Software Informatics?

M. Cameron Jones
Yahoo! Research
2821 Mission College Blvd.
Santa Clara, CA 95134 USA
mcjones@yahoo-inc.com

Michael B. Twidale
Graduate School of Library and Information Science
University of Illinois at Urbana-Champaign
501 E. Daniel St. Champaign, IL 61820 USA
twidale@illinois.edu

ABSTRACT

The paper proposes the term 'software informatics' to describe research on the individual, collaborative, and social aspects of software production and use, spanning multiple representations of software from design, to source code, to application. It does this with a particular focus on information processes around software development. As more applications are web-based or available online for download and increasing amounts of source code is also available online, the information processes surrounding software are changing the way that software is created, appropriated and redesigned. Traditional distinctions between software developer and end-user are blurring, with software development processes occurring along a continuum of the proportion of original code written to develop the application, from build from scratch, through library and API calls, copy-paste programming, web mashup development, end-user programming, to creative design through the selection and combination of existing applications. All these design activities have much in common and can benefit from being studied as a whole.

Categories and Subject Descriptors

D. Software

General Terms

Measurement, Documentation, Design, Human Factors.

Keywords

Software engineering, Social informatics, bibliometrics, appropriation, end-user programming, collaborative software development.

1. INTRODUCTION

The ways that people develop software are themselves changed as a consequence of the software that they develop. This should not surprise us. It is a direct consequence of the task artifact cycle [6]. In this speculative paper, we want to explore whether it should also affect the ways that we might want to study this process – who and what we study, and how we do it. There are a number of related approaches to the study of software as information and the interconnections between this and software development and innovation-in-use. This work is being done by a variety of researchers, many of whom, perhaps not coincidentally, happen to be located in iSchools and other departments with a strong focus on the concepts, theories and methods around informatics and information science.

Associated with these studies are a number of emerging trends:

- Software is becoming increasingly abundant and ubiquitous. Many applications are free or extremely low cost. They can be downloaded quickly and easily, or are web based and so need no downloading at all.
- Open Source software contributes to this abundance. By making not only the use of the application free, but also the underlying source code freely available for download, it creates a resource for other programmers to appropriate and incorporate into their applications. Code fragments are a part of this as well as complete OSS applications.
- Software development is thus able to exploit more reuse of code fragments, components, API calls or other open access software services. This manifests in a variety of forms including web mashups.
- There are changes in the nature and degree of socialization around software development, including distributed design teams and agile methods.

As a result, the notion of programming is broadening from build from scratch, through library and API calls, copy-paste programming, to web mashup development. These are all still highly reliant on considerable programming skill. However certain software has been designed to enable activities requiring less programming skill. These include end user programming, customization, tailoring, adaptive programs, or simply the selection and composition of multiple discrete applications to create a new collaborative infrastructure. This broadening means that it can be harder to draw a clear distinction between programmers and non-programmers, experts and novices, and software development and software use. Often it is precisely the variability of people along these continua (and variations of an individual over time) that are a necessary component of the analysis.

We propose the term 'Software Informatics' as an organizing concept for integrating research on these issues that is spread across numerous domains and disciplines. The aim is to bring together research around these trends with a focus on software and a deeper understanding of software as information.

2. RECENT RESEARCH TRENDS

Recent research in a variety of disciplines reflects a trend towards studying software as information. Software as information has been discussed by Wang [49], in the context of cognitive informatics. This approach has explored the nature of software as an abstract entity, and how that affects people's understandings of it. However, there are many more implications which arise from treating software as information. Researchers have been capitalizing on the recent abundance of available software and source code to explore new avenues of interdisciplinary research. In many ways, the research questions and topics are familiar; yet, their application to software makes them unique. However, they share a common focus on the topic of software, which brings these research areas together.

2.1 Authorship in software development

Considering software as information invites questions of its origin and authorship. Several specific research topics fall within the study of authorship in software. Gray et al.'s work on "software forensics" has established techniques and methods for identifying and discriminating authors of source code [15]. The research borrows from work on author identification in linguistics and humanities studies of text. Newby et al. [31] looked at open source software and tried to model the productivity of programmers in open source projects, using Lotka's model of authorship borrowed from bibliometrics. The analysis not only cast light on the nature of open source software authorship, but also raised issues for the traditional methods of parameter estimation and model fitting typically used in bibliometrics.

