TUM

TECHNISCHE UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK

Using Network Analysis for Recommendation of Central Software Classes

Daniela Steidl

TUM-I127

Using Network Analysis for Recommendation of Central Software Classes

Daniela Steidl
Technische Universität München
Garching b. München, Germany
Email: steidl@in.tum.de

***Abstract*—Program comprehension is a complex task, especially for large software systems. Understanding an unknown system requires a significant amount of time. To speed up the learning process, developers focus on understanding central classes first. If other developers are available, they usually suggest which classes should be read at the beginning. In the absence of this knowledge, an independent algorithm is needed to measure the centrality of a class and give a recommendation for the developer. This paper presents an approach to retrieve central classes by using network analysis on the dependency graph of the system. An empirical study on open source projects evaluates the results of our algorithm based on a survey among the system's developers.**

I. Introduction

In many situations a new developer is asked to become familiar with an unknown software system, in order to either further develop the software, generate code covering test cases or review the code for quality management purposes. Therefore, the shorter the training period of the new developer, the lower the time and costs. To quickly understand the system, the ordering in which software artifacts are read is crucial. Usually, developers focus on understanding the central classes first to achieve a steep learning curve. We assume this simplifies the learning process, reduces its duration and therefore also decreases the cost of development.

However, for a new developer it is not obvious which classes are the most important ones. If other developers of the system are available, they recommend certain classes to focus on first. Nevertheless, they are biased towards the subset of classes they frequently work with and therefore they do not have an objective overview of the entire system. Instead of relying on the availability of other developers, this paper proposes an algorithm which determines the most important classes of the system based on its dependency graph.

In the field of network analysis, a couple of metrics have been proposed to define the importance of a node in a graph. However, how well these metrics suit the concept of centrality in the context of large software systems is an open research question. In this paper we evaluate the use of centrality indices on dependency graphs to measure the centrality of a software class.

Research problem. The purpose of the paper is to calculate an importance ranking among software classes to determine the centrality of a class. The top classes of the ranking serve as a recommendation for a new developer to focus on them first in order to quickly understand the rest of the system. Ideally, the ranking should provide an optimal recommendation. However, the optimality of an importance ranking is not clearly defined and subjective by nature. We measure optimality by evaluating the results based on the opinion of the developers.

Contribution. In this paper, the importance or centrality of a class is measured based on its dependencies on other classes. Appliyng network analysis on the dependency graph of the system, we experiment with a variety of network metrics which define centrality in different ways. An empirical study compares the results achieved by combining various centrality indices with different kinds of dependency graphs. The study comprises four open source projects, which are stand-alone applications from different contexts and written in Java. The size of the projects was about 160.000 lines of code on average. For each project the developers named the most important classes of their system. The individual rankings of the developers were used to evaluate the results of the centrality indices.

II. Related Work

Network analysis has been used in software engineering before. This section gives an overview of its previous applications.

In [1], the authors use network analysis on dependency graphs in order to predict defects in software systems. With the help of an empirical study they show that centrality measurements can successfully find central, and therefore critical, escrow binaries. With a recall of 60 percent on the Windows Server 2003, using closeness centrality retrieves twice as many critical binaries as previous complexity metrics. For most of their network measures the authors observe significant correlations, most of them being positive and moderate. Compared to this paper, we evaluate the use of similar centrality indices on dependency graphs. However, we interpret the ranking determined by an index to estimate the importance of a class, not its defects.

The authors of [2] propose a recommendation system for software testers. Given a finite amount of resources, the system recommends the tester to focus on critical classes first. The importance of a class is defined on a two-dimensional grid based on evolution cost (x-Axis) and PageRank (y-Axis), [3].

Classes are critical if they have a high PageRank value and high evolution costs. Although there is no empirical evidence of a correlation between error proneness and criticality of a class, the authors claim that classes identified by their approach play an important role and should be tested thoroughly. For the purpose of our study, we neglect evolution costs, but evaluate PageRank as one possible centrality index.

Pich et al. use network analysis of the dependency graph, with the goal of visualizing a software system, [4]. In their visualization tool, the vertical coordinate refers to the importance of a system component measured with the PageRank value. The horizontal coordinates describe similarity of objects based on shortest-path-measurements. Similar objects are placed close to each other, dissimilar objects are separated.

This section showed that network analysis on dependency graphs is commonly used among software engineers. However, there is no empirical evidence of which centrality index is mostly related to the importance of a class as determined by the software developers.

III. Terms & Definitions

Dependency graph. A dependency graph is a graph $G = (V, E)$ where the vertices V represent the interfaces/classes of the system. If the graph is directed, an edge $e = (v_1, v_2)$ connects vertex v_1 to vertex v_2, if v_1 depends on v_2. If the graph is undirected, an edge $e = \{v_1, v_2\}$ connects vertices v_1 and v_2, if v_1 depends on v_2 or vice versa.

Recommendation set. A recommendation set of the algorithm is the set containing the classes with the highest centrality values. For example, we will consider the top ten recommendation set, which includes the top ten classes of the ranking as produced by the algorithm.

(Recommendation) precision. The (recommendation) precision denotes the fraction of *correct* recommendations. A recommendation of the recommendation set is correct if it was listed by at least one of the developers participating in the survey.

IV. Approach

This section describes the approach of ranking classes of a software system according to their importance. The top classes of the ranking will then serve as a recommendation for a new developer to focus on them first in order to quickly understand the rest of the system.

In our approach, a class of a software system is considered to be important/central if many other classes depend on it. Dependency relations between classes are captured in the dependency graph of a system. Among network analysis, a variety of centrality indices are commonly used and constitute metrics for the importance of a single node within a graph. Computing a centrality index results in an importance ranking among all classes of the system. Thereby, using different indices leads to different results. Furthermore, these rankings also depend on the dependency types used for creating the graph. Different information such as data dependencies or call dependencies can be included.

The approach mainly consists of two phases: First, the dependency graph is extracted (IV-A). Second, the algorithm calculates a centrality index for each node of the graph (IV-B) and determines the recommendation set.

We implemented the approach with the open source quality analysis framework ConQAT[1]. The current implementation takes the source and byte code of software systems written in Java as an input.

A. Design of Dependency Graph

In a first step, the algorithm extracts the dependency graph of the system. We distinguish between different kinds of dependencies as follows: An edge $e = (v_1, v_2)$ represents a dependency iff one of the following statements holds

- v_1 implements/extends the interface/class v_2 (**Inheritance dependency**)
- v_1 has a field of type v_2 (**Field dependency**)
- v_1 calls a method of v_2 (**Method dependency**)
- a method of v_1 returns an object of type v_2 (**Return dependency**)
- a method of v_2 takes an object of v_1 as a parameter (**Parameter dependency**)

It is not obvious if all dependency-edges should be included in the graph or if a higher recommendation precision is achieved by including only a subset. An empirical study will answer this question and determine the edge set of the dependency graph (Section V).

In general, each node corresponds to exactly one interface or class. However, we sometimes merge interfaces with their implementation: The decision when to combine nodes is based on the inheritance tree of the graph, where an edge $e = (v_1, v_2)$ indicates that v_1 implements v_2. Interface I_A is merged with its implementation A, iff A is the only child of I_A in the inheritance tree which does not have any children itself. This means that interface I_A is implemented by only one single class A. However, it could be implemented by more interfaces I_B, I_C, which, in turn, have their own implementation classes. We merge interface I_A with its single implementation A because of the following reason: If class A is used within the source code, then only its interface I_A will occur in any dependency. This is due to the purpose of an interface to hide the details of its implementing class. Hence interface I_A will have many incoming dependency edges. The outgoing edges, however, belong to class A, because only the concrete class makes calls and references to other classes of the system. When A is the single implementation of I_A, the interface can be identified with its implementation and therefore we merge both nodes in the dependency graph.

B. Centrality indices

In the second step, the algorithm calculates a centrality index for the given dependency graph. Over the years, re-

[1] http://www.conqat.org/

searchers proposed a variety of different centrality measurements. For the purpose of this paper, we evaluate the results of using common measurements such as betweenness centrality [5], PageRank [3], PageRank with priors [6], HITS [7], HITS with priors [6], and Markov [6]. Furthermore we introduce a hierarchical flow model.

Betweenness. The shortest-path betweenness centrality of a node v belongs to the category of stress centralities and is calculated as

$$c_B(v) = \sum_{s \neq v \in V} \sum_{t \neq v \in V} \delta_{st}(v) \ , \tag{1}$$

where $\delta_{st}(v)$ denotes the fraction of shortest paths between s and t that contains v. The betweenness centrality of a node measures the control over communication between others.

