# Proposal:
# The Impact of Disruptive Events on Software Systems

## 1 Introduction

Software projects regularly suffer trauma. Consider the following examples. 1) As a release approaches, developers take shortcuts that introduce technical debt. If this technical debt is not repaired, the longterm quality of the system will suffer. 2) A lead developer who owns an important part of the codebase leaves. If steps to train other developers are not taken, it will become a dead area of the system and will be difficult to modify and maintain. 3) A security vulnerability is exploited in deployed software and requires the company to produce an immediate fix. During the security "panic," developers will rush to produce a fix, and technical debt may accrue with longterm negative impacts on the system. 4) Management often reorganizes the developers on a company's projects, with the result that developers move to codebases for which they have less experience. The reorganization introduces new perspectives and expertise that can lead to innovation; however, it can also result in a drop in productivity and the unnecessary re-writing of large portions of the system that the new developers do not understand. 5) Not all disruptive events have an obvious traumatic effect on a project. For example, a management directive might dictate that all code must be reviewed before it is added to the system. Review might improve the number of defects found. However, review is a mentally intensive activity that, when applied indiscriminately, can reduce the time developers spend working on critical areas of the system.

The overarching goals of this research are to 1) iteratively identify and taxonomize disruptive events, 2) study their longterm impact on software systems, 3) respond to and mitigate the risks associated with disruptive events.

There will be a multitude of factors affecting each event, so one significant challenge will be to identify the factors that lead to success across multiple software projects that have experienced similar disruptive events. We will mine data from development archives and interview project participants to create statistical models that predict software project outcomes and provide a fuller understanding of the behaviours and interactions that alleviate or aggravate the impact of events.

To identify disruptive events, we will look for anomalies in archival data as well as asking developers to recall the last disruptive event they experienced. Unlike most empirical software engineering studies, we are not interested in a single artifact or in an entire system, but the disruptive events that occur on multiple systems. To measure and describe the impact of a disruptive event, we will adapt our study methodology based on the type of event.

We have complementary research partners that will help us in achieving our ambitious objectives. Researchers at Concordia will mine data from Open Source Software (OSS) and Microsoft[1] projects as well as interview developers to identify common disruptive events. The Department of National Defense (DND) will provide a dedicated team at the Land Software Engineering Centre (LSEC) on which to apply successful findings in new software development environments. KDM Analytics will provide personnel, training, and tools to identify disruptive events related to system security. We will also work with KDM Analytics to develop prototypes and integrate the models of disruptive events into the KDM Workbench and other tools that will aid in targeting development effort and predicting problematic areas of the system.

### 1.1 Summary of Milestones

The goal of this work is to identify, understand the impact of, and provide successful responses to disruptive events on software systems. We have five overlapping milestones (Please see the Activity Schedule for

[1]We will work with Dr. Bird at Microsoft Research

start and completion dates). First, the identification and classification of disruptive events. Second, the measurement of and controlling for the factors that are associated with common disruptive events. Third, the creation of statistical models to understand the impact of event on common outcome measures (*e.g.,* defects found post-release). Fourth, the identification of successful practices that mitigate disruptive event as well as models to optimally focus development effort. Fifth, the development of a practitioner's guide to disruptive events and tool support to help developers identify and avoid the negative impacts of events.

**M1: Identify and Classify Disruptive Events**

What causes a disruptive event and which events often disrupt software projects?

As a starting point, we believe that disruptive events can be divided into those that occur "naturally" and those that are the result of a managerial directive. For example, a natural variation occurs when shortcuts are taken near a release, which result in technical debt and code that has not been sufficiently tested or reviewed. In contrast, a managerial directive might dictate that all code be reviewed before it is committed to the system. The challenge of this objective will be to devise measures and interview questions that identify common disruptive events across projects. The methods we will employ are discussed in Section 3. We will use statistical analyses to identify anomalous fluctuations across time periods (*e.g.,* reduced testing, release points) and interview developers to identify when they last experienced a disruptive event. The outcome of this objective will be a classification, ranked by importance, of disruptive events that regularly occur on software projects. In this proposal we describe possible disruptive events; however, we purposefully leave the specific events and definition open to allow us to identify the most impactful events on professional software projects.

