

Secure Data-Flow Compliance Checks between Models and Code based on Automated Mappings

 UNIVERSITÄT
KOBLENZ · LANDAU UNIVERSITY OF
GOTHENBURG CHALMERS
UNIVERSITY OF TECHNOLOGY Fraunhofer

Sven Peldszus¹, Katja Tuma², Daniel Strüber²,
Jan Jürjens^{1,3}, Riccardo Scandariato²

- 1) University of Koblenz-Landau, Germany
- 2) University of Gothenburg and
Chalmers University of Technology, Sweden
- 3) Fraunhofer Institute for Software and
Systems Engineering, Germany

Motivation

The New York Times

After a Data Breach, British Airways Faces a Record Fine



A photograph of a British Airways airplane on a runway. The aircraft is white with red and blue stripes on the tail and side. The word "BA" is visible on the tail fin.

- Security vulnerabilities can be expensive
- Nearly every kind of software can be affected

BBC Sign in News Sport Reel Worklife

NEWS

Home | Video | World | UK | Business | Tech | Science | Stories | Technology

Apple's App Store infected with XcodeGhost malware in China

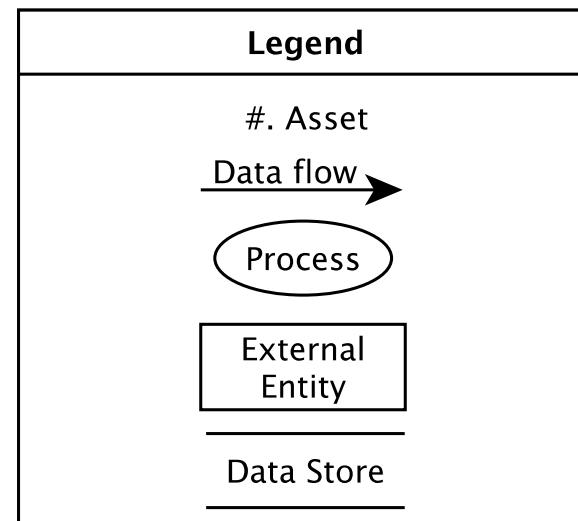
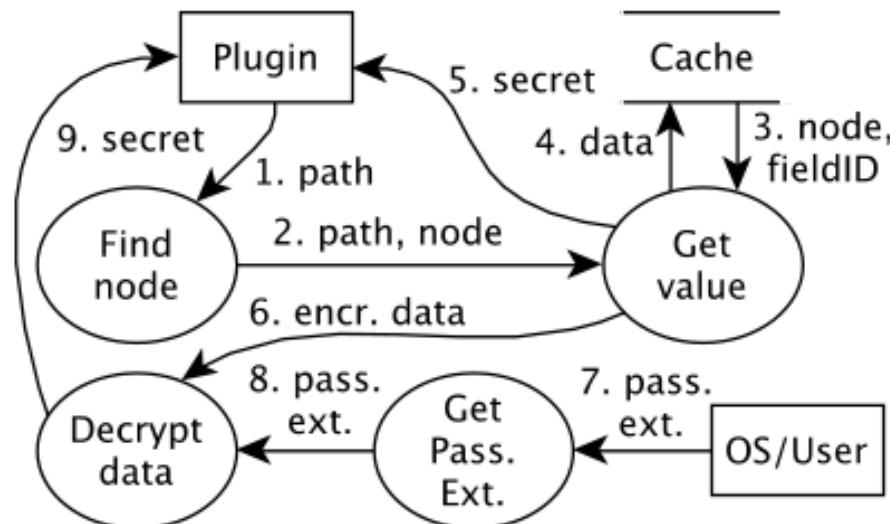
© 21 September 2015   



A large Apple logo is centered on a background composed of a geometric pattern of overlapping triangles in shades of green, blue, and white.

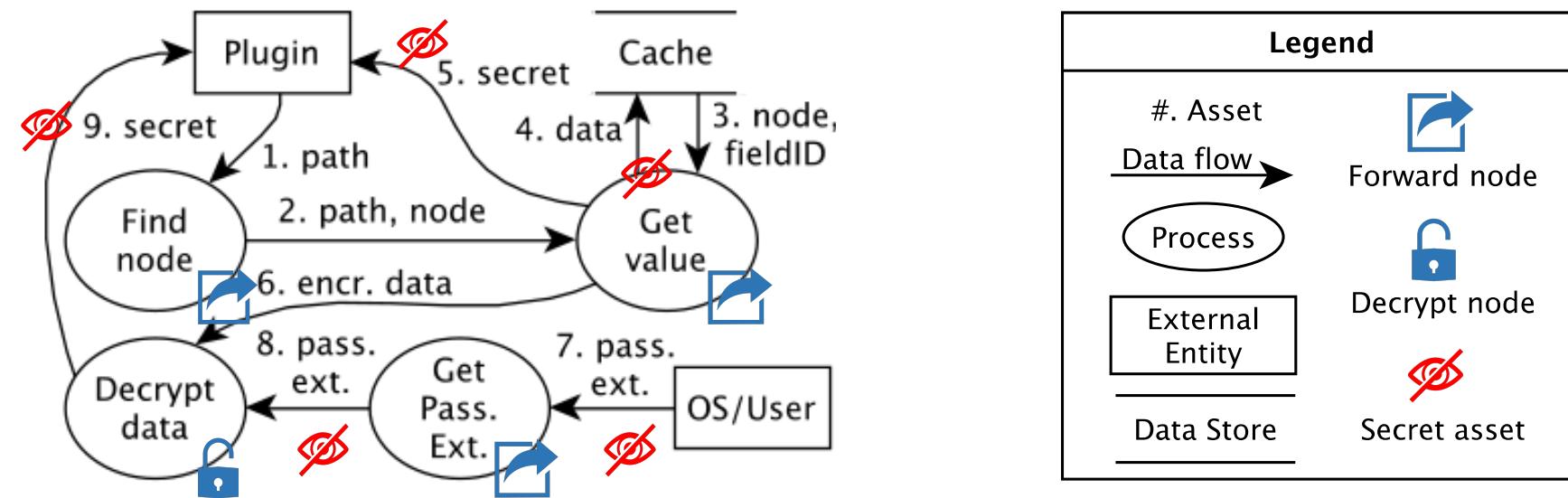
Model-based Security Engineering

- **Security by design:** System's assets and threats have to be defined early
- Specification of the system (e.g. Eclipse Secure Storage) using a **Security Data Flow Diagram (SecDFD)**



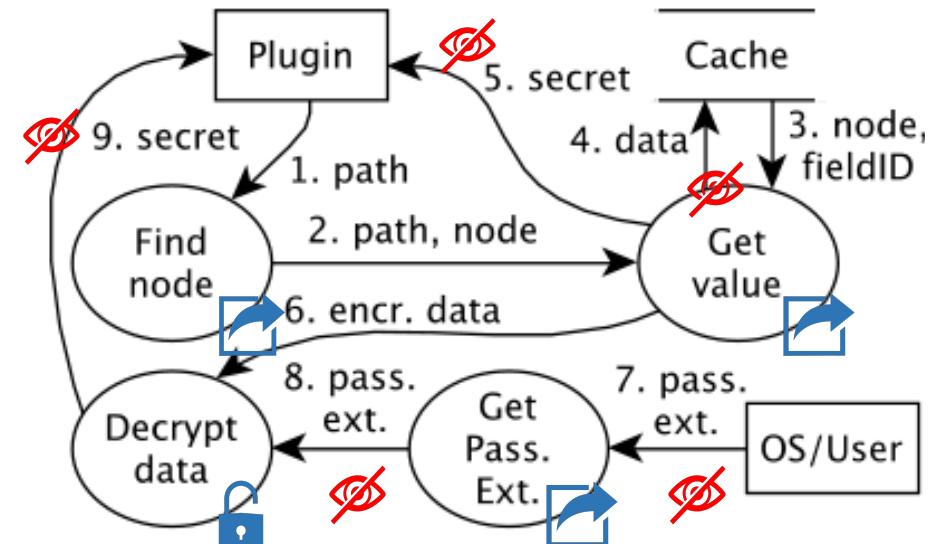
Model-based Security Engineering

- **Security by design:** System's assets and threats have to be defined early
- Specification of the system (e.g. Eclipse Secure Storage) using a **Security Data Flow Diagram (SecDFD)**



Implementation of the Models

- > **Security by design:** System's assets and threats have to be defined early
- > Specification of the system (e.g. Eclipse Secure Storage) using a **Security Data Flow Diagram (SecDFD)**

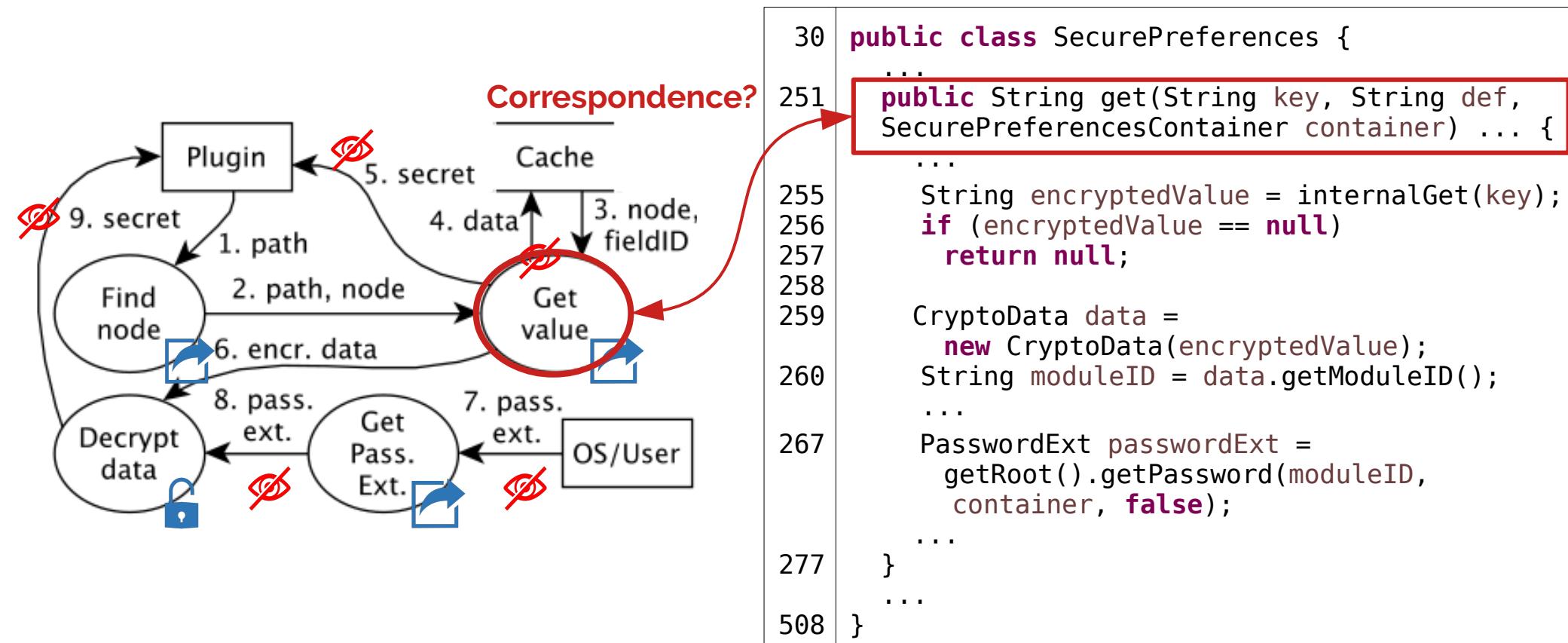


```

30 public class SecurePreferences {
...
251     public String get(String key, String def,
SecurePreferencesContainer container) ... {
...
255         ...
256         String encryptedValue = internalGet(key);
257         if (encryptedValue == null)
258             return null;
259
260         CryptoData data =
261             new CryptoData(encryptedValue);
262         String moduleID = data.getModuleID();
...
267         PasswordExt passwordExt =
268             getRoot().getPassword(moduleID,
269             container, false);
...
277     }
...
508 }
```

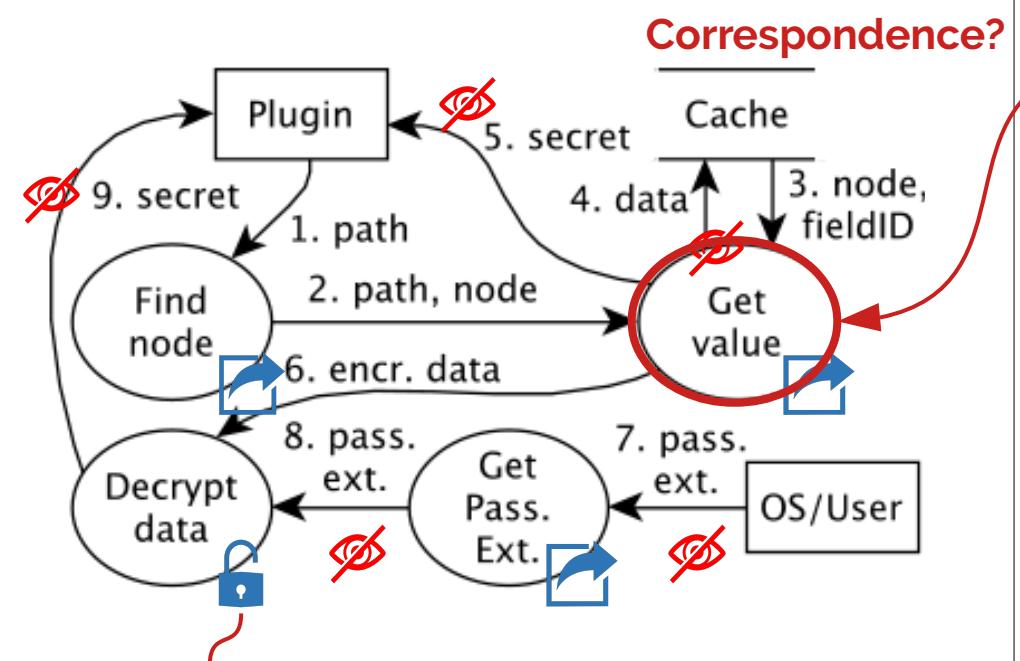
Compliance between Models & Code

- **Security by design:** System's assets and threats have to be defined early
- Specification of the system (e.g. Eclipse Secure Storage) using a **Security Data Flow Diagram (SecDFD)**



