

Visualization of Exception Handling Constructs to Support Program Understanding

Hina Shah

Carsten Görg and Mary Jean Harrold

Supported by NSF under CCR-0205422, CCF-0429117, CCF-0541049,
and CCF-0725202, and Tata Consultancy Services, Ltd.

Presenter supported by SIGSOFT CAPS
(patents pending)

Existing Techniques

Analyses

- simplify exception structure [Robillard and Murphy 2000]
- identify exception flow [Sinha et al. 2004]
- identify re-thrown exception chains [Fu and Ryder 2007]

Visualizations

- ExPo : throw-catch pairs as flow graph. [Sinha et al. 2004]
- EXTEST : exception propagation and navigation path as tree [Fu, Ryder 2005]
- ExceptionBrowser : exception propagation as tree [Chang et al. 2002]

- focus on improving analysis techniques
- provide no visual representation

- focus on low-level abstraction
- provide no system-wide view
- provide no context

Existing Techniques

<p>Ana</p> <ul style="list-style-type: none">• sim• stru• Mur• ide• [Sin• ide• exc• Ryd	<h2>Goal</h2> <p>Create Visualization that provides</p> <ul style="list-style-type: none">• different perspectives for viewing exception-handling information• at different levels of detail• along with context information <h2>Result</h2> <p>Visualization with three views</p> <ul style="list-style-type: none">• shows static-analysis information of exception-handling constructs• at different levels of detail• along with context	<p>irs as 004]</p> <p>ation 005]</p> <p>as</p>
<ul style="list-style-type: none">• focu• ana• prov• rep		<p>view</p>

Outline

- Exception handling in Java
- Survey
- Visualization
- Evaluation
- Conclusion

Outline

- Exception handling in Java
- Survey
- Visualization
- Evaluation
- Conclusion

Exception Handling in Java

Exception Handling constructs

- exception *types*
- *try* blocks
- *throw* statements
- *catch* blocks
- *finally* blocks

```
method(){
    MyException e = new MyException();
    try{
        ...
        if(flag==null){
            throw e;
        }else{
            // normal execution
        }
    }catch(MyException e){
        // recovery code
    }
    finally{
        // clean up code
    }
}
```

Exception Handling in Java

Exception Handling constructs

- exception *types*
- *try* blocks
- *throw* statements
- *catch* blocks
- *finally* blocks

```
method(){  
    MyException e = new MyException();  
    try{  
        ...  
        if(flag==null){  
            throw e;  
        }else{  
            // normal execution  
        }  
    }catch(MyException e){  
        // recovery code  
    }  
    finally{  
        // clean up code  
    }  
}
```

Exception Handling in Java

Exception Handling constructs

- exception *types*
- *try* blocks
- *throw* statements
- *catch* blocks
- *finally* blocks

```
method(){
    MyException e = new MyException();
    try{
        ...
        if(flag==null){
            throw e;
        }else{
            // normal execution
        }
    }catch(MyException e){
        // recovery code
    }
    finally{
        // clean up code
    }
}
```

Exception Handling in Java

Exception Handling constructs

- exception *types*
- *try* blocks
- *throw* statements
- *catch* blocks
- *finally* blocks

```
method(){  
    MyException e = new MyException();  
    try{  
        ...  
        if(flag==null){  
            throw e;  
        }else{  
            // normal execution  
        }  
    }catch(MyException e){  
        // recovery code  
    }  
    finally{  
        // clean up code  
    }  
}
```

Exception Handling in Java

Exception Handling constructs

- exception *types*
- *try* blocks
- *throw* statements
- *catch* blocks
- *finally* blocks

```
method(){  
    MyException e = new MyException();  
    try{  
        ...  
        if(flag==null){  
            throw e;  
        }else{  
            // normal execution  
        }  
    }catch(MyException e){  
        // recovery code  
    }  
    finally{  
        // clean up code  
    }  
}
```

Exception Handling in Java

Exception Handling constructs

- exception *types*
- *try* blocks
- *throw* statements
- *catch* blocks
- *finally* blocks

```
method(){
    MyException e = new MyException();
    try{
        ...
        if(flag==null){
            throw e;
        }else{
            // normal execution
        }
    }catch(MyException e){
        // recovery code
    }
    finally{
        // clean up code
    }
}
```

Survey

Goal

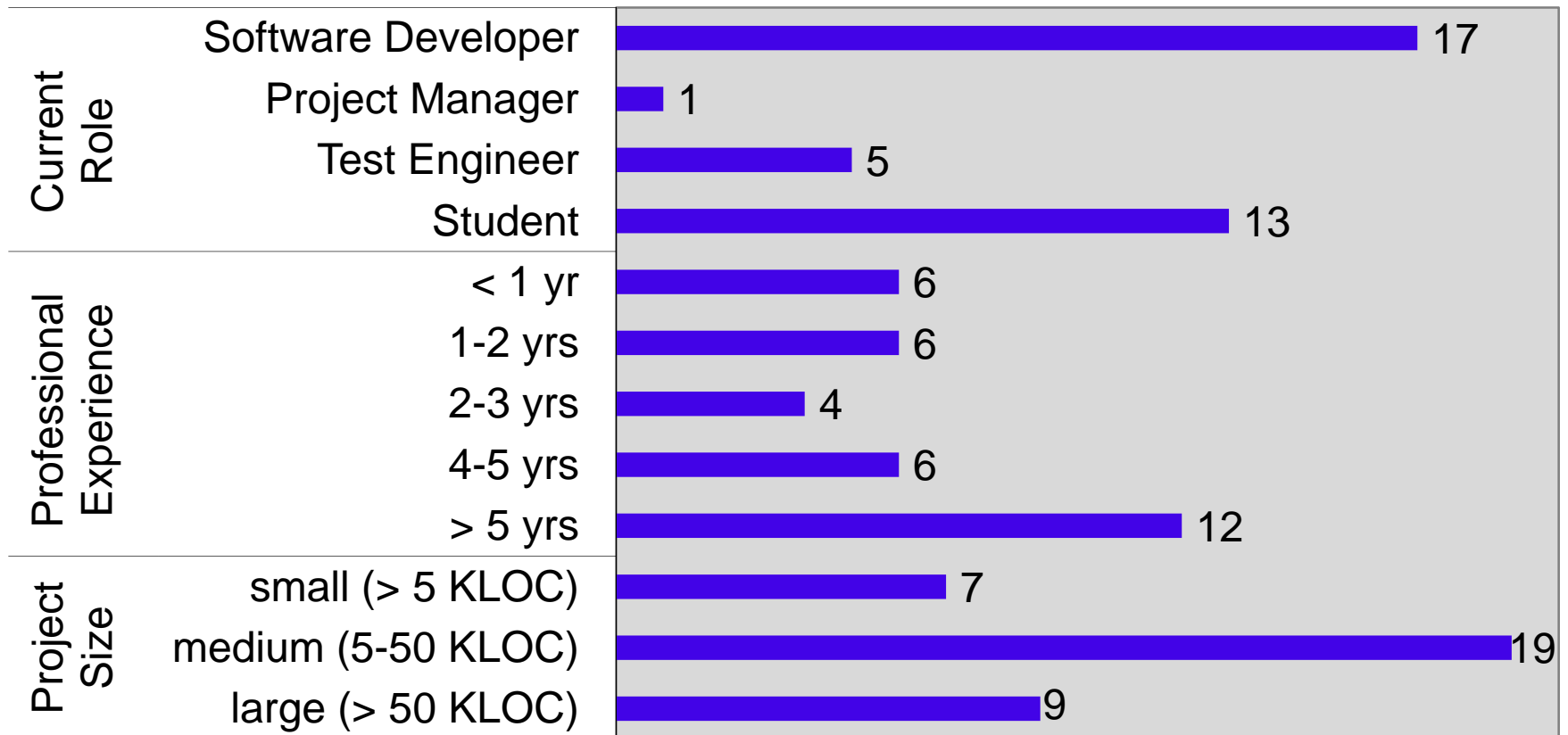
- understand the needs of developers
- inform the design of our visualization system