2.2 Collaborative software development

Studies of programmers have reinforced the collaborative nature of the software development process. Perry et al. [35] found that over half of developers' time was spent interacting with coworkers. Collaboration occurs not only in co-located settings (e.g. [7], [40] & [45]) but also in distributed software development where collaboration is more challenging, but still necessary [18].

Collaboration has many advantages, but also costs in maintaining awareness [16] and dealing with interruption [28]. Collaboration is also a highly significant factor in end user tailoring [30], [37] & [46]. The structure of collaboration across all of these contexts has been studied using social network analysis methods to determine the degree of control centralization within various open source developer communities [9].

2.3 Software and information seeking

Software development is not just about the production of information in the form of code. It also involves the search for information to support the production process. A number of researchers have studied the information seeking behaviors of software developers, (e.g. [25] & [42]), and have developed information seeking models of programmers [34]. Not surprisingly this work draws on theories and methods in Library and Information Science.

As well as using local documents and colleagues, developers are increasingly searching online for information. Online resources may include example code fragments, modules, or even whole programs from which extracts may be copied unchanged into a program, or used as a structure for modifying to the programmer's unique needs. This activity may be termed "programming by Google", and is a dramatic extension of traditional in-house software reuse practices.

Like programmers, the users of software also search for information online, looking for technical help relating to their application. This may be in response to a confusion they have about use, or in response to a system error, or it may be because they are looking to use the application in a novel way (see section 2.7), and want to know if others have tried this before or have any advice on how to proceed. This collaborative help-giving in online forums occurs for both commercial and open source applications. In the case of open source, a number of researchers have been investigating both why people do this [26] and how [41].

2.4 Programmer as user

A variety of tools have been developed that are intended to support different aspects of the software development process. Inevitably, these tools are made by other programmers (or themselves). The assumption that programmers know what other programmers need, and how to design and build for them is not always valid – careful observation of existing practice, such as the work mentioned in section 2.2 can greatly improve the effectiveness of developed tools. Such tools include text editors, integrated development environments, compilers, debuggers, version control systems, bug tracking systems, task management software, and software visualization tools.

As with all applications, these tools need to be evaluated and refined in situated use (e.g., [29]). Such evaluations often reveal not only unexpected problems with the adoption and continued use of the applications, but also unexpected benefits, as the users (programmers) appropriate the tools in novel creative ways. Both findings can be used to develop implications for redesign. Examples of this approach include: [4] and [17]. Evaluation has even been applied to the software APIs programmers use, focusing how APIs are learned [52], how they are used in collaborative software development [10], and how they can be designed for improved usability [8] & [43].

2.5 User as programmer

The people who use applications are far from passive recipients of products developed by others. With software abundance comes the ability to choose which software to adopt and when or if to replace it with another application, or use both together for different purposes. Applications may also be developed that enable users to play a direct role in tailoring them to their own particular needs, creating, modifying or extending a software artifact [27]. Work in this area may be termed end user programming, end user development or end user software engineering [5].

There are a number of approaches. Some work aims to make general programming more accessible to more people, by use of various support environments such as visual programming [20]. Other approaches aim for a more restricted, less complex resource than a general programming language, focusing more on tailorability or configurability (e.g. [32] & [51]).

2.6 Software remix

The increased availability of software, source code, and online programming resources is changing the way developers write programs. Recent research points to the pervasive reuse and remixing existing source code in software development. Widespread software reuse has long been the dream of the software engineering community; however, reuse in this context means formal mechanisms for abstraction and packaging, through entities like function and class libraries, and APIs (application programming interfaces). In this view, software is created by linking to, and calling code in pre-existing modules, saving developers the time and effort of writing that code themselves. However, there is an alternate sense of reuse, which is more accurately described as remix, in which developers write code through copying and pasting existing code, from various sources. Some classify this kind of programming as "post-modern" [33] because it involves building new software through the decomposition and reassembly of existing code. Copying and pasting code has been documented in a number of contexts, using a variety of research methods, including surveys of developers

[38], observations of programmers [39], and locating and tracing copied code within source code collections [19] & [23].

