SOFTWARE AND STRICT PRODUCTS LIABILITY: TECHNICAL CHALLENGES TO LEGAL NOTIONS OF RESPONSIBILITY

CLARK SAVAGE TURNER
Department of Computer Science
California Polytechnic State University
San Luis Obispo, CA. 93407
(805) 756 6133, csturner@calpoly.edu

DEBRA J. RICHARDSON
Department of Info. and Computer Science
University of California, Irvine
Irvine, CA. 92697-3425
(949) 856 2131, djr@ics.uci.edu

Keywords: consumer protection, software safety, products liability.

Abstract

The law of products liability in tort is designed to maintain a reasonable balance between the inevitable social costs and the benefits of innovative product technologies. Technological development must be supported but only to the extent that society receives sufficient benefits to make the sacrifice worthwhile. The entire basis for implementation of this balance resides in the objective categorization of any given product defect into one of the following: (1) manufacturing defects: failures to correctly implement safety features from the design; and (2) design defects: failures of the design itself to exhibit socially acceptable levels of safety.

Software has been described as an artifact with fundamentally different properties than other engineered artifacts. This paper will show that software has unique characteristics that increase the overlap between the categories of defect such that known distinctions break down and become useless in promoting the social and technical goals that supported their creation in the law.

1. INTRODUCTION

Software[1] is a relatively new technological artifact to reach the consumer market. It has made possible a new array of technological possibilities with its speed, efficiency and general applicability. It is increasingly being tasked to control products that are socially valuable but have the potential for personal injury. It has already been involved in cases of human injury and death.[2]

When a consumer is injured and the common law of products liability is invoked, two different legal standards are available: negligence and strict liability. These standards have different incentive structures for the parties and may result in different outcomes when applied to similar cases. The application of each standard to the proper case is necessary to support the overall goals of the tort liability system. The applicable legal standard is determined relative to the category of the particular defect considered. A reliable test to determine defect category is therefore critical to the operation of products liability law.

Software products may be fundamentally different from traditionally engineered products.[3] If this is true, to what extent will the nature of this new product affect the application of current legal and engineering tools used to determine the standard of liability? Both legal and technical aspects of the question are at issue here.

Clark Savage Turner is an Associate Professor of Computer Science at the California Polytechnic State University in San Luis Obispo, CA. He holds a J.D. (Maine) and a Ph.D. in Computer Science (UC Irvine). His research interests include software systems safety and legal implications of software control.

Debra J. Richardson is Chair of the Department of Information and Computer Science at the University of California, Irvine. She holds a B.S. in Math and a Ph.D. in Computer Science (U. Mass.) She has done pioneering work in establishing the use of formal specifications to guide and evaluate testing and analysis of software systems.

[1] For a definition of "software," see generally Schach, *Classical and Object-Oriented Software Engineering*, 4TH Ed., McGraw-Hill, 1999; Gemignani, Product Liability and Software, *8 Rutgers Computer & Technology L. J.*, 173 (1981).
[2] Leveson, Turner, An Investigation of the Therac-25 Accidents, *IEEE Computer*, Vol. 26, no. 7, July 1993.
[3] Hamlet, Are We Testing for True Reliability? *IEEE Software*, 21, July 1992.

This paper is concerned with software embedded[4] in a larger physical system that is *already considered a product* for purposes of products liability law. Examples include avionics systems, ABS brakes, computerized ignition systems, and medical devices.

2. PRODUCTS LIABILITY

The modern law of products liability developed from roots in negligence and warranty causes of action. In the 1960's *strict products liability in tort* developed in response to the perceived inadequacies of negligence and contract-warranty causes of action when applied to products of modern complexity involved in personal injury.[5] It was to be based on proof of *product defect* rather than proof of *fault*. Once a product was proven defective, damages could be awarded. On the other hand, for a negligence case, fault (unreasonable conduct) must be proved, the injury may be merely economic in nature, and a product need not be the instrumentality of the injury (it could be a service). Further, products liability cases in tort *are not subject to contract or license disclaimers of liability*.[6] The common "limitations of liability" and bold statements negating the manufacturer's responsibility for the behavior of the product have no bearing on a products liability case.