**M2: Factors that Contribute to Disruptive Events**

Across multiple projects, which factors are common to a disruptive event and which are confounding?

Unlike a lab experiment where the experimenter controls for certain factors, we are studying existing, large software projects. Disruptive events on these projects will be affected by a multitude of confounding factors. In order to compare the impact of disruptive events, we will need to control confounds in our statistical models, such as the complexity of the system, developer expertise, and the system size. These confounds are the most significant risk to the proposed work. We will mitigate this risk by using a mixed method approach – we will measure factors and qualitatively code interactions between developers.

**M3: Impact of Disruptive Events on Software Outcome Measures**

What is the long-term impact of disruptive events on a software system?

As a starting point for evaluating the long-term impact of disruptive events, we will use time series analyses to compare outcome measures before and after a event, across different system areas, and through distinct time periods. The outcome measures will be based on the available archival data and the specific goal of each case study. For example, to determine the impact of testing on a system, we will look for disruptive events that targeted part of a system for increased testing. Does the number of defects decrease over time relative to other, less well tested, parts of the system?

Furthermore, large, long lived systems will have gone through multiple disruptive events. For example, a single system may have gone through a period where management decided that testing is more important than review and another period where the opposite was true. Since review and testing find different types of defects and have different costs, this type of project will provide an ideal opportunity to determine which technique is most effective at reducing post-release defects. We plan to compare the findings across multiple projects to increase their relevance and generality.

**M4: Allocating Resources and Risk Mitigation**

Which factors mitigate the risks associated with a disruptive event? How can we optimally allocate project resources after an event?

The purpose of this objective is twofold: 1) to examine the practices that successful projects employed to mitigate the risks associated with each disruptive event identified in our taxonomy and 2) to determine where developers should focus their effort for optimal efficiency. For example, consider the disruptive event of a key developer leaving a project. We would expect that successful teams would begin training a new developer before the key developer leaves the project, or that all important areas of the system would be owned by more than one developer. This mitigating factor would be apparent by who made commits to which part of the system. We would do a long-term comparison of the number of defects observed in systems that gradually introduced a new developer with those that did not.

Our goal is to optimally allocate resources on a project team after an event, apply appropriate quality assurance techniques to different parts of the system, and increase the awareness of the development practices that help mitigate the risk of event. We will work with KDM Analytics to integrate our findings into tool support for software engineers.

**M5: Practitioner's Guide and Tool Support**

A taxonomy of disruptive events will contain too much detail to be of interest to practitioners. We plan to develop a practitioner's guide which will name the common types of disruptive events and the successful and unsuccessful manners in which real projects dealt with them. Since our measures and statistical models will be too complex for immediate use by practitioners, we will integrate them into tools that practitioners already use (*e.g.,* Eclipse).

## 2 Background

In this section, we describe our work that relates to this proposal and then provide a brief discussion of the related literature.

**Software Peer Review**

My dissertation represents the first systematic and most comprehensive empirical examination of peer review on open source software (OSS) projects. Prior to my work, many OSS advocates claimed that "massive" distributed peer review occurred on OSS projects, but there was little empirical evidence to support this claim. I found that there is actually a minimalist review process that efficiently fits the relatively small group of experts that drive large OSS projects. Details of this and other work can be found in form 100 (referred to here as Ji, Ci). Through archival analysis, I extracted measures and developed statistical models that demonstrated the efficiency and effectiveness of OSS review. The original work was published at ICSE (C3), and a replication of the work, including five new case studies, an advanced statistical analysis, and multiple new measures, is under review at TOSEM (J2). To triangulate and enrich my findings, I used grounded theory on archival reviews and interviews with developers to make explicit the underlying behaviours and mechanisms that allowed the process to function effectively (C2). As a result of this work, DND asked me to evaluate the peer review process used by the Land Software Engineering Centre (LSEC). The LSEC review process is an adhoc mix of formal inspection practices combined with informal shortcuts. We discussed their requirements and, based on the findings from my dissertation and from other relevant literature, I described a number of possible lightweight, measurable peer review alternatives. I generalized and published these lessons and recommendations as an IEEE Software paper (J1).