PageRank. The PageRank algorithm is a feedback centrality, so the score of one node depends on the number and scores of its neighbours. The PageRank algorithm is based on the random-surfer-model [5]. The random-surfer-model simulates the navigation of a user through the web as a random walk: After reading one web page, the random surfer either follows a link to another page or randomly jumps to a new page. Applied to the context of a dependency graph, the surfer is considered to be the new developer who reads through the source code. After visiting one class he either follows a link (dependency edge) to another class or randomly jumps with probability α to a new class. The random-jump-probability α is an important parameter of the algorithm, that needs to be chosen appropriately. Considering the random-surfer-model over infinite time, the PageRank gives a stationary probability distribution to represent the likelihood that the developer will read any particular class.

In an extension, the algorithm can generate biased ranks by using *priors*. Priors represent nodes of the system which are known to be important prior to running the algorithm. Prior nodes have higher initial probabilities and therefore the output ranking is biased towards these nodes.

HITS. Similar to PageRank, HITS is an algorithm originally designed to rank web pages. However, it does not assign a single value to each node, but calculates two scores, the hub and the authority score. A good hub represents a node that points to many good authorities and a good authority represents a page that is pointed to by many good hubs. Roughly speaking, good authorities are nodes with a large number of incoming links and hubs are pages with a large number of outgoing links. Preliminary experiments showed that in our context it is better to use the authority score as centrality metric than the hub score.

In the same way as PageRank, the HITS algorithm incorporates a random-jump-probability as a parameter input. HITS can also be extended and supplied with previous knowledge about priors.

Markov. The Markov approach is to view the dependency graph as a first-order Markov chain, where a "token" traverses the graph in a stochastic manner for an infinitely long time. The stationary distribution denotes the fraction of time that the token spends at any single node [6]. The Markov centrality of a node v then denotes the inverse of the average *mean first passage time* in the Markov chain. The mean first passage time m_{sv} is defined as the expected number of steps taken until the first arrival of the token at v starting at node s. The average is taken over all nodes, that are specified as priors of the algorithm. In contrast to PageRank and HITS, the specification of prior nodes is not optional, but required.

Hierarchical flow model. In addition to commonly used centrality metrics, we also designed a hierarchical flow model specifically for the context of software systems. To calculate a centrality measurement, the hierarchical flow model is built in two steps: First, a centrality index is used on an aggregated dependency graph, where each node corresponds to one package rather than one class. Edges of each package node represent the accumulation of all edges of the classes/interfaces within the package. The centrality index then provides an importance ranking among all packages. In preliminary experiments we worked with PageRank, Markov and HITS as centrality indices. The results did not differ significantly, mostly the same packages were considered to be the most important, only in different orders. Hence we arbitrarily chose to use PageRank for the first step.

In a second step, a flow model is built for each package. The graph G_F of the flow model contains a single vertex for each class/interface of the package. In addition, the graph contains a source-vertex and a sink-vertex, which represent the rest of the system: $G_F = (V \cup \{source, sink\}, E)$ with $E = \{e = (v_1, v_2) |\ v_1$ depends on $v_2 \wedge v_1 \in V \wedge v_2 \in V\} \cup \{e = (source, v_2) |\ v_2$ has an incoming edge from a node outside the package $\} \cup \{e = (v_1, sink) |\ v_1$ has an outgoing edge to a node outside the package $\}$. Figure 1 illustrates an example of the flow model for the package org.conqat.engine.commons from our tool ConQAT. Note that the resulting graph is not necessarily acyclic as the one in Figure 1. Also, not every node has to be connected to the source (or the sink) if the class does not depend on the rest of the system.

The edges of the graph have a capacity according to the following weight function w:

$$w((v_1, v_2)) = \begin{cases} occ(v_1, v_2), & v_1 = source \\ sum, & \text{otherwise} \ , \end{cases} .$$

where $occ(v_1, v_2)$ denotes the accumulative occurrences of the dependencies between v_1 and v_2 and

$$sum = \sum_{v \in V} occ(source, v) \ .$$

The capacities are designed such that flow coming in from the source is not restricted by the capacity of any edge along a path to the sink.

The score of each vertex within the package is calculated as the decrease in the maximum flow of the graph when the node

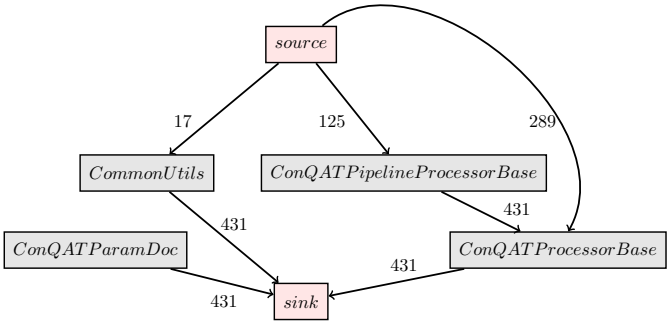

Fig. 1. Flow model of org.conqat.engine.commons

TABLE I
RANKING ON ORG.CONQAT.ENGINE.COMMONS

Class	rank
ConQATProcessorBase	414
ConQATPipelineProcessorBase	125
CommonUtils	17
ConQATParamDoc	0

and all incident edges are removed. The higher the decrease of the maximum flow, the more central we assume the node to be. In the context of a software system, the graph models the flow of information coming in from the rest of the system, flowing through the package and back to the rest of the system. The score is designed such that it represents the control each vertex has over the flow. The scores for vertices of the package org.conqat.engine.commons can be found in Table I.

It remains to be determined how the overall recommendation of the model is calculated (see Section V). The recommendation should include a certain number of the most central nodes of each package in the ordering (or a variant of it) as determined by the PageRank values in step one.

V. CASE STUDY DESIGN

The large number of different centrality indices leads to a variety of possible results. This section describes the design of an empirical case study that evaluates these results, and concludes with a suggestion for the most useful recommendation algorithm.

A. Research Questions

The following questions guided the design of the case study:

RQ1: What is the influence of the priors? Some of the centrality indices require prior nodes or take them as optional user input. We investigate how the choice of different prior nodes effects the outcome of the algorithm.

RQ2: Should the dependency graph be directed or undirected? We evaluate the results based both on the directed and undirected dependency graphs.

RQ3: Which dependencies should be represented as an edge of the dependency graph? We investigate how the kinds of dependencies included in the dependency graph influence the recommendation precision. We consider five different kinds of dependencies, as described in Section IV-A.

RQ4: Which centrality index yields the best result? We examine which index suits our recommendation model best. Some indices have additional parameters (see Section V-D), which are chosen according to preliminary experiments.

RQ5: How does the algorithm perform compared to recommendations of a single developer? We compare the recommendation set of the algorithm with the recommendation of each single developer and also investigate the intersection of the developer's opinion per project.

RQ6: How much better is the algorithm compared to a trivial approach? As a trivial approach the largest classes are recommended. We measure the size of a class in lines of codes and recommend the ten, twenty and fifty largest classes.

B. Study Objects

The study was conducted on four Java open source projects (see Table II). JEdit and jMol are two open source projects available from SourceForge:[2] JEdit is a text editor, jMol a visualization tool for chemical structures in 3D. The third project, ConQAT Engine, is the core of the software quality analysis tool ConQAT. The voTUM framework visualizes optimization techniques of compilers.[3] Table II gives an overview of the size of the projects, measured in LoC (lines of code), and denotes the number of vertices and edges in the directed dependency graph. The vertices are counted after merging interfaces with single implementations. The number of edges includes all five dependencies mentioned in Section IV-A.

C. Evaluation

To evaluate the recommendation set of the algorithm, developers of each project were asked to name the ten most important classes of their system, which they would recommend a new developer for his training period. Table III shows the number of developers of each project who replied to the survey.

For research questions RQ1 - RQ4, we evaluate the results based on the recommendation precision, calculated over the

[2]http://sourceforge.net/
[3]http://www2.in.tum.de/votum

TABLE II
OPEN SOURCE PROJECTS EVALUATED IN THE STUDY

Project	Version	LoC	Vertices	Edges
ConQAT Engine	2011.9	186.486	1571	7116
jEdit	4.5	164.783	499	2817
jMol	12.2	229.980	455	2671
voTUM	0.7.5	60.792	275	1393

TABLE III
OPEN SOURCE DEVELOPERS PARTICIPATING IN THE STUDY

Project	# developers
ConQAT Engine	4
jEdit	2
jMol	3
voTUM	3

union of the developers' opinions. A recommendation in the recommendation set is considered to be correct if it was named by at least one developer. Thus, we do not take any ordering into account. We consider the top ten, top 20, and top 50 classes for the recommendation set of the algorithm and refer to the corresponding precisions as *RP10*, *RP20* and *RP50*.

For research question RQ5 we calculate the recommendation precision for each single developer of the project. Hence, a recommendation is only considered to be correct if it was named by the specific developer under evaluation.

