Societal Computing

Thesis proposal

Swapneel Sheth
Department of Computer Science
Columbia University
New York, NY 10027
swapneel@cs.columbia.edu

Advisor: Gail Kaiser

January 30, 2013

Abstract

As Social Computing has increasingly captivated the general public, it has become a popular research area for computer scientists. Social Computing research focuses on online social behavior and using artifacts derived from it for providing recommendations and other useful community knowledge. Unfortunately, some of that behavior and knowledge incur societal costs, particularly with regards to Privacy, which is viewed quite differently by different populations as well as regulated differently in different locales. But clever technical solutions to those challenges may impose additional societal costs, e.g., by consuming substantial resources at odds with Green Computing, another major area of societal concern. We propose a new crosscutting research area, *Societal Computing*, that focuses on the technical tradeoffs among computational models and application domains that raise significant societal issues. We highlight some of the relevant research topics and open problems that we foresee in Societal Computing. We feel that these topics, and Societal Computing in general, need to gain prominence as they will provide useful avenues of research leading to increasing benefits for society as a whole. This thesis will consist of the following four projects that aim to address the issues of Societal Computing.

First, privacy in the context of ubiquitous social computing systems has become a major concern for society at large. As the number of online social computing systems that collect user data grows, concerns with privacy are further exacerbated. Examples of such online systems include social networks, recommender systems, and so on. Approaches to addressing these privacy concerns typically require substantial extra computational resources, which might be beneficial where privacy is concerned, but may have significant negative impact with respect to Green Computing and sustainability, another major societal concern. Spending more computation time results in spending more energy and other resources that make the software system less sustainable. Ideally, what we would like are techniques for designing software systems that address these privacy concerns but which are also sustainable — systems where privacy could be achieved "for free," *i.e.,* without having to spend extra computational effort. We describe how privacy can indeed be achieved for free — an accidental and beneficial side effect of doing some existing computation — in web applications and online systems that have access to user data. We show the feasibility, sustainability, and utility of our approach and what types of privacy threats it can mitigate.

Second, we aim to understand what the expectations and needs to end-users and software developers are, with respect to privacy in social systems. Some questions that we want to answer are: Do end-users care about privacy? What aspects of privacy are the most important to end-users? Do we need different privacy mechanisms for technical vs. non-technical users? Should we customize privacy settings and systems based on the geographic location of the users? We have created a large scale user study using an online questionnaire to gather privacy requirements from a variety of stakeholders. We also plan to conduct follow-up semi-structured interviews. This user study will help us answer these questions.

Third, a related challenge to above, is to make privacy more understandable in complex systems that may have a variety of user interface options, which may change often. Our approach is to use crowdsourcing to find out how other users deal with privacy and what settings are commonly used to give users feedback on aspects like how public/private their settings are, what common settings are typically used by others, where do a certain users' settings differ from a trusted group of friends, etc. We have a large dataset of privacy settings for over 500 users on Facebook and we plan to create a user study that will use the data to make privacy settings more understandable.

Finally, end-users of such systems find it increasingly hard to understand complex privacy settings. As software evolves over time, this might introduce bugs that breach users' privacy. Further, there might be system-wide policy changes that could change users' settings to be more or less private than before. We present a novel technique that can be used by *end-users* for detecting changes in privacy, *i.e.,* regression testing for privacy. Using a social approach for detecting privacy bugs, we present two prototype tools. Our evaluation shows the feasibility and utility of our approach for detecting privacy bugs. We highlight two interesting case studies on the bugs that were discovered using our tools. To the best of our knowledge, this is the first technique that leverages regression testing for detecting privacy bugs from an end-user perspective.

Contents

1 Introduction to Societal Computing

Today's college students do not remember when social recommendations, such as those provided by Amazon, Netflix, Last.fm, and StumbleUpon, were not commonplace. The rise of Web 2.0 and social networking has popularized social computing as a research area. Established research communities such as Human Factors [108], Computer Supported Cooperative Work, and Software Engineering have fostered emerging topics such as Recommender Systems [42,62,116] and Social Software Engineering [11,46,70,90,91]. However, social computing is primarily concerned with achieving individual benefits from community participation, and not so much with addressing the societal downsides – in particular that those individual benefits may come at community expense or even the longer-term expense of the individual.

We present a novel problem – or perhaps a novel way of looking at known problems – that we believe has not yet been explored by the community. Thus we propose and define "**Societal Computing**," a new research area for computer scientists in general and software engineering and programming language communities (SE/PL, hereafter) in particular, concerned with the impact of computational tradeoffs on societal issues. Societal Computing research will focus on aspects of computer science that address significant issues and concerns facing the society as a whole such as Privacy, Climate Change, Green Computing, Sustainability, and Cultural Differences. In particular, Societal Computing research will focus on the research challenges that arise due to the tradeoffs among these areas. An example of such a tradeoff could be a complex software system that needs to comply with varying laws in different regions or countries. While complying with such laws is important for the society as it might safeguard the interests of individuals, doing so might require investing considerable computer resources through the entire software lifecycle, which might not be a good idea when Green Computing is concerned. As complying with laws would be mandatory, the option should not be to ignore the law but perhaps to lobby to change the law, by making regulators aware of the green computing implications, or perhaps choose not offer this software (or maybe just turn off the affected features) in locales that retain the expensive laws. Most of these are legal/business decisions and not an SE/PL concern, but the SE/PL community can make it easier to orthogonalize features (and thus make simple to turn off without breaking everything else) whose compliance with local laws might be expensive.

Such tradeoffs can affect the entire software lifecycle, from conceptualization and development to deployment and operation. Many Societal Computing issues stem from recent trends in social computing, and possibly may even be solved by drawing on social computing models, such as the wisdom of crowds, collaborative filtering and so on; but many of the concerns are orthogonal and could possibly be addressed by novel approaches grounded in the SE/PL communities. We feel that the SE/PL community has a special role to play as it provides the substrate - the languages, compilers, design techniques, architectures, testing approaches, etc. on which all software systems are founded.

We describe our motivation in the next section and briefly outline some initial Societal Computing topics in the following sections. We then highlight a few research challenges posed by tensions between prospective technologies targeting these subareas.

1.1 Motivation

Anthony Kronman in his book "Education's End: Why Our Colleges and Universities Have Given Up on the Meaning of Life" opines that graduate programs at universities have become increasingly specialized. He argues that initially universities were much more broader in their scope and increasing emphasis on the research ideal has resulted in them becoming very specialized. He says: "Graduate students learn to restrict their attention to a single segment of human knowledge and to accept their incompetence to assess, or even understand, the work of specialists in other areas. [. . .] They are taught to understand that only by accepting the limits of specialization can they ever hope to make an "original contribution" to the ever-growing body of scholarship in which the fruits on research are contained." [59]

In the field of Computer Science as well, this increasing specialization is evident by the increasing number of publication venues that exist now and by how this number has changed over the years. A good indicator of the number of publication venues is the number of proceedings (for conferences and workshops) that are available in the ACM Digital Library [1]. This is shown in Figure 1. We see an exponential increase in the number of the publication venues in the last ten years. As the number of publication venues has increased, it has resulted in specialization of Computer Science into subareas and sub-subareas.

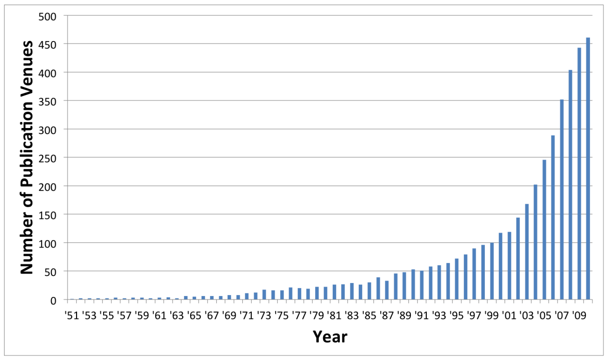


Figure 1: Number of Publication Venues in the ACM Digital Library from 1951 to 2010

While this research specialization is important and has resulted in our increased understanding of the field, it has also made our scopes very narrow. Our problem is an inverse to that of being a "jack of all trades and a master of none." Researchers have become experts in their specialized subareas (and sub-subareas) on Computer Science while being relatively unaware of the other subareas. Due to this narrowing of scope, researchers are not very aware of the advances made in the other subareas and in particular, the tradeoffs that might exist between them. Advanced research and progress made in one research subarea may have a negative effect on some other research subarea. This notion of tradeoffs is analogous to the concept of Pareto Efficiency [79] in Economics, which deals with the distribution of goods among a set of individuals in society. Pareto Efficiency refers to the state of distribution where it's not possible to make an individual "better off" without making some other individual "worse off." Not being in a Pareto Efficient state would imply that it is possible to optimize both (or multiple) areas; being in a Pareto Efficient state would imply that it is not possible to optimize one area without affecting the other one. In our case of Societal Computing, identifying such a state will be an important research challenge and this identification may not be possible without a detailed understanding of the different areas that we're trying to optimize.

We feel that such tradeoffs exist in many different areas and that a broadening of research scope is necessary to effectively address them. We need to take a much more holistic view of research. We describe some subareas of Societal Computing and the tradeoffs among them in the following sections.

1.2 Societal Computing Topics

In this section, we describe some research areas relevant to Societal Computing and we will highlight the tradeoffs among these areas in Section 1.3.

1.2.1 Privacy

Privacy in the context of social computing systems has become a major concern for the society at large. A search for the pair of terms "facebook" and "privacy" gives nearly two billion hits on popular search engines. Recent feature enhancements and policy changes in social networking and recommender applications – as well as their increasingly common use – have exacerbated this issue [17,40,53,121]. With many online systems that range from providing purchasing recommendations to suggesting plausible friends, as well as media attention (e.g., the AOL anonymity-breaking incident reported by the New York Times [7]), both users and non-users of the systems (e.g., friends, family, co-workers, etc. mentioned or photographed by users) are growing more and more concerned about their personal privacy [94].

Social computing systems, when treated in combination, have created a threat that we call "Correlation Privacy." Narayanan and Shmatikov [72] demonstrated a relatively straightforward method to breach privacy and identify individuals by correlating anonymized Netflix movie rating data with public IMDb data. A similar approach could potentially be applied to any combination of such data-gathering systems, so how to safeguard again these "attacks" may be a fruitful research direction. This is analogous to earlier work addressing queries on census data but, at that time, there were relatively few prospective attackers [2, 10]. There has been some initial work towards retaining privacy while still benefiting from recommendation systems (e.g., [12, 95]). There have also been other approaches such as k-anonymity [99], differential privacy [29], and applications of differential privacy to different domains [85, 89].

A related challenge is to make the existence of privacy threats more understandable to ordinary users who do not have a technical background and/or in cases where it's not very clear how users' information might be employed by the system, particularly germane for systems that provide APIs making it easier (than screen scraping) for third parties to utilize that information (e.g., [5, 34, 60]). There has been some recent work (e.g., [54, 97]) towards this end. One interesting option might be to make privacy more quantifiable, perhaps by introducing a notion of "Gullibility Factor" for privacy settings, say ranging from 0 to 1 with 1 being the least private. For example, we could say that the default settings for Facebook have a Gullibility Factor of 0.5, whereas for Twitter it's 0.2 (we made up these numbers). A simple scoring scheme might steer away fearful users, while encouraging the merely puzzled to consult one of the numerous "how to" guide articles on privacy settings from sources such as the New York Times [87] and the BBC [86].

While research efforts in this area have been promising, there is a lot of scope for further research. Researchers will need to be aware, in particular, of the tradeoff of Privacy with other Societal Computing topics and we elaborate on this in Section 1.3.

1.2.2 Cultural Differences

Different regions and cultures around the world vary in their notions and perceptions on what is acceptable and what shouldn't be allowed. An increasingly important area of Societal Computing will be building systems that can adapt automatically to different regions and cultures.

One important cultural difference we highlight here is Legal Challenges. Legal issues present additional research challenges for a wide range of software systems beyond just social computing applications. For example, privacy-related laws vary tremendously from country to country, e.g., consider Germany compared to the United States. Systems such as Facebook and Google Street View, which have been accepted by the government and individuals in the US, are facing many obstacles in Germany [94]. After the Second World War, Germany has legislated very strict privacy laws to prevent the government from persecuting its citizens. It is illegal in Germany to publish names or images of private individuals (including felons) without their permission [74]. Before allowing Google's Street View service, German data protection agencies asked Google to audit the information collected by their street view mapping cars. During the audit, they discovered that the cars were collecting personal information such as emails and phone numbers from unsecured wifi networks [8].

One intriguing example open research problem would consider the implications of regulation diversity on software. To deal with different cultures and customs, we would require novel modularization mechanisms beyond those employed for software localizations of keyboard, written language, the customs of different geographical regions, etc. We call this "Regulatory Localization." We feel that multidisciplinary research with other areas of Computer Science such as natural language processing would be highly beneficial.

Another example of a cultural difference is censorship. Different regions believe different things should be censored. For certain topics there is almost universal agreement on censorship (e.g., child pornography [106]); some others are more debatable (e.g., political or government secrets [93], web search results [14, 107], marijuana [45]). What's acceptable would depend a lot on the region and cultural preconceptions in place. As we build software systems that have users all over the world, we would need novel techniques for dealing with such issues. This also relates to the current debate on Net Neutrality [114]. If, for example, we know that certain states/regions are not being net neutral, would we design, develop, test, or operate our software systems differently?

A recent paper [41] also discusses how local laws can affect software engineering. Those authors focus on intellectual property laws and licensing, warranty and liability, and transborder data flows, and propose "lawful software engineering" research directions concerned with coping with the wide variety of legal constraints during software development and deployment. While that work falls within the scope of our proposal, we are primarily concerned with the potential interactions with other aspects of Societal Computing.

1.2.3 Green Computing

Green Computing (or Green IT) is "the study and practice of designing, manufacturing, using, and disposing of computers, servers, and associated subsystems [...] efficiently and effectively with minimal or no impact on the environment" [71]. With our oil reserves projected to exhaust in less than fifty years [110], and renewable energy sources still providing only a small fraction [32], Green Computing here and now is becoming more and more important and, indeed, vital to our children and grandchildren.

Investigating how to build greener software systems from an SE/PL perspective, in addition to the complementary algorithmic efficiency and systems perspective such as resource allocation, platform virtualization, and power management pursued by other computer science subdisciplines [68] will be important. For instance, say we could quantify complex software systems' behavior in terms of energy expended. There has been some initial research in this area such as [113], which tries to quantify the carbon footprint of a Google Search. Then we could investigate ways to make this quantification more modular, devise software architectures and design patterns intended to give developers and end-users more control over energy use, and invent testing methods that check for energy violations. And, further, rethink testing in general, perhaps pushing more testing into the field ("perpetual testing" [76]), to reduce pre-deployment energy consumption and, perhaps, better spread the burden across energy sources. If we could make this quantification more modular (perhaps to the level of individual functionality provided by large systems), we could then provide easy means for operators and end-users to disable unneeded modules, which may play a critical role in Green Computing, to reduce energy consumption on server farms, desktops, and the increasingly abundant mobile platforms. As a simple example, a system like Netflix could inform each user that it would save X amount of energy to disable automated recommendations and only enable them when and if really needed (note that user altruism is a very different kind of model than charging extra for certain functionality [75]). But quantifying which user-visible functionality saves how much energy may not be easy, particularly when systems are built by integrating components.