Process

- created initial questionnaire design
- conducted pilot runs
- revised questionnaire based on feedback
- conducted the actual survey
 - demographics (34 participants)
 - exception-related questions

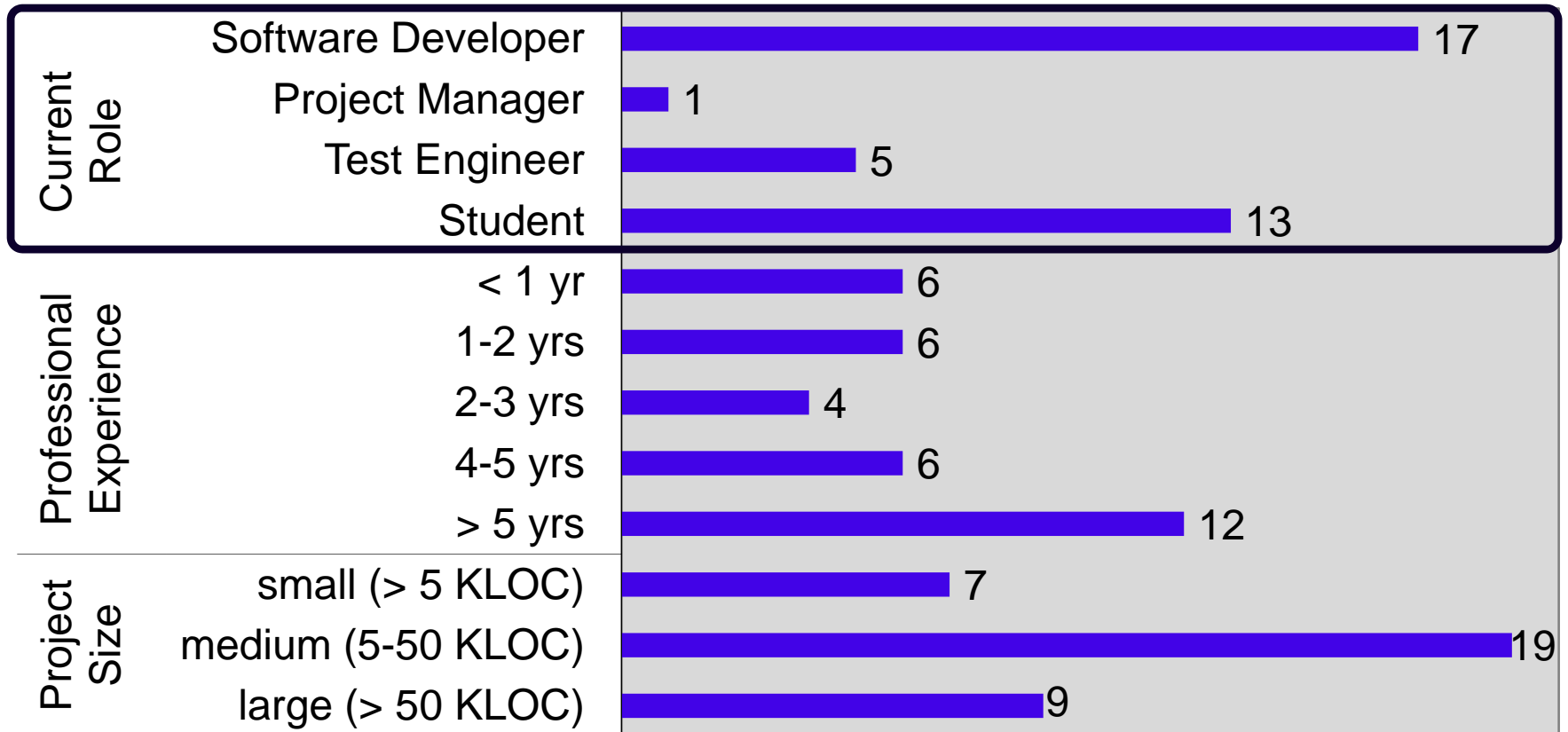
Demographics (34 Participants)

■ Number of Participants



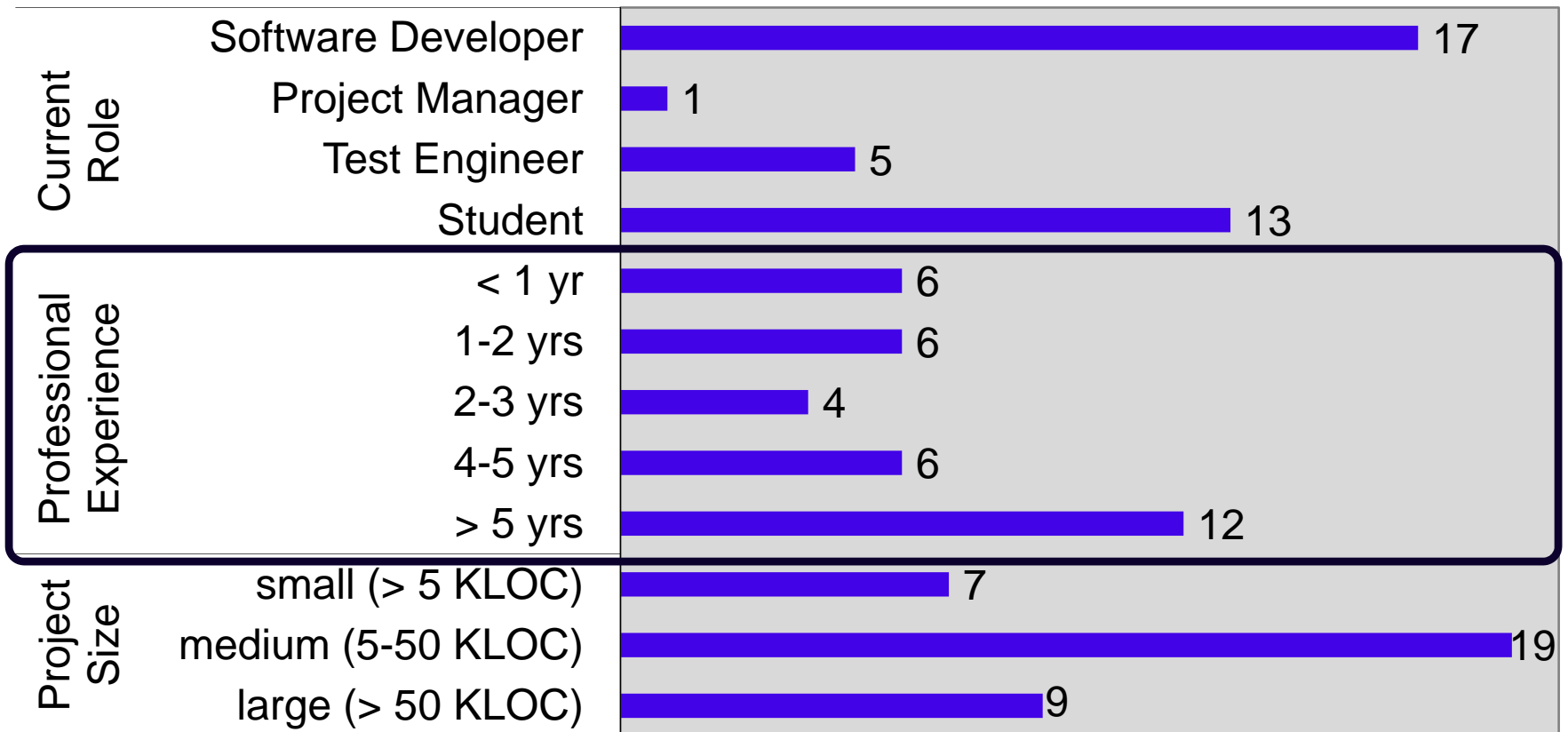
Demographics (34 Participants)

