Civil Law and the Development of Software Engineering

Martina Shapiro

September 25, 1996

Contents

Abstract

Abstract

This paper provides software engineers with an understanding of the basic tenets of civil law and how legal liability principles may be applied by the courts so as to affect the future direction of software engineering. The issues discussed are based on a review of selected literature in software engineering, civil law reported case decisions and texts as well as on interviews with legal consultants. Examples of some court rulings in cases involving software malfunction are included, along with some projections relating to the possible implications for software development arising from the anticipated reaction of the courts to the novel issues presented by software engineering.

1 Introduction

Since the 1950's, the cost of producing software has been rising and the cost of producing hardware has been dropping. Hardware seemed to become more reliable, while software was becoming more complex and less reliable. This trend has remained quite consistent over the years.

Software, being a part of many products, now pervades all aspects of our lives. The fast growing, profit-driven industrial need for software of great complexity, together with the ad-hoc software development methods still practiced by the majority of North American software development companies, have resulted in many project schedule and budget overruns. Software errors have caused not just considerable economic damage, but also human injury and loss of life, since software is an integral part of many safety-critical systems.

Some well publicized software-related failures include the Therac-25 radiation overdose incidents that claimed many lives[9], the failed first launch of the space shuttle in 1981 that occurred because of a timing problem in one of its computers and the AT&T phone system failure in 1990, amongst others[53].

Software development has traditionally been treated more like an art-form than a science, which has led to both quality and production problems. Software engineering grew out of the need to make software development consistent, disciplined and quantifiable, and thus better able to find the right balance between product quality, project schedule and budget.

Software engineering is a relatively young and evolving discipline, with no widely agreed-upon standards, and with no required licensing of its engineers similar to that of other professions such as doctors, civil engineers and accountants. As public exposure to software-controlled, safety-critical products continues to increase, so too does the risk of legal liability accruing to anyone who is involved in the development and marketing of software products.

Because of the nature of what software is and because of the fast-changing state-of-the-art of software engineering, the judicial application of civil law principles that were developed over the past two centuries to the relatively new discipline of software engineering is not yet well defined. It is therefore not surprising that software development companies often prefer the certainty of settling out of court, rather than risk the wrath and predisposition to perverse damage awards of relatively unsophisticated judges and juries.[34]

There are no easy answers on how to apply existing law to information technology, and traditional legal theories may not easily be made to address the peculiarities inherent in software. Software is an intangible and therefore is often considered to be an idea which has, on occasion, led to a legal definition of software as a service rendered to solve a problem. Software, however, has to be recorded on a physical medium if it is to be of any practical use, and so it has also been legally defined as a product. This distinction between product and service is important, since different legal theories apply in each case.

Another issue, quite specific to software, is that software solutions are being developed in many application areas. Software developers may be experts in software engineering, but typically they do not have expert knowledge in each application area for which they are developing software. Collecting and analyzing requirements thus becomes tricky, since they have to rely on others to provide them with specific information about the area in question in order to derive a requirements specification that is reasonably complete.

This leads to an interesting question relating to the spreading of liability between the client contracting for the software, the company developing the software and expert consultants hired by either party in cases where some safety-critical requirements were not discovered.

Many accidents can be attributed to human error, but, under closer scrutiny, inappropriate design of the interface may be the real reason for the error of the operator who was interacting with the software program. Defining legal responsibility for designing "user friendliness" will be another difficulty faced by both the courts and software developers.

To protect oneself from potential liability arising out of contract and tort-related lawsuits, it is necessary to understand who might be held liable and for what damages when software fails, and how far-reaching that liability can be. This issue is also of interest to insurance companies who have to quantify the financial risk involved in undertaking software coverage in order to determine their insurance premiums. Even though a company is liable for all of its employees, one can be held personally liable, even when working for a company, whether as an employee or officer: US cases have established that officers of a company may be personally liable to pay compensation for incidents relating to safety of the company's products as well as environmental damage if their own acts or omissions prove to be the direct cause of the loss which forms the subject of the claim.[37] The company CEO can also be liable to criminal prosecution for a company's failure to comply with certain duties imposed by statute.[37]
Third parties outside of the contractual nexus with the company may also look to individual software engineers employed by the company where they can trace their loss to that person.

Because of the nature of software and its incursion into all aspects of today's economy, awareness of the legal issues that arise in software development as well as an understanding of basic civil law principles, can assist practitioners in dealing with the many trade-off situations faced in industrial software development, including the mitigation of potential liability exposure of both individuals and companies involved in developing and marketing software-based products and systems.

The author makes no warranties nor representations as to the completeness of the legal research contained in this paper as it has been prepared by a non-lawyer, based upon a cursory overview of civil law jurisprudence as it pertains to software development. The paper is meant to be an overview of basic legal issues as they apply to the design, development and marketing of software, and as such should prove to be of primary interest to people associated with the software development industry. If specific legal questions require to be answered, the reader is advised to seek the professional advice of legal counsel.

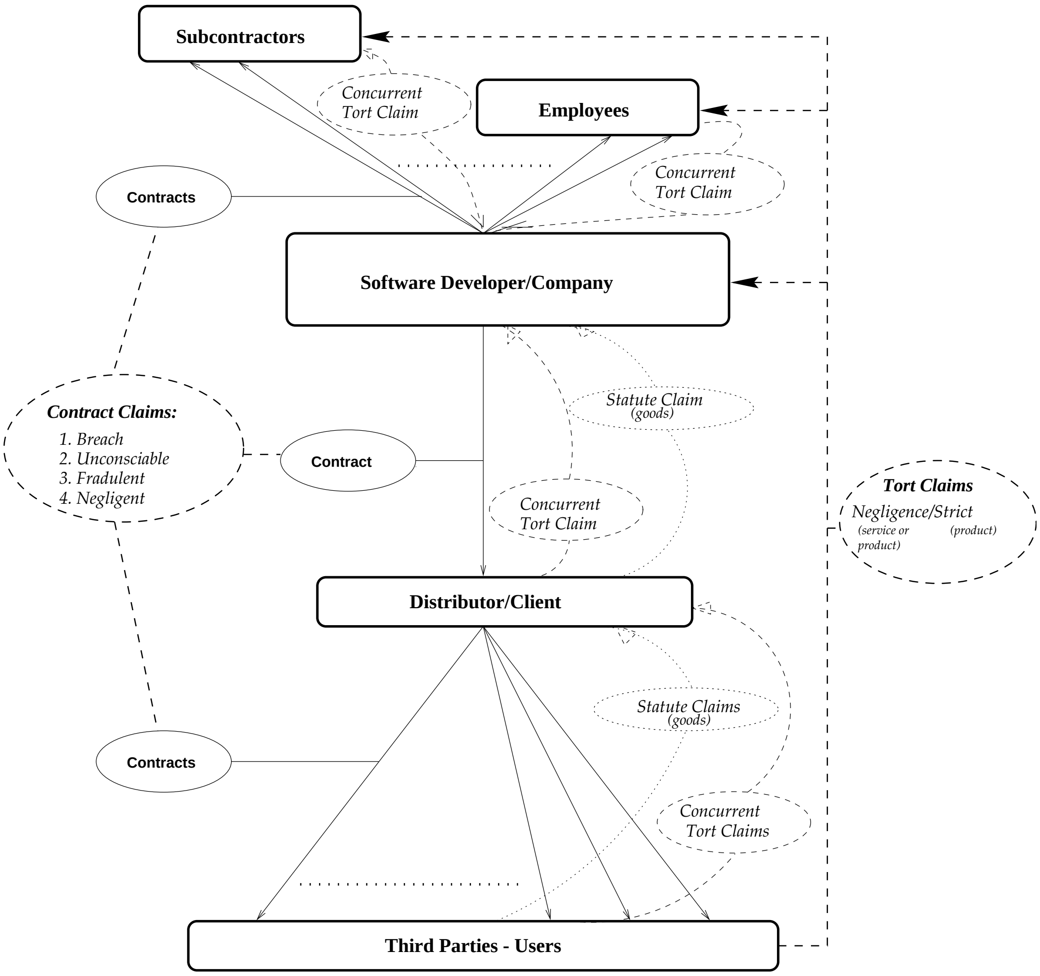

Figure 1: Possible Claims

2 Civil Law Concepts

In this section, civil law concepts are defined and explained.(Figure 1.) This section results from the author's research of various legal texts as well as from discussions with legal practitioners with expertise in this area.[34] For a detailed explanation of the legal concepts, see, for example, [28], [29], [30].