Compliance between Models & Code

- **Security by design:** System's assets and threats have to be defined early
 - Specification of the system (e.g. Eclipse Secure Storage) using a **Security Data Flow Diagram (SecDFD)**

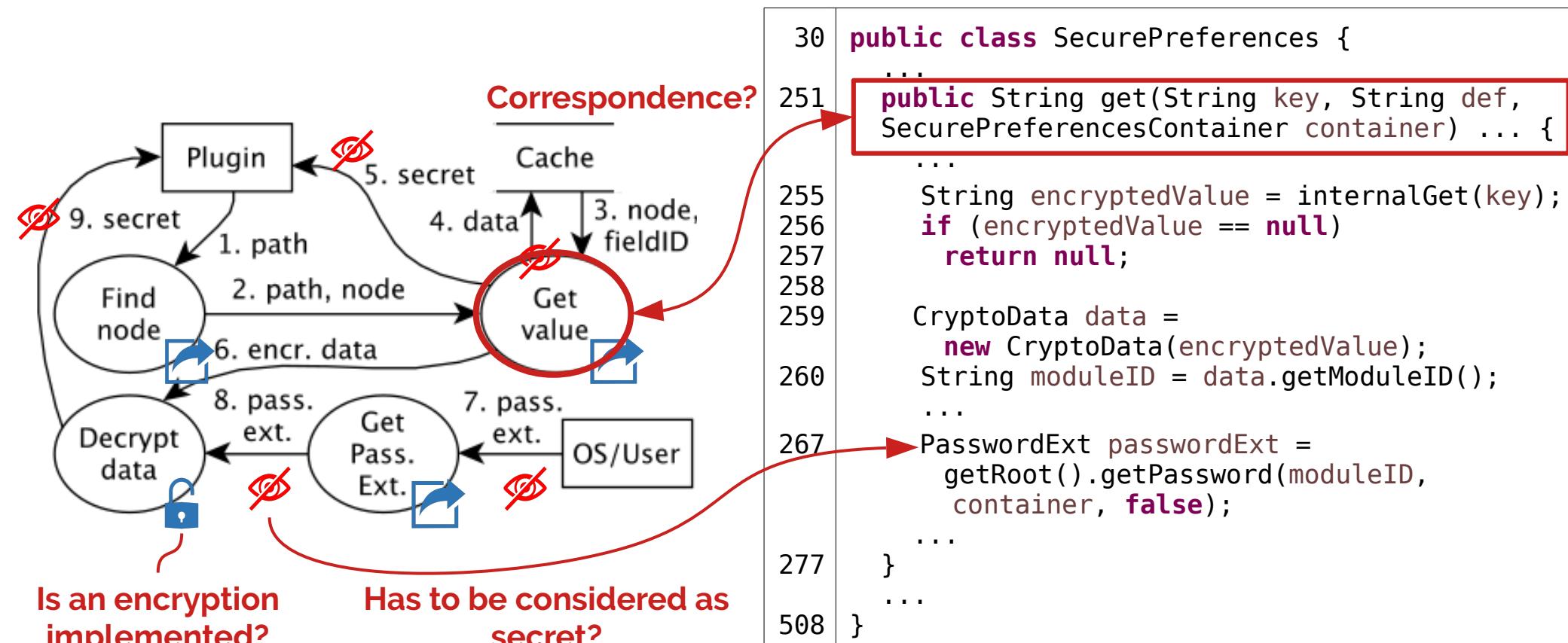


Is an encryption implemented?

```
30 public class SecurePreferences {  
31     ...  
251     public String get(String key, String def,  
252                         SecurePreferencesContainer container) ... {  
253         ...  
254         String encryptedValue = internalGet(key);  
255         if (encryptedValue == null)  
256             return null;  
257         ...  
258         CryptoData data =  
259             new CryptoData(encryptedValue);  
260         String moduleID = data.getModuleID();  
261         ...  
262         PasswordExt passwordExt =  
263             getRoot().getPassword(moduleID,  
264                 container, false);  
265         ...  
266     }  
267     ...  
508 }
```

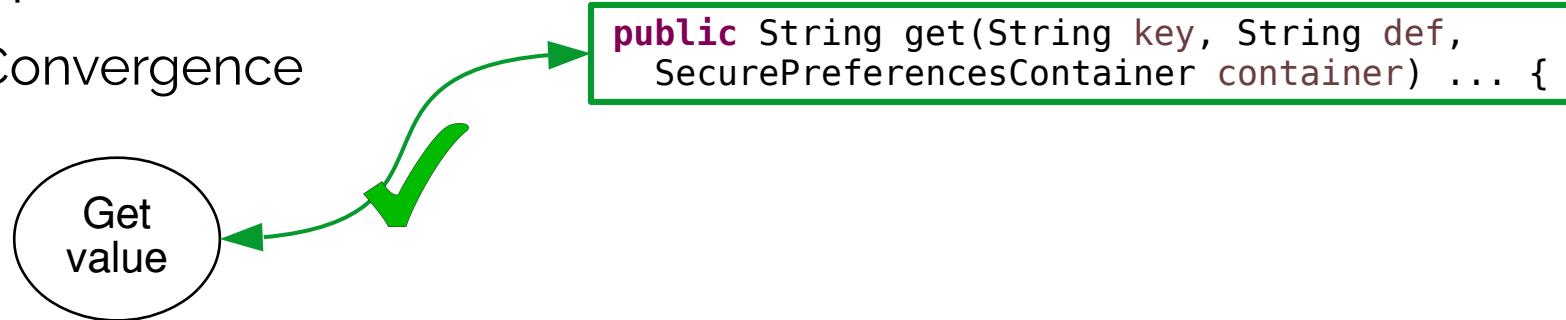
Compliance between Models & Code

- > **Security by design:** System's assets and threats have to be defined early
- > Specification of the system (e.g. Eclipse Secure Storage) using a **Security Data Flow Diagram (SecDFD)**



Compliance between Models & Code

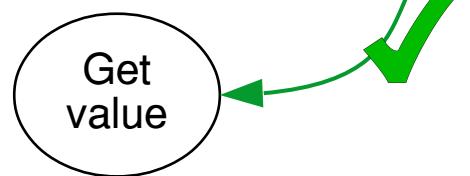
- Compliance Checks
 - Convergence



Compliance between Models & Code

➤ Compliance Checks

➤ Convergence



```
public String get(String key, String def,  
SecurePreferencesContainer container) ... {
```

➤ Divergence

value

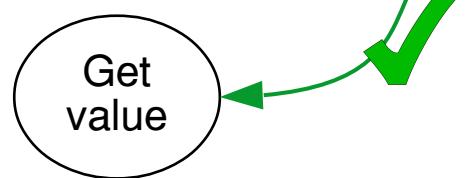


```
public Hash delete>Password pw) {  
Hash hash = pw.getHash();  
pw.delete();  
return hash;
```

Compliance between Models & Code

➤ Compliance Checks

➤ Convergence



```
public String get(String key, String def,  
SecurePreferencesContainer container) ... {
```

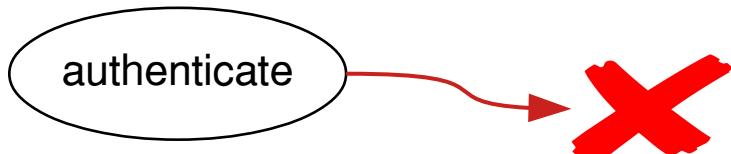
➤ Divergence

value

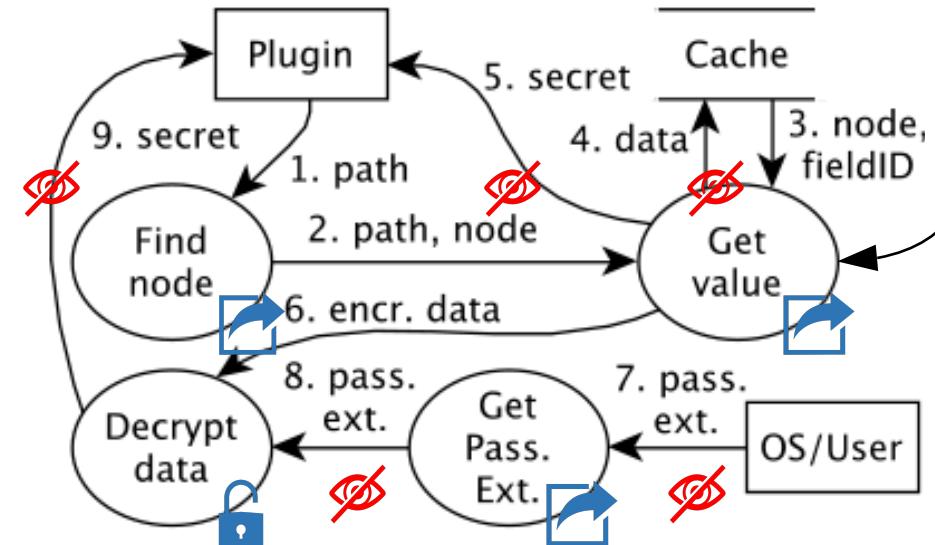
delete

```
public Hash delete>Password pw) {  
Hash hash = pw.getHash();  
pw.delete();  
return hash;
```

➤ Absence



Security Analysis of Source Code



```

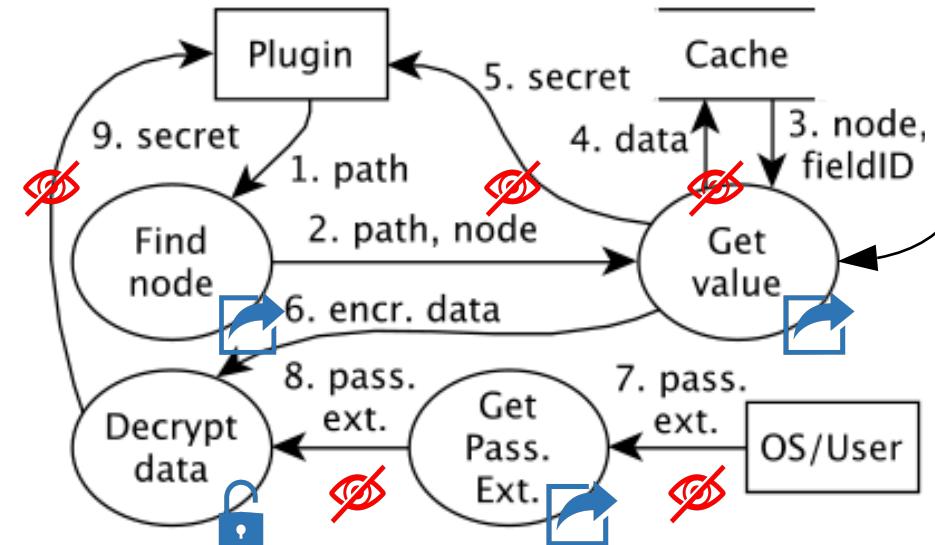
30 public class SecurePreferences {
...
251   public String get(String key, String def,
SecurePreferencesContainer container) ... {
...
255     ...
256     ...
257     ...
258     ...
259     ...
260     ...
267     ...
277     ...
508   }
...
}
  