D. Parameter configurations

As described in Section IV-B, some of the centrality indices require parameters. In the following we list the configuration with which we ran the case study:

Random-Jump-Probability. Page-Rank and HITS require a random-jump-probability α. In general, α should be chosen between 0 and 1. To find the best α, we used preliminary experiments and ran Page-Rank and HITS with different values for α. These experiments showed that the higher the α, the more uniform the final distribution. Because a non-uniform final distribution with high variance between two different node values is desirable, we choose α to be very small and used a random-jump-probability of 0.001.

Priors. Some algorithms take priors as an optional or required input. Research question 1 will discuss in Section VI and VII which priors are best to use. Based on that conclusion, Table IV shows the priors used for the case study.

Flow model. For the hierarchical flow model, we decide to use the top 25 packages according to the PageRank algorithm. Within each package we use the top two classes according to the maximum decrease in the flow value. To get a total order, we rank the top two classes of the most important package first, followed by the top two classes of the second-most important package etc.

With manual inspection, we investigated if the overall ranking of the flow model could be constructed differently.

TABLE IV
PRIOR NODES USED IN THE STUDY

Project	prior
ConQAT Engine	org.conqat.engine.core.driver.Driver
jEdit	org.gjt.sp.jedit.jEdit
jMol	org.jmol.applet.WrappedApplet/ org.jmol.applet.Jmol
voTUM	de.tum.in.wwwseidl.votum.gui.VoTUM

TABLE V
DIFFERENT SETS OF PRIORS FOR CONQAT ENGINE

Test	Priors
1	Driver
2	ConQATProcessorBase
3	IConQATProcessor
4	Driver, IConQATProcessor, ConQATProcessorBase
5	Driver, WebconsoleMain, ConQATRunner
6	JavaDocAnalyzer, ResourceBuilder CloneEditPropagator

We determined that approximately only the top ten packages according to PageRank contained classes that were named by one of the developers. Within each package only the top two classes seemed to be relevant. We evaluated if different rankings, e.g. a ranking containing the top class of the top ten packages first, followed by the second-most important classes of the top ten packages etc. would perform better. However, we believe that the overall ranking as mentioned above suits our requirements best.

VI. EXPERIMENTS & RESULTS

RQ1. To investigate the influence of the priors, we compare the results of using different prior test sets for ConQATEngine, as shown in Table V. The class Driver.java contains a main method and is the entry point of the system. ConQATProcessorBase and IConQATProcessor were both named by at least one developer as the most important class. ConQATProcessorBase is also determined to be important by our algorithm and frequently found in the top ten recommendation set. In contrast, IConQATProcessor is not considered to be important by our algorithm and is usually not included in the top twenty recommendation set. In addition to the three individual test sets, test set 4 includes all three of them. Test set 5 consists of three classes that contain a main method each and test set 6 contains three randomly chosen classes.

We use these priors to calculate the markov centrality, the PageRank with priors, and the HITS centrality with Priors on ConQAT Engine. For the dependency graph we experiment with different combinations of dependency edges: inheritance dependency (I), parameter dependency (P), return dependency (R), field dependency (F) and method dependency (M). We take the combinations I, IPR, IFM and IFMPR. Figure 2, 3 and 4 show the resulting recommendation precisions RP10, RP20 and RP50 for using each test set on the dependency graphs I, IPR, IFM and IPRFM. Thereby, RP10, RP20 and RP50 are displayed in one column: The bottom section of each column represents RP10 (highest opacity). The bottom and the middle section together represent RP20, and the entire column represents RP50.

We also ran similar experiments on the other study objects. However, since the results for ConQAT engine are representative, we do not include additional tables.

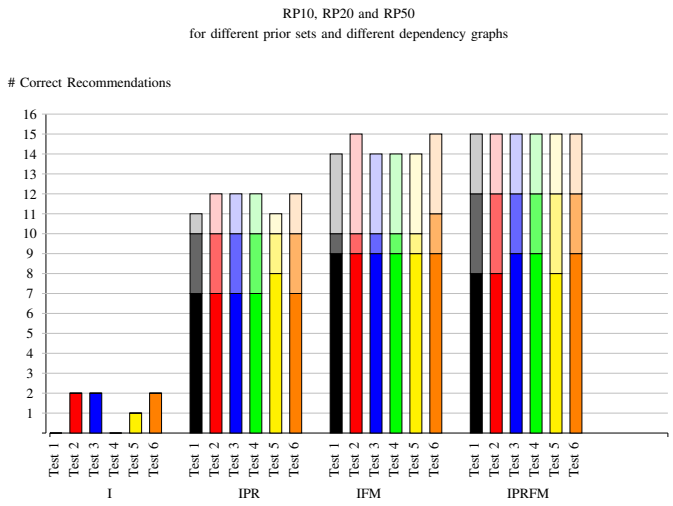

Fig. 2. Influence of choosing different priors for markov centrality, run on ConQAT Engine

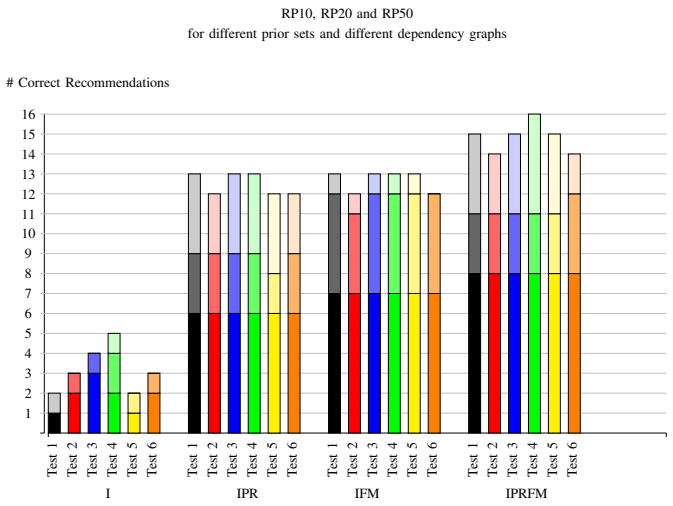

Fig. 4. Influence of choosing different priors for HITS, run on ConQAT Engine

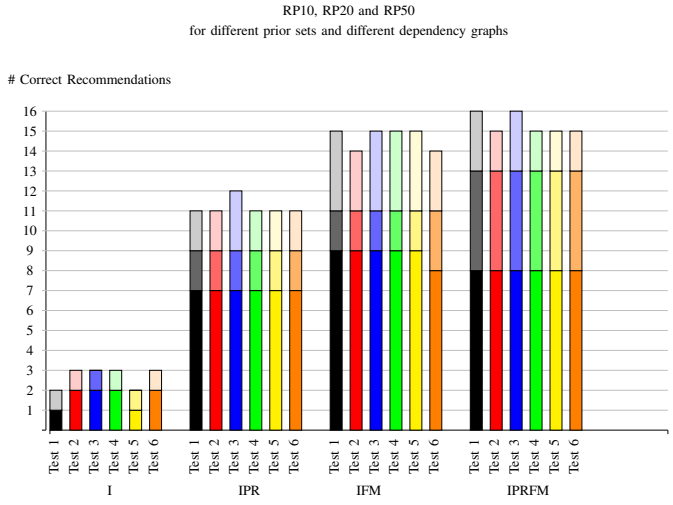

Fig. 3. Influence of choosing different priors for PageRank, run on ConQAT Engine

RQ2 - RQ4. Since research questions RQ2, RQ3 and RQ4 can not be answered separately (the best type of dependency graph might vary for different centrality indices), we design an experiment to answer the three questions together: We evaluate the recommendation precisions RP10, RP20, RP50 for different combinations of centrality index and dependency graph. As index we use closeness-betweenness, PageRank, PageRank with priors, HITS, HITS with priors, Markov and the flow model. In addition, we work with the dependency graphs I, IPR, IFM and IPRFM in both the directed and the undirected version. Tables VI, VII, VIII, and IX show the results: Rows represent the kinds of dependency graphs, columns the centrality indices. Each cell contains the values of RP10, RP20 and RP50. For each kind of edge set, the highest recommendation precision is printed in bold. The global maximum precision per project is marked with a grey cell.

RQ5. Developers of the same project often have a different view on their software. Their feedback on our survey reveals that developers' recommendation on central classes can differ. For each project Table X shows the size of the intersection of the developers' opinion. For ConQAT Engine and jEdit the developers agree on three classes to be among the top ten. For voTUM and jMol the intersection size is even only one.

We further evaluate the recommendation precision of our algorithm based on the opinion of each individual developer. Figure 5 shows the results for the project ConQAT Engine, evaluated on the dependency graphs I, IFM, IPR and IPRFM, using Markov. Figure 5 is representative for the results on the other case study objects. Hence we do not attach further graphs.

