Design implications for task-specific search utilities for retrieval and re-engineering of code

Iqbal, R. , Grzywaczewski, A. , Halloran, J. , Doctor, F. and Iqbal, K.

Original citation & hyperlink:
Iqbal, R. , Grzywaczewski, A. , Halloran, J. , Doctor, F. and Iqbal, K. (2015) Design implications for task-specific search utilities for retrieval and re-engineering of code. Enterprise Information Systems, volume (In Press)
http://dx.doi.org/10.1080/17517575.2015.1086494

DOI 10.1080/17517575.2015.1086494
ISSN 1751-7575
ESSN 1751-7583

Publisher: Taylor and Francis

This is an Accepted Manuscript of an article published by Taylor & Francis in Enterprise Information Systems on 9th October 2015, available online: http://www.tandfonline.com/10.1080/17517575.2015.1086494

Design Implications for Task-Specific Search Utilities for Retrieval and Reengineering of Code

Author A, Author B, Author C, Author D
Affiliation,
Country,
Email Addresses

Abstract: The importance of information retrieval systems is unquestionable in the modern society and both individuals as well as enterprises recognise the benefits of being able to find information effectively. Current code focused information retrieval systems such as Google Code Search, Codeplex or Koders produce results based on specific keywords and therefore they do not take into account developers' context such as development language, technology/framework, goal of the project, project complexity and maturity or developer's domain expertise. They also impose additional cognitive burden on users in switching between different interfaces and clicking through to find the relevant code. Hence, they are not used by software developers. In this paper, we discuss how software engineers interact with information and general purpose information retrieval systems (e.g. Google, Yahoo) and investigate to what extent domain-specific search and recommendation utilities can be developed in order to support their work related activities. Based on our user studies, we found that software engineers followed many identifiable and repeatable work tasks and behaviours. These behaviours can be used to develop implicit relevance feedback based systems. Based on our results, we discuss the implications for the development of task-specific search and collaborative recommendation utilities embedded with the Google standard search engine and Microsoft IntelliSense for retrieval and reengineering of code. We have implemented a prototype of the proposed collaborative recommendation system which was evaluated in a controlled environment simulating the real world situation of professional software engineers. The evaluation has achieved promising initial results on the precision and recall performance of the system.

Keywords- Personalised information retrieval, opportunistic software development, development by example, retention actions, copy and paste, pseudo-relevance, software development, information seeking behaviour, implicit relevance feedback

I. INTRODUCTION

The last decade or so has seen a significant change in the software development process. Software developers no longer work from beginning to end on a software solution designing and incorporating their own code. Instead they use already existing code and modify it to construct a suitable solution. Often the coding process itself is tightly controlled by existing frameworks which provide an integrated development environment (IDE). This framework is ideally suited for adding existing code from different sources both from the Internet as well as from the in-house library of the organisation. Usually these IDE's make it easy to code, test and implement parts of the software iteratively as well as maintaining version control of the changes. This becomes a more cost-effective means to construct and implement software solutions as well as speeding up the whole process, compared with traditional software development. It also prevents duplication of effort constructing code which is used again and again in every software solution (e.g. memory management, database I/O, etc.) as well as making it easier to modify the solutions to meet changing business needs.

The amount of digital information available on the Internet and various Intranets often causes information overload, significantly increasing the amount of time and cognitive resource needed to acquire relevant and accurate information. When searching for information to address complex problems, users spend a significant amount of time clicking through search-query results and reformulating the search query if the results are not satisfactory. This is a tedious and challenging task and has a negative impact on users' performance. Also, once the task that they are performing is complete and the results of their effort are embedded into the source code this knowledge is very rarely shared with the software development community. As a consequence other software developers when faced with the same or similar problem have to solve it by themselves wasting a significant amount of time and resource on duplication. The results illustrated in Table 1 show the average time software developers spend on information retrieval tasks.

TABLE I. AVERAGE TIME SOFTWARE DEVELOPERS SPEND ON INFORMATION RETRIEVAL / COMMUNICATION

Key References	From	To	Participants	Research method
Liu et al. [1]	60%	60%	9	Questionnaire/ interview

Brandt et al. [2]	14%	41%	20 students	Observation
Singer et al. [3]	24%	45%	13	Questionnaire
King and Griffiths [4]	20%	80%	-	-
Freund et al. [5]	20%	30%	14 software engineers / consultants	Interviews
LaToza et al. [6]	5%	20%	187 software developers	Survey
Perry et al. [7]	16%	16%	13	Diary study
Tenopir and King [8]; King et al. [9]	40%	66%		Literature review

Current code focused search systems [10-15] used keywords based algorithms and they do not take into account developers' context such as development language, technology/framework, goal of the project, project complexity and maturity, or developer's domain expertise. Mostly, software developers use general purpose search engines [16]. In order to produce accurate search results according to a user's current needs it is necessary to investigate search engine personalization for source codes optimization purposes. One of many personalisation techniques investigated in the literature is relevance feedback. Relevance feedback is based on knowledge of how relevant the particular piece of information (document) is to the user and how its content can be reused in order to find documents that are similar. These have a very high probability of relevance. There are two techniques for relevance feedback: explicit and implicit. In explicit feedback, users explicitly mark the documents as relevant or not relevant, whereas in implicit feedback, relevance is estimated based on behavioural observation such as reading time, click count, etc. A hybrid approach combines both techniques [17]

Unfortunately, even though explicit relevance feedback produces very good results in experimental settings, it is difficult to implement in the real world because users, especially under cognitive strain, are not willing to provide search systems with relevance information. Implicit feedback has a number of advantages as compared to explicit feedback: (i) the user does not have to actively provide the information to the system, which is the main limitation of explicit feedback; (ii) implicit feedback implementations gather a large amount of behavioral information especially when the observation is performed on the client side. This provides rich behavioral information and therefore can be used not only to adjust search results but also information presentation mechanisms and supportive search tools. Most of the research in implicit feedback aims at interpreting different behavioral factors as indicators of relevance. The observable factors can be classified as follows [18]: (i) the minimum scope at which the factor can be observed: whether it can be observed at the level of the information segment, object or class; (ii) the behavior category: whether the behavior is focused on examination, retention, referencing, annotating or creating the information.

Modern enterprise systems should be able to meet dynamic user needs, share information with other systems or services [50, 51], be self-resilient [52, 53], adaptive [54] and secure [55]. In this paper, we present a novel approach for the development of task-specific search utilities for retrieval and reengineering of code. The approach has been applied to the domain of software development to address the information needs of software developers by using implicit feedback with a special focus on retention actions. Based on our effective and rigorous analysis of user information behavior, we have developed recommendation utilities which are embedded with the Google standard search engine and Microsoft IntelliSense for retrieval and reengineering of code. The rest of the paper is organised as follows. Section II discusses the use of retention actions and opportunistic programming. Section III discusses our user studies consisting of a questionnaire and automated observation of user interactions with the browser and software development environment. Section IV presents our results based on a feasibility study and automated observation of user behaviour. Section V discusses the development of task-specific search utilities embedded within the Google standard search engine and Microsoft IntelliSense for retrieval and reengineering of code. Section VI discusses our evaluation results based on Precision and Recall. Section VII concludes the paper and provides outlines for our future work.

II. THE USE OF RETENTION ACTIONS

As discussed above it is widely recognised that software developers frequently use online resources to solve their development problems. These resources are used in a variety of ways. Software developers engage in a number of retention actions aimed at maintaining the retrieved content or even directly reusing it in the development process. This phenomenon is widely understood in the literature and is frequently referred to as "opportunistic programming" [19], "example centric programming" [20] or "programming by example". The retention actions as understood in this research include actions such as:

- Direct reuse of code through copy and pasting into the project.
- Code file or library reuse through introduction into the development project.
- Library or platform reuse through download and installation.
- Other forms of direct information retention such as saving, printing or bookmarking web pages.
- Indirect forms of information retention manifested through the information note taking or code development based on the information retrieved from online resources.

The use of retention actions as a source of implicit relevance feedback has a long history [21, 22, 23, 24, 25, 26]. Various retention actions used during generic IR were investigated in the literature including:

- organisation of personal files, organisation of email, organisation of web information [27].
- saving a reference, saving an information object, printing, deleting [21].
- bookmarking, purchasing, subscribing [28].
- downloading a file or a resource [29].
- Subscribing to RSS feeds [30].
- Download of the online multimedia content [31].

