add(feelingColumn);
```

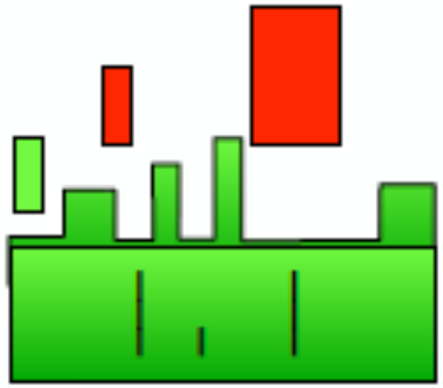
Close Interplay

```
public class EmojiCell<T> extends TableCell<T, Feeling> {

    private final ImageView image;

    public EmojiCell() {..

    @Override
    protected void updateItem(Feeling item, boolean empty) {
        super.updateItem(item, empty);

        if (empty || item == null) {
            // set back to look of empty cell
            setText(null);
            image.setImage(null);
        } else {
            // set image and text for non-empty cell
            image.setImage(item.getEmoji());
            setText(item.getValue());
        }
    }
}
```

Instantiate the class: **TableView**
Instantiate the class: **TableColumn**
Implement the Interface: **Callback**
Overwrite the method: **call()**
Call the method: **add()**
Subclass the class: **TableCell**
Overwrite the method: **updateItem()**

# Automated Extraction of Instantiation Actions

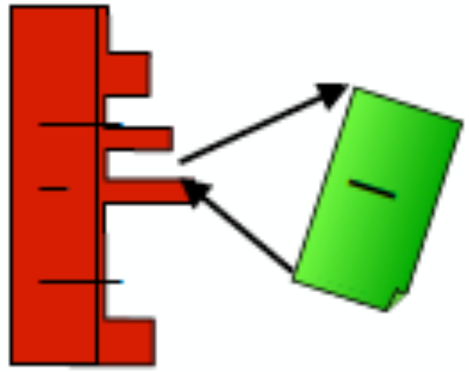
Framework Instantiation Mining



using a framework

**Necessary or common to extend or customise its functionality.**

Library Usage Mining



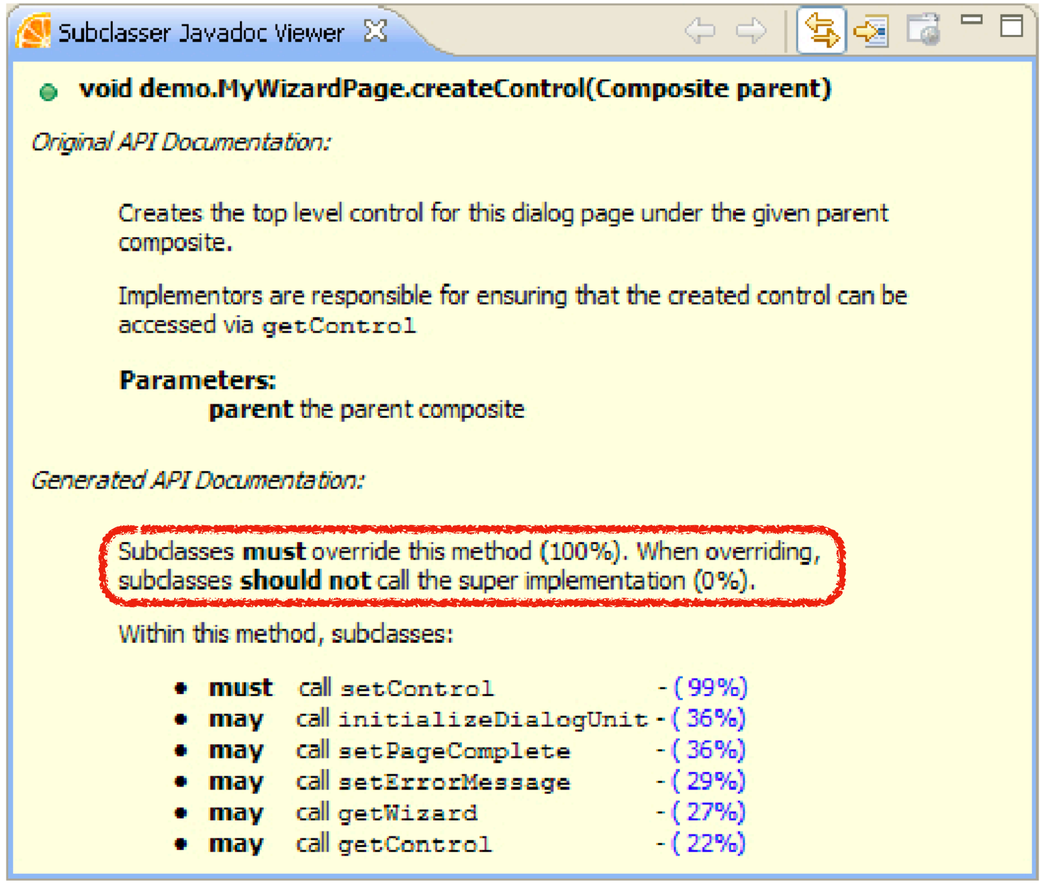
using a library

**Only the functionality it provides can be used.**

# Automated Extraction of Instantiation Actions

## Framework Instantiation Mining

Which framework classes are designed to be subclassed?
Which methods therein are designed to be overridden in an application-specific manner?




[1] R. Johnson. Documenting frameworks using patterns, 1992.
[2] M. Bruch, M. Mezini, and M. Monperrus, “Mining subclassing directives to improve framework reuse”, 2010.

# Automated Extraction of Instantiation Actions

## Framework Instantiation Mining

```
TableView<Item> table = new TableView<>(FXCollections.observableArrayList(
        new Item(Feeling.HAPPY),    new Item(Feeling.HAPPY)
          ));


TableColumn<Item, Feeling> feelingColumn = new TableColumn<>("Feeling");

feelingColumn.setCellFactory(new Callback<TableColumn<Item, Feeling>, TableCell<Item, Feeling>>() {

    @Override
    public TableCell<Item, Feeling> call(TableColumn<Item, Feeling> param) {
        return new EmojiCell<>();
    }
});
          table.getColumns().add(feelingColumn);
```

Implement the Interface **Callback**
Overwrite the **call()** method

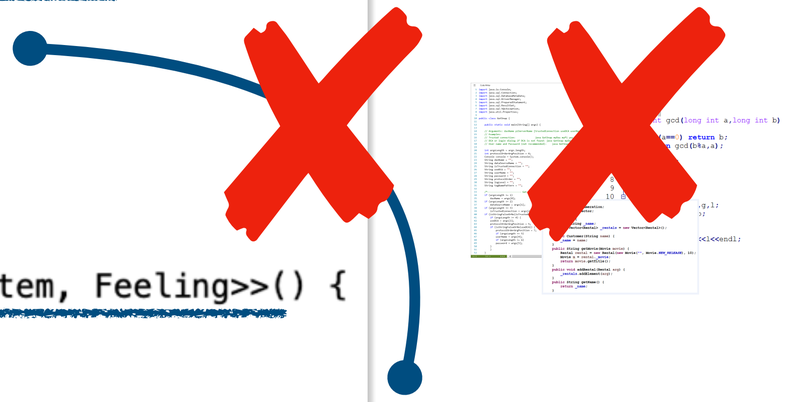
Subclass the class **TableCell**
Overwrite the **updateItem()** method

```
public class EmojiCell<T> extends TableCell<T, Feeling> {

    private final ImageView image;

    public EmojiCell() {..