There are two prerequisites for any case of products liability in tort: (1) personal injuries or harm to other property must be involved, "pure" economic damages will not support a case; and (2) the instrumentality causing the injury must be considered a *product* or a product *component*; provision of services won't support a case.[7] This paper is concerned with cases involving personal injury, so the first of these prerequisites is not at issue. The second involves a legal judgment that the software will be subject to the law of products liability. Some scholars have wondered whether software is really part of a "product" or more in the nature of a "service" performed for a client. If all software is classified as the provision of a service, any strict products liability analysis ends because the case does not involve a product.[8] Though this issue has not been settled by the courts, there is a general consensus among commentators that certain software systems are considered products for the purpose of products liability.[9] Software that is part of a hardware system (already a product) that is mass produced for the consumer market is the primary example.[10] The class of software systems considered here falls within this group.[11]

In 1998, after years of work and revision, the Restatement of the Law, Third Edition, Torts, Products Liability was published.[12] Section One provides,

> **Section 1. Liability of a Commercial Seller or Distributor for Harm Caused by Defective Products**
>
> **One engaged in the business of selling or otherwise distributing products who sells or distributes a defective product is subject to liability for harm to persons or property caused by the defect.**

The central legal idea that triggers liability for products liability law is therefore *product defect*. Two distinct categories of defect emerge in the common law.

---

[4] Embedded software is an integral part of a physical system, considered a distinct "component" by many software analysts. See generally Leveson, *Safeware*, Addison-Wesley, 1995.

[5] For a good historical view of the development of strict products liability, see generally Birnbaum, Unmasking the Test for Design Defect: From Negligence [to Warranty] to Strict Liability to Negligence, *33 Vanderbuilt Law Review, 593* (1980)

[6] Henningsen v. Bloomfield Motors, Inc., 161 A. 2d 69 (NJ 1960)

[7] Prosser, Keeton, *Prosser and Keeton on Torts*, 5th Edition, West Publishing Co., MN, 1984.

[8] The negligence standard applies to the provision of services. If software is indeed seen as service oriented, the attendant professional negligence issues will come to the forefront as pressure from the plaintiff's bar rises. For a general discussion of such issues, see, Kaner. Software Negligence and Testing Coverage, *Software QA Quarterly*, Vol. 2, No. 2, 1995.

[9] See generally Miyaki, Computer Software Defects: Should Computer Software Manufacturers Be Held Strictly Liable For Computer Software Defects? (Comment) *8 Computer & High Technology Law Journal 121* (1992).

[10] Products liability has already been used in a Georgia case involving an embedded software system, though the issues appear not to have been raised. See GMC v. Johnston, 592 So. 2d 1054 (Sup. Ct. Ala., 1992)

[11] It is an integral part of a physical product. One might also argue that the software is a "component" of a larger product system as in Wolpert, Product Liability and Software Implicated in Personal Injury, *Defense Counsel Journal*, October 1993, 519, 523.

[12] *Restatement Third, Torts: Products Liability*, American Law Institute, 1998

2.1 MANUFACTURING DEFECTS

As early as the 1200's, some forms of liability for *manufacturing defects* was imposed by the law. In the 1960's, American courts began to recognize that a commercial seller of any product having a manufacturing defect should be *strictly liable* in tort for harm caused by the defect. Liability is based completely upon the product's failure to satisfy the manufacturer's own design intention. The product is "more dangerous than it was designed to be." Once the product is shown to have caused injury, the proof of a manufacturing defect is sufficient to result in liability. Thus, the liability is to attach even if the manufacturer's quality control was reasonable. "Due care" of the manufacturer is irrelevant!

2.2 DESIGN DEFECTS

In the late 1960's and early 1970's, questions of *design defects* began to arise when the product in question satisfied the intended design but the design itself was unacceptably risky. In such cases, defects cannot be judged by reference to the manufacturer's own design standards because those are the very standards under scrutiny. The design defect involves a social judgment about the trade-offs necessary to determine which accident costs are more fairly and efficiently borne by those who incur them (the victims) and which are best borne by product users and consumers through internalization of the accident costs (by the manufacturers) and having product prices reflect the relevant costs. Judgment of design defectiveness is often based on the availability of a cost-effective alternative design that would have prevented the harm.[13] This is *not* a strict liability standard, but one more in the nature of *negligence*, based on lack of due care during the design process.[14] It is a legal conclusion based on social standards for design adequacy. Injuries may indeed be caused by design decisions, but unlike the manufacturing defect, if due care was exercised, the manufacturer is not held liable.