Our Societal Computing initiative envisions investigating the tradeoffs of Green Computing with the other areas and we highlight these tradeoffs in the following section.

1.3 Societal Computing Tradeoffs

While there is a lot of potential for novel research in these individual areas of Societal Computing, in this thesis we focus on the tradeoffs between these different areas and the research challenges that arise out of these tensions. A central discussion point is to consider the problem of how software methodologies and technologies aimed at reducing societal costs in one area can sometimes raise societal costs for another. For example, there may be clever ways to engineer social computing and other applications to protect privacy or enforce regulations that inherently consume vast CPU cycles and other resources, which could be considered "anti-green." We need a holistic view.

1.3.1 Privacy vs. Green Computing

Say we have developed an awesome new social computing system S whose privacy-preservation properties may be suspect. One possible approach would be to try correlating S with other popular social systems, such as Netflix, IMDb, Facebook, Amazon, etc., to determine whether privacy can indeed be breached and to what degree (e.g., are potentially all users at risk, only those who use a specific other social system, or only a small fraction of the latter with unique information). We might do this prior to public use of S, e.g., using an internal test team and/or informed beta testers (who might invent phony identities). Such an experiment could give us an estimate of the likely privacy breaches, and possibly point towards steps that could be taken to safeguard against them.

However, straightforward mechanisms that poke or data-mine for potential breaches would likely require substantial computational resources; while this kind of testing may be a good idea where Correlation Privacy is concerned, it may not be so good for Green Computing. And it also does not address correlation against future social computing systems or unexpected uses of our system. So instead we could wait until S has been populated by the general public and then periodically correlate a sampling, which might require fewer resources and/or better distribute the resource burden, as well as draw on other new social systems as they are launched. But by then any privacy threats could be actual rather than hypothetical, and consequent protective measures too late. What design and testing techniques can we devise to balance privacy with green computing, particularly in a context where subsystems might be developed by different organizations? Broadening of research scope will be important to be able to effectively address these concerns.

1.3.2 Privacy vs. Cultural Differences

As countries are increasingly trying to pass new privacy laws [9, 67] and companies are being taken to court and getting fined for privacy violations [23, 69], legal issues dealing with privacy will become even more complex. We believe that as countries mandate new requirements for privacy, there will be an important tradeoff between these laws and privacy issues - in particular, the Gullibility Factor. Say we have an awesome new system S that has users in different parts of the world. As each country might have (slightly) different privacy laws, our system would need to comply with all the different regulations. Imagine a user Fred who is a US citizen. We would need to comply with US regulations in this case and Fred would have set his privacy settings as needed. Now if Fred decides to travel to another country (say, Germany) for business or a conference, we might also need to comply with the German regulations for privacy. In addition, we might also need to comply with the EU regulations, which may or may not be the same as the German regulations. Having to comply with all these different regulations will only end up making privacy threats and settings harder to understand for users and might also result in less usable systems. Note that such conflicts and confusions needn't arise due to travel to different countries, but might also exist due to the different city, state, and federal rules. What techniques can we use to make privacy and privacy settings more understandable to ordinary users when we need to also comply with complex legal regulations? An understanding of the different research areas involved will be crucial to address the various research challenges that we face.

1.3.3 Green Computing vs. Green Computing

There is also an interesting (and recursive) tradeoff of Green Computing with itself. As part of the development of greener software systems, we may need to invest substantial computer resources. For example, social recommendation systems tend to rely on expensive data-mining, but developing a greener recommendation system that is kinder to the environment could also be quite expensive. In the worst case, the amount of resources spent on building such green systems may far outweigh the energy benefits of replacing their less-green counterparts with these new systems, a classic example of being "penny wise, pound foolish." How can we efficiently analyze this in advance of expending those resources?

1.4 How can we contribute?

A common theme in these tradeoffs is finding the right balance between the different areas of Societal Computing. If we haven't reached the Pareto Efficient state yet, it might be possible to optimize different areas simultaneously. Once we reach the Pareto Efficient state, trying to improve one of these areas might have an adverse effect of some other area. An important concern and a big research challenge will be trying to identify such a state - would this be pair-wise for the different subareas? would this be multi-variable across all possible areas? We believe that this will require a detailed understanding of the various Societal Computing areas. What to do once we reach the Pareto Efficient state gives us further food for thought. One approach to consider, even though it might be considered an anathema to all technological advances, is to spend more *human* time to reduce reliance on non-renewable resources. Most technology (since the dawn of time) has been designed to make humans more productive and to reduce the burden of work for humans. However, as resources start becoming scarce, humans may need to take on more of this burden. This might imply a greater reliance on design or code review instead of execution testing. We would then need to figure out how we could do reviews across different systems, e.g., to manually find Correlation Privacy problems. We might also encourage human policing of privacy violations and/or time spent in end-user training to reduce the Gullibility Factor rather than automated ways for detecting these.

One argument towards Societal Computing might be that many different communities - such as operating systems and networks - need to look at these problems as well. We agree that multidisciplinary research is crucial and we feel that the SE/PL community needs to expand its scope towards more multidisciplinary research efforts. As the rest of the CS community usually ends up writing software to implement their research ideas and put them into practice, we have the special (and perhaps enviable?) role of cutting across the various domains as we provide the underlying platform -i.e., languages, compilers, development techniques, etc.- for implementation for most systems. Naturally it behooves us to come up with ways of dealing with Societal Computing concerns. We could make available means such as design patterns, architectural metaphors, better tools, APIs, smarter compilers, better testing techniques, and new programming languages to deal with some of these concerns. We can help the other communities make an easier decision when it comes to the tradeoffs. We can also address how to implement these balanced systems. We feel that a broadening of research scope is very important and necessary to address the research challenges and in particular, the tradeoffs among the different areas of Societal Computing. Finding the right balance among the tradeoffs in these different research areas will be crucial.

To limit the scope of this thesis, we will focus primarily on privacy and its tradeoffs with other areas of Societal Computing.

2 Money for Nothing, Privacy for Free

As mentioned earlier, an important aspect of societal computing is the need to balance the tradeoffs amongst the different areas. The best case scenario is such a situation would be if it was possible to improve one area without adversely affecting any other area. E.g., if we could get privacy for free (*i.e.,* without having to worry about, say, green computing) in certain systems, it would make balancing the other tradeoffs easier. We now describe a research project where this is possible; we call it "Money for Nothing, Privacy for Free."

2.1 Differential Privacy for Free

In this project, we propose an approach, which we call "Privacy for Free," targeted towards online social systems. In particular, we focus on systems that already have access to user data such as purchase history, movie ratings, music preferences, and friends and groups and that use complex data mining techniques for providing additional social benefits such as recommendations, top-n statistics, and so on to their users. The problem we deal with is users who have intentionally disclosed data on a public system, entering their data via web browsers onto some website server that is known to make publicly available certain data-mined community knowledge gleaned from aggregating that data with other users — but the users don't want their data to be personally identifiable from the aggregate.

The main research question we try to answer here is — Is there an approach that can be used with complex web applications and software systems, that will achieve privacy without spending any extra resources on computational overhead? We believe it is — our key insight is that we can achieve privacy as an accidental and beneficial side effect of doing already existing computation.

The already existing computation in our case is weighing user data in a certain way — weighing recent user data exponentially more than older data to address the problem of "concept drift" [112] — to increase the relevance of the recommendations or data mining. This weighing is very common and used in a lot of systems [25, 50, 57, 70]. Recent work in the databases/cs theory communities on Differential Privacy [29, 66] led to our insight that our already existing computation for weighing user data is very similar to one of the techniques for achieving differential privacy. Intuitively, differential privacy ensures that a user's participation (versus not participating) in a database doesn't affect his privacy significantly. We provide more detailed information on Differential Privacy in Section 2.2. This resulted in the formulation of our hypothesis — if we change the concept drift computation so it matches the technique for achieving differential privacy (which would be a very minor and straightforward code change as the two techniques are very similar), would we get privacy as a beneficial side effect of addressing a completely different problem?

We show that it is indeed possible to get privacy as a beneficial side effect of addressing concept drift — thus, privacy for free — and this is the main contribution of this project. Our approach can be used in certain social computing systems and web applications to achieve "privacy for free," and we show the feasibility, sustainability, and utility of using this approach to building software systems. We also contribute to the discussion in the privacy community about how to define privacy and how to achieve it. Specifically, we suggest a new direction for designing (differentially, or otherwise) private algorithms and systems motivated by using the beneficial side-effects of doing some already existing computation.

2.2 Background

Here we provide some background information on Differential Privacy and Concept Drift.

2.2.1 Differential Privacy

In the 1970s, when research into statistical databases was popular, Dalenius [24] proposed a desideratum for statistical database privacy — access to a statistical database should not enable someone to learn something about an individual that cannot be learned without access to the database. While such a desideratum would be great for privacy, Dwork *et al.* [27, 29] showed that this notion of absolute privacy is impossible using a strong mathematical proof. The problem with the desideratum is the presence of "Auxiliary Information". Auxiliary Information is similar to, and a generalization of, the notion of Correlation Privacy mentioned earlier.

Dwork gives a nice example to explain how Auxiliary Information can be a problem when privacy is concerned — "Suppose one's exact height were considered a highly sensitive piece of information, and that revealing the exact height of an individual were a privacy breach. Assume that the database yields the average heights of women of different nationalities. An adversary who has access to the statistical database and the auxiliary information "Terry Gross is two inches shorter than the average Lithuanian woman" learns Terry Gross' height, while anyone learning only the auxiliary information, without access to the average heights, learns relatively little." An interesting observation made by Dwork is that the above example for breach of privacy holds regardless of whether Terry Gross' information is part of the database or not.

To combat Auxiliary Information, Dwork proposes a new notion of privacy called Differential Privacy. Dwork's paper is a culmination of the work started earlier and described in papers such as [15, 26, 28]. Intuitively, Differential Privacy guarantees privacy by saying that if an individual participates in the database, there is no additional loss of privacy (within a small factor) versus if he had not participated in the database. Formally, Differential Privacy is defined as follows: A Randomized function K gives ϵ-differential privacy if for all data sets D_1 and D_2 differing on at most one element, and all $S \subseteq Range(K)$,

$$\Pr[K(D_1) \in S] \leq \exp(\epsilon) \times \Pr[K(D_2) \in S] \tag{1}$$

The notion of all data sets D_1 and D_2 captures the concept of an individual's information being present in the database or not. If the above equation holds, it implies that if an individual's information is present in the database, the breach of privacy will be almost the same if that individual's information was not present. Differential Privacy is now commonly used in the database, cryptography, and cs theory communities [16, 27, 30, 88].

We like the definition of Differential Privacy due to its strong mathematical foundations, which can allow us to prove/disprove things theoretically. From a software web application developer's point of view, they can tell their users — "Look, our system is differentially private. So if you decide to use our system and give it access to your data, you are not losing any additional privacy (within a small factor) versus if you did not use our system. In other words, the probability of bad things happening to you (in terms of privacy) is roughly the same whether you use our system or not."

2.2.2 Achieving Differential Privacy

Dwork describes a way of achieving differential privacy by adding random noise. In the Terry Gross height example above, instead of giving the true average, the system would output average$\pm\delta$, where δ would be randomly chosen from a mathematical distribution. Thus, the adversary wouldn't be able to find out the exact height of Terry Gross. Since then, there have been many papers that have proposed different mechanisms for achieving differential privacy [16, 27, 30, 88].

A mechanism of note for achieving differential privacy was proposed by McSherry and Talwar [66] called the "Exponential Mechanism" (EM). The EM algorithm is as follows: Given a set of inputs, and some scoring function that we are trying to maximize, the algorithm chooses a particular input to be included in the output with probability proportional to the exponential raised to the score of the input using a scoring function. Thus, inputs that have a high score from the scoring function have an exponentially higher probability of being included in the output than those inputs that have a low score. McSherry and Talwar prove that this EM algorithm is differentially private.

Consider the Terry Gross example from above and let's assume that the database has historical data going back 100 years. The average heights of people change over time so giving an average height over the 100 years is not very useful. If the scoring function we use is to maximize the recency of data, newer data elements will be chosen with exponentially higher probability that older data elements to be included in the average. Since we are doing this probabilistically, the exponential probability weighing ensures that the exact answer is not revealed and that differential privacy is maintained. This EM algorithm is one of the corner stones of our "Privacy for Free" approach and we describe how it's used in the next section.

2.2.3 Concept Drift

People's preferences change over time — things that I like doing today may not be things I liked doing 10 years ago. If data is being mined or recommendations being generated, the age of the data needs to be accounted for. To address this problem, the notion of Concept Drift was formed [112]. This problem needs to be addressed by any field that deals with data spanning some time frame (from a few hours to months and years). An example class of systems that need to address the problem on Concept Drift is Recommender Systems. Many recommender systems use Collaborative Filtering (CF), *i.e.,* recommending things to an individual by looking at what other users similar to the individual like [49, 70, 117]. CF algorithms typically look at the activities of individuals from the past (movies watched, things bought, etc.) and use this to derive recommendations. However, people's preferences change over time. For example, when I am in college and taking a lot of classes, I might buy a lot of textbooks from Amazon. When I graduate, I may not need textbook recommendations. This is exactly the kind of problem that Concept Drift tries to address.

Other example classes of systems that need to address this problem are social software systems [11], systems for collaboration and awareness [111], systems that mine online software repositories [19], etc. For these kinds of systems, there is a lot of old and recent data available and weighing certain data differently might be essential.

2.2.4 Addressing Concept Drift

There have been many different solutions proposed to address the problem of Concept Drift [55, 58, 112]. A particular solution of note is the Exponential Time Decay Algorithm [22] (ETDA, henceforth). ETDA weighs things done recently exponentially higher than things done in the past. It gradually decays the weight of things done in the past so that things done in the distant past do not affect the outcome as much as things done recently, thus addressing the problem on Concept Drift.

$$g(x) = \exp(-l \times x), \text{ for some } l > 0 \tag{2}$$

The non-increasing decay function using by ETDA is shown in Equation (2). ETDA is very popular and used by a lot of systems [25, 50, 57, 70]. For the rest of the thesis, we refer to this as the CD (Concept Drift) algorithm.

Consider the Terry Gross example again and let's assume that the database has historical data going back 100 years. As average heights change over time, the CD algorithm will weigh newer data exponentially higher than older data resulting in a weighted average height. This would reflect

```java
public double getWeightedValue() {
  double value = 0;
  for(int i=0; i<array.length; i++) {
    double weight = Math.exp(-i);
    value += weight*array[i];
  }
  return value;
}
```

Listing 1: Java code for the CD algorithm

```java
public double getWeightedValue() {
  double value = 0;
  for (int i = 0; i < array.length; i++) {
    double weight = Math.exp(-i)*0.5;
    double probability = Math.random();
    if (probability < weight) {
      value = array[i];
      return value;
    }
  }
  return value;
}
```

Listing 2: Java code for the EM algorithm

the recent trends but also account for older data. The CD algorithm is the another corner stone of our approach and we build on it more in the next section.