Success in using retention actions for relevance prediction varies. Significantly different levels of relevance estimation accuracy are achieved by commercial and research focused implementations. Retention actions strongly expressing user intent such as purchasing or subscription are widely used in e-commerce systems (such as Amazon, E-Bay, etc.). Everyday online retention actions have a very selective nature though and are not observed frequently enough to significantly affect relevance estimation accuracy [32, 33].

The problem of retention action use during software development is significantly different. Developers engage in retention actions very frequently and the reuse of code related resources has a strong correlation to relevance [30, 19, 31, 34, 35, 36]. Software developers understand that the use of retention actions accelerates their work, allows them to distribute their cognitive resources and improves the maintainability of code [35]. Moreover the use of retention actions is, in the majority of cases, not perceived negatively and is socially appropriate.

This does not mean that all retention actions can always be used as a strong indicator of relevance. Software developers interact with information of different granularity (i.e. length of the copied code) depending on the problem at hand and their perception of information retention changes significantly with this factor. Similarly the stage of the project and the role of the developer in the project changes the way retention actions are perceived and used. This significantly affects the type of information objects retrieved, as well as retrieval frequency.

III. USER STUDY

A. *Feasibility Study*

We conducted a user study in which 14 experienced professional software developers whose main role was coding and testing software participated. The goal of our study was to concentrate on the copy and paste functions used by the software developer.

- Firstly whether in some way the copy and paste activity reflects relevance and leads to retention.
- Secondly whether the copy and paste frequency could be used to build a suitable knowledge base or profile of the developer.
- Thirdly to study the nature of the copy and pasted information (e.g. instruction, group of instructions, whole procedure, etc.).
- Lastly how much the type of copy and paste activity affects successful outcome and whether it could be used as an indicator of relevance?

B. *Automated Observation*

The second phase of the user study was a long-term observation of the behaviour of the software developers when engaged in the development task. This was achieved by incorporating an "add-in" into their development environment which collected information from the coding activity performed by the developer as shown in Figure 1. A report was produced from this study which was initiated when the developer closed the document on which they were working. 665 reports were produced which were considered non-trivial (e.g. contained copy/paste functions and the session was greater than 20 seconds.

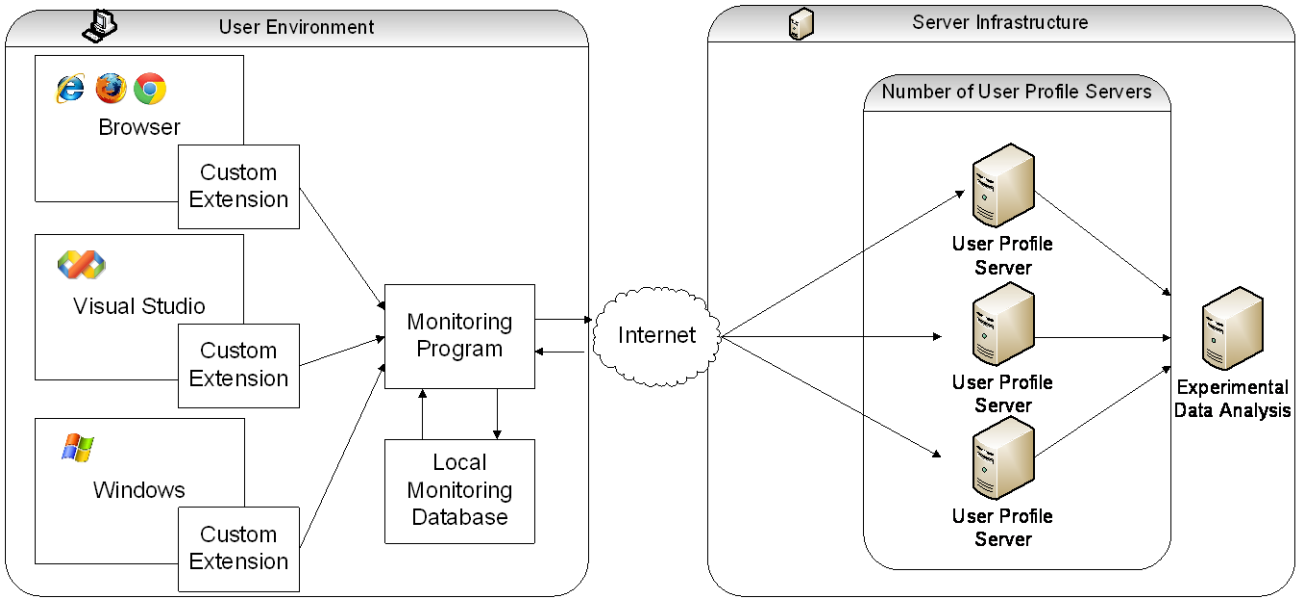

Figure 1. Software Architecture.

IV. RESULTS

A. *Feasibility Study*

The results of the feasibility study show that software developers interact with the Internet or Intranet in average 5.15 times per day (number of sessions) to solve their work related problem. This interaction also include 1.97 times per day to use Internet/Intranet to look for code documentation and code examples. Software developers reuse the code snippets from external sources to rework in their project on average 0.37 times per day out of 5.15. Out of 14 software developers, 12 (85.71%) of them copy frequently the code snippet or code examples which shows copy and paste as one of the main retention actions indicating content relevance. This confirms the observation that the developers engage most frequently in opportunistic practices of small block or sub system level elements of code. This is consistent with the observations from the literature illustrating that the reuse of small sections of code by software developers is a frequent replacement for note taking and documentation review and is used in order to decrease/distribute the cognitive load associated with work.

It was also found through the user study that software developers deliberately avoid the code documentation up to the point when they are forced to use it due to the lack of other resources. Developers are interested in retrieving even loosely related big sections of code and in an iterative manner reworking them up to the point where the copied segment meets their needs. As discussed earlier this practice is referred in the literature as the 'Development by Example' or 'Example-centric Programming' [31].

Overall the feasibility study validated a number of key questions related to developers' retention actions and IR:

- It confirmed that developers interact with information resources very frequently.
- It confirmed that developers engage in retention actions very frequently.
- It confirmed that the retention actions are correlated with information relevance.
- It illustrated that the main retention actions used during the development process are:
  - Copy and pasting code.
  - Using the information to work on the problem directly whilst reading the information.
- It illustrated that the following retention actions are rarely observed during the development process:
  - Printing.
  - Bookmarking, saving and emailing.
  - Discussing the findings with colleagues (directly or indirectly).
  - Any form of note taking.
  - In neither case was the act of subscription, framework or library purchase mentioned.
  - Blog or other content publication.

- It showed that the most frequent information objects which are retrieved on a day to day basis are small, block or sub-system levels sections of code (e.g. code snippets or single methods / classes).
- It showed that users are not willing to use other forms of code related information and refer to it only if the primary information is not available.
- It also illustrated that the use of retention actions is highly driven by the project phase, task and user role in the development process.

The feasibility study laid the foundations for the automated user study and elucidated the key factors affecting the observed behaviour as well as the key information objects and retention actions that should be included in further investigation.

B. Automated observation of user behaviour

We recorded a total of 1296 non-trivial copy and pastes out of which 379 originated from outside the development environment.

1) Copy and Paste general statistics

The copy and paste data source selection during the development activity was captured through the operating system service installed on the user machine. The monitoring program observed all of the copy and paste actions in the operating system and if the content was pasted to the development environment it recorded the source, destination and the copy and pasted content. Figure 2 shows the distribution of the recorded programs/processes.