The basic legal principles presented in this section, have evolved over the last 250 years and form the basis of the civil law in the USA, Canada, UK and many commonwealth countries. These concepts will not change fundamentally in a court's attempts to apply them to software engineering. What is subject to change is state legislation (see Statute Law), but these changes will not affect the basic principles, just the way these are applied to specific fact situations. An understanding of the basic principles can therefore assist a software engineer in anticipating the potential liability exposures or likely outcomes relative to specific software design situations.

2.1 Tort Law

2.1.1 Negligence

A claim of negligence, in the context of software development, usually arises when a software product causes economic damage or physical injury to a "third party" that does not have a contract with the manufacturer of the product. The challenge for the injured party (plaintiff) is to prove who was negligent and how the negligent act caused the plaintiff's injury.

Black's Law Dictionary[30] defines negligence as:
"The omission to do something which a reasonable man, quided by those ordinary considerations which ordinarily regulate human affairs, would do, or the doing of something which a reasonable and prudent man would not do. " (p. 1184)

A finding of negligence is thus based upon an objective test which applies to every field of endeavour, profession or calling. It is the doing of something which a fictional "reasonable man" would or would not have done under similar circumstances, given the state of the art of his/her profession or calling at the time that the act complained of was committed. For negligence to exist, someone must be found to have failed to meet the test described above, and that failure must have affected someone else within the forseeable risk of harm in a deleterious manner so as to cause that person injury, loss or damage, either of a physical or economic nature.

Whether or not the objective test has been met is ultimately determined by a judge or a jury looking at all of the evidence, including expert evidence, if required. Note that the "reasonable man" does not have to be perfect. An error in judgment or a mistake is not necessarily indicative of negligence, and thus does not automatically incur tort liability.

The reasonable man standard has been strengthened for professionals, who are expected to perform to a higher standard than an ordinary prudent lay person.[29] A good definition of reasonability is cited in [29]:
"The degree of skill consistent with the function discharged, that is, consistent with the

measure of skill displayed by others reasonably competent in that profession touching matters of like kind. Perfection is not expected; the world of work, not the ideal of the debating area, is the standard."(p. 134)

2.1.2 Strict Liability

Strict Liability applies to products that are inherently dangerous or have the potential to cause damage or injury to others. In the software development context, strict liability arises most often when a software product causes physical injury and the product can be classified as being inherently dangerous in itself, or a software defect which cannot be empirically observed by the user renders the product "unreasonably dangerous". In such cases, the plaintiff does not have to prove any negligent act on the part of the manufacturer. The monetary awards for damages in these situations can be huge, and may include punitive damages imposed on the manufacturer and paid to the plaintiff. The Therac-25 radiation therapy machine is a good example of a product to which strict liability can be applied. The main cause of the machine's failures which caused radiation overexposure to many people was considered to be a software error.[9] The victims who suffer physical injury as a direct result of the use of such a product or the families of deceased victims, can file suits, both for negligence and strict liability. However, a claim based upon strict product liability in such a case might be easier to prove, and might result in a significantly larger monetary award than a claim based on negligence alone.

Black's Law Dictionary[30] defines injury as follows:
"Any wrong or damage done to another, either in his person, rights, reputation or property...The words "damage", "loss" and "injury" are used interchangeably, and, within legislative meaning and judicial interpretation, import the same thing." (p. 924)

Strict liability is often used in situations where a product, inherently dangerous in its own right, is allowed to damage others who are within the foreseeable risk of harm[1]. Therefore, it has been historically used to compensate claimants who suffered loss or injury from water, fire, electricity, gas and explosives.

According to strict liability, a person may be required to compensate another for injury or damage, even though the loss was neither intentionally nor negligently inflicted. One may attempt to avoid a finding of negligence by testing one's products for all forseeable applications. But such testing cannot prevent a finding of strict liability for products which, in and of themselves, may have inherently dangerous characteristics which may cause damage.

The concept of strict liability is employed by the courts to maximize protection to the public by placing responsibility on manufacturers whose products are potentially dangerous to others. The fact that the courts will impose strict liability upon a manufacturer in the absence of fault acts as a deterrent to manufacturers producing defective products for sale in the marketplace.

The courts will consider what an ordinary consumer of the product in question would expect. This "ordinary consumer", like the "reasonable person" in negligence cases, is a fictional character, and acts as an objective standard. This ensures the court's objectivity, since the performance of the product is compared to what this "ordinary consumer" would have expected of it, rather than what the particular person who is claiming damages did or did not expect of it. [18]

2.2 Contract Law

2.2.1 Contract

Black's Law dictionary[30] defines a contract as:
" A promissory agreement between two or more persons that creates, modifies, or destroys a legal relation." (p. 3940)

A contract is thus a voluntary agreement, which may or may not be in writing, supported by consideration passing from one party to another. Contracts may contain warranties, disclaimers of responsibility and limitations of liability.

In the context of software development, contract-related claims arise most often because of project schedule and budget overruns or because the delivered software does not function as claimed by the seller. As long as the contract is considered valid, all the limitations in the contract usually apply, which might mean that any monetary compensation and the liability for loss and damage incurred will also be limited by the contract. A well written contract can, to a large extent, protect against potential liability exposure (except for strict liability). Such a contract, in all but the simplest of cases, can be very complex and should be prepared with the assistance of an experienced lawyer who specializes in both contract and computer law.

1. **Warranty**
   A warranty is typically contained within the terms of a contract for the sale of a product, specifying the parameters of the product's performance and ascribing responsibility for its defects or shortcomings. It is a qualification on a contractual obligation from one party to another, narrowing the scope of responsibility for defects or deficiencies in a product to those which are expressly provided for.

2. **Disclaimer**
   A disclaimer is a warning or notice to another party that a product either should not be relied upon or that it should be relied upon only for a limited purpose. The disclaiming language notifies others about the limited reliance that they should place upon the product.

3. **Limitation of Liability**
   A limitation of liability is an agreement between two parties that the supplier of the product or service will only be responsible for a limited period of time and/or a limited sum of money for any [2] loss, injury or damage which may arise out of the manufacture, sale, distribution and use of a product or service. Typically, a limitation applies to physical and economic damage and/or the time within which a claim must be initiated.

---

[1] *According to the objective test of reasonable forseeability, ie. what a reasonable man could forsee.*

2.2.2 Breach of Contract

A breach of contract is the failure of one party in a contractual relationship to live up to its material obligations as required by that relationship, and gives rise to a claim for damages which are caused by and are the direct result of that failure. After a breach of contract has occurred, the non-breaching party may elect to either waive the breach and carry on with the contract or declare the contract to be terminated and sue for damages directly arising from the breach.