2.3 Privacy for Free

The EM algorithm can use a variety of scoring functions — McSherry and Talwar show different scoring functions for privacy preserving auctions [66]. In such scenarios, the EM and CD algorithms are not similar. Using the timestamp scoring function is what makes them similar to each other. The CD algorithm uses exponential weighing over the data while the EM algorithm chooses inputs with probability proportional to the exponential of the scoring function.

Only if we choose the scoring function for the EM algorithm to be the timestamp of the data, the two algorithms becomes similar. The CD algorithm is deterministic and weighs new data exponentially higher than older data; the EM algorithm is probabilistic and chooses new data with an exponentially higher probability than older data.

The Java code for the CD algorithm and the EM algorithm using the timestamp scoring function are shown in Listings 1 and 2 respectively. In terms of running time complexity, the CD algorithm is $O(n)$. For EM (using the timestamps scoring function), the worst case is also $O(n)$. However, as we use randomization, the expected running time is sublinear — $o(n)$.

This is the crux of this project — if existing systems that already use the CD algorithm modify the code to use the EM algorithm instead, they would, as an added benefit, get the main advantage of the EM algorithm — differential privacy. Further, this privacy would not require any extra computational overhead and thus, we would get privacy for free.

Since these two algorithms are very similar, it would require a very small and straightforward change to the code to change from the CD algorithm to the EM algorithm. We would need to replace the CD code with the EM code shown above. This would be a one-time change and could be done by adding a new library method for EM or done statically via refactoring and could even be automated.

The important requirement for the differential privacy guarantees to hold are that all the data access must be done via the EM algorithm, which could be implemented as a separate class or be part of a library or the data model, etc.

2.4 Evaluation

Our approach requires implementing (or substituting an existing implementation of the CD algorithm with) the EM algorithm. To evaluate our approach, we implemented the EM and CD algorithms and investigated the differences in these. Our goal was to answer the following research questions:

RQ1: Feasibility—Does using our approach guarantee differential privacy?

RQ2: Utility—Does using our approach affect the utility of the system to give meaningful recommendations or mine data?

RQ3: Sustainability—Can our approach be sustainable? Can using our approach result in no additional computational resources for privacy?

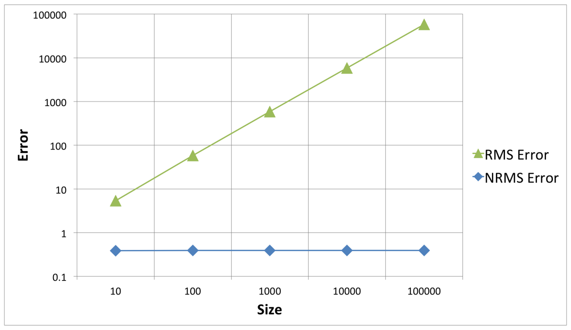
With RQ1, we aim to prove the primary benefit of our approach — guaranteeing privacy. Our goal is to show that it does indeed guarantee differential privacy making it suitable to be used in a variety of social systems and web applications.

With RQ2, we explore the utility of using our approach. A "straw man" way to guarantee privacy for any recommender/data mining system is to give a random answer every time. This would not require any clever technical solutions, but this would be very bad for the overall utility of the system — the goal of most such systems is to provide relevant information. There exists a tradeoff between accuracy and privacy and we explore this here. We aim to show that, using our technique, there is a small loss in accuracy and that this loss in accuracy scales very well (roughly constant) as the size of the system increases. Thus, if a small loss in accuracy is acceptable, we can get privacy for free without spending any additional computational resources.

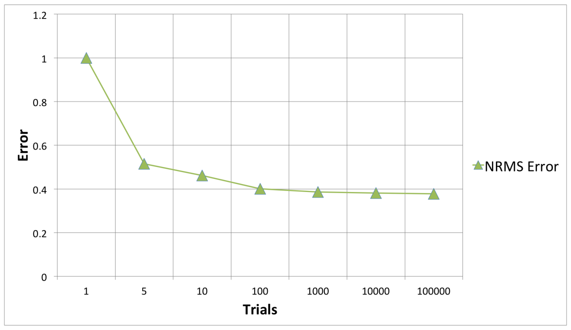
With RQ3, we aim to show the sustainability benefits of using our approach. We show that using our approach (and the EM algorithm) requires less CPU time than the equivalent CD algorithm. Not only do we not need any additional computational resources, we should be able to reduce computational needs by using our approach.

2.4.1 RQ1 — Feasibility

Our approach requires the use of the EM algorithm for all access to the data. The EM algorithm that we require is exactly the same as the one proposed by McSherry and Talwar [66]. The algorithm they propose can work with different scoring functions that weigh the data differently — in our case, the scoring function we use is the timestamp of the data. Our use of the EM algorithm in



(a) RMS and NRMS Error vs. Size of data set



(b) NRMS Error vs. Number of Trials

Figure 2: Experimental Results

our approach can thus be viewed as an instantiation of the general EM algorithm. McSherry and Talwar show a theoretical proof for the EM algorithm to be differentially private. We do not repeat the proof here and we encourage the interested reader to look at the paper (page 5 of [66]). As all data access happens via the EM algorithm, our approach also guarantees differential privacy.

2.4.2 Methodology

For RQ2 and RQ3, we carried out experiments to validate our hypotheses. We use synthetic data as there are no benefits of using real world data for our hypotheses. We create an array of size n and randomly fill it with values from 0 to $n - 1$. Each element has a timestamp associated with it to simulate user activity — for the purpose of this experiment, we assume that the timestamp is the array index. A lower array index indicates that the item is newer. Thus, we want to prefer items with a lower index in the output as these items indicate things that are done recently.

Using the differential privacy EM algorithm [66], we choose the scoring function to be maximized by returning a value with as low an array index as possible. Thus, we choose elements from the array with probability based on their array index.

In the experiments, we randomly generate the array and compute the score using the CD and the EM algorithms. We then plot the RMS and normalized RMS errors between these two algorithms. The error is the difference in the score returned by the CD and the EM algorithm. The CD algorithm will give us the "true" score; the EM algorithm (as it tries to preserve privacy) will give us a close approximation. We discuss the results in the following subsections.

2.4.3 RQ2 — Utility

For the first set of experiments, we varied the size of the array and plotted the RMS and normalized RMS errors between the CD and EM algorithms. The results are shown in Figure 2a. To smooth out the noise in the experimental results (as CD is a deterministic algorithm while EM is a probabilistic one), we ran the experiment 1000 times with each array size and took averages. The graph shows us that as the size of the input array increases, the RMS error increases linearly — this is expected as with larger array sizes, the entries in the array have correspondingly larger values (due to our methodology), resulting in linearly increasing RMS error. Meanwhile, the normalized RMS error is roughly constant.

This shows us the tradeoff between accuracy and privacy. We observe that in these experiments, the loss of accuracy is relatively small — the normalized RMS error is less than 0.4. Thus, irrespective of the data set size, switching to the EM Algorithm (as required by our approach) from the CD Algorithm will not worsen the accuracy of the algorithm by more than the constant factor, and we have the added benefit that the EM algorithm also guarantees differential privacy. Whether the loss of accuracy is acceptable or not (or a worthy price to pay for the free privacy) is subjective and we deliberately do not enter a philosophical debate here (is accuracy of the system more "important" than user privacy? who decides this? the user? the web application developers?). Many papers in the database and theory communities have explored the tradeoffs between privacy and accuracy (*e.g.,* [15,26,65,66]) — our key point in this section is that yes, there is a loss of accuracy, but no worse than accepted in [65]. A limitation of our approach is that if this loss of accuracy is not acceptable for certain systems, our approach will not work.

For our second set of experiments, we varied the number of trials keeping the size of the array fixed to 1000. As the value computed using the EM algorithm is probabilistic in nature, we carry out multiple runs (called trials here) and take the average value over all the trials to smooth out the value. The graph plotting the NRMS error vs. the number of trials is shown in Figure 2b. This graph shows us that as the number of trials increases, the NRMS error reduces. Thus, initially, even though there may be a bigger error between the CD and EM algorithms, in the long run, the error will be small (but not zero, as a zero error would imply returning the accurate answer and thus, not preserving privacy).

With these set of experiments, we explored the utility of our approach. For an existing system (that may already use an algorithm similar to the CD one), a one-time change would be required to add in the EM algorithm and retrofit the system to our approach. This change is relatively straightforward and could even be automated. Making such a change, albeit results in a small loss of accuracy, gives the huge benefit of getting privacy for free without spending any additional computational resources.

2.4.4 RQ3 — Sustainability

For RQ3, we want to show the sustainability of our approach. With the EM algorithm in place, what we ideally want is that our system does not take any additional computational resources. We decided to use the CPU processing time to estimate the computational resources needed by the two algorithms. We instrumented the CD and EM algorithms and measured how long they took in the first set of experiments in Section 2.4.3 above. The resultant graph is shown in Figure 3. The graph shows us that for all data sizes the EM algorithm took less CPU time than the CD algorithm.

Not only does the EM algorithm not require any additional computational resources, it actually reduces the existing computation. Thus, changing to our approach will make the software system even more sustainable.

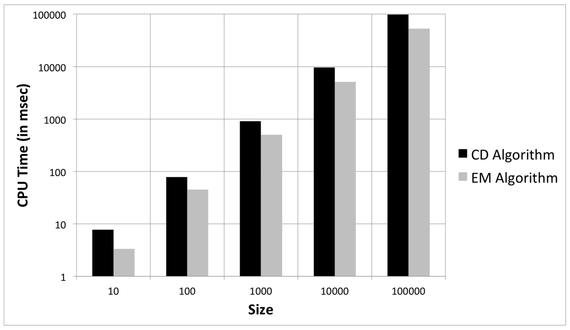


Figure 3: CPU Time (in msec) vs Size of data set

2.4.5 Threats to Validity

The notion of Differential Privacy may not relate to the user-centric view of Privacy as users might think it “strange” that the system assumes that bad things can happen anyway — the guarantee it gives is just regarding whether the user data is part of the system or not. While that is true, we feel that differential privacy has many compelling arguments in its favor — the biggest, for us, is not having to decide what data is sensitive and what is not. The differential privacy algorithms treat all data as sensitive making it easier not to leak data by accident. One would, therefore, not have to deal with the subjective nature of deciding what’s sensitive. We also feel that the guarantee might actually make it even more compelling for the user. From their point of view — “if bad things are going to happen anyway, it’s not going to hurt me much more if I participate.. so there’s no harm in participating.”

We used synthetic data in our evaluations rather than real-world data. For the research questions that we had — feasibility, utility, sustainability — synthetic data was sufficient. For Feasibility (RQ1), we use the theoretical proof from [66] so don’t need data. For Utility (RQ2) and Sustainability (RQ3), we care only about the comparisons between the CD and the EM (and not the actual numbers in the experiments), so synthetic data — which was easier to work with — suffices. We would, however, need real data if we were doing, *e.g.,* surveys and our research question was if people thought the new system gave similar usability.

Finally, this work doesn’t help in scenarios of non-temporal data access. We used the IMDB/Netflix examples earlier to make the general problem familiar to the reader; we address a special case of the problem where timestamps are available. In the differential privacy area, it’s proven that for *any* method that has any utility, there exists side information that will break privacy on individual records. With differential privacy approaches such as the EM algorithm, the guarantees that exist for each individual are that participating in the database will not add to the risks that are already there.

2.5 Related Work

Privacy has become an increasingly important topic for the community at large. A lot of different research communities are looking at the impact of privacy and techniques for improving privacy for users. Some examples of these communities are sociologists, computer scientists, HCI, etc. We discuss some of the relevant related work next.

Fang and LeFevre [39] proposed an automated technique for configuring a user’s privacy settings in online social networking sites. Paul *et al.* [80] present using a color coding scheme for making privacy settings more usable. Squicciarini, Shehab, and Paci [98] propose a game-theoretic approach for collaborative sharing and control of images in a social network. Toubiana *et al.* [101] present a system that automatically applies users’ privacy settings for photo tagging. All these papers propose new techniques that are targeted to making privacy settings “better” (*i.e.,* more usable, more visible) from a user’s perspective. Our approach, on the other hand, targets the internal algorithms such as recommendations used by these systems.

There have been some recent papers on data privacy and software testing. Clause and Orso [21] propose techniques for the automated anonymization of field data for software testing. They extend the work done by Castro *et al.* [20] using novel concepts of path condition relaxation and breakable input conditions resulting in improving the effectiveness of input anonymization. Our work is orthogonal to the papers on input anonymization. The problem they address is — how can users anonymize sensitive information before sending it to the teams or companies that build the software? The problem we address is — how can systems that already have access to user data (such as purchase history, movie preferences, and so on) be engineered so that they don't leak sensitive information while doing data mining on the data? Further, the aim of our approach is to provide privacy "for free," *i.e.,* without spending extra computational resources on privacy. The input anonymization approaches require spending extra computation (between 2.5 minutes to 9 minutes) as they address a different problem. We believe that the our approach can be combined with the input anonymization approach if needed. If users are worried about developers at the company finding out sensitive information, input anonymization is essential. If, however, they are worried about accidental data leakage through the data mining of their information, using the "Privacy for Free" approach may be more suitable. This would also make the software system more sustainable as we don't spend any computation doing the anonymization of the inputs.

Taneja *et al.* [100] and Grechanik *et al.* [44] propose using k-anonymity [99] for privacy by selectively anonymizing certain attributes of a database for software testing. Their papers propose novel approaches using static analysis for selecting which attributes to anonymize so that test coverage remains high. Similar to above, our approach is orthogonal as we focus on an approach that will prevent accidental leakage of sensitive information via data mining or similar techniques. Further, these approaches using k-anonymity also require significant additional computational resources and thus, may not be sustainable when energy resources are scarce.

The testing problem above is concerned with internal data that users keep on their own computers and do not want to disclose outside their own computer (or put into a server and the testing is on that server software, but the data was understood to be specific to that user and never aggregated with other users). The problem we deal with instead is users who have intentionally disclosed data on a public system, entering their data via web browsers onto some website server that is known to make publicly available certain data-mined community knowledge gleaned from aggregating that data with other users — but the users don't want their data to be personally identifiable from the aggregate.

Finally, work on input anonymization and k-anonymization both focus on software testing whereas our approach focuses on an approach for building privacy preserving systems or re-engineering existing software systems with minimal code changes (since only the parts affected need to be changed) with a specific goal — to make privacy sustainable and not require additional resources.

There has also been a lot of work related to data anonymization and building accurate data models for statistical use (*e.g.,* [3, 33, 61, 83, 109]). These techniques aim to preserve certain properties of the data (*e.g.,* statistical properties like average) so they can be useful in data mining while trying to preserve privacy of individual records. Similar to these, there are has also been work on anonymizing social networks [13] and anonymizing user profiles for personalized web search [120] The broad approaches include aggregating data to a higher level of granularity or adding noise and random perturbations. As we are interested in sustainable ways of achieving privacy, these approaches are not applicable as they typically require (a lot of) extra computational effort.