RQ6. In another experiment we evaluate how much better our approach is compared to a trivial one. The trivial approach recommends the ten (twenty or fifty) largest classes of the system, measuring size in lines of code. Table XI shows the

TABLE X
SIZE OF THE INTERSECTION SET OF DEVELOPERS' OPINIONS

Project	intersection
ConQAT Engine	3
jEdit	3
jMol	1
voTUM	1

TABLE VI
RESULTS ON PROJECT CONQAT ENGINE

Graph	Betweenness	PageRank	PageRankPr	HITS	HITSPr	Markov	Flow
I undirected	$\frac{3}{10},\frac{3}{20},\frac{4}{50}$	$\frac{2}{10},\frac{2}{20},\frac{3}{50}$	$\frac{1}{10},\frac{1}{20},\frac{2}{50}$	$\frac{2}{10},\frac{2}{20},\frac{3}{50}$	$\frac{1}{10},\frac{1}{20},\frac{2}{50}$,	$\frac{0}{10},\frac{0}{20},\frac{0}{50}$	$\frac{3}{10},\frac{3}{20},\frac{4}{50}$
I directed	$\frac{2}{10},\frac{3}{20},\frac{6}{50}$	$\frac{4}{10},\frac{5}{20},\frac{6}{50}$	$\frac{2}{10},\frac{2}{20},\frac{2}{50}$	$\frac{2}{10},\frac{2}{20},\frac{6}{50}$	$\frac{1}{10},\frac{2}{20},\frac{2}{50}$	$\frac{4}{10},\frac{5}{20},\frac{6}{50}$	$\frac{2}{10},\frac{4}{20},\frac{4}{50}$
IPR undirected	$\frac{6}{10},\frac{9}{20},\frac{12}{50}$	$\mathbf{\frac{7}{10},\frac{9}{20},\frac{11}{50}}$	$\mathbf{\frac{7}{10},\frac{9}{20},\frac{11}{50}}$	$\frac{7}{10},\frac{9}{20},\frac{9}{50}$	$\frac{6}{10},\frac{9}{20},\frac{13}{50}$	$\mathbf{\frac{7}{10},\frac{10}{20},\frac{11}{50}}$	$\frac{4}{10},\frac{6}{20},\frac{8}{50}$
IPR directed	$\frac{6}{10},\frac{9}{20},\frac{11}{50}$	$\frac{3}{10},\frac{4}{20},\frac{9}{50}$	$\frac{1}{10},\frac{2}{20},\frac{4}{50}$	$\frac{4}{10},\frac{4}{20},\frac{4}{50}$	$\frac{4}{10},\frac{5}{20},\frac{6}{50}$	$\frac{4}{10},\frac{5}{20},\frac{9}{50}$	$\frac{4}{10},\frac{4}{20},\frac{7}{50}$
IFM undirected	$\frac{6}{10},\frac{8}{20},\frac{12}{50}$	$\frac{9}{10},\frac{11}{20},\frac{14}{50}$	$\mathbf{\frac{9}{10},\frac{11}{20},\frac{15}{50}}$	$\frac{7}{10},\frac{9}{20},\frac{10}{50}$	$\frac{7}{10},\frac{12}{20},\frac{13}{50}$	$\frac{9}{10},\frac{10}{20},\frac{14}{50}$	$\frac{4}{10},\frac{7}{20},\frac{10}{50}$
IFM directed	$\frac{1}{10},\frac{6}{20},\frac{12}{50}$	$\frac{5}{10},\frac{7}{20},\frac{15}{50}$	$\frac{1}{10},\frac{1}{20},\frac{2}{50}$	$\frac{8}{10},\frac{11}{20},\frac{14}{50}$	$\frac{8}{10},\frac{11}{20},\frac{15}{50}$	$\frac{5}{10},\frac{8}{20},\frac{15}{50}$	$\frac{0}{10},\frac{3}{20},\frac{5}{50}$
IPRFM undirected	$\frac{6}{10},\frac{8}{20},\frac{14}{50}$	$\frac{8}{10},\frac{13}{20},\frac{15}{50}$	$\mathbf{\frac{8}{10},\frac{13}{20},\frac{16}{50}}$	$\frac{8}{10},\frac{12}{20},\frac{14}{50}$	$\frac{8}{10},\frac{11}{20},\frac{15}{50}$	$\frac{8}{10},\frac{12}{20},\frac{15}{50}$	$\frac{6}{10},\frac{10}{20},\frac{12}{50}$
IPRFM directed	$\frac{5}{10},\frac{9}{20},\frac{13}{50}$	$\frac{4}{10},\frac{7}{20},\frac{12}{50}$	$\frac{1}{10},\frac{1}{20},\frac{5}{50}$	$\frac{8}{10},\frac{11}{20},\frac{14}{50}$	$\frac{7}{10},\frac{11}{20},\frac{14}{50}$	$\frac{2}{10},\frac{4}{20},\frac{12}{50}$	$\frac{3}{10},\frac{5}{20},\frac{10}{50}$

TABLE VII
RESULTS ON PROJECT JEDIT

Graph	Betweenness	PageRank	PageRankPr	HITS	HITSPr	Markov	Flow
I undirected	$\frac{0}{10},\frac{0}{20},\frac{3}{50}$	$\frac{0}{10},\frac{1}{20},\frac{4}{50}$	$\frac{2}{10},\frac{2}{20},\frac{2}{50}$	$\frac{0}{10},\frac{0}{20},\frac{0}{50}$	$\frac{2}{10},\frac{2}{20},\frac{2}{50}$	$\frac{0}{10},\frac{0}{20},\frac{0}{50}$	$\frac{0}{10},\frac{0}{20},\frac{1}{50}$
I directed	$\frac{0}{10},\frac{0}{20},\frac{1}{50}$	$\frac{0}{10},\frac{2}{20},\frac{3}{50}$	$\frac{1}{10},\frac{1}{20},\frac{2}{50}$	$\frac{0}{10},\frac{2}{20},\frac{4}{50}$	$\frac{1}{10},\frac{1}{20},\frac{2}{50}$	$\frac{0}{10},\frac{1}{20},\frac{4}{50}$	$\frac{1}{10},\frac{1}{20},\frac{1}{50}$
IPR undirected	$\frac{4}{10},\frac{5}{20},\frac{9}{50}$	$\frac{4}{10},\frac{6}{20},\frac{10}{50}$	$\frac{4}{10},\frac{6}{20},\frac{10}{50}$	$\frac{0}{10},\frac{0}{20},\frac{0}{50}$	$\frac{4}{10},\frac{7}{20},\frac{8}{50}$	$\frac{6}{10},\frac{8}{20},\frac{10}{50}$	$\frac{3}{10},\frac{4}{20},\frac{4}{50}$
IPR directed	$\frac{6}{10},\frac{8}{20},\frac{11}{50}$	$\frac{1}{10},\frac{3}{20},\frac{7}{50}$	$\mathbf{\frac{6}{10},\frac{9}{20},\frac{10}{50}}$	$\frac{0}{10},\frac{0}{20},\frac{0}{50}$	$\frac{4}{10},\frac{5}{20},\frac{9}{50}$	$\frac{3}{10},\frac{4}{20},\frac{9}{50}$	$\frac{3}{10},\frac{3}{20},\frac{4}{50}$
IFM undirected	$\frac{2}{10},\frac{6}{20},\frac{9}{50}$	$\frac{5}{10},\frac{7}{20},\frac{11}{50}$	$\frac{5}{10},\frac{7}{20},\frac{11}{50}$	$\frac{4}{10},\frac{7}{20},\frac{12}{50}$	$\frac{4}{10},\frac{5}{20},\frac{10}{50}$	$\frac{4}{10},\frac{9}{20},\frac{13}{50}$	$\frac{4}{10},\frac{5}{20},\frac{5}{50}$
IFM directed	$\mathbf{\frac{5}{10},\frac{8}{20},\frac{10}{50}}$	$\frac{1}{10},\frac{3}{20},\frac{9}{50}$	$\frac{3}{10},\frac{7}{20},\frac{10}{50}$	$\frac{5}{10},\frac{6}{20},\frac{13}{50}$	$\frac{2}{10},\frac{5}{20},\frac{9}{50}$	$\frac{1}{10},\frac{3}{20},\frac{9}{50}$	$\frac{3}{10},\frac{5}{20},\frac{5}{50}$
IPRFM undirected	$\frac{4}{10},\frac{6}{20},\frac{10}{50}$	$\frac{4}{10},\frac{7}{20},\frac{12}{50}$	$\frac{4}{10},\frac{7}{20},\frac{12}{50}$	$\frac{4}{10},\frac{7}{20},\frac{12}{50}$	$\frac{4}{10},\frac{6}{20},\frac{10}{50}$	$\frac{4}{10},\frac{8}{20},\frac{12}{50}$	$\frac{4}{10},\frac{4}{20},\frac{4}{50}$
IPRFM directed	$\mathbf{\frac{5}{10},\frac{8}{20},\frac{10}{50}}$	$\frac{3}{10},\frac{5}{20},\frac{11}{50}$	$\frac{4}{10},\frac{8}{20},\frac{13}{50}$	$\frac{5}{10},\frac{7}{20},\frac{13}{50}$	$\frac{4}{10},\frac{6}{20},\frac{10}{50}$	$\frac{3}{10},\frac{7}{20},\frac{11}{50}$	$\frac{3}{10},\frac{4}{20},\frac{4}{50}$