2.7 Innovation in the use of software

The development of new features and new applications is no longer the exclusive preserve of expert software developers. As von Hippel [48] found in a number of different markets, end user innovation can be a critical source of inspiration. This can involve tailoring an application, adding new features, or using it unmodified but in concert with a number of different applications [2], [11] & [36]. This activity can be to optimize an existing use of the application, or alternatively it can be an act of appropriation [13] where the application is used in a way, in a context, or for a purpose that the designers of that application had not intended. This activity seems to have many commonalities with at least some of the components of software development (particularly requirements capture). It can be useful to consider innovation, appropriation and tailoring activities at various points along the continuum from pure software development through to innovative adoption but with no actual programming.

New media theorists study certain innovations involving appropriation, such as machinima and animutations (e.g. [3] & [22]). For those researchers, the focus is typically on what is produced, whereas from the perspective of software informatics, the interest is more on how the artifacts are produced, the appropriations, innovations, tailorings and tool building that occurred, and how the skills, processes and tools were propagated and refined in use, often as part of a learning community.

As another example, Twidale & Floyd [47] have been considering the case of top-down versus bottom cyberinfrastructures. Top-down infrastructure development follows a conventional software engineering process, ideally with a strong user-centered design orientation. Various alternative bottom-up processes are possible, including researchers rapidly cobbling together a rough kind of ad hoc cyberinfrastructure out of free or low-cost components such as email, Skype, instant messaging, Google Documents, Yahoo Groups, Twitter, etc. This is effectively an infrastructure for collaboration just as much as an integrated collaboratory, but with different strengths and weaknesses. It also seems to be a kind of software development involving innovation in the coordinated use of software, even if in its extreme case it involves no actual programming. Assembly of multiple resources into a single web-page, development of web-mashups, construction of file manipulation macros, etc. are points nearer to software development on this same continuum.

What we find striking about these examples is that they appear thematically to be more similar to each other, perhaps more so than to the individual disciplines from which they came. They appear to indicate that there is a common ground at the intersection of software with many disciplines, including: computer science, software engineering, library and information science, informatics, social informatics, social sciences, and psychology (Figure 1). There are likely other related disciplines and topics and one of the objectives of identifying "software informatics" is to facilitate the bringing together of related research.

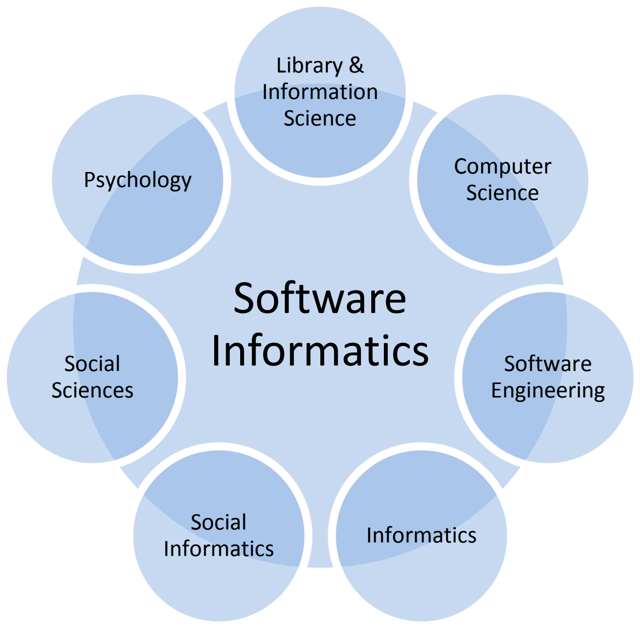

Figure 1. Software informatics sits at the intersection of several disciplines.

3. DEFINING SOFTWARE INFORMATICS

Software informatics necessarily draws on the more general definition of informatics. Fourman [14] defines informatics as "the science of information," and later defines compound, discipline-specific topics, such as bio-informatics, as "the specialization of informatics to the management and processing of data, information and knowledge in the named discipline". Software informatics can be defined as the science of information, practice, and communication around software. It studies the individual, collaborative, and social aspects of software production and use, spanning multiple representations of software from design, to source code, to application.

Like many other informatics areas (e.g., social informatics), software informatics is defined around a common research interest rather than any specific methods or theories. This is partly due to the emergence of software informatics from many related disciplines, which do not necessarily share common methods or theories. These disciplines include software engineering, information science, social informatics, and several others.