2.3 DISTINGUISHING DEFECTS

The *Restatement* distills the two theories[15] of product defectiveness into the following description of the law:

Section 2. Categories of Product Defect

A product is defective when, at the time of sale or distribution, it contains a manufacturing defect, is defective in design, [...] A product:

(a) contains a manufacturing defect when the product departs from its intended design even though all possible care was exercised in the preparation and marketing of the product;

(b) is defective in design when the foreseeable risks of harm posed by the product could have been reduced or avoided by the adoption of a reasonable alternative design by the seller or other distributor, or a predecessor in the commercial chain of distribution, and the omission of the alternative design renders the product not reasonably safe;

Note that Courts seek "intended design" as the marker to determine defect category and set the proper legal standard. Caselaw exhibits two ways that Courts determine this marker:[16] (1) design specifications, and, (2) deviation from the norm.[17] These are discussed, in turn, below.

When the Court searches for the manufacturer's intended design, a natural starting point is internal design documentation for the product. After all, the manufacturer often uses such documentation in its own efforts at quality control. In this sense, these documents can exhibit the manufacturer's intended design: a precise definition of what the manufacturer intended to produce. Ideal design specification documents contain a complete, consistent, correct, unambiguous, comprehensible expression of the product design. If such design documents were always on hand, if product features were always traceable to their counterparts in the specification, and if the specification counterpart could be used to answer the question, "is the specification satisfied?" they would be sufficient to reliably distinguish manufacturing defects from design decisions. This is not true in general.[18] Real design documentation often does not

[13] Banks v. ICI Americas, Inc, 450 S.E.2d 671 (Ga. 1994)
[14] See Birnbaum, *supra.,* note 5.
[15] Note that we do not consider the theory of "defective warning" here since it would not add to the discussion.

[16] See, for example, the cases cited in the *Restatement Third, Products Liability*, section 2 in the Reporter's Notes, comment c, Manufacturing defects.
[17] This test was so named by Justice Traynor in Traynor, The Ways and Meanings of Defective Products and Strict Liability, 32 *Tenn. L. Rev.* 363, 367 (1965).
[18] Requirements always contain conflicts such as speed and safety, efficiency and cost, etc. "It is quite impossible for any design to be the 'logical outcome of the requirements' simply because, the requirements being in conflict, their logical outcome is an impossibility."

provide enough information to make an unambiguous comparison to arbitrary product features.

If the specification is missing, incomprehensible, incorrect, inconsistent or ambiguous, it cannot be used to reliably divine "intended design." Some other way is needed to distinguish a manufacturing from a design defect. The "deviation from the norm" test compares the product in question to a number of others from the same production run.[19] If the given product defect is found in all the others, then it is said to be one of design intention. This fact gives rise to the descriptive term `generic' defects, referring to those defects that affect the entire product line (by design).[20] If the product feature in question is unique, it is likely a manufacturing flaw, the result of a "mistake," not a plan. This test is easy to understand and its genius lies in the ability to infer intended design from some sample of products - without reference to the design specifications at all. Simple and practical, it circumvents many known problems of a test to the design specifications.

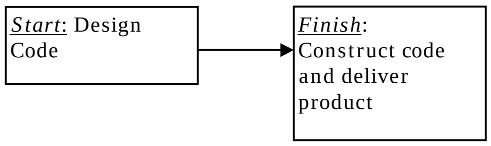
3. APPLICATION TO THE SOFTWARE PRODUCT

Similarly to other engineered products, the software product involves human decisions that exhibit engineering tradeoffs: reliability versus safety, cost versus safety, performance versus safety, etc.[21] These decisions are ideally recorded in software design specifications. Software research explains that the problems of consistency, correctness, completeness and ambiguity are serious and continuing ones for software products of nontrivial size and complexity.[22] These practical and omnipresent deficiencies with software specifications become problematic for a Court whose task is to determine whether a software product departs from its design intention. As we have seen, these problems are not unique to software.[23] Engineering experience and the common law have provided the deviation from the norm test to determine whether the product departs from its specifications without reference to the written design specifications. Will this test work for the software component of a product? No.

When software source code is constructed from design specifications (or other engineering "intention"), it is compiled into an executable form with other programs that give a perfect translation for practical purposes. This means that if the actual code deviates from design intention (a manufacturing defect), that defect appears in every product identically. Design defects have been called "generic" to indicate their presence in every product, but for software, the manufacturing defects that appear in the code are also generic.[24] This defeats the operation of the deviation from the norm test, the defects are no longer distinct from the norm for software! Thus, it is seen that for embedded software, the defect categorization fails to accomplish its goal of establishing the proper legal standard for liability.