The terminating party must be correct in ascribing responsibility for the breach of contract to the other party. If it is incorrect, then the terminating party will be held to be in breach for improperly terminating the contract, and will itself be responsible to the other party for all damages which directly flow from its own breach. Termination of a contract means that the contract is at an end and the parties to the contract are no longer obliged to carry on with their respective responsibilities under the contract.

2.2.3 Rescission of a Contract

Rescission means the retrospective cancellation of a contract from its inception. Black's Law Dictionary[30] defines it as follows:
"Annulling or abrogation or unmaking of contract and the placing of the parties to it in status quo." (p. 1472)

Reasons for obtaining rescission are:

1. **Innocent Misrepresentation**
   In innocent misrepresentation, one party misleads another innocently as to a material element of the contract, without intending to mislead. The remedy in this case will be rescission of the contract, provided that the parties to the contract are capable of being put back into the same position as if the contract had never existed. If this is not possible, then the contract will continue to be in effect. Damages will not be awarded for innocent misrepresentation.

2. **Negligent Misrepresentation**
   Negligent misrepresentation occurs when one party misrepresents the facts to a second party, where the first party, acting reasonably, should have known that the facts were not accurate and the second party would likely rely upon them. This type of misrepresentation will give rise to rescission of the contract if the parties can be put back into substantially the same position they were in at the time the contract was entered into. Damages that directly arise out of the negligent misrepresentation can also be claimed.

3. **Fradulent Misrepresentation**
   Fradulent misrepresentation occurs when one party misrepresents the true facts to another party, where the first party knew the facts were not true when it stated

[2] *If the type of damage/injury/loss are specified, then the limitation will only apply to exactly those, and no others.*

them, but stated them anyway, recklessly, assuming that the second party would act upon the misrepresentation to its detriment. Fradulent misrepresentation will result in a claim for rescission as well as a claim for damages. As in 1. and 2., one is only entitled to rescission if the parties to the contract can be put back into substantially the same position they were in at the time the contract was entered into.

2.2.4 Strict Liability and the Contract

Two parties to a contractual relationship can agree as to the scope of responsibilities and obligations that each will have to the other. These may be beyond or less than the scope of responsibilities and obligations imposed by the common law. However, when a product is inherently dangerous, or if it is against public policy for one party to hide behind limiting words contained within a contract, particularly where the public health, safety and welfare are concerned, then the courts will impose strict tort liability notwithstanding the existence of limiting provisions within the contract. That is, in this type of situation, the contract will have no effect in limiting the scope of liability or amount of damages that can be recovered.

2.2.5 Vicarious Liability

Vicarious liability is the liability that one party has for another, whom it engages either as an employee, subconsultant or subcontractor. That is, the employer is vicariously responsible for everyone he engages contractually to perform a service on his behalf, even when the service represents a portion of production of a larger product which is sold by the employer to a third party.

2.3 Statute Law

Statute Law is the law enacted by a legislature. For example, the Sale of Goods Act of the Province of British Columbia is a statute enacted by the legislature of British Columbia, and applies to the sale of goods within the Province, unless it is excluded by the terms of a contract. The Sale of Goods Act implies requirements that products be fit for the purpose for which they are intended and that they be of merchantable quality. Goods may be considered to be of "merchantable quality" or "fit for their purpose" even though they may have a number of defects, as long as they are capable of performing their main function.[15] The exact meaning of these terms has been subject to considerable debate in the case law.[26] The statute will protect those who purchase products without the benefit of explicit warranties contained within written contracts.

In the USA, the Uniform Commercial Code is employed by all states, except Lousianna and Washington, D. C., to standardize sale of goods legislation throughout the country.[34]

2.4 The Legal Process

When a person is injured or suffers economic damage or loss as a result of using a product or service, he may seek a remedy through the legal system. He visits a lawyer and advises him of his concerns. The lawyer evaluates the facts to determine if a basis for a legal claim exists and ascertains if there is any potential for the complaining party (plaintiff) to be awarded damages. If there is, then the lawyer will initiate a lawsuit by drafting a Writ of Summons and filing it in a court of appropriate jurisdiction.

When the Writ is stamped by the court, it becomes an official court document and is served by the plaintiff on the defendant. The Writ of Summons officially notifies the defendant that a legal action has been initiated against him. The defendant should respond by filling an Appearance in court, which is a declaration of his intention to answer the plaintiff's claims by the time prescribed by the Rules of Court.

After the Appearance document is stamped by the court, it is served on the plaintiff, notifying him that the defendant intends to defend himself at a future trial of the matter.

Next, the plaintiff's lawyer elaborates on the Writ of Summons by setting out in more detail the factual basis of the claim and the legal remedies to which the plaintiff feels he is entitled. The resulting document is called a Statement of Claim and again has to be filed in the court first and then served on the defendant. The defendant answers by filling a Statement of Defence, which is a detailed rebuttal to the Statement of Claim. All these court documents are referred to as "pleadings" and serve to define the issues in dispute and the scope of the litigation.

The pleadings may be amended at any time, including at trial, with the permission of the judge as more evidence becomes available during the pre-trial discovery process and later during cross-examination at the trial itself.

After all the pleadings and other relevent documents have been received by the court (this ensures that the other party was notified), the examination for discovery process can begin. During this process, witnesses are formally examined by both parties according to the rules of evidence and Rules of Court.

The trial can only begin after the discovery process has been completed and all the facts and documents are available. (Figure 2.)
The discovery process consists mainly of the following elements:

1. Discovery of Documents.
All documents that are relevant to the issues in dispute must be produced by each party to a claim to the other, prior to oral examination for discovery and trial on those issues.

2. Discovery of Witnesses of Fact.
Each party is entitled to cross-examine a witness or witnesses from the other side as to the witness's knowledge of the facts relating to the issues in dispute. This discovery is taken under oath before a court reporter, and the answers given may be later selectively read in at the trial by legal counsel, without the necessity of calling the witness at the trial.

3. Discovery of Expert Witnesses.
Expert witnesses may be called to testify in fields directly related to the products and processes giving rise to the damages claimed. An expert witness must be qualified academically and practically in his field of endevour in order to be accepted by the judge as