3.1 Comparison to Software Engineering

One of the closest neighboring disciplines to software informatics is software engineering. Software engineering, as defined by the IEEE Computer Society, is "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software" [1]. From this definition, software engineering is particularly concerned with formal methods of software development, with the primary objective of improving software quality. It could be argued that the literal definition of software engineering does not allow for studies of the social networks of programmers in open-source software development, for example, as this is not an engineering question. However, the objective of defining software informatics as a distinct field is not to engage in a turf war with software engineering, but rather to highlight a broader perspective on software as a focus of research and interest which goes beyond engineering. Indeed, there is substantial desirable and productive overlap with software engineering and software informatics.

We do not see these overlaps as a problem, but as an indication in the emphasis of research. For example the focus of much software engineering research is on large scale professional software projects. Examples used often include control systems for nuclear power stations, air traffic control, financial records processing, etc. Open source software development has become a growing area of interest in the field, but can be treated as a somewhat anomalous process to be studied for its differences from conventional methods. In any case it still typically involves skilled programmers. However, as well as these kinds of software development, software informatics examines development activities further down the programming skill spectrum to include issues of tailoring and end user programming.

Additionally certain aspects of the larger process of software development may have a different emphasis or approach. Thus tool development is an important part of software engineering as are quantitative methods for the evaluation of those tools. Software informatics might focus more on the interface to the tool, the qualitative nature of situated use of the tool, innovations, and appropriations of the tool, and explorations of why a tool was adopted in one context but not in another, or initially adopted but abandoned in a third. Also, drawing more heavily on information science than software engineering usually does, software informatics may look more at the information seeking practices in the software development process and how that plays out in conversations, paper and electronic documents, code fragments and how-to help.

3.2 Comparison to Social Informatics

Informatics sciences as a whole include "the study of communication as a process that links people together, to affect the behavior of individuals and organizations" [14] which ties informatics broadly to social processes and practices. As a result, software informatics overlaps with social informatics. However, social informatics focuses on the "design, uses and consequences of information technologies that takes into account their interaction with institutional and cultural contexts." [24]. Many topics which we have identified above as being part of software informatics may also be considered social informatics. However, there are some subtle differences which distinguish the two.

In the simplest sense, social informatics is concerned with the effects of information technologies on social and cultural contexts. Information technologies are typically treated as products, or artifacts within a social context of use, which may co-evolve with the social context; however, the focus is on the social context, and the interactions between the technology and its usage and evolution.

By comparison, software informatics takes the software as the focus, and studies how it is created and changed through use. Social practices may emerge and evolve around the development and use of software. However, there are numerous topics which are related to the design, development, and use of software which do not fall within the scope of social informatics; for example, the study by psychologists of the process of programming (e.g., [50]). We might also consider the measurement and tabulation of features of the software source code (e.g., tracking the occurrence of bugs within or between software systems) as software informatics, but is not typically something one would consider within social informatics.

3.3 Software as information

A defining characteristic of software informatics is the treatment of 'software as information'. That phrase has several possible meanings, as there are many different information flows which may be studied, including: design specifications, source code, documentation, software libraries, applications and services, code repositories, revision histories, and more. Treating these software artifacts as information objects or documents (c.f., [44]), opens up new lines of inquiry around their origin, evolution, description, organization, dissemination, use, and impact.

As noted software engineering focuses on software's origins and methods of production. However, the science of software production is broader than software development methods. An analogy can be drawn to the information science study of informetrics and bibliometrics, which characterizes the general processes of scholarly information production and use as documented in information objects and bibliographic records. This is a powerful complement to research in science and technology studies that more typically uses ethnographic methods to observe and understand the behaviors of scientists in the lab. Both informetrics and science and technology studies tell us about the production of scientific knowledge, but in different ways and answering different sub-questions.

