Using Network Analysis for Recommendation of Central Software Classes

Daniela Steidl Benjamin Hummel
Technische Universität München, Garching b. München, Germany
{steidl,hummelb}@in.tum.de

Elmar Juergens
CQSE GmbH, Garching b. München, Germany
juergens@cqse.eu

***Abstract*—As a new developer, getting to know a large unknown software system is a challenging task. If experienced developers are available, they can suggest which classes to read first, helping new developers to quickly grasp the system's most fundamental concepts. In practice, however, experienced developers often are no longer available. In these cases, the set of most important classes must be reverse engineered. This paper presents a thorough analysis of using different network analysis metrics on dependency graphs to retrieve central classes. An empirical study on four open source projects evaluates the results based on a survey among the systems' core developers. It demonstrates that the algorithmic results can compete with the suggestions of experienced developers.**

I. Introduction

Over the life-time of a software system, the core development team often changes. In many cases, *e. g.*, when outsourcing providers are exchanged, complete hand-overs to different teams occur. However, before a new developer can successfully work on a specific task, he needs to gain a general understanding of the system. As becoming familiar with an unknown system at once is not feasible, central classes provide a starting point for task-independent system understanding.

For a new developer, it is not obvious which classes are the best entry point. Thus, as part of knowledge transfer to new team members, experienced developers typically recommend certain classes to focus on first before assigning specific tasks to the new team member. However, from our industrial experience, there are many cases in which experienced developers are not available when taking over the software. Reasons include switching development of a system to a new subcontractor without transition time or loss of core developers due to fluctuation. In such cases, missing knowledge of experienced developers needs to be reverse engineered.

A possible approach to a recommendation algorithm for central classes is provided by the field of network analysis, where numerous metrics have been proposed to define the centrality of a node in a graph. Applying these metrics on a graph representation of a system, *e. g.*, a dependency graph, provides a ranking of the system elements (such as classes). How well these metrics capture the notion of an experienced developer, who recommends classes as a starting point for system understanding, however, is an open research question.

Research problem. The goal of our work is to provide new developers with a list of the classes that are most important when getting to know a new system independent from a specific task. We seek an automated approach to calculate the centrality of each class in a system. The algorithm should weight centrality similar to an experienced human developer.

Contribution. The paper compares various algorithmic approaches based on network analysis of dependency graphs: With the help of a case study, we provide a thorough and systematic analysis of existing centrality metrics and evaluate their usefulness in practice. The study comprises four open source projects written in Java. For each project, several core developers named the most important classes of their system. Individual rankings of the developers were used to evaluate the results of the centrality metrics. The study shows that the algorithmic results can compete with recommendations of developer groups, and are even better than recommendations of single developers, as the algorithm balances out individual developer preferences.

II. Related Work

Previous work ranks software artifacts, predicts defects and maintenance effort based on network analysis and also detects central classes with non-graph-based approaches.

A. Ranking Software Artifacts

Perin et al. [1] use PageRank for ranking software artifacts in order to list search results in reengineering platforms. The authors rank classes of the Pharo Smalltalk system based on a dependency graph representing class-inheritance and class-references. For evaluation, they extract the class names mentioned in an introductory book and compare them to the PageRank ordering. The authors also rank code entities in Moose, considering class inheritance, class references, method invocations and attribute access. The paper contains a listing of the top-5-ranked classes for several test cases, but an evaluation of the accuracy is missing. Perin et al. claim: "The authors of the respective Smalltalk frameworks confirmed the correctness of the results, although they also reported some other core classes that did not show up in the top 5." In contrast to this vague statement, we present a case study with a detailed evaluation, which includes several different centrality indices.