    @Override
    protected void updateItem(Feeling item, boolean empty) {
        super.updateItem(item, empty);

        if (empty || item == null) {
            // set back to look of empty cell
            setText(null);
            image.setImage(null);
        } else {
            // set image and text for non-empty cell
            image.setImage(item.getEmoji());
            setText(item.getValue());
        }
    }
}
```

# Automated Extraction of Instantiation Actions

## Framework Instantiation Mining

```
TableView<Item> table = new TableView<>(FXCollections.observableArrayList(
        new Item(Feeling.HAPPY),    new Item(Feeling.HAPPY)
          ));


TableColumn<Item, Feeling> feelingColumn = new TableColumn<>("Feeling");

feelingColumn.setCellFactory(new Callback<TableColumn<Item, Feeling>, TableCell<Item, Feeling>>() {

    @Override
    public TableCell<Item, Feeling> call(TableColumn<Item, Feeling> param) {
        return new EmojiCell<>();
    }
});
          table.getColumns().add(feelingColumn);
```



Implement the Interface **Callback**
Overwrite the **call()** method

Subclass the class **TableCell**
Overwrite the **updateItem()** method

```
public class EmojiCell<T> extends TableCell<T, Feeling> {

    private final ImageView image;

    public EmojiCell() {..

    @Override
    protected void updateItem(Feeling item, boolean empty) {
        super.updateItem(item, empty);

        if (empty || item == null) {
            // set back to look of empty cell
            setText(null);
            image.setImage(null);
        } else {
            // set image and text for non-empty cell
            image.setImage(item.getEmoji());
            setText(item.getValue());
        }
    }
}
```

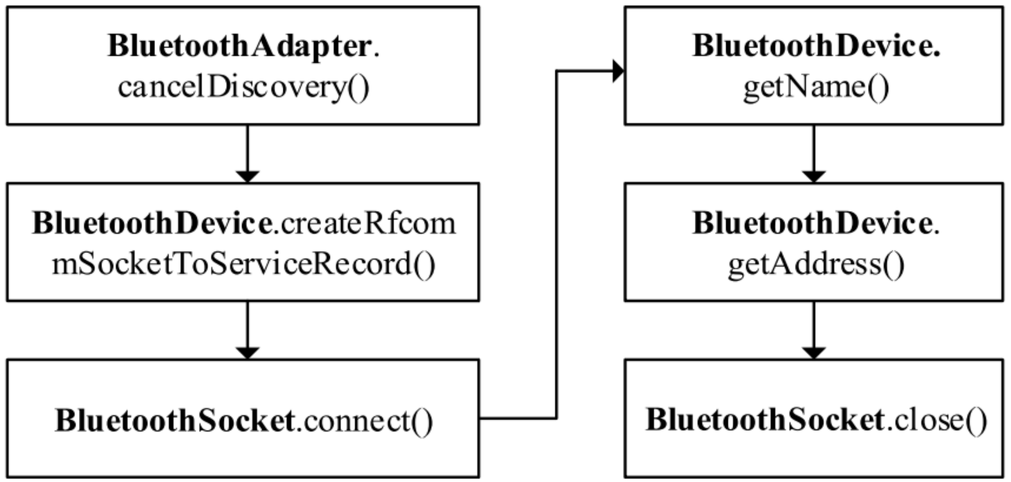
# Automated Extraction of Instantiation Actions

Library Usage Mining

Which API methods should this piece of client code invoke, considering that it has already invoked these other API methods?

## BluetoothSocket:connect()

**Usage Pattern $P_a$**




**Method GetInformation()**

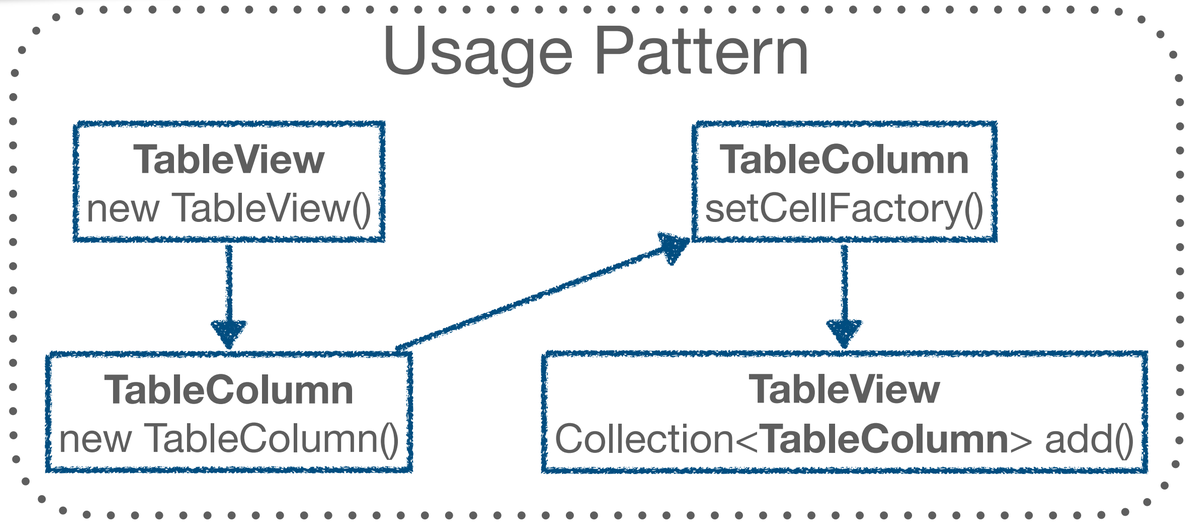
```
public void GetInformation(BluetoothDevice mmDevice) {
    ...
mAdapter.cancelDiscovery();
BluetoothSocket mmSocket = mmDevice.createRfcommSocketToServiceRecord(...);
// Make a connection to the BluetoothSocket
try {
mmSocket.connect();
} catch (Exception e) {
    Log.e(TAG, "Connection to " + mmDevice.getName() + " at "
     + mmDevice.getAddress() + " failed:" + e.getMessage());
// Close the socket
mmSocket.close(); }
}
```

[3] H. Niu, I. Keivanloo, and Y. Zou, “API usage pattern recommendation for software development”, 2017.

# Automated Extraction of Instantiation Actions

## Library Usage Mining