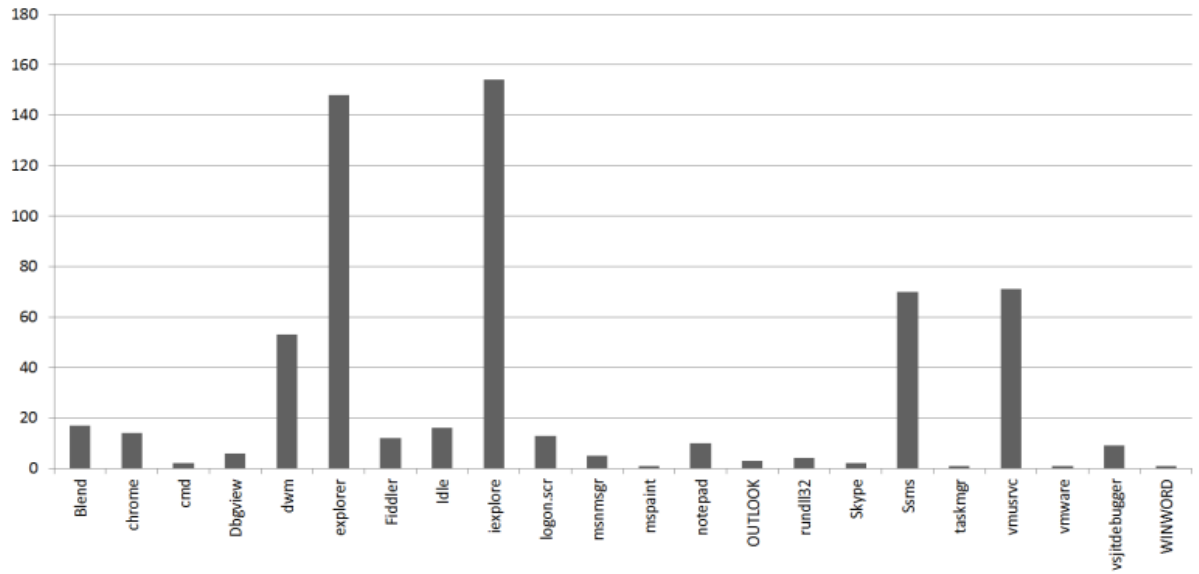

Figure 2 Copy and Paste process distribution

Figure 2 provides an overview of the processes used as a source of information which was pasted into the development environment. Due to the policy imposed by the administration and the fact that IE was the main development target in this experiment it is the most frequently used browser and a source of copy and paste as well as focus change action. Figure 2 does not show the interaction within the development environment due to a significant count of those interactions (917 non-trivial interactions), which obstructs the scale of the distribution. Tools such as “Blend”, “cmd”, “Dbgview”, “Fiddler”, “Ssms” or “vsjitdebugger” are complementary development tools but were not included in the direct observation. The “Vmusrvc” process is a part of Microsoft Virtual PC which is used extensively during code development for hosting the development and test environments locally on the machine. Due to the way developers work, the majority of their development environments are virtualized and frequently distributed across multiple virtual machines on the same physical machine. This creates a need for frequent information movement between the environments as well as distribution of the tools used by the developer across different virtual machines. As a consequence it was essential for the experiment to monitor all of the user virtual environments and also monitor the data flow between them. The content of the data copied from the “Vmusrvc” was manually annotated in terms of its source based on the information content.

2) Complexity analysis

In order to assess the correlation between the problem complexity and retention actions further analysis was carried out based on the copy and pasted content. The data captured during the experiment was processed and each copy and paste event was analysed in terms of its content length and subjective complexity measured on a three point scale:

- 0 - The content is encoded with complexity of zero if it does not carry any logic (e.g. class names).
- 1- The content is encoded with complexity of one if the logic is copied but is of marginal complexity (e.g. data base connection snippets)
- 2 – The content is encoded with complexity of two if complex logic is copied.

This annotation process was simple and allowed for a clear distinction between content of varying complexity, reducing ambiguity. It also allowed for statistical analysis of the correlation between complexity and the copy and paste activity, as well as the amount of time the user spent working on the content.

On the basis of Levene's test, we were unable to reject the hypothesis regarding the homogeneity of variance ($F(2,1143) = .982, p = .375$). The Kruskal-Wallis test shows the existence of a statistically significant difference in complexity in the three distinct populations ($\chi^2(2, N = 1146) = 18.782, p = .000$). A post hoc paired comparison following the Kruskal-Wallis test using Mann-Whitney identified that with the increase of copy and pasted code complexity both the number of copy and pastes increases in the project; and also that the amount of time the user spends on project interaction increases.

The second analysis focuses on the impact of the information source program on the complexity of the copied content, the copy and paste frequency and work time. Comparison of the complexity of the interaction when the information was sourced from the browser, with the complexity when the information was sourced exclusively from the development environment, revealed a significantly higher complexity of browser based interactions ($U(N = 881) = 27177, p < .01$). Similarly, contrasting the complexity of development environment interactions with all other monitored copy and paste interactions revealed significantly higher complexity of other interactions ($U(N = 1066) = 92888, p < .005$).

Further analysis of data length and its distribution across different sources of copy and pasting shows that the three distributions are non-normal and their variance is not homogenous. After transformation with a natural logarithm the hypothesis of data log normality cannot be accepted (based on Kolmogorov-Smirnof test). The assumption of data variance homogeneity based on Levene's test cannot be accepted either, regardless of logarithmic data transformation ($F(2,1143) = 23.466, p = .000$). The data copied from the browser was significantly longer from the data copied from other sources and in the light of the above results the difference can be considered significant.

Our complexity related results indicate that the presence of copy and pasting from the browser is an indicator of higher problem complexity. This is also shown by a much longer average active time across sessions that involve copy and pasting from the browser. Moreover the data copied from the browser has higher complexity and is longer.

3) Copy and Paste Frequency Analysis

We carried out the analysis of user copy and paste frequency. The analysis reveals the log normal nature of the data distribution. The Kolmogorov-Smirnov test confirming the assumption of log-normality was carried out for three different use cases:

- Data copied from the browser.
- Data copied from the development environment or other development tools.
- Data copied from other sources.

Levene's test of homogeneity of variance suggests that the three populations have significantly different variances ($F(2,816) = 5.548, p = .004$) and as a consequence belong to three different populations.

The frequency data was then divided into two separate time intervals in order to assess the stability of copy and pasting behaviour in time. The Kolmogorov-Smirnof test did not support the hypothesis regarding the normal distribution of the copy and paste frequency data (p=.000). The Levene's test supported the hypothesis of variance homogeneity between the samples ($F(1,817) = .074, p = .786$). The Kruskal-Wallis test did not allow rejection of the hypothesis regarding the similarity of the two selected populations ($\chi^2(1, N = 819) = .136, p = .713$). This illustrates that the frequency of the copy and paste activity is fairly stable in time and does not change significantly as long as the user does not change the role or task.

4) Active Time Analysis

Next the analysis of user active time in the project was carried out along with the analysis of the impact various factors had on the active time distribution. Table 5 illustrates general statistics of this analysis.

TABLE II. ACTIVE TIMES OF INTERACTION: GENERAL STATISTICS

Description	Q1	Q2	Q3	Average

Browser	40.25	133	242.25	203.69
Non-browser	25.25	87	117.25	98.40
Development	24	69	74	74.82

The analysis of the active time distribution clearly indicates its non-normal nature once again suggesting a log-normal distribution. The Kolmogorov-Smirnov test confirmed the assumption of log-normality of the captured data and the Levene's test carried out on the data transformed by natural logarithm supported the hypothesis of homogeneity of variance ($F(2,1155) = 1.547, p = .213$).

Since the assumptions of the one-way ANOVA are met by the data this test was carried out to investigate the differences between the active times based on the source of the copy and pasted information. There was a statistically significant effect of the source of copy and pasting (sourcing information from the browser, development environment and other sources) on the observed active time as determined by the one-way ANOVA ($F(2,1155) = 48.372, p = .000$).

Post hoc comparisons using the Tukey HSD test (based on the data transformed by a natural logarithm) indicated that the mean score for the active time when the information is sourced from the development environment ($\mu = 3.39, \sigma = 1.58$) was significantly different from both the browser ($\mu = 4.45, \sigma = 1.49$) and the other source condition ($\mu = 4.32, \sigma = 1.50$). However, the browser and other source condition did not significantly differ from the development environment condition. The differences between the active times of different participants were also analysed using the method discussed above and since the Levene's test could not support the hypothesis of the homogeneity of variance we must conclude that there are significant differences between the users as far as the distribution of active time is concerned.

The analysis of active time variance in the two time intervals did not allow rejection of the hypothesis of population similarity (Kolmogorov-Smirnof: $p=.000$, Levene's test: $F(1,1156) = .089, p = .76$, Kruskal-Wallis test: $\chi^2(1, N = 1158) = 1.997, p = .158$), leading to failure to identify statistically significant differences between the populations. This illustrates that similarly to the copy and paste as long as the user does not change the role in the project or task the overall development effort remains the same.

5) Copy and Pasted Content Analysis

Table 6 shows the results of the analysis of the copy and pasted text depending on the source of copy and paste. The average number of characters copy and pasted from the browser was more than two times higher than the average number of characters copied from other sources.