TABLE VIII
RESULTS ON PROJECT JMOL

Graph	Betweenness	PageRank	PageRankPr	HITS	HITSPr	Markov	Flow
I undirected	$\frac{1}{10},\frac{1}{20},\frac{1}{50}$	$\frac{1}{10},\frac{1}{20},\frac{5}{50}$	$\frac{2}{10},\frac{2}{20},\frac{2}{50}$	$\frac{0}{10},\frac{0}{20},\frac{0}{50}$	$\frac{2}{10},\frac{2}{20},\frac{2}{50}$	$\frac{0}{10},\frac{0}{20},\frac{0}{50}$	$\frac{1}{10},\frac{1}{20},\frac{1}{50}$
I directed	$\frac{0}{10},\frac{1}{20},\frac{2}{50}$	$\frac{1}{10},\frac{1}{20},\frac{3}{50}$	$\frac{4}{10},\frac{4}{20},\frac{5}{50}$	$\frac{1}{10},\frac{1}{20},\frac{2}{50}$	$\frac{3}{10},\frac{4}{20},\frac{5}{50}$	$\frac{2}{10},\frac{2}{20},\frac{3}{50}$	$\frac{0}{10},\frac{1}{20},\frac{2}{50}$
IPR undirected	$\frac{3}{10},\frac{4}{20},\frac{10}{50}$	$\frac{6}{10},\frac{8}{20},\frac{12}{50}$	$\frac{6}{10},\frac{8}{20},\frac{13}{50}$	$\mathbf{\frac{7}{10},\frac{9}{20},\frac{13}{50}}$	$\frac{4}{10},\frac{7}{20},\frac{11}{50}$	$\frac{6}{10},\frac{9}{20},\frac{12}{50}$	$\frac{3}{10},\frac{5}{20},\frac{6}{50}$
IPR directed	$\frac{4}{10},\frac{5}{20},\frac{9}{50}$	$\frac{1}{10},\frac{3}{20},\frac{8}{50}$	$\frac{3}{10},\frac{5}{20},\frac{10}{50}$	$\frac{6}{10},\frac{7}{20},\frac{11}{50}$	$\frac{2}{10},\frac{3}{20},\frac{8}{50}$	$\frac{0}{10},\frac{0}{20},\frac{1}{50}$	$\frac{1}{10},\frac{4}{20},\frac{5}{50}$
IFM undirected	$\frac{3}{10},\frac{6}{20},\frac{12}{50}$	$\frac{5}{10},\frac{7}{20},\frac{14}{50}$	$\frac{5}{10},\frac{7}{20},\frac{14}{50}$	$\frac{5}{10},\frac{8}{20},\frac{12}{50}$	$\frac{3}{10},\frac{6}{20},\frac{13}{50}$	$\frac{5}{10},\frac{7}{20},\frac{13}{50}$	$\frac{4}{10},\frac{6}{20},\frac{6}{50}$
IFM directed	$\mathbf{\frac{5}{10},\frac{9}{20},\frac{12}{50}}$	$\frac{3}{10},\frac{4}{20},\frac{9}{50}$	$\frac{4}{10},\frac{4}{20},\frac{9}{50}$	$\frac{4}{10},\frac{5}{20},\frac{9}{50}$	$\frac{3}{10},\frac{7}{20},\frac{8}{50}$	$\frac{2}{10},\frac{3}{20},\frac{8}{50}$	$\frac{4}{10},\frac{5}{20},\frac{6}{50}$
IPRFM undirected	$\frac{3}{10},\frac{8}{20},\frac{11}{50}$	$\frac{5}{10},\frac{8}{20},\frac{14}{50}$	$\frac{5}{10},\frac{8}{20},\frac{14}{50}$	$\mathbf{\frac{6}{10},\frac{9}{20},\frac{11}{50}}$	$\frac{3}{10},\frac{7}{20},\frac{12}{50}$	$\frac{5}{10},\frac{9}{20},\frac{13}{50}$	$\frac{4}{10},\frac{6}{20},\frac{6}{50}$
IPRFM directed	$\frac{5}{10},\frac{9}{20},\frac{13}{50}$	$\frac{3}{10},\frac{5}{20},\frac{11}{50}$	$\frac{3}{10},\frac{6}{20},\frac{12}{50}$	$\frac{4}{10},\frac{7}{20},\frac{11}{50}$	$\frac{3}{10},\frac{7}{20},\frac{8}{50}$	$\frac{4}{10},\frac{7}{20},\frac{12}{50}$	$\frac{4}{10},\frac{6}{20},\frac{6}{50}$

TABLE IX
RESULTS ON PROJECT VOTUM

Graph	Betweenness	PageRank	PageRankPr	HITS	HITSPr	Markov	Flow
I undirected	$\frac{4}{10},\frac{7}{20},\frac{12}{50}$	$\mathbf{\frac{6}{10},\frac{7}{20},\frac{12}{50}}$	$\frac{2}{10},\frac{3}{20},\frac{5}{50}$	$\frac{3}{10},\frac{4}{20},\frac{6}{50}$	$\frac{2}{10},\frac{3}{20},\frac{5}{50}$	$\frac{2}{10},\frac{2}{20},\frac{4}{50}$	$\frac{3}{10},\frac{3}{20},\frac{4}{50}$
I directed	$\frac{5}{10},\frac{7}{20},\frac{10}{50}$	$\frac{5}{10},\frac{8}{20},\frac{13}{50}$	$\frac{2}{10},\frac{5}{20},\frac{8}{50}$	$\frac{4}{10},\frac{7}{20},\frac{13}{50}$	$\frac{2}{10},\frac{4}{20},\frac{8}{50}$	$\frac{5}{10},\frac{7}{20},\frac{13}{50}$	$\frac{6}{10},\frac{6}{20},\frac{8}{50}$
IPR undirected	$\frac{4}{10},\frac{8}{20},\frac{17}{50}$	$\frac{6}{10},\frac{10}{20},\frac{17}{50}$	$\frac{6}{10},\frac{11}{20},\frac{17}{50}$	$\frac{6}{10},\frac{9}{20},\frac{12}{50}$	$\frac{6}{10},\frac{10}{20},\frac{16}{50}$	$\mathbf{\frac{8}{10},\frac{10}{20},\frac{14}{50}}$	$\frac{7}{10},\frac{7}{20},\frac{9}{50}$
IPR directed	$\frac{5}{10},\frac{8}{20},\frac{15}{50}$	$\frac{3}{10},\frac{4}{20},\frac{9}{50}$	$\frac{4}{10},\frac{6}{20},\frac{14}{50}$	$\frac{2}{10},\frac{3}{20},\frac{12}{50}$	$\frac{5}{10},\frac{10}{20},\frac{15}{50}$	$\frac{2}{10},\frac{2}{20},\frac{12}{50}$	$\frac{2}{10},\frac{5}{20},\frac{8}{50}$
IFM undirected	$\frac{4}{10},\frac{7}{20},\frac{12}{50}$	$\frac{6}{10},\frac{8}{20},\frac{15}{50}$	$\frac{6}{10},\frac{8}{20},\frac{15}{50}$	$\frac{6}{10},\frac{8}{20},\frac{10}{50}$	$\mathbf{\frac{6}{10},\frac{10}{20},\frac{15}{50}}$	$\frac{5}{10},\frac{9}{20},\frac{12}{50}$	$\frac{6}{10},\frac{6}{20},\frac{8}{50}$
IFM directed	$\frac{5}{10},\frac{6}{20},\frac{9}{50}$	$\frac{5}{10},\frac{7}{20},\frac{14}{50}$	$\frac{3}{10},\frac{4}{20},\frac{7}{50}$	$\frac{6}{10},\frac{9}{20},\frac{10}{50}$	$\frac{5}{10},\frac{8}{20},\frac{13}{50}$	$\frac{5}{10},\frac{7}{20},\frac{14}{50}$	$\frac{5}{10},\frac{7}{20},\frac{9}{50}$
IPRFM undirected	$\frac{5}{10},\frac{8}{20},\frac{13}{50}$	$\frac{4}{10},\frac{11}{20},\frac{14}{50}$	$\frac{4}{10},\frac{11}{20},\frac{14}{50}$	$\frac{6}{10},\frac{7}{20},\frac{12}{50}$	$\frac{5}{10},\frac{10}{20},\frac{16}{50}$	$\mathbf{\frac{6}{10},\frac{10}{20},\frac{13}{50}}$	$\frac{5}{10},\frac{6}{20},\frac{8}{50}$
IPRFM directed	$\frac{4}{10},\frac{6}{20},\frac{13}{50}$	$\frac{2}{10},\frac{5}{20},\frac{9}{50}$	$\frac{4}{10},\frac{6}{20},\frac{12}{50}$	$\frac{5}{10},\frac{8}{20},\frac{12}{50}$	$\frac{4}{10},\frac{9}{20},\frac{15}{50}$	$\frac{2}{10},\frac{4}{20},\frac{9}{50}$	$\frac{4}{10},\frac{5}{20},\frac{8}{50}$