Egghe and Rousseau [12] introduced the concept of the "information production process" (IPP) in order to bring together numerous informetric and bibliometric theories and models. In the IPP framework, entities are divided into two classes: *sources* and *items*. The IPP framework characterizes the production of *items* by *sources*. Egghe and Rousseau provide IPP-descriptions of several research fields, enumerating the sources and items which could be measured in each field. For example: econometrics studies the production of goods by workers; linguistics, the production of words in texts; bibliometrics, the production of publications by authors or journals, or the production of library circulation activity by books, or the production of citations by publications. The central concept of information production, brings together a diverse body of research from many fields, and similarly it can serve as a useful framework for describing some aspects of software informatics.

One can apply the two-part framework of the information production process to software, and begin to enumerate several research topics. This discussion is provided as a demonstration of what can be gained by taking a software informatics approach, which draws upon multiple disciplines for methods, theories, and frameworks for analysis and understanding. In the software universe, some *sources* which could be considered include: code snippets, functions/methods, classes, modules/packages, source code files, source code directories, code libraries, projects, code repositories, code revisions, code bugs, code clones, patches, applications, programmers, and users. Most of these entities can also be considered *items*, and some additional *items* include: dependencies, revisions, copies, errors, comments, branches, forks, installations, bug reports, design specifications, lines of code, patterns, and file formats.

Pairing these *sources* and *items* in a production relationship allows researchers to interrogate the productive output of a given source; model the collective productivity of all sources; compare sources based on their productivity; and examine the interconnections between sources and items. For example, in bibliometrics, applying the IPP framework to citations among scholarly papers not only allows you to see who cites whom, but also how papers/authors/journals are related through patterns of common citation (i.e., bibliographic coupling); a similar analysis on software references (e.g., include statements) could produce a graph of software components, or clusters of related components. Additionally, comparisons can be drawn between the world of software and other domains, by comparing the patterns and distributions of similar processes.

Of course, software informatics is more than just the application of IPPs to software production, and indeed the enumeration of sources and items given above is not even complete with respect to that topic. Rather, this example has been provided as a thought exercise on how thinking of software as information can broaden the research space, and provide new contexts for comparison.

4. WHY SOFTWARE INFORMATICS?

Applying the label "software informatics" to a body of existing and emerging research in and of itself can be useful as a way to improve accessibility to the literature, and help researchers characterize their interests. Additionally, having a term to describe a field of research contributes to the future definition of boundaries for that field, and the refinement of a specific focus. We have introduced our notion of software informatics which has fairly porous and fuzzy boundaries with neighboring disciplines, and do not presume to speak for all those who may have a stake in the field. As such, it is fully expected that the focus of software informatics will evolve, and the boundaries between software informatics and other disciplines may shift or be refined. However, that discussion can only carry forward within the context of a notion of software informatics.

Labeling the field of software informatics can also promote the development of new methods and theories. By bringing together a body of related literature, there may emerge common practices and methods of data collection, analysis, and interpretation which may give rise to new theories of software informatics. Developing a common way of talking about software and software practice is necessary for this advancement, and there is little hope such convergence will occur when the research is dispersed among numerous fields. Eventually, software informatics may give rise to its own journals, conferences, and other communication and publication venues which will further facilitate discovery of researchers, institutions, and scholarship in the area.

Beyond benefits to academic discourse, software informatics can give scholars in different domains new understandings of software and software development. One of the most significant contributions software informatics can have is that collectively it is broader than any one approach. Currently, software is viewed through a number of different perspectives: software engineering views it from the perspective of engineering and building; human-computer interaction views it from the perspective of use and usability; social informatics views software systems as an agent which affects, and is affected by, organizational and social systems. Individually, each of these perspectives illuminates certain aspects of software, yet none captures the full spectrum.

On a related note, software informatics can focus attention on neglected practices which are understudied. For example, the focus on formal methods of design and development in software engineering may blind researchers to the mundane, and/or ad-hoc everyday practice of programmers. It is well acknowledged that programmers do a large amount of copying and pasting while coding, although this behavior gets little attention from researchers. Web developers who cut their teeth in the mid-1990's will likely admit to learning HTML by viewing the source markup of other pages and copying bits they liked. Yet, no research documented this practice which may have informed our understanding of abstraction in programming. Similarly there is a relative lack of attention given to information seeking, help-giving and learning by programmers as they develop software. Also practices by end users that are remarkably similar to elements of the software development activities of professional programmers are overlooked, or treated as wholly distinct, when that may not be the most productive approach. Looking at multiple points along a continuum of activity has been found to be very advantageous in software product development. It is likely to be also productive in research and in the design of products to support software development-like activities.