TABLE III. COPY AND PASTED CONTENT ANALYSIS

Description	Average
Average number of characters copied from the browser	230.05
Average number of characters copied internally in the development environment.	107
Average number of copy and pastes from the browser in a single report	2.24
Average number of copy and pastes from the browser in a single report (the reports having just one copy and paste were excluded from the calculation)	3.86
Average number of copy and pastes internally in the development environment	5
Average logic complexity copied from the browser	0.46
Average logic complexity copied from the development environment	0.20

On the other hand users copied the information from the browser much less frequently than internally in the development environment. Single copy and paste accounts for 60% of copied data, with the maximum of 8 copy and pastes in a single report (a single interaction with a source code file).

The data for development environment copy and pasting also contains a lot of isolated events with 60% of copy and pastes occurring only 1-3 times in the report (a single interaction with a source code file). The remainder of the reports contain significantly higher values ranging from 44 to 46, indicating significant amount of work carried out on the document.

6) Focus Change as an Indicator of Retention

This subsection focuses on the second behaviour reported by the participants of the feasibility study; that is, direct reuse of information. During the feasibility study participants indicated that they very frequently use the information found on the internet directly in their development environment just swapping between the content and the code. This is manifested in the operating system as the application focus change which is under the investigation in this section. Since no eye/gaze tracking solution was

used in the study due to its ethnographic nature the information regarding user focus is limited to the information about the active program/document and the interaction of users with the programs/documents. As a consequence the focus change information is very similar to the data captured while observing copy and pasting behaviour. The key difference is the fact that the data was transferred between the programs without the assistance of the computer (directly by the user). As a consequence in this analysis the exact content that the user focused on could not be taken into account.

It is hypothesised that the correlation of user active time to the focus change is similar to the correlation of the copy and pasting activity. The analysis of the distribution of the active time once again confirms its non-normal nature (Kolmogoto-Smirnov p=.000). Levene's test confirms the assumption of homogeneity of variance therefore the ($F(2,1155) = q.547, p = .213$) further non parametric analysis was carried out. The Kruskal-Wallis test showed statistically significant differences between the active times depending on the program group used for focus changes ($\chi^2(2, N = 1158) = 89.752, p = .000$). Post hoc paired comparison using Kruskal-Wallis illustrated that the active time when the focus changes does not occur or occurs in the development environment is significantly smaller than when the focus changes occur in the browser ($\chi^2(1, N = 876) = 31.972, p = .000$) or other programs ($\chi^2(1, N = 1068) = 31.972, p = .000$). Even though the mean active time for development interaction was higher for focus changes taking advantage of the browser, the differences were not statistically significant ($\chi^2(1, N = 372) = .243, p = .622$).

Very similar results can also be observed for the focus change frequency data depending on the focus change source. Similarly to the previous analysis the data shows significantly different variances depending on the focus change source with the assumption of homogenous variance rejected by Levene's test ($F(2,816) = 159.901, p = .000$). We have to conclude that the three groups are indeed significantly different.

7) Copy and Pasting and Project Building

Finally the relationship between the copy and pasting and project building was investigated:

- When the user copy and pasted the information from the browser, the build action was executed in 57.5% of cases after the paste and with the focus in the same window.
- After copy and pasting the information from the development environment, the code build took place in 11.14% of cases after the paste and with the focus in the same window.

This indicates that the user was not certain about the results of copy and pasting and used the build option to validate the correctness of the code and to further test it.

8) Summary

The user studies discussed above were designed to address the key research questions discussed in section III.

1. Is the copy and paste retention action an indicator of relevance?

   Overall the copy and paste action was indeed reported by all of the participants as an indicator of relevance. The results of our feasibility study and the automated observation were further supported by the interviews. On the other hand the perception of relevance reported by the participants is quite complex and related not only to the similarity of the code to the problem they want to solve but also the completeness of code and its usage license.

2. Does the code copy and paste occur frequently enough to support the development of Implicit Feedback Recommender Systems?

   The copy and paste action of block and sub-system components did occur frequently enough to consider its usage as a source of implicit relevance feedback with a non-trivial copy and paste occurring every 120 minutes.

3. What is the nature of the copy and pasted content?

   The code that was copy and pasted from the browser was complex and averaged in length to 230.05 characters (with values ranging from 6 to 2031 characters). This means that the users were most frequently interested in the block or sub-system level content that they could directly reuse in their solutions.

4. To what extent does the building of the code accompany the copy and paste action; and can it, along with the observation of user focus change, improve the assessment of relevance?

   The action of building the pasted code can be treated as an ultimate expression of content relevance. For the development project to build, the code has to be not only relevant to the problem at hand but also syntactically correct and consistent (in terms of variables, execution context and naming) with the rest of the code. The build action is observed very frequently during the IR process and is an integral part of the development by example process. The build event is also closely associated with focus changes, especially for more complex problems.

The results of both the feasibility analysis and automated observation show how the copy and paste action of code content from the browser can be a good indicator of relevance. The contextual association of this action in terms of observing how code

content is reused within the development environment and in context to other activities such as build actions can also be used to indicate relevance of the originating source of information to the developer.

The above results allow us to safely assume that copy and pasting information from the browser to the development environment is a strong indicator of information relevance. The results also indicate that this is especially true when copy and pasting has been followed up by project manipulation and building. This concept is a foundation for the proposed system development as it allows for creation of information chains as illustrated in Figure 3.

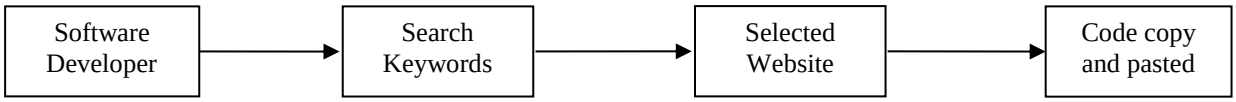

Figure 2. User flow showing stages of information seeking behaviour.

Whenever the information is copied from the browser to the development environment the user provided us with useful information:

1. The user describes the problem at hand using a human interpretable search query.
2. The user points our system to the website containing the solution to the problem described by the query.
3. The user points our system to the section of the website containing the code used for further development.

Based on the above, we developed user and search query profiles using classical collaborative techniques which is used for the development of new search interfaces and solutions for software developers.

V. IMPLEMENTATION

The proposed Collaborative Information Recommender (CIR) system is composed of a number of distinct elements as briefly discussed below and shown in Figure 3:

- Activity monitoring components: deployed to the developer's machine, these are responsible for direct observation of user interaction with information.
- User profiling and recommendation components: these form part of the web infrastructure and are deployed on a simple windows server/IIS machine.
- User interface components: these are deployed on the developer's machine and are responsible for providing the user with a search experience consistent with the existing information interaction utilities and minimizing the cognitive load required in search.