The impact of this work has been a deeper understanding of OSS peer review and the possible ways it may be transferred to industry. In the current proposal, I plan to take the measures I developed during my dissertation to identify disruptive events (*e.g.,* a decrease in review near a release) and to quantify outcomes (*e.g.,* number of defects found). Furthermore, during qualitative analysis, I noted the discussions behind some disruptive events, such as introducing a review policy. I have also studied why developers leave an OSS projects (C6), and now propose to examine the impact of both these types of events.

**Centralized vs Distributed Version Control**

In traditional Centralized Version Control (CVC) events flow up and down (and publicly) via a central repository. In contrast, Distributed Version Control (DVC) facilitates collaboration in which work output can flow sideways (and privately) between collaborators. These sideways flows are a relatively new concept. The emergence of DVC has sparked much discussion of its merits and shortcomings, but until our work (C4,C5), DVC had not been empirically studied. We examined the repositories of 60 open source projects that had moved from CVC to DVC and interviewed lead developers on six projects (C4). We created simulations to model how increased branching might benefit developers by allowing them to work in isolation and then merge their events. This work was done in 2008 when DVC was a new technology that was not well understood by most developers. Five years later, we want to understand if the switch to DVC had a significant longterm impact on the software projects.

**Identifying Code Elements in Freeform Text and Code Snippets**
During a nine month postdoc with Dr. Robillard at McGill, I developed a technique and tool called ACER that can extract code elements (*e.g.,* classes, methods, fields) from freeform text and code snippets with an average precision and recall at or above 90%. In a paper that will appear at ICSE 2013 (C1), I used ACER to identify code elements and then classified them as salient to the informal documentation that contained them. ACER was developed in the context of informal documentation (*e.g.,* StackOverflow), but is not restricted to this document type and can process any textual document. ACER is fundamental part of this proposal and will be used to link developer discussions of disruptive events to the source code elements that may be effected.

## 2.1 Literature Review

Empirical software engineering and natural language processing describe the measures and methods necessary to identify and target disruptive events. The extensive literature on software development practices and risk management provide possible techniques to mitigate issues related to a disruptive event.

**Empirical Software Engineering and Software Analytics**

The goal of empirical software engineering is to measure and model software systems and processes [14]. There have been many practical findings in this field, including understanding system development [25, 15], defect prediction and bug triaging [21, 3], collaboration networks among developers [6], global software development [12], and recommendations on where to focus unit testing [31]. Our previous work has been in this area and we will apply the measures we have developed along with those created by other members of this community to identify and predict the outcomes of disruptive events.

Software analytics [9] uses the measures and methods from empirical software engineering to help developers and managers make decisions. Our proposal fits into this emerging area because we will perform analyses of disruptive events to aid developers in targeting areas of the system for increased quality assurance and maintenance activities.

**Natural Language Processing**

Hindle *et al.* [17] found that documents written in English are regular and predictable and that developers write code in even more regular and predictable ways. Much of the success in mining software repositories relies on the regular manner in which software developers work (*e.g.,* [30, 18]). However, NLP techniques have limited effectiveness when applied to developer discussion because it is difficult to discriminate between code elements and natural English. For example, a bag-of-words approach will result in code elements being processed in the same way as English words and will likely decrease precision and recall.

Identifying code elements using information retrieval techniques (*e.g.,* LSI, VSM) have very low precision and recall [2, 4]. In contrast, RecoDoc can identify code elements with a precision and recall above 90% by using the context in which a term occurs [11]. ACER also uses code element context, but can process more general corpora (See Form 100 for more detail and definition of code element context). We plan to use NLP techniques to identify discussions of disruptive events. We will then use ACER to identify the code elements in the discussions. Ultimately, we will be able to link discussions of disruptive events to the areas of the system effected by the event.

