Risk Management for Safety-Critical Software: A Unique Problem on the Horizon

Clark Savage Turner, J.D., Ph.D.
Associate Professor of Computer Science
California State Polytechnic University
San Luis Obispo, CA. 93407
(805) 756 6133
csturner@calpoly.edu

Abstract

This paper begins by examining the overall social risk management function of tort law. The common law of products liability has developed rules involving the categorization of product defects in order to operationalize this function. The legal rules are designed to provide economic incentives to balance unrestricted innovation with a desire for perfect safety. The incentives become critical input to the risk management process for software developers, managers, investors and insurers.

The software product has often been described as unique among artifacts of engineering innovation. In this paper, that uniqueness is shown to prevent a rational application of common law rules to basic examples of software defect. This defeats the desired operationalization of the social risk management function of tort law. Risk managers face a new uncertainty when considering safety-critical software products. Commonly considered solutions to the problem are examined and dismissed as inadequate.

Overview of my Argument

Risk managers must analyze risks and expected costs of liability in their long term planning and investment strategies. Tort law performs the function of operationalizing socially acceptable constraints on risk management for safety-critical products sold on the market. It provides risk managers a level of predictability and a measure of expected costs for injuries inevitably caused by such products. The constraints are implemented by application of specific legal rules in products liability cases.

Products liability rules have not yet been applied to software as a product or as a product control component. Software is said to be unique among the products of engineering. Therefore we must ask if this set of legal rules applies to software in such a way as to:

- rationally implement the social risk management function of tort law; and,
- produce a reasonable level of predictability and measure of expected costs for accidents as input to the risk management process.

The answer is, unfortunately, “no.” The legal rules cannot be applied in a rational way when software is considered.

This paper begins by an overview of the social risk management function of tort law and the way it produces input for the risk management process. The law of products liability in tort is then detailed and the fundamental distinctions upon which it is based are explained. These distinctions are then applied to software and shown to be unworkable. Technical progress is considered and seen as inadequate to a solution. Commonly discussed legal solutions are also considered and serious problems accompany each.

Introduction

Homo Faber, “man, the maker.” People build things, new things that have not existed before. Things that change the world. Designers use science, experience, and intuition to create new artifacts with desired properties. Aircraft, automobiles, nuclear power plants are but a few examples. The benefits to society are manifold: cheaper, more efficient transportation, new sources of energy.

If society had little desire for such technical progress, safety might be reasonably assured.[1] Long established, safe designs could be carefully improved in tiny increments. Verification techniques could be highly refined. Technical progress would then be limited by the designers' ability to fully understand and predict the behavior of their artifacts. This limitation would slow such progress to a snail’s pace. Further, the expense involved in creating fully safe designs would limit their usefulness, practicality and availability to large markets. This would bound the allocation of resources devoted to technical progress. For instance, would air travel have developed to its current state of technology (and economy) if accidents were not to be tolerated?

Society will not tolerate the arrest of technical progress in exchange for complete safety. We want more, better, faster and cheaper. We are sometimes quite willing to sacrifice a measure of safety for more performance, for more versatility, or for economy. Many risk tradeoffs are socially acceptable, but not all are.[2]

The presence of risk in design artifacts has serious social consequences. A tire blowout at high speed on a crowded freeway can result in human disaster. Such disasters cause economic damages and hardship to the victims. Older notions of justice would hold the artifact’s creators responsible as the “cause” of the victim’s damages.[3] Such a notion strictly enforced on an industry would prove too expensive and limit efforts at technical

[1] See generally, Petroski, *To Engineer is Human*, Vintage Books, NY, 1992.
[2] For a celebrated example, see Grimshaw v. Ford Motor Co., 174 Cal Rptr 348 (Cal. App. 1981)
[3] Witherell, *How to Avoid Products Liability Lawsuits and Damages*, Noyes Publishing, 1985.

progress. This would correspondingly limit progress in transportation, medicine, energy and other fields highly valued by society.

