Software Ecosystems as Networks
Advances on the FASTEN project

Paolo Boldi
Università degli Studi di Milano
Italy
The FASTEN Project

- Fine-Grained Analysis of Software Ecosystems as Networks
- Part of the EU H2020-ICT-2018-2020 Program
- Consortium
Why FASTEN?
Sharing through software libraries
Sharing through software libraries

- Internet made the **dream** of collaborative development a **reality**, by means of libraries that are made available:
Sharing through software libraries

- Internet made the **dream** of collaborative development a **reality**, by means of libraries that are made available:
  - on **repositories** (SourceForge, GitHub, BitBucket, …)
Internet made the **dream** of collaborative development a **reality**, by means of libraries that are made available:

- on *repositories* (SourceForge, GitHub, BitBucket, …)
- or *forges* (Maven, PyPi, CPAN, …)
Industrial revolution
at the harbour of software development
All trades, arts, and handiworks have gained by **division of labour**, namely, when, instead of one man doing everything, each confines himself to a certain kind of work distinct from others in the treatment it requires, so as to be able to perform it with greater facility and in the greatest perfection. Where the different kinds of work are not distinguished and divided, where everyone is a jack-of-all-trades, there manufactures remain still in the greatest barbarism.

Immanuel Kant

*Groundwork for the Metaphysics of Morals* (1785)
Dependency graphs
Dependency graphs

- Library+versions and their dependencies form (complex, huge) dependency networks
Dependency graphs

- Library+versions and their dependencies form (complex, huge) dependency networks
- Version constraints make these networks more complicated than simple graphs
Dependency graphs

- Library+versions and their dependencies form (complex, huge) dependency networks
- Version constraints make these networks more complicated than simple graphs
- Package manager will finally determine which version is chosen for each library
The dependency heaven
The dependency heaven

- Relying on an ecosystem of easy-to-use well written libraries made the dream of code reuse a reality
The dependency hell
The dependency hell

- A bug or security breach or legal issue concerning one single piece...
- ...can make the whole tower fall!
Recent dependency nightmares
Recent dependency nightmares

- The *leftpad* incident (2016): millions of websites affected
Recent dependency nightmares

- The leftpad incident (2016): millions of websites affected
- The Equifax breach (2017): costed 4B$
Ecosystems
Ecosystems

❖ Ecosystems grow at mind boggling speed
Ecosystems grow at mind boggling speed

JavaScript projects have an average of 80 (Zimmerman et al., 2019) transitive dependencies
Ecosystems grow at mind boggling speed

- JavaScript projects have an average of 80 (Zimmerman et al., 2019) transitive dependencies
- 50% of dependencies change in a 6-month time (Hejderup et al., 2019)
Ecosystems grow at mind boggling speed

- JavaScript projects have an average of 80 (Zimmerman et al., 2019) transitive dependencies
- 50% of dependencies change in a 6-month time (Hejderup et al., 2019)
- And deteriorate almost as rapidly
Ecosystems

- Ecosystems **grow** at mind boggling speed
  - JavaScript projects have an average of 80 (Zimmerman et al., 2019) transitive dependencies
  - 50% of dependencies change in a 6-month time (Hejderup et al., 2019)
- And **deteriorate** almost as rapidly
  - Existence of package bottlenecks (the removal on one single package can bring down almost 40% of the system)
Ecosystems

- Ecosystems grow at mind boggling speed
  - JavaScript projects have an average of 80 (Zimmerman et al., 2019) transitive dependencies
  - 50% of dependencies change in a 6-month time (Hejderup et al., 2019)
- And deteriorate almost as rapidly
  - Existence of package bottlenecks (the removal on one single package can bring down almost 40% of the system)
  - Rich get richer: few maintainers dominate most packages
Epidemics in dependency graphs

Lib D, vers 3.0 → Lib C, vers 1.5

Lib C, vers 1.5 → Lib A, vers 1.0, Lib B, vers 2.5
Epidemics in dependency graphs

Lib D, vers 3.0 → Lib C, vers 1.5 → Lib A, vers 1.0
Lib C, vers 1.5 → Lib B, vers 2.5

A vulnerability alert is issued about Lib D, vers 3.0
Epidemics in dependency graphs

All libraries in this graph are infected!

Lib D, vers 3.0

Lib C, vers 1.5

Lib A, vers 1.0

Lib B, vers 2.5

A vulnerability alert is issued about Lib D, vers 3.0
GitHub security alerts

But is this enough?
Isn’t this kind of tool enough?
Isn’t this kind of tool enough?