5. EDUCATIONAL IMPLICATIONS

The teaching of software developers naturally resides in departments of computer science. However, iSchools that are not also computer science departments, or do not explicitly teach aspects of software development may still have an opportunity to make a significant contribution. Such schools may draw on research in software informatics to inform what should be taught and how.

When well understood, practice-specific information seeking behaviors can be improved. A critical aspect of information literacy skills is understanding ways to search more efficiently, and to assess the relative value, quality and reliability of different results. The importance of multidisciplinary design teams is increasingly being recognized; particularly in the context of user centered design and user experience design [21]. This creates at least two challenges: how to enable people with skills other than programming to join and effectively participate in a software development team, and how to enable a diverse team to continue to function effectively – particularly when this involves distributed work. These challenges will be addressed by diverse soft skills integrating with the skills of rapid efficient reliable coding. They are already a component of the field of software project management, but an iSchool approach to teaching these skills might be a very valuable contribution.

Additionally, software is nearly always deployed in a pre-existing complex socio-technical infrastructure. The application will need to integrate with other legacy applications and future innovations. It will also need to integrate with non computational systems; workflows; norms; organizational, professional and national cultures, etc. Socio-technical systems design has to take all these larger issues into account, ideally at the time of software development, not at the final moment of deployment. This involves skills that can also be addressed in an iSchool.

Finally, certain applications are themselves inherently socio-technical. Consider tools to support online technical help, collaborative software development, online communities of patients with a particular illness, or online games. Sustaining distributed teams or communities using an application and informing the improvement of that application through reflecting on its use is clearly a socio-technical design skill. Some people are very good at it, but currently they seem to mostly learn as they go.

As the number of people participating in this online community-nurturing activity grows, is this something that we can and should teach? If so, it would seem that iSchools are ideally positioned to teach it well.

6. CONCLUSION

We introduce the term software informatics as a way of describing a range of research issues relating to the processes of software development. This research seems to have coherence around what is studied and how it is studied. We do not think it is a coincidence that many (though not all) of the researchers we identify in this category are located in schools of information. This is a highly speculative paper. Our thinking has been evolving as we have been writing it. We hope it will provoke discussion and reaction. Our definition of software informatics is strictly preliminary and open to refinement.

We have tried to show that there are some differences between software informatics and both software engineering and social informatics. But we are reluctant to even try to draw precise lines. There is a huge amount of overlap and any particular piece of research might easily belong in all three. It may be that software informatics turns out not to be a new field, but rather a distinct emphasis or ‘accent’ in what is studied and how.

At the very least we wish to reinforce that the study of how software is created, adopted, appropriated, tailored, refined and revised is an important area of study in iSchools. This is another inherently multidisciplinary area, strongly analogous with others that are well established as iSchool areas such as medical informatics. We hope that this paper has laid out the intellectual geography of the topics we are collecting and calling software informatics, and shown how they are part of the multi-disciplinary iSchool space. Finally we believe that this area highlights various educational implications for the teaching of design in iSchools in the context of the development of software applications, how to participate in multidisciplinary design teams, and how to sustain distributed design teams as a socio-technical skill. All of these can benefit from pre-existing research and pedagogic expertise in iSchools.

7. REFERENCES

[1] Abran, A., & Moore, J. W. (2004). Guide to the Software Engineering Body of Knowledge. IEEE. http://www.swebok.org/swebokcontents-ch1.html#ch1

[2] Balka, E. & Wagner, I. 2006. Making things work: dimensions of configurability as appropriation work. In Proceedings of CSCW '06. 229-238.

[3] Bardzell, J. (2007). Creativity in amateur multimedia: Popular culture, critical theory, and HCI. Human Technology, 3(1):12-33.

[4] Biehl, J.T., Czerwinski, M., Smith, G., & Robertson, G.G. 2007. FASTDash: a visual dashboard for fostering awareness in software teams. In Proceedings of CHI '07. 1313 – 1322.

[5] Burnett, M., Cook, C., & Rothermel, G. (2004). End-user software engineering. Communications of the ACM 47(9) 53-58.