The law of tort, in recognition of greater social goals of progress, does not always allocate the costs of accidents to the creators of the artifact. The goal of this area of law is to maintain a reasonable social balance of risk and benefit by its allocation of costs due to accidents.[4] When the activity that led to the accident is socially valued more than the risk, the creators of the artifact may externalize the cost. The victim's interests are sacrificed for the good of the greater society, and he bears the costs alone.[5] However, if the social value of the risk outweighs the benefits, the creators of the artifact may be forced to internalize the costs of the accident by compensation to the victim. This scheme allows for a reasoned economic advantage to be given to designers of artifacts that promote long term social welfare. It is illustrated by simple diagram in Figure 1 below: [6]

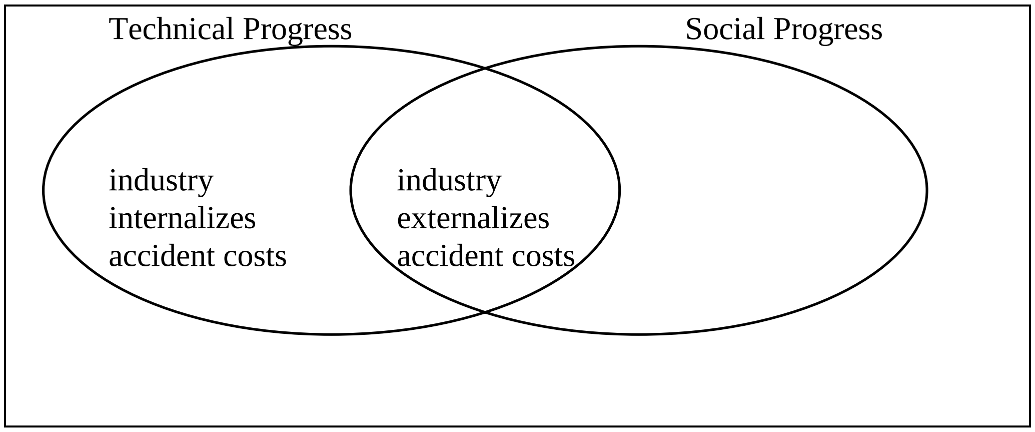

Technical Progress
Social Progress
industry internalizes accident costs
industry externalizes accident costs

Figure 1 - Social Risk Management

Products liability in tort deals with artifacts of design if they are sold to the public and involve personal injury. It implements the social risk-benefit analysis by its declaration of products as "defective" and therefore undeserving of social support.

[4] Prosser, Keeton, *Prosser and Keeton on Torts*, 5th Ed., West Publishing, MN. 1984.
[5] Church, family and other social welfare programs may ultimately spread these costs through society.
[6] This diagram is generally attributable to Nelson and Winter, *An Evolutionary Theory of Economic Change*, Belknap Press of Harvard Press, 1982.

The Law of Products Liability in Tort

Since the law of products liability in tort is a branch of the common law, it may evolve differently in each of the 50 states. However, the overarching principles of tort law by which states' rules are derived remain basically the same in all states. The American Law Institute has, for some years, prepared and published a standard text covering the basic tort principles and rules therefrom in its *Restatement of the Law* series. The Restatement is often considered authoritative for the common law and is cited by common law judges in tort cases.[7]

In 1998, after several years of scholarly work and revision, the Restatement of the Law, Third Edition, Torts, Products Liability was published.[8] Section One of this text 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*.

Product "Defect"

The Webster's Ninth New Collegiate Dictionary defines *defect:*

> ***1.*** **a: an imperfection that impairs worth or utility: SHORTCOMING [...]**
> ***2.*** **a lack of something necessary for completeness, adequacy, or perfection: DEFICIENCY [...]**

In general, then, a defective product will be one that contains some imperfection that makes it legally inadequate. This general idea has evolved through common law decisions for some time.[9]

[7] Smith v. Keller Ladder Co., 645 A. 2d., 1269 (NJ Super Ct. 1994).
[8] *Restatement of Torts, Third, Products Liability*, American Law Institute, MN, 1998.
[9] See *ibid*. for a brief historical perspective.