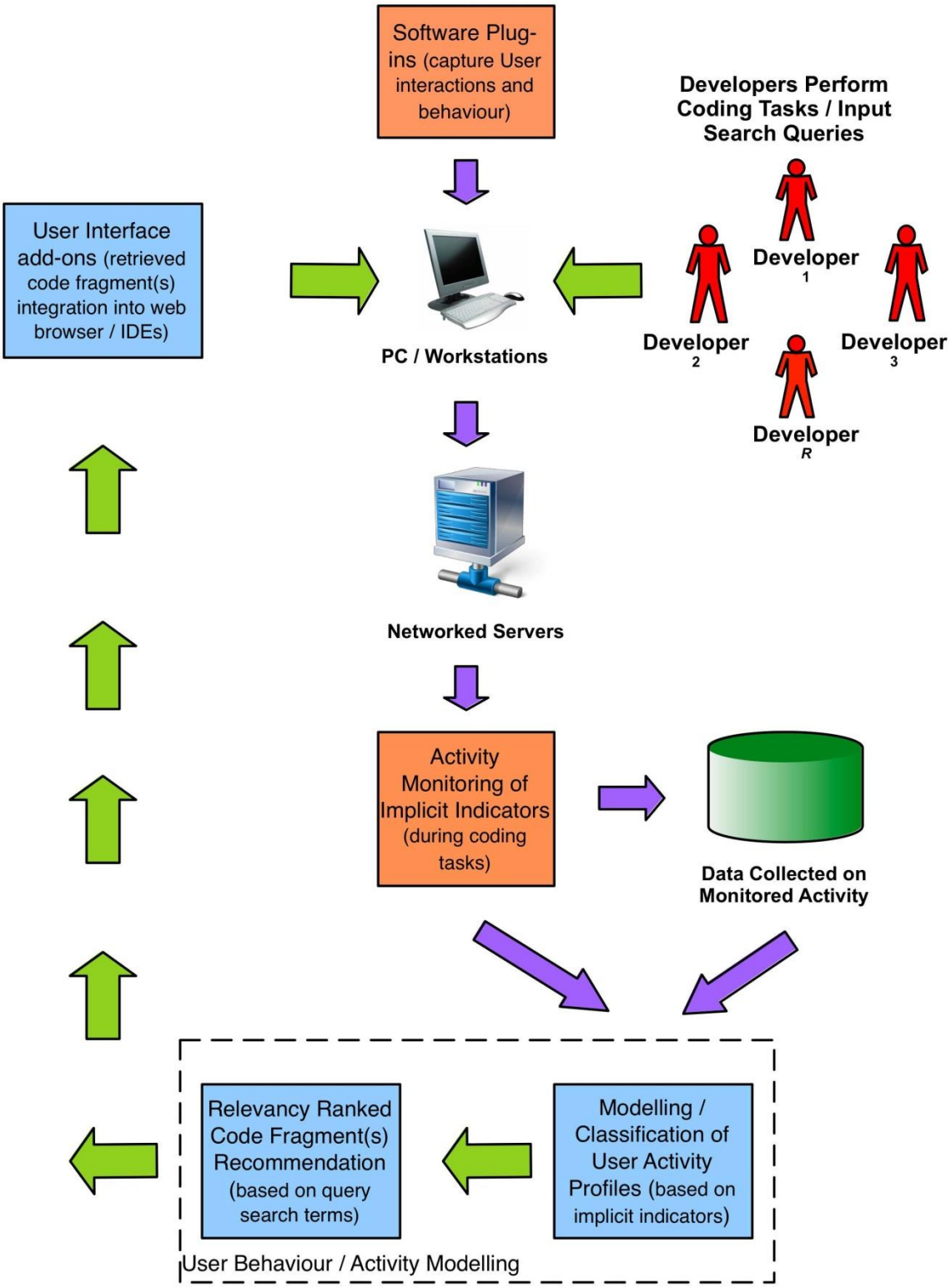

Figure 3. Flow diagram of the proposed collaborative information recommender system.

A. Activity monitoring Components

The key component of the proposed system is a set of tools responsible for user monitoring. The heart of the monitoring system is a windows service which runs in the background of the software developer's machine. The service is not directly responsible for the observation of user interaction but also provides an aggregating platform for other observation utilities distributed in the user system by the installation package. The application is responsible for caching the observation information, its compression, encryption and distribution to appropriate user server.

Within the user environment the observation service communicates with a number of plug-ins observing user interaction in different information focused programs. Currently the following observation components were implemented:

- Internet Explorer add-in allowing observation of user interaction with web content.
- Chrome add-in providing the same functionality as the above.
- FireFox add-in providing the same functionality as the above.
- Visual Studio extension allowing for the observation of the information flow in the development environment.
- Windows service allowing for observation flow between the applications through copy and pasting or focus changes.

The key advantage of carrying out the user observation on the user machine and not on the web server (through analysis of interaction logs), is that this allows the capture of very detailed behavioral information at the exact moment of time the interaction took place.

B. User Profiling and Recommendation Components

The implementation of the User Profiling and Recommendation components is carried out based on a number of servers responsible for:

- Routing the user traffic and load management (Windows based IIS server)
- Managing user profiles and data collection (Windows based IIS server)
- Data aggregation and calculation of indexes and user profiles (small Hadoop/Mahout farm)
- Recommendation and interaction with the UI extensions (Windows based IIS server)

C. User behaviour/ activity model

When the user formulates the search query and after several attempts identifies the relevant piece of code to reuse he is not only indicating that the chosen page/section of code is relevant to him but also annotating it with the search phrase (or multiple phrases if the search process was composed of multiple steps). By observing this behaviour across many developers, we were able to create a three dimensional representation of user information seeking behaviour. For example, this representation shows the relation of the user to the retrieved content and the relation between the content and the search queries.

Normally problems of this nature, so called collaborative filtering problems, are solved through various forms of matrix factorization.

$$S = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{pmatrix} \approx \begin{pmatrix} \quad \\ \quad \\ \quad \\ \quad \\ \quad \end{pmatrix} \Big(\quad \Big) \Big\}$$

Latent factors

Latent factors

Figure 4. Standard matrix factorisation approach to collaborative filtering.

As illustrated in Figure 4, once the data in the matrix is captured it then can be factorised using techniques such as Principal Component Analysis (PCA). The decomposition processes reveals latent factors pertaining to the users information seeking behaviour. This enforces a three dimensional problem representation rather than a classical two dimensional matrix format based on the relation of the user to the retrieved content and the relation between the content and the search queries, as shown in Figure 5.

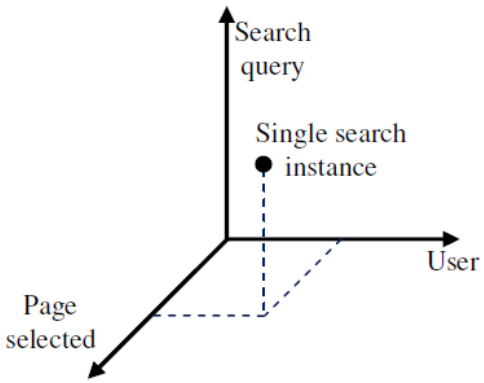

Figure 5. Three dimensional problem representation

Due to the limited capability of a search query to represent a problem and significant differences in problem implementation across multiple platforms, operating systems, web servers, programming languages or libraries, it is not accurate to recommend a solution to the problem described by the same or similar search query. It is therefore necessary to distinguish between various user groups of different experience levels, using different technologies and platforms.

The most prominent techniques to achieve this are cluster analysis-based approaches [38] such as K-means clustering [39], neural networks-based clustering algorithms such as Self-Organising Map (SOM) [40], Adaptive Resonance Theory (ART) [41], or hybrid variants such as unsupervised Snap-Drift neural network [42].

We used unsupervised Snap-Drift neural network (SDNN) [42] for determining the initial clusters on different behaviour characteristics. The SDNN is an unsupervised modal learning algorithm [41], which is trained on the data acquired from monitoring the individual developers. The SDNN can then learn to group the input patterns according to their common features and discovers clusters showing task and user-related relevancy groupings of source content. This is achieved by mapping the clusters (search query, user behavior / activity, and content) to assigned relevancy categories based on explicit content analysis with domain experts used to determine different grades of source relevancy associated to each cluster. The data points in each cluster pertaining to specific code blocks and fragments found on associated web pages are ranked to find the most representative, useful source code fragments and associated search terms. The clusters are then used to build classification models of user activity profiles that relate to high or low relevant content (related to the retrieved code fragments). An individual developer's activity can then be continually matched against these user activity profiles to identify coding related tasks. The user can then query the system for relevant code fragments which would be recommended based on the content associated to the matched activity profile and its associated search terms matching the user specified queries.

The system can continuously improve and fine tune its learnt user behavior model overtime. This can be achieved by periodically regenerating the user activity profiles based on the previously accumulated data and the new data generated from continuously-monitoring the developer's activities.

C. User Interface Components

As discussed earlier in this paper, dedicated code or software development utilities are not widely accepted by software developers [43]. The primary reason for this lack of acceptance is a very high cognitive and manual cost of having to choose different search utilities for different types of problems. Considering the low accuracy of existing solutions this cost is not balanced by the results developers obtain, which leads to search solutions being abandoned.

In order to prevent developers from leaving the service and to maximise their satisfaction, the cognitive and manual cost of engaging in the search activity using the proposed system is minimised. This is achieved primarily by embedding the proposed system into existing search solutions such as:

- Standard web search engines (see Figure 6)
- Standard development tool information support systems such as:
    - Microsoft Visual Studio IntelliSense (see Figure 7).
    - Microsoft Visual Studio Peak View (allowing the user to see related code content in the context of a current window).

In both cases the integration was possible through implementation of software add-ins, capture of the search query provided to the utility, and injection of the search results if they are identified. The prototypes of both these user interfaces are presented in Figure 7 and Figure 8.

In the above discussed case the browser extension (IE/Chrome/Firefox) is responsible for monitoring the web traffic going through the browser. When Google (or potentially any other search engine) is detected and the search query is issued by the user the query is captured and forwarded to the recommendation server. If recommendations are available for the selected search query and user group, the server responds to the extension which injects the HTML to the Google results web page as illustrated in Figure. 6.