```
TableView<Item> table = new TableView<>(FXCollections.observableArrayList(
        new Item(Feeling.HAPPY),    new Item(Feeling.HAPPY)
          ));


TableColumn<Item, Feeling> feelingColumn = new TableColumn<>("Feeling");

feelingColumn.setCellFactory(new Callback<TableColumn<Item, Feeling>, TableCell<Item, Feeling>>() {

    @Override
    public TableCell<Item, Feeling> call(TableColumn<Item, Feeling> param) {
        return new EmojiCell<>();
    }
});
          table.getColumns().add(feelingColumn);
```

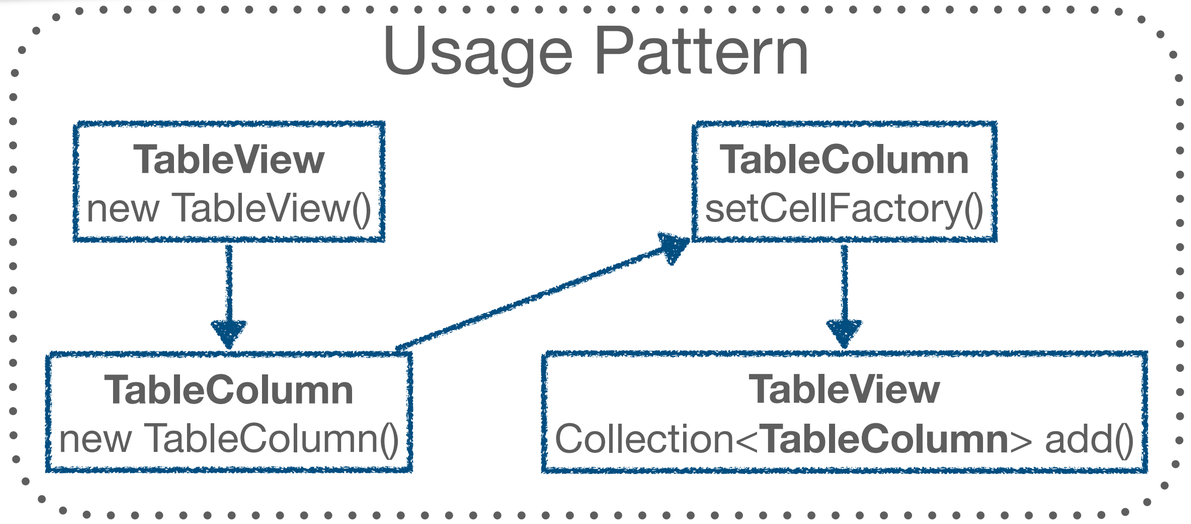



```
public class EmojiCell<T> extends TableCell<T, Feeling> {

    private final ImageView image;

    public EmojiCell() {..

    @Override
    protected void updateItem(Feeling item, boolean empty) {
        super.updateItem(item, empty);

        if (empty || item == null) {
            // set back to look of empty cell
            setText(null);
            image.setImage(null);
        } else {
            // set image and text for non-empty cell
            image.setImage(item.getEmoji());
            setText(item.getValue());
        }
    }
}
```

# Automated Extraction of Instantiation Actions

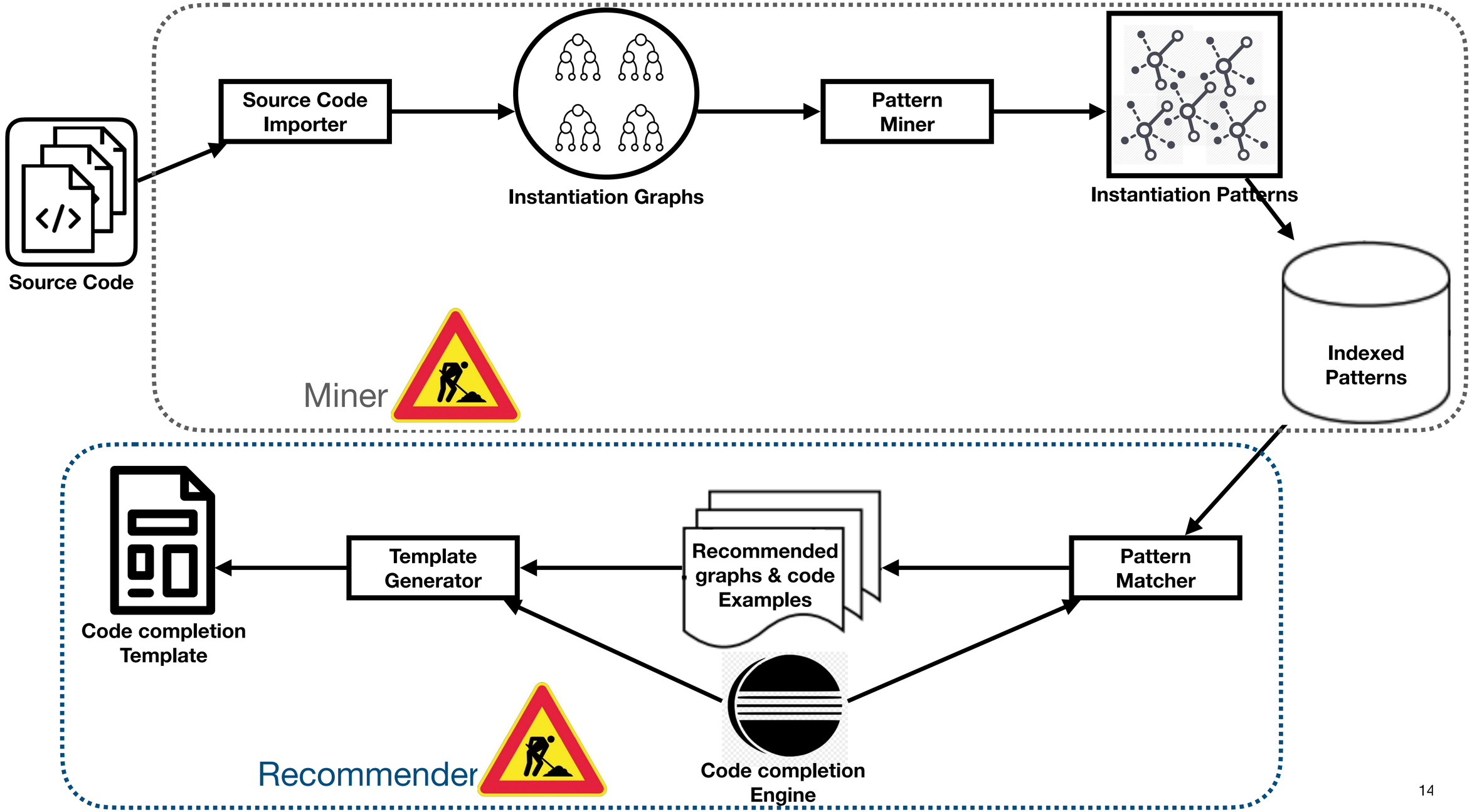
## Library Usage Mining