```



A3-Sensitive Data Exposure

Security Analysis of Source Code

- Based on Security Labels
 - Calculate Security Metrics – e.g. Number of Classes containing Secret Data
 - Secure Data Flow Analysis
 - Runtime-Monitoring



```

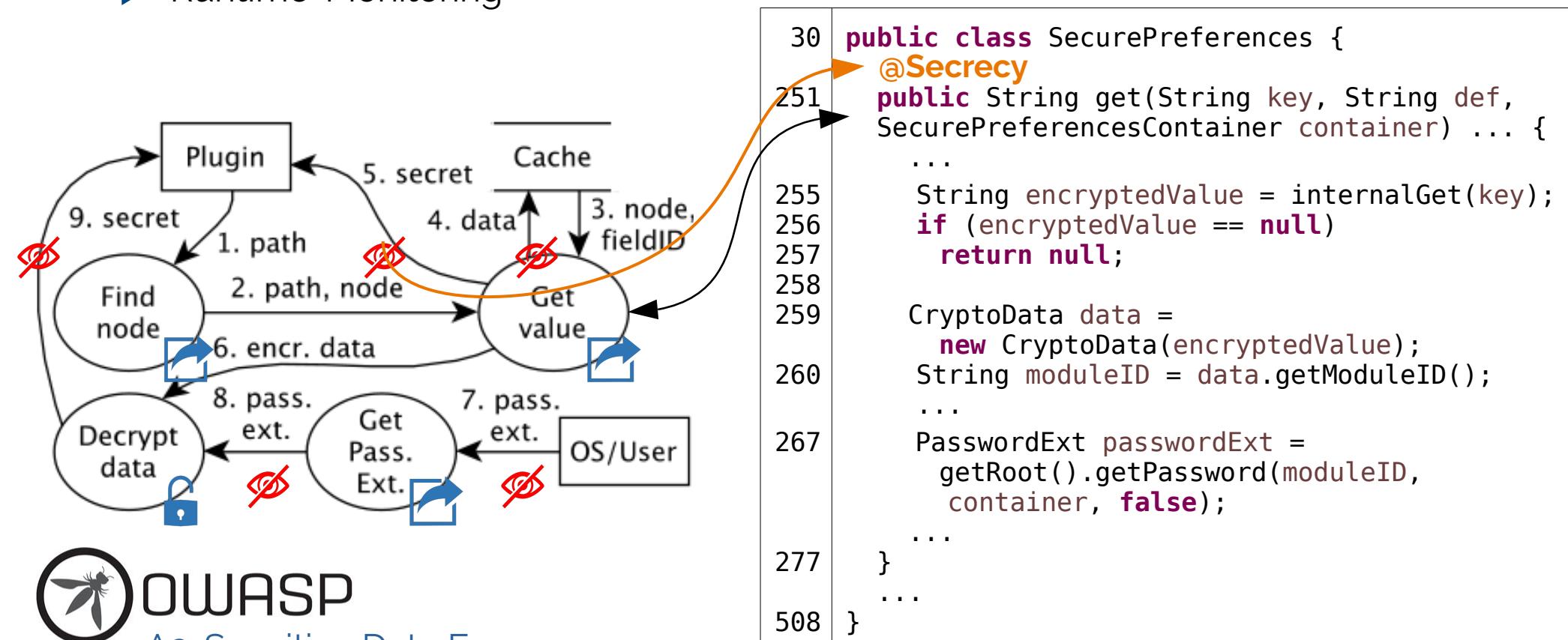
30 public class SecurePreferences {
...
251     public String get(String key, String def,
252                         SecurePreferencesContainer container) ... {
253         ...
254         String encryptedValue = internalGet(key);
255         if (encryptedValue == null)
256             return null;
257
258         CryptoData data =
259             new CryptoData(encryptedValue);
260         String moduleID = data.getModuleID();
261
262         ...
263
264         PasswordExt passwordExt =
265             getRoot().getPassword(moduleID,
266             container, false);
267
268         ...
269     }
270
271     ...
272 }
273
274 }
```



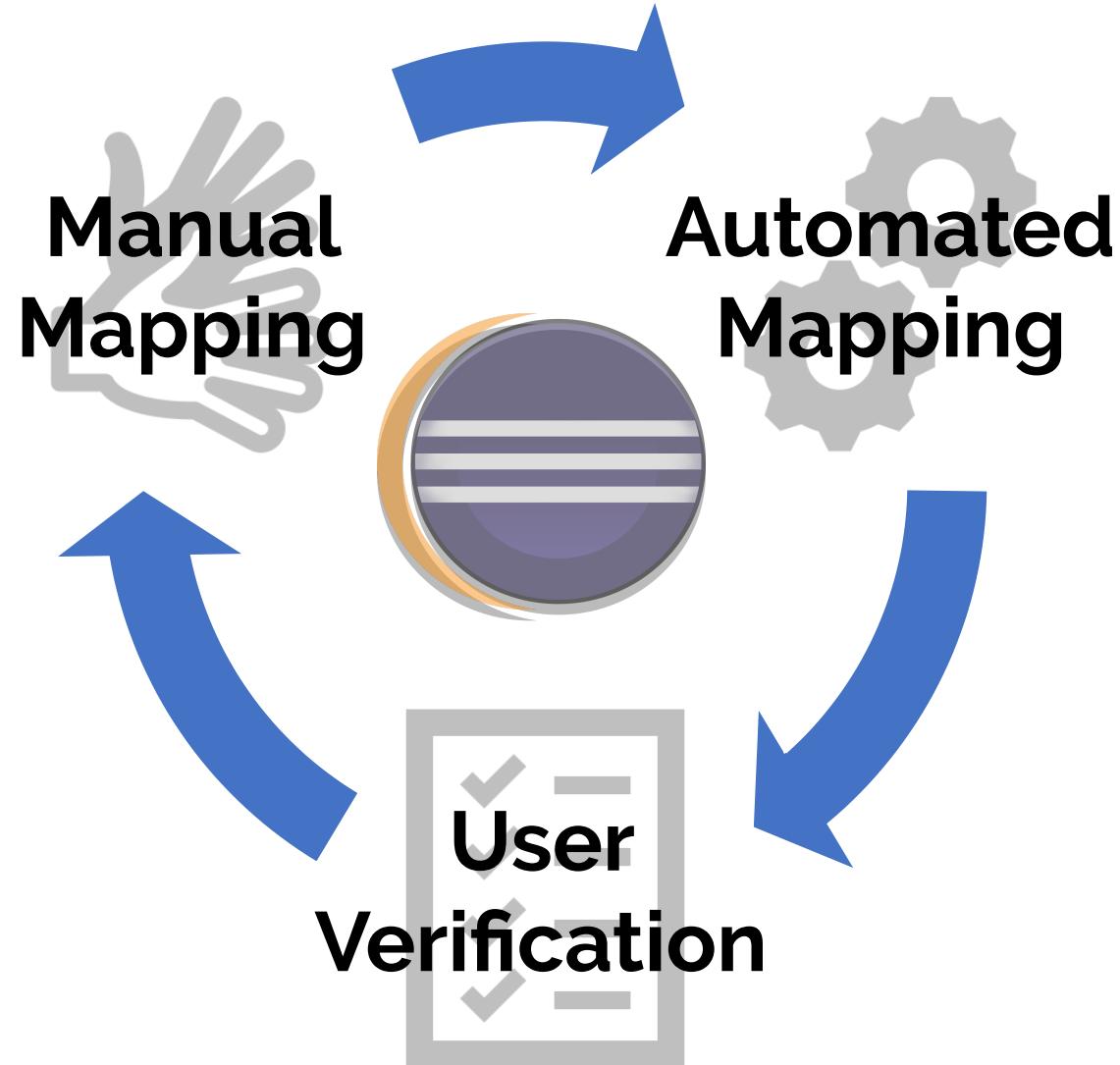
A3-Sensitive Data Exposure

Security Analysis of Source Code

- Based on Security Labels
 - Calculate Security Metrics – e.g. Number of Classes containing Secret Data
 - Secure Data Flow Analysis
 - Runtime-Monitoring

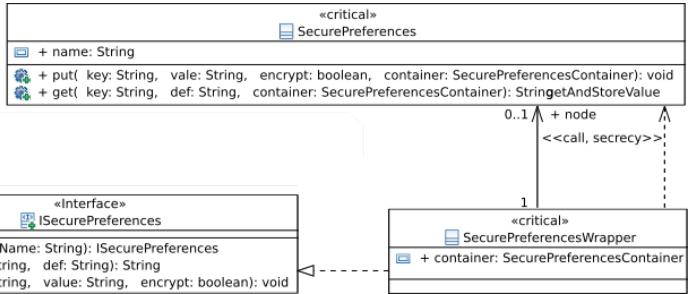


Mapping Generation: Semi-Automated Process



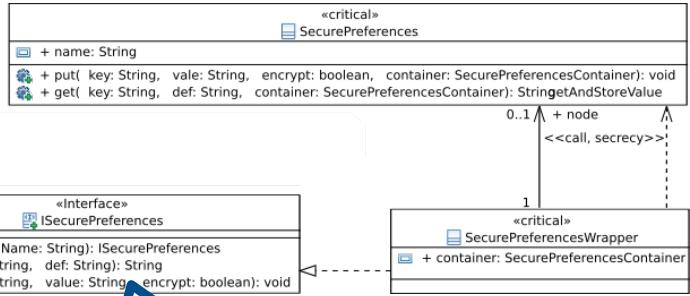
Abstract Program Representation for Design Flaw Detection: Program Model

UML Model



Abstract Program Representation for Design Flaw Detection: Program Model

UML Model

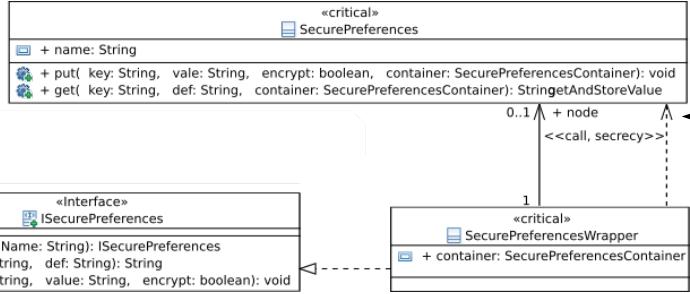


Misses crucial semantic information:

- method calls
- field accesses
- ...

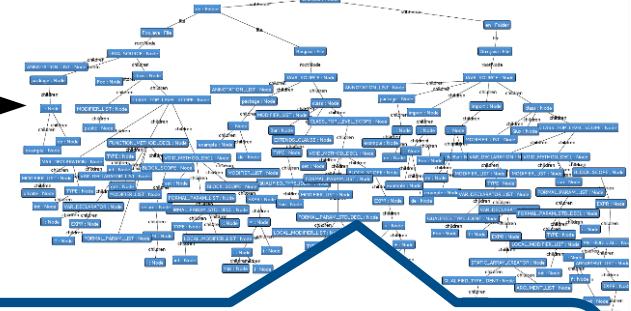
Abstract Program Representation for Design Flaw Detection: Program Model

UML Model



abstraction

Abstract Syntax Tree

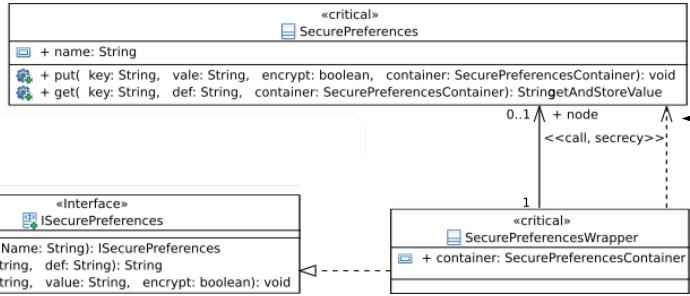


Too detailed and impractical:

- complete method implementations
- no simple entry points

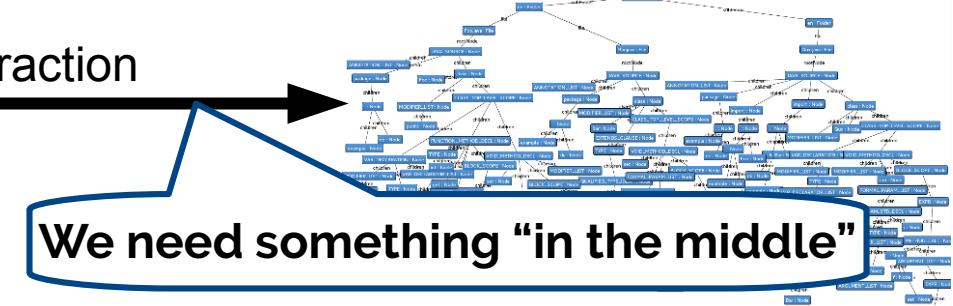
Abstract Program Representation for Design Flaw Detection: Program Model

UML Model



abstraction

Abstract Syntax Tree

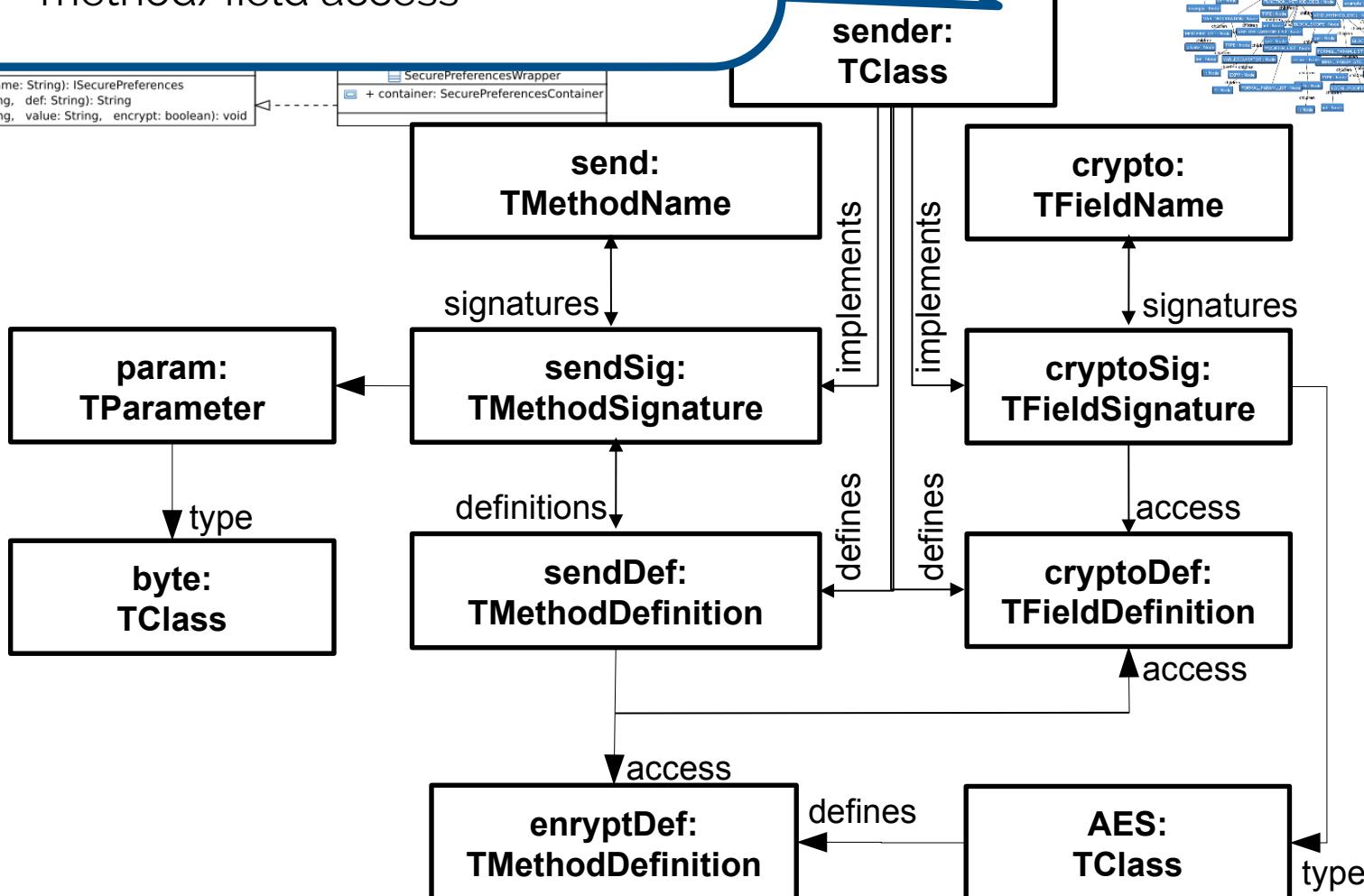
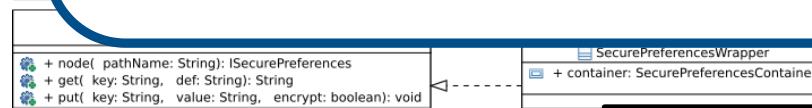