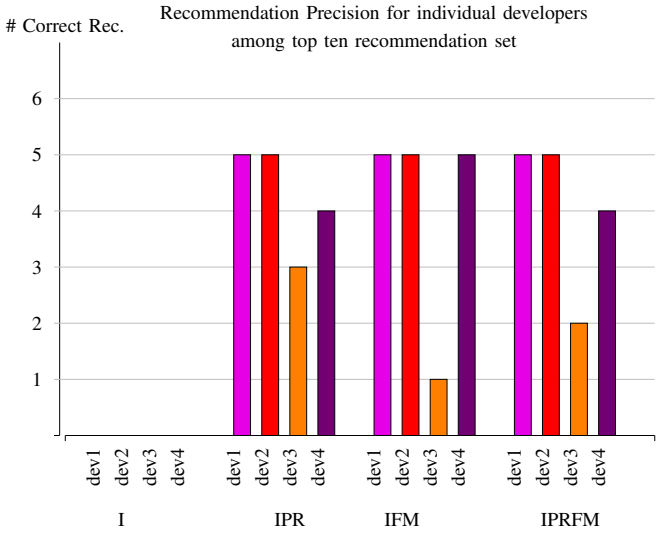

Fig. 5. Recommendation precision of individual developers of ConQAT Engine

recommendation precision RP10, RP20, and RP50 as well as the size of the largest class.

VII. Discussion

RQ1: What is the influence of the priors? The results as shown in Figures 2, 3 and 4 reveal that using different sets of priors does not change the outcome significantly. On graphs IPR, IFM, IPRFM, the values of RP10 for example differ by at most one, often they are the same for all sets of priors. On dependency graph I, the results vary slightly more. However, the recommendation precisions on this graph are lower than on the other graphs, so we will not use this kind of graph.

For an independent algorithm, priors should be chosen such that the least amount of developers' knowledge is required. We decide to use a class containing the main method. However, in most of our test systems there are multiple classes containing a main method, so the prior needs to be chosen manually or randomly. Table IV previously showed the priors we selected manually for the remaining research questions.

RQ2: Should the dependency graph be directed or undirected? Throughout all four test projects, PageRank, PageRank with priors, Markov, and the flow model perform better on the undirected dependency graphs than the directed ones.

TABLE XI
Results of the trivial approach by measuring the size of a class

Project	RP10, RP20, RP50
ConQAT Engine	$\frac{2}{10}, \frac{2}{20}, \frac{3}{50}$
jEdit	$\frac{4}{10}, \frac{7}{20}, \frac{11}{50}$
jMol	$\frac{7}{10}, \frac{7}{20}, \frac{12}{50}$
voTUM	$\frac{2}{10}, \frac{4}{20}, \frac{7}{50}$

HITS and HITS Prior often obtain similar results for directed and undirected graphs. For undirected graphs, both scores authority and hubs are the same. For directed graph, we use the authority score as centrality index, because the authority score leads to better results.

The betweenness-centrality sometimes performs better on the directed graph. However, in most cases, the betweenness-centrality is outperformed by the other algorithms. In the few cases, where betweenness-centrality achieves the highest recommendation precision for one type of graph, Markov and PageRank reveal similar results on the undirected graph. Therefore it is legitimate to say that it is outperformed in the general case. This indicates that the usage of shortest-path measurements on dependency graphs is not particularly useful for a centrality recommendation system. This is in contrast to the results of [1], who are most successful applying their shortest-path-centrality.

We conclude that the best recommendations are given when the dependency graph is undirected.

RQ3 & RQ4: Which dependencies should be represented as an edge of the dependency graph? Which centrality index yields the best result? In all test projects except of ConQAT Engine the highest precision is found for the IPR dependency graph, which includes inheritance, parameter and return dependencies. However, the index with the highest precision varies.

Figure 6 visualizes the results of Tables VI-IX on the undirected IPR dependency graph. The figure shows only the results of PageRank, PageRank with priors, HITS, and Markov, because the betweenness centrality and the flow model are generally outperformed. Considering only the undirected version of the IPR graph, the best indices (primarily

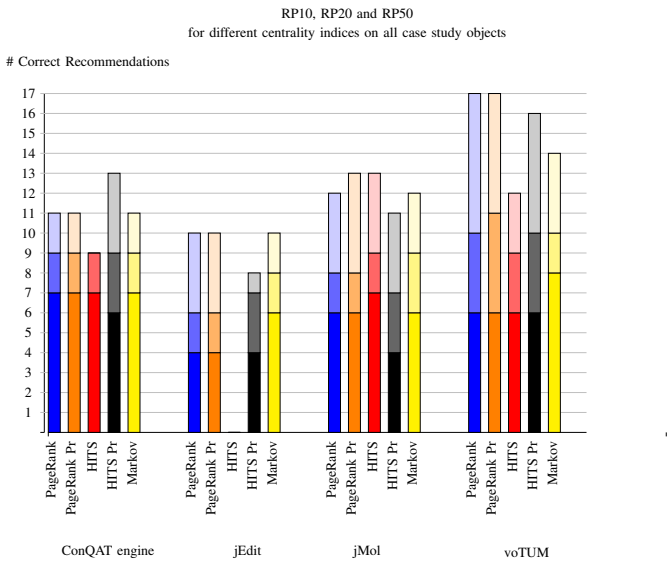

Fig. 6. Results of applying different centrality indices on the IPR undirected dependency graph

based on RP10) are Markov for projects jEdit and voTUM. On ConQAT engine, PageRank, PageRank with priors, and Markov perform equally well. For jMol, HITS leads to the best results, directly followed by Markov as the second best index. We conclude that applying the Markov centrality on the IPR depdency graph yields the best recommendation in accordance with the developers' opinion, because it performs the best on three projects and second-best on the fourth one.

On projects ConQAT, jEdit and jMol all centrality indices perform very poorly when applied to the inheritance graph (I). For voTUM the results of the inheritance graph are slightly better. In general this shows (as expected) that inheritance information on its own is not sufficient for a recommendation tool but needs to be combined with other dependency information. Why results are better for voTUM is speculative. VoTUM is the smallest project of all four study objects and the one with the strongest framework character. Maybe this leads to an increase in the ratio of inheritance dependency to the rest of the dependencies.

In most cases, the flow model is outperformed by random-walk based centralities. Manual inspection revealed that using the decrease in the flow value within one package does produce useful results in accordance with the developers' opinions. However, ranking the packages among themselves based on PageRank and recommending the top two vertices for each package does not have strong correlation with the developers' recommendation.

On the undirected versions of the dependency graphs, PageRank and Markov obtain similar results in terms of recommendation precision. Appendix A shows two recommendation sets using Markov and PageRank. In many cases, both algorithms end up with the same recommendation set, just in slightly different orders. This observation is in accordance to the results of [6], which evaluates the same network algorithms on a variety of real world data sets.

RQ5: How does the algorithm perform compared to recommendations of a single developer? Figure 5 shows that the recommendation precision of the algorithm based on the opinion of a single developer is at most 50 percent. Calculating the recommendation precision based on the union of all developers' opinions as in RQ1-RQ4 leads to much better results, with a precision up to 90 percent. The small intersection size of the developers' opinions (Table X) explains the difference: Developers agree only on a small number of classes to be central. Depending on which parts of the system they work with the most, they consider different classes to be important. However, the union over all developers' opinions matches the output of the algorithm.

In a second survey, we showed the ConQAT developers the results of the algorithm. They agree that their individual importance rankings reflect the parts of the system they work on. Furthermore they consider all classes of the top ten recommendation set to be important, even if they did not list them in their answer. The algorithm provides a better overview of the system than a single programmer could give on his own and recommends classes in agreement with the developers.

TABLE XII
RP10 OF THE TRIVIAL APPROACH COMPARED TO USING MARKOV ON THE UNDIRECTED IPR DEPENDENCY GRAPH.