```
TableView<Item> table = new TableView<>(FXCollections.observableArrayList(
        new Item(Feeling.HAPPY),    new Item(Feeling.HAPPY)
        ));

TableColumn<Item, Feeling> feelingColumn = new TableColumn<>("Feeling");

feelingColumn.setCellFactory(new Callback<TableColumn<Item, Feeling>, TableCell<Item, Feeling>>() {
    @Override
    public TableCell<Item, Feeling> call(TableColumn<Item, Feeling> param) {
        return new EmojiCell<>();
    }
});
        table.getColumns().add(feelingColumn);
```




```
public class EmojiCell<T> extends TableCell<T, Feeling>

    private final ImageView image;

    public EmojiCell() {

    @Override
    protected void updateItem(Feeling item, boolean empty) {
        super.updateItem(item, empty);

        if (empty || item == null) {
            // set back to look of empty cell
            setText(null);
            image.setImage(null);
        } else {
            // set image and text for non-empty cell
            image.setImage(item.getEmoji());
            setText(item.getValue());
        }
    }
}
```

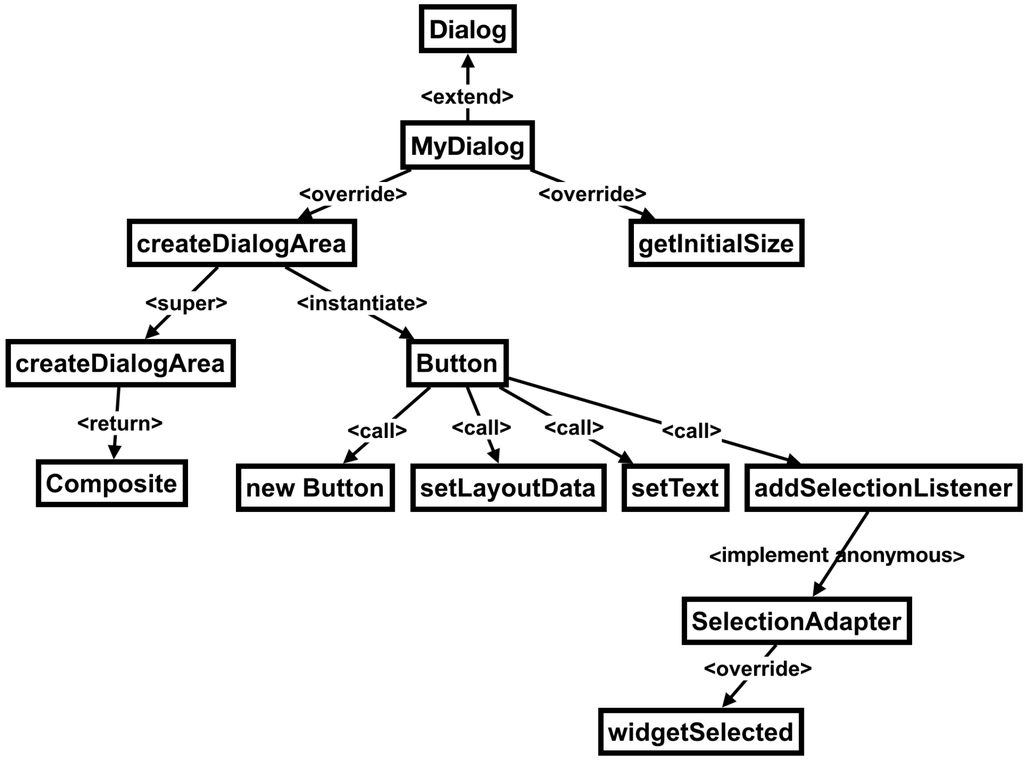
# Proposed approach



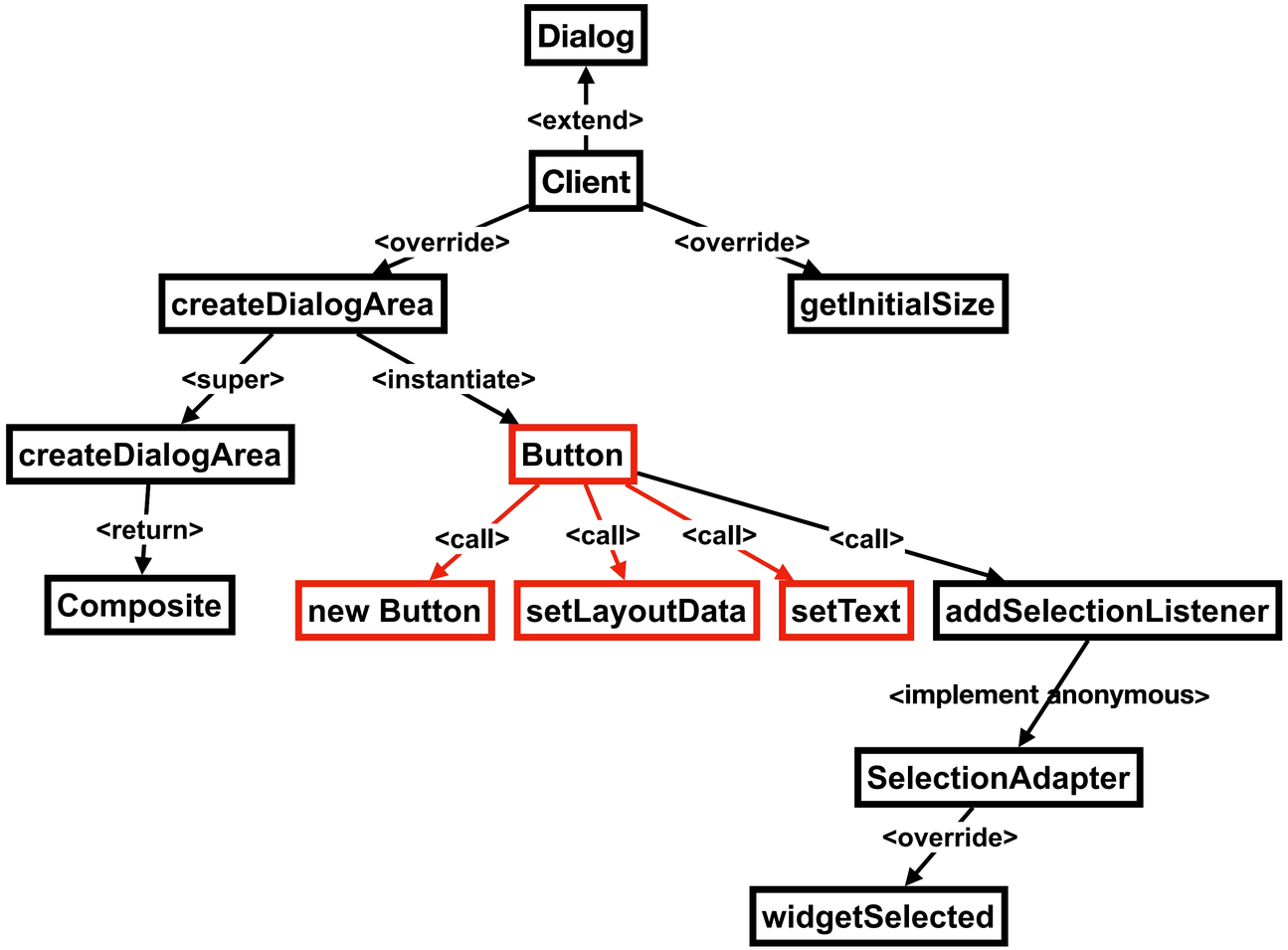
# Miner: Instantiation Graphs

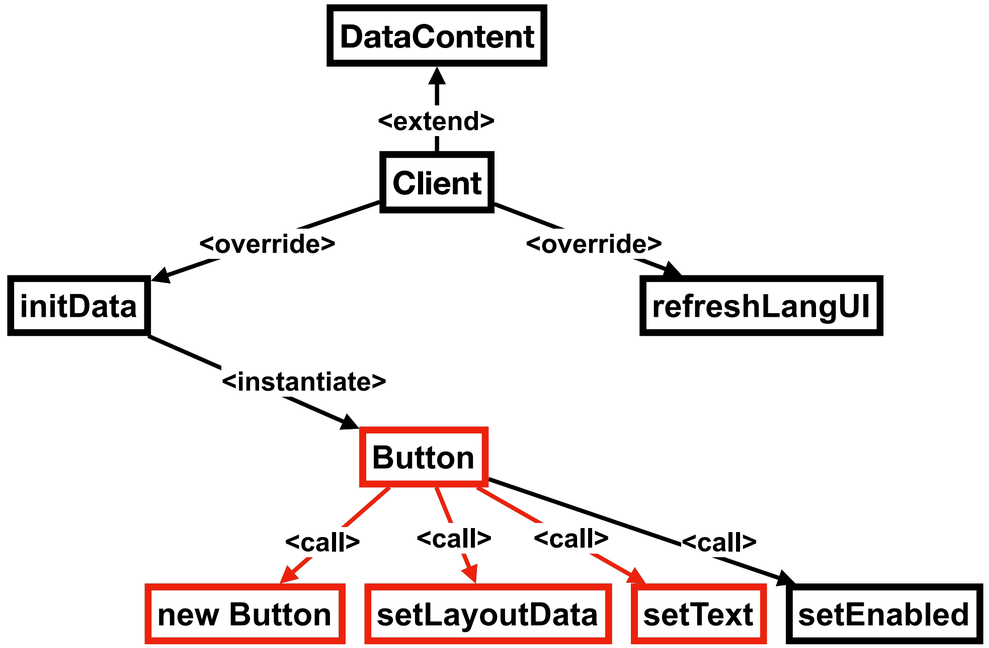
```
public class MyDialog extends Dialog {
    public MyDialog(Shell parentShell) {
        super(parentShell);
    }