Technical Admission of Product Defect

The first sort of defect is easy to understand: the product is "more dangerous than it was designed to be."[10] Liability is simply based on the product's failure to meet the manufacturer's own design specifications. Indeed, if the product itself is built as some dangerous variation of an otherwise safe design, this product is defective by internal, technical standards, no legal analysis is required. Note that the good intentions, or "due care" of the manufacturer is irrelevant! There is no legally recognized utility to the creation of dangerous variations of safe designs. Such liability is said to be *strict* since no risk-utility analysis is required. This sort of defect is called a *manufacturing defect*.

Social Judgment of Defect in the Product Design Process

Some products adequately satisfy internal design specifications but the designs themselves evidence socially unacceptable risk taking. 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 and having product prices reflect the relevant costs.[11] Judgment of design defectiveness is often based on the availability of a cost-effective alternative design that would have prevented the harm.[12] 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.[13] It is a legal conclusion based on social standards for design adequacy. Unlike the case of a manufacturing defect, if due care was exercised, the manufacturer is not held liable.

The *Restatement* distills the above 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, or is defective because of inadequate instructions or warnings. 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;**

---

10 Prosser, *supra* note 4.
11 *Ibid*.
12 Banks v. ICI Americas, Inc, 450 S.E. 2d, 671 (GA 1994).
13 Birnbaum, Unmasking the Test for Design Defect: From Negligence [to Warranty] to Strict Liability to Negligence, 33 Vanderbuilt Law Review, 593 (1980).

(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;

(c) is defective because of inadequate instructions or warnings when the foreseeable risks of harm posed by the product could have been reduced or avoided by the provision of reasonable instructions or warnings by the seller or other distributor, or a predecessor in the commercial chain of distribution, and the omission of the instructions or warnings renders the product nor reasonably safe.

This work is limited in scope to analysis of the first two kinds of defects as they relate to software.[14] In general, defective warnings (the third category) are part of the product design and this is reflected in the similarity of the legal standard.

Legal Tests to Determine Defect Category

Notice that once the design intention is known to a Court, the category distinction can be made. If the product does not satisfy that intention, a manufacturing defect is present. If the intention is satisfied, the Court must look further to see if a design defect is present. Thus, the *Restatement Third,* section 2(a), 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 in a real case:[15] (1) design specifications, and, (2) deviation from the norm.[16]

1. *Design specifications as an expression of intended design.* When the Court searches for the manufacturer's intended design, a natural starting point is internal design documentation for the product. The manufacturer often uses such documentation in its own efforts at quality control. Ideally, these documents exhibit the manufacturer's intended design: a precise definition of what the manufacturer intended to produce.

2. *"Deviation from the norm."* If the specification is missing, if it is not comprehensible, if it is incorrect, inconsistent or ambiguous, it cannot reliably be used to divine "intended design. The "deviation from the norm" test compares the product

[14] Also note that the duty to warn has been questioned about its applicability to software. Warnings must be specific, and if a software designer knew of the specific risks, wouldn't the software be redesigned to reduce or eliminate the risk (as it relates to software)? See Gemignani, *Law and the Computer*, CBI Publishing Co. Inc., MA. 1981.
[15] See, for example, the cases cited in the *Restatement, supra note 8,* section 2 in the Reporter's Notes, comment c, Manufacturing defects.
[16] This test for manufacturing defects 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).

in question to a number of others from the same production run.[17] If the alleged defect is found in all the others, 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).[18] If the product feature in question does not appear in the majority of others, the defect is likely one in manufacturing: an unintentional failure to execute the design properly for that individual product.[19]

[Social] Risk Management in Operation through Liability Rules

Besides an interesting academic exercise, what is the importance of understanding the distinctions between defect categories? As discussed, these distinctions operationalize social notions of responsibility surrounding design artifacts that cause personal injury. When an artifact inadvertently fails to satisfy its own design intention, the possibility of social progress vanishes and the costs of accidents are to be internalized to that activity as manufacturing defects.[20] However, when the risks are intentionally taken as part of the designers' attempts at social progress, they must be carefully considered through the "due care" standard of negligence.