■ Number of Participants



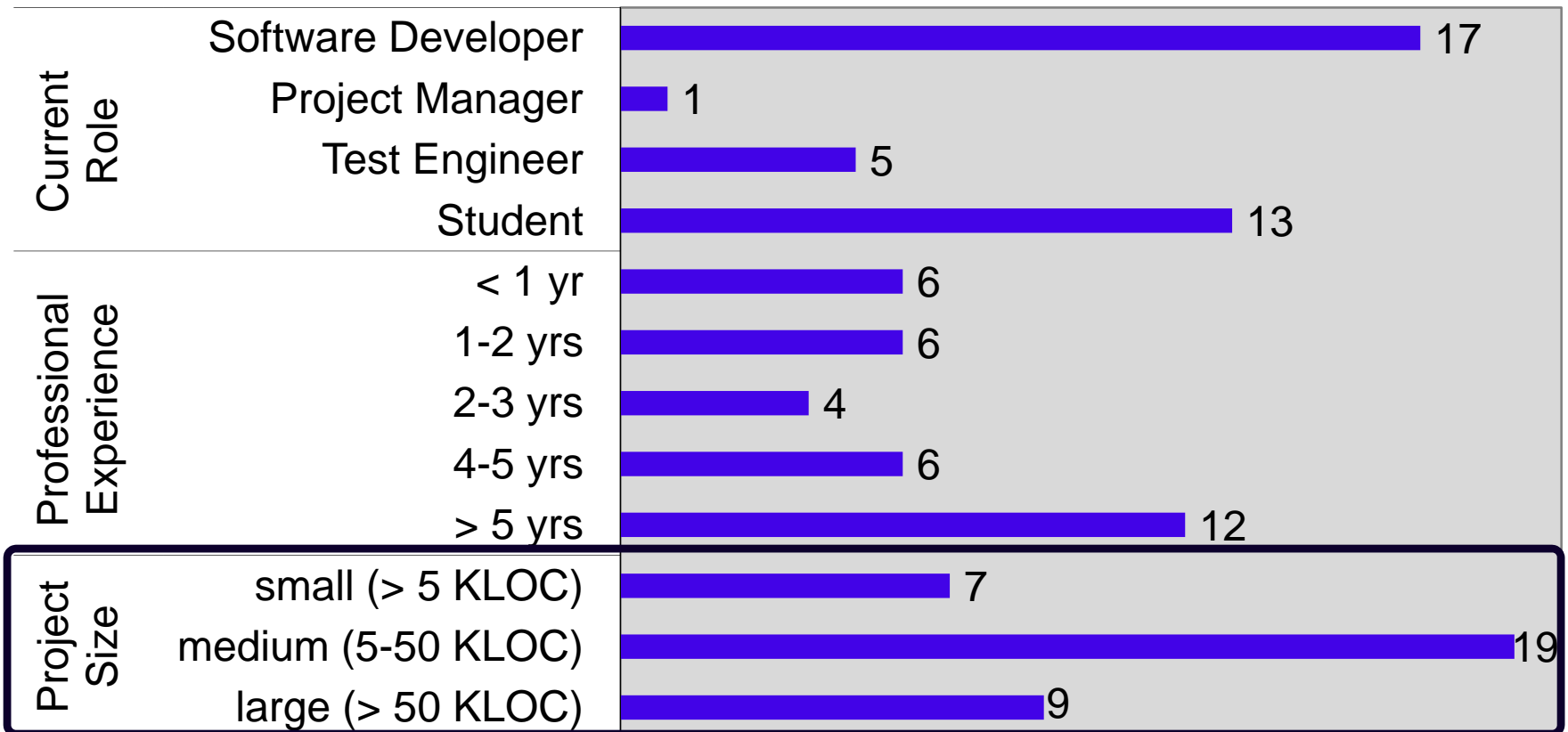
Demographics (34 Participants)

■ Number of Participants



Demographics (34 Participants)

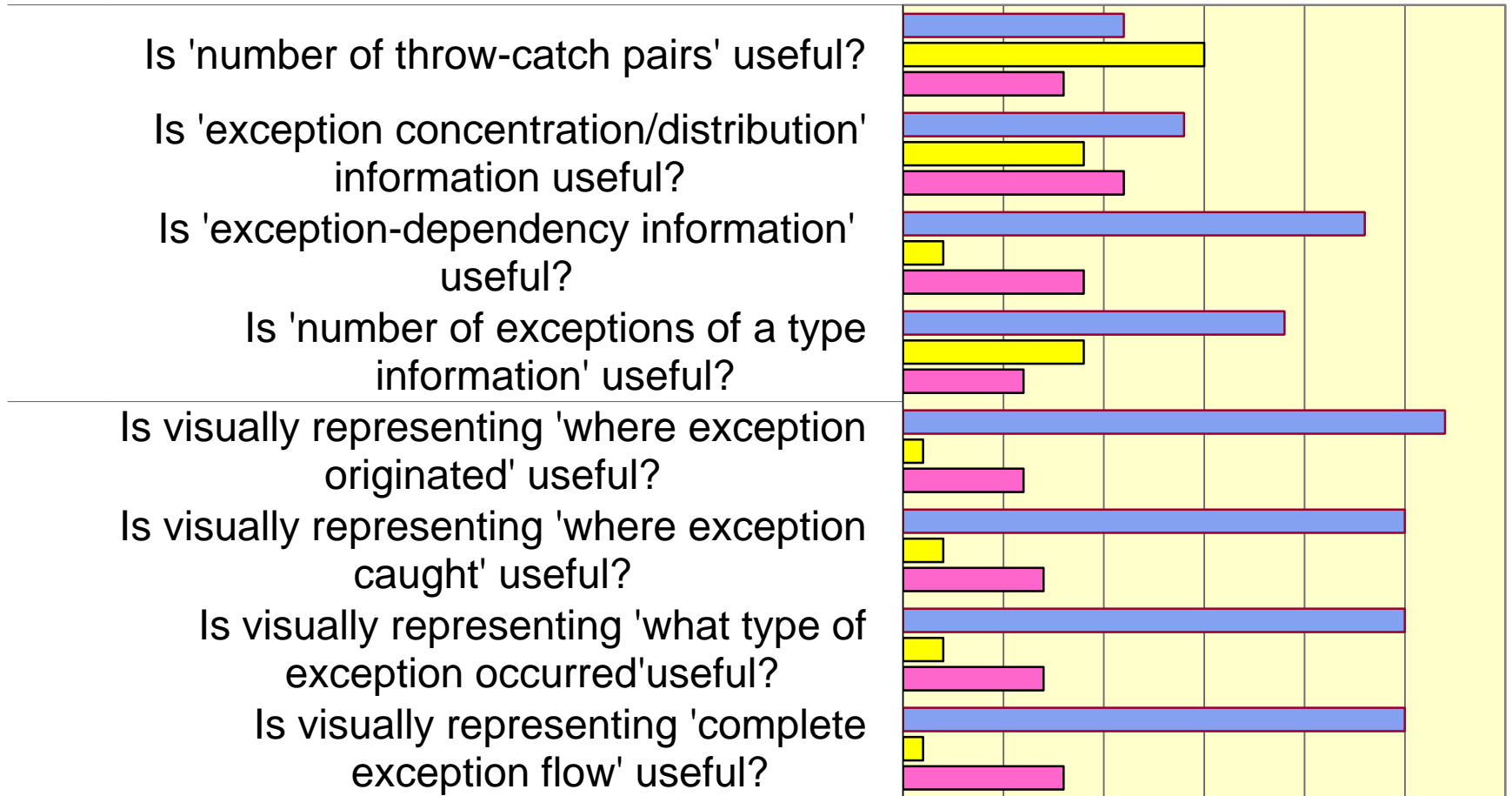
■ Number of Participants



Exception-Related Questions

Number of Participants

0 5 10 15 20 25 30



Exception-Related Questions

Number of Participants

0 5 10 15 20 25 30

Quantitative

Is 'number of throw-catch pairs' useful?