We need something “in the middle”

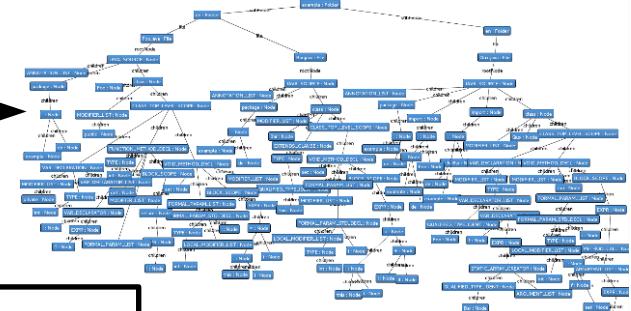
Abstract Program Representation for program Model

GRaViTY program model:

- **abstraction** with focus on class/method/field level
- enriched with **semantic information**:
 - overloading/overriding
 - method/field access



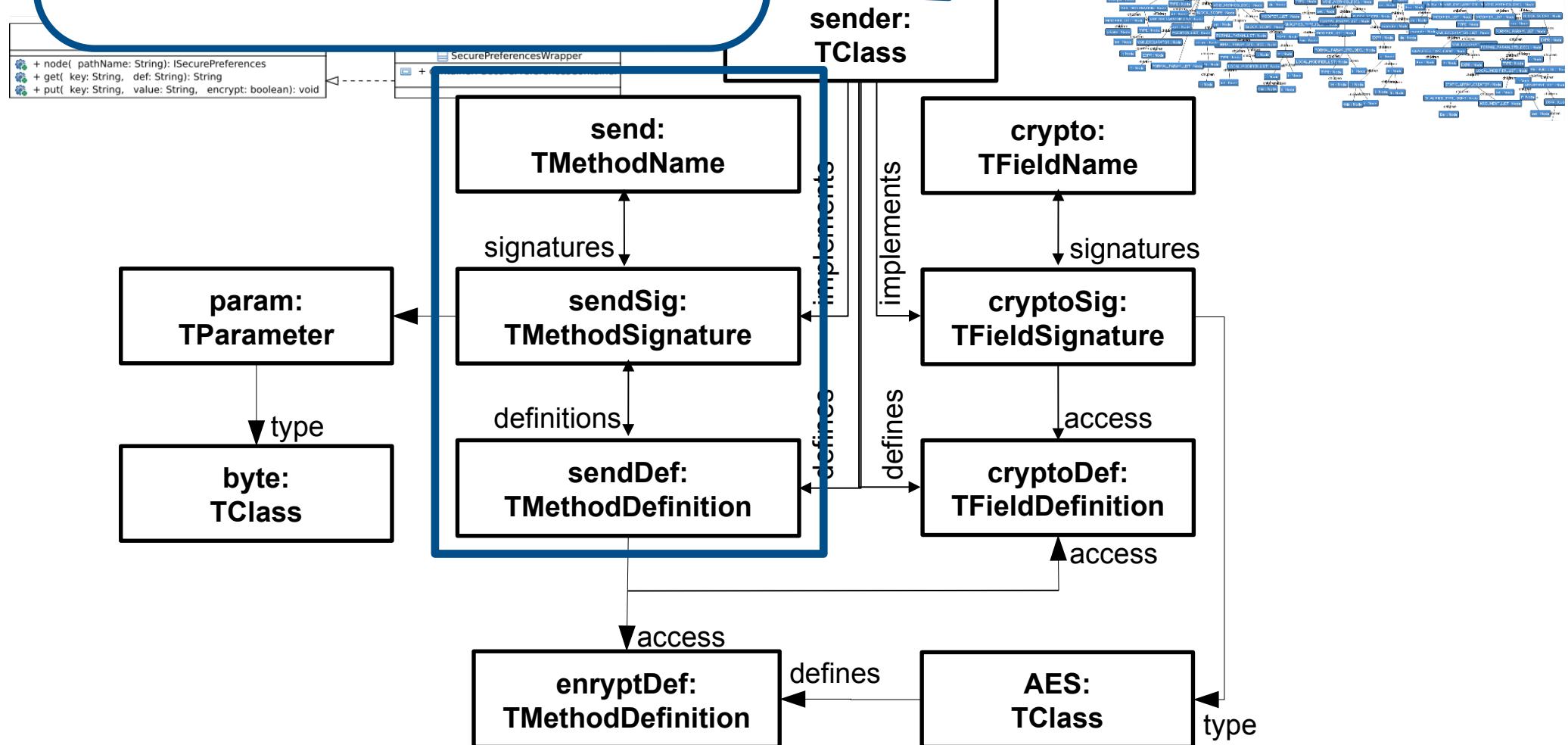
Abstract Syntax Tree



Abstract Program Representation for program Model

GRaViTY program model:

- **abstraction** with focus on class/method/field level
- enriched with **semantic information**:
 - overloading/overriding
 - method/field access

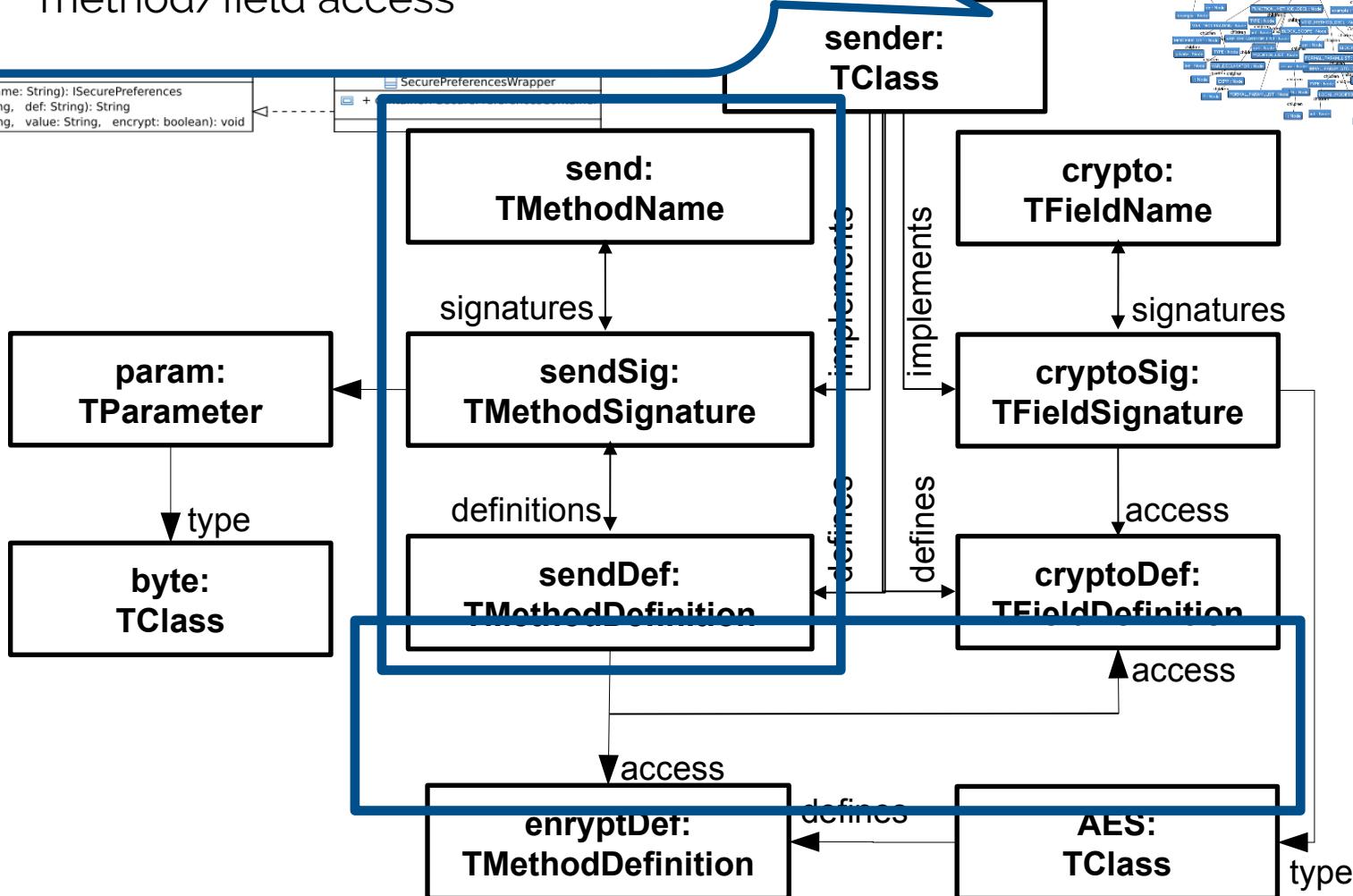
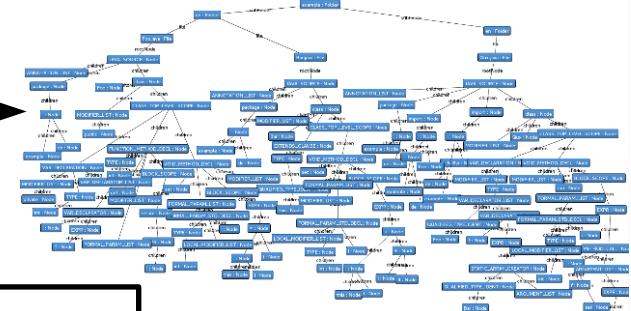


Abstract Program Representation for program Model

GRaViTY program model:

- **abstraction** with focus on class/method/field level
- enriched with **semantic information**:
 - overloading/overriding
 - method/field access

Abstract Syntax Tree

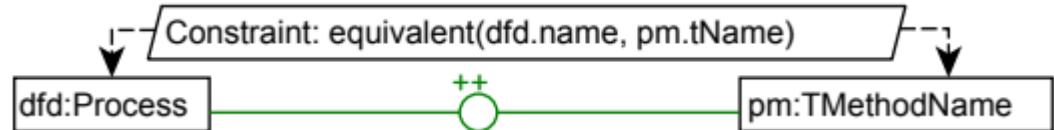


Mapping a DFD to the Program Model

- Iterative mapping

Mapping a DFD to the Program Model

- Iterative mapping
 - 1) Mapping of Names:
 - Using a Heuristic
 - Selecting all with a certainty higher than 70%



Mapping a DFD to the Program Model

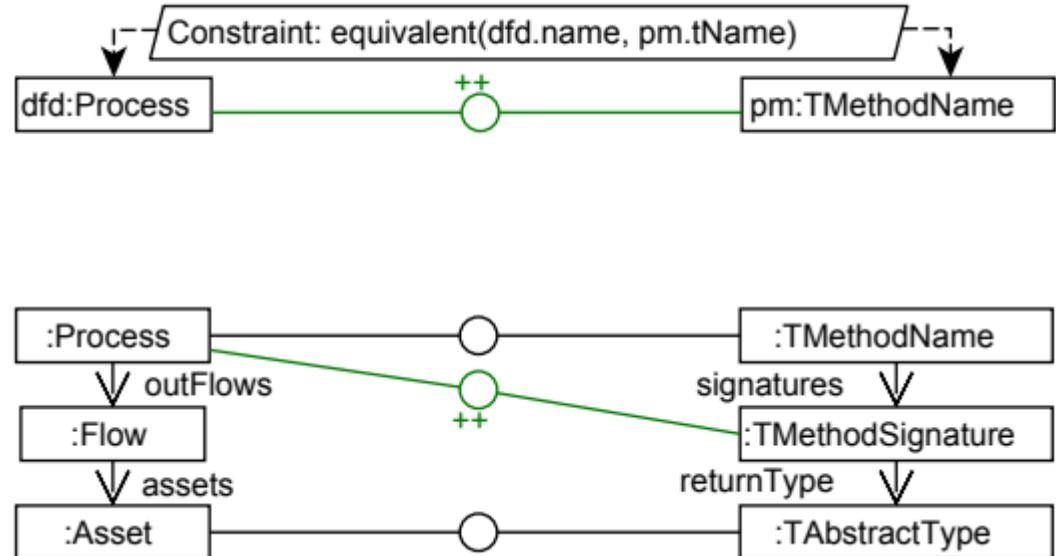
➤ Iterative mapping

➤ 1) Mapping of Names:

- Using a Heuristic
- Selecting all with a certainty higher than 70%

➤ 2) Mapping of Signatures:

- Selecting all with at least one parameter or return type mapping
- Certainty based on number of matches and name matching



Mapping a DFD to the Program Model

> Iterative mapping

> 1) Mapping of Names:

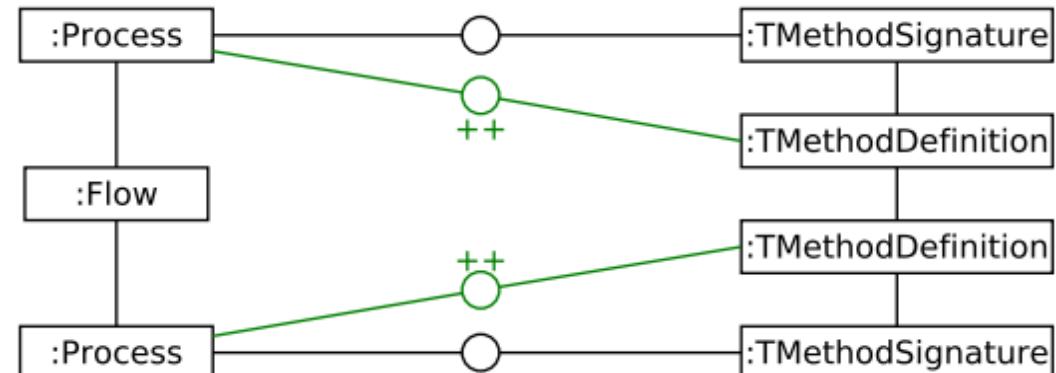
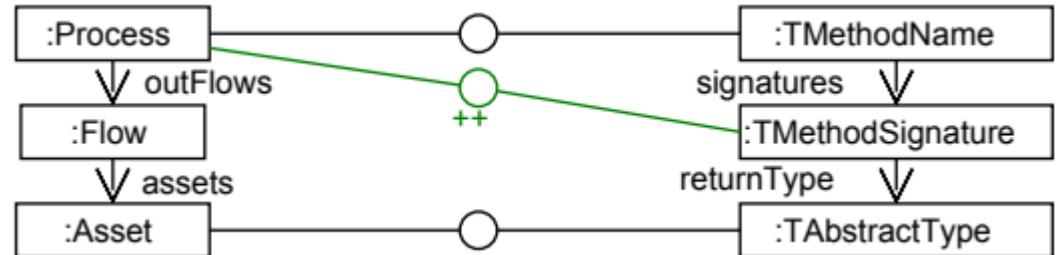
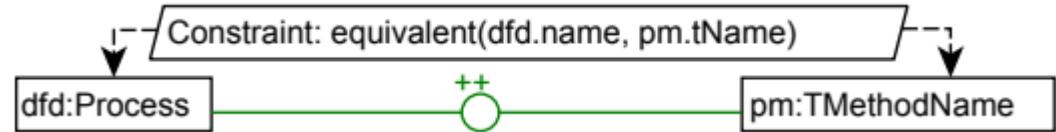
- > Using a Heuristic
- > Selecting all with a certainty higher than 70%

> 2) Mapping of Signatures:

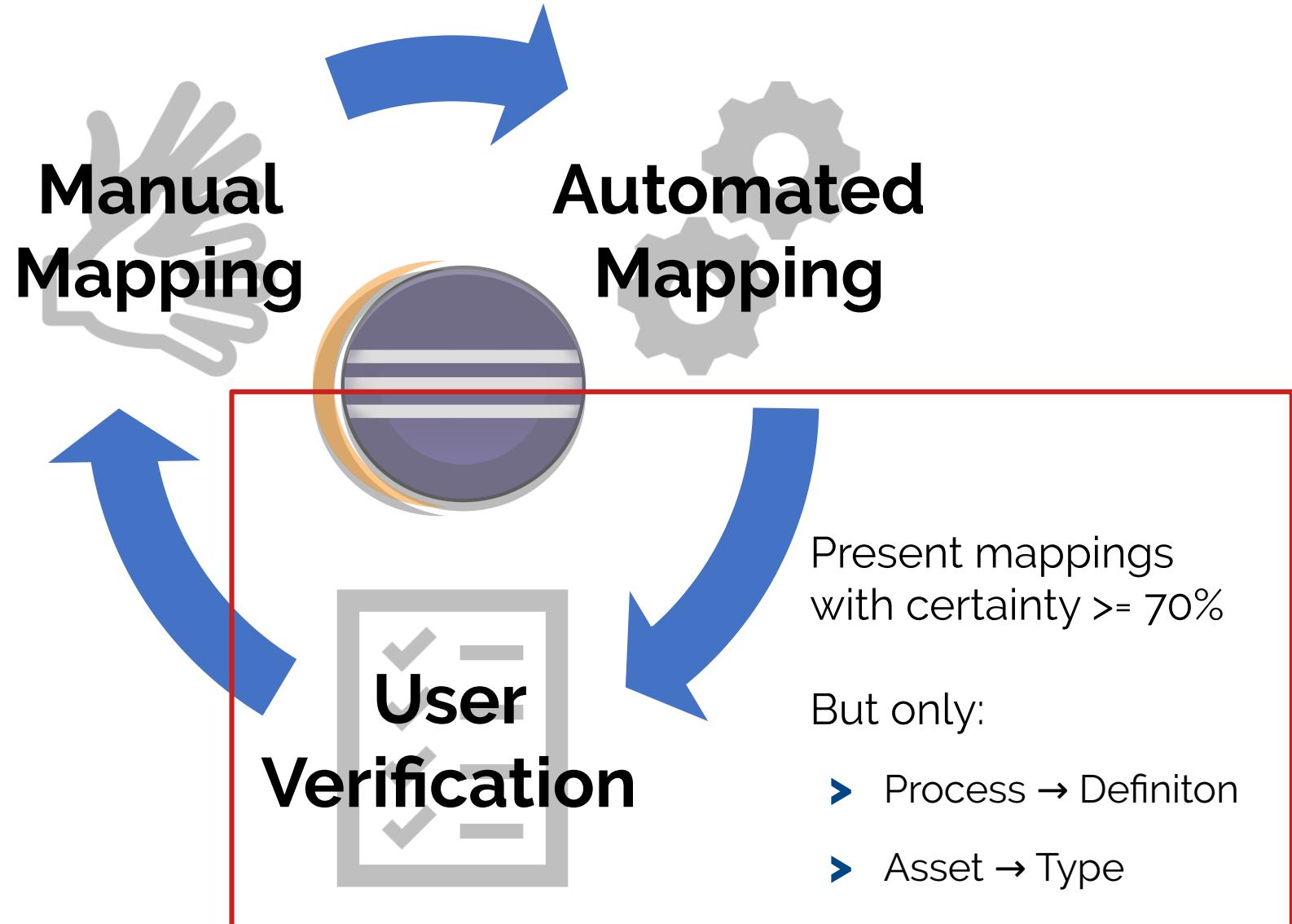
- > Selecting all with at least one parameter or return type mapping
- > Certainty based on number of matches and name matching

> 3) Mapping of Definitions:

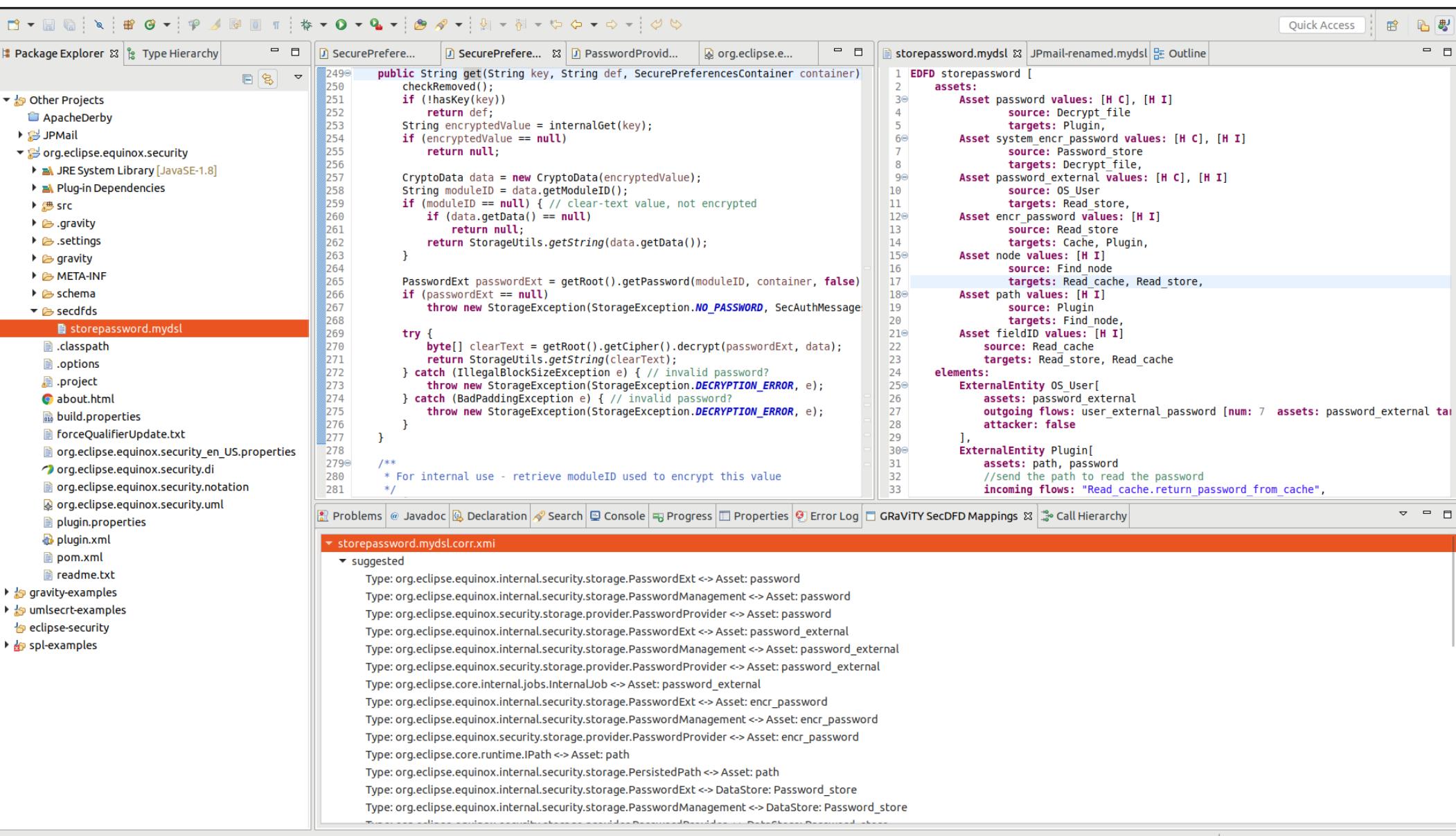
- > Certainty based on signature mappings



Mapping Generation: Semi-Automated Process



User Verification



The screenshot shows an IDE interface with several windows open, illustrating the process of verifying user data flow between code and models.

- Package Explorer:** Shows the project structure, including "org.eclipse.equinox.security" and its subfolders like "src", ".settings", and "META-INF". A file named "storepassword.mydsl" is selected.
- SecurePreferencesContainer.java:** A Java code editor showing a method for retrieving a password. It checks if the key exists, decrypts it if encrypted, and handles various exceptions related to decryption and password storage.
- storepassword.mydsl:** A GravITY model editor showing a state transition diagram. The initial state is "EDFD storepassword [assets:". It transitions through several states involving "Asset password values", "Asset system_encr_password values", and "Asset password_external values". The final state is "elements: ExternalEntity OS_User[assets: password_external outgoing flows: user_external_password [num: 7 assets: password_external target: attacker: false]], ExternalEntity Plugin[assets: path, password //send the path to read the password incoming flows: "Read_cache.return_password_from_cache",]".
- storepassword.mydsl.corr.xmi:** A table showing suggested mappings between the Java code and the GravITY model. It lists various types such as "Type: org.eclipse.equinox.internal.security.storage.PasswordExt <> Asset: password" and "Type: org.eclipse.core.runtime.IPath <> Asset: path".
- Bottom Bar:** Includes tabs for Problems, Javadoc, Declaration, Search, Console, Progress, Properties, Error Log, GRaViTY SecDFD Mappings, and Call Hierarchy.

User Verification

Screenshot of the Eclipse IDE interface showing the verification process between code and models.

The left sidebar shows the Package Explorer with various projects and files, including `storepassword.mydsl` which is currently selected.