Project	trivial	Markov
ConQAT Engine	20%	70%
jEdit	40%	60%
jMol	70%	60%
voTUM	20%	80%

RQ6: How much better is the algorithm compared to a trivial approach? We compare the trivial algorithm to our approach, using Markov on the undirected IPR graph, and show the results in Table XII. For the three projects ConQAT Engine, jEdit and voTUM our algorithm clearly outperforms the trivial approach: On ConQAT Engine, the trivial approach only enumerates two out of ten classes correctly, whereas our approach has a precision of eight out of ten. On the fourth project, jMol, both approaches perform equally well. With seven compared to six correct recommendations, the trivial approach achieved a slightly higher precision among the top ten set. However, we do not consider the difference of one correct class as significant.

VIII. VALIDITY OF THE CASE STUDY

Based on our experience, data from developers to evaluate the case study is difficult to collect. In the absence of more available data, the case study comprises only four open source projects. However, the projects were chosen from different domains so we believe that they represent a large area of software applications. We also tried to gather more data by using the Mylyn framework. However, the obtained data did not suit our purpose very well, as shown in Appendix B.

Our evaluation metric precision is designed such that it depends on the number of available developer opinions: More developer participating in the survey make it likelier for our algorithm to achieve a higher precision. One could argue that a large enough number of participating developers will result in a precision of 100 percent. Hence our evaluation metric is invalid as a class should not belong to the recommendation set of the algorithm because one single developer thinks that it is important, but because a vast majority of developers agrees on its centrality. However, we conducted another survey (see RQ5, Section VII) in which we showed the recommendation set the ConQAT Engine developers. Without exception, they commonly agreed that the classes of the recommendation set are central. Therefore we believe that our algorithm does produce very useful results.

We conducted a couple of preliminary experiments to narrow down the parameter space of the case study. We could have considered more centrality indices, more combinations of dependency edges and more priors, for example. However, it is not possible to make an exhaustive search through the parameter space. To our best knowledge, we chose the

preliminary experiments such that the case study experiments are the most representative and useful.

IX. Conclusion and Future Work

This paper has shown that network analysis on dependency graphs constitutes a useful foundation in order to recommend important classes of a system. An empirical case study was designed to find the best combination of centrality measurement and dependency graph. The case study which included four open source projects revealed a variety of interesting results: The centrality indices work best on an undirected dependency graph. Regarding the included dependency information, the subset of inheritance, parameter and return dependencies constitutes a promising approach. Furthermore using the Markov centrality leads to the best results, with a precision between 60 and 80 percent in the top ten recommendation set.

For future work, it might be a challenging task to confirm those results on a larger data base or investigate the results of the algorithm on a large industry software system. We also plan on evaluating the algorithm on software projects which are not written in Java, such as C/C++ or C#.

References

[1] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proceedings of the 30th international conference on Software engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 531–540. [Online]. Available: http://doi.acm.org/10.1145/1368088.1368161

[2] S. Kpodjedo, F. Ricca, P. Galinier, and G. Antoniol, "Not all classes are created equal: toward a recommendation system for focusing testing," in *Proceedings of the 2008 international workshop on Recommendation systems for software engineering*, ser. RSSE '08. New York, NY, USA: ACM, 2008, pp. 6–10. [Online]. Available: http://doi.acm.org/10.1145/1454247.1454250

[3] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," 1999.

[4] C. Pich, L. Nachmanson, and G. G. Robertson, "Visual analysis of importance and grouping in software dependency graphs," in *Proceedings of the 4th ACM symposium on Software visualization*, ser. SoftVis '08. New York, NY, USA: ACM, 2008, pp. 29–32. [Online]. Available: http://doi.acm.org/10.1145/1409720.1409725

[5] U. Brandes and T. Erlebach, *Network analysis: methodological foundations*, ser. Lecture notes in computer science: Tutorial. Springer, 2005. [Online]. Available: http://books.google.de/books?id=TTNhSm7HYrIC

[6] S. White and P. Smyth, "Algorithms for estimating relative importance in networks," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '03. New York, NY, USA: ACM, 2003, pp. 266–275. [Online]. Available: http://doi.acm.org/10.1145/956750.956782

[7] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *J. ACM*, vol. 46, no. 5, pp. 604–632, Sep. 1999. [Online]. Available: http://doi.acm.org/10.1145/324133.324140

APPENDIX A
RANKINGS FOR CONQAT ENGINE

Tables XIII and XIV show the resulting ranking of using Markov and PageRank on the undirected IPR dependency graph.

APPENDIX B
MYLYN DATA

In order to have more data for evaluation of our algorithm available, we also tried to use data provided by Mylyn[4]. Mylyn is a task-focused interface, integrated in the development environment of Eclipse. Mylyn monitors all programm activities while a developer is working on a certain task of his projects, e.g. bug fixing, producing code, testing etc. These programm activities are recorded in a *task-content*, specified in an XML format.

The bugzilla platform of Eclipse[5] records bugs for a variety of Eclipse projects and thereby also provides data recorded by Mylyn. In particular, bug fixing activities for the project Mylyn itself were monitored with Mylyn. We downloaded all .zip-files for any bug of one of the Mylyn projects (Mylyn, Mylyn Builds, Mylyn Commons, Mylyn Context, Mylyn Tasks ...) and created a statistic over the files that were read or edited while fixing a bug. The statistic shows which classes were selected or edited the most.

We then compared the results of the statistic with the results of our algorithm, using Markov or PageRank on the IPR undirected dependency graph. To run our algorithm, we used the source and the byte code of Mylyn, which is automatically downloaded while installing Eclipse. The source code can be found in several .jar-files in the plugin-folder of the Eclipse program.

The first hypothesis was that the most central classes of the algorithm are also read the most during a bug fix. The available data did not support this hypothesis. Using a statistic that contains only the selected files (and not the edited ones), one class of the top 50 of the statistic was included in the top ten recommendation set of the PageRank algorithm, similar with Markov. Including the edits to the statistic did not improve the correlation with the recommendation set. However, it seems intuitive that the most important classes are so well-known among the developers that they are mostly not affected by a bug and also not opened during any bug fix.

In a second hypothesis we claimed that the ranking of our algorithm contains a middle section of classes, that are not the most central, but also not peripheral. We claimed that these classes should be selected, but not edited during a bug fix. They should be selected, because they are not important enough such that every developer remembers them. They should not be edited, because they are nevertheless so important that we assumed them to be bug-free. We expected that the ranking of these classes within the statistic should be linearly related to the ranking position in the recommendation set of the algorithm. However, the data also did not support this hypothesis. Figure 7 shows the ranking in the statistic (x-axis) compared to the ranking in the recommendation set (y-Axis). We experimented both with including only the selected files in the statistic as well as the selected and the edited files, but the results were the same: No correlation could be found.

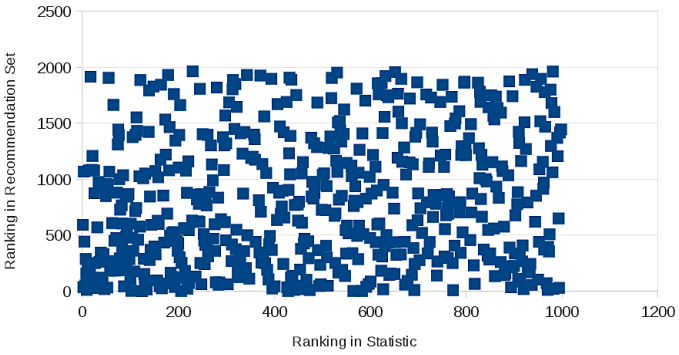

Fig. 7. Correlation between Mylyn statistic and recommendation set (PageRank, IPR, undirected)

While working with Mylyn data the following problems occurred: First, among the top 1000 classes of the statistic, only approximately half of them were found in the current source code. Manual inspection revealed that many classes changed name or were moved during development. Some classes were explicitly located in a folder called “src-old”. Others were external classes used from libraries and frameworks outside the source code.

Second, the Mylyn task content does not seem to differentiate between different edits of a file: Marking a method with the mouse without making a change results in the same edit event as renaming the method. To test hypothesis two we were therefore not able to filter those files that were opened, read, but not edited.

We conclude that using data gathered by Mylyn is not useful to evaluate the results of our algorithm.