The distinctions are also important to anyone potentially involved in such a case because the different defect categories are subject to a different legal standard of proof. For a manufacturing defect, the plaintiff-victim has a relatively low expected cost for prosecution of the case: it is simply a matter of proof that the product did not meet the manufacturer's own design standards. The defendant-manufacturer has a relatively greater expected cost for such cases since damages are awarded simply on proof that the defect caused the harm: due care is no defense. For a design defect, the plaintiff-victim has a much higher expected cost of prosecuting the case, as it must be proved that the defendant-manufacturer did not act reasonably given the state-of-the-art. This involves a much greater burden of proof. Proving the state of the art is often a matter of disagreement even between experts. Correspondingly, the defendant-manufacturer now has a chance to prove that its design was sufficient even though the injury occurred ("due care"), thus the expected cost of such cases can be lower than that for manufacturing cases.[21]

[17] *Ibid*.

[18] The term, "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).

[19] 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.

[20] Owen, Defectiveness Restated: Exploding the Strict-Liability Products Myth, 1996 U. Ill. Law Review, 743.

[21] Prosser, *supra*, note 4.

Notice that, to the extent that manufacturing cases are seen as genuinely advantageous to victims, there will be pressure from the plaintiff's bar to *characterize defects as those of manufacture whenever possible.* A high level flow chart of a products liability case is shown in figure 4 to illustrate the pertinent concepts discussed so far:

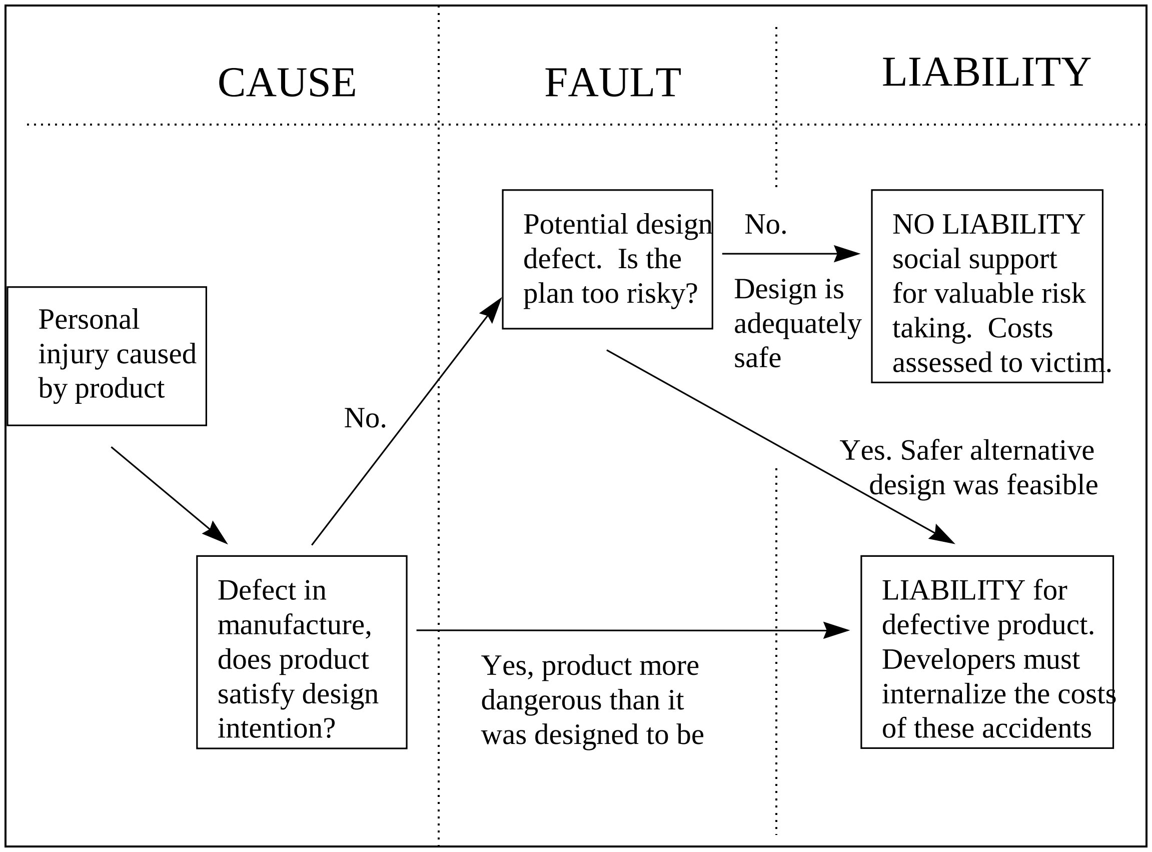

CAUSE
FAULT
LIABILITY
Personal injury caused by product
Defect in manufacture, does product satisfy design intention?
No.
Potential design defect. Is the plan too risky?
No.
Design is adequately safe
NO LIABILITY social support for valuable risk taking. Costs assessed to victim.
Yes. Safer alternative design was feasible
Yes, product more dangerous than it was designed to be
LIABILITY for defective product. Developers must internalize the costs of these accidents

Figure 2 - Anatomy of a Case

Once a product is proved to have caused or contributed to a personal injury, either kind of defect may be shown. The burden of proof and expected costs of prosecuting a case motivate the plaintiff to allege a defect of manufacture. This is the first issue to be decided, and if a manufacturing defect is identified, the plaintiff wins the lawsuit.[22] If the alleged defect is not one of manufacture, then defective design is at issue. Expert testimony can now be introduced by the defendant-designer in defense of the reasonability of the design. This defense was not available in the manufacturing case.

[22] In some cases, there are good reasons to go on to prove design defect in addition to manufacturing defect, such as the availability of punitive damages. Indeed, both classes of defect may be alleged and present in a product. See Prosser *supra*, note 4.

The determination of liability is then made respecting the much more complex issues of design and state of the art.

Enter Software

Software is a new artifact of design to enter the world of risk. It appears to be novel among other design artifacts: it doesn't break or wear out, it is not easily visualized in physical space, it consists of machine instructions and not physical components. It has been used to control nuclear power plant safety systems, commercial avionics, automobile brakes and medical devices. Flaws in such software can cause or contribute to personal injuries. For instance, in the most widely publicized software product accident to date, the Therac-25 medical linear accelerator caused several deaths and injuries over a two year period. An FDA investigation revealed that the software played a significant role in the accidents.[23] Software designers know that they cannot guarantee the safety of these systems, even with unlimited resources.[24] As society moves toward increased use of computer control of safety systems, we should expect more software accidents and resulting lawsuits.[25]

The common law that categorizes defectiveness has not yet been applied to software. When lawsuits were filed in the Therac case, the parties could not reasonably predict the outcome. Before the judge was forced to face novel legal questions about defectiveness of software products, the suits were settled to the satisfaction of the several victims or their survivors. No legal precedent was set. Note that there is little argument today about whether safety-critical software will be subject to products liability law.[26] The question is only how the law will be applied.

The Question Raised

Given the social notion of risk management embedded in the law of products liability, the investors, insurers, and developers have a stake in resolving uncertainties related to its application to the safety-critical software product. As has been shown, the categorization of defects in products cases is designed to implement extant notions of social risk

[23] Leveson, Turner, An Investigation of the Therac-25 Accidents, *IEEE Computer*, Vol. 26, No. 7, July, 1993.
[24] Leveson, *Safeware*, Addison-Wesley, 1995.
[25] Weber, Bad Bytes: The Application of Strict Products Liability to Computer Software, 66 St. John's Law Review, 469 (1992).
[26] The point has been argued respecting other sorts of software. For a good explanation and other arguments, see generally, 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-Nov, 2000.

management. This is implemented by economic incentives that provide input to the product risk management process. Even if the particular notions are not optimal, the rules provide the predictability needed to provide a rational basis for product risk management. How does this social risk management apply to the software product?

Given the unique nature of safety-critical software as a product, do the legal rules properly implement the underlying social risk management function of tort law?