Is 'exception concentration/distribution information' useful?

Is 'exception-dependency information' useful?

Is 'number of exceptions of a type information' useful?

Is visually representing 'where exception originated' useful?

Is visually representing 'where exception caught' useful?

Is visually representing 'what type of exception occurred' useful?

Is visually representing 'complete exception flow' useful?

Yes

No

Maybe

Exception-Related Questions

Number of Participants

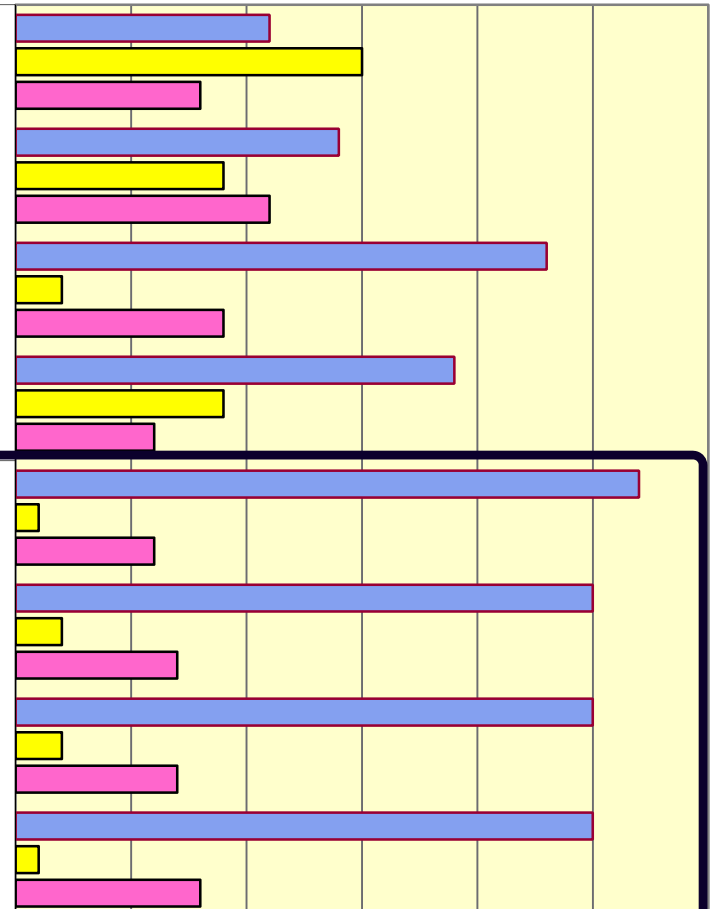
0 5 10 15 20 25 30

Quantitative

- Is 'number of throw-catch pairs' useful?
- Is 'exception concentration/distribution' information useful?
- Is 'exception-dependency information' useful?
- Is 'number of exceptions of a type' information useful?

Contextual

- Is visually representing 'where exception originated' useful?
- Is visually representing 'where exception caught' useful?
- Is visually representing 'what type of exception occurred' useful?
- Is visually representing 'complete exception flow' useful?