While there has been a lot of interest (and research) in data anonymization, we would like to reiterate that only data anonymization might not be enough. Narayanan and Shmatikov [72] demonstrate a relatively straightforward way of breaking the anonymity of data. They show how it is possible to correlate public IMDb data with private anonymized Netflix movie rating data resulting in the potential identification of the anonymized individuals. Backstrom *et al.* [6] also describe a series of attacks for de-anonymizing social networks that have been anonymized to be made available to the public. They describe two categories of attacks — active attacks where an evil adversary targets an arbitrary set of users and passive attacks where existing users try to discover their location in the network and thereby cause de-anonymization. Their results show that, with high probability and modest computational requirements, de-anonymization is possible for a real world social network (in their case, LiveJournal [18]). Finally, Zheleva and Getoor [119] show it's possible to infer private profiles of users on social networks based on their groups and friends.

2.6 Discussion

The crux of this project, and the novel idea, is that it is possible to combine two existing approaches to *increase* the degree of privacy in social computing systems, under certain conditions. This poses an interesting open problem — Are there other algorithms that we currently use for solving some problem that also accidentally provide privacy or some other added benefit?

A lot of research in the Theory and Cryptography community on Differential Privacy has focused on Mechanism Design [16, 27, 30, 88]. Mechanism Design is the process of coming up with new mechanisms that are differentially private and solve certain problems in domains such as machine learning and statistics. The previous sections hint at an interesting avenue of future research — Mechanism Discovery. We discovered how the CD algorithm as a side effect may provide differential privacy for free. It might be fruitful to look at currently used algorithms in varying domains and see if they too, as a side effect, provide differential privacy. This might lead to the discovery of generalized mechanisms for differential privacy that can be used in other domains, which have not yet been proposed or discovered by Theory and Cryptography researchers. Mechanism Discovery might act as a great complement to the Mechanism Design research.

In order for Mechanism Discovery to be successful, a greater emphasis must be placed on Multidisciplinary research. Even though there is some research in recommender system privacy [12, 95], most of the papers do not use a formal and precise definition of privacy. Our community could benefit a lot from the precise and formal use of differential privacy. Similarly, most of the Theory and Cryptography community may not be aware of the privacy research done by our, or other, communities. There might be a lot of interesting discoveries of mechanisms suitable for differential privacy. The only way any of this can be achieved is by a greater emphasis on Multidisciplinary research using areas such as systems, theory, cryptography, web, and databases.

3 Privacy Requirements — Understanding the needs of the users and developers

Societal Computing aims to address the tradeoffs among the different areas such as Privacy, Green Computing, Cultural Differences, and so on. The "Privacy for Free" approach outlined in the previous section is the best-case scenario — an example of a situation where we can improve something in one area (in this case, privacy) without making something else worse (in this case, green computing). In general, this might be hard to do and we would need to gain a better understanding of the individual areas to understand what the tradeoffs are. The first step in this direction is a (currently on-going) research paper that aims to understand what end-users and software developers expect from social software systems as far as privacy is concerned.

A few open questions and hypotheses, which we aim to explore and answer, are:

- **Q**: For an end-user, does privacy matter? What aspects of privacy are more important?
- **H**: End users may not care as much about privacy if they know exactly how their data is being used. If this is true, then we can focus on making privacy understandable rather than coming up with clever technical solutions
- **Q**: As a software developer/program manager, what do I need to build a system that ensures the privacy of the users?
- **H**: End users and software developers have very different concerns as far as privacy is concerned. If this is true, we need to ensure that systems that are built take into account both of these points of view (and possibly others from other stakeholders).
- **Q**: Does geography have any impact of privacy expectations?
- **H**: Since the US has much weaker privacy laws than the EU, US users and developers care less about privacy. If this is true, should we customize systems based on where the users are?
- **Q**: Do we need different privacy mechanisms for technical vs. non-technical users?
- **H**: Non-technical users prefer more coarse-grained options whereas technical users prefer more control and fine-grained options. If this is true, should we customize privacy-relevant user interfaces based on the technical background of the users?

To answer these questions and validate/invalidate the hypotheses, we created a large-scale user study using an online questionnaire followed by semi-structured interviews. Our questionnaire is shown in Appendix A. We already have IRB approval for the study (valid till 07/16/2014) and we have started the study. So far, we have 157 respondents who have filled out the online questionnaire and 15 participants for the follow-up interviews. The respondents have been primarily from Asia and North America and we plan to get more respondents, in particular, from Europe.

4 Making Privacy Understandable — Crowdsourcing Privacy Settings

A similar, and related, challenge to the above is to make privacy more understandable to end-users. Imagine systems like Facebook which have complex user interfaces and that change the privacy settings often. Numerous studies have shown that the privacy settings of users are not what they intended them to be [52, 63, 64].

Our approach towards solving this problem is to use Crowdsourcing. For example, if most of my friends have their home address as private, maybe that would indicate to me that I should change my settings to make my address private as well. Thus, for any user, we can analyze what their settings are using the API and compare this to canonical settings. This canonical list can be generated in multiple ways as preferred by the users — it could be a list of "trusted" friends, it could be all friends of that user, it could be all of my friends, etc. This comparison will be shown to the user in multiple forms (entire list of differences, a "score" between 0 and 1 that tells the users how private/public they are compared to the canonical list, etc.). Users can now choose to keep their settings the same as they were before, modify them manually, or just mimic the settings from the canonical list.

Even if the privacy settings are what they intended, maybe the users can benefit from seeing what others do on the social networking website. Note, an important issue here is trust: I, typically, would not trust any random friend or user of the website; my real life trust of a set of friends will impact who I want to mimic.

We have already collected user data for 516 users on Facebook (as part of the project described in the next section). We have a draft of the user study (shown in Appendix B) and we plan to submit this for IRB approval in the next few weeks. An added benefit to this project is that the collected data will give us an ethnographic view of what people's settings on Facebook (and other such systems) are.

5 Detecting Privacy Bugs via End-User Regression Testing

End-user Software Engineering (EUSE) is becoming an increasingly important as "computer programming, almost as much as computer use, is becoming a widespread, pervasive practice." [56]. EUSE ranges from requirements and design to testing and debugging. End-user testing, in particular, is important for privacy because the end-users have little or no say in the functional specifications of or changes to social computing software, and because its online software they cannot avoid upgrading after each change or continue to use an "old version." Plus well-known social computing systems have an established history of making changes that breach privacy with no a priori ability for end-users to opt out [40]. But the end-users are not, in general, trained software engineers so any methodology and technology must be simple and easy to use without training.

Consider the following scenario for Pete – a user of a social system like Facebook. Pete is comfortable using websites and computers, but doesn't have a very strong technical background in Computer Science or Software Engineering. He is worried about his privacy when he uses Facebook though. There has been a lot of media coverage about privacy concerns, how they keep changing their privacy policy periodically, how hard it is to figure out all the privacy settings, and so on and this has caused Pete some concern. Pete likes using the system to keep in touch with his friends and professional colleagues, but he doesn't want strangers to have access to his personal information, photos, likes, dislikes, etc. He has used some of the "how to" guides to configure his settings to what he wants to them (or so he thinks).

A scenario like this raises a number of interesting software engineering research challenges:

1. **R1**: Users' Mental Model of Privacy — How can we make complex privacy settings easier to understand and verify for Pete? (*e.g.,* If I think my photos are shared with only my friends, is that really the case?) — **Requirements Engineering for Privacy.**
2. **R2**: Code/API Bugs — How can we detect if privacy settings that are in place remain the same as the software evolves and changes over time? (*e.g.,* If my photos are currently only shared with my friends, how do I know that they won't "automatically" get shared with everyone due to a software bug?) — **Regression Testing for Privacy.**
3. **R3**: Policy Changes — How can we detect system wide policy changes that might cause privacy settings to change? (*e.g.,* If my photos are private right now, how do we detect if there a policy change that makes all photos publicly accessible?) — **Regression Testing for Privacy.**

Ideally, for users like Pete who do not have access to the source code of systems like Facebook, we want to do this from an *end-user* perspective. In this project, we present a novel technique that leverages a social, crowdsourced approach for detecting bugs from the end-user perspective. To the best of our knowledge, this is the first technique that leverages regression testing for detecting privacy bugs.

Continuing the scenario above – consider Roger, a friend of Pete on the social system. Roger can manually monitor what part of Pete's information is visible to him. This monitoring can be done periodically as often as Pete/Roger deem necessary – say, every day, every hour, once a month, and so on.

Using this monitoring, Roger can inform Pete when the information he sees changes. For example, he might suddenly see a whole lot of new information that is now visible. This might be due to: (1) Pete added more information manually; (2) Pete changed his privacy settings either deliberately or accidentally; (3) Pete didn't do anything – there is a bug in the code or API, possibly due to code changes; and (4) Pete didn't do anything – the social system made a wide policy change where this information for many or all of its users is now visible. Pete, now, using this feedback from Roger, can decide whether it's ok for the new information to be visible and take the appropriate actions such as changing the settings back to what he wants them to do, reviewing the privacy settings or doing nothing.

This, however, can be very tedious for Roger to have to do all this manually, particularly, if frequently done, and he could easily forget to check certain things. Thus, automated monitoring is essential and can be done if the system provides an API. A lot of social systems like Facebook [34], Twitter [102], Last.fm [60], and Google+ [43] do provide an API whose main purpose is to build an ecosystem of app developers for the system. We can leverage such APIs where possible and if an API doesn't exist, the same goal can be achieved via screen scraping.

This is our broad approach — using one's friends for detecting potential privacy violations. We call this ***Social Testing***. There are more specific details that need to be dealt with based on the platform and API and we discuss this in Sections 5.2 and 5.3. This latter section also contains details about the feasibility of this approach and the kinds of privacy bugs it has helped us uncover. The main advantages of this approach are:

1. We don't need access to source code for detecting privacy bugs. Hence, this makes it very suitable to be employed by end-users (rather than software programmers building the system).
2. It leverages the social nature of these systems for detecting these bugs.
3. It can detect privacy bugs due to changes in the code, *i.e.,* regression testing for privacy.

The contributions of this project are:

- A novel software testing technique, called social testing, for the social circles of end-users to detect privacy bugs using regression testing. Social testing could potentially also be used for applications other than privacy preservation in social systems, such as in the multi-player gaming community;
- Two prototype tools that implement our technique for Facebook and Twitter; and
- A large empirical evaluation of our technique that demonstrates: (1) the feasibility and utility of our technique; and (2) the different kinds of bugs it can help detect.

5.1 Background and Motivation

Many recent studies on online social networks show that there is a (typically, large) discrepancy between users' intentions for what their privacy settings should be versus what they actually are [52, 63, 64]. For example, Madejski *et al.* report that, in their study on Facebook, 94% of their participants ($n = 65$) were sharing something they intended to hide and 85% were hiding something that they intended to share. Liu *et al.* [63] found that Facebook's users' privacy settings match their expectations only 37% of the time. This is **R1** mentioned in the previous section.

In addition to the problem of understanding existing privacy settings, there are two orthogonal problems. First, there might be software bugs in the implementation of the privacy settings, which results in over-sharing or under-sharing of information, and as software evolves over time, this might introduce new bugs. This is **R2** mentioned earlier.

Second, systems like Facebook change their policies on privacy often and these changes in policy usually end up confusing users even more. Dan Fletcher [40] writes: "In the past, when Facebook changed its privacy controls, it tended to automatically set users' preferences to maximum exposure and then put the onus on us to go in and dial them back. In December, the company set the defaults for a lot of user information so that everyone — even non-Facebook members — could see such details as status updates and lists of friends and interests. Many of us scrambled for cover, restricting who gets to see what on our profile pages." This is **R3** mentioned earlier and these are the main research problems that we are trying to solve with our approach.

5.2 The Social Testing Approach

The broad technique for our social testing approach is to use one's friends to help with software engineering problems. Our approach leverages the inherently social aspect of these systems, which are used for interacting and communicating with other users. This approach will apply only to systems where users are members of possibly overlapping groups and input information intended to be shared with some of these groups they are members of but not with other groups they are members of. This includes the cases of the singleton group – just me – and the universal group – everyone who uses the system, or anyone who uses the internet since many social systems often allow certain access with no login at all.

This technique could apply towards many different functional and non-functional requirements for end-users such as privacy, performance, and so on. In this project, we focus on privacy testing and in particular, on **R2** (detecting privacy bugs in code/API implementations) and **R3** (detecting system wide policy changes for privacy). There are two possible kinds of privacy violations:

- Over-sharing — From a user's point of view, this piece of information should have been private, but it can be viewed by others.
- Under-sharing — From a user's point of view, this piece of information should have been public, but it cannot be viewed by others.

We deal with both of these types of privacy violations. As our technique is intended for end-users, we assume that there is no access to source code. The main crux of our technique is — a user can choose his/her friends to periodically monitor what's visible to them via the social system. When they see a change in what's visible, this might be a privacy violation and they can inform the user. Thus, the aforementioned privacy violations, from the point of view of the tester, become: (1) Over-sharing — seeing more than you should; and (2) Under-sharing — seeing less than you should.

Thus, we use a social approach for detecting privacy bugs. The algorithm for our technique (from the tester's point of view) is outlined next: *a*) Implement/Download/Build a wrapper that can "talk" to the system under test (via an API, screenscraping, etc.). *b*) Generate a list of users to monitor. *c*) Decide on the policies (how often to monitor, which things to monitor). *d*) Based on the policies, use the wrapper to monitor the user(s). *e*) Generate diffs (*i.e.,* differences) between the information just received and from the previous run. *f*) If there is a diff, inform the user on what changed. *g*) Repeat steps 4-6, as needed and update steps 2-3, when necessary.

We discuss the platform and API specific implementation issues, examples of privacy bugs that we found, and how this technique can help address **R2** and **R3** in the next section.

5.3 Empirical Evaluation

For our empirical evaluation, we built two prototype tools: one for Facebook; one for Twitter. Using these tools, we evaluated how our technique could help addressing **R2** and **R3**. In particular, we had two specific research questions for our empirical evaluations:

RQ1: Feasibility — Does using our technique help in detecting privacy bugs?
RQ2: Utility — What kinds of bugs does it detect? Does it help with, both, **R2** (Code/API Bugs) and **R3** (Policy Changes)?

5.3.1 Privacy and Facebook

Facebook is a great example for doing an empirical evaluation on privacy as it follows a fine-grained privacy model. It has many different privacy parameters and options; broadly, users can choose who gets to see what type of data with a lot of granularity. It provides an API, called the Graph API, that represents the Facebook social graph using objects and connections between objects [35]. Examples of the objects include User, Events, Groups, and Applications. The User object contains fields such as name, gender, and birthday and "connections" such as albums, family, groups, likes, movies, and videos. Table 2 in Appendix C shows a complete list of User Connections in Facebook and is available on the Facebook User API page [36].