B. Using Network Analysis for Prediction

Several papers have used network analysis for prediction: Zimmermann and Nagappan [2] 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 successfully find central, and therefore critical, escrow binaries on the Windows Server 2003, and outperform previous complexity metrics. Besides predicting bug severity, maintenance effort, and defect-prone releases, Bhattacharya et al. [3] also use source-code based graph metrics to reveal differences and similarities in structure and evolution of software systems. Kpodjedo and Ricca [4] propose a recommendation system for software testers, using evolution cost and PageRank. 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 should be tested thoroughly. Compared to these three papers, we evaluate the use of similar centrality indices. However, we do not use network analysis for prediction, but for finding a useful entry point to a software system.

C. Recommending Software Artifacts

Others use completely different approaches for finding central classes: Čubranić and Murphy [5] recommend pertinent software development artifacts for newcomers to open-source software projects based on an implicit group memory containing information about source versions, bugs, archived electronic communication, and web documents. The authors tackle the same problem of introducing a new team member when mentoring is not possible (such as open source communities being geographically separated and located across time zones), but they focus on recommending existing artifacts relevant to a specific task. In contrast, our work contributes to initial task-independent system understanding prior to the first task assignment. The authors state that newcomers can have trouble determining the relevance of recommendations when the recommendation list is long. Prior task-independent system understanding can potentially help solving this problem.

Storey et al. [6] recommend related files for a given task as a developer navigates the software system. The work is based on the assumption that navigation patterns reveal relatedness between files. In contrast to our work, the authors focus on task dependency. Interestingly, they lack a meaningful ranking strategy when displaying file recommendations. Currently, they rank based on occurrence time, considering a combination of recency and frequency for future work. It might be challenging to determine whether our approach contributes to this aspect of their work.

Further, [7] and [8] recommend relevant files for modification tasks based on mining of change patterns in change history. Again, our work should be seen as a prior step for task-independent system understanding before working on modification tasks.

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. In directed graphs, edge $e = (v_1, v_2)$ connects vertex v_1 to vertex v_2, if v_1 depends on v_2. In undirected ones, 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. The top ten recommendation set includes the top ten classes of the algorithm's ranking.

(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 software systems according to their importance. The top-ranked classes serve as a recommendation for an entry point.

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. In 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 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 which subset of dependency edges achieves the highest recommendation precision. An empirical study

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

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 of itself. This means that interface I_A is implemented by only one single class A. However, I_A could be extended by more subinterfaces 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, researchers proposed a variety of different centrality measurements. In preliminary experiments we evaluated a large set of centrality indices to narrow down the choice. Thereby, simple centrality measurements such as degree-based centralities (in-degree, out-degree, degree) did not lead to promising results. We choose betweenness centrality [9], PageRank [10], PageRank with priors [11], HITS [12], HITS with priors [11], and Markov [11] for further experiments. Additionally, we introduce a hierarchical flow model.

Betweenness. The shortest-path betweenness centrality of a node v is calculated as $c_B(v) = \sum_{s \neq v \in V} \sum_{t \neq v \in V} \delta_{st}(v)$ 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 [9]. 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. For more detailed information about the algorithm, see [12], [11]. 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 [11]. 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 in network analysis, we also designed a hierarchical flow model. The flow model is specifically designed for the context of software systems and models information flow through the system. 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: Any non-empty set of all edges between classes/interfaces of two packages defines one single edge between the two corresponding package nodes. Packages in Java are considered to have a tree-like structure so that subpackages are aggregated to the package on the next higher level in the tree. The final level of aggregation can be tuned as an input parameter.

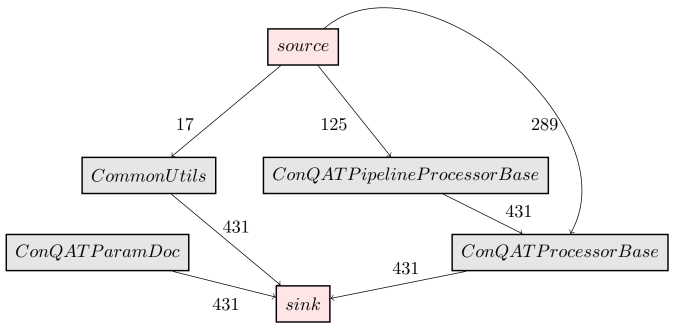

Fig. 1. Flow model of org.conqat.engine.commons

The centrality index 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 set V consisting of one vertex for each class/interface of the package. In addition, the graph contains an artificial source and an artificial sink vertex, which represent the rest of the system:

$$G_F = (V \cup \{source, sink\}, E)$$

with

$$\begin{aligned} E = \{(v_1, v_2)|\ & v_1 \text{ depends on } v_2 \wedge v_1 \in V \wedge v_2 \in V\} \\ \cup\ \{(source, v_2)|\ & v_2 \text{ has an incoming edge from a node} \\ & \text{outside the package}\} \\ \cup\ \{(v_1, sink)|\ & v_1 \text{ has an outgoing edge to a node} \\ & \text{outside the package}\} \end{aligned}$$

For example, let v be a class within the package and x and y a class outside the package. Further, let x call a method from v and y call a different method from v. Both dependencies will result in a single edge $e = (sink, v)$. 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:

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

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

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

TABLE I
RANKING ON ORG.CONQAT.ENGINE.COMMONS

Class	**rank**
ConQATProcessorBase	414
ConQATPipelineProcessorBase	125
CommonUtils	17
ConQATParamDoc	0

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 and all incident edges are removed. The maximum flow is calculated with the Edmonds-Karp algorithm [13]. 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 of each vertex 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 for evaluation, and concludes with a suggestion for the most useful recommendation algorithm.

A. Research Questions

The following questions guided the design of the case study. Questions 1-4 investigate the set up of the approach, whereas questions 5 and 6 evaluate the best set up found.

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 developers' opinions per project.

RQ6: How much better is the algorithm compared to a trivial approach? We compare our approach to a trivial approach, which is neither based on a centrality index nor on the system dependencies. We choose the size of a class in lines of code as a trivial measurement and build the recommendation set based on the largest classes of the system.

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 answered the following question:

> *Assuming a new developer who does not have previous knowledge about your software system: What are the 10 most important classes of your system, which you would first suggest him to look at?*

Table III shows the number of developers of each project who replied to the survey. For ConQAT all four developers were core developers, who have participated in the development of the system from its beginning. The two jEdit developers are both registered as administrators of the source forge project and commit on a daily basis. For voTUM two experienced core developers answered as well as one student. The three jMol study participants were the project leader since 2007, a core developer since 2009 and one additional developer. Hence, we assume that the developers have enough experience to provide a meaningful evaluation of our algorithm.

For research questions RQ1 - RQ4, we evaluate the results based on the recommendation precision, calculated over the 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 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 with the help of preliminary experiments, which priors are best to use. Based on that conclusion, Table IV shows the priors used for the case study.

TABLE III
OPEN SOURCE DEVELOPERS PARTICIPATING IN THE STUDY

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

[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 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

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. 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, the overall ranking as mentioned above suits our requirements best in terms of recommendation precision.

VI. EXPERIMENTS & RESULTS

RQ1. To investigate the influence of the priors, we compare the results of using different prior test sets for ConQAT Engine, as shown in Table V. The class Driver.java contains a main method and is the entry point of the system. ConQAT-ProcessorBase 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

