WCRE 2013
Koblenz, Germany

Find Your Library Experts

Cédric Teyton, Jean-Rémy Falleri,
Floréal Morandat, Xavier Blanc

Software Engineering (http://se.labri.fr)
*LaBRI, Université de **Bordeaux**, France*

Libraries & Expertise

- Libraries are heavily used by modern software
  - → Dozens of libraries, large range of applications
- Libraries experts are needed to :
  - → Assist developers when using a library
  - → Ensure maintenance of libraries (update or migrate ?)
  - → Resolve bugs related to a library

How to find library experts ?

Find Library Experts : Problem

- Self-evaluation ? Peers reviews ? Ok, but :

  → Time-consuming

  → Subjective, based on people opinion
  - evaluations are not consistent between each other

  → Hard to keep up-to-date and to automatize
  - what if I now tame a new library ?

Find Library Experts : Challenge

- We want a solution :

  → Automatic : experts are identified and ranked from source code analysis

  → Objective : experts are identified from *“what they really do”*

Find Library Experts : Road Map

- How to extract library expertise ?
- How to measure library expertise ?
- How to define “*more expert than ?*” and compare expertise ?

Libraries

- A library :
  - → A name (ex: JUnit)
  - → A version (ex : 3.8.1)
  - → A list of symbols (ex : org.junit.AssertTrue(boolean))
- Symbols are public functions, fields accessible through a well-defined API
- Developers ↔ Library ?

A developers knows a symbol once she used it

(1) Expertise Extraction : Problem

- Libraries usage :
  - → Is scattered [1], minor part source files contents
  - → Ex : Google Guava in Apache HBase* project
    - used in 228 Java files
    - usage : 492 LOC out of 113,000 LOC (0.004 %)
- Can not rely on coarse-grained analysis
- Touch file ≠ “know” its content !

[1] Bauer et al, Understanding API Usage to Support Informed Decision Making in Software Maintenance. CSMR 2012
** https://github.com/apache/hbase*

(1) Expertise extraction

- Fine-grained analysis of source code contributions
  - → Detection of added parts of the code using diff
  - → Search symbols in a pre-built symbols index

- Exemple : commit from Alice

```
import java.util.logging.Logger;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class StudyScheduler {

        private final static Logger LOGGER = Logger.getLogger("fr.labri.harmony.scheduler");
        private static Logger LOGGER;
        public StudyScheduler(SchedulerConfiguration schedulerConfiguration) {
                LOGGER = LoggerFactory.getLogger(getClass());
                this.schedulerConfiguration = schedulerConfiguration;
        }
```

1 symbol used

Developer	Library	Version (opt.)	Symbol
Alice	slf4j	?	LoggerFactory.getLogger(Class)

(2) Measure Library expertise

- Library expertise targets a library and optionally two filters on versions and symbols
  - → (library, [version], [symbol])
  - → Ex : (junit), (junit,3.8.1), (junit,3.8.1,org.junit.runners.*)

- Measure : ratio of knowledge of symbols for a library expertise definition
  - → A real value between 0 and 1
  - → 0 = no expertise, I = complete expertise

(3) Compare Expertise

- Compare developers for one or more expertise
- Ex : rank experts for the 3 library expertise {(junit,3.8.1), (guava), (log4j)} :

1. Get for all the developers the 3 related expertise scores { i,j,k } with 0 <= i,j,k <= 1
2. Euclidean distance computation to a virtual reference point (1,1,1) **R** that represents a complete expertise
3. The closest to **R**, the highest ranked-expert

Play With Expertise

- A prototype LibTic and a user-friendly language to manipulate our model
  → ex : (junit,3.8.1,org.junit.runners.*) and (guava)

- Few keywords :
  → **Who** junit = 3.8.1 {org.junit.runners.*} guava
  (returns developers with a score value)
  → **How** junit = 3.8.1 {org.junit.runners.*} guava
  (expends the list of symbols per constraint)

Experiments : Setup

- 6330 Java projects from Github managed under Maven
  - → Dependencies retrieval + versions information
  - → Allows for symbol index construction
  - → Each project revision analyzed
- 3705 developers, 1,026 libraries and 51,585 symbols
- Purposes :
  1) Verify that we find consistent results
  2) Show interest of LibTic in a software project context

Experiments (Experts search)