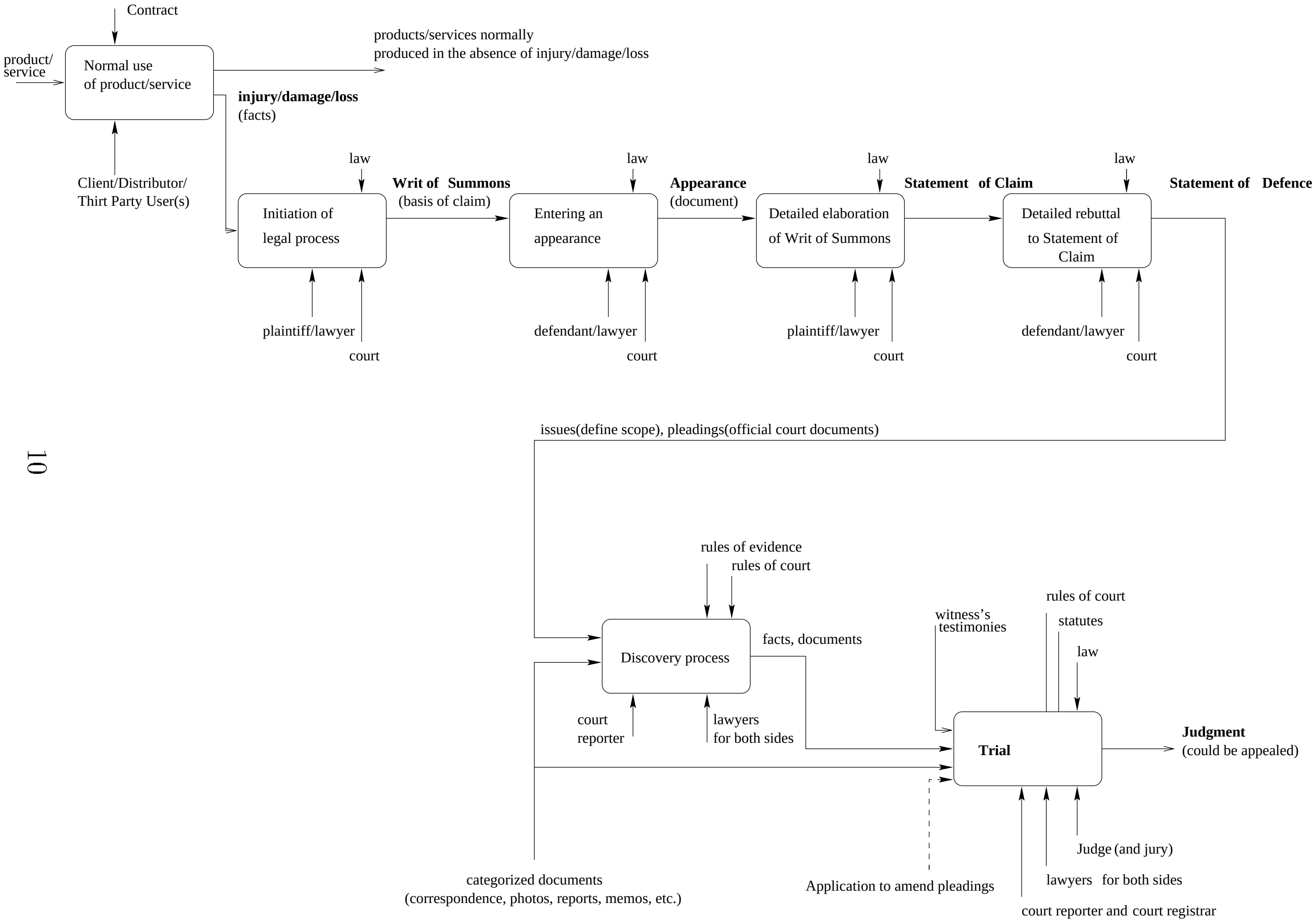

Figure 2: The Legal Process

an expert witness whose evidence may be given and received at the trial. The evidence of expert witnesses engaged by the parties must be produced to the other party in British Columbia at least 60 days before the witness gives evidence at the trial of the matter.

2.5 International Differences

Expanding international markets for computer software are indicative of the increasing globalization of the world economy. This trend has resulted in the need to regulate international transactions, including software transactions. The differing laws and legal systems between countries will affect these transactions and result in significantly disparate remedies between jurisdictions. Software consumers will thus naturally try to enforce their legal claims and seek damages in the jurisdiction that provides the most likely forum for recovery, and with the greatest potential damage awards.[34] The legal aspects that can arise under these circumstances are very complex and are not being dealt with in this paper. The purpose of this section is to bring this issue to the reader's attention.

There can be significant differences in the magnitude of damages awarded in the USA and in England or Canada. In the US, where juries decide on issues of liability and financial compensation, damages can be awarded that appear to be out of proportion to the loss suffered. US juries are also far more likely to award punitive damages than would a Canadian or U.K. court.[34] In Canada and UK cases are heard, for the most part, without juries, and the judge determines issues of liability and the level of damages. Judges tend to keep damage awards reasonable and in accordance with the actual loss suffered.[34] A plaintiff can 'shop' for the best forum, even if he has very little, if any, connection with the legal system where he sues.[36] However, a properly drafted commercial software contract will specify the law of a particular jurisdiction that will apply to all claims or disputes arising out of that contract.[34],[36]

Legislation that will affect the international exchange of software systems includes the United Nations Convention on Contracts for the International Sale of Goods, passed in 1980. This was an attempt to unify the international sale of goods laws as was the Single European Act of 1986, that relates to the economic unification of Europe with goals such as common standards, broad definition of what constitutes "goods", and the removal of legal and fiscal barriers to trade.[45]

Most EC States have implemented the EEC directive on product liability. The Consumer Protection Act of 1987, for example, defines damage as death, personal injury or property damage, excluding damage to the product itself. The term " product" in this legislation includes "any goods or electricity", and does not include services.[36] There are a number of other European directives that can be applied to computer controlled systems. The Health and Safety at Work Act 1974 legislation in England imposes general duties upon manufacturers producing or supplying anything intended for use at work. [36]

Other EC directives relate to product safety requirements for control systems of machinery to be "user friendly", and stipulate that errors in logic should not lead to dangerous situations.[15] They do not provide a definition of these terms.[15] Other proposals call for a directive that would introduce strict liability for suppliers of services, thus making the issue of whether software is a product or service irrelevant. This results in, for example, the suppliers of software maintenance services having equal responsibility with the producer of the software.[15]

3 Software Engineering Concepts

This section is meant to provide a brief introduction to the discipline of software engineering, its origins, purpose and challenges.

The term “engineering” in the discipline’s name is irritating to some, since it indicates the use of scientific methods and the production of valid results which are not always achieved by software engineering. In fact, it is not even clear how to define who is and who is not a software engineer as there still is no industry-wide agreement on a standard for this discipline.

Software engineers study how best to manage, control, measure and evaluate the software development process, as well as how best to select and combine individual methods, standards, documentation, human resources and skills, while mitigating both overall project risks and the technical risks associated with the final product.

The number and nature of variables arising from the effort to find the right combination of management, engineering, computer science, human and other factors necessary to the development of diverse software systems operating in differing environments makes it difficult, if not impossible, to run controlled experiments and thus obtain valid results. It is therefore questionable whether software engineering will ever become a ‘true’ science.

3.1 Software Development

In the late 1950’s and early 60’s, the software development process consisted, for the most part, of writing all of the software code with little or no prior planning or proper reasoning about requirements and design. The now obvious problems resulting from this approach led to an understanding of the need to develop conceptual models for organizing, staffing, budgeting and managing software projects, and marked the beginnings of software engineering as a discipline.

Software engineering grew out of the hardware and systems engineering fields in an effort to bring discipline, consistency and quantifiable application of engineering practices to the software development process. The field is continuously evolving and its main objective is to optimize the quality of software products and systems, while meeting specified requirements, budgets and delivery schedules. The challenges faced relate to the exploding complexity of software, software volatility, ambiguity in specifications and verifiability of final products. Initial software engineering efforts resulted in the development of the first software life-cycle models, while later efforts focused on defined software processes, measurement and improvement strategies.

Among the earliest life-cycle models was the “document-driven” waterfall model, which became the basis for most software acquisition standards.[48] The basic development stages defined in this model were the requirements, design, coding and testing, integration and deployment stages followed by operations and maintenance stage. The name "waterfall" reflected the idea of water flowing down a series of steps. The need for feedback loops between successive stages was, however, quickly recognized, and the model was reworked but not renamed.

Alternative models were developed to address some fundamental shortcomings, including the waterfall model's emphasis on complete documentation after each phase and a lack of support for evolutionary change.[48] Newer models address the incremental and evolutionary development of software, but, it is safe to say, that none of them is universally applicable. Companies typically adapt these models to their particular needs and then use them to define in-house software development processes.