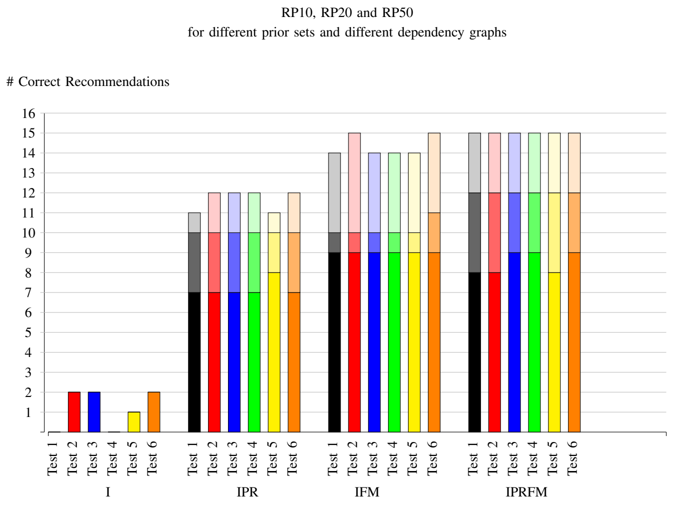

Fig. 2. Influence of choosing different priors for Markov centrality, run on ConQAT Engine

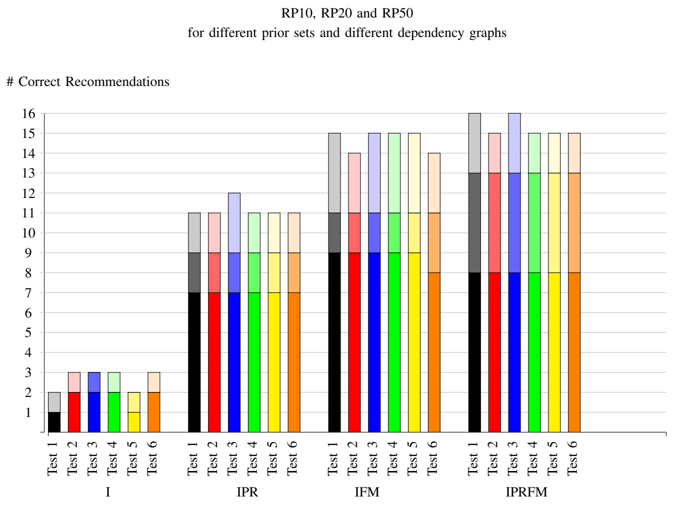

Fig. 3. Influence of choosing different priors for PageRank, run on ConQAT Engine

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.

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

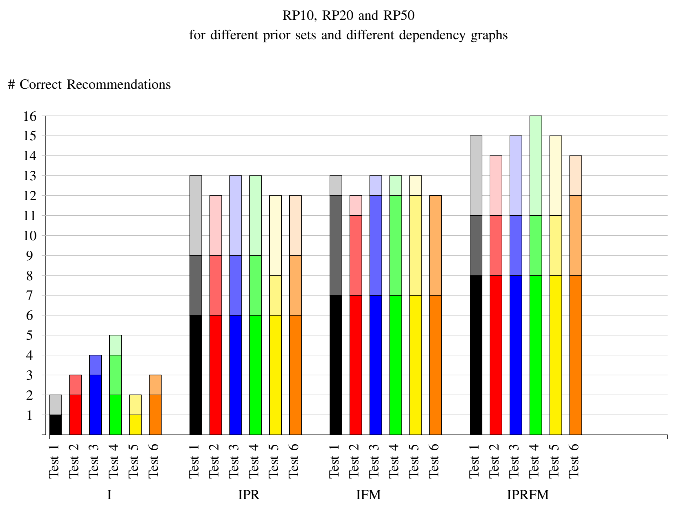

Fig. 4. Influence of choosing different priors for HITS, run on ConQAT Engine

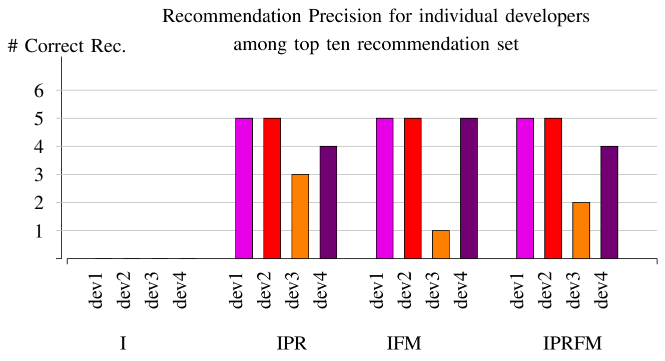

Fig. 5. Recommendation precision of individual developers of ConQAT Engine

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 RP10, RP20 and RP50 values. 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 human recommendation on central classes can differ. For each project Table X (middle column) shows the intersection size 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 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 X (right column) shows the recommendation precision RP10, RP20, and RP50.