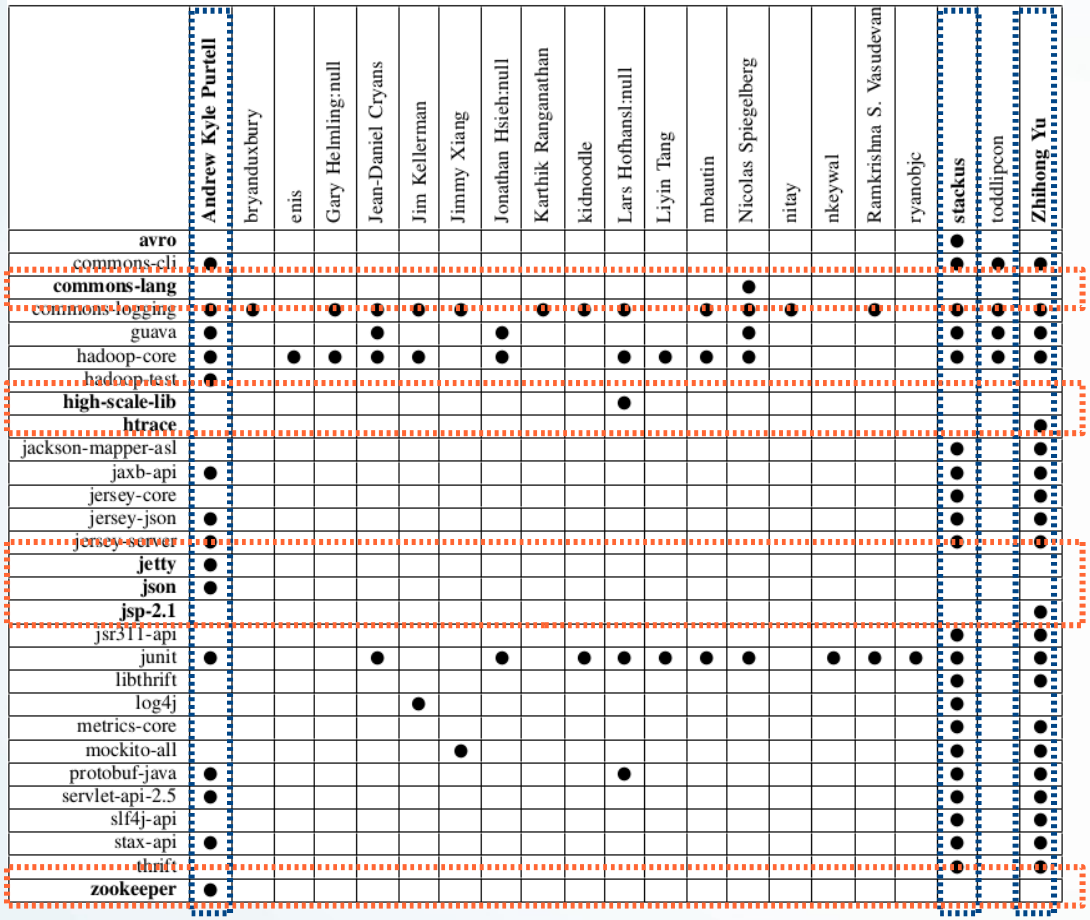
1) Select top-2 experts of 3 libraries and 1 pair of libraries
   → 8 developers contacted to confirm their expertise

2) Results
   → 4 confirmed their strong expertise
   → 3 did not answer, but good confidence based on their Github activities
   → 1 did not answer and no information about her

→ Consistent and valid data
→ Experts identification is possible

Experiments (Project Management)

- Assessing Library Knowledge : coverage matrix



Project : Apache HBase

→ Identify critical resources

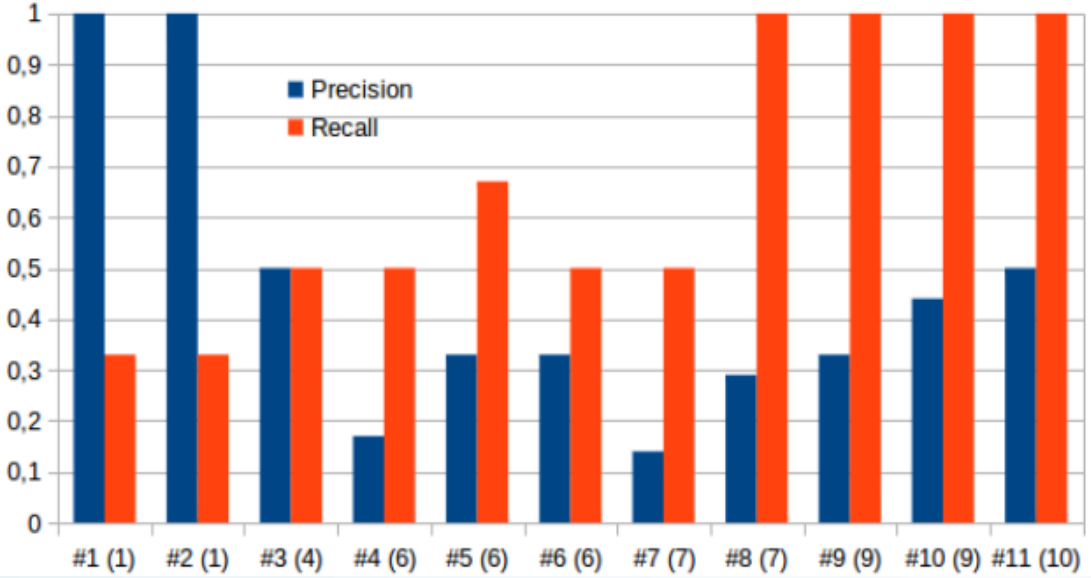
→ Critical human resource : if she leaves, it will impact the level of expertise of the team

→ Critical technological resource : too few developers know a library. Should some devs be trained with it ?

→ Anticipate needs and risks

Experiments (Task Recommendation)

- 11 issues (from 2010 to 2012) related to guava library
- Collected the devs involved in the resolution process
- What if all guava experts had been notified ?



#BugId (known Guava experts)

→ compute precision/recall against the true data

→ just an insight, does not prove the bug triaging aspect

Summary

- Need to assess 3rd-party libraries expertise
  - → Hard to identify and to maintain
- We propose an automatic approach to :
  - → Measure developer library expertise
  - → Rank experts for one or several expertise
- What's next ?
  - → Overcome precision issues
  - → Strengthen bug triaging part
  - → Extend to more technological resources