Rethinking Software Process: the Key to Negligence Liability

Clark Savage Turner, J.D., Ph.D., Foaad Khosmood
Department of Computer Science
California Polytechnic State University
San Luis Obispo, CA. 93407
(805) 756 6133
csturner@calpoly.edu, foaad.khosmood@intel.com

Keywords: workflow modeling and applications, process constraints, negligence

Abstract

The risk of negligence liability can be considerably reduced by application of basic legal concepts early in the development lifecycle. The costs of preparing and defending a suit can be considerably reduced as well. This paper develops a simple model for integration of these considerations into the development process. The model stresses the following:
1. early consideration of process constraints derived from negligence law;
2. explicit consideration of such constraints during the development process; and,
3. documentation of all significant tradeoffs that is traceable to key constraints.

Introduction

Negligence liability costs can be among the most expensive for any safety-critical software development organization [See generally LT93, Ka95a]. Such costs commonly include not only a final award made by a judge or jury, but also the enormous investigation effort required to prepare a defense (even when a case never sees a court!) A common approach to this problem is to retain legal counsel and initiate a thorough investigation only when absolutely necessary: after a lawsuit has been filed. This approach has two distinct disadvantages:

1. the evidence crucial to the case may not have been recorded (or exist) because the software development lifecycle did not include legal process constraints; and,

2. the evidence gathering process becomes difficult and expensive because the lawyers are forced to evaluate the development process from meager process documentation.

The model presented in this paper will attempt to reduce costs by both reducing risks of negligence liability and reducing costs of defense against a potential lawsuit. It does so by integrating negligence considerations explicitly into the development process. This model requires the creation of a database where important legal constraints are linked to specific development aspects that address them. The database can serve as a rich, persistent source of evidence for the purposes of legal analysis. It enables partial automation of the legal evaluation that is often done manually, years after the fact.

Negligence Law

Negligence is based upon conduct that is socially unreasonable. It involves the endeavor to ".. strike some reasonable balance between the plaintiff's claim to protection against damage and the defendant's claim to freedom of action for his own ends" [Pro71]. Negligence imposes a set of expectations on behavior through the concept of duty. We all have a duty to act with "reasonable prudence" whenever our conduct foreseeably creates a threat of injury to others. This defines a portion of the economic exposure of an industry to foreseeable damages caused by its products (and services) whenever its failure to take reasonable precautions contributes to such damages.

Since the duty in negligence focuses on behavior, it yields constraints on development processes and not the products themselves. Processes that fail to meet negligence constraints may be a basis for liability. Processes that satisfy these constraints will not be the basis for liability, even if the product caused harm to an innocent party. Distinguish this from strict products liability for manufacturing defects [TR00].

Evaluation of processes for satisfaction of negligence constraints is a matter of evidence. The evidence usually comes in the form of engineers' testimony and process documentation. It must be evaluated by Court approved experts in the relevant field who look for strengths and weaknesses in engineering tradeoffs. The Court evaluates the tradeoff process according to a social risk-benefit analysis. Documentary evidence is persistent and not easily dismissed. Total reliance on human testimony about intricate process details (often years after the fact) is a very risky strategy [Wit85].

Unfortunately, many organizations must rely heavily on human testimony due to the absence of process documentation traceable to particular negligence considerations. Even when evidence is available, it is often difficult to locate and link appropriately. This time-consuming task falls upon the legal team, often a costly proposition.

Negligence Consideration in the Development Lifecycle

As Parnas noted in 1986, the software development process suffers from many inherent difficulties [Par86]. Among these is the lack of useful documentation. He recommends "faking" the ideal process by producing a *post-hoc* rationalization. However, he goes further to mandate a policy of recording all of the design alternatives that were considered and rejected. He requires an explanation of the tradeoffs that led to the choice. He shows that this sort of documentation aids in maintenance of the software. We believe that this simple principle can be extended to the realm of negligence: this sort of documentation is necessary to defend a negligence case.

The software development process is subject to the current set of negligence constraints as are other development processes [see, for example, the analysis in Ka95b]. Similarly, the process of developing the constraints themselves is subject to the current state of the art in software development processes [TRK96]. See figure 1.