**Formal Processes and Agile Development**

An early reaction to the large number of failing software projects was the CMMI [19]. With CMMI, software projects can be at one of five levels, with higher levels having a greater degree of repeatability, formal processes, and measurement. CMMI certification and formal processes in general have a high cost which many companies do not feel is warranted. However, studies have shown that CMMI tends to improve project outcomes [16, 1]. In contrast, Agile methods emphasize "people over process" and "responding to event rather than following a plan" [5]. Many empirical results have shown that process tends to have a minimal impact on system quality when compared to product factors such as developer expertise and system complexity [29, 26].

Regardless of the development approach, studies in this area focus on the process or the project and the related outcomes. Our goal is to focus on the similar disruptive events that happen to multiple projects and to describe the practices and confounding factors that contributed to success or failure.

**Software Risk Management**

There is a substantial and important body of literature on risk in software engineering. Boehm and others have identified the most important risks as encountered by software development managers and described successful risk management practices [8, 20, 7]. Some of the risks identified are related to disruptive events, such as the introduction of a new technology, but most are macro risks associated with running a project, such as developing the wrong functionality. General risk mitigation strategies can be difficult to apply to specific disruptive events. We plan to classify and model the effectiveness of the risk mitigation practices we observe on software projects.

## 3 Detailed Proposal and Methodology

The proposed work falls into the area of empirical software engineering. Our theoretical perspective is that each study should involve multiple data sources and methodological techniques to develop pragmatic, grounded theories with the greatest "problem-solving effectiveness" [22]. Without triangulation, we can have statistical models that blindly ignore systematic bias or that have construct validity issues and that measure the wrong quantities [23]. Likewise, qualitative studies can result in significant investigator bias or a series of descriptive quotations instead of a grounded theory [13]. Mixed-methods research limits the bias that arises from 'blind' data mining and from overly descriptive research [10, 32]. Through my graduate work in Dr. Storey's research lab (she is the Canada Research Chair in Human Computer Interaction for Software Engineering), I have designed and conducted many different types of studies, such as controlled experiments, surveys, interviews, and archival analyzes, and used many different methods, such as statistical models and grounded theory. Below, we elaborate on the methodology that we will use to understand disruptive events. The timeline for each milestone is contained in the Activity Schedule section of this proposal.

**Identification and Modeling Outcomes**

In Milestone 1, our goal is to identify disruptive events, we will look for fluctuations and anomalies in the archival data (*e.g.,* a developer who consistently made 10 commits per week suddenly makes no commits for a month) as well as asking developers to recall the last disruptive event they experienced. The KDM Workbench, which contains advanced static analysis, will help us identify outcome measures related to security vulnerabilities. We expect to discover other techniques for identifying disruptive events, including examining software at particular stages in the development lifecycle (*e.g.,* just before a release) and with NLP techniques (C6). The result of this research milestone will be to identify the most important and frequent disruptive events.

In Milestone 3, our goal is to measure and describe the outcome of a disruptive event, we will adapt our study methodology based on the type of event. Disruptive events range from one-factor events to dramatic events that contain many confounding factors. One-factor disruptive events can be studied by master's students. For example, a master's student could examine the size of test cases relative to the size of the code base and determine the outcome of fluctuations in tests. Do new tests come in bursts? What is the effect of writing tests after code? A small survey of OSS developers would provide sufficient triangulation.

In contrast, a PhD student could study the impact of drastically reorganized project teams at DND. We would use a predominately qualitative methodology because there will be too many confounding factors to create a statistical model. In this latter case, we will interview new project leads to determine which qualitative themes influence whether a reorganization will be a success or failure.

**Factors and Confounds**

In Milestone 2, we will mine archival data. Data mining can be problematic because the archive can be missing the most important factors. At this stage, a significant risk is that we will be unable to extract and control for the most important factors when describing a disruptive event and determining its outcome on software projects. However, in my dissertation, I successfully created and validated proxy measures from archives records that allowed for comparison of OSS peer review to the results from studies of traditional inspection practices. Validating new proxy measures is a significant part of this proposal that will be of sufficient scope for master's theses. Currently, I have a master's student running comparisons between the standard system complexity measures, such as Halstead and McCabe's Cyclomatic complexity, with the software change complexity measures I developed during my thesis (*e.g.,* the distance between hunks in a diff file). Another example would involve comparing the developer expertise measures used in the literature [27] and validating them in the context of surveys or small controlled experiments.