Yes

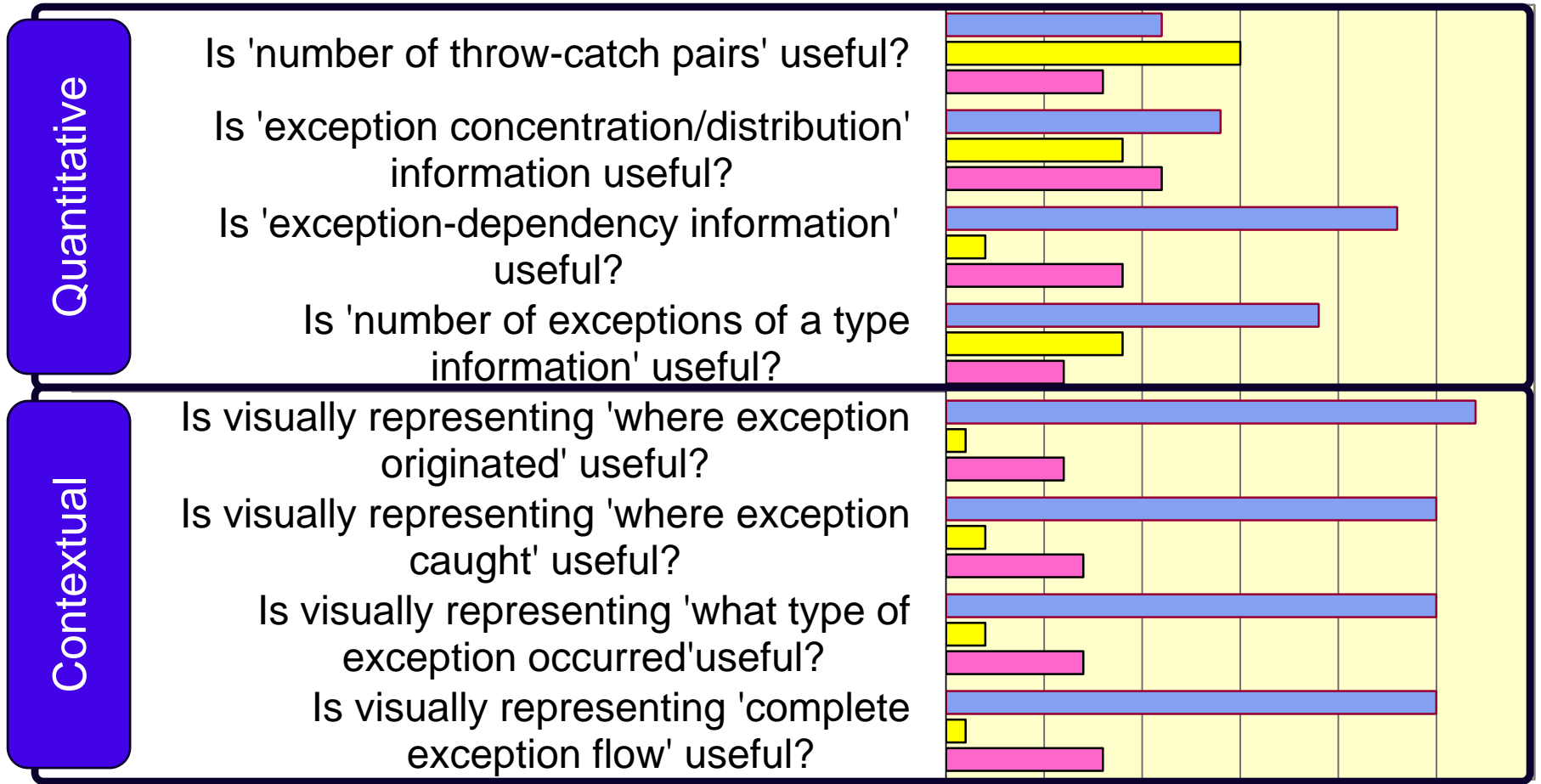
No

Maybe

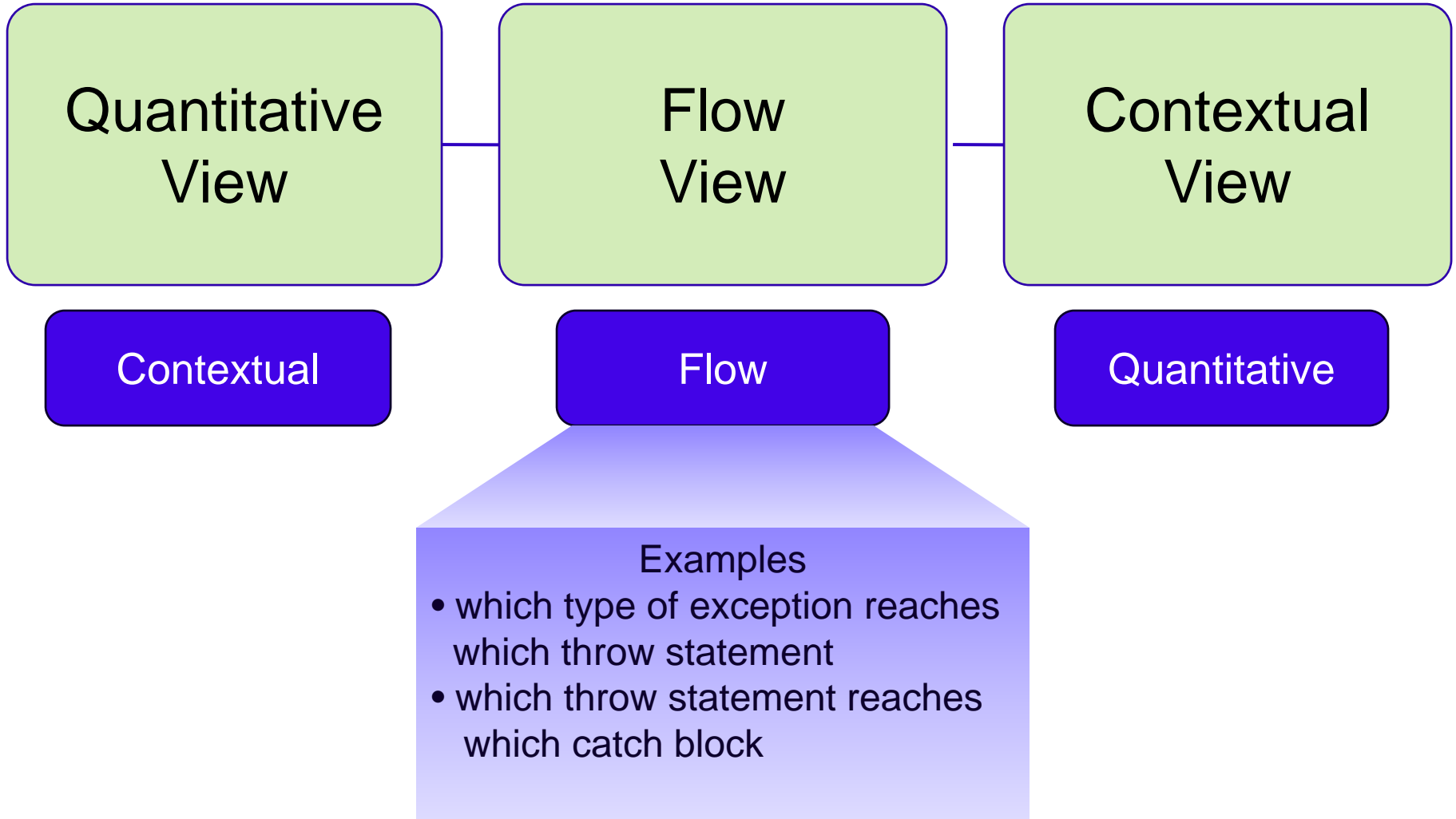
Exception-Related Questions

Number of Participants

0 5 10 15 20 25 30



Visualization

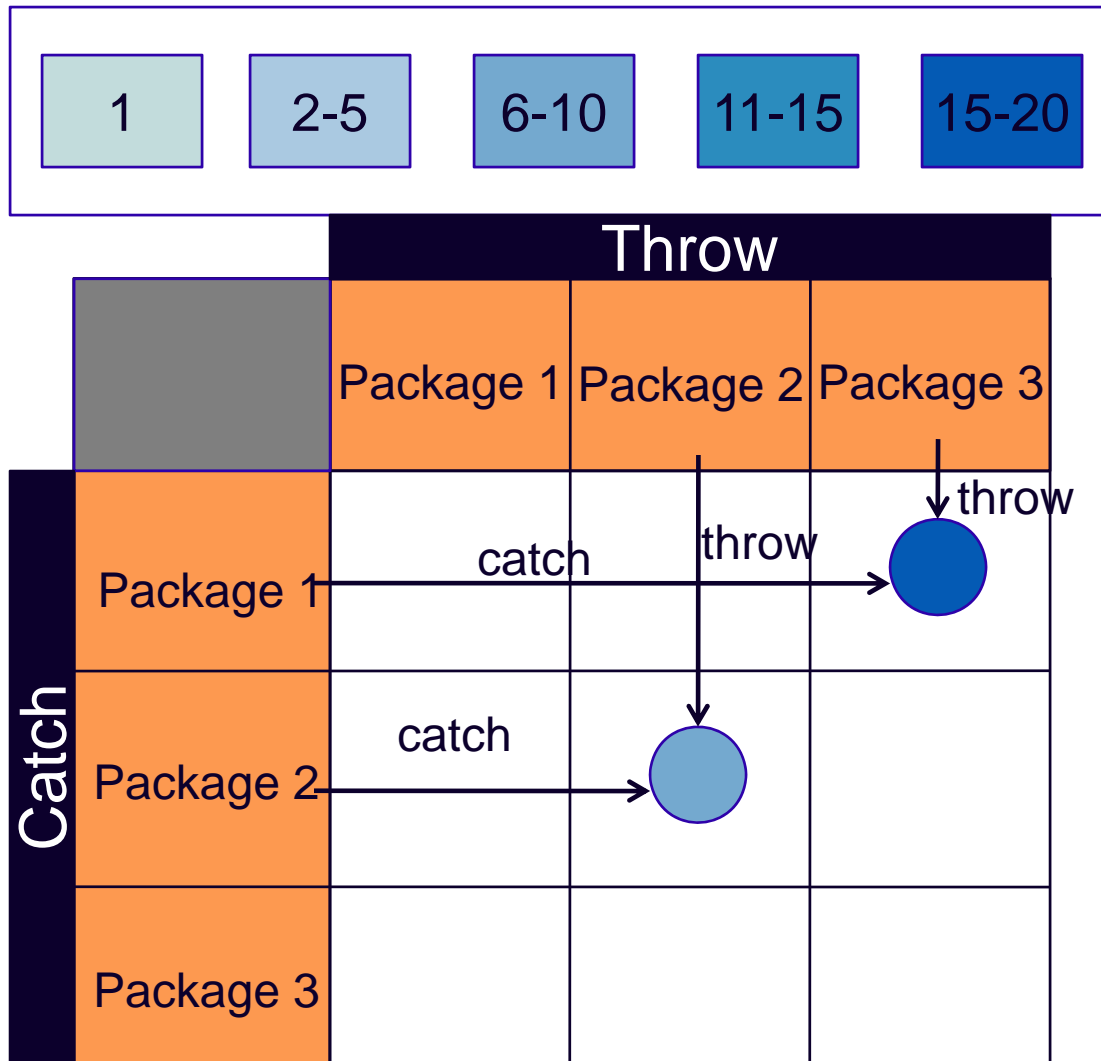


Visualization

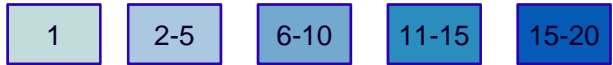
Color Scheme
Packages
Classes
Methods
Exception-handling constructs