The so-called "spiral model" of evolutionary software development was driven by experience gathered from using the waterfall model and its refinements for large, mission-critical government software projects.[48] This model is often described as *risk-driven* and accomodates most previous models. It is depicted as a spiral, with the inner cycles representing early analysis and prototyping, the outer cycles depicting the classical system life-cycle phases, with the radial distance representing cumulative development costs. The angular dimension represents the progress made during each development spiral. Risk analysis is carried out in each cycle of the spiral.[49]

Each spiral cycle begins with stating the objectives of the part of the final product in question, such as required performance and functionality, and the alternative ways of implementation and their constraints. The next step consists of evaluation and the identification of uncertain areas that could be sources of project risk. The following step then strives to resolve or mitigate these risk sources. The other potential steps in the model are implemented only to the extent required by risk considerations. For example, a software prototype of a new algorithm might be developed in order to reduce the risks related to the algorithm's performance. This allows flexibility and the choice of a mixed strategy, using the best features of other software process models and, at the same time, avoiding many of the problems associated with these models.[48]

The risk-driven approach emphasized by this model and the efforts to apply and refine this model have led to the establishment of the software risk management discipline, including techniques for risk identification, analysis, prioritization and risk-management planning.[48] It should be apparrent that effective risk mitigation goes hand-in-hand with liability mitigation, that is, effective risk reduction should reduce the likelihood of failed projects and unsafe products.

Research and development in software engineering is on-going and dynamic, dealing with both higher level processes and lower level methods and support tools. While more effort and valid results are needed for both the process and the method aspects, the emphasis appears to shift from one to the other. The focus on defining the underlying processes in software development started in the 80's, mainly with the work of the Software Engineering Institute (SEI). The aim was to define processes that are repeatable, produce consistent results and can be constantly improved.

Software processes focus on the specification of higher-level development activities, broken down into sub-processes, identifying the inputs and outputs for each activity, together with the required entry and exit criteria, methods, standards and tools to be used and the specific roles and skills of the teams and individuals required to carry them out. Process and product metrics are identified, collected and analyzed within this context.

Software methods can be thought of as detailed level processes that do not lend themselves to the more general purpose process representations. They are typically unique to the specific software product in terms of notations and tools (eg. state-charts for requirements specification, object-oriented design methods, complexity metric method, etc.).

Processes and methods have already been defined which, if properly implemented, can reduce both the overall project risk and the risk associated with the safety of the end product. Adoption of these processes is thus of interest to companies trying to mitigate the risk and the liability associated with the production of software systems.

Research in software processes and methods was initially driven by the desire to make software development more predictable in order to increase productivity. As the software employed in systems and products became increasingly complex, quality and safety considerations increased in importance.

Formal methods came into existence as a means of reducing errors in critical systems. Through the use of formal methods, it is theoretically possible to write all the requirements in a mathematical notation, prove them correct, then sucessively refine this initial specification, proving each step, until the refinement is so detailed that it can be directly translated into code or the specification itself can be executed and even used to program a customized chip. If this can be done, we have proof that the final code exactly corresponds to the initial requirements, and has the stated safety properties. The initial requirements, of course, can be incomplete or not the ones intended. Furthermore, the proofs themselves could be erroneous and/or the tools used to facilitate this process might represent an additional source of error. In practice, formal methods are used selectively because of the costs involved in applying them. They are often used to assist in writing unambiguous requirements specification, to formalize and prove correct the safety-critical portions of a system and to enhance the testing process. Together with other safety analysis methods and techniques, and an optimized testing process, formal methods can be very valuable when trying to achieve proveable dependability of safety-critical systems. For a thorough discussion of issues related to the safety of systems containing software, including many heuristics for building them, see [54].

The area of formal methods is still developing and the research has so far been mostly academic and has not yet resolved all of the problems of their application in the industry. A variety of projects for critical systems have successfully applied some of the formal techniques, but usually only for fairly small amounts of software code. Work in this area has been most active in Europe, but has been receiving considerably more attention in North America in recent years.[52][51][50]

3.2 Standards and Certification

Standards for an industry are documents and accepted practices which represent technical criteria precise enough to be consistently used as rules and guidelines in order to ensure that the products and services, developed through the use of these standards, are fit for their purpose. In a negligence lawsuit, a civil court assesses these standards to determine the level of competence and the conduct of those involved in the manufacture of the product in question.[19] If the software industry does not regulate itself by developing appropriate standards and by certifying its practitioners, the number of lawsuits may grow significantly as the courts may lack sufficient guidelines from the industry so as to properly assess the achievement or failure to achieve a required objective standard of care. It should always be remembered that as far as the courts are concerned, the health, safety and welfare of the public is always the primary public policy consideration.[34]

In software engineering, there is still no consensus as to what constitutes the discipline's state-of-the-art and thus there is no single agreed-upon standard. More research is needed to determine what standards should contain and how specific they should be. Better cooperation between software practitioners, standard developers, academia and regulatory bodies is necessary in order to facilitate this research.[52][55] It has been argued that standards should prescribe goals, rather than particular methods, not just to ensure their validity over time, but also to hold the manufacturers accountable and responsible for selecting those methods that will achieve prescribed safety and reliability.[52]

There have been many software engineering standards developed over the last two decades by a number of international, national and industrial organizations. These include safety and reliability oriented standards, such as civil aviation standards (DO-178A), nuclear industry standards (IEC 880), NASA software assurance standards, military standards, the DoD (Defense System Software Development Standards), the ISO/IEC standards relating to programming languages, environments and system software interfaces, and many others.[19][57] In [21], a methodology for selecting and evaluating standards is proposed, and a selection of an appropriate mix of standards is advocated, as opposed to trying to develop an "ultimate" standard.

The certification of software engineers is an issue related to the development of a widely agreed-upon standard for the discipline. The term "software engineer" is not yet defined, and the engineering component of it is not quite clear. For example, in some US states, this term can only be used by registered professional engineers.[34] Whether or not this is the right practice is questionable, since not every professional engineer is trained in computer science. On the other hand, many of those who are developing software today have never even attended a university. There is a lot of debate on whether registration practices, such as licensing and certification, are desirable or even practical, and what impact such certification would have on education, insurance and the marketplace.[56]

The setting of software standards is a critical aspect of the industry, since the so-called "state-of-the-art" in software design will often represent the difference between safe and unsafe applications of software to economically sensitive or safety-sensitive situations. As stated earlier, the courts will be attuned to public safety and well-being considerations, and will move quickly to require standards of the software industry that correspond with this public protection objective.

3.3 Software Development Process Improvement

Standards, such as IEEE standards, are mostly just template documents for core processes in software engineering and for product documentation. They do not provide the mechanism for assessing organizational and individual capabilities.

The Capability Maturity Model (CMM), developed by the Software Engineering Institute (SEI), attempts to quantify the capability of an organization to consistently and predictably develop software products of high quality.[58] The model's levels, numbered 1-5, are referred to as initial, repeatable, defined, managed and optimized. Organizations at Level 1 develop software without formalized procedures, proper plans and cost estimates. This practice is usually described as "ad-hoc" development. Until the process is recognized, it cannot be repeated, so the organization must establish basic management practices, such as product assurance and configuration control in order to move to Level 2. Organizations at this level have achieved a stable process and are thus able to reasonably meet basic schedule and cost commitments, but the introduction of newer technologies is still problematic. Level 3 organization has defined the software development process to the extent that change can be managed. Organizations at this level have trained process specialists who collaborate closely with management and focus on process improvement. Level 4 indicates the introduction of comprehensive process measurment and analysis and the establishment of a process database that enables them to statistically manage their work and to set and achieve quality goals. Level 5 organization has achieved a solid foundation for continuous optimization of the software process and for meeting challenging productivity and quality goals. An organization at this last level has implemented defect prevention strategies, technology change management and process change management. It is thus possible to analyze trends in order to track the types of defects that have been encountered and to identify defects that can recur, to identify new technologies, experiment with them and incorporate the selected ones into the organization's standard software process, as well as to plan continuous process improvement. For a more detailed explanation see, for example, [58][1][60].