- In theory. But in practice:
Isn’t this kind of tool enough?

- In theory. But in practice:
  - Developers don’t update
Isn’t this kind of tool enough?

- In theory. But in practice:
  - Developers don’t update
  - → Vulnerabilities proliferate
Isn’t this kind of tool enough?

- In theory. But in practice:
  - Developers don’t update
  - → Vulnerabilities proliferate
- Why?
Isn’t this kind of tool enough?

- In theory. But in practice:
  - Developers don’t update
  - → Vulnerabilities proliferate

- Why?
  - Our tools are not sharp enough for what we want
Examples of what people want

<table>
<thead>
<tr>
<th>Update</th>
<th>Developers</th>
<th>Maintainers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does this outdated dependency really break my code?</td>
<td>How do I update without breaking too many of my important clients?</td>
<td></td>
</tr>
</tbody>
</table>

| Violations   | Am I violating anyone’s copyright? | How do I spot instances of my code being distributed without permission? |
Epidemics in dependency graphs
Epidemics in dependency graphs
Epidemics in dependency graphs

A vulnerability alert is issued about Lib D, vers 3.0, function f3
A vulnerability alert is issued about Lib D, vers 3.0, function f3
Epidemics in dependency graphs

A vulnerability alert is issued about Lib D, vers 3.0, function f3

Much more informative!
Epidemics in dependency graphs

A vulnerability alert is issued about Lib D, vers 3.0, function f3
Epidemics in dependency graphs

Avoid the cry wolf effect!

A vulnerability alert is issued about Lib D, vers 3.0, function f3
Examples
Examples

- Fully precise change impact analysis: “How many libraries are affected if I remove/modify a certain method/interface?”
Examples

❖ Fully precise change impact analysis: “How many libraries are affected if I remove/modify a certain method/interface?”

❖ Fully precise license compliance: “Is my library compliant with the licenses of the libraries that I depend from (directly or indirectly)? (e.g., am I linking any GPL code?)”
Examples

❖ Fully precise change impact analysis: “How many libraries are affected if I remove/modify a certain method/interface?”

❖ Fully precise license compliance: “Is my library compliant with the licenses of the libraries that I depend from (directly or indirectly)? (e.g., am I linking any GPL code?)”

❖ Fully precise risk profiling: “Does this vulnerability affect my code?”
Examples

❖ Fully precise change impact analysis: “How many libraries are affected if I remove/modify a certain method/interface?”

❖ Fully precise license compliance: “Is my library compliant with the licenses of the libraries that I depend from (directly or indirectly)? (e.g., am I linking any GPL code?)”

❖ Fully precise risk profiling: “Does this vulnerability affect my code?”

❖ Centrality analysis: “What methods/functions are more central within a given ecosystem? are there bottlenecks? critical points?”
The FASTEN toolchain
The FASTEN toolchain

- Project information
- Security alerts
- Repositories
The FASTEN toolchain
The FASTEN toolchain
The FASTEN toolchain

Project information

Security alerts

Repositories

FASTEN server

Call-graph construction

Data stream

publish

publish

publish
The FASTEN toolchain

- Project information
- Security alerts
- Repositories

Data stream

- Call-graph construction
- Storage layer

FASTEN server
The FASTEN toolchain

- Project information
- Security alerts
- Repositories

Data stream

- Call-graph construction
- Storage layer
- Analysis layer

FASTEN server
The FASTEN toolchain

- Project information
- Security alerts
- Repositories

FASTEN server

Data stream

- Call-graph construction
- Storage layer
- Analysis layer

REST Api

Publish

publish
The FASTEN toolchain

- Project information
- Security alerts
- Repositories

Data stream

- Call-graph construction
- Storage layer
- Analysis layer

REST API

Web UI

FASTEN server

Repositories

- Maven
- RubyGems

Security alerts

- CVE

Project information

- OWASP

publish

publish

publish
The FASTEN toolchain

- Project information
- Security alerts
- Repositories

FASTEN server

- Data stream
  - Call-graph construction
  - Storage layer
  - Analysis layer

REST API

Web UI

Continuous integration server
The FASTEN toolchain

Project information

Security alerts

Repositories

FASTEN server

Data stream

Call-graph construction
Storage layer
Analysis layer

REST API
Web UI

Continuous integration server
The FASTEN toolchain

- Project information
- Security alerts
- Repositories

Publish to:

- Data stream
- FASTEN server

Components:
- Call-graph construction
- Storage layer
- Analysis layer
- REST API
- Web UI

Integration servers:
- Continuous integration
- Developer
Preliminary results
Server-side highlights
Dataflow example: CG generation

1. Forge
2. New Package
3. Codefeedr/Custom Scraper
4. Package Release
5. Filtering
6. Call Graph
7. FASTEN DB
8. CG Generator
9. CG Stitching
10. Error Message
11. Package Release
## Universal function identifiers

How to uniquely reference a function in a global namespace?