Example
NanoXML <ul style="list-style-type: none">• 2700 LOC• 3 packages• 5 classes• 85 methods <p>(Downloaded from: http://nanoxml.sourceforge.net/ original)</p>

Quantitative View



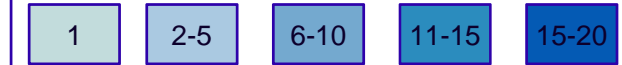
Quantitative View



	Package 1	Package 2	Package 3
Package 1			● (darkest blue)
Package 2		● (medium blue)	
Package 3			

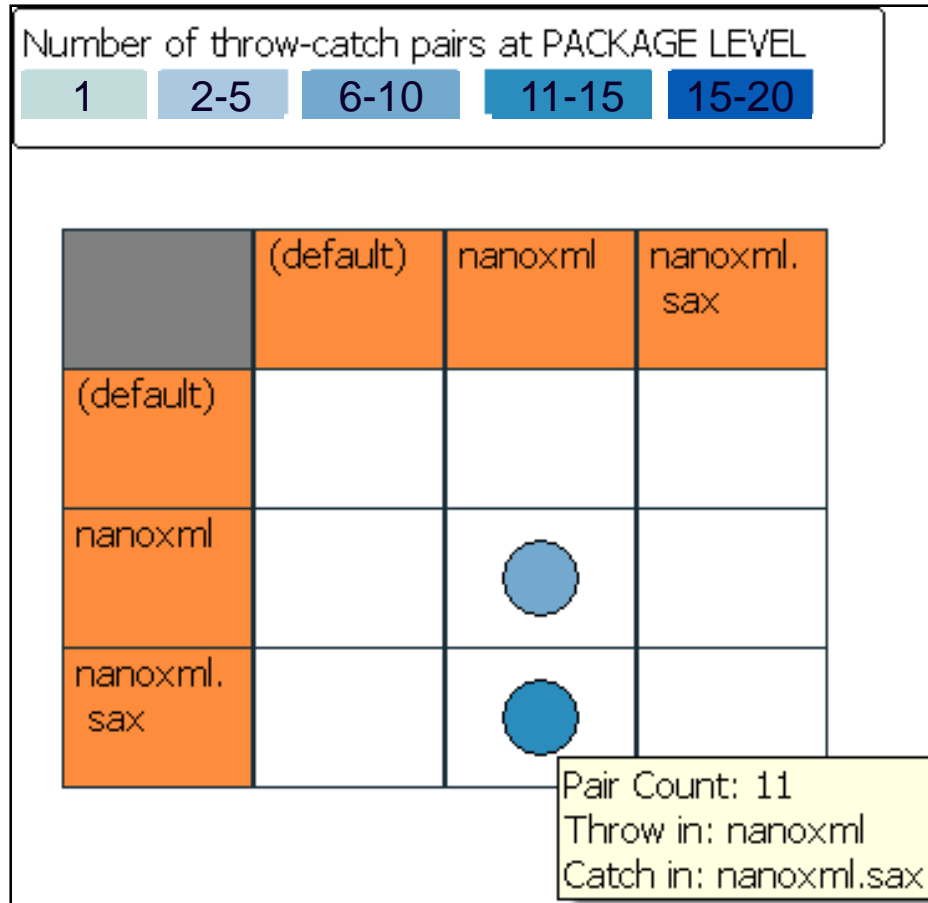


	Class 1	Class 2	Class 3
Class 1			○ (lightest blue)
Class 2	● (darkest blue)		
Class 3		● (medium blue)	



	Method 1	Method 2	Method 3
Method 1	● (medium blue)		
Method 2			● (darkest blue)
Method 3	○ (lightest blue)		

Quantitative View - NanoXML



Flow View

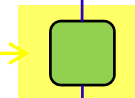
```
method(){
  MyException e = new MyException();
  try{
    ...
    if(flag==null){
      throw e;
    }else{
      // normal execution
    }
  }catch(MyException e){
    // recovery code
  }
  finally{
    // clean up code
  }
}
```

Flow View

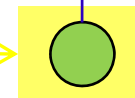
```
method(){  
  MyException e = new MyException();  
  try{  
    ...  
    if(flag==null){  
      throw e;  
    }else{  
      // normal execution  
    }  
  }catch(MyException e){  
    // recovery code  
  }  
  finally{  
    // clean up code  
  }  
}
```