**Targeting and Predicting**

In Milestone 4, our goal is to use the models and factors we will have developed above to aid management decisions and help in targeting development effort. For example, one of DND's objectives for this work is to understand which quality assurance technique should be applied to which parts of the system and at which times. Even large organizations, like DND, that deal with highly critical applications have limited resources to expend on quality assurance. To answer this question, we will conduct multiple case studies of large, critical projects in the OSS community, at LSEC, and at Microsoft to determine the impact on quality of fluctuations and explicit events in quality assurance regime. Our findings will allow us to make recommendations based on multiple factors. For example, we might find that when someone leaves a project, extra effort expended in review is superior to additional testing. A possible explanation would be that when developers perform peer review they must read and comprehend the code. We might also find that a one month delay in adding unit tests increases the number of post-release defects. We hope to contrast the impact of using testing, review, and static analysis on a software project.

## 4 Training of Highly Qualified Personnel (HQP)

Highly qualified personnel will learn how to conduct empirical studies in the context of software engineering. While many undergraduate students possess in-depth technical knowledge, Computer Science and Software Engineering degree programs do not typically produce researchers. Basic research and writing skills can be acquired by a determined individual, but developing intriguing, coherent research questions and stories requires an apprenticeship with a mentor.

I will mentor students in conducting both quantitative and qualitative studies. Students will learn to formulate research questions, collect and clean data, create scripts to operationalize measures, store and link measures and concepts in databases, and use data mining techniques and statistical analyses to answer our research questions. For qualitative studies, students will learn to apply grounded theory to abstract quotations from interviews and other sources into a small set of core, highlevel themes.

KDM Analytics will provide researchers at Concordia with licenses to the KDM Workbench and KDM Blade. These tool contains advanced static analysis and security vulnerability analysis and will be used in identifying disruptive events and security outcome measures. They will give a training session to Concordia researchers on system assurance and an information session on applied research and standardization in system assurance. Familiarity with the KDM products and training will position students at the forefront of the growing and important software security assurance industry.

Since this work is done in an industrial software development setting, students will be able to foster relationships with the organizations that they study and will interact with professional developers. The training they receive will be useful to them whether they continue in academia or join a company. Should students decide to join industry, they will have been exposed to disruptive events in real development settings. As professional developers and managers, this knowledge and experience will be useful in guiding the decisions they make in their work. Collaborating with professionals at KDM Analytics to develop tools to support developers in their daily work will allow students to gain insight into turning research ideas and prototypes into robust tools.

The proposed work falls into the area of software analytics where students conduct empirical studies to understand the impact of disruptive events and use this knowledge to aid development decisions and predict where problems will occur. While these analytics skills will be developed in a software engineering context, they are transferable to general analytics problems. Students will be ideally poised to work for companies that need to mine large data repositories, such as Google or Facebook.

PhD students will work on Milestones 1, 3, and 4, while master's students will work on Milestones 2 and 5. The work done by master's students in validating measures that can be used as variables in a statistical model or to control for confounding factors, will complement the work of PhD students who will be examining the most important disruptive events across multiple projects. For example, a master's student, Vatti, is examining possible change complexity measures that will be helpful in controlling for the complexity of the software during modeling of disruptive events. Since the milestones overlap and are complementary, PhD students will be able advise and collaborate with master's students. From my startup grant I am also funding a master's student, Donadelli, to perform a preliminary investigation of disruptive changes on the Apache Server, Linux, and Android projects. Undergraduate summer intern students will be responsible for developing tool prototypes that will be helpful to KDM Analytics as they decide which features are practical enough to be included in their tool suite. For more details on HQP please see form 101.

## 5 Team Expertise

This proposal is composed of a complementary group of partners. Concordia University will provide students and expertise in mining software archives. Researchers at Concordia will identify disruptive events from archival data and interviews with developers on OSS projects as well as projects at Microsoft. LSEC will contribute a professional software development environment that will have requirements on which we can adapt and validate techniques for mitigating the risk and predicting where to focus effort during a disruptive event. If a technique is successful at LSEC, then it may be adopted at DND. KDM Analytics will provide expertise in identifying disruptive events related to system security and will incorporate models and other findings into the KDM Workbench and other products they develop.