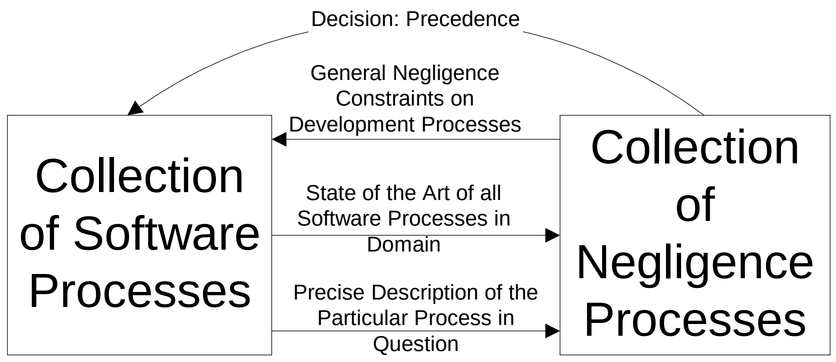
Decision: Precedence
General Negligence Constraints on Development Processes
Collection of Software Processes
State of the Art of all Software Processes in Domain
Precise Description of the Particular Process in Question
Collection of Negligence Processes

Figure 1: Software and Negligence Interdependent Processes

Consider this figure as a very high level view of the [co-evolutionary] interaction between the current *state of the art in software process* and *negligence law itself*. Through the decisions issued in particular cases, Courts establish general principles that outline the scope of *reasonability* in industry practices. In order to apply these principles accurately to a given process, Courts must rely on evidence regarding the state of the art in software process. Therefore, the well informed software organization will prepare to produce evidence regarding the following:

1. the state of the art in software process for the given domain; and,
2. a precise and accurate picture of the process in question.

The Court's ultimate judgment is negligence liability if 2 is significantly outside the bounds established in 1. On the other hand, if the organization can demonstrate that their process is well within the bounds of 1, they are judged non-negligent.

Negligent behavior may be found during any stage of the lifecycle. It may occur as a part of testing, requirements or the design process. Thus, negligence considerations must permeate every stage of the lifecycle. With this in mind, consider figure 2.

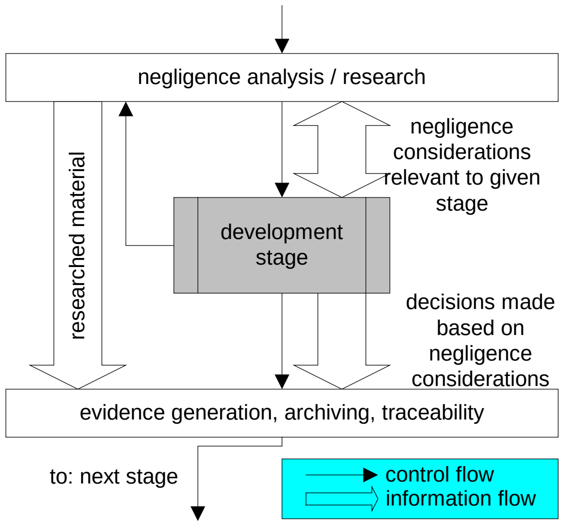
negligence analysis / research
researched material
negligence considerations relevant to given stage
development stage
decisions made based on negligence considerations
evidence generation, archiving, traceability
to: next stage
control flow
information flow

Figure 2: Proposed Enhancement to the Existing Software Development Stage

In this model, a single stage within the lifecycle is augmented to include negligence considerations. Any of the lifecycle stages may be placed in the gray box at the center. This model takes the original stage and encapsulates it in two additional steps that perform and record the negligence analysis. The first step, “negligence analysis / research,” consists of examining prior law and industry trends in order to understand what will constitute a reasonable process. A legal team on a medical device software project will examine industry trends in medical practices and similar software packages asking critical questions to help in the design stage. Have there been negligence lawsuits brought involving these sorts of packages in the recent past? If yes, what was the basis of the negligence? What is the industry consensus on what is reasonable when it comes to software design in this domain? What sort of evidence was relevant in such cases?

The legal data produced will be saved until the stage is nearly completed. Then it will be compared with the specific tradeoffs made during the process. This data provides new feedback to the stage itself: that is, tradeoffs could be made or altered based on the research that has just been done. Conversely, if a particular decision appears to warrant further legal or industry research, control is passed back to the first step and relevant research milestones can be accomplished before moving on in the process.