    protected Control createDialogArea(Composite parent) {
        Composite container = (Composite) super.createDialogArea(parent);
        Button button = new Button(container, SWT.PUSH);
        button.setLayoutData(new GridData(SWT.BEGINNING, SWT.CENTER, false, false));
        button.setText("Press me");
        button.addSelectionListener(new SelectionAdapter() {
            @Override
            public void widgetSelected(SelectionEvent e) {
                System.out.println("Pressed");
            }
        });
        return container;
    }

    newShell.setText("Selection dialog");
}

    protected Point getInitialSize() {
        return new Point(450, 300);
    }
}
```


## Instantiation Graph

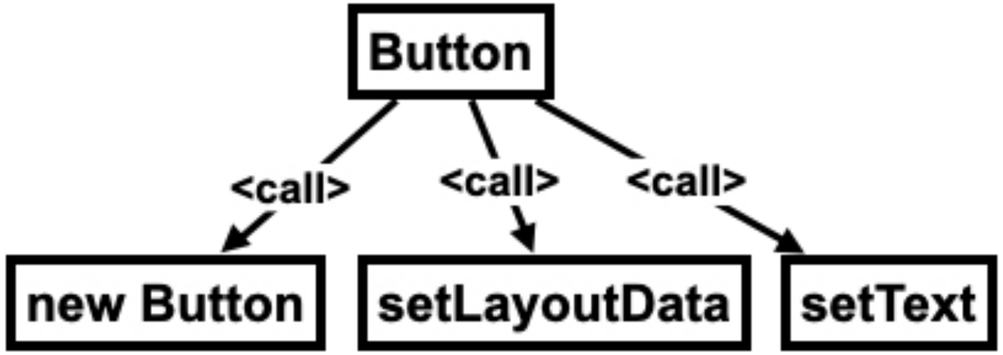


# Miner: Instantiation pattern



# Miner: Instantiation pattern



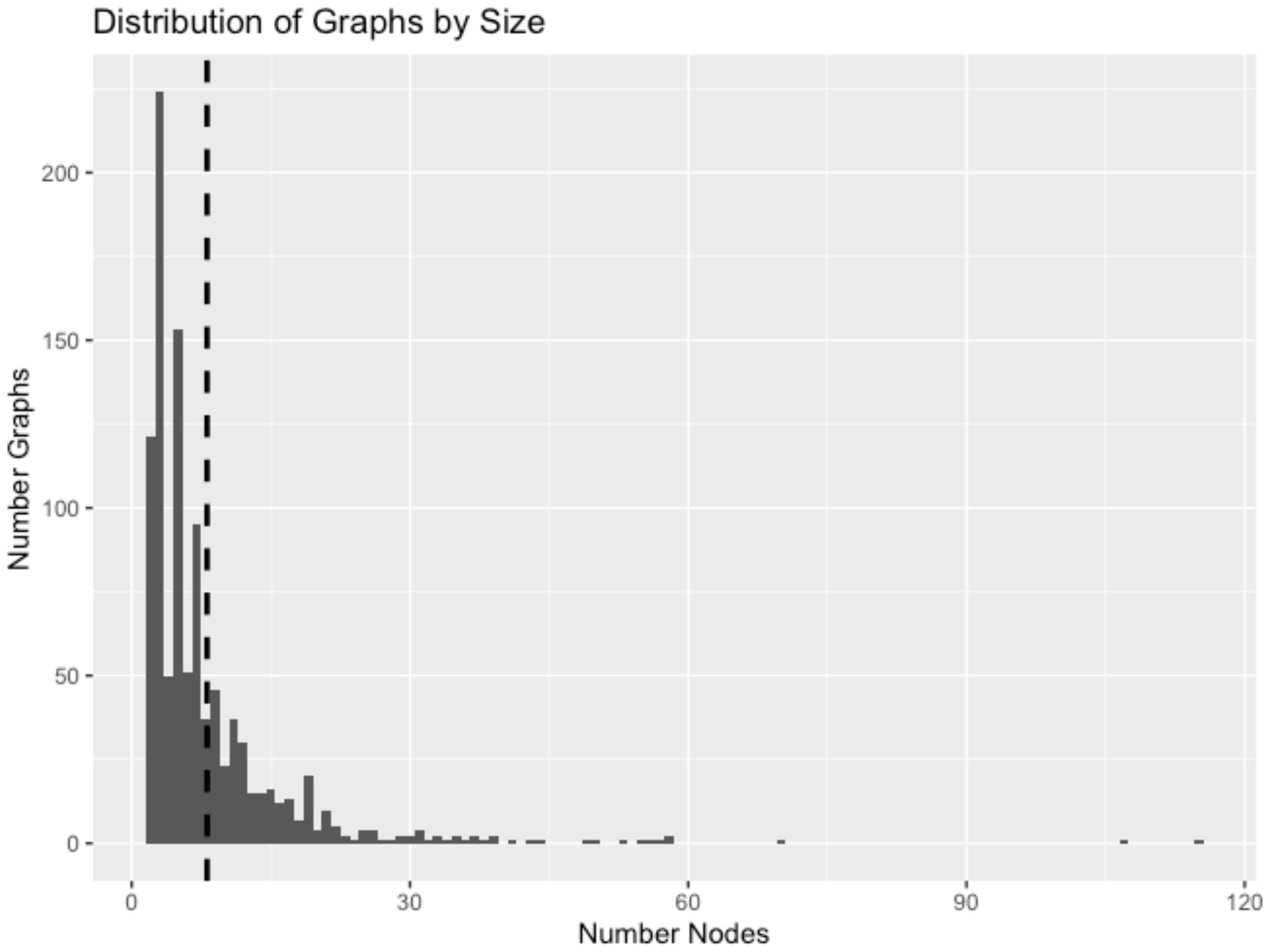

Frequent Subgraph

**TKG** algorithm for mining the **top-k frequent subgraphs**

SPMF