The central area displays two code snippets:

- Java Code (SecurePreferencesContainer.java):**

```

249 public String get(String key, String def, SecurePreferencesContainer container)
250     checkRemoved();
251     if (!hasKey(key))
252         return def;
253     String encryptedValue = internalGet(key);
254     if (encryptedValue == null)
255         return null;
256
257     CryptoData data;
258     String moduleID;
259     if (moduleID == null)
260         if (data.get(moduleID))
261             return Stor
262         return null;
263
264     PasswordExt pass;
265     if (passwordExt == null)
266         throw new S
267     try {
268         byte[] clea
269         return Stor
270     } catch (Illegal
271         throw new S
272     } catch (BadPad
273         throw new S
274     }
275
276     /**
277      * For internal use
278     */
279 }
280
281 /**
282  * For internal use
283 */

```
- Model (storepassword.mydsl):**

```

1 EDFD storepassword [
2     assets:
3         Asset password values: [H C], [H I]
4             source: Decrypt_file
5             targets: Plugin,
6             Asset system_encr_password values: [H C], [H I]
7                 source: Password_store
8                 targets: Decrypt_file,

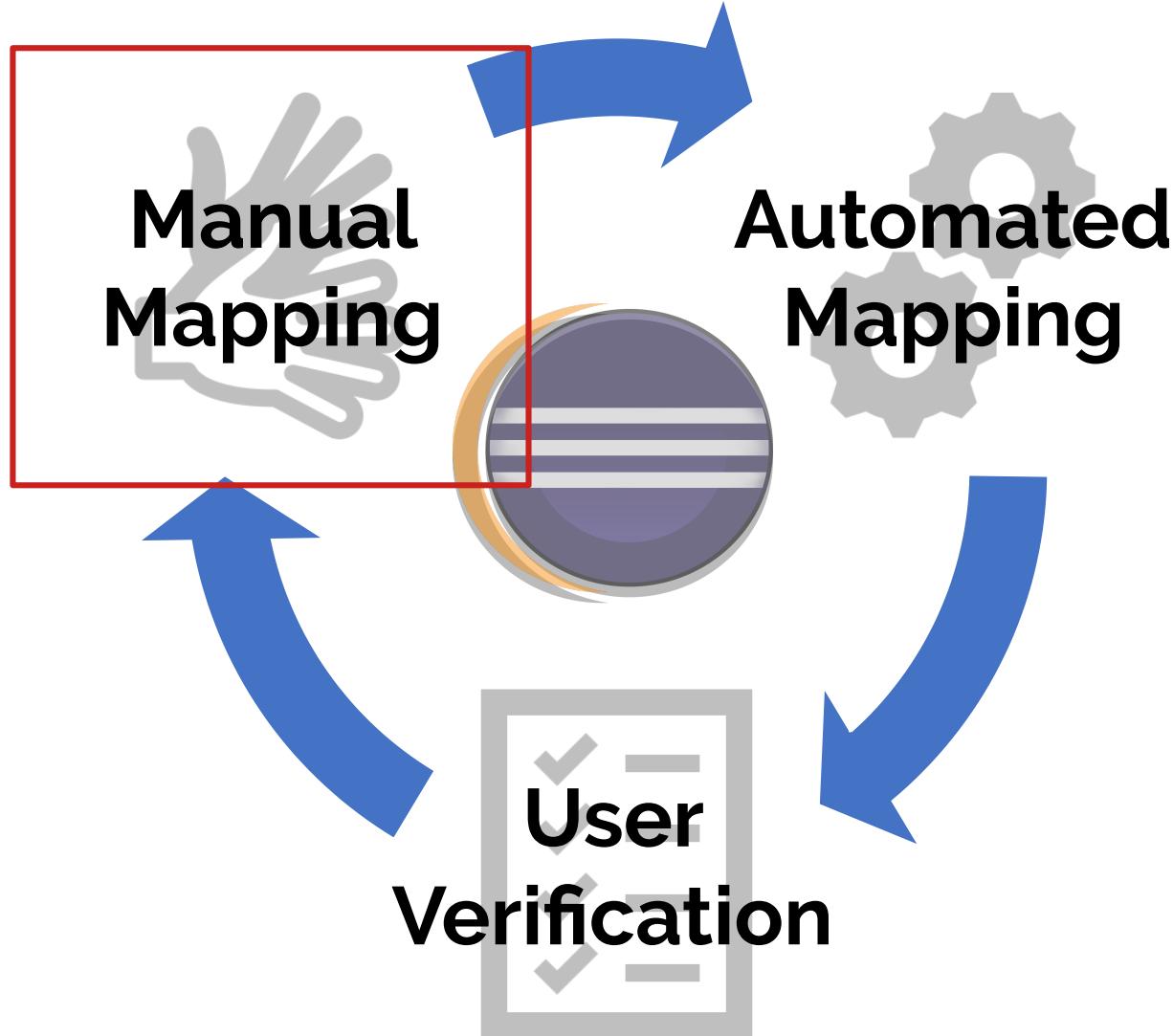
```

The bottom right shows the GRaViTY SecDFD Mappings view, listing suggested assets and processes.

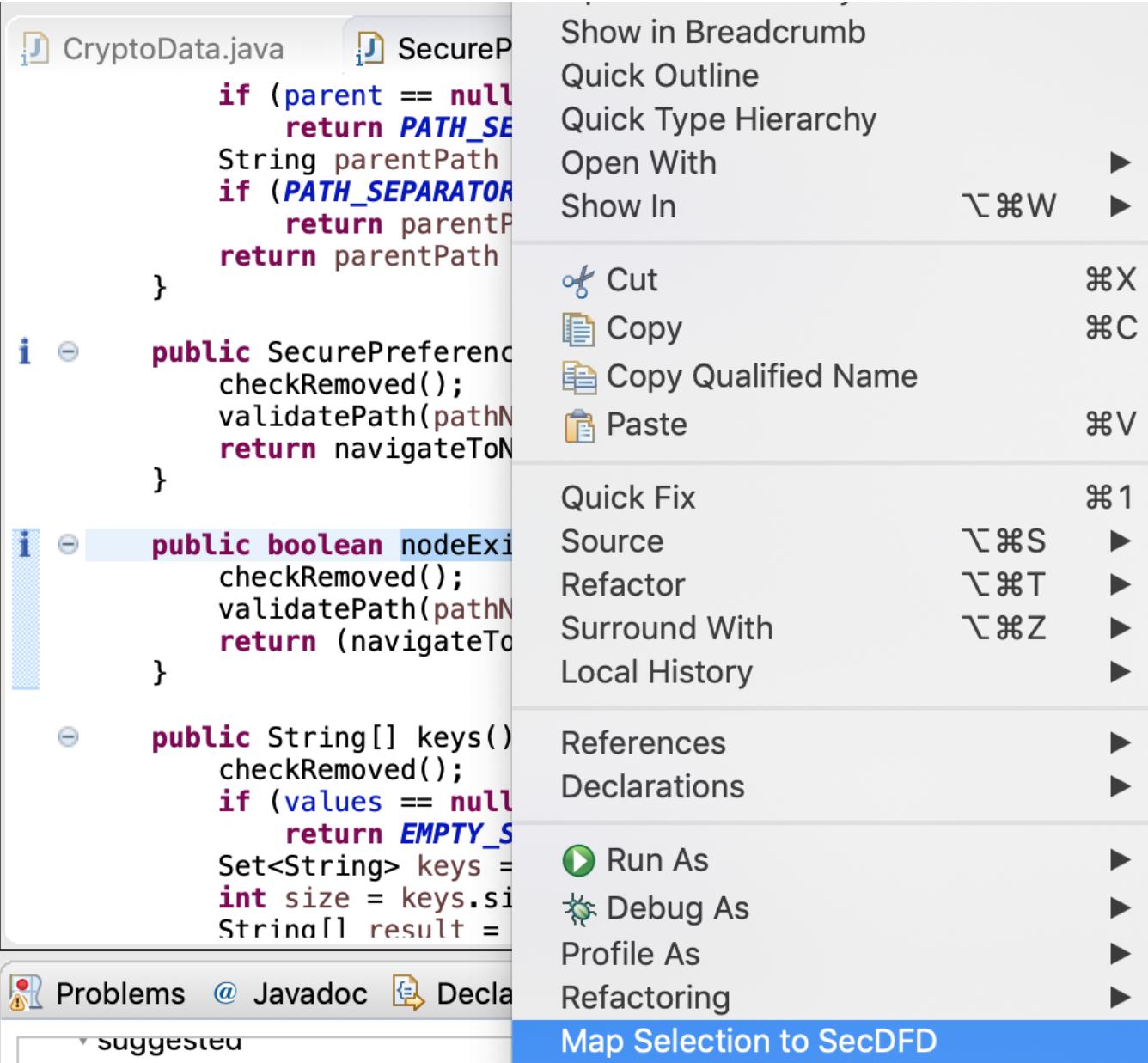
A context menu is open over the suggested assets, with the "accept" option highlighted in blue. The menu includes:

- reject
- accept
- + TRUE POSITIVE
- Team
- Compare With
- Replace With
- Show EClass information
- Show References
- Discovery
- Plug-in Tools
- eMoflon
- Browse Code in Text Editor

Mapping Generation: Semi-Automated Process



Manual Mapping



The screenshot shows an IDE interface with the following details:

- Left Panel:** Shows two files: `CryptoData.java` and `SecureP...`. The code for `SecureP...` includes methods like `if (parent == null)`, `String parentPath`, and `public SecurePreference(...)`.
- Right Panel:** A context menu is open over the code in `SecureP...`. The menu items include:
 - Show in Breadcrumb
 - Quick Outline
 - Quick Type Hierarchy
 - Open With
 - Show In ⌘W
 - Separator
 - Cut ⌘X
 - Copy ⌘C
 - Copy Qualified Name
 - Paste ⌘V
 - Separator
 - Quick Fix ⌘1
 - Source ⌘S
 - Refactor ⌘T
 - Surround With ⌘Z
 - Local History
 - Separator
 - References
 - Declarations
 - Separator
 - Run As
 - Debug As
 - Profile As
 - Refactoring
- Bottom Bar:** Shows tabs for `Problems`, `@ Javadoc`, and `Declar...`. The `Problems` tab has a red exclamation mark icon. The `Declar...` tab has a blue arrow icon.
- Bottom Right:** A blue bar with the text `Map Selection to SecDFD`.

Evaluation

- **RQ1.** What is the correctness of the automated mappings?
 - In terms of precision and recall
- **RQ2.** What is the impact of the user on the correctness of mappings?
 - How much is the recall increased in each iteration by
 - the user
 - the automated mappings

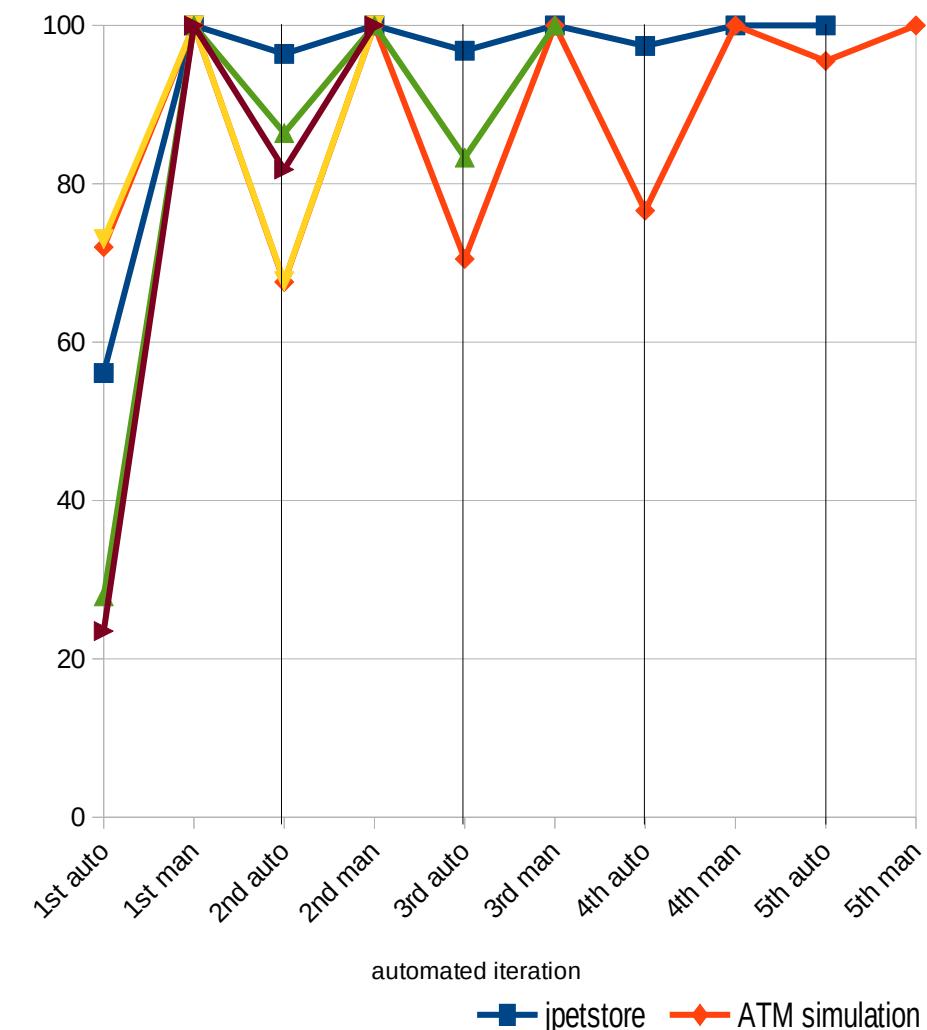
Evaluation – Testdata

- On 5 realistic Java projects
 - Reverse-engineered DFDs
 - Created expected mappings