The position of SEI seems to be that the problems in software development lie in poor management practices, rather than in the inadequacy of technology.[1],[60] The opinions about the CMM model vary in the software industry, the main concern being the resource commitments required for the assessment and evaluation and for the subsequent progression to a higher level. There have been several reported successes in the industry achieved through the implementation of this model. These success stories do not validate the model, but they do seem to link the application of the CMM model to the production of higher quality software at lower cost, with a resulting improvement in a company's reputation.[58] As an example, the Software Engineering Division of the Hughes Aircraft Corporation was assessed by the SEI at Level 2 maturity in 1987. The recommended improvements were implemented, and two years later, a second assessement by SEI found Hughes at a strong Level 3. Hughes reported that the assessment itself cost approximately $45,000, and the improvement costs were $400,000, while the resulting annual savings were estimated at $2,000,000.[59] This and other success stories are also mentioned in [58], together with a critical look at the CMM model. The CMM model does not directly address expertise in a specific application domain, systems engineering, teamwork and concurrent engineering, nor does it advocate specific tools, methods and technologies, nor prescribe how to hire and motivate competent practitioners.[60]

The Personal Software Process (PSP) is a stuctured set of forms, standards and procedures, designed to be used by an individual developer to help manage his work and improve his skills. In simple terms, the PSP is the CMM model scaled down for an individual.[61]

The CMM and the PSP models thus address the company's and the individual's software process and, as a bi-product, offer a framework for assessment. Other software process models exist, such as the SPICE initiative, carried out under ISO/IEC that is trying to develop an international standard for software process assessment.

3.4 Section Summary

The expectations sometimes placed upon the software engineering discipline may be too high, given limited research funding. Software engineers operate under many constraints placed upon them by their clients and/or employers in the form of available resources and tight schedules, and so are constantly facing trade-off decisions in their work. It is not yet known (and it might never be) how to absolutely guarantee the safety and reliability of every software system, while preserving its intended functionality. Even though it is theoretically possible to prove the compliance of final code with stated requirements, it is not possible to formally prove, for every software system, that these requirements are the ones intended, and that the set of requirements, including safety requirements, is complete. We can increase our confidence in a requirements specification by using various requirements engineering techniques, even several of them in parallel, just as we can increase our confidence in the final product by optimizing the testing process, depending upon available resources and expertise.

It could be argued that with the progression of an organization from Level 1 of the CMM model to the higher levels, the organization's capability to produce a 'good' requirements specification should also increase. The introduction of new and untried research results and new technologies poses the least risk for an organization at Level 5 and the highest risk for a Level 1 organization. Such an argument is based more on common sense than on valid results. Valid results, in this case, could only be gained if controlled experiments were run, and that would be extremely costly, to say the least.

Fortunately, the law does not require infallibility, merely the exercise of reasonable care. What was or was not reasonable depends on all of the circumstances specific to each case that comes to court. Standards play an important role in evaluating the defendant's conduct, together with other factors. The development of a unified standard for software engineering could involve the CMM and the PSP models, coupled with safety, reliability and product documentation standards. If project types could be categorized according to their criticality, and if matching could be defined between these project categories, organization's and individual developer's capabilities and the types of methods, standards and documentation for each project category, then this matching could serve as an objective standard for software engineering. Risk assessment would provide the means for the definition of the matching among these three categories.

4 Applying Civil Law to Software Engineering

4.1 Review of Literature

This section is devoted to a discussion of the nature of the relationship between the principles of civil law and the discipline of software engineering.

Because the discipline of software engineering is still relatively new, most of the case examples that are used in the literature to illustrate the application of civil law to software try to draw analogies from civil engineering, medical and other areas. It is obvious from these case examples that the methods that the law employs to determine liability for damages of one party to another are not so easily transferred by analogy from a discipline with a well developed infrastructure, standards and state-of-the-art criteria such as civil engineering to one with a dynamic, fast-evolving body of knowledge and state-of-the-art such as software engineering.

In the case law, analogies have been made between software and, for example, electricity, music or a book in the courts' attempts to justify their judgments in particular cases. The problem is that, when considering software, analogies can be made to any or none of these, depending on circumstances, which also explains the widely differing conclusions made by the courts and the ongoing debate about the legal status of software, all of which affects the extent that the law will influence and regulate software development.

This section attempts to alert software engineers to the important legal issues that arise specifically in software development. Awareness of these issues can assist a software developer in preparing his risk analysis as well as in making informed decisions throughout each project's life-cycle in an effort to mitigate potential legal liability exposure.

4.1.1 How do standards relate to negligence?

Without widely used and accepted standards, there can be no consensus within the software industry as to minimum acceptable conduct that will facilitate the judicial assessment of negligence in a given situation.[34] If a unified standard for software development is ever accepted, it is most likely going to be a general one, leaving more room for the software engineer to exercise his professional judgement.[44] As long as standards of conduct remain ill-defined and not widely accepted for one reason or another, the ability of the courts and the software engineering profession to anticipate when liability may attach for particular actions will be hindered.[34]

On the other side, it should be noted that even if a well-defined, widely accepted standard is established for software engineering, such that a court will be able to objectively determine what a "reasonable software engineer" would or would not have done under certain circumstances, this standard will not be regarded as the "bible" insofar as a court's determination of professional liability is concerned.[34] It is but one of the several criteria of conduct that a court may have reference to in making its liability determination. An analogy would be the application of the National Building Code of Canada to a design prepared by a structural engineer. His adherence to the Code or his designing to a standard greater than or less than the standard required by the Code is not necessarily indicative of negligence, but it is one of many factors that a court will review to make its finding.[34] That is, compliance with trade organization standards does not by itself mean that reasonable care was used.[18] Courts look to the entirety of the actions of the individual in the context of that party's profession and all of its objective standards, and will also look to factors such as budgetory and schedule constraints imposed on the design process which may impact on the ability of the software engineer to meet the client's stated requirements.[34]

Some of the factors, considered by the courts are illustrated in [37]:

Claiming that a particular system or technique is not commonly used in the field or sector may not be effective in defending a negligence claim, if the use of such a system or method would have prevented or reduced injury or damage. Another aspect is resource availability: In a US case, a doctor in a remote area was not expected to achieve the same standard of care as he would in a hospital where he had access to all of the latest technology. In another lawsuit involving a bank, a customer requested a stop on payment but quoted the amount on the cheque incorrectly and the bank's system did not permit the payment to be traced by other information supplied by the customer. In this case, it was up to the bank to show that other search methods that could have found the payment regardless of the incorrect amount quoted by the customer, were not available.[37] Except in safety-critical situations, the courts will consider the budgetary and schedule constraints of a project in the sense that they will not expect the software developer to make use of each and every new research result in the software engineering area if it would have been impractical to do so in all of the circumstances of a given case.[37][34]

As long as there is no widely accepted standard, the courts will find it difficult to determine adherence to a particular standard of conduct expected of the "reasonable software engineer", and the judgments rendered in similar cases can differ widely as they will only be based on the arguments of selected experts and on the skills of the lawyers in each case.

If there is an accepted objective standard, and the licensing of software engineers is required, then whoever represents himself as a software engineer will be expected to perform at or above this standard, and so there might be more predictability in how courts will handle similar cases.