An Open-Source Data Mining Library

**Preliminary** experiment on a corpus of **105** Java projects that use the **JavaFX** framework.

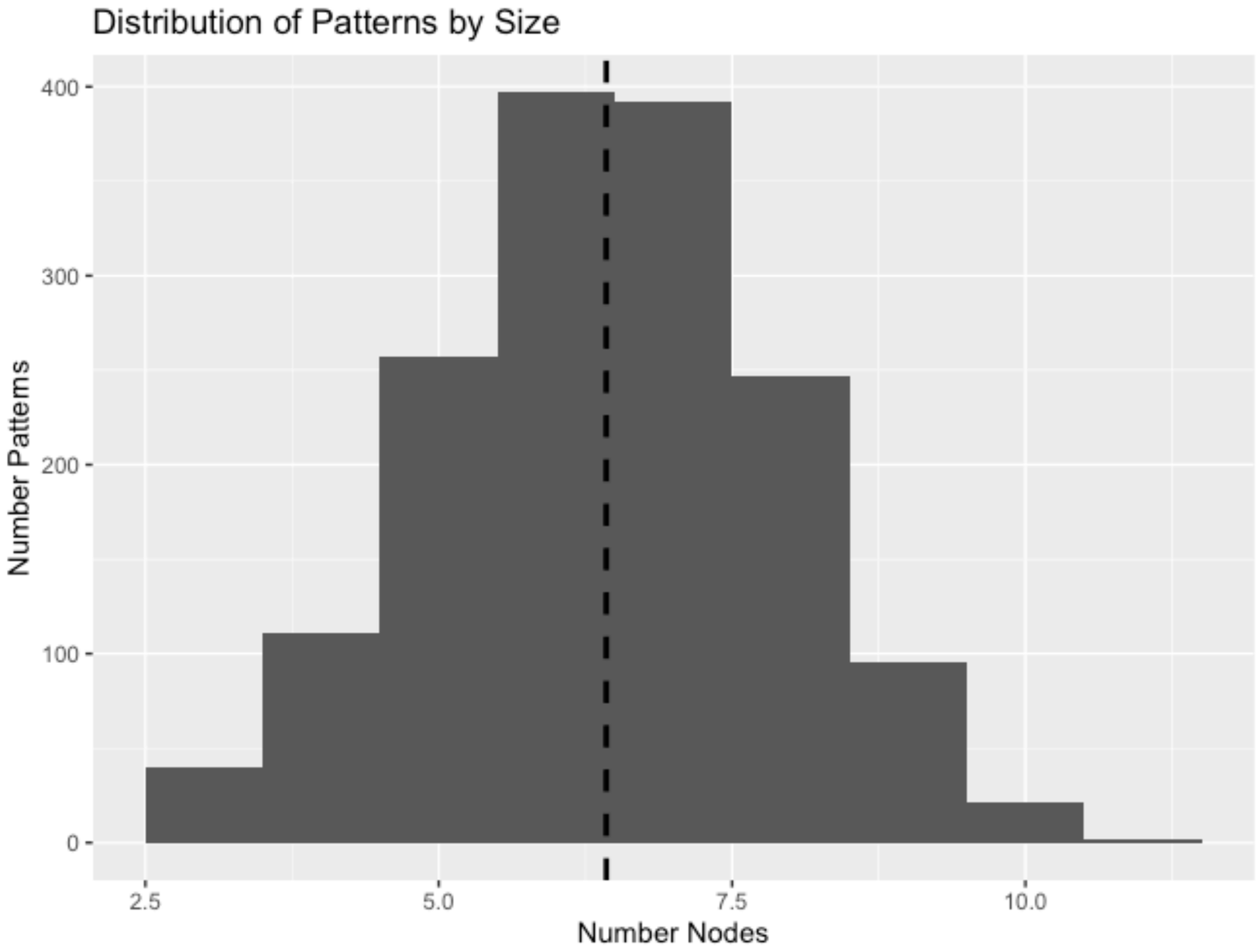
# Preliminary Results: Instantiation Graphs




Framework: **JavaFX**
Total projects: **105**
Projects with No graphs: **23**
Instantiation graphs: **1031**
Graphs containing related framework types: **704**

Ave Size: 8 nodes.
**Small Graphs** are supposed to produce **Small Patterns**.

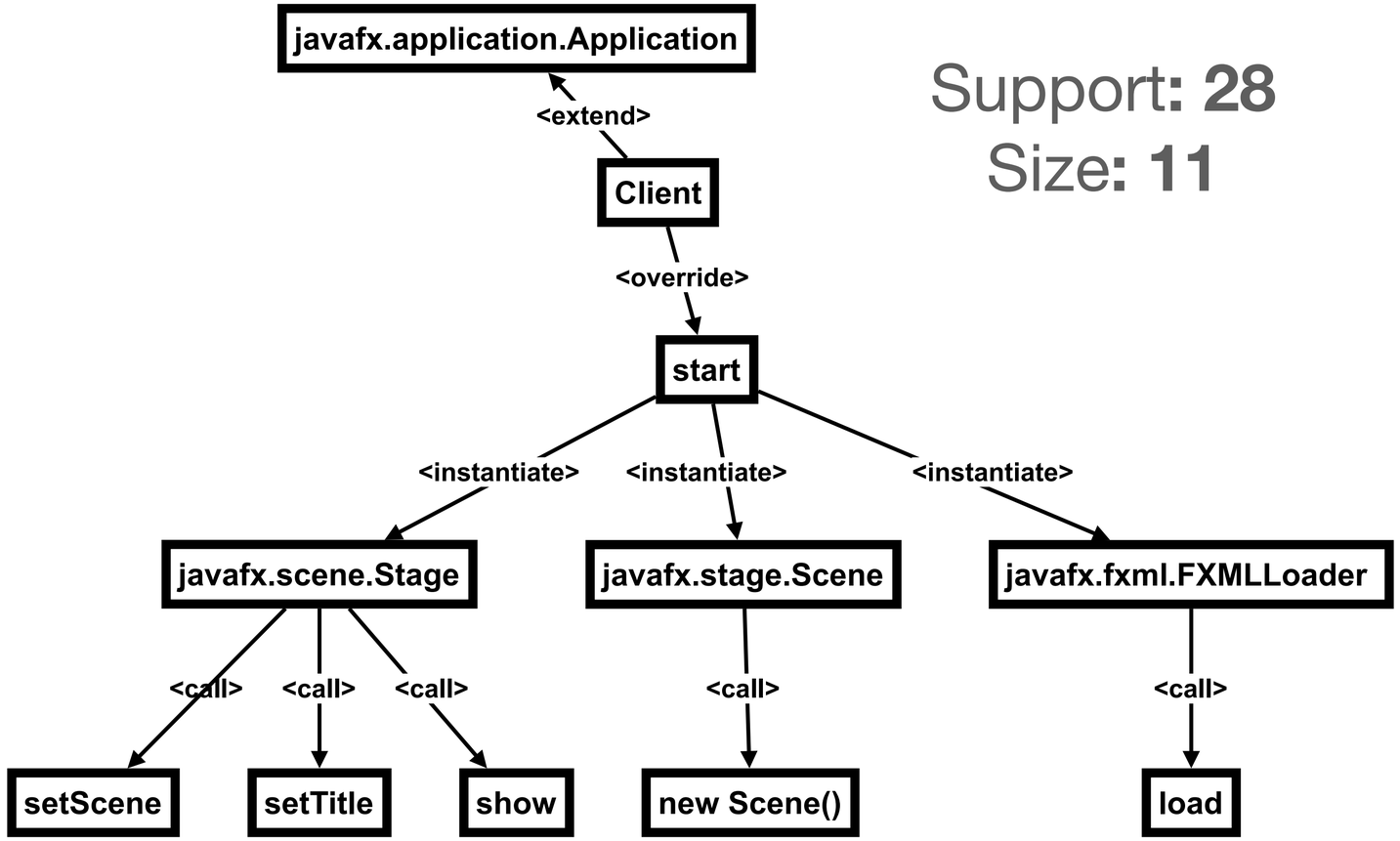
# Preliminary Results: Instantiation Patterns




Instantiation Patterns: **1563**
Min support: **20**

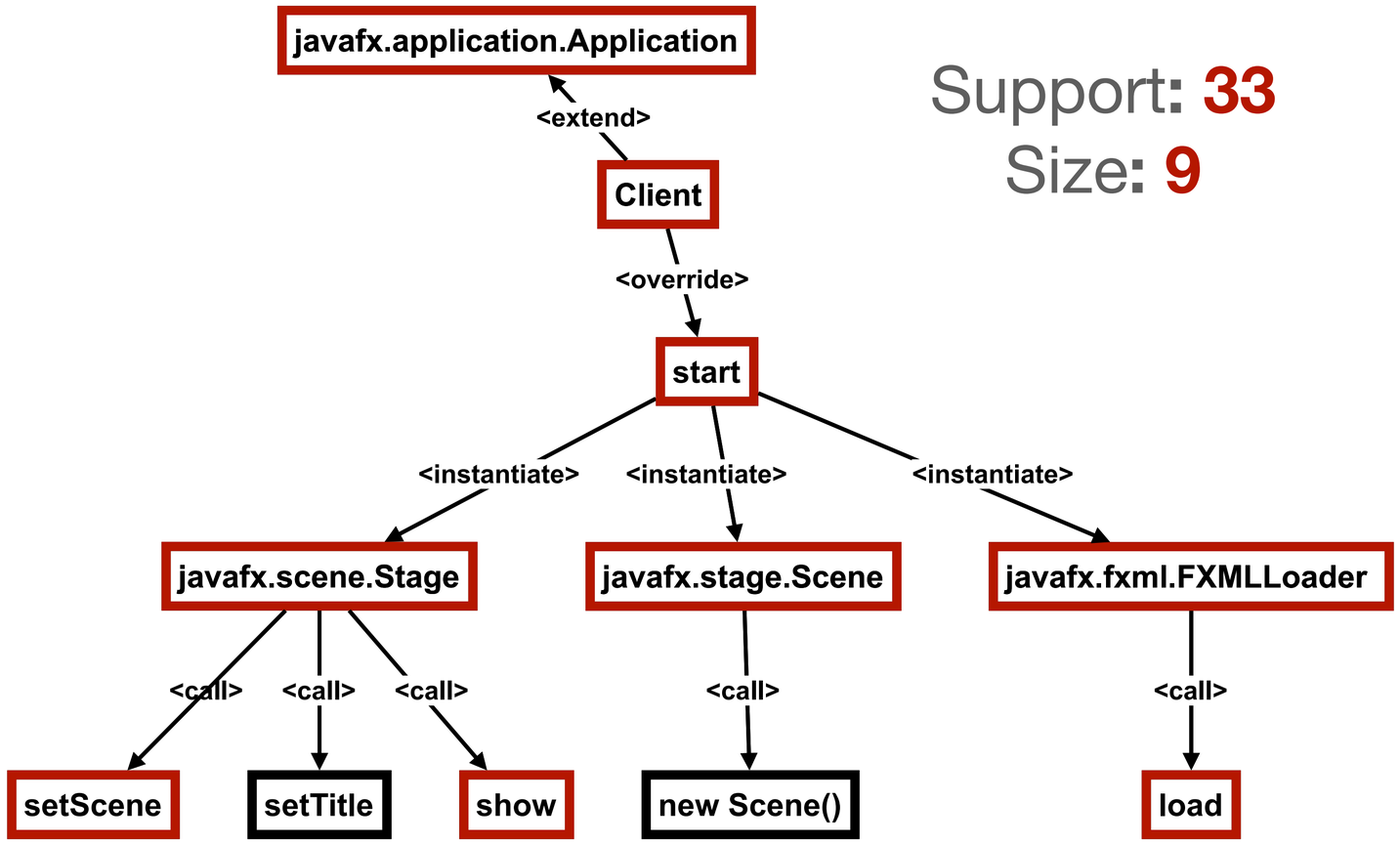
**Small Patterns** contain less information about possible interplays between framework elements.

# Preliminary Results: Instantiation Patterns



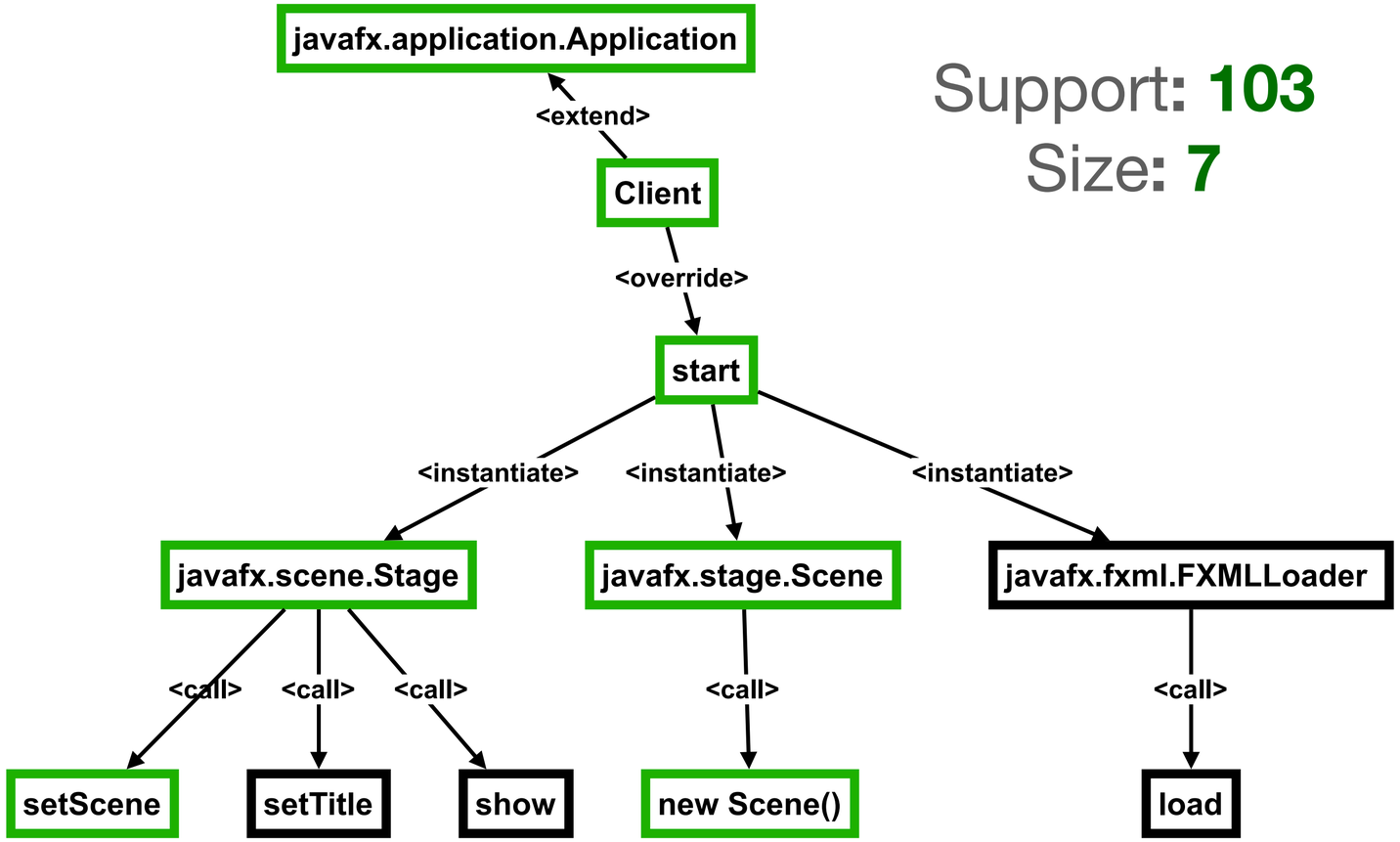

Support: **28**
Size: **11**

# Preliminary Results: Instantiation Patterns




Support: **33**
Size: **9**

# Preliminary Results: Instantiation Patterns




Support: **103**
Size: **7**

If there is a **large frequent pattern** (high support), all its subgraphs are also frequent.

**Large number of patterns related to a small number of Framework Types**

# Conclusion & Future Work

**More Questions than Answers!!!**

Is the employed mining algorithm suitable for our purpose of capturing relations between Instantiation Actions?

Is the current graph representation able to capture more information that could be relevant in the recommendation process?

- Repetition and iteration
- Generalisation of different sub-types in a a common super-type

Could we obtain better results if we mine on projects hat use a different framework than **JavaFx**?

# BENEVOL 2020

## Mining and Recommending Instantiation Patterns for Java Framework Applications


Yunior Pacheco ypacheco@vub.be
Jonas De Bleser jdeblese@vub.be
Coen De Roover cderoove@vub.be