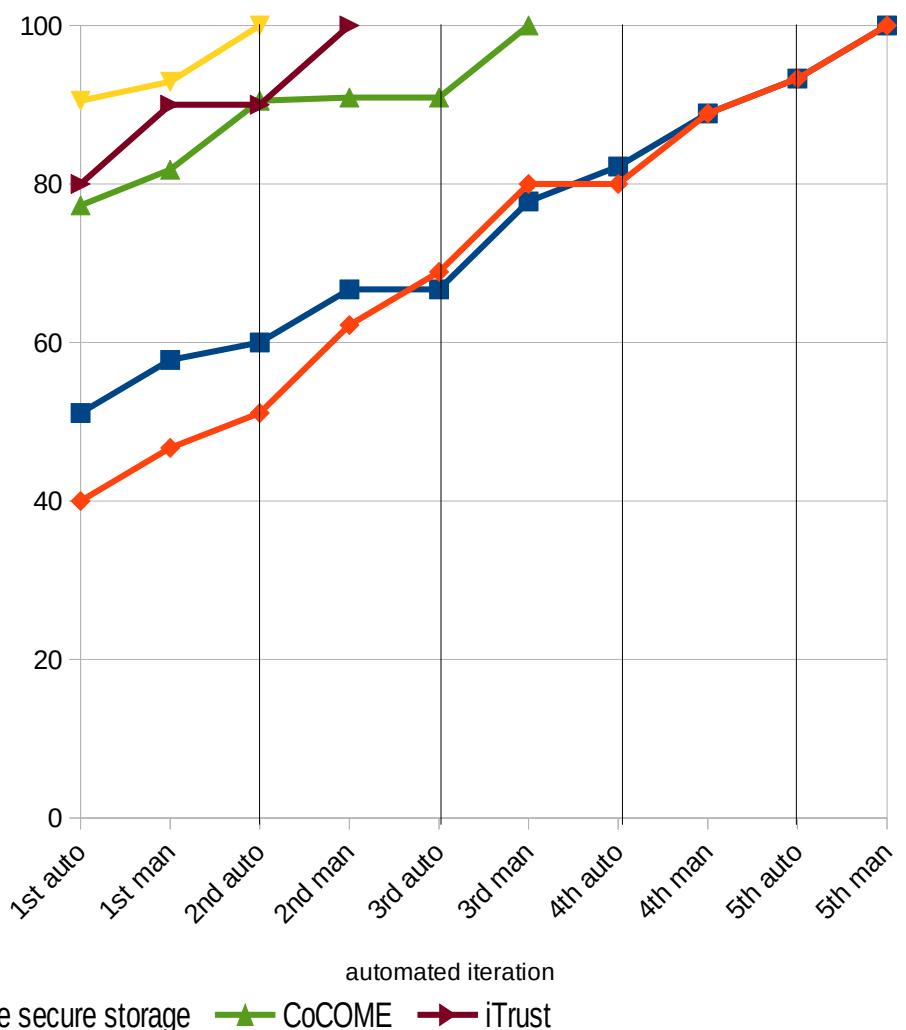
project	source code			DFD
	lloc	classes	methods	elements
jpetstore	1,221	17	277	47
ATM simulation	2,290	57	225	85
Eclipse secure storage	2,900	39	330	41
CoCoME	4,786	120	512	44
iTrust	28,133	423	3,691	31