Professional licencing will not shield software engineers from liability. If, for example, a particular software engineer should have performed at or above the industry standard, but did not, thereby causing injury or damage, the existence of the standard will not protect him from liability.[34]

4.1.2 Company is liable for employees, but...

Standard civil law negligence principles can lead to liability for a company and for each of the employees involved in the development of software where third parties within the foreseeable risk of harm suffer loss or injury due to the failure of the company and its employees to achieve objective standards accepted in the industry. This is the case in jurisdictions where the test for negligence is the standard of reasonable care. Otherwise, the test will be according to statutory strict liability principles in jurisdictions where such legislation has been enacted.[34] Companies are vicariously responsible for the errors, omissions and negligent acts of their employees, carried out within the course and scope of their employment. However, claimants tend to look primarily to the companies and their insurers, rather than to the employees and team managers, in order to recover damages. It is usually difficult for a court to trace liability to an individual programmer, when software is developed by a team comprising requirements and design analysts, testing specialists and many others.[8]

It has been established in the case law that a company that uses systems in its operations is required to properly train it's employees and others who make use of these systems. This means that a company will not escape liability claiming that a particular system was available for use if it was used negligently by the company's employees.[37] Employees can be sued for negligence by their own employer, which usually happens when they make some extreme error or if the employees violate the company's policies or internal procedures.[8] Individual employees can also be held personally liable to third parties, where the third parties can trace their loss to that individual.

Officers of a company, including the CEO, may be held personally liable for incidents relating to safety of the company's products as well as for environmental damage if their own acts or omissions prove to be the direct cause of the loss which forms the subject of the claim.[37]

A company may limit the liability exposure of itself and its employees to its clients in contract. It cannot, however, so limit its liability or the liability of its employees to third parties who are outside of the contract. It is always advisable for every employee to thoroughly understand the terms of his own employment contract with the company as well as the contractual procedures of the company that he is working for.

4.1.3 Benefits and limits of a contract

A contract, if properly drafted between parties with equal bargaining power, can go a long way in limiting the liability of a software developer, both in terms of the time period in which a claim can be brought as well as with respect to the amount of monetary damages that can be recovered.[34] Such a contract, however, is usually very complex, and the assistance of legal counsel is necessary because many such contractual limitations are governed by statute, making some contract limitations or disclaimers unenforceable.[8] Properly worded exclusion clauses in contracts can protect, for example, against the implied warranty of fitness for a particular purpose as well as merchantability, provided these are referred to specifically and clearly. Remedies for repair and replacement of the product, often provided in contracts, do not exclude these implied conditions.[26]

Overly zealous software engineering companies may try to set up impregnable shields around themselves by using special disclaimers in their contracts, warning of the inherent dangers involved in using a particular software product. Nonwithstanding such disclaimers, the courts will often impose strict liability on such companies in situations where their product is inherently dangerous or rendered unreasonably dangerous due to an unexpected software error. Such contractual protections will prove to be illusory and will not reduce the exposure of the company, but may only succeed in compromising the company's professional reputation for quality, reliability and integrity.

Contractual liability in the context of software development usually arises when software does not conform to specifications or is not properly documented, is too slow or otherwise does not meet the needs of the user. This kind of liability can be avoided by limitations in the contract, but the client can still claim that the software was developed negligently.[8] If the contractual limitation of liability provisions between the parties do not explicitly exclude a separate claim in tort, then a plaintiff can sue under the contract and also file a concurrent claim for the tort of negligence.[34]

Misrepresentation of the system's capabilities to the client, such as "demos" that hide and cover up a system's imperfections, can result in a claim based on fradulent or negligent misrepresentation. For example, one US court has found that stating that a system can be modified "quickly and cheaply" can be considered to be a basis for a civil fraud lawsuit, if the company knew or should have known that there was no basis for making the statement in the first place.[8]

4.1.4 Safety first, contract and budget are secondary

A contract between a software developer and a client defines, among other things, the budget and time schedule for the project. If it turns out, during the course of the project, that the budget and the time shedule are inadequate to meet the safety requirements for the system being developed, then the software developer and his company would do well to keep in mind the predisposition of the courts to require a higher level of conduct where public health and safety might be compromised. If a particular budgetary constraint restricts the full measure of research necessary to produce a safe system, then the software development company and/or their employees or the project team must notify the client in writing as to their reservations regarding the ultimate safety and useability of the final product.[34] If the client refuses to commit more resources to the project, then the company must decline to produce a system whose release into the marketplace could foreseeably cause injury and damage, either to the client or to third parties.[34] The client could theoretically sue the company for breach of contract, but such a claim would likely not be successful, given the facts of this scenario. If, on the other hand, the software company continues with the contract under these circumstances, it risks the payment of huge damages, including punitive damages for losses incurred in the use of their system.

Therefore, in a trade-off situation during software development, choosing compliance with the contract over the safety of the final product is likely to result in substantial liability exposure to the software developer.

4.1.5 Changing state-of-the-art during project development

Some software development projects can take several years to complete. During the course of development, the state-of-the-art of software engineering may change in a material way. In such event, in order to avoid liability exposure, the software developer would be required to design according to the latest state-of-the-art developments which are known in the industry. Failure to do so, could result in a finding of liability, again with a potential for punitive damages. The software engineer is always required to meet the current established state-of-the-art. He should anticipate this in his contract, where it should be stated that if substantial changes in the discipline arise, such that substantially more effort (time) will be required to design according to the latest developments, the client will be required to finance those efforts. This is fair, since the client will receive the benefit of these additional efforts. If this is not anticipated in the contract, then the software engineer will likely not receive any compensation for his additional efforts.

4.1.6 Incomplete requirements specification - who's fault is it?

The safety and reliability of a software system depends, to a large extent, on the completeness of the requirements specification for that system. How familiar does a software engineer have to be with the application area for which he is developing software, and how does his knowledge of this area impact upon his potential liability? Software, unlike many other products, can be used to facilitate and automate work in almost any area. With so many potential areas, it is therefore conceivable that the client who is contracting for the development of a software system will have a better knowledge of the application area, including the related potential safety considerations, than the software developer himself.

In these cases, does the client share in the liability for an unsatisfactory result, or does the total burden of responsibility fall upon the software developer, as the individual with the purported expertise to design a safe and reliable product based upon the client's stated requirements? These questions are not easily answered as many of them relate to hypothetical situations which have not yet been the subject of judicial interpretation.

The courts will tend to attribute responsibility to a party who claims to have an area of expertise which he represents to others such that he knows they will rely upon him, and who renders advice in this area with the intention that it will be acted upon by the other party, and with the knowledge that if he does not act with due care, skill and diligence, that the other party may suffer economic or physical loss, injury or damage.[34]

As stated earlier, where a client has similar or even greater expertise than the software developer he employs to develop software for a particular application, it is possible for that client to be contributorily negligent in a proportion to be determined by the court. Any additinal consultants or experts hired by either the client or the software company in order to, for example, analyse the application area and collect the requirements, can also become contributorily negligent. This "split liability" can result in the court assigning responsibility in percentage terms, among the various parties to the design, marketing and use of a particular software product.[34]

An interesting case related to the issues presented above is described in [19]. In 1992, a computerized dispatching system of the London Ambulance Service (LAS) failed. The LAS, operating on a tight budget, had assigned the development of this system to a small software company with no experience in the development of safety-critical systems. The contracting company had no knowledge of the application area and relied upon the LAS to provide the necessary input. The system's safety requirements were never properly specified and implemented. The LAS knew nothing about safety-critical systems design, yet they were responsible for selecting a competent company. It would not be considered "reasonable" for LAS to understand that the development of the dispatching system required safety-critical methods and procedures. Such knowledge, however, would have enabled the LAS to select a competent company to design the system. The contracting company claimed that the client did not contract for the development of a safety-critical system and they, not being familiar with the application area, did not identify the system as being safety-critical. In [19] the author states that this particular point has not been directly addressed by a court, but the duty of the contractor to be aware of his own limitations and advise the client to seek expert advice from other sources has been identified.