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 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. 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 directed graphs. However, in most cases, the betweenness-

TABLE X
Size of the intersection set of developers' opinions and results of the trivial approach by measuring the size of a class

Project	intersection	RP10, RP20, RP50
ConQAT Engine	3	$\frac{2}{10}, \frac{2}{20}, \frac{3}{50}$
jEdit	3	$\frac{4}{10}, \frac{7}{20}, \frac{11}{50}$
jMol	1	$\frac{7}{10}, \frac{7}{20}, \frac{12}{50}$
voTUM	1	$\frac{2}{10}, \frac{4}{20}, \frac{7}{50}$

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	$\frac{4}{10},\frac{6}{20},\frac{13}{50}$	$\frac{2}{10},\frac{5}{20},\frac{9}{50}$	$\frac{4}{10},\frac{6}{20},\frac{12}{50}$	$\frac{5}{10},\frac{8}{20},\frac{12}{50}$	$\frac{4}{10},\frac{9}{20},\frac{15}{50}$	$\frac{2}{10},\frac{4}{20},\frac{9}{50}$	$\frac{4}{10},\frac{5}{20},\frac{8}{50}$

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 useful for a centrality recommendation system. This is in contrast to the results of [2], 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 for 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. It shows only the results of PageRank, 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 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.

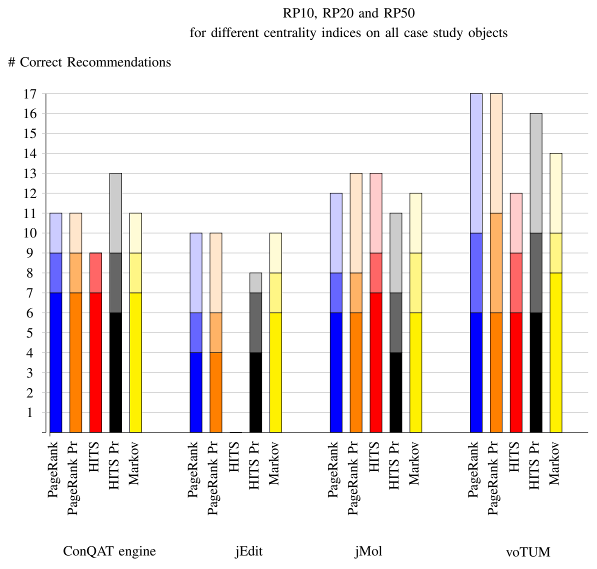

Fig. 6. Results of applying different centrality indices on the IPR undirected dependency graph

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. In many cases, both algorithms end up with the same recommendation set, just in slightly different orders. A comparison between the two recommendation sets can be found in [14]. This observation is in accordance to the results of White and Smyth [11], who evaluate 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%. Calculating the precision based on the union of all developers' opinions as in RQ1-4 leads to much better results, with a precision up to 90%. 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.

RQ6: How much better is the algorithm compared to a trivial approach? We compare the trivial algorithm with our one, using Markov on the undirected IPR graph, and show the results in Table XI. For the three projects ConQAT, 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 we achieve 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. In general, the class size in jMol is much higher than in the other projects. It seems that over time the central classes grew into god classes. Hence, the biggest classes match the ones named by the developers.

TABLE XI
CORRECT RECOMMENDATIONS AMONG THE TOP TEN OF THE TRIVIAL APPROACH COMPARED TO MARKOV ON THE UNDIRECTED IPR DEPENDENCY GRAPH.