Evaluation – RQ1

> Precision

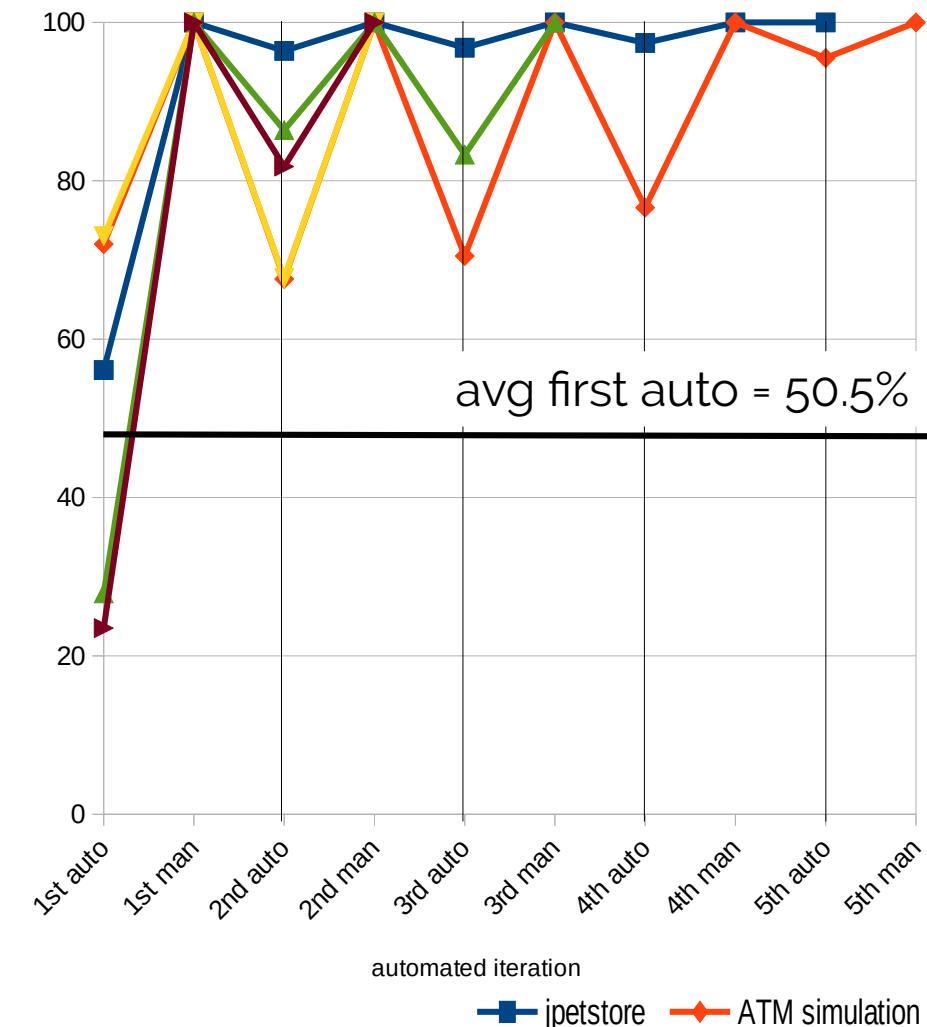


> Recall

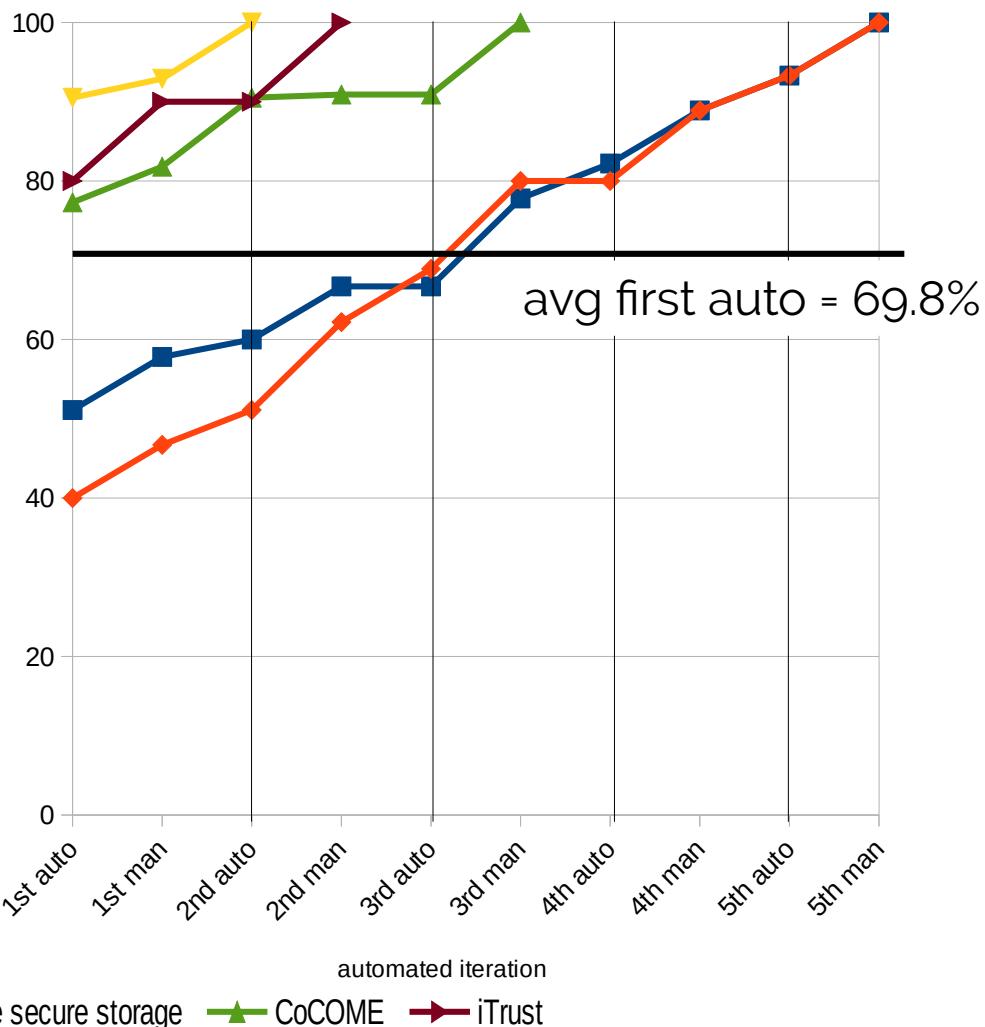


Evaluation – RQ1

> Precision

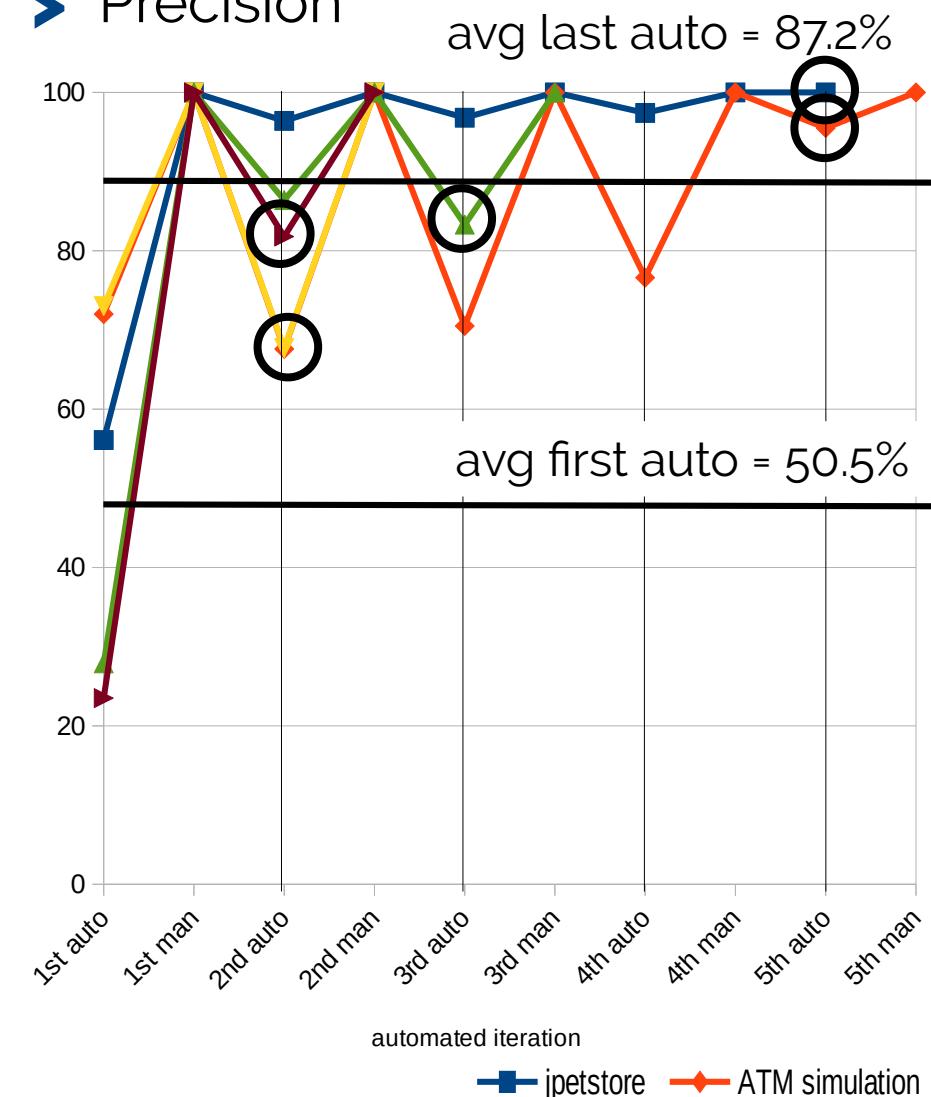


> Recall

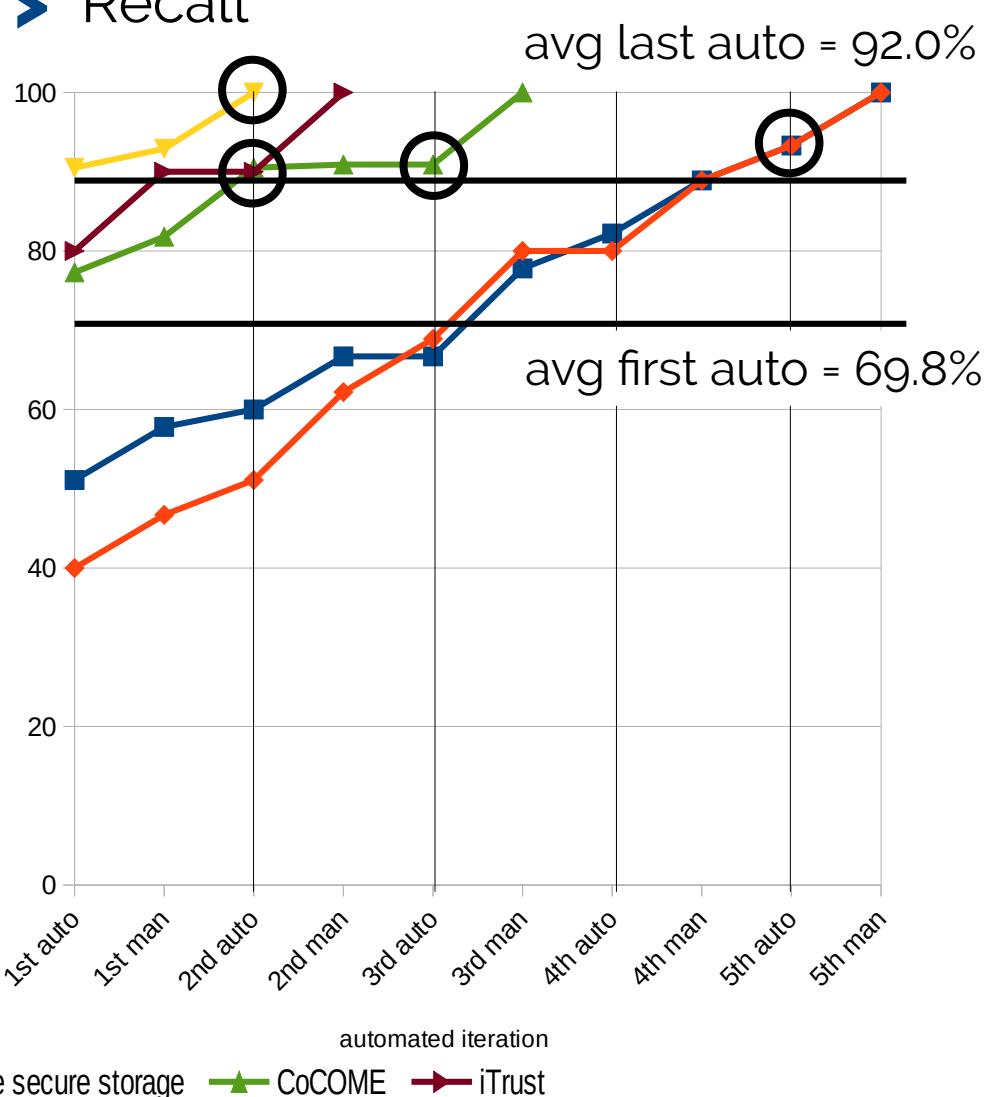


Evaluation – RQ1

Precision

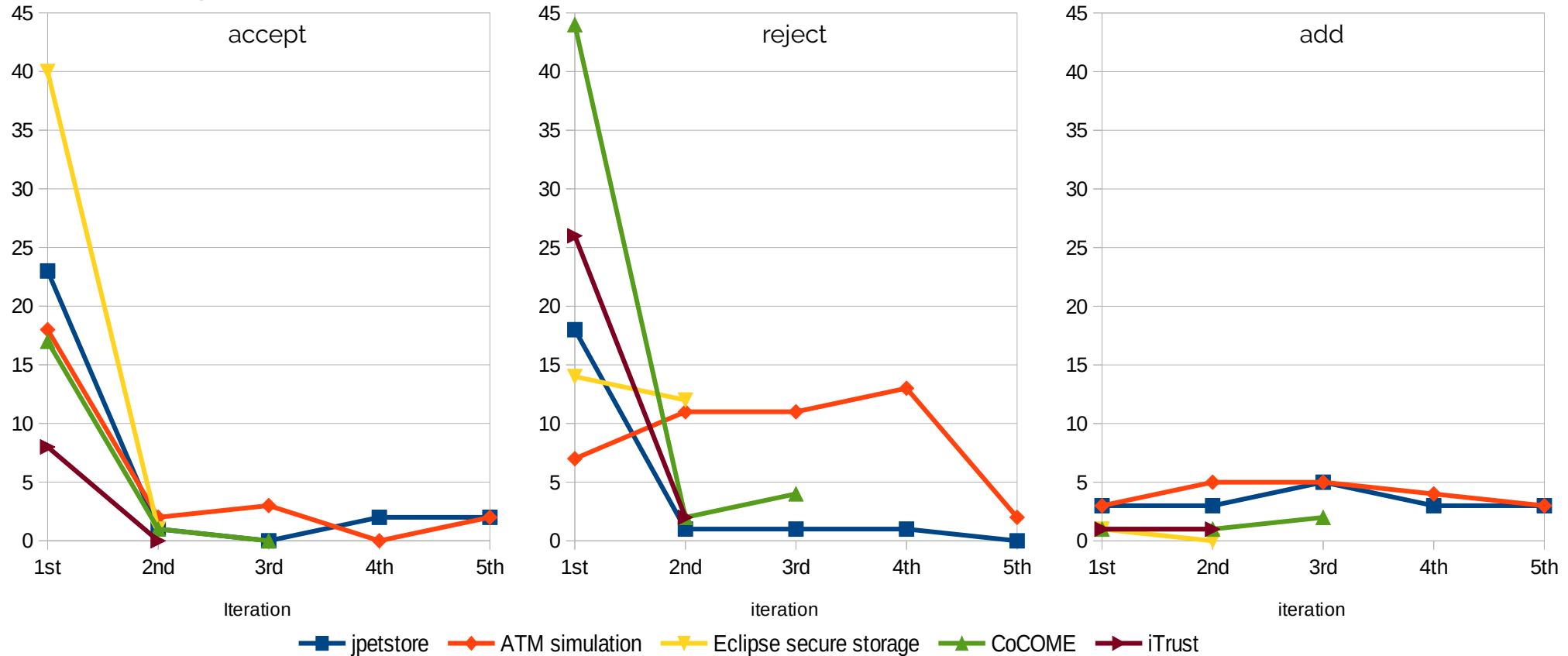


Recall



Evaluation – RQ2

- User actions per iteration:
 - Avg accepts: 7
 - Avg rejects: 9.6
 - Avg adds: 2.6
- Average user impact on recall
 - user: 7.9%
 - automated: 92.1%

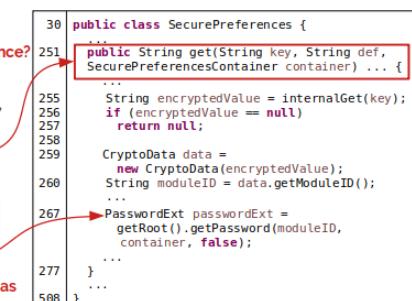


Conclusion

Compliance between Models & Code

According to the principle of **security by design**, the system's assets and threats have to be defined in the earliest phases of development.

Specification of the system (e.g. Eclipse Secure Storage) using a **Security Data Flow Diagram (SecDFD)** [2]

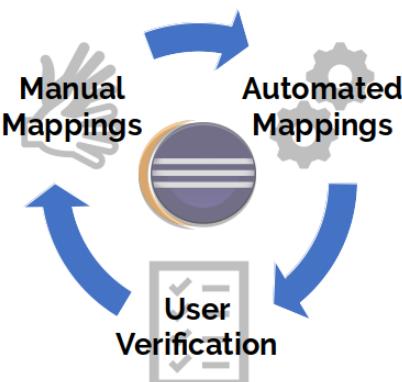


```

public class SecurePreferences {
    ...
    public String get(String key, String def,
                     SecurePreferencesContainer container) {
        ...
        String encryptedValue = internalGet(key);
        if (encryptedValue == null)
            return null;
        CryptoData data =
            new CryptoData(encryptedValue);
        String moduleID = data.getModuleID();
        ...
        PasswordExt passwordExt =
            getRoot().getPassword(moduleID,
                                  container, false);
        ...
    }
}
  
```

Sven Peldszus | Secure Data-Flow Compliance Checks between Models and Code based on Automated Mappings

Semi-automated Mappings

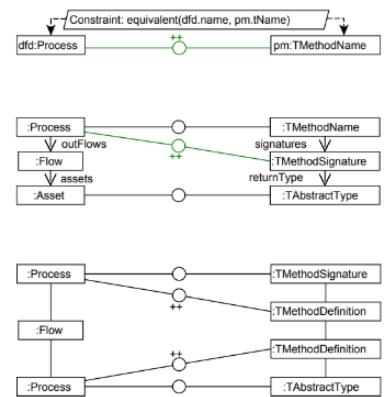


Sven Peldszus | Secure Data-Flow Compliance Checks between Models and Code based on Automated Mappings

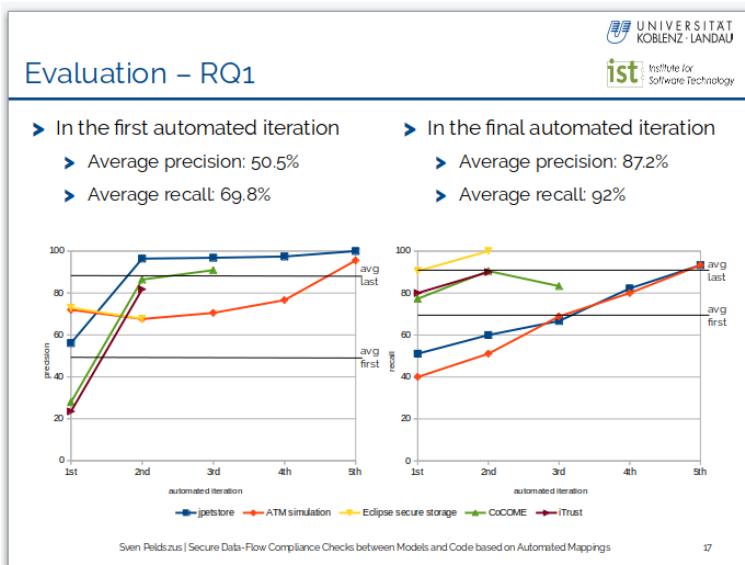
Mapping a DFD to the Program Model

Iterative mapping

- 1) Mapping of Names:
 - Using a Heuristic
 - Selecting all with a certainty higher than 70%
- 2) Mapping of Signatures:
 - Selecting all with at least one parameter or return type mapping
 - Certainty based on number of matches and name matching
- 3) Mapping of Definitions:
 - Certainty based on signature mappings



Sven Peldszus | Secure Data-Flow Compliance Checks between Models and Code based on Automated Mappings



Thank you!

Compliance between Models & Code

According to the principle of **security by design**, the system's assets and threats have to be defined in the earliest phases of development.

Specification of the system (e.g. Eclipse Secure Storage) using a **Security Data Flow Diagram (SecDFD)** [2]

```

public class SecurePreferences {
    ...
    public String get(String key, String def, SecurePreferencesContainer container) ... {
        ...
        String encryptedValue = internalGet(key);
        if (encryptedValue == null)
            return null;
        ...
        CryptoData data =
            new CryptoData(encryptedValue);
        String moduleID = data.getModuleID();
        ...
        PasswordExt passwordExt =
            getRoot().getPassword(moduleID, container, false);
        ...
    }
    ...
}
  
```

Sven Peldszus | Secure Data-Flow Compliance Checks between Models and Code based on Automated Mappings 5

Semi-automated Mappings

Sven Peldszus | Secure Data-Flow Compliance Checks between Models and Code based on Automated Mappings

Visit me at the poster session

Secure Data-Flow Compliance Checks between Models and Code based on Automated Mappings

Sven Peldszus¹, Katja Tuma², Daniel Strüber², Jan Jürjens^{3,4}, Riccardo Scandariato²
¹University of Koblenz-Landau ²University of Gothenburg and Chalmers University of Technology ³Fraunhofer IISI

CONTEXT
According to the principle of **security by design**, the system's assets and threats have to be defined in the earliest phases of development.

PROBLEM DESCRIPTION
> Does the implementation comply to the model?
> Do the designed security properties hold?
→ For checking the compliance we need a mapping between models and code

```

public class SecurePreferences {
    ...
    public String get(String key, String def, SecurePreferencesContainer container) ... {
        ...
        String encryptedValue = internalGet(key);
        if (encryptedValue == null)
            return null;
        ...
        CryptoData data =
            new CryptoData(encryptedValue);
        String moduleID = data.getModuleID();
        ...
        PasswordExt passwordExt =
            getRoot().getPassword(moduleID, container, false);
        ...
    }
    ...
}
  
```

Specification of the Eclipse Secure Storage using a **Security Data Flow Diagram (SecDFD)** [2]

SEMI-AUTOMATED APPROACH
Mappings between SecDFDs and program models for:
> Compliance checks > Transfer of security labels
> Convergence > Security metrics
> Divergence > Secure data flow analyses
> Absence > Runtime-monitoring

Automated Mapping of Elements

User Verification of Mappings and Manual Mapping of Elements integrated into the Eclipse IDE
> Accept Mapping → Taken as valid
> Reject Mapping → Never shown again

1) Automated Mapping of Elements
2) User Verification of Mappings
3) Manual Mapping of Elements

GET THE TOOL!
secdfd.gravity-tool.org

QR Code

Mapping a DFD to the Program Model

Iterative mapping

- 1) Mapping of Names:
 - Using a Heuristic
 - Selecting all with a certainty higher than 70%
- 2) Mapping of Signatures:
 - Selecting all with at least one parameter or return type mapping
 - Certainty based on number of matches and name matching
- 3) Mapping of Definitions:
 - Certainty based on signature mappings

Sven Peldszus | Secure Data-Flow Compliance Checks between Models and Code based on Automated Mappings 10