Similarly the Visual Studio extension is responsible for monitoring user input and detecting the use of IntelliSense. In a similar fashion, the search query is captured by the extension and forwarded to the server which responds with the appropriate code recommendation. The extension is then responsible for injecting the new information recommendation to the existing IntelliSense list. This approach has a number of advantages:

- It does not require the user to take any additional steps in order to search the utility. At no point is the user requested to browse to a separate web page and type custom/utility specific search queries.
- It does not require additional knowledge or experience from the user in order to evaluate the provided results. The results are displayed in a format consistent with the core functionality of the utilities used on a day to day basis.
- The cognitive cost of providing the user with information of a lower relevance than the baseline tool is minimal as the user can still use the baseline tool without any additional effort. Inevitably such design makes the utility indistinguishable from the baseline tool.
- This removes the cognitive load from the user which would otherwise be required during the explicit search process. This also minimizes the negative impact of low relevance estimation on the user perception of the quality of the system.
- This was achieved by minimizing the cognitive load related to the assessment of the retrieved information as well as the number of actions that have to be taken to retrieve the code specific information. The proposed add-in, which uses Google Search as an example, was not designed to require any specific interaction from the user. It could embed into any standard Search Page (through the use of plug-ins).

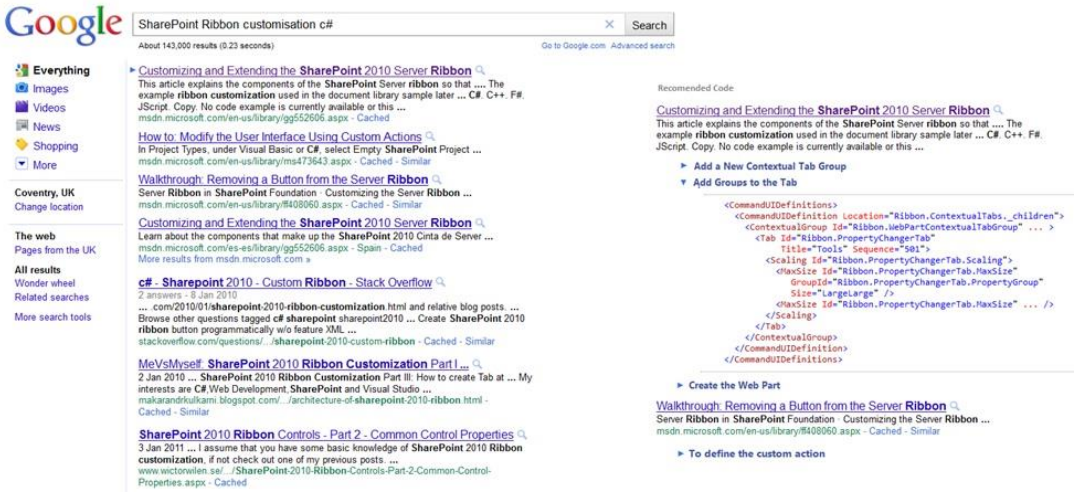

Figure 6. System integration with a commercial search engine

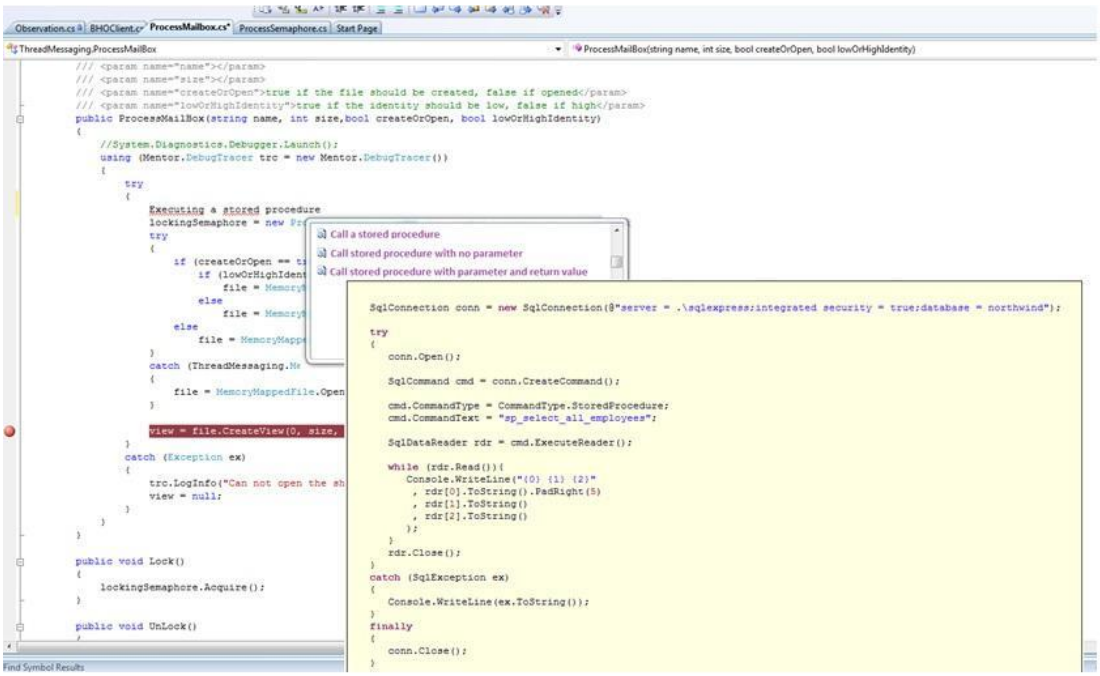

Figure 7. System integration with Microsoft IntelliSense

VI. EVALUATION

Code prediction is the core of the proposed recommender system and in order to evaluate this, we captured data through automated monitoring software. The software was embedded into the key programs that the users used in the workplace and this enabled automated observation of their interaction. The software did not have any user interfaces (apart from the credential and tracing screens not normally accessible by the users) and did not require any user interaction. We conducted content analysis and based on this we extracted code segments, websites visited by the developers as well as successful queries and keywords used during the software developers' interactions with websites and development environments. We also extracted the data related to programming problems they were trying to solve in order to re-generate tasks. We re-constructed 25 tasks and then randomly distributed these tasks among the participants within the same context (e.g., development environment, nature of problem etc). Based on this information, we tagged pieces of source code according to whether they were relevant or irrelevant to the programming problems and queries.

The experiments were run in a controlled environment simulating the real world situation of professional software engineers. We invited 12 participants to perform the experiments. Each of the participants was given two software development related tasks (containing class and logic). The participants were allowed to use their own keywords to browse or check pieces of source code and/or refine their search. The experiments were run for up to 3 hours depending on the nature of the task and participants experience. We collected all information related to the participants' interaction with browsers and software development environments as shown in Figure 9.

Data captured in the browser (IE, Chrome and Firefox)	Data captured in Visual Studio	Data captured by the general program OS wise
- Timestamp when the page is opened - Timestamp when the page is closed - Active reading time of the page - Amount of scrolling - Amount of clicks on the page - Browser type - Domain name of the page for classification of content as work / non work related - GUID of the interaction - GUID of the previous interaction (allowing to analyse the entire session and not only the individual pages) - Amount of focus changes - URL of the page - As a consequence of the URL the search activities and queries - As a consequence of the URL the page content (under the condition the page is public) - IP of the requesting host	- Solution identifier - Project identifier - File identifier - File membership in a project and solution - Timestamp when the file is opened - Timestamp when the file is closed - Amount of scrolling - Amount of clicks - Number of build actions - Usage of the refactoring tools	- Copy and paste pattern determining from which sources the data was copied and finally where it was pasted (focusing mainly on the browser) - Focus change pattern determining how the user changed focus between the different types of information (focusing mainly on the browser and development environment) - IP of the machine hosting the program

Figure 8. The data captured for evaluation of the Recommender Systems

Following this we carried out our offline analysis of usage prediction [44] based on the three possible outcomes as shown in Table VII and adopted from [45]. This explains the number of predictions which were recommended and used by the participants (tp); number of predictions which were recommended by the system but not used by the participants (fp); number of predictions which were not recommended, but used by the participants (fn) [45]. We did not use the metrics related to 'not recommended' and 'not used' because we were only interested in the copy and paste actions as a strong indicator for relevance as well as where the predictions were offered but not used by the participants. IR research suggests that Precision and Recall are effective metrics to evaluate the recommender systems [46]. Using the achieved results, we calculated the Precision and Recall using equation (1) and (2) as follows [45]:

TABLE IV. CLASSIFICATION OF THE POSSIBLE RESULTS

	Recommended %	**Not recommended %**
Used	*True-Positive (tp)* *55*	*False-Negative (fn)* *41*
Not used	*False-Positive (fp)* *4*	*N/A*

$$\text{Precision} = tp/(tp + fp) \quad (1)$$

$$\text{Recall} = tp/(tp + fn) \quad (2)$$

We achieved a Precision value of 0.932 and a Recall value of 0.572 as shown in Figure 10. By refining the recommendation algorithm, we can produce more predictions which will increase the value of Recall. However this may also reduce the value of Precision. It is considered better to have higher Precision because user tolerance to less relevant content that does not satisfy their task related information needs will be much lower. Therefore, the focus is on quality of the returned content even at the price of lower recall. It is also possible that the results have some positive bias due to the re-generation of the task scenarios. However this is a preliminary evaluation of the prototype which aims to demonstrate how domain-specific search and recommendation utilities can be developed in order to support software developers in their work related activities. The full deployment of the system was not possible due to time constraints as the system needs to accumulate sufficient data based on developers' activities over an extended period of time in order to learn their interaction model and evaluate the quality of recommendations under real operational scenarios with a larger number of developers.