<table>
<thead>
<tr>
<th>scheme</th>
<th>fasten://</th>
</tr>
</thead>
<tbody>
<tr>
<td>forge</td>
<td>/mvn</td>
</tr>
<tr>
<td>artifact</td>
<td>/org.slf4j.slf4j-api</td>
</tr>
<tr>
<td>version</td>
<td>/1.2.3</td>
</tr>
<tr>
<td>namespace</td>
<td>/org.slf4j.helpers</td>
</tr>
<tr>
<td>function</td>
<td>/BasicMarkerFactory.getDetachedMarker</td>
</tr>
<tr>
<td>argument(s)</td>
<td>(%2Fjava.lang%2FString)</td>
</tr>
<tr>
<td>return type</td>
<td>%2Forg.slf4j%2FMarker</td>
</tr>
</tbody>
</table>
# Universal function identifiers

How to uniquely reference a function in a global namespace?

<table>
<thead>
<tr>
<th>Scheme</th>
<th>fasten://</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forge</td>
<td>/mvn</td>
</tr>
<tr>
<td>Artifact</td>
<td>/org.slf4j.slf4j-api</td>
</tr>
<tr>
<td>Version</td>
<td>/1.2.3</td>
</tr>
<tr>
<td>Namespace</td>
<td>/org.slf4j.helpers</td>
</tr>
<tr>
<td>Function</td>
<td>/BasicMarkerFactory.getDetachedMarker</td>
</tr>
<tr>
<td>Argument(s)</td>
<td>(%2Fjava.lang%2FString)</td>
</tr>
<tr>
<td>Return Type</td>
<td>%2Forg.slf4j%2FMarker</td>
</tr>
</tbody>
</table>

**Generic format**
```
<scheme>://
<forge>/
<artifact>/
<version>/
<namespace>/
<function>/
<argument(s)>(%2F<language>%2F<type>)
<return type>%2F<language>%2F<type>
```

---

**Languages**
- Java
- Python
- C
Call graph transport

```json
{
  "product": "foo",
  "forge": "mvn",
  "depset": [
    {
      "product": "a",
      "forge": "mvn",
      "constraints": ["[1.2..1.5]", "[2.3..]"]
    },
    {
      "product": "b",
      "forge": "mvn",
      "constraints": ["[2.0.1]"]
    }
  ],
  "version": "3.10.0.7",
  "cha": {
    "/name.space/A": {
      "methods": {
        "0": "/name.space/A.A()%2Fjava.lang%2FVoidType",
        "1": "/name.space/A.g(%2Fjava.lang%2FString)%2Fjava.lang%2FInteger"
      },
      "superInterfaces": [ "/java.lang/Serializable" ],
      "sourceFile": "filename.java",
      "superClasses": [ "/java.lang/Object" ]
    }
  },
  "graph": {
    "internalCalls": [
      [ 0, 1 ]
    ],
    "externalCalls": [
      [ "1", "////their.package/TheirClass.method()Response", { "invokeinterface": "1" } ]
    ]
  }
}
```
Call graph transport

{
    "product": "foo",
    "forge": "mvn",
    "depset": [
        {
            "product": "a",
            "forge": "mvn",
            "constraints": ["[1.2..1.5]", "[2.3..]"
        },
        {
            "product": "b",
            "forge": "mvn",
            "constraints": ["[2.0.1]"
        }
    ],
    "version": "3.10.0.7",
    "cha": {
        "/name.space/A": {
            "methods": {
                "0": "/name.space/A.A()%2Fjava.lang%2FVoidType",
                "1": "/name.space/A.g(%2Fjava.lang%2FString)%2Fjava.lang%2FInteger"
            },
            "superInterfaces": [ "/java.lang/Serializable" ],
            "sourceFile": "filename.java",
            "superClasses": [ "/java.lang/Object" 
        },
    },
    "graph": {
        "internalCalls": [ [ 0, 1 ] ],
        "externalCalls": [ [ "1", "/\//their.package/TheirClass.method()Response", { "invokeinterface": "1" } ] ]
    }
}

Generic format +
Java
Python
C

Done
Language-dependent call graph generation
Language-dependent call graph generation