[4]http://www.eclipse.org/mylyn/
[5]https://bugs.eclipse.org/bugs/

TABLE XIII
RECOMMENDATION LIST CONTAINING THE TOP 50 CLASSES FOR CONQAT ENGINE, USING MARKOV ON IPR, UNDIRECTED

Class/Interface	Markov
org.conqat.engine.commons.node.IConQATNode	0,014
org.conqat.engine.resource.text.ITextElement/org.conqat.engine.resource.text.TextElement	0,009
org.conqat.engine.core.logging.IConQATLogger	0,008
org.conqat.engine.commons.node.IRemovableConQATNode	0,008
org.conqat.engine.resource.IElement/org.conqat.engine.resource.base.ElementBase	0,008
org.conqat.engine.resource.text.ITextResource	0,007
org.conqat.engine.resource.IResource/org.conqat.engine.resource.base.ResourceBase	0,007
org.conqat.engine.sourcecode.resource.ITokenElement/org.conqat.engine.sourcecode.resource.TokenElement	0,006
org.conqat.engine.commons.ConQATProcessorBase	0,005
org.conqat.engine.sourcecode.resource.ITokenResource/org.conqat.engine.sourcecode.resource.TokenContainer	0,005
org.conqat.engine.commons.ConQATPipelineProcessorBase	0,005
org.conqat.engine.java.resource.IJavaElement/org.conqat.engine.java.resource.JavaElement	0,005
org.conqat.engine.commons.pattern.PatternList	0,005
org.conqat.engine.commons.traversal.ETargetNodes	0,005
org.conqat.engine.commons.findings.Finding	0,005
org.conqat.engine.code_clones.core.CloneClass	0,004
org.conqat.engine.commons.util.ConQATInputProcessorBase	0,004
org.conqat.engine.code_clones.detection.CloneDetectionResultElement	0,004
org.conqat.engine.java.resource.IJavaResource/org.conqat.engine.java.resource.JavaContainer	0,004
org.conqat.engine.resource.IContentAccessor/org.conqat.engine.resource.base.ContentAccessorBase	0,004
org.conqat.engine.core.bundle.BundlesConfiguration	0,004
org.conqat.engine.commons.node.ConQATNodeBase	0,003
org.conqat.engine.core.core.IConQATProcessorInfo	0,003
org.conqat.engine.commons.findings.FindingGroup	0,003
org.conqat.engine.core.bundle.BundleInfo	0,003
org.conqat.engine.core.driver.instance.BlockInstance	0,003
org.conqat.engine.code_clones.core.Unit	0,003
org.conqat.engine.graph.nodes.ConQATGraph	0,003
org.conqat.engine.code_clones.core.Clone	0,003
org.conqat.engine.core.driver.specification.BlockSpecification	0,003
org.conqat.engine.commons.traversal.NodeTraversingProcessorBase	0,003
org.conqat.engine.commons.findings.FindingReport	0,003
org.conqat.engine.resource.base.ElementTraversingProcessorBase	0,003
org.conqat.engine.resource.IContainer/org.conqat.engine.resource.base.ContainerBase	0,003
org.conqat.engine.commons.node.ListNode	0,003
org.conqat.engine.core.driver.info.BlockInfo	0,003
org.conqat.engine.sourcecode.parsed.IParsedElement/org.conqat.engine.sourcecode.parsed.ParsedElement	0,003
org.conqat.engine.commons.traversal.TargetExposedNodeTraversingProcessorBase	0,003
org.conqat.engine.simion.extraction.IChunkExtractionStrategy	0,002
org.conqat.engine.core.driver.declaration.IDeclaration/	
org.conqat.engine.core.driver.declaration.DeclarationBase	0,002
org.conqat.engine.core.driver.instance.ProcessorInstance	0,002
org.conqat.engine.simulink.scope.ISimulinkElement/org.conqat.engine.simulink.scope.SimulinkElement	0,002
org.conqat.engine.core.core.IProgressMonitor/org.conqat.engine.core.driver.ConQATInstrumentation	0,002
org.conqat.engine.core.driver.specification.ProcessorSpecification	0,002
org.conqat.engine.simion.extraction.Chunk	0,002
org.conqat.engine.resource.analysis.ElementAnalyzerBase	0,002
org.conqat.engine.architecture.overlap.IArchitectureComponent/	
org.conqat.engine.architecture.scope.ComponentNode	0,002
org.conqat.engine.java.base.JavaAnalyzerBase	0,002
org.conqat.engine.core.driver.error.ErrorLocation	0,002
org.conqat.engine.simulink.scope.ISimulinkResource/org.conqat.engine.simulink.scope.SimulinkContainer	0,002

TABLE XIV
RECOMMENDATION LIST CONTAINING THE TOP 50 CLASSES FOR CONQAT ENGINE, USING PAGERANK ON IPR, UNDIRECTED

Class/Interface	PageRank
org.conqat.engine.commons.node.IConQATNode	0,026
org.conqat.engine.commons.ConQATProcessorBase	0,022
org.conqat.engine.resource.text.ITextElement/org.conqat.engine.resource.text.TextElement	0,014
org.conqat.engine.resource.IElement/org.conqat.engine.resource.base.ElementBase	0,012
org.conqat.engine.sourcecode.resource.ITokenElement/org.conqat.engine.sourcecode.resource.TokenElement	0,011
org.conqat.engine.code_clones.core.CloneClass	0,009
org.conqat.engine.commons.node.IRemovableConQATNode	0,009
org.conqat.engine.core.logging.IConQATLogger	0,009
org.conqat.engine.resource.IResource/org.conqat.engine.resource.base.ResourceBase	0,009
org.conqat.engine.resource.text.ITextResource	0,009
org.conqat.engine.commons.ConQATPipelineProcessorBase	0,008
org.conqat.engine.java.resource.IJavaElement/org.conqat.engine.java.resource.JavaElement	0,008
org.conqat.engine.core.bundle.BundleInfo	0,006
org.conqat.engine.commons.traversal.ETargetNodes	0,006
org.conqat.engine.commons.pattern.PatternList	0,006
org.conqat.engine.sourcecode.resource.ITokenResource/org.conqat.engine.sourcecode.resource.TokenContainer	0,006
org.conqat.engine.commons.util.ConQATInputProcessorBase	0,006
org.conqat.engine.persistence.store.IStore	0,006
org.conqat.engine.code_clones.core.Clone	0,005
org.conqat.engine.code_clones.detection.CloneDetectionResultElement	0,005
org.conqat.engine.commons.findings.Finding	0,005
org.conqat.engine.graph.nodes.ConQATGraph	0,005
org.conqat.engine.resource.IContentAccessor/org.conqat.engine.resource.base.ContentAccessorBase	0,005
org.conqat.engine.core.driver.specification.BlockSpecification	0,004
org.conqat.engine.persistence.store.IStorageSystem	0,004
org.conqat.engine.simion.extraction.Chunk	0,004
org.conqat.engine.java.resource.IJavaResource/org.conqat.engine.java.resource.JavaContainer	0,004
org.conqat.engine.code_clones.core.Unit	0,004
org.conqat.engine.simulink.scope.ISimulinkElement/org.conqat.engine.simulink.scope.SimulinkElement	0,004
org.conqat.engine.commons.node.ConQATNodeBase	0,003
org.conqat.engine.architecture.overlap.IArchitectureComponent/	
org.conqat.engine.architecture.scope.ComponentNode	0,003
org.conqat.engine.commons.findings.FindingGroup	0,003
org.conqat.engine.model_clones.model.INode/org.conqat.engine.simulink.clones.model.SimulinkNode	0,003
org.conqat.engine.simion.extraction.IChunkExtractionStrategy	0,003
org.conqat.engine.commons.findings.FindingReport	0,003
org.conqat.engine.commons.traversal.NodeTraversingProcessorBase	0,003
org.conqat.engine.commons.node.ListNode	0,003
org.conqat.engine.commons.traversal.TargetExposedNodeTraversingProcessorBase	0,003
org.conqat.engine.resource.IContainer/org.conqat.engine.resource.base.ContainerBase	0,003
org.conqat.engine.core.driver.specification.ProcessorSpecification	0,003
org.conqat.engine.core.bundle.BundlesConfiguration	0,003
org.conqat.engine.model_clones.model.IDirectedEdge/	
org.conqat.engine.simulink.clones.model.SimulinkDirectedEdge	0,003
org.conqat.engine.sourcecode.parsed.IParsedElement/org.conqat.engine.sourcecode.parsed.ParsedElement	0,003
org.conqat.engine.code_clones.core.constraint.ICloneClassConstraint/	
org.conqat.engine.code_clones.core.constraint.ConstraintBase	0,002
org.conqat.engine.java.base.JavaAnalyzerBase	0,002
org.conqat.engine.core.core.IConQATProcessorInfo	0,002
org.conqat.engine.resource.base.ElementTraversingProcessorBase	0,002
org.conqat.engine.core.driver.declaration.IDeclaration/	
org.conqat.engine.core.driver.declaration.DeclarationBase	0,002
org.conqat.engine.core.driver.error.ErrorLocation	0,002
org.conqat.engine.core.driver.info.BlockInfo	0,002
org.conqat.engine.sourcecode.shallowparser.framework.ShallowEntity	0,002