The second step is “evidence generation, archiving and traceability” which occurs immediately after the stage is considered complete. The purpose of this step is to document the tradeoffs made and match them with the key constraints found in negligence from step 1. The results must be stored in a database with easy indexing for future reference. This activity creates well-documented and researched evidence that can be conveniently retrieved at the time of a legal investigation. It has the potential to provide the legal team with a fair understanding of the situation quickly and at low cost.

The breadth and depth of this analysis is set according to a basic risk analysis. A riskier project requires much more effort than one of low risk. Safety-critical projects require extensive efforts while simple accounting package projects normally entail less.

Implications

The model presented here can enhance the product lifecycle in the following ways:

1. By focusing on relevant legal principles early and continuously in the process, and by providing the feedback mechanism back to step 1, the end product is likely of a higher quality prior to release [Ka95a].

2. By adopting and following the model presented, software engineering organizations can demonstrate a reflective built-in process that explicitly considers negligence. Merely having this process in place is a socially responsible thing to do! (For many reasons, it may be crucial to establishing that the development process was socially reasonable to a Court.)
3. By maintaining a comprehensive and well-established database, evidence for potential liability lawsuits can be readily at hand and easily produced. The legal team benefits from lower operating costs by reducing the investigation costs. This in turn may save further costs by leading to a fast verdict or a speedy settlement.
4. By carefully archiving the research and the evidence generated from previous products, the organization will be able to accomplish the same tasks faster and for lower cost in future ventures. The research and evidence generation techniques will fine-tune themselves in an evolutionary manner.

Shortcomings and Future Work

First, the scope of the application is narrow: the model presented in this paper is most appropriate for safety-critical development. Furthermore, the principles of negligence liability do not generally apply when a contract is involved. These facts exclude the vast majority of the software development organizations.

Secondly, the model as it currently stands does demand additional cost and time investment compared to the traditional development cycle. Adopting this model means an organization will have to incur higher costs prior to release for a promise of lower costs after the release. This can be difficult to justify especially since there may be no lawsuit in any event. It is the contention of the authors, however, that such costs would be far more reasonable under this model in the long run. The involvement of the legal team at an early stage would be cheaper and less time-consuming than an after-the-fact investigation.

In order to effectively implement negligence considerations throughout the life cycle of the software, the said considerations must be represented as additional milestones and constraints in the organization workflow. Most organizations already have detailed workflow documentation in place. This documentation, consisting of task procedures and scheduling information, is becoming increasingly electronic. Future considerations by the authors will include a demonstration of an electronic workflow system that implements their proposed model.

About the Authors

Clark Savage Turner is an Attorney and Associate Professor of Computer Science at California Polytechnic State University in San Luis Obispo, CA. His research interests include software systems safety and legal implications of software control.

Foaad Khosmood is Senior Software Engineer at Intel Corporation and a graduate student at California Polytechnic State University, San Luis Obispo.

References

[Ka95a] Kaner, "Software Quality & the Law" in The Gate, the newsletter of the San Francisco Section of the American Society for Quality Control, July, 1995, p. 1.

[Ka95b] Kaner, “Software Negligence and Testing Coverage,” The Software QA Quarterly, Vol. 2, No. 2, 1995.

[LT93] Leveson, Turner, “An Investigation of the Therac-25 Accidents,” IEEE Computer, Vol. 26 No. 7, July 1993.

[Par86] Parnas, “A Rational Design Process: How and Why to Fake It,” IEEE Trans. on Software Engineering, No. 2, Feb. 1986.

[Pro71] Prosser, Handbook of the Law of Torts, 4th Edition, West Publ, St. Paul, MN. 1971.

[TR00] Turner, Richardson, “Software and Strict Products Liability: Technical Challenges to Legal Notions of Responsibility,” Proceedings of the IASTED International Conference on Law and Technology, San Francisco, Oct. 2000.

[TRK96] Turner, Richardson, King, “Legal Sufficiency of Testing Processes,” Proceedings of the 15th International Conference on Computer Safety, Reliability and Security, Vienna, Austria, Oct 1996.

[Wit85] Witherell, How to Avoid Products Liability Lawsuits and Damages, Noyes Publications, 1985.