We will be collaborating directly with Djenana Campara and Dr. Mansourov, KDM Analytic executives, who co-authored "System Assurance: Beyond Detecting Vulnerabilities." Their book describes a standards based semi-automated approach to combining disparate tools to produce a cohesive solution to software assurance [24]. They specialize in software modernization, vulnerability detection, and assurance. KDM Analytics' approach is realized by extracting security and other specifications in a formal software assurance model. This model is combined with information extracted from source code and executables to create a formal language independent Knowledge Data Metamodel. The Knowledge Data Metamodel is an Object Management Group (OMG) standard that KDM Analytics helped develop. A semi-automated analysis, which is based on static analysis, is then performed to provide proofs of software compliance. Their approach is realized in the KDM Workbench, which won the GCN Best of FOSE, and is used to model software threats and to perform architectural vulnerability analysis.

The proposed partnership will combine the formal techniques used in the KDM Workbench with the measures we have extracted from software archives, including freeform text discussions between developers. Their formal standards based approach complements Concordia's pragmatic extraction of measures from available data. Furthermore, KDM's previous approach has been limited to source code and executables and was system centric. In this work, we plan to link multiple artifacts to understand and model the impact of disruptive events. KDM Analytics' expertise in relating formal software assurance will be extremely valuable when working in a critical, military software environment.

## 6 Benefits to Canada: Value of the results and industrial relevance

Our work will provide a catalog of disruptive events and their impact on software systems. The findings and statistical models in this catalog will allow us to optimize the development of particular parts of the system, improve overall quality of projects and help teams dealing with events to adopt effective practices. The factors we identify and the models we create will be integrated into tools that will help developers and managers in their daily work. Students involved in this work will develop empirical and mining skills, interact with professional developers, and receive training on the state-of-the-art software security assurance tools. They will be well prepared to continue in academia or join an analytics firm.

As a direct outcome of this research, we will produce "The Practitioners Guide to Disruptive Events." The guide will increase practicing software engineers' awareness of disruptive events and increase productivity by describing the appropriate risk mitigating responses to each type of event. Our findings will allow us to develop prototype tools and evaluate them in industrial development environments at LSEC. Successful prototypes will be included in KDM Blade and other software products.

A significant advantage of partnering with KDM Analytics, who specialize in system security and assurance, is that we will be able to identify the impact of disruptive events related to software security. Identifying system security vulnerabilities is beneficial to the Canadian economy because cybercrime was estimated to have affected 7.3 million Canadian internet users in 2010 and to have a direct cost of $840-million and a cost of $4.7-billion in lost productivity to the Canadian economy [28]. A disruptive event that we will investigate with KDM Analytics are security panics. A security panic occurs when a vulnerability is discovered in deployed software and a fix must be immediately developed. We are interested in how the team organizes itself during the crisis period. Also, during the panic, development effort will be diverted from normal work, and in the rush to produce a fix, technical debt may accrue with longterm negative impacts on the system. Our goal is to create models of the areas of the system that are affected by a security panic and to describe the successful and unsuccessful responses to security panics.

With the funding cuts to many branches of the Canadian government and the high quality requirements of military software at LSEC, it is critical to optimally allocate resources. Our models will help in determining when and where to use static analysis, testing, or review. Our qualitative, grounded theories will help in describing successful practices that save time and money. When disruptive events occur, such as staffing cuts on a development team, we will be positioned to provide advice that will mitigate the negative impacts associated with these events. The proposed work will make Canada a leader in understanding the practices and developing the tools that will allow software development to become a mature engineering discipline.

## References

[1] M. Agrawal and K. Chari. Software effort, quality, and cycle time: A study of cmm level 5 projects. *IEEE Trans. Softw. Eng.*, 33(3):145–156, Mar. 2007.

[2] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *Software Engineering, IEEE Transactions on*, 28(10):970–983, 2002.

[3] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 361–370, 2006.

[4] A. Bacchelli, M. Lanza, and R. Robbes. Linking e-mails and source code artifacts. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 375–384. ACM, 2010.