The answer is, unfortunately, "no." Software is unique in that inadvertent mistakes in building the specified product (manufacturing defects) cannot always be distinguished from design intention by any currently known legal or software engineering method.

This looks like a legal question. It is. It looks like a technical question. It is. The legal, technical and social understandings that lead to rational normative rules depend completely on accurate descriptions of the software artifact. Ultimately, it is the software designers who will explain the nature of their artifact to a Court and assist in the process of creating accurate understandings and working rules to implement social goals.[27] We therefore examine the software product from the technical perspective. We then apply the legal categorization of defects to see the result.

Defect Categorization for the Software Product

Recall that Courts use two tests to divine "intended design" as the marker to distinguish defect categories: (1) the product design specification; and, (2) the "deviation from the norm." First consider the deviation from the norm test in the context of software code produced from software designs. Next, design specifications are considered. Finally, a simple programming example is provided to illustrate the practical difficulties that have been discussed analytically.

The Deviation from the Norm Test Fails for Software

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 all practical purposes.[28] If the actual code as written does

[27] For a wonderful discussion of these general issues, see Jasonoff, *Science at the Bar*, Harvard Press, 1995.

[28] Copies of software are produced from some master, usually a CD or diskette, and each copy is the same, to a very high degree of certainty. Though defects in this process would indeed be considered to be manufacturing defects, this paper will not consider them further. For more information about this, see Hamlet, Are We Testing For True Reliability, *IEEE Software*, July, 1992.

not correctly implement the design, then the code deviates from its design intention and the defect is one of "manufacture." Thus, *any given defect appears in every "copy" of the software product identically*. Recall that 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.[29] *This defeats the operation of the deviation from the norm test, the defects are no longer distinct from the norm for software!*

Since the deviation from the norm test fails for software due to its very nature, a Court is forced to look to other sources to divine "intended design." The other test involves product design specifications.

A Test to Design Specifications Fails for Software

Similar to other engineered products, the software product involves human decisions that exhibit engineering tradeoffs: reliability versus safety, cost versus safety, performance versus safety, etc.[30] These decisions are ideally recorded in software design specifications. They exhibit the ultimate intentions of the designers. Notice that a specification that does not correctly capture design intention will not be of any use to a Court whose job is to find true design intention. Similarly, inconsistency in the specification will require the Court to guess at true intention by means other than the specification. Even worse, an incomplete specification provides no guidance at all wherever there are gaps.

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.[31] *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.*[32]

Thus, it is seen that for the software product, the defect categorization completely fails to accomplish its goal of establishing the proper legal standard for liability.

[29] It is interesting to note that another commentator has thought about a similar problem: the "inadvertent design defect." See Henderson, *supra* note 18. The term "generic manufacturing defect" was suggested by another researcher in this area, Cem Kaner, during our discussions about software defects.
[30] See generally Leveson, *supra* note 24.
[31] 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.
[32] These problems may not be new to software. See Petroski, *supra*, note 1.

A Basic Example of Software Defect

In order to illustrate the practical problems this paper has discussed analytically, a simple example is provided.

Consider the following fragment of a software design specification:

> **"... increment this_variable."**

Now consider the code segment written to implement this simple design:

```
this_variable := this_variable + 1 ;
```

This is reasonable. However, suppose the coder made a simple typing mistake to produce:

```
this_variable := this_variable * 1 ;
```

This simple error is not always easy to detect by inspection or testing. It may produce results arbitrarily different than intended, and it is clearly not one of design intention. It is in the nature of an unintentional mistake, a "manufacturing defect" since it meets the legal definition: the product does not satisfy its own specification.