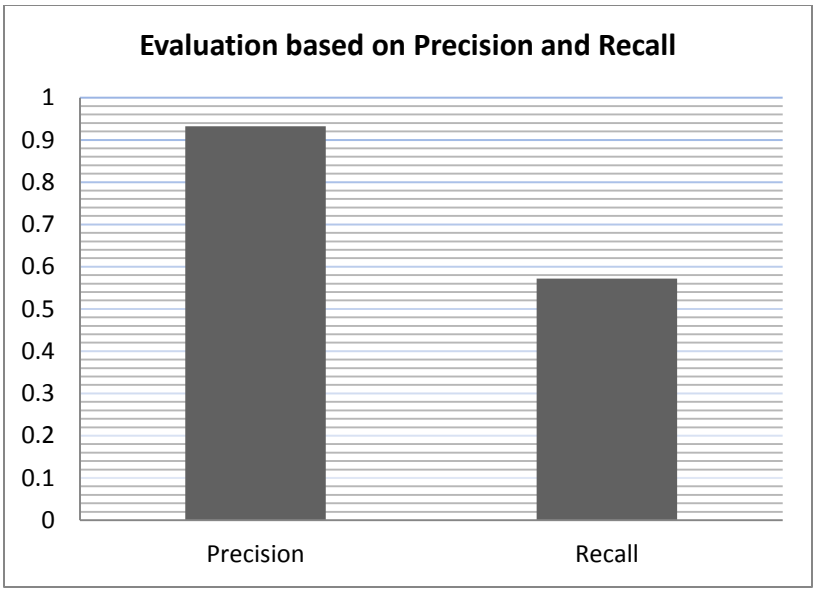

Figure 9. Evaluation results showing Precision and Recall values.

VII. CONCLUSIONS

It is important to notice that the development of task-specific search systems and interfaces is much more sophisticated than general purpose IR systems as it requires a detailed analysis of user behaviour and identification of the repeatable aspects of this behaviour. This can reveal the user's interest, and help tailor the design of user interfaces to the tasks at hand. Because of this, the development of domain specific IR systems remains a challenging task without an easily available methodology which can be used by commercial developers. By providing developers with utilities for behavioural investigation and analysis, this process can be made significantly simpler and as a consequence more cost-effective.

The literature review presented in this paper illustrates that the software development process is composed of a fixed amount of repeatable development tasks. Across each of those tasks and also across different stages of the software development lifecycle, a software developer faces a fixed number of information need types. One such need is to reuse the code found on the internet to decrease the cognitive burden of software development and increase its speed. To investigate this need a comprehensive user study and analysis of 14 professional software developers engaged in day-to-day coding tasks was performed. The findings of the study revealed that it was possible to infer content relevance from the copy and paste action performed by the user, which can be perceived as a strong implicit feedback indicator for the relevance of retrieved code fragments.

The analysis of software development tasks allowed us to develop a prototype system for retrieval and reengineering of code. The prototype was developed to recommend relevant code fragments based on modelling the user's behaviour associated with copy and paste actions of information content (page selected). The prototype also demonstrated how the code retrieval interface could easily be integrated into a commercial search engine and IDE to reduce cognitive and manual cost of the developer having to navigate between different search utilities and their development environment. An initial evaluation of the system was performed by 12 participants and the results showed good accuracy in predicting and recommending code for reuse. Our future work includes further refinement and evaluation of task-specific search systems for retrieval and reengineering of code for professional developers as well as students in an academic setting. We envisage that the software development company where the testbeds were run would adopt the system for widespread use.

REFERENCES

[1] Liu, W., Chen, C., Lakshminarayanan, V., & Perry, D. (2005). A Design for Evidence-based Software Architecture Research.

[2] Brandt, J., Guo, P., Lewenstein, J., Dontcheva, M., & Klemmer, S. (2009). Two studies of Opportunistic Programming: interleaving Web Foraging, Learning, and Writing Code. In Proceedings of the 27th International Conference on Human factors in Computing Systems (CHI '09). ACM, New York, NY, USA, 1589-1598

[3] Singer, J., Lethbridge, T., Vinson, N., & Anquetil, N. (1997). An examination of Software Engineering Work Practices. In Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative research (CASCON '97), IBM Press, 174-188.

[4] King, D., & Griffiths, J. (1991). Indicators of the Use, Usefulness and Value of Scientific and Technical Information. In D. Raitt (Ed.), Proceedings of the 15th International Online Meeting London, December 10-12. Oxford, UK: Learned Information Ltd., 361-377

[5] Freund, L., Toms, E., & Waterhouse, J. (2005). Modeling the Information Behaviour of Software Engineers using a Work-task Framework. Proceedings of the American Society for Information Science and Technology, 42(1), n-a.

[6] LaToza, T., Venolia, G., & DeLine, R. (2006). Maintaining Mental Models: a Study of Developer Work Habits. In Proceedings of the 28th International Conference on Software engineering (ICSE '06). ACM, New York, NY, USA, 492-501.

[7] Perry, D., Staudenmayer, N., & Votta, L. (1994). People, Organizations, and Process Improvement. Software, IEEE, 11(4), 36-36.

[8] Tenopir, C. and King, D.W. (2004) Communication Patterns of Engineers. J.Wiley/IEEE.

[9] Brandt, J., Guo, P., Lewenstein, J., Dontcheva, M., & Klemmer, S. (2009). Two studies of Opportunistic Programming: interleaving Web Foraging, Learning, and Writing Code. In Proceedings of the 27th International Conference on Human factors in Computing Systems (CHI '09). ACM, New York, NY, USA, 1589-1598

[10] Poshyvanyk, D., Marcus, A., & Dong, Y. (2006). JIRiSS - an Eclipse Plug-in for Source Code Exploration. In Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC '06). IEEE Computer Society, Washington, DC, USA, 252-255.

[11] Begel, A. (2007). Codifier: a Programmer-centric Search User Interface. Paper presented at the Workshop on Human-Computer Interaction and Information Retrieval. Cambridge, Massachusetts, USA, 23-24

[12] Hoffmann, R., Fogarty, J., & Weld, D. (2007). In Proceedings of UIST 2007: ACM Symposium on User Interface Software and Technology, pages 13–22, Newport, Rhode Island, 2007.

[13] Gallardo-Valencia, R., & Elliott Sim, S. (2009). Internet-Scale Code Search. In Proceedings of the 2009 ICSE Workshop on Search- Driven Development-Users, Infrastructure, Tools and Evaluation (SUITE '09). IEEE Computer Society, Washington, DC, USA, 49-52.

[14] Bajracharya, S., Ossher, J., & Lopes, C. (2009). Sourcerer: An Internet-scale Software Repository In Proceedings of the 2009 ICSE Workshop on Search-Driven Development -Users, Infrastructure, Tools and Evaluation (SUITE '09). IEEE Computer Society, Washington, DC, USA, 1-4.

[15] Brandt, J., Dontcheva, M., Weskamp, M., & Klemmer, S. (2010). Example-centric Programming: Integrating Web Search into the Development Environment. In Proceedings of the 28th International Conference on Human factors in Computing Systems (CHI '10). ACM, New York, NY, USA, 513-522.

[16] Umarji, M., Sim, S., & Lopes, C. (2008). Archetypal Internet-scale Source Code Searching. Open Source Development, Communities and Quality, 257-263.