In general, there is a lot of flexibility for the user to choose the privacy settings for all of these. Users can share the data with no one, with selected friends, with all friends, with friends of friends, with certain networks (such as "Columbia University"), and with everyone. Users can also allow certain apps to access this information.

Prototype Tool Our tool is a prototype implementation of the social technique for detecting privacy bugs in Facebook. It is easily configurable and can fetch any data provided by the Facebook API. For the purposes of this study, we focused on getting only the User Connections from [36]. The prototype tool consists of two separate components: the Data Monitor and the Diff Visualizer.

The Data Monitor is implemented as a set of Ruby scripts. It uses the Koala library [4], which is a Facebook library for Ruby and supports the Graph API. The Data Monitor works as follows: First, given a list of users, for each user, it uses Koala to get data for that user. It gets all the data

```
======May 25 2012 and May 27 2012======
friends - Old: 2, 783, New: 2, 784
events - Old: 2, 1, New: 1, 0
feed - Old: 2, 15, New: 2, 16
likes - Old: 2, 202, New: 2, 200
posts - Old: 2, 6, New: 2, 7
tagged - Old: 2, 9, New: 2, 10
======May 27 2012 and May 28 2012======
friends - Old: 2, 784, New: 2, 783
posts - Old: 2, 7, New: 2, 8
tagged - Old: 2, 10, New: 2, 9
```

Listing 3: Sample Diff Output for a user. events is an example of a Major Difference; the others are all Minor Differences.

listed in [36] (except the "picture" connection, which didn't exist when we started collecting data). The Facebook API supports a "Batch mode" for making data requests and we use this mode to reduce the load of the servers and to get data more efficiently. Once the Data Monitor has the data, it writes it out to a log file. We create a separate log file per user. To limit the data that needs to be stored and also for privacy reasons, we do not store the entire data; we keep only the count of data items and codify the data received according to the schema defined below:

- **0, nil** — This is used when the API returns an error. This is typically a permissions error, but can also include other server side errors.
- **1, 0** — This is used when the API returns an empty data set. This means that either there is no data in that category or that the data exists but has been hidden by the user. Note that with the latter case, it will return a 1, 0 and not a 0, nil.
- **2, x** — This is used when the API returns some data. x is the count of the number of items received. For example, if there were 10 locations that the user had been tagged at, we would log 2, 10.

The log file, thus, contains multiple lines of data. Each line contains two things: (1) The current timestamp when the data was received; and (2) An array containing the username and the 41 arrays for the connections encoded into the schema shown above.

The Diff Visualizer, which is also a set of Ruby scripts, parses each log file and creates a human readable output if there is a diff (*i.e.,* difference) between any consecutive runs for a user. If there is a difference, it will print out the pairwise timestamps and what the old and the new values are. We divide a difference into two categories: a Major Difference and a Minor Difference. A Major Difference occurs when, for a certain connection, the data received changes the codifying categories. For example, if music was 2, 8 and became 1, 0, this would be a Major Difference. A Minor Difference, on the other hand, occurs when the data changes, but does not change the codifying category. For example, if music was 2, 10 and became 2, 12, this would be a Minor Difference. We, thus, have two variants of the Diff Visualizer, which will print out either Major or Minor Differences as needed. A sample output containing both Major and Minor Differences is shown in Listing 3.

```
=====Apr 24 2012 and Apr 25 2012=====
feed – Old: 2, 19, New: 1, 0
photos – Old: 2, 25, New: 1, 0
posts – Old: 2, 19, New: 1, 0
tagged – Old: 2, 5, New: 1, 0
videos – Old: 2, 1, New: 1, 0
locations – Old: 2, 1, New: 1, 0
```

(a) Turning On the Privacy Settings

```
=====Apr 25 2012 and Apr 27 2012=====
feed – Old: 1, 0, New: 2, 19
photos – Old: 1, 0, New: 2, 25
posts – Old: 1, 0, New: 2, 20
tagged – Old: 1, 0, New: 2, 4
videos – Old: 1, 0, New: 2, 1
locations – Old: 1, 0, New: 2, 1
```

(b) Turning Off the Privacy Settings

Figure 4: Changing Privacy Settings — Output as seen from our tool

Feasibility The first step in our empirical evaluation was to show the feasibility of our approach and tool. For this step, we used the tool to access the Facebook data of a research colleague of the first author. Facebook uses OAuth 2.0 [47], which is an open standard for authentication. We provided our tool with the first author's OAuth 2.0 access token so that the tool can access the same data that the first author can. This is the equivalent of the research colleague, using our social approach, asking the first author to monitor his information on Facebook. For this step of the evaluation, we did the following:

1. We accessed the Facebook data of the research colleague (name anonymized, for privacy reasons).
2. After the data was accessed, we asked the colleague to turn on privacy controls and make the data less visible. This would enable us to check if our tool could detect changes in privacy, where data is made less visible. The colleague did this by adding the first author to one of his pre-defined friend lists that had very limited access to his profile. We accessed the data using our tool again.
3. Finally, we asked the colleague to turn off the privacy controls and make the data more visible. This would enable us to check if our tool could detect changes in privacy, where data is made more visible. We accessed the data again using our tool.

The output from the Diff Visualizer is shown in Figure 4. As the figure shows, turning on the privacy settings reduces visibility — things like photos, locations, and feed were visible earlier and the Facebook API responses contained data; with the privacy settings on, the Facebook API returns an empty set. Turning off the privacy settings makes the data visible again, as seen in the right hand side of the figure.

Our Facebook prototype tool can thus detect changes in privacy settings. These changes can either be data being made more private or data being made less private. Thus, if someone suddenly starts sharing more or less data than before, our tool would detect this and this could indicate a privacy bug. Next, we show some examples of bugs our tool can help detect.

Facebook Bugs — Family and Friendlists We ran our Facebook prototype tool using the first author's access token and collected data for all his Facebook friends ($n = 516$). The data was collected roughly every day for each user for approximately eleven weeks (from May 1, 2012 to July 20, 2012). For each user, the number of data points (*i.e.,* the number of days on which we

```
======May 04 2012 and May 07 2012======
feed - Old: 2, 21, New: 2, 20
posts - Old: 2, 15, New: 2, 14
======May 11 2012 and May 14 2012======
feed - Old: 2, 20, New: 2, 19
tagged - Old: 2, 11, New: 2, 10
======May 18 2012 and May 21 2012======
posts - Old: 2, 14, New: 2, 15
tagged - Old: 2, 10, New: 2, 8
======May 22 2012 and May 23 2012======
albums - Old: 2, 3, New: 1, 0
feed - Old: 2, 19, New: 1, 0
likes - Old: 2, 2, New: 1, 0
photos - Old: 2, 25, New: 1, 0
posts - Old: 2, 15, New: 1, 0
tagged - Old: 2, 8, New: 1, 0
family - Old: 2, 1, New: 1, 0
groups - Old: 2, 2, New: 1, 0
locations - Old: 2, 4, New: 1, 0
```

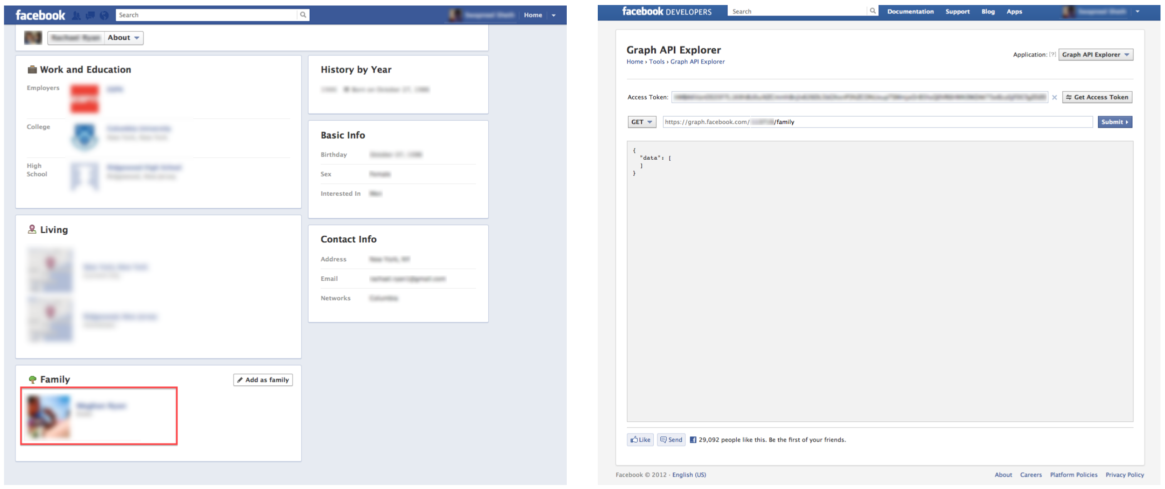
Listing 4: Facebook Family Bug — Output as seen from our tool

successfully got data from the Facebook servers) was, on average, 36.24 (σ = 3.26, median = 36, max = 44, min = 25). (Please see Section 5.4 for a discussion on the number of data points and on the robustness of our approach.)

Upon running the Diff Visualizer, we found that 63.18% (326 out of 519) of the users had Major and Minor Differences during the data monitoring period. Out of these, there were a total of 5065 Minor Differences (on average, 15.54 per user) and 780 Major Differences (on average, 2.39 per user). Not all of these differences necessarily imply privacy bugs; some of these differences would arise from the "normal" use of these systems, *i.e.,* users adding new photos from a recent trip and so on. But even in these cases, the users may not be aware with whom they are now sharing this new information.

We now highlight, in this and the next subsection, a couple of interesting case studies on the bugs that were discovered using our tool. For a certain user, there was one family member being shown for the first three weeks of the data monitoring period. On May 23, 2012, our tool found a lot of Major Differences (there had been other, unrelated, Minor Differences previously) for that user. The output from our tool for the entire monitoring period is shown in Listing 4. The last diff shows a lot less data being visible. Was this a case of a user turning on privacy settings? Or perhaps did something change in the Facebook Code or API resulting in a bug? We tried to find out the cause. We were, of course, limited in our efforts as we don't have access to the source code. We decided to focus on the family connection, which lists a user's family members. This is highlighted in red in Listing 4.

The first step of our investigation was to see the actual page on Facebook. On the page, the family member was still visible. This is shown in Figure 5a, highlighted in red (user details have been blurred out to protect privacy). To ensure that there were no bugs in our tool implementation, we used the Facebook Graph API Explorer and verified that this returned an empty data set. This is shown in Figure 5b.



```
{
  "data": [
  ]
}
```

(a) The Facebook Website — The highlighted red rectangle shows the family member.

(b) API — Using the API, the list of family members is empty.

Figure 5: Facebook Family Bug — On the website, you can see the family member; Using the API, you cannot see the family member.

Finally, we started looking at Facebook bug reports to see if anyone else had reported a similar issue. We found two relevant bug reports. The most relevant bug report was titled "Can no longer access FriendList members on test users" [37]. In this bug report, a user had created two test users and added each user to the other's family list. When the user tried accessing the members of one of the test user's family, the Graph API returned an empty data set. This is exactly the same behavior as what we observed here. Facebook have confirmed that the bug exists and assigned it to a developer with medium priority. The second bug report, titled "Some of the friendlists do not show members from Graph API, why??" [38], reported a similar problem to the previous bug report. In this bug report, the user had many friend lists, which is a generalization of the family connection. Upon accessing the data from the Graph API, some of the friend lists return all the members; some return only a subset of the members. Facebook has responded to this bug report by triaging it with low priority.

These bug reports and our tool output, taken together, lead us to believe that there is a bug in the Facebook API, that was recently introduced due to code changes and should not have passed the regression testing phase. This is an example of under-sharing — less information is public than it should be. Our tool was able to detect this privacy bug using end-user regression testing.

Facebook Bugs — Links made public At the start of July, our tool discovered another example of a privacy bug — this time, it was an example of over-sharing. The output from our tool for one user is shown in Listing 5. There was a lot of data made public about links posted by a user, which is highlighted in red. The interesting part was that this was not data that was recently added by the user; some of these links were added back in 2009. This was data that had been on Facebook for a while and not visible to friends of the users; now, it was visible.

Analyzing the data further, we found that 73 out of 516 users were now sharing a lot more links than before and that this new data was first visible towards the start of July. Of the 73 users, 57

```
======Jul 02 2012 and Jul 06 2012======
feed - Old: 2, 14, New: 2, 15
links - Old: 1, 0, New: 2, 23
posts - Old: 2, 3, New: 2, 5
tagged - Old: 2, 14, New: 2, 15
```

Listing 5: Facebook Links Bug — Output as seen from our tool

(*i.e.,* 11.05%) were sharing more than one link, thus indicating that this was not new information added by the user recently.

To understand the cause behind this over-sharing, we conducted an informal open-ended interview with one of the users, Keith (name changed for privacy reasons). Excerpts from the interview are shown next (reproduced with permission from Keith). On being told that he is now sharing 23 links, which weren't visible earlier, Keith responded: "whoa [. . .] ok...... that is weird." After looking at the links that were visible, Keith said: "ok, all of these links are valid, but, am surprised you can see them [. . .] I, as a developer, opened my account for developer access. it's the only way possible and I just thought I was authorizing that one app. they must have their permissions f***ed up [. . .] it's either that, or facebook changed my settings automatically. " Keith said that he would change his settings back so that the links would not visible anymore and ended the interview with: "good thing your app [sic] was able to catch it."

Keith is currently a student at Columbia University pursuing a Ph.D. in Computer Science. Given that someone with a technical background didn't know about his data being public and wasn't completely sure what changed, a *non-technical end-user* would generally have a much harder time figuring out changes in privacy settings. This is the main strength of our tool — giving end-users an easy way for detecting potential privacy breaches. Since this particular bug had not affected all the users, it seems to indicate that this a Policy Change that is being rolled out gradually by Facebook. Alternatively, this could be another example of a bug (affecting only some users) in the Facebook Code or API. Either way, our tool was able to detect this.

We also extended our prototype tool to detect privacy bugs in Twitter. Due to space reasons, this section is shown in Appendix D.

5.4 Discussion

5.4.1 Flexibility

One advantage of this approach for detecting privacy bugs is the flexibility. A user could choose different sets of friends to monitor different things, if the social system has a fine-grained privacy model. For example, he could have a friend check the privacy settings of his photos and check-in locations. He could have someone from his network (such as "New York") check his music and movies. He could have a friend of a friend check his feed. He could also create overlapping groups — his friends should be able to see albums and locations; his network can only see the albums. Thus, he could use different sets of friends to verify that the privacy settings indeed are what he expects them to be and to alert him when they can see more or less than what they saw before.

A user can also choose how often the data is fetched based on how active he is on the social system, the API rate limits, and personal preferences such as the tradeoff between the load on the social system's servers and his privacy needs.