3.1 SOFTWARE IS DIFFERENT

Similarly to all manufactured products, software production is discussed in terms of discrete stages where distinct activities occur.[25] First, software requirements and design activities are performed resulting in a software design document. After that step is complete, programmers construct computer code to implement that design. This is illustrated in the following figure.



This figure turns out to be a vast oversimplification. More realistic views of the software process recognize continuous feedback loops showing that these stages are not really discrete, but inevitably intertwined,[26] as shown in the next figure.

Pye, T*he Nature and Aesthetics of Design*, Van Nostrand Reinhold Company, 1978.

[19] See Traynor, *supra* note 17.

[20] "Generic" as applied to product defects is discussed in Henderson, Judicial Review of Manufacturers' Conscious Design Choices: The Limits of Adjudication, 73 *Columbia Law Review* 1530, 1543 (1973).

[21] See generally Leveson, *supra* note 4.

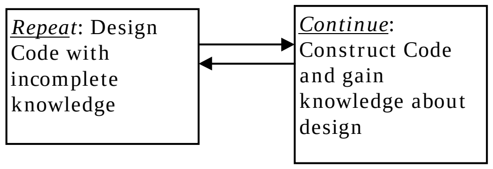
[22] See Jaffe, *Completeness, Robustness, and Safety in Real-Time Software Requirements Specifications: A Logical Positivist Looks at Requirements Engineering,* Dissertation, University of California, Irvine, UMI, 1988.

[23] Petroski, *To Engineer is Human*, Vintage Books, NY., 1992.

[24] It is interesting to note that another commentator has thought about a similar problem: the "inadvertent design defect." See Henderson, *supra* note 20. The term "generic manufacturing defect" was suggested by another researcher in this area, Cem Kaner, during our discussions about software defects.

[25] Most evident in the "waterfall model" of the software process, Schach, *supra* note 1.

[26] For a wonderful discussion of this issue, see generally Parnas, Clements, A Rational Design Process, How and


Repeat: Design Code with incomplete knowledge
Continue: Construct Code and gain knowledge about design

This feedback loop results from broad categories of intentional decisions[27] that are possible, necessary, and occur frequently during the programming activity for software.[28]

The scale and extent of this design activity during construction (programming) for the software product is of a degree and on a scale that is new to engineers.[29] If major design is necessarily done concurrently with construction, then the two activities merge as an activity and the line between them becomes very murky or vanishes. This mixing of design and implementation activities goes to the heart of our ability to distinguish design intention from construction activities in code.

For many traditionally engineered products such as automobiles, the medium of design specification is logical description, drawings, models and other ways of capturing design intention so that the product may be constructed in a physical medium, characterized by physical constraints. The design specification can be used to construct a product within acceptable "tolerance" and be said to meet that specification.[30] There is a workable dividing line between the design and construction of the product in that the physical medium is normally distinguishable from the medium of design. In difficult cases, the deviation from the norm test can be used to categorize a given defect.[31]

For software, the medium of design and the medium of construction are the same. The software coder, in the general sense, is only as constrained as the designer was in the construction of the product. With automobiles and many other traditional physical products, the construction of a particular product is heavily constrained by physical laws, by tooling, by training, by the parts made available by the management in the plant, etc.[32] These constraints are, for the most part, either missing or not as prevalent in software construction. Consider the following chart comparing software products to automobiles:

	Software	**Automobile**
Medium of Design	*Logic*	*Logic, drawings, ...*
Medium of Construction	*Logic (correctness)*	*Physical (tolerance)*

The code (software construction) activity is seen as one where both implementation (construction, inadvertent mistakes) and design (intentional choices) activities take place side by side. Since the deviation from the norm test fails for software products, the only possibility for distinctions is to find design intention in the design specifications and documentation. This method has already been shown to be unreliable.

3.2 TECHNICAL SOLUTION?

As seen above, the nature of the software product dictates that design activities necessarily accompany construction activities, recorded and intermingled in the code. Ideally, the design decisions made (or changed) in code are then recorded in some design document through the feedback loop shown above.[33] In the end, after the fact, this would result in a more "complete" set of design documents. However, what is the real incentive structure to such ideal maintenance of the design documents?

Why to Fake It, *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 2, 251 (1986).