As this paper asserts, some software defects will not be easy (or possible) to categorize. Consider another specification:

> **"... safety_variable shall contain 0 on a safe condition."**

This would be a "flag" to some other part of the system that something should or should not happen while a "safe" condition is present. Now look at a sample code fragment written to implement this specification:

```
safety_variable := 0 ;
while (unsafe condition is true)
        safety_variable := safety_variable + 1 ;
```

Of course, **safety_variable** has a certain "size" in computer memory, when it is incremented beyond its limit, it "rolls over to 0" and begins again. This means that if the unsafe condition persists for a certain amount of time, the flag **safety_variable** may hold (maybe just for a short machine cycle) a **0** during an unsafe condition.[33] Since this variable involves the safety of the system, serious consequences can result. This seems to contradict the specification. This defect may be explained in terms of an unintentional defect in manufacture: the coder may have intended to write:

```
safety_variable := 0 ;
while (unsafe condition is true)
        safety_variable := 1 ;
```

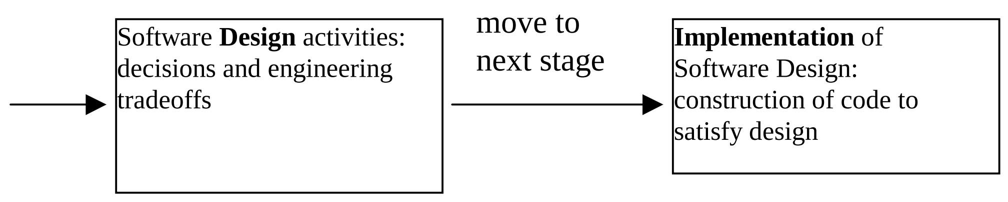
Incrementation of **safety_variable** would have been an unintentional mistake in typing and nothing more. Variable "roll over" is not a problem for this code. But now consider the specification itself. Does it contain enough information to guide the coder to avoid the safety conflict? It does not, for example, absolutely require that **safety_variable** be set to a nonzero value when the system is in an unsafe condition. The designers may have imagined a very low probability that the system would ever remain in an unsafe state for enough machine cycles to make any difference, or the chances that the machine would misbehave on a single machine cycle may have been considered and dismissed as unlikely. The coder may have had reasons (cycle counting or timing, for example) to increment the variable rather than repeatedly assigning it a static value. We cannot tell from the specifications or from the code exactly what happened. This software defect cannot be objectively categorized as the law requires.

What is Unique about Software that Causes Problems with Defect Categorization?

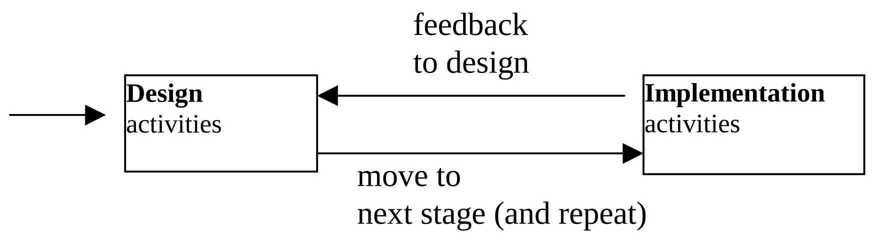
Similar to all manufactured products, software production is often discussed in terms of discrete stages where distinct activities occur.[34] Consider the following diagrams to illustrate this process.

[33] This example is abstracted from an actual example of personal injury that involved software programming defects. See Leveson, Turner, *supra, note 24.* For a more detailed analysis of software programming defects respecting products liability, see Turner, *Software as Product: the Technical Challenges to Social Notions of Responsibility*, Ph.D. dissertation, Department of Info. and Computer Science, University of California, Irvine, August, 1999.

[34] Most evident in the "waterfall model" of the software process. See generally, Schach, *Classical and Object-Oriented Software Engineering*, 4th Ed., McGraw-Hill, 1998


Software **Design** activities: decisions and engineering tradeoffs
move to next stage
Implementation of Software Design: construction of code to satisfy design

However, more realistic views of the software process exhibit feedback loops showing that these stages are not really discrete, but intertwined.[35]


feedback to design
Design activities
Implementation activities
move to next stage (and repeat)

The feedback loop from software implementation to design results from broad categories of intentional decisions [36] that are possible, necessary, and occur frequently during the programming activity for software.[37]

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.[38] If major design is necessarily done concurrently with construction, then the two activities may merge as an activity and the dividing 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 (implementation) activities in software code.