Finally, in terms of implementation, our prototype tools were stand-alone tools that ran off the command line. Social systems like Facebook and Twitter provide rich ecosystems for apps. Our approach can be implemented as apps that run on Facebook, for example. Other alternatives include a browser plugin that automatically runs the regression testing when the user logs into one of these systems or on a periodic basis (every hour, every day, and so on), a "normal" desktop application with a GUI, and so on. We used Ruby for our implementations; this was of out choice and is not a constraint. Any programming language that can access the web can be used. Having a wrapper library for that programing language does help as it obviates the need to deal with lower level protocol details. For example, Twitter has a list of libraries for 14 programming languages ranging from Java, .NET, and Python to Erlang, Scala, and Clojure [104]. There are no inherent implementation or UI limitations as far as our approach is concerned.

5.4.2 Generalizability

Another advantage of this approach for detecting privacy bugs is the generalizability of the technique. In general, it can work with any social system regardless of what kinds of privacy controls and features it has. The previous Section showed how it could work with systems at two extreme ends of the privacy spectrum: Facebook, with its fine grained settings for choosing who sees what and Twitter with its coarse grained settings, which is essentially an on/off switch.

Having an API to use makes it much easier to implement a tool for a particular social system. This, however, is not a limiting factor — if there is no API, the same approach can be combined with alternatives such as screen or web scraping.

5.4.3 Robustness

Our approach is also very robust — it does not need that the Data Monitor is run every day or that the Data Monitor successfully fetches data for each user every day (or on every run). In spite of the Facebook API having slow response times [81] and timing out occasionally, which resulted in our Data Monitor fetching data successfully on average 36 times (out of possibly about 75 times) for each user in an eleven week time period, our tool still works fine and detects bugs as shown in the previous Section. The tradeoff with fetching data less often is that we will not be able to catch transient bugs. For example, if something is made public only for a few minutes and then it is private again, if our Data Monitor is not active then, we will not be able to detect it.

In contrast to the Facebook API, the Twitter API, in our experience, was much more stable. Regardless of the stability and reliability of the social system under test, as mentioned above, our approach can still detect bugs due to its highly flexible nature.

5.4.4 Limitations and Threats to Validity

A limitation of our prototype tools is that we keep track of the number of items in data fetched, rather than the actual data. For example, in the Facebook tool, for a user, we log that the user had ten photos rather than what the photos are. We do this for two reasons: (1) to reduce the data that needs to be stored; and (2) for privacy reasons. Due to this, our tools might miss out on changes in privacy if a user, for example, deletes one photo and adds one photo, our tool would see this as no change having occurred. This, however, is a limitation of our *prototype tools*, and not of our *approach*. If an end-user wishes to keep track of the exact data, rather than the number of data items, our tools can be modified to do that. An added challenge, in this case, would be to find the semantic similarities between data items, which would be easier in some cases (*e.g.,* checkin locations, groups) than others (*e.g.,* albums, interests, likes).

An inherent limitation of our approach is the possibility of false negatives, *i.e.,* privacy bugs that exist in the system that our tool/approach is not able to catch. There might be bugs that have existed since the first version of the software; if there is no change in the code, our regression testing approach will not work. There might be privacy bugs that affect only some of the users; if the users that are currently using our tool are not affected by this bug, we won't be able to detect it. The main reason for this is that our approach is intended for *end-users* and we don't have access to the source code of the system under test. From an *end-user* perspective, it's hard to detect bugs using our approach that may not have an external manifestation or change in behavior for our users.

Finally, coming back to our software engineering research questions, even though **R1** is beyond the scope of this project, we make a couple of observations based on our empirical results. If there are never any changes in the social software, then **R2** and **R3** won't happen, but in a limited indirect way just trying to use our tool (which will keep saying "no change") might make users more aware of privacy settings issues and thus in a very small way help with **R1**. Now if there *are* changes, so **R2** and/or **R3** come into play, then the awareness with respect to **R1** would be stronger because users would then be prompted to go look more closely at the particular settings that were affected and thus would understand them better and adjust their mental model accordingly.

Statistical Conclusion — Do we have sufficient data to make our claims? For our Facebook tool, we fetched data for 516 users resulting in almost 18,700 data points. For our Twitter tool, we fetched data for 10 users resulting in almost 350 data points. The goal of the empirical evaluations was to find examples of privacy bugs to show the feasibility and utility of our approach, which we did find as described in Section 5.3.

External Validity — Do our results generalize to other systems? Our prototype tools were implemented for two different systems: Facebook and Twitter. Our approach is broad and can apply to any social system as discussed in Sections 5.4.1 and 5.4.2 above.

5.5 Related Work

There have been some recent papers on data privacy and software testing. Clause and Orso [21] propose techniques for the automated anonymization of field data for software testing. They extend the work done by Castro *et al.* [20] using novel concepts of path condition relaxation and breakable input conditions resulting in improving the effectiveness of input anonymization. Taneja *et al.* [100] and Grechanik *et al.* [44] propose using k-anonymity [99] for privacy by selectively anonymizing certain attributes of a database for software testing. These papers propose novel approaches using static analysis for selecting which attributes to anonymize so that test coverage remains high. Peters and Menzies [82] propose an anonymization technique for sensitive data so that it can be used for cross-company defect prediction. They show that it is possible to make data less sensitive and still maintain high utility for data mining applications.

Our work is orthogonal to these papers on data anonymization. The problem they address is — how can one anonymize sensitive information before sharing it with others (*e.g.,* sending it to the teams or companies that build the software, sharing information for testing purposes, and sharing data across multiple companies, respectively)? The problem we address is - how can *end-users* verify if the software systems they are using are handling privacy correctly? Further, all these papers are trying to protect the privacy of the data. We, on the other hand, are trying to detect privacy violations and test if the systems have any privacy bugs.

There has been a lot of work in the field of regression testing mainly towards test case selection and test case prioritization [48, 51, 84, 118], including a very detailed, and excellent, recent survey by Yoo and Harman [115] and the references therein. Our work builds on, and differs from, all of the above in two aspects – our regression testing approach is targeted towards *end-users* and is targeted towards finding *privacy* bugs.

There has also been some recent work in using taint analysis for detecting security and privacy violations [31, 92]. These approaches require access to source code for taint analysis. Our approach, on the other hand, is targeted towards end-users who do not have source code access to the social systems that they are using. Our social testing approach is similar in some ways to "do you see what I see," a technique proposed in the networking community to support distributed fault detection and diagnosis from the client-side [96], although there the actual end-users are not directly involved, and is also related to the network security community's collaborative intrusion detection, e.g., [78], where the goal is to share data about penetration attempts against different organizations' enterprise networks but without inadvertently sharing any private information.

6 Research plan

Table 1 shows my plan for completion of the research.

Work Description		**Date**	**Progress**
Project	**Task**		
Privacy for Free (§2)	Finish initial prototype	Sep. 2011	complete
	Conduct/Update experiments	Sep. 2012	complete
	Submit Paper to WWW	Dec. 2012	complete
Privacy Requirements (§3)	Create User Study	Jun. 2012	complete
	Submit Study for IRB Approval	Jul. 2012	complete
	Conduct User Study	Sep. 2012	ongoing
	Submit Paper to ICSE 2014	Jul. 2013	
Crowdsourced Privacy (§4)	Gather Data for users	Apr. 2011	complete
	Create User Study	Dec. 2011	complete
	Submit Study for IRB Approval	Feb. 2013	
	Conduct User Study	Apr. 2013	
	Submit Paper to CHI/CSCW 2014	Jul. 2013	
Privacy Regression Testing (§5)	Finish initial prototype for Facebook	Mar. 2012	complete
	Extend prototypes of other systems	Apr. 2012	complete
	Collect data, find privacy bugs	Feb. 2013	ongoing
	Submit paper UIST 2013	Feb. 2013	

Table 1: Plan for completion of research

Appendices

A Privacy Requirements User Study

The online questionnaire for the Privacy Requirements user study is shown next. We already have IRB approval for the study (valid till 07/16/2014) and so far, 157 participants for the online questionnaire and 15 participants for the follow-up interviews.

Privacy Requirements Survey

1. Software Development Experience

1) Do you have any experience doing software development?
   a) Yes
   b) No

If No, go directly to **Section 2**.

2) How long have you been involved in software development?
   a) Less than 1 year
   b) 1-5 years
   c) 5-10 years
   d) More than 10 years

2. Concerns about privacy as an end user

Imagine you are a user of a system that might have access to sensitive data. These systems can range from social networks like Facebook, Google+ to any system that collects user data like Amazon, Netflix, etc.

1) How important is the privacy issue in Software Engineering?
   a) Very important
   b) Important
   c) Normal
   d) Less important
   e) Least important
2) Would you be willing to use the system if you are worried about privacy issues?
   a) Definitely – I don't care about privacy
   b) Probably yes
   c) Unsure
   d) Probably not
   e) Definitely not – if there are privacy concerns, I won't use this system
3) Would the following lead to privacy concerns?
   a) Aggregation of data – On collecting data over a period of time and aggregating it, new trends in your behavior might emerge, which might not be expected
      i) Yes
      ii) No
      iii) Unsure
   b) Distortion of Data – The data processing/recommendation system might distort the data or the your intent and may be highly misleading
      i) Yes
      ii) No
      iii) Unsure
   c) Data Sharing – The collected data might be used not just for this system, but given to third parties for purposes like advertising
      i) Yes
      ii) No
      iii) Unsure
   d) Data/Privacy breaches – Malicious users might be able to get access to sensitive information about you (and other users) in the system
      i) Yes
      ii) No
      iii) Unsure
   e) **Other additional concerns? Why?**

4) What will help in reducing user concerns about privacy?
   a) **Privacy Policy (or EULA, etc.)** – Describing what the system will/won't do with the data
      i) Yes
      ii) No
      iii) Unsure
   b) **Privacy Laws** – Describing which local/national law the system is complaint with (e.g., HIPAA in the US, European privacy laws)
      i) Yes
      ii) No
      iii) Unsure
   c) **Anonymizing all data** – Ensuring that none of the data has any identifiers to tie the data to an individual user
      i) Yes
      ii) No
      iii) Unsure
   d) **Technical Details –** Describing the algorithms/source code of the system in order to achieve higher trust (e.g., encryption of all data)
      i) Yes
      ii) No
      iii) Unsure
   e) **Details on usage** – Describe (preferably, in easy to understand words) what/how the data will be used
      i) Yes
      ii) No
      iii) Unsure
   f) **Other additional approaches? Why?**

5) Should the system have special mechanisms in place to protect your privacy based on the following:
   a) **Gender**
      i) Yes
      ii) No
      iii) Unsure
   b) **Users with varying levels of concern about privacy**
      i) Yes
      ii) No
      iii) Unsure
   c) **Users with varying levels of technical expertise**
      i) Yes
      ii) No
      iii) Unsure
   d) **Other additional categories? Why?**

6) What information about you should systems <u>never capture</u>? Why?

7) What information about you can be captured? Why?

8) What information about you should systems <u>never share</u>? Why?

9) What information about you can be shared? Why?

10) Do you have any specific additional privacy concerns for the following types of systems?
    a) Social Network Systems
    b) E-commerce Systems
    c) Recommendation Systems
    d) Location-based Services
    e) Others (please specify)

Please go to **<u>Section 4</u>**.

3. Concerns about privacy as a software developer

Imagine that you are a software developer building/modifying a complex system that has access to sensitive user information.

1) How important is the privacy issue in Software Engineering?
    a) Very important
    b) Important
    c) Normal
    d) Less important
    e) Least important
2) Do you think users will be willing to use the system if they are worried about privacy issues?
    a) Definitely – most users don't care about privacy
    b) Probably yes
    c) Unsure
    d) Probably not
    e) Definitely not – if there are privacy concerns, no one will use this system
3) Would the following lead to privacy concerns?
    a) Aggregation of data – On collecting data over a period of time and aggregating it, new trends in user behavior might emerge, which the user might not expect
        i) Yes
        ii) No
        iii) Unsure
    b) Distortion of Data – The data processing/recommendation system might distort the data or the user's intent and may be highly misleading
        i) Yes
        ii) No
        iii) Unsure
    c) Data Sharing – The collected data might be used not just for this system, but given to third parties for purposes like advertising
        i) Yes
        ii) No
        iii) Unsure
    d) Data/Privacy breaches – Malicious users might be able to get access to sensitive information about other users in the system
        i) Yes
        ii) No
        iii) Unsure
    e) **Other additional concerns? Why?**

4) What will help in reducing user concerns about privacy?
   - **a) Privacy Policy (or EULA, etc.)** – Describing to the user what the system will/won't do with the data
     - i) Yes
     - ii) No
     - iii) Unsure
   - **b) Privacy Laws** – Describing to the user which local/national law the system is complaint with (e.g., HIPAA in the US, European privacy laws)
     - i) Yes
     - ii) No
     - iii) Unsure
   - **c) Anonymizing all data** – None of the data has any identifiers to tie the data to the user
     - i) Yes
     - ii) No
     - iii) Unsure
   - **d) Technical Details** – Describing the algorithms/source code of the system in order to achieve higher trust (e.g., encryption of all data)
     - i) Yes
     - ii) No
     - iii) Unsure
   - **e) Details on usage** – Describe (preferably, in easy to understand words) what/how the data will be used
     - i) Yes
     - ii) No
     - iii) Unsure
   - f) **Other additional approaches? Why?**

5) Should the system have special mechanisms in place to protect the privacy of the users based on the following:
   - **a) Gender of the users**
     - i) Yes
     - ii) No
     - iii) Unsure
   - **b) Users with varying levels of concern about privacy**
     - i) Yes
     - ii) No
     - iii) Unsure
   - **c) Users with varying levels of technical expertise**
     - i) Yes
     - ii) No
     - iii) Unsure
   - d) **Other additional categories? Why?**

6) What information about the user should systems never capture? Why?

7) What information about the user can be captured? Why?

8) What information about the user should systems never share? Why?

9) What information about the user can be shared? Why?

10) Do you have any specific additional privacy concerns for the following types of systems?
    a) Social Network Systems
    b) E-commerce Systems
    c) Recommendation Systems
    d) Location-based Services
    e) Others (please specify)

4. User demographics

1) Are you or have you been involved with academia (as a researcher, faculty member, PhD student or postdoc)?
   a) Yes
   b) No
2) Are you or have you been involved with industry?
   a) Yes
   b) No
3) In which continent have you primarily lived in the last few years?
   a) North America
   b) South America
   c) Europe
   d) Asia
   e) Africa
   f) Multiple – Please specify
   g) Other – Please specify
4) In which continent were you born?
   a) North America
   b) South America
   c) Europe
   d) Asia
   e) Africa
   f) Other – Please specify
5) In which continent have you lived the longest?
   a) North America
   b) South America
   c) Europe
   d) Asia
   e) Africa
   f) Multiple – Please specify
   g) Other – Please specify
6) Would you be interested in participating in a follow-up telephone or in person interview?
   a) Yes
   b) No

If Yes, please provide your contact details below:

Name:
Email:

B Crowdsourcing Privacy Settings User Study

Crowdsourced Privacy Settings Survey

1. Facebook Privacy Settings Background

1) How often do you use Facebook?
   a) Daily
   b) Weekly
   c) Monthly
   d) Rarely

2) Are you aware of the fine-grained privacy settings?
   a) Yes
   b) No

3) How happy are you with the default settings?
   a) Very Happy
   b) Happy
   c) Neutral
   d) Unhappy
   e) Very Unhappy

4) Why?