[5] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. The Agile Manifesto. 2001.

[6] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *MSR: Proceedings of the Third International Workshop on Mining Software Repositories*, pages 137–143. ACM Press, 2006.

[7] B. Boehm. Get ready for agile methods, with care. *ACM Computer*, 35(1):64 –69, 2002.

[8] B. W. Boehm. Software risk management: Principles and practices. *IEEE Softw.*, 8(1):32–41, Jan. 1991.

[9] R. P. L. Buse and T. Zimmermann. Information needs for software development analytics. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 987–996. IEEE Press, 2012.

[10] J. Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage Publications, Inc., 2009.

[11] B. Dagenais and M. P. Robillard. Recovering traceability links between an api and its learning resources. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 47 –57, june 2012.

[12] D. Damian and D. Moitra. Guest editors' introduction: Global software development: How far have we come? *IEEE Softw.*, 23(5):17–19, Sept. 2006.

[13] B. Glaser. *Doing grounded theory: Issues and discussions*. Sociology Press Mill Valley, CA, 1998.

[14] M. W. Godfrey, A. E. Hassan, J. Herbsleb, G. C. Murphy, M. Robillard, P. Devanbu, A. Mockus, D. E. Perry, and D. Notkin. Future of mining software archives: A roundtable. *IEEE Softw.*, 26(1):67–70, 2009.

[15] M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In *Proceedings of the International Conference on Software Maintenance (ICSM'00)*, ICSM '00, pages 131–. IEEE Computer Society, 2000.

[16] J. Herbsleb, A. Carleton, J. Rozum, J. Siegel, and D. Zubrow. Benefits of cmm-based software process improvement: initial results. Technical report, DTIC Document, 1994.

[17] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu. On the naturalness of software. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 837–847, 2012.

[18] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 117–125, 2005.

[19] W. S. Humphrey. Characterizing the software process: A maturity framework. *IEEE Softw.*, 5(2):73–79, Mar. 1988.

[20] M. Keil, P. E. Cule, K. Lyytinen, and R. C. Schmidt. A framework for identifying software project risks. *Commun. ACM*, 41(11):76–83, Nov. 1998.

[21] S. Kim, T. Zimmermann, E. Whitehead, and A. Zeller. Predicting faults from cached history. In *Proceedings of the 29th international conference on Software engineering*, pages 489 –498, May 2007.

[22] L. Laudan. A problem-solving approach to scientific progress. *Scientific Revolutions*, 1981.

[23] J. Maindonald and J. Braun. *Data analysis and graphics using R: an example-based approach*. Cambridge University Press, 2003.

[24] N. Mansourov and D. Campara. *System Assurance: Beyond Detecting Vulnerabilities*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2010.

[25] A. Mockus, R. Fielding, and J. Herbsleb. A case study of open source software development: The apache server. *ICSE: Proceedings of the 22nd international conference on Software Engineering*, pages 262–273, 2000.

[26] A. Mockus, R. T. Fielding, and J. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):1–38, 2002.

[27] A. Mockus and J. D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *ICSE: Proceedings of the 24th International Conference on Software Engineering*, pages 503–512. ACM Press, 2002.

[28] Norton. Norton cybercrime report 2011.

[29] A. Porter, H. Siy, A. Mockus, and L. Votta. Understanding the sources of variation in software inspections. *ACM Transactions Software Engineering Methodology*, 7(1):41–79, 1998.

[30] S. Rastkar, G. Murphy, and A. Bradley. Generating natural language summaries for crosscutting source code concerns. In *Proceedings of the 27th IEEE International Conference on Software Maintenance*, pages 103–112. IEEE, 2011.

[31] E. Shihab, Z. M. Jiang, B. Adams, A. E. Hassan, and R. Bowerman. Prioritizing the creation of unit tests in legacy software systems. *Softw. Pract. Exper.*, 41(10):1027–1048, Sept. 2011.

[32] R. K. Yin. *Case Study Research: Design and Methods*, volume 5 of *Applied Social Research Methods Series*. Sage Publications Inc., 3 edition, 2003.