It follows that a software contractor must be fully aware, not only of the limitations of his own expertise, but also as to the level of sophistication of the client who provides the software engineer with functional requirements for the project. A software designer is responsible for ascertaining all of the critical factors that could affect the client's system before proceeding with the design of a project. As stated, it is therefore critical that software development companies understand their own capabilities and limitations, and perform some form of risk analysis of the proposed system in order to ensure that their software development methods as well as their in-house expertise are adequate for the project in question.

4.1.7 Can software be an "unavoidably dangerous product"?

Some products which are available to the public are harmfull, yet strict liability does not apply to them. Such dangerous products commonly used in our society include some prescription drugs that may save a person's life, but have serious known side-effects. In these cases, not producing such a product would be a source of harm to the public interest (ie. loss of life) and therefore, producing the product, with its known side-effects is deemed to be acceptable conduct.

Some software developers may be tempted to evade strict liability by arguing that software falls into this category of unavoidably dangerous products. Such considerations, however, will not succeed as far as software is concerned. Catastrophic software system failures occur unexpectedly, because the developer of the system did not anticipate the particular situation that led to the failure. This means that the system's users could not have been warned of the impeding danger. As noted in [40], if the software manufacturer knew how and when the software would fail, he would likely have fixed the problem. In the case of a drug with known side-effects, such as a drug that saves life but causes the failure of an organ, it may not be known how to save life without causing the organ's failure. In these situations, the user is explicitly warned about the drug's side-effects, and accepts the risks, given the dire consequences of rejecting treatment with the drug. In the case of software, however, it is known in the industry how to fix software errors that are known, and a software developer will therefore be expected to do so.

The bottom line is if the safety of a software system cannot be ensured, either by proper design and implementation or by the use of proper warnings, then such a system should not be developed and marketed.

4.1.8 Should courts enforce strict liability for software?

If a software engineer employs his best efforts within the known standards of his discipline or according to the state-of-the-art that applies in the development of a software system, then in a court proceeding, he will be judged as having attained to a standard of reasonable care, providing the jurisdiction in which the work is being carried out has not enacted strict liability legislation. Strict liability legislation is designed to protect the public against potentially dangerous products, and as such, imposes liability on the software developer that is far beyond the objective standards of reasonable care and skill applied in negligence lawsuits.

Imposing strict product liability upon software developers might result in an even slower transfer of research from academia to the industry, hindering innovation in the software field, and so may not prove to be in the public interest.[40] A quite opposing view is presented in [42], where the author argues that the Northwest Airlines Flight 255 crash in 1987 should not have been attributed to pilot error, but instead to an inadequate human factors design, thus making the manufacturer strictly liable. It is stated in this paper that such a decision would have forced manufacturers to pay more attention to human factors design which might have resulted in the development of safety-critical systems that reduce the likelihood of human error.

The degree to which the courts' determination of liability on the part of software designers will affect the future development of software engineering is difficult to determine, and would require a separate study which is outside the scope of this paper.

4.1.9 Operator error or "unfriendly" interface?

What is or is not "user-friendly" has not been legally defined, even though there are some European directives that require (but don't define) "user friendliness".[15] The Flight 255 crash example, discussed in [42] is interesting, since the real reason for some accidents may well be innapropriate design of the interface, rather than operator error. The courts will often assign liability according to what expert testimony they accept in a given case. It is conceivable, that strict liability would be imposed on the manufacturer if the lack of attention to human factors in the design of the system is proven to have caused the error of the user of that system. Alternatively, given the appropriate facts and expert evidence, a court can choose to split liability between the user and the manufacturer, rather than assign fault to one or the other.

4.1.10 Product vs. service dilemma

The debate about whether software should be legally considered a service rendered to solve a problem or whether it is a product is ongoing.

If software is a service, then software engineers are analogous to lawyers or doctors in the sense that they do not guaranty the result of their work, only the use of reasonable care in the provision of the service. For example, doctors cannot guaranty that they will cure their patients. If, however, software is considered to be a product, then strict product liability can be applied to it, making the manufacturer strictly liable when software fails and causes injury. In [19] it is argued that software engineering is likely to fall into the category of a profession, where the practitioner should guaranty the safety of the end result (ie. product).

In the US, software embodied in a physical medium has been thought of as a transformation of intangible ideas into physical form and so is generally treated as a good, analogous to a book or a CD.[45] There have been attempts to define the nature of software to be "coded information" instead of a good, but as said in [45], such arguments are analogous to considering only a blank CD to be a good, thus confusing intellectual property with the physical medium containing that property. The courts have also held on several occasions that electricity was a product, even though it is intangible.[11] This 'product vs. service' issue has been more prominent in the area of medical expert systems, where the system provides advice to a physician. This particular problem is best discussed in [47]. In [43], a number of examples of inconsistency on the part of the courts in handling this issue is mentioned: Software licensed for use in a medical laboratory has been considered to be "goods"; another US court decided that a contract to develop customized software is a contract for services when no hardware is supplied, while yet another court held just the opposite.

4.1.11 Recovery of damages, punitive damages

When a lawsuit is initiated, the claimant's lawyer will typically use the "shotgun approach" and will sue everyone who could conceivably have had a hand in the development and marketing of the software system which results in loss or injury to others. If additional parties that could be blamed are discovered after the lawsuit has been initiated, they can be added as defendants to the original lawsuit, without the need to commence a new suit.

In the lawsuit, the claimant (plaintiff) will list all of his legal claims (causes of action), and may typically have several of them in one claim, including for example, negligence, breach of contract or a claim in strict liability. The requirement for strict liability to apply is the sale of a product and there is still no consensus as to whether software constitutes a product or a service. Therefore, in some cases, strict liability may not apply, and so the other liability theories may be pursued.[8]

The theory of both tort and contractual liability in civil law is that a claimant should never recover more than 100% of his loss. Therefore, if a court apportions liability among various defendants, the plaintiff should never succeed in recovering more than the loss which it can prove directly flows from the act complained of.[34] In some US jurisdictions, the concept of punitive or exemplary damages may be employed by the courts to "punish" the defendant(s) where their reckless conduct exposes others to substantial injury or damage.[34] These punitive damages are imposed on the defendant and paid to the plaintiff, and in those cases, the plaintiff does, in fact, recover more than his loss.

Punitive damages often arise in medical malpractice cases, but in some US states, legislation has been proposed to establish statutory damages for computer-related negligence as well.[8] These overly dramatic damage awards which are often far beyond the actual damages incurred by the plaintiff are almost never awarded in Canada or the UK.[34]

4.2 Case Examples

4.2.1 Diversified Graphics v. Groves; USA Court of Appeals, 1989

Diversified Graphics, Ltd. (DG) was a screen printer and apparel manufacturer. It hired Ernst & Whinney (EW), a firm engaged in public accounting and related areas, as consultants. EW were to select and implement an in-house computer data-processing system for DG. DG later initiated a lawsuit against J. Groves, chairman of EW, alleging negligence, breach of fiduciary duty and a breach of contract.

DG claimed that:

1. A long relationship with EW had resulted in the development of great trust and reliance on EW's services, given DG's lack of computer expertise. DG explicitly anticipated EW's superior knowledge in this area.
2. EW promised to find a "turnkey", fully operational system, the use of which would not require extensive employee training. Instead, DG received a system that did not meet its needs and was difficult to operate. The term "turnkey" was intended to mean a self-sufficient system, that would only require DG to "turn the key" to operate. DG claimed that it should