[6] Carroll, J.M., Kellogg, W.A. & Rosson, M. B. (1991): The Task-Artifact Cycle. In: Carroll, John M. (Ed.) Designing Interaction: Psychology at the Human-Computer Interface. Cambridge University Press.

[7] Cherubini, M., Venolia, G., DeLine, R. & Ko, A.J. (2007): Let's go to the whiteboard: how and why software developers use drawings. In Proceedings of CHI 2007. 557-566.

[8] Clarke, S. (2004). Measuring API Usability. Dr. Dobbs Journal, May 2004, pp S6-S9.

[9] Crowston, K., Howison, J. (2005). The social structure of Free and Open Source software development. First Monday, February, 2005.

[10] de Souza, C.R.B., Redmiles, D., Cheng, L.-T., Millen, D., & Patterson, J. (2004) How a Good Software Practice thwarts Collaboration—The Multiple Roles of APIs in Software Development, Foundations of Software Engineering (FSE 2004), 221-230.

[11] Dourish, P. (2003). The Appropriation of Interactive Technologies: Some Lessons from Placeless Documents. Computer Supported Cooperative Work (CSCW) - The Journal of Collaborative Computing, 12 (4). 465-490.

[12] Egghe, L. & Rousseau, R. (1990). Introduction to Informetrics. Quantitative methods in library, documentation and information science. Amsterdam: Elsevier

[13] Eglash, R. (2004). Appropriating Technology: an introduction. In Appropriating Technology: Vernacular Science and Social Power edited by Ron Eglash, Jennifer Crossiant, Giovanna Di Chiro, and Rayvon Fouché University of Minnesota Press.

[14] Fourman, M. (2002). informatics. In John Feather and Paul Sturges, editors, International Encyclopedia of Information and Library Science (second edition). Routledge. http://publish.inf.ed.ac.uk/publications/online/0139.pdf

[15] Gray, A., Sallis, P., & MacDonnell, S. (1997). Software Forensics: Extending Authorship Analysis Techniques to Computer Programs. In Proceedings of the 3rd Biannual Conference of the International Association of Forensic Linguists (IAFL), 1-8.

[16] Gutwin C., Penner R., & Schneider K., (2004) Group awareness in distributed software development. In Proceedings of CSCW 2004. 72 - 81

[17] Halverson, C.A., Ellis, J.B., Danis, C., & Kellogg, W.A. (2006). Designing task visualizations to support the coordination of work in software development. In Proceedings of CSCW '06. 39-48.

[18] Herbsleb, J.D., Mockus, A., Finholt, T.A., & Grinter, R.E. (2000). Distance, dependencies, and delay in a global collaboration. In Proceedings of CSCW '00. 319-328.

[19] Jones, M.C. (2007). Copy-Paste Programming: An exploratory analysis of software clones in Open-Source. In Proceedings of CSNA 2007.

[20] Jones, M.C., Twidale, M.B., & Churchill, E.F. (2008). Mashing up Visual Languages and Web Mash-ups. In Proceedings of VL/HCC'08 143-146.

[21] Kelley, T., & Littman, J. (2005). The Ten Faces of Innovation. Random House.

[22] Kendall, L. (2008) 'Noobs' and 'chicks' on Animutation Portal: power and status in a community of practice. International Journal of Web Based Communities.4(4), 491-502.

[23] Kim, M., Bergman, L., Lau, T. & Notkin, D. (2004) An Ethnographic Study of Copy and Paste Programming Practices in OOPL, International Symposium on Empirical Software Engineering.

[24] Kling, R. (1999). What is Social Informatics and Why Does it Matter? D-Lib Magazine, January 5(1).

[25] Ko, A.J., DeLine R. & Venolia, G. (2007) Information Needs in Collocated Software Development Teams. In Proceedings ICSE'07 344-353.

[26] Lakhani, K.R. & von Hippel, E. (2003). How Open Source Software Works: Free User to User Assistance. Research Policy 32(6) 923-943.

[27] Lieberman, H., Paternò, F., & Wulf, V. (2006). End User Development (p. 492). Springer.

[28] Mark, G., Gonzalez, V. M., & Harris, J. (2005). No task left behind?: examining the nature of fragmented work. In Proceedings CHI '05 321-330.