5) Have you ever changed the default privacy settings?
   a) Yes
   b) No

6) How often do you change your privacy settings?
   a) Never
   b) Set once and forget
   c) Regularly

7) What is the main purpose of privacy settings for you?
   a) Block a few friends
   b) Block family
   c) Block everyone other than friends
   d) Other

8) How happy are you with the privacy settings provided by Facebook?
   a) Very Happy
   b) Happy
   c) Neutral
   d) Unhappy

    e) Very Unhappy

9) Why?

10) How easy is it to change your privacy settings?
    a) Very Easy
    b) Easy
    c) Neutral
    d) Hard
    e) Very Hard

11) How comfortable are you with:
    a) Other people sharing your information
        i) Very comfortable
        ii) Comfortable
        iii) Neutral
        iv) Uncomfortable
        v) Very uncomfortable
    b) Applications accessing your data
        i) Very comfortable
        ii) Comfortable
        iii) Neutral
        iv) Uncomfortable
        v) Very uncomfortable

12) Should you have some control over this?
    a) Other people
        i) Yes
        ii) Uncertain
        iii) No
    b) Applications
        i) Yes
        ii) Uncertain
        iii) No

2. Crowdsourced Privacy Settings

====Based on our analysis, your score is _____ =====
Friends Average Score – X/100
(Higher score => more sharing, less privacy)

1) What is your reaction to the score?
   a) Very surprised
   b) Surprised
   c) Neutral
   d) Sort of expected
   e) Exactly as expected

2) Would you like a mechanism where your privacy settings mimic the privacy settings of (a group of) friends?
   a) Yes
   b) Uncertain
   c) No

3) Would you like a mechanism where your privacy settings towards a person are identical to the privacy settings of that person to you (E.g., if person A has hidden his photos from you, your photos will also be hidden from A)?
   a) Yes
   b) Uncertain
   c) No

4) Would you like privacy settings in the following manner?
   a) List of all privacy settings (current Facebook default)
      i) Strongly Agree
      ii) Agree
      iii) Neutral
      iv) Disagree
      v) Strongly Disagree
   b) Point Score of your settings (similar to the start of this section)
      i) Strongly Agree
      ii) Agree
      iii) Neutral
      iv) Disagree
      v) Strongly Disagree
   c) Coarse Grained Settings – Very Private, Neutral, Very Public (similar to video games with modes like Easy/Medium/Hard)
      i) Strongly Agree
      ii) Agree
      iii) Neutral
      iv) Disagree

C User Connections on Facebook

Table 2 shows the user connections that are available via the Facebook API.

Name	**Description**
accounts	The Facebook apps and pages owned by the current user.
achievements	The achievements for the user.
activities	The activities listed on the user’s profile.
albums	The photo albums this user has created.
apprequests	The user’s outstanding requests from an app.
books	The books listed on the user’s profile.
checkins	The places that the user has checked-into.
events	The events this user is attending.
family	The user’s family relationships
feed	The user’s wall.
friendlists	The user’s friend lists.
friendrequests	The user’s incoming friend requests.
friends	The user’s friends.
games	Games the user has added to the Arts and Entertainment section of their profile.
groups	The Groups that the user belongs to.
home	The user’s news feed.
inbox	The Threads in this user’s inbox.
interests	The interests listed on the user’s profile.
likes	All the pages this user has liked.
links	The user’s posted links.
locations	Posts, statuses, and photos in which the user has been tagged at a location.
movies	The movies listed on the user’s profile.
music	The music listed on the user’s profile.
mutualfriends	The mutual friends between two users.
notes	The user’s notes.
notifications	The notifications for the user.
outbox	The messages in this user’s outbox.
payments	The Facebook Credits orders the user placed with an application.
permissions	The permissions that user has granted the application.
photos	Photos the user (or friend) is tagged in.
picture	The user’s profile picture.
pokes	The user’s pokes.
posts	The user’s own posts.
questions	The user’s questions.
scores	The current scores for the user in games.
statuses	The user’s status updates.
subscribedto	People you’re subscribed to.
subscribers	The user’s subscribers.
tagged	Posts the user is tagged in.
television	The television listed on the user’s profile.
updates	The updates in this user’s inbox.
videos	The videos this user has been tagged in.

Table 2: User Connections as listed on the Facebook User API [36]

D Privacy Testing and Twitter

Twitter, as opposed to Facebook, has a completely different model with respect to privacy. While Facebook has a very fine-grained control model for controlling what's visible to whom, Twitter has a very coarse-grained model. Users can choose if their accounts are "protected" or not, with the account being not protected as the default setting [103]. If the account is not protected, all tweets are public and can be viewed by anyone. If the account is protected, it can only be viewed by the followers of that person. There is no mechanism for deciding this on a per-tweet basis, for example. Regardless of whether the account is protected or not, anyone can still see the number of tweets, the number of followers, and the number of following for any user.

Thus, the only setting that matters in terms of privacy is whether the account if protected or not. Our Twitter tool, described below, uses the same social approach for detecting if the privacy settings change. In addition to this, to show the generalizability of our approach in dealing with different kinds of social systems, we decided to treat some other user information as "sensitive" — *i.e.,* if Twitter had more fine-grained controls for these types of information, our approach (and tool) would still be able to detect changes in privacy settings. A partial list of user fields from the Twitter User API is shown in Table 3. For the purposes of our tool and empirical study, we treated these as sensitive information as well and inform the user when these change.

Field	**Description**
description	The user-defined UTF-8 string describing their account.
favourites_count	The number of tweets this user has favorited in the account's lifetime.
followers_count	The number of followers this account currently has.
friends_count	The number of users this account is following (AKA their "followings").
geo_enabled	When true, indicates that the user has enabled the possibility of geotagging their Tweets.
listed_count	The number of public lists that this user is a member of.
protected	When true, indicates that this user has chosen to protect their Tweets.
statuses_count	The number of tweets (including retweets) issued by the user.
verified	When true, indicates that the user has a verified account.
withheld_in_countries	When present, indicates a textual representation of the two-letter country codes this user is withheld from.
withheld_scope	When present, indicates whether the content being withheld is the "status" or a "user."

Table 3: Partial list of Users Fields from the Twitter User API [105]

D.1 Prototype Tool

Our prototype tool can detect privacy bugs for Twitter. Similar to the Facebook prototype tool described in Section 5.3.1, it is easily configurable and can fetch any data provided by the Twitter API. For the purposes of this study, we focused on getting only the partial list of User fields shown

```
========2012−06−19 and 2012−06−19========
friends_count−Old: 2, 140, New: 2, 142
========2012−06−19 and 2012−06−19========
protected-Old: 2, false, New: 2, true
========2012−06−19 and 2012−06−19========
protected-Old: 2, true, New: 2, false
========2012−06−21 and 2012−06−22========
followers_count−Old: 2, 138, New: 2, 137
statuses_count−Old: 2, 548, New: 2, 551
```

Listing 6: Privacy Monitoring of @swapneel — Output as seen from our tool

in Table 3. Similar to the Facebook tool, the Twitter tool consists of two components: the Data Monitor and the Diff Visualizer.

The Data Monitor is implemented as a set of Ruby scripts. It uses the Twitter library [73] to get data from the Twitter API. The Diff Visualizer, similar the Facebook Diff Visualizer, is also a set of Ruby scripts that parses each log file and creates a human readable output if there is a diff between any consecutive runs for a user. The rest of the workings of the Data Monitor are similar to the Facebook tool described in Section 5.3.1. We do not repeat the implementation details of the tools due to space limitations. The main difference is that this tool focuses on the user fields shown in Table 3; the rest of the implementation is similar. The other difference is that, for Twitter due to its lack of fine-grained privacy controls, we do not distinguish between Major and Minor Differences.

```
========2012−06−19 and 2012−06−20========
statuses_count−Old: 2, 904, New: 2, 906
========2012−06−20 and 2012−06−21========
listed_count−Old: 2, 67, New: 2, 68
statuses_count−Old: 2, 906, New: 2, 910
========2012−06−21 and 2012−06−22========
followers_count−Old: 2, 877, New: 2, 876
statuses_count−Old: 2, 910, New: 2, 912
========2012−06−22 and 2012−06−23========
followers_count−Old: 2, 876, New: 2, 878
statuses_count−Old: 2, 912, New: 2, 913
========2012−06−23 and 2012−06−25========
followers_count−Old: 2, 878, New: 2, 879
========2012−06−25 and 2012−06−26========
followers_count−Old: 2, 879, New: 2, 880
========2012−06−26 and 2012−06−27========
followers_count−Old: 2, 880, New: 2, 882
========2012−06−27 and 2012−06−28========
followers_count−Old: 2, 882, New: 2, 883
========2012−06−28 and 2012−06−29========
followers_count−Old: 2, 883, New: 2, 885
friends_count−Old: 2, 1015, New: 2, 1016
```

Listing 7: Privacy Monitoring of @ICSEConf — Output as seen from our tool

D.2 Feasibility

We ran our Twitter prototype tool and collected data for some of the first author's research colleagues ($n = 10$). The data was collected roughly every day for each user for approximately four weeks (from May 19, 2012 to July 20, 2012).

We also collected data for the first author (@swapneel) and changed the account to protected (and back to open) and verified if the tool can detect changes in privacy settings. The tool could, indeed, pick up the changes in privacy settings. The output from the Diff Visualizer (highlighted in red) in shown in Listing 6.

One of the twitter accounts for which the data was collected was the official ICSE twitter account (@ICSEconf). If we assume that the fields listed in Table 3 are sensitive information, our tool can detect changes in these as well. The partial output from the Diff Visualizer is shown in Listing 7.

Recently, a bug was found in Twitter where users who wanted to follow others were "arbitrarily, randomly, and haphazardly" unfollowed [77]. This "unfollow" bug was acknowledged by the Twitter team and they said that they were working on a fix. Our tool would have been able to detect this bug as well as follows: Say I started following two new users today. If the output from the Diff Visualizer was anything other than two, we know that there is a bug with following someone. The user could then check which user got unfollowed and follow the user again, if needed.

References

[1] ACM. ACM Proceedings. http://dl.acm.org/proceedings.cfm, 2011.

[2] Nabil R. Adam and John C. Worthmann. Security-control methods for statistical databases: a comparative study. *ACM Comput. Surv.*, 21(4):515–556, 1989.

[3] Dakshi Agrawal and Charu C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 247–255, New York, NY, USA, 2001. ACM.

[4] Alex Koppel. Koala. https://github.com/arsduo/koala/, April 2010.

[5] Apple Developer. iOS Dev Center. http://developer.apple.com/devcenter/ios/index.action, 2007.

[6] Lars Backstrom, Cynthia Dwork, and Jon Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 181–190, New York, NY, USA, 2007. ACM.

[7] M. Barbaro, T. Zeller, and S. Hansell. A face is exposed for AOL searcher no. 4417749. *New York Times*, 9, 2006.

[8] BBC. German Street View goes live with enhanced privacy. http://www.bbc.co.uk/news/technology-11673117, November 2010.

[9] BBC. Governments 'not ready' for new European privacy law. http://www.bbc.co.uk/news/technology-12677534, March 2011.

[10] Leland L. Beck. A security mechanism for statistical database. *ACM Trans. Database Syst.*, 5(3):316–3338, 1980.

[11] Andrew Begel, Khoo Yit Phang, and Thomas Zimmermann. Codebook: discovering and exploiting relationships in software repositories. In *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 125–134, New York, NY, USA, 2010. ACM.

[12] S. Berkovsky, Y. Eytani, T. Kuflik, and F. Ricci. Enhancing privacy and preserving accuracy of a distributed collaborative filtering. In *RecSys '07: Proc. of the 2007 ACM conf. on Recommender systems*, pages 9–16, 2007.

[13] Smriti Bhagat, Graham Cormode, Balachander Krishnamurthy, and Divesh Srivastava. Privacy in dynamic social networks. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 1059–1060, New York, NY, USA, 2010. ACM.

[14] Bloomberg News. Baidu Sued in New York Court for Censoring China Internet Search Results. http://www.bloomberg.com/news/2011-05-18/baidu-com-accused-in-u-s-lawsuit-of-aiding-chinese-internet-censorship.html, May 2011.

[15] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: the sulq framework. In *PODS '05: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 128–138, New York, NY, USA, 2005. ACM.

[16] Avrim Blum, Katrina Ligett, and Aaron Roth. A learning theory approach to non-interactive database privacy. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 609–618, New York, NY, USA, 2008. ACM.

[17] Bianca Bosker. Facebook CEO 'Doesn't Believe In Privacy'. http://www.huffingtonpost.com/2010/04/29/zuckerberg-privacy-stance_n_556679.html, April 2010.

[18] Brad Fitzpatrick. LiveJournal. http://www.livejournal.com/, 1999.

[19] Gerardo Canfora, Luigi Cerulo, Marta Cimitile, and Massimiliano Di Penta. Social interactions around cross-system bug fixings: the case of freebsd and openbsd. In *Proceeding of the 8th working conference on Mining software repositories*, MSR '11, pages 143–152, New York, NY, USA, 2011. ACM.

[20] Miguel Castro, Manuel Costa, and Jean-Philippe Martin. Better bug reporting with better privacy. In *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, ASPLOS XIII, pages 319–328, New York, NY, USA, 2008. ACM.

[21] James Clause and Alessandro Orso. Camouflage: automated anonymization of field data. In *Proceeding of the 33rd international conference on Software engineering*, ICSE '11, pages 21–30, New York, NY, USA, 2011. ACM.

[22] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *Proc. of the 22nd ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS)*, pages 223–233, 2003.

[23] Noam Cohen. It's Tracking Your Every Move and You May Not Even Know. `http://www.nytimes.com/2011/03/26/business/media/26privacy.html`, March 2011.

[24] T. Dalenius. Towards a methodology for statistical disclosure control. *Statistik Tidskrift*, 15:429–444, 1977.

[25] Yi Ding and Xue Li. Time weight collaborative filtering. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 485–492, New York, NY, USA, 2005. ACM.

[26] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 202–210, New York, NY, USA, 2003. ACM.

[27] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. *Theory of Cryptography*, pages 265–284, 2006.

[28] C. Dwork and K. Nissim. Privacy-preserving datamining on vertically partitioned databases. *Lecture Notes in Computer Science*, pages 528–544, 2004.

[29] Cynthia Dwork. Differential privacy. *IN ICALP*, 2:1–12, 2006.

[30] Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil Vadhan. On the complexity of differentially private data release: efficient algorithms and hardness results. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 381–390, New York, NY, USA, 2009. ACM.

[31] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proc. of the 9th USENIX Conf. on Operating systems design and impl.*, pages 1–6, 2010.

[32] U.S. Energy Information Administration. International Energy Outlook 2010 - Highlights. `http://www.eia.doe.gov/oiaf/ieo/highlights.html`, May 2010.

[33] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 211–222, New York, NY, USA, 2003. ACM.

[34] Facebook. Facebook Developers. `http://developers.facebook.com/`, 2007.