type definition

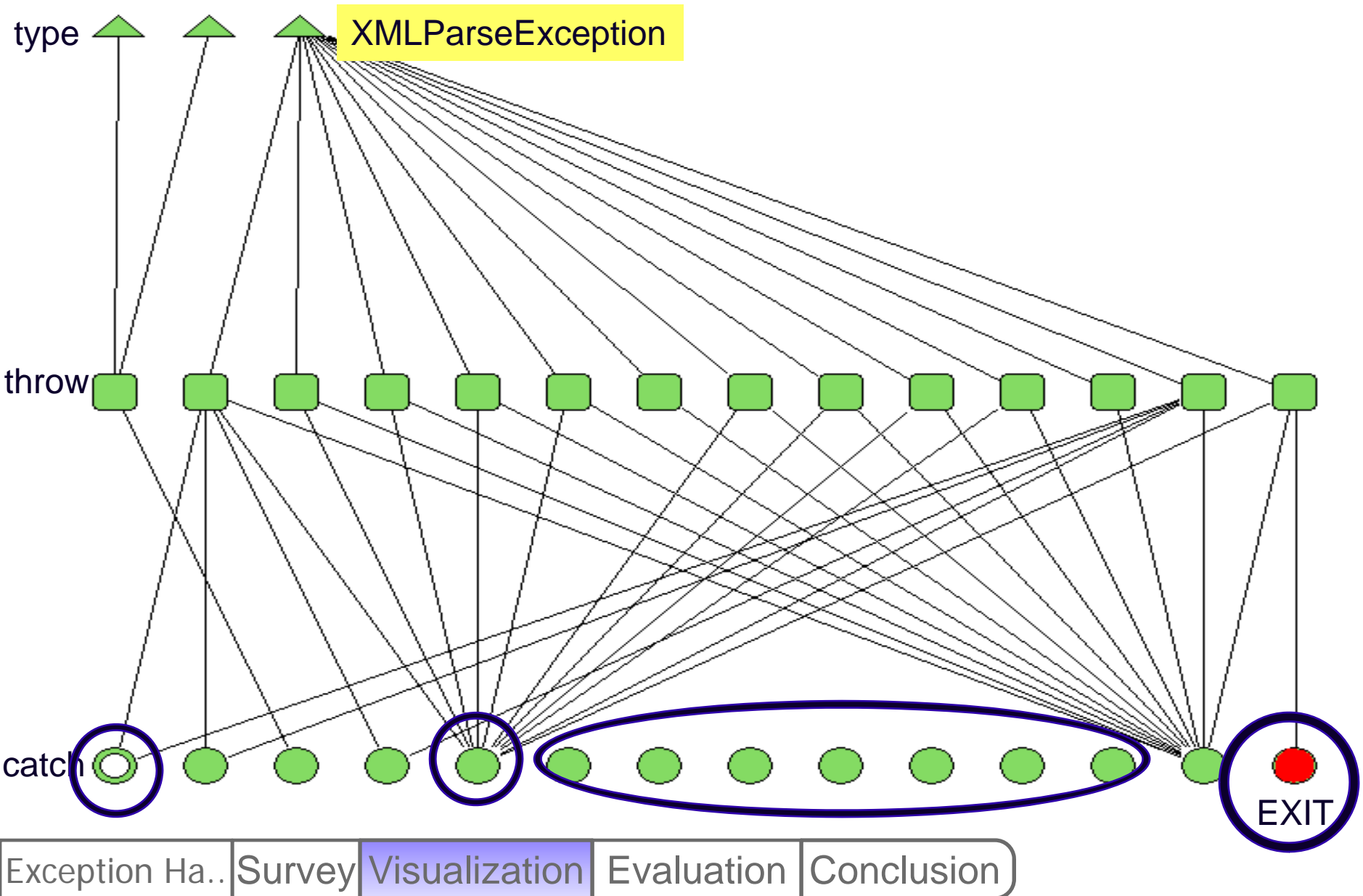


Throw statement

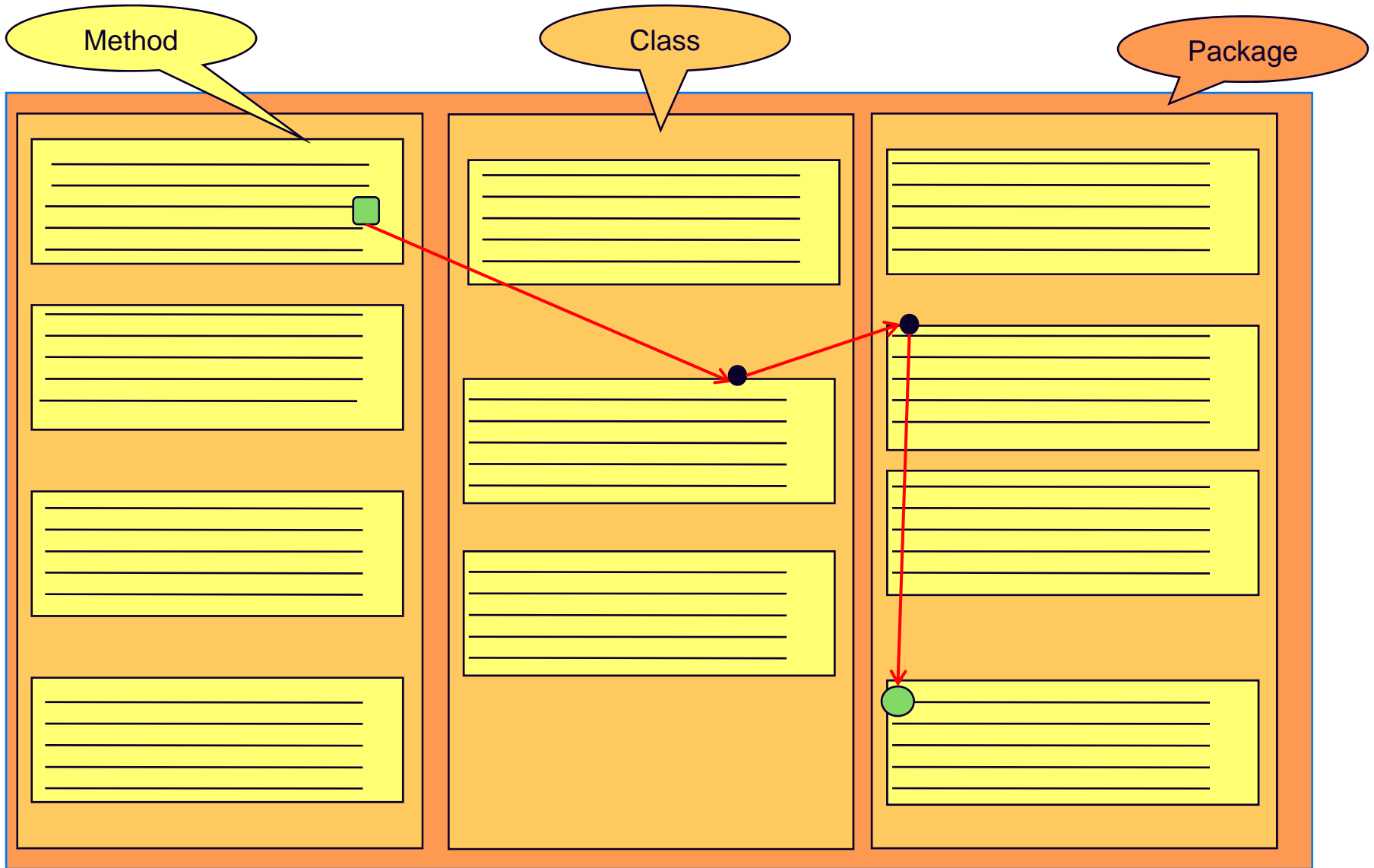


Catch statement

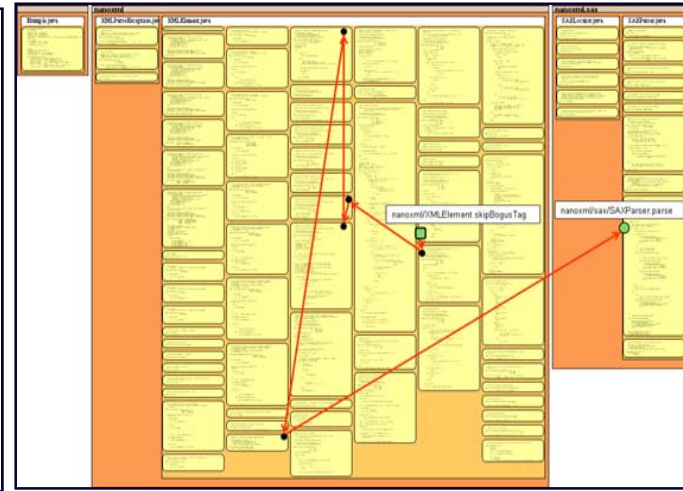
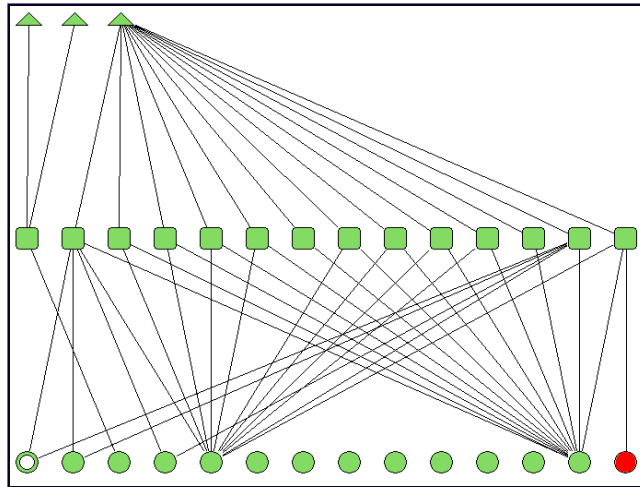
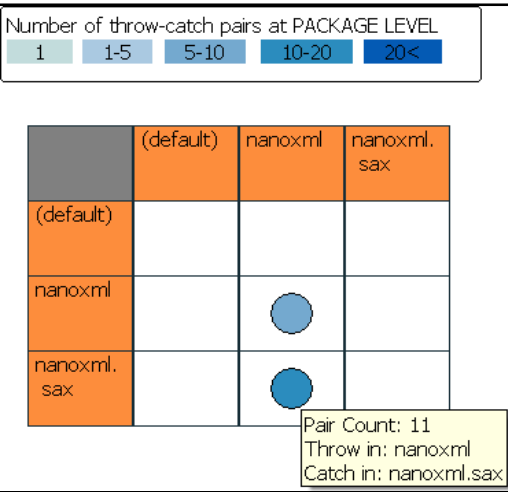
Flow View - NanoXML