[29] Murphy-Hill, E. (2006). Improving usability of refactoring tools. In proceedings of the Dynamic Languages Symposium, ACM SIGPLAN.

[30] Nardi, B.A. (1993). A Small Matter of Programming: Perspectives on End User Computing. MIT Press.

[31] Newby, G., Greenberg, J., & Jones, P. (2003). Open Source Software Development and Lotka's Law: Bibliometric Patterns in Programming. Journal of the American Society for Information Science and Technology, 54 (2): 169-178.

[32] Newman, M.W., Sedivy, J.Z., Neuwirth, C.M., Edwards, W. K., Hong, J.I., Izadi, S., Marcelo, K., & Smith, T.F. (2002). Designing for serendipity: supporting end-user configuration of ubiquitous computing environments. In Proceedings of DIS '02. 147-156.

[33] Noble, J. & Biddle, R. (2002). Notes on Postmodern Programming. In Proceedings of the Onward! stream at the Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA).

[34] O'Brien, M.P. & Buckley J. (2005) Modelling the Information-Seeking Behaviour of Programmers - An Empirical Approach. icpc, 13th International Workshop on Program Comprehension (IWPC'05), pp.125-134.

[35] Perry, D.E., Staudenmayer N.A. & Votta L.G. (1994). People, Organizations and Process Improvement. IEEE Software, July, 36–45.

[36] Pipek, V. (2005) From Tailoring to Appropriation Support: Negotiating Groupware Usage. Faculty of Science, Department of Information Processing Science. PhD thesis, University of Oulu, Oulu, Finland.

[37] Robertson, T. (1998). Shoppers and Tailors: Participative Practices in Small Australian Design Companies. Computer Supported Cooperative Work (CSCW), 7(3-4). 205-221.

[38] Rosson, M.B., Ballin, J. &Rode, J. (2005). Who, What, and How: A Survey of Informal and Professional Web Developers. In Proceedings of VL/HCC'05.

[39] Rosson, M.B., Carroll, J.M. (1993). Active Programming Strategies in Reuse. In Proceedings of ECOOP'93, p. 4-20.

[40] Sawyer, S., Farber, J., Spillers, R. (1997). Supporting the social processes of software development Information Technology & People, 10(1): 46.

[41] Singh, V & Twidale, M.B. (2008) The Confusion of Crowds: non-dyadic help interactions. In Proceedings of CSCW2008. 699-702.

[42] Sonnenwald D.H.& Iivonen, M. (1999) An Integrated Human Information Behavior Research Framework for Information Studies, Library & Information Science Research, 21(4, 429-457.

[43] Stylos, J., Graf, B., Busse, D. K., Ziegler, C., Ehret, R., Karstens, J. (2008). A case study of API redesign for improved usability. In Proceedings of VL/HCC'08.

[44] Taylor, A. (2004). *The Organization of Information*. 2nd ed. Westport, Conn: Libraries Unlimited.

[45] Teasley, S. D., Covi, L. A., Krishnan, M. S., & Olson, J. S. (2002). Rapid software development through team collocation. IEEE Trans. Softw. Eng. 28(7), 671-683.

[46] Törpel, B., Pipek, V., & Rittenbruch, M. (2003). Creating Heterogeneity – Evolving Use of Groupware in a Network of Freelancers. Computer Supported Cooperative Work 12(4), 381-409.

[47] Twidale, M.B. & Floyd, I.R. (2008) Infrastructures From the Bottom-Up and the Top-Down: Can They Meet in the Middle? In Proceedings PDC 2008. 238-241.

[48] Von Hippel, E. (2005) Democratizing Innovation. MIT Press.

[49] Wang, Y. (2002). On the Informatics Laws of Software. In Proceedings of ICCI'02.

[50] Weinberg, G.M. (1998). The Psychology of Computer Programming: Silver Anniversary Edition. Dorset House.

[51] Wulf, V., Pipek, V. & Won, M. (2008). Component-based tailorability: Enabling highly flexible software applications. Int. J. Hum.-Comput. Stud. 66 (1) 1-22.

[52] Ye, Y., Yamamoto, Y., Nakakoji, K., Nishinaka, Y., & Asada, M. (2007). Searching the library and asking the peers: learning to use Java APIs on demand. In Proceedings of PPPJ '07, vol. 272. 41-50.