[27] Nothing is really new here. This is true for automobile manufacturing, too. There is certainly feedback from the assembly line, especially during the early stages of production, where construction to the given design proves problematic and the designers must rework the design to accommodate production and physical reality. See generally, Petroski, *supra.*, note 23.

[28] This is explained in Parnas, *supra,* note 26. Note also that design choices that occur during construction are easily distinguished from the inadvertent manufacturing defects *for physical products* by the deviation from the norm test.

[29] See generally, Parnas, et. al., Evaluation of Safety-Critical Software, *Communications of the ACM*, no. 6, p. 636 (June 1990).

[30] Notice also that for physical systems characterized by "tolerance," overdesign, or design strength may be used to increase safety. Software cannot be easily characterized by "strength" to increase safety factors.

[31] This test can be used to determine the defect category "well enough" for social and engineering purposes. No arguments have been made that this is a bad way to decide, and it has been in use for some time. See generally Traynor, *supra* note 17.

[32] For example, the guys on the Ford Taurus line cannot just decide to build an Oldsmobile, and they cannot decide to build a boat that day. However, the software programmer is generally as free as the designer!

[33] Or maybe in the comments of the source code!

What is the possibility of creating such ideal sets of design documents for software? There are substantial factors that militate against such an ideal document capable of distinguishing design intention from inadvertent coding mistakes in the software product:[34]

1. it is expensive
2. it is time consuming
3. it is difficult, maybe impossible
4. many incentives for incomplete, ambiguous design documents[35]

The first 2 reasons may go without much explanation, it takes time and resources to accomplish such ideal documentation for a complex product. The difficulties of creating such ideal documentation are known and active areas of research in the software engineering community. Even the possibility of creating truly ideal design documentation is in doubt among commentators in the field of software engineering,[36] though methods for improvement are of great interest and the subject of large efforts.[37] Even if such ideal documentation could be created (or approached through application of great resources), the realities of the market must be addressed: the limitations of finite amounts of time and other resources.

4. CONCLUSIONS

Injuries involving software products will be dealt with under the law of products liability. However, in real cases, we need a method to distinguish the types of defect in order to properly support the goals of the law of products liability. This paper shows that there is currently no reliable method by which we can distinguish a manufacturing from a design defect in the software product.[38]

Embedded software products present new challenges to the law of products liability to form rules that support the basic goals of a safer society with balanced concern for the promotion of innovation. Discussions of the core issues must take place now, involving the main stakeholders: software engineers and personal injury attorneys. These discussions must take place before the difficult cases arise so that all involved may have the benefit of a thorough discussion of the issues that will confront them, and not be stuck making decisions that may prove to be irrational (or oppose basic social goals) in the long run.

[34] See Parnas, *supra*, note 26.

[35] In response to some discussions I had with a fellow student of software engineering, Arthur Reyes, he explained that in response to the definitions of product defect he would like to create a design document that is always satisfied, so that he could pull all cases against his product into the negligence realm. This is possible by writing specifications that are much less detailed, at a high level of abstraction, or “trivial” in the sense that most any product could technically satisfy them.

[36] Arguments against such ideal documentation may be found in Parnas, *supra*, note 26.

[37] Certainly the formal methods proponents believe that they have much to add to the value of design documentation, including preciseness and ability to create products consistent with design documents. See generally, Schach, *supra*, note 1 for pointers to some work in this area. The arguments are discussed in detail in Turner, *Software as Product: the Technical Challenges to Social Notions of Responsibility*, Ph.D. dissertation, Department of Information and Computer Science, University of California, Irvine, August, 1999.

[38] We might also ask whether we can solve the problem by use of the legal standards. Can we just apply the design standard to software, subjecting *all* defects to a negligence standard? This has been suggested by Miyaki *supra* note 9. It is a possibility. However, a fundamental anomaly in the law of products liability would appear by this “solution.” Is software, in general, of higher social value than medical devices and pharmaceuticals? Pharmaceuticals and medical devices are subject to the strict liability standard for manufacturing flaws even though the design standard is much different than ordinary negligence in recognition of such high social value. In fact, if software products were subject to a pure negligence standard a strange incentive structure is set up: incorporate more software with more responsibility for safety in any product and enjoy freedom from strict liability for inadvertent coding defects! According to noted software safety researchers Leveson and Turner (see note 2), replacement of mechanical safety systems by software will produce riskier products, not safer ones. This is in direct opposition to the fundamental goal of products liability: safer products.