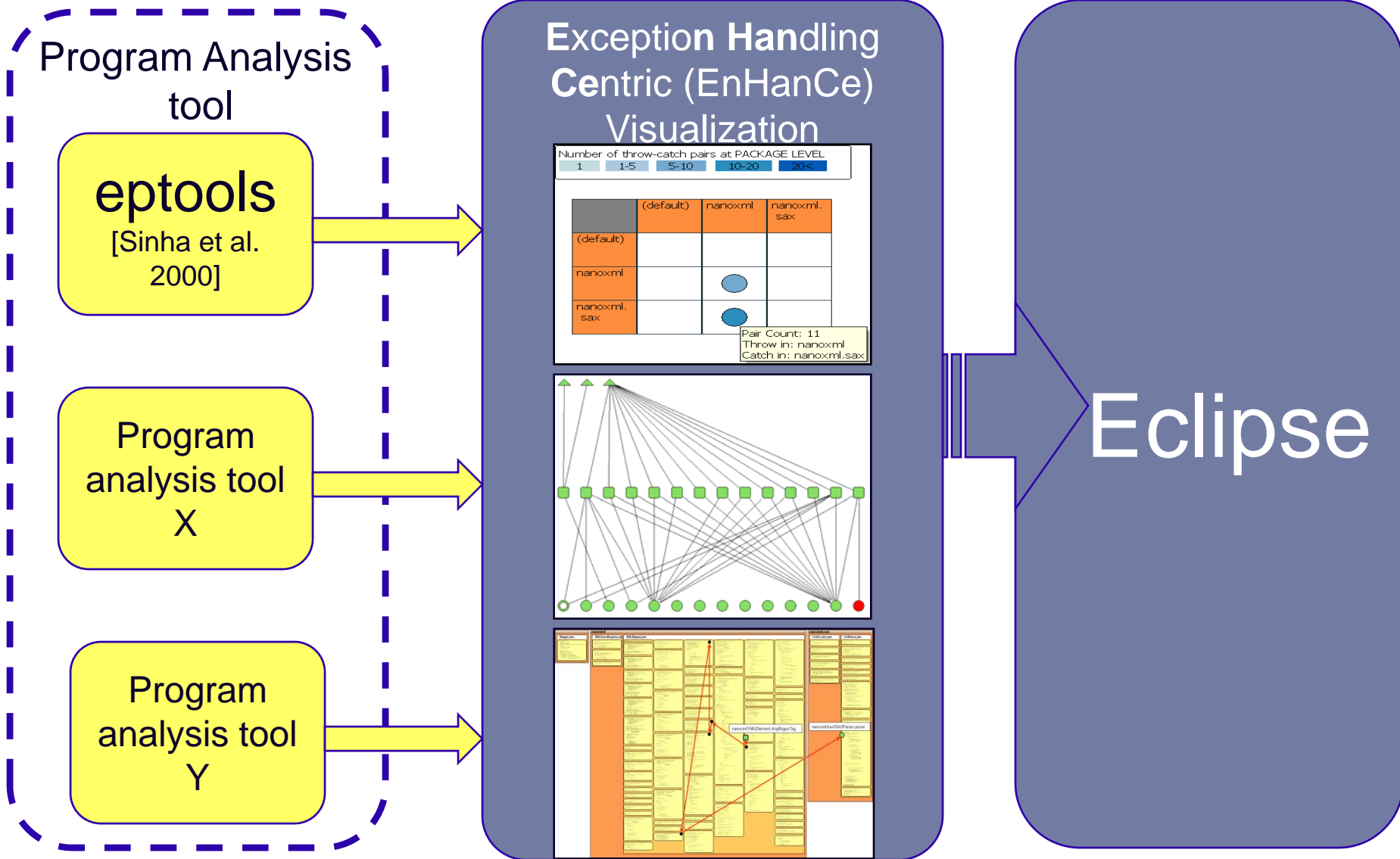
Contextual View



Three Visualizations



EnHanCe Visualization



EnHanCe Screenshot

The screenshot shows the Eclipse IDE with the following components:

- Code Editor:** Displays the `XMLParserFactory.java` file with the following code snippet:

```
cls = Class.forName(XMLParserFactory.VALIDATOR_CLASS);
validator = (IXMLValidator) cls.newInstance();
catch (Exception e) {
    // we can safely ignore any exceptions here
}
```
- EnHanCe View:** A tool window with three tabs: Contextual View, Flow View, and Quantitative View. The Contextual View is active, showing a tree structure of exception types and statements.
 - Exception Type:** A tree showing `Exception` (checked), `XMLParseException` (checked), and `IndexOutOfBoundsException` (checked).
 - Throw Statements:** A tree showing `NanoXML(2)` (checked), `(default package)` (checked), and `EqualizeLineNumt` (checked).
 - Catch Statements:** A tree showing `(default package)` (checked), `EqualizeLineNumt` (checked), and `EqualizeLineN` (checked).
 - Finally Statements:** A tree showing `NanoXML(2)` (checked).
- Flow Graph:** A graph visualization with two rows of nodes. The top row has 13 green square nodes, and the bottom row has 13 green circular nodes. A red circular node is at the far right of the bottom row. Lines connect the nodes, representing control flow or exception handling relationships.

Evaluation

Goals

Understand

- *current approach*- how developers deal with exception-handling constructs
- *visualization usefulness* - whether visualization will help better understand exception-handling

Process

- created interview guide
- conducted pilot study with three participants (1-4 years experience)
- conducted detailed study with eight participants (1-10 years experience)

Results

- Current approach
 - “ignore-for-now” approach
 - primarily for debugging
 - perceived as forced
- Visualization usefulness
 - + flow view reveals interesting patterns
 - + contextual view simplifies search tasks
 - quantitative view less clear
 - information about *finally*, rethrown exceptions absent
 - questions about scalability

Conclusion and Future Work

- Improvements
 - scalability
 - quantitative view redesign
- Enhancements
 - show *finally* related information
 - show rethrown *exceptions* information
 - show runtime exception-handling
- Additional studies

Contributions

1. Conducted survey to understand developers needs
2. Designed a visualization with three views
 - Quantitative View
 - Flow View
 - Contextual View
3. Implemented visualization as an Eclipse plugin
 - EnHanCe
4. Conducted studies to evaluate the visualization tool

QUESTIONS!

THANK YOU!

Extra --- EnHanCe Patterns

Patterns

- Type centric
 - Subgraph consisting of type definition (under consideration), reachable throws, and reachable catches
- Throw centric
 - Subgraph consisting of reachable type, throw statement (under consideration), and reachable catches
- Catch centric
 - Subgraph consisting of reachable type, reachable throws, and catch (under consideration)

Helps in testing and refactoring

Extra --- Quantitative View Concerns

- Quantitative view
 - is not intuitive (difficult to understand the information it is representing, without being explained); instruction info will help
 - Blue shades need to be changed
 - Confused with the columns and row reading