[35] For a wonderful discussion of this issue, see generally Parnas, Clements, *A Rational Design Process, How and Why to Fake It*, IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, 251 (1986).
[36] 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 1.
[37] This is explained in Parnas, *supra,* note 35. Note also that the design choices that occur during construction are easily distinguished from the inadvertent manufacturing defects *for physical products* by the deviation from the norm test.
[38] See generally, Parnas, et. al., *Evaluation of Safety-Critical Software,* Communications of the ACM, no. 6, p. 636 (June 1990).

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.[39] 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. And in difficult cases, the deviation from the norm test can be used to place a given defect in a category. Consider the following chart comparing software products to automobiles:

	SOFTWARE	AUTOMOBILE
MEDIUM of design	Logic	Logic, drawings, ...
MEDIUM of construction	**Logic** (characterized by "correctness")	**Physical** (characterized by "tolerance")

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.[40] These constraints are, for the most part, either missing or not as prevalent in software construction.

Progress in Software Engineering Fails to Solve this Problem

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.[41] In the end, after the fact, this would result in a more "complete" set of design documents. However, what is the real incentive

[39] 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.

[40] 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!

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

structure to such ideal maintenance of the design documents? 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:

1. it is expensive,
2. it is time consuming,
3. it is difficult, impossible in most situations[42].

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,[43] though methods for improvement are of great interest and the subject of large efforts.[44] 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. With no technical solution in sight, we turn attention to some often suggested legal solutions.

Commonly Discussed Legal Solutions Fail

License disclaimers have often been used to attempt to avoid liability (externalizing the costs of defects by contract). Can they be used in this situation to give some predictability to the process? No. Under the law of products liability, disclaimers are deemed as ineffective because they are seen as fundamentally unfair whenever personal injury is involved.[45]

Another basic solution would be to apply a single legal standard to both categories of software defect. The standard would be known in advance and risk managers could use it to predict expected costs as needed. Consider each possible standard in turn:

- All software products could be subject to the strict liability legal standard alone. This would appear to be a very consumer friendly approach, but is easily dismissed as

[42] It is interesting to note that in response to the threat of manufacturing liability for his product, a fellow student of software engineering proposed “trivial” specifications that would be technically satisfied by most any code product written.
[43] Arguments against such ideal documentation are outlined in Parnas, *supra*, note 35.
[44] 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. The arguments are discussed in detail in Turner, *supra note* 33.
[45] See *Restatement*, *supra note 8*, section 18.

unworkable due to the strong disincentive to innovate and take socially valuable risks in design.[46]

- All software products could be subject to the negligence standard. This has been suggested by Miyaki.[47] It is a possibility. However, a fundamental anomaly in the law of products liability would arise. Pharmaceuticals and medical devices are subject to the strict liability standard for manufacturing defects even though the design standard is much different than ordinary negligence in recognition of its high social value.[48] But is software, in general, of higher social value than medical devices and pharmaceuticals? 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 products and enjoy freedom from strict liability! According to noted software safety researchers Leveson and Turner, wholesale replacement of mechanical safety systems by software safety systems will produce riskier products, not safer ones.[49] This is in direct opposition to the fundamental goal of products liability: an incentive towards reasonably safe products for society.

Conclusions

Product developers, insurers and investors are risk averse and require information to perform their long term goals. Predictability in risk and expected costs for products liability forms the basis for such analysis and planning.

Software products present new challenges to the law of products liability. The biggest challenge is to form rules that support socially responsible risk management and a favorable environment for innovation. This challenge is not a simple one, this paper shows the obstacles are fundamental, not simply superficial.

Software engineers, lawyers, and product risk managers must engage in discussions of core issues before the problem becomes immediate. The solution must involve serious changes in our understandings of products sold on the market.

[46] This is the reason for the "design" category of product defectiveness. See generally, Owen, *supra* note 20.
[47] 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).
[48] See *Restatement, supra* note 8 section 6.
[49] See Leveson, Turner, *supra, note 23*.