- **Java**: Based on tools from the OPAL project (stg-tud/opal)
- **Python**: New static analysis tool: *PyCG* (*Submitted ICSE 2020*)
- **C**: CScount for static call graphs; gprof, callgrind for dynamic calls
## Current CG results

<table>
<thead>
<tr>
<th>Language / Ecosystem</th>
<th>Total Packages</th>
<th>Results</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C / Debian Buster</td>
<td>7.380 (757 analyzed) *</td>
<td>Packages: 531</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nodes: 491.721</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Edges: 579.253</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Success Rate: 70%</td>
<td></td>
</tr>
<tr>
<td>Java / Maven</td>
<td>2.7M artifacts</td>
<td>Packages: 2.4M</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nodes: ~5B+</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Edges: ~56B+</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Success Rate: 89.13%</td>
<td></td>
</tr>
<tr>
<td>Python / PyPI</td>
<td>~740 K</td>
<td>Packages: ~520K</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nodes: ~211M</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Edges: ~310M</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Success Rate: 70%</td>
<td></td>
</tr>
</tbody>
</table>

* Technical issues prohibited us from downloading the rest of the packages.
Call graph stitching

How to scale call graph processing to $10^6$ package versions?
Call graph stitching

How to scale call graph processing to $10^6$ package versions?

- **Idea:** Decouple package resolution from call graph generation
Call graph stitching

How to scale call graph processing to $10^6$ package versions?

- **Idea:** Decouple package resolution from call graph generation
- Build and store call graphs per package version, incl.:
Call graph stitching

How to scale call graph processing to $10^6$ package versions?

- **Idea:** Decouple package resolution from call graph generation
- Build and store call graphs per package version, incl.:
  - unresolved calls

In progress
Call graph stitching

How to scale call graph processing to $10^6$ package versions?

- **Idea:** Decouple package resolution from call graph generation
- Build and store call graphs per package version, incl.:
  - unresolved calls
  - class hierarchies (Java, Python)
Call graph stitching

How to scale call graph processing to $10^6$ package versions?

- **Idea:** Decouple package resolution from call graph generation
- Build and store call graphs per package version, incl.:
  - unresolved calls
  - class hierarchies (Java, Python)
- **Call graph stitching:** Resolve unresolved calls given a dependency tree

In progress
The database schema
Examples of queries:

largest packages (# of functions)

```sql
select p.package_name, pv.version, count(*)
from package_versions pv
    join packages p on pv.package_id = p.id
    join modules m on m.package_version_id = pv.id
    join callables c on c.module_id = m.id
group by p.package_name, pv.version
order by count(*) desc
limit 10;
```

<table>
<thead>
<tr>
<th>package_name</th>
<th>version</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.bouncycastle:bcprov-jdk15on</td>
<td>1.54</td>
<td>16912</td>
</tr>
<tr>
<td>com.google.guava:guava</td>
<td>20.0</td>
<td>13956</td>
</tr>
<tr>
<td>xalan:xalan</td>
<td>2.7.2</td>
<td>13058</td>
</tr>
<tr>
<td>org.apache.pdfbox:pdfbox</td>
<td>2.0.8</td>
<td>6727</td>
</tr>
<tr>
<td>external_callables_library</td>
<td>0.0.1</td>
<td>5457</td>
</tr>
<tr>
<td>org.apache.santuario:xmlsec</td>
<td>2.0.9</td>
<td>4783</td>
</tr>
<tr>
<td>org.apache.santuario:xmlsec</td>
<td>2.0.8</td>
<td>4780</td>
</tr>
<tr>
<td>org.apache.commons:commons-collections4</td>
<td>4.1</td>
<td>4607</td>
</tr>
<tr>
<td>org.apache.commons:commons-lang3</td>
<td>3.6</td>
<td>3432</td>
</tr>
<tr>
<td>org.apache.httpcomponents:httpclient</td>
<td>4.5.3</td>
<td>3024</td>
</tr>
</tbody>
</table>

(10 rows)
Examples of queries:
Packages depending on vulnerable package

```sql
SELECT package_version_id, p.package_name, pv.version
FROM dependencies d
JOIN package_versions pv ON pv.id = d.package_version_id
JOIN packages p ON p.id = pv.package_id
WHERE d.dependency_id =
  (SELECT id
   FROM packages
   WHERE package_name = 'com.google.guava:guava')
  AND '20.0' = ANY(d.version_range);
```