[17] Wen, H., Fang, L;, Guan., L., (2012)"A hybrid approach for personalized recommendation of news on the web", Expert systems with applications, v39 n. 5, p., 5806-5812, Elsevier.

[18] Kelly, D. (2005). Implicit feedback: Using behavior to infer relevance. In A. Spink and C. Cole (Eds.) New Directions in Cognitive Information Retrieval. Springer Publishing: Netherlands (pp.169-186).

[19] Hartmann, B., Dhillon, M., & Chan, M. K. (2011, May). HyperSource: bridging the gap between source and code-related web sites. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 2207-2210). ACM.

[20] Sun, J. T., Zeng, H. J., Liu, H., Lu, Y., & Chen, Z. (2005, May). Cubesvd: a novel approach to personalized web search. In Proceedings of the 14th international conference on World Wide Web (pp. 382-390). ACM.

[21] Kelly, D. and Teevan, J. (2003). Implicit feedback for inferring user preference. SIGIR Forum, 37 (2), 18-28.

[22] Oard, D. W., and Kim, J. (2001). Modeling information content using observable behavior. In Proceedings of the 64th Annual Meeting of the American Society for Information Science and Technology, USA, 38-45.

[23] Kapser C. J. (2009) Towards an Understanding of Software Code Cloning as a Development Practice. PhD Thesis, Waterloo Univeristy, Ontario

[24] Kavita Philip, Medha Umarji, Megha Agarwala, Susan Elliott Sim, Rosalva Gallardo-Valencia, Cristina V. Lopes, and Sukanya Ratanotayanon. 2012. Software reuse through methodical component reuse and amethodical snippet remixing. In Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work (CSCW '12). ACM, New York, NY, USA, 1361-1370.

[25] Johan Bollen, Michael L. Nelson, Gary Geisler, and Raquel Araujo. 2007. Usage derived recommendations for a video digital library. *J. Netw. Comput. Appl.* 30, 3 (August 2007

[26] Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., & Klemmer, S. R. (2009, April). Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 1589-1598).

[27] Oard, D. W., and Kim, J. (2001). Modeling information content using observable behavior. In Proceedings of the 64th Annual Meeting of the American Society for Information Science and Technology, USA, 38-45.

[28] Kavita Philip, Medha Umarji, Megha Agarwala, Susan Elliott Sim, Rosalva Gallardo-Valencia, Cristina V. Lopes, and Sukanya Ratanotayanon. 2012. Software reuse through methodical component reuse and amethodical snippet remixing. In Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work (CSCW '12). ACM, New York, NY, USA, 1361-1370.

[29] Johan Bollen, Michael L. Nelson, Gary Geisler, and Raquel Araujo. 2007. Usage derived recommendations for a video digital library. *J. Netw. Comput. Appl.* 30, 3 (August 2007

[30] Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., & Klemmer, S. R. (2009, April). Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 1589-1598).

[31] Brandt, J., Dontcheva, M., Weskamp, M., & Klemmer, S. R. (2010, April). Example-centric programming: integrating web search into the development environment. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 513-522). ACM.

[32] Kapser C. J. (2009) Towards an Understanding of Software Code Cloning as a Development Practice. PhD Thesis, Waterloo Univeristy, Ontario

[33] Kavita Philip, Medha Umarji, Megha Agarwala, Susan Elliott Sim, Rosalva Gallardo-Valencia, Cristina V. Lopes, and Sukanya Ratanotayanon. 2012. Software reuse through methodical component reuse and amethodical snippet remixing. In Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work (CSCW '12). ACM, New York, NY, USA, 1361-1370.

[34] Jones, W., Bruce, H. and Dumais, S. (2001). Keeping found things found on the web. Proceedings of the 10th Conference on Information and Knowledge Management. (pp. 119-134).

[35] Kyuyong Shin, Douglas S. Reeves, Winnowing: Protecting P2P systems against pollution through cooperative index filtering, Journal of Network and Computer Applications, Volume 35, Issue 1, January 2012, Pages 72-84

[36] Juan J. Samper, Pedro A. Castillo, Lourdes Araujo, J.J. Merelo, Óscar Cordón, Fernando Tricas, NectaRSS, an intelligent RSS feed reader, Journal of Network and Computer Applications, Volume 31, Issue 4, November 2008, Pages 793-806

[37] Adomavicius, G.; Tuzhilin, A. (2005) Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions, Knowledge and Data Engineering, IEEE Transactions on Data and Date Engineering , vol.17, no.6, pp. 734- 749.

[38] Hartigan, J. A. and Wong, M, A, (1979) "Algorithm AS 136: A k-means clustering algorithm," Applied Statistics, vol. 28, no.1, pp. 100 – 108.

[39] Kohonen, T. (1989) Self-Organisation and Asssociative Memory. 3rd ed. Springer-Verlag, Berlin, Heilderberg, Germany (1989).

[40] Carpenter, G.A. & Grossberg, S. (2003), Adaptive Resonance Theory, The Handbook of Brain Theory and Neural Networks, Second Edition (pp. 87-90). Cambridge, MA: MIT Press.

[41] Lee, S.W., Palmer-Brown, D., Tepper, J.A., Roadknight, C.M.: Snap-drift: real-time, performance-guided learning. In: International Joint Conference on Neural Networks, Portland, OR, USA, 20-24 July 2003. Piscataway, NJ, USA: IEEE, pp. 1412-1416 (2003).

[42] Lee, S.W., Palmer-Brown, D., Roadknight, C.M (2004) 'Performance guided Neural Network for Rapidly Self Organising Active Network Management', Neurocomputing, 61 (2004), pp. 5-20.

[43] Umarji, M., Sim, S., & Lopes, C. (2008). Archetypal Internet-scale Source Code Searching. Open Source Development, Communities and Quality, 257-263.

[44] Yi, J., Chen, Y., Li, J., Sett, S., Yan, W. T.(2013): Predictive model performance: offline and online evaluaitons. Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, p 1294-1302.

[45] Shani G, Gunawardana A (2011) Evaluating recommendation systems. In: Ricci F, Rokach L, Shapira B, Kantor P (eds) Recommender systems handbook. Springer, Berlin, pp 257-298.

[46] Bellogin, A., Castells, P., Cantador, I., (2011):" Precision-Oriented Evaluation of Recommender Systems: An Algorithmic Comparison", In Recommender Sysems (RecSys), pp., 333-336 ACM Press.

[47] Grzywaczewski, A., Iqbal, R., James, A., Halloran, J., (2009): "An Investigation of User Behaviour Consistency for Context-Aware Information Retrieval Systems ", International Journal of Advanced Pervasive and Ubiquitous Computing (IJAPUC), 1(4), pp. 69-90

[48] Grzywaczewski, A., Iqbal, R., James, A., Halloran, J.,(2011): "An Approach for Capturing Human Information Behaviour", Emering Pervasive and Ubiquitous Aspects of Information Systems: Cross-Disciplinary Advancements, (DOI: 10.4018/978-1-60960-487-5.ch018) pp. 69-90.

[49] Grzywaczewski, A., and Iqbal, R., (2012):"Task-Specific Information Retrieval Systems for Software Engineers", Journal of Computer and System Sciences, Elsevier, Volume 78, Issue 4, pp., 1204-1218.

[50] Iqbal, R., Shah, N., James, A., and Cichowicz, T., (2013):"Integration, optimization and usability of enterprise applications, Journal of Network and Computer Applications, Elsevier , Volume 36, Issue 6, 1480-1488.

[51] Wang, J.W. Wang, H.F. Ding, J.L. Furuta, K. Kanno, T. Ip W.H. and Zhang W.J. (2013) On domain modelling of the service system with its application to enterprise information systems, EIS, DOI:10.1080/17517575.2013.810784.

[52] Wanga J. W., Gao F. and Ip W. H. (2010) Measurement of resilience and its application to enterprise information systems, EIS, Volume 4, Issue 2, May 2010, pages 215-223.

[53] Zhang W. J. and Lin Y. (2010) On the principle of design of resilient systems – application to enterprise information systems, EIS, Volume 4, Issue 2, May 2010, pages 99-110.

[54] Obada, A., Iqbal, R., and Faiyaz, D., (2015) An Adaptive Recommender System Based On Task, User And Document Profiling Using Fuzzy Rule Based Summarisation, In FUZZ-IEEE , in press.

[55] Yan Q., (2008): A security evaluation approach for information systems in telecommunication enterprises, EIS, Volume 2, Issue 3, pages 309-324.