Project	trivial	Markov
ConQAT Engine	2	7
jEdit	4	6
jMol	7	6
voTUM	2	8

VIII. THREATS TO VALIDITY

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 projects. However, they were chosen from different domains so that they represent a large area of software applications. As we compare our algorithm to the opinions of humans, the experience of the case study participants influences the results. As mainly core developers responded to the survey, we believe that their opinion constitutes a meaningful foundation for our evaluation. For knowledge transfer, there is currently no other possibility than relying on the opinion of core developers. We are aware of the limitation of this approach with respect to the size of the project. Once the project's sizes exceeds a certain limit, no developer is able to give a complete overview of the system. Hence for very big systems, there is no comparison for our algorithm. However, for such large systems, knowledge will be transferred for each subsystem individually. Consequently, for smaller subsystems, which are within human grasp, our algorithm can be applied and validated.

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 more likely for our algorithm to achieve a higher precision. One could argue that a large enough number of participants will result in a precision of 100%. 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 to the ConQAT developers. Without exception, they commonly agreed that the recommended classes are central. Therefore we believe that our algorithm does produce very useful results.

One could argue that evaluating the precision only is not sufficient, because the recall of the algorithm is not taken into account. However, for the purpose of knowledge transfer, we validated that the recommendations of the algorithm are in accordance to the developers' opinions and, furthermore, that the algorithm is less biased towards a specific part of the system than a single developer. Limited by a small number of participants, the recall could not be evaluated appropriately. If more developers' opinions were available, the recall could have been determined based on inter-rater agreement. We consider this to be an interesting and challenging task for future work.

We conducted some preliminary experiments to narrow down the parameter space of the case study. 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 they did not affect the results.

IX. CONCLUSION AND FUTURE WORK

This paper has shown that using network analysis on dependency graphs successfully retrieves important classes of a system and results in a recommendation which can compete with recommendations given by core developers. The algorithmic results are even better than the recommendation of a single developer, because they are less biased towards personal preferences. 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 including information about inheritance, parameter and return dependencies. Using the Markov centrality leads to the best results, with a precision between 60% and 80% in the top ten recommendation set.

For future work, confirming those results on a larger data base and extending the evaluation to industry software system will be challenging. Evaluating the recall of the algorithm based on inter-rater agreement constitutes an interesting task. We also plan on evaluating the algorithm on software projects which were written C/C++ or C#.

REFERENCES

[1] F. Perin, L. Renggli, and J. Ressia, "Ranking software artifacts," in *Workshop on FAMIX and Moose in Reengineering, ICSM '10*, 2010.
[2] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *ICSE '08*, 2008.
[3] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *ICSE '12*, 2012.
[4] S. Kpodjedo, F. Ricca, P. Galinier, and G. Antoniol, "Not all classes are created equal: toward a recommendation system for focusing testing," in *RSSE '08*, 2008.
[5] D. Čubranić and G. C. Murphy, "Hipikat: recommending pertinent software development artifacts," in *ICSE '03*, 2003.
[6] J. Singer, "Navtracks: Supporting navigation in software maintenance," in *ICSM '05*, 2005.
[7] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Trans. Softw. Eng.*, vol. 30, no. 9, Sep. 2004.
[8] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *ICSE '04*, 2004.
[9] U. Brandes and T. Erlebach, *Network analysis: methodological foundations*. Springer, 2005.
[10] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," in *COMPUTER NETWORKS AND ISDN SYSTEMS*, 1998.
[11] S. White and P. Smyth, "Algorithms for estimating relative importance in networks," in *KDD '03*, 2003.
[12] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *J. ACM*, vol. 46, no. 5, pp. 604–632, Sep. 1999.
[13] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, no. 2, 1972.
[14] D. Steidl, "Using network analysis for recommendation of central software classes," Tech. Rep., 2012. [Online]. Available: http://mediatum.ub.tum.de/?id=1108182