<table>
<thead>
<tr>
<th>package_version_id</th>
<th>package_name</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>org.digidoc4j.dss:dss-utils-google-guava</td>
<td>5.0.d4j.5</td>
</tr>
<tr>
<td>41</td>
<td>org.digidoc4j.dss:dss-utils-google-guava</td>
<td>5.0.d4j.4</td>
</tr>
<tr>
<td>81</td>
<td>org.digidoc4j:digidoc4j</td>
<td>1.0.8.beta.2</td>
</tr>
<tr>
<td>107</td>
<td>org.digidoc4j:digidoc4j</td>
<td>1.0.7.beta.2</td>
</tr>
<tr>
<td>119</td>
<td>org.digidoc4j:digidoc4j</td>
<td>1.0.7.2</td>
</tr>
<tr>
<td>133</td>
<td>org.digidoc4j:digidoc4j</td>
<td>1.0.7.1</td>
</tr>
<tr>
<td>156</td>
<td>org.digidoc4j.dss:dss-utils-google-guava</td>
<td>5.1.d4j.5</td>
</tr>
<tr>
<td>142</td>
<td>org.digidoc4j.dss:dss-utils-google-guava</td>
<td>5.0.d4j.3</td>
</tr>
</tbody>
</table>
Graph analytics
(results shown refer to Java CG’s)
Graph analytics
(results shown refer to Java CG’s)

- Graph stored using WebGraph (UMIL)
Graph analytics
(results shown refer to Java CG’s)

- Graph stored using WebGraph (UMIL)
- For 1.1M graphs (2.3B nodes, 18B edges):
Graph analytics
(results shown refer to Java CG’s)

- Graph stored using WebGraph (UMIL)
- For 1.1M graphs (2.3B nodes, 18B edges):
  - 3.6 bits per edge, plus global ID storage for each node (9.0 bits per edge overall)
Graph analytics (results shown refer to Java CG’s)

- Graph stored using WebGraph (UMIL)
- For 1.1M graphs (2.3B nodes, 18B edges):
  - 3.6 bits per edge, plus global ID storage for each node (9.0 bits per edge overall)
  - DB size: 38GB → we can fit the whole of Maven in RAM
Graph storage

Compression results

Deflation

Graph size

Graph size: 10 100 1000 10000 100000 1 \times 10^6

Deflation: 100% 50% 25% 10%
Vulnerability Plugin

- Gathering vulnerability information (at package and callable level)
- A normalized Vulnerability Object definition is injected in the metadata database
- Normalization is needed to smooth out the different sources of information
- The plugin continuously pulls updates for new information and keeps storing the results
Analysis plug-ins

RAPID: Risk Analysis and Propagation Inspection for Security and Maintainability risks

- On the server side (to enrich the metadata DB):
  - Plugin for code maintainability analysis:
    V1 deployed, processed 126K Maven coordinates to date
  - Plugin for security vulnerability propagation

- On the client side:
  - A user application to model and present risks
License and Compliance analysis

- QMSTR Plugin consists of 3 steps:
  1. Use the CG generator to gather information about all the generated artifacts that will be distributed together with the source code
  2. Execution of static analysis tools that augment the build graph with license and compliance metadata
  3. Generation of a report with package's relevant license and authorship metadata that is finally distributed
Client-side highlights
REST API

- Implementation of endpoints to expose canned queries from the metadata database

- In development:
  - Full DB entity support
  - Custom extension points
Use cases

❖ Endocode
  ❖ Endocode developed a license-compliance solution, called Quartermaster
  ❖ They are integrating FASTEN to improve the precision of their compliance offering
❖ SIG
  ❖ Integration of FASTEN in BetterCodeHub, their GitHub-connected code quality monitoring product
❖ XWiki
  ❖ Risk validation in the dependencies at Maven build time
  ❖ Risk validation in the installed extensions of an XWiki instance
  ❖ Filter out available compatible extensions for an XWiki instance
  ❖ Discoverability of XWiki components in available extensions
Future timeline
The future

- **End 2020**: REST API, first full version of knowledge base, CG enrichment, build graph integration, first public announcement
- **Q1 2021**: Impact analysis, integration with MVN / PyPI; first external user
- **Q2 2021**: Industrial use cases integrated; first external adoption
- **Q3 2021**: Licensing and security fully integrated; Data-driven API evolution
- **Q4 2021**: Project finished; external integrations
- **Q1 2022**: FASTEN 2?
Network analysis will be the next step for the future of software development
Network analysis will be the next step for the future of software development
Questions?

Paolo Boldi
Università degli Studi di Milano
Italy
paolo.boldi@unimi.it