[35] Facebook. Graph API. `https://developers.facebook.com/docs/reference/api/`, 2007.

[36] Facebook. User. `https://developers.facebook.com/docs/reference/api/user/`, 2007.

[37] Facebook Developers. Bugs – Can no longer access FriendList members on test users. `https://developers.facebook.com/bugs/368623589859564`, June 2012.

[38] Facebook Developers. Bugs – Some of the friendlists do not show members from Graph API, why?? `https://developers.facebook.com/bugs/400876833291706`, April 2012.

[39] Lujun Fang and Kristen LeFevre. Privacy wizards for social networking sites. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 351–360, New York, NY, USA, 2010. ACM.

[40] Dan Fletcher. How Facebook Is Redefining Privacy. `http://www.time.com/time/business/article/0,8599,1990582.html`, May 2010.

[41] Daniel M. German, Jens H. Webber, and Massimiliano Di Penta. Lawful software engineering. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 129–132, New York, NY, USA, 2010. ACM.

[42] W. Geyer, C. Dugan, D. R. Millen, M. Muller, and J. Freyne. Recommending topics for self-descriptions in online user profiles. In *RecSys '08: Proc. of the 2008 ACM conference on Recommender systems*, pages 59–66, 2008.

[43] Google Developers. Google+ API – Google+ Platform. `https://developers.google.com/+/api/`, 2011.

[44] Mark Grechanik, Christoph Csallner, Chen Fu, and Qing Xie. Is data privacy always good for software testing? *Software Reliability Engineering, International Symposium on*, 0:368–377, 2010.

[45] Ryan Grim. Facebook Blocks Ads For Pot Legalization Campaign. `http://www.huffingtonpost.com/2010/08/24/facebook-blocks-ads-for-p_n_692295.html`, August 2010.

[46] I. Hadar, S. Sherman, and O. Hazzan. Learning Human Aspects of Collaborative Software Development. *Journal of Information Systems Education*, 19(3):311–319, 2008.

[47] E. Hammer-Lahav, D. Recordon, and D. Hardt. The OAuth 2.0 authorization protocol. `http://tools.ietf.org/html/draft-ietf-oauth-v2/`, 2010.

[48] Mary Jean Harrold, James A. Jones, Tongyu Li, Donglin Liang, Alessandro Orso, Maikel Pennings, Saurabh Sinha, S. Alexander Spoon, and Ashish Gujarathi. Regression test selection for java software. In *Proc. of the 16th ACM SIGPLAN Conf. on OO prog., systems, languages, and appl.*, pages 312–326, 2001.

[49] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22:5–53, January 2004.

[50] Francis Heylighen and Johan Bollen. Hebbian algorithms for a digital library recommendation system. *Parallel Processing Workshops, International Conference on*, 0:439, 2002.

[51] Vilas Jagannath, Qingzhou Luo, and Darko Marinov. Change-aware preemption prioritization. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ISSTA '11, pages 133–143, 2011.

[52] M. Johnson, S. Egelman, and S.M. Bellovin. Facebook and privacy: It's complicated. In *Symp. on Usable Privacy and Security*, 2012.

[53] Steven Johnson. Web Privacy: In Praise of Oversharing. `http://www.time.com/time/business/article/0,8599,1990586.html`, May 2010.

[54] Patrick Gage Kelley, Lucian Cesca, Joanna Bresee, and Lorrie Faith Cranor. Standardizing privacy notices: an online study of the nutrition label approach. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 1573–1582, New York, NY, USA, 2010. ACM.

[55] Ralf Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intell. Data Anal.*, 8(3):281–300, 2004.

[56] Andrew J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. The state of the art in end-user software engineering. *ACM Comput. Surv.*, 43(3):21:1–21:44, April 2011.

[57] Yehuda Koren. Collaborative filtering with temporal dynamics. *Commun. ACM*, 53(4):89–97, 2010.

[58] I. Koychev and I. Schwab. Adaptation to drifting user's interests. In *Proceedings of ECML2000 Workshop: Machine Learning in New Information Age*, pages 39–46. Citeseer, 2000.

[59] A.T. Kronman. *Education's End: Why Our Colleges and Universities Have Given Up on the Meaning of Life*. Yale University Press, 2007.

[60] Last.fm. API. `http://www.last.fm/api`, 2009.

[61] Neal Lathia, Stephen Hailes, and Licia Capra. Private distributed collaborative filtering using estimated concordance measures. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pages 1–8, New York, NY, USA, 2007. ACM.

[62] Soo Ling Lim, Daniele Quercia, and Anthony Finkelstein. Stakenet: using social networks to analyse the stakeholders of large-scale software projects. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 295–304, New York, NY, USA, 2010. ACM.

[63] Yabing Liu, Krishna P. Gummadi, Balachander Krishnamurthy, and Alan Mislove. Analyzing facebook privacy settings: user expectations vs. reality. In *Proc. of the 2011 SIGCOMM Conf. on Internet measurement conf.*, pages 61–70, 2011.

[64] Michelle Madejski, Maritza Johnson, and Steven M. Bellovin. A study of privacy settings errors in an online social network. *Pervasive Computing and Comm. Workshops, IEEE Intl. Conf. on*, 0:340–345, 2012.

[65] Frank McSherry and Ilya Mironov. Differentially private recommender systems: building privacy into the net. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 627–636, New York, NY, USA, 2009. ACM.

[66] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *FOCS '07: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 94–103, Washington, DC, USA, 2007. IEEE Computer Society.

[67] Joseph Menn. White House calls for online privacy law. `http://www.ft.com/cms/s/0/7267c2c4-500d-11e0-9ad1-00144feab49a.html`, March 2011.

[68] Microsoft. *Green Computing*, volume 18. The Architecture Journal, 2008.

[69] Claire Cain Miller and Tanzina Vega. Google Introduces New Social Tool and Settles Privacy Charge. `http://www.nytimes.com/2011/03/31/technology/31ftc.html`, March 2011.

[70] C. Murphy, S. Sheth, G. Kaiser, and L. Wilcox. genSpace: Exploring Social Networking Metaphors for Knowledge Sharing and Scientific Collaborative Work. In *1st Intl. Workshop on Social Software Engg. and Applications*, pages 29–36, September 2008.

[71] S. Murugesan. Harnessing green it: Principles and practices. *IT Professional*, 10(1):24 –33, jan.-feb. 2008.

[72] Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the netflix prize dataset. *CoRR*, abs/cs/0610105, 2006.

[73] John Nunemaker, Wynn Netherland, Erik Michaels-Ober, and Steve Richert. The Twitter Ruby Gem. `http://twitter.rubyforge.org/`, 2006.

[74] Kevin O'Brien. Technology Butts Up Against Germany's Privacy Laws. `http://www.nytimes.com/2010/07/12/technology/12disconnect.html`, July 2010.

[75] Andrew Odlyzko. A modest proposal for preventing internet congestion. Technical report, AT&T Labs - Research, 1997.

[76] L. Osterweil. Perpetually testing software. In *Proc. of the The Ninth International Software Quality Week*, May 1996.

[77] Jeremiah Owyang. Coping With Twitter's Unfollow Bug. `http://techcrunch.com/2012/03/27/unfollowbug/`, March 2012.

[78] Janak J. Parekh, Ke Wang, and Salvatore J. Stolfo. Privacy-preserving payload-based correlation for accurate malicious traffic detection. In *Proc. of the SIGCOMM Workshop on Large-scale attack defense*, pages 99–106, 2006.

[79] Vilfredo Pareto. The new theories of economics. *The Journal of Political Economy*, 5(4):pp. 485–502, 1897.

[80] Thomas Paul, Martin Stopczynski, Daniel Puscher, Melanie Volkamer, and Thorsten Strufe. C4ps: colors for privacy settings. In *Proceedings of the 21st international conference companion on World Wide Web*, WWW '12 Companion, pages 585–586, New York, NY, USA, 2012. ACM.

[81] Sarah Perez. Facebook Wins “Worst API” in Developer Survey. `http://techcrunch.com/2011/08/11/facebook-wins-worst-api-in-developer-survey/`, August 2011.

[82] F. Peters and T. Menzies. Privacy and utility for defect prediction: Experiments with morph. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 189–199. IEEE, 2012.

[83] H. Polat and Wenliang Du. Privacy-preserving collaborative filtering using randomized perturbation techniques. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 625–628, Nov. 2003.

[84] Xiao Qu, Myra B. Cohen, and Gregg Rothermel. Configuration-aware regression testing: an empirical study of sampling and prioritization. In *Proc. of the 2008 Intl. Symp. on Software testing and analysis*, pages 75–86, 2008.

[85] Jason Reed, Adam J. Aviv, Daniel Wagner, Andreas Haeberlen, Benjamin C. Pierce, and Jonathan M. Smith. Differential privacy for collaborative security. In *Proceedings of the Third European Workshop on System Security*, EUROSEC ’10, pages 1–7, New York, NY, USA, 2010. ACM.

[86] LJ Rich. A guide to protecting your privacy on Facebook. `http://news.bbc.co.uk/2/hi/programmes/click_online/8717750.stm`, June 2010.

[87] Riva Richmond. Gadgetwise: A Guide to Facebook’s New Privacy Settings. `http://gadgetwise.blogs.nytimes.com/2010/05/27/5-steps-to-reset-your-facebook-privacy-settings/`, May 2010.

[88] Aaron Roth and Tim Roughgarden. Interactive privacy via the median mechanism. In *STOC ’10: Proceedings of the 42nd ACM symposium on Theory of computing*, pages 765–774, New York, NY, USA, 2010. ACM.

[89] Indrajit Roy, Srinath T. V. Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: security and privacy for MapReduce. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI’10, pages 20–20, Berkeley, CA, USA, 2010. USENIX Association.

[90] Anita Sarma, Larry Maccherone, Patrick Wagstrom, and James Herbsleb. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE ’09, pages 23–33, Washington, DC, USA, 2009. IEEE Computer Society.

[91] Zachary M. Saul, Vladimir Filkov, Premkumar Devanbu, and Christian Bird. Recommending random walks. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC-FSE ’07, pages 15–24, New York, NY, USA, 2007. ACM.

[92] E.J. Schwartz, T. Avgerinos, and D. Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 317–331. IEEE, 2010.

[93] Israel Shamir. The Guardianís Political Censorship of Wikileaks. `http://www.counterpunch.org/2011/01/11/the-guardian-s-political-censorship-of-wikileaks/`, January 2011.

[94] Maggie Shiels. Germany officials launch legal action against Facebook. `http://news.bbc.co.uk/2/hi/technology/8798906.stm`, July 2010.

[95] Reza Shokri, Pedram Pedarsani, George Theodorakopoulos, and Jean-Pierre Hubaux. Preserving privacy in collaborative filtering through distributed aggregation of offline profiles. In *RecSys ’09: Proceedings of the third ACM conference on Recommender systems*, pages 157–164, New York, NY, USA, 2009. ACM.

[96] V.K. Singh, H. Schulzrinne, and K. Miao. Dyswis: An architecture for automated diagnosis of networks. In *Network Operations and Management Symposium*, pages 851–854. IEEE, 2008.

[97] S. Spiekermann and L.F. Cranor. Engineering privacy. *Software Engineering, IEEE Transactions on*, 35(1):67 –82, jan.-feb. 2009.

[98] Anna Cinzia Squicciarini, Mohamed Shehab, and Federica Paci. Collective privacy management in social networks. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 521–530, New York, NY, USA, 2009. ACM.

[99] Latanya Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.

[100] Kunal Taneja, Mark Grechanik, Rayid Ghani, and Tao Xie. Testing software in age of data privacy: a balancing act. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, SIGSOFT/FSE '11, pages 201–211, New York, NY, USA, 2011. ACM.

[101] Vincent Toubiana, Vincent Verdot, Benoit Christophe, and Mathieu Boussard. Photo-tape: user privacy preferences in photo tagging. In *Proceedings of the 21st international conference companion on World Wide Web*, WWW '12 Companion, pages 617–618, New York, NY, USA, 2012. ACM.

[102] Twitter. Twitter Developers. `https://dev.twitter.com/`, 2008.

[103] Twitter. About Public and Protected Tweets. `http://support.twitter.com/entries/14016`, 2012.

[104] Twitter Developers. Twitter Libraries. `https://dev.twitter.com/docs/twitter-libraries/`, 2012.

[105] Twitter Developers. Users. `https://dev.twitter.com/docs/platform-objects/users`, 2012.

[106] UNICEF. Optional Protocol on the sale of children, child prostitution and child pornography. `http://www.unicef.org/crc/index_30204.html`, June 2011.

[107] Jessica Vascellaro and Loretta Chao. Google Defies China on Web. `http://online.wsj.com/article/SB10001424052748704117304575137960803993890.html`, March 2010.

[108] Gina Venolia, John Tang, Ruy Cervantes, Sara Bly, George Robertson, Bongshin Lee, and Kori Inkpen. Embodied social proxy: mediating interpersonal connection in hub-and-satellite teams. In *CHI '10: Proceedings of the 28th international conference on Human factors in computing systems*, pages 1049–1058, New York, NY, USA, 2010. ACM.

[109] Vassilios S. Verykios, Elisa Bertino, Igor Nai Fovino, Loredana Parasiliti Provenza, Yucel Saygin, and Yannis Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Rec.*, 33(1):50–57, 2004.

[110] John Vidal. The end of oil is closer than you think. `http://www.guardian.co.uk/science/2005/apr/21/oilandpetrol.news`, April 2005.

[111] Jim Whitehead. Collaboration in software engineering: A roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 214–225, Washington, DC, USA, 2007. IEEE Computer Society.

[112] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[113] Alex Wissner-Gross. How you can help reduce the footprint of the Web. `http://www.timesonline.co.uk/tol/news/environment/article5488934.ece`, January 2009.

[114] Edward Wyatt. Court Rejects Suit on Net Neutrality Rules. `http://www.nytimes.com/2011/04/05/technology/05net.html`, April 2011.

[115] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *Softw. Test. Verif. Reliab.*, 22(2):67–120, March 2012.

[116] V. Zanardi and L. Capra. Social ranking: uncovering relevant content using tag-based recommender systems. In *RecSys '08: Proc. of the 2008 ACM Conf. on Recommender systems*, pages 51–58, 2008.

[117] J. Zhang and P. Pu. A recursive prediction algorithm for collaborative filtering recommender systems. In *RecSys '07: Proc. of the 2007 ACM conference on Recommender systems*, pages 57–64, 2007.

[118] Lu Zhang, Shan-Shan Hou, Chao Guo, Tao Xie, and Hong Mei. Time-aware test-case prioritization using integer linear programming. In *Proc. of the 2009 Intl. Symp. on Software testing and analysis*, pages 213–224, 2009.

[119] Elena Zheleva and Lise Getoor. To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 531–540, New York, NY, USA, 2009. ACM.

[120] Yun Zhu, Li Xiong, and Christopher Verdery. Anonymizing user profiles for personalized web search. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 1225–1226, New York, NY, USA, 2010. ACM.

[121] Mark Zuckerberg. Making Control Simple. `http://blog.facebook.com/blog.php?post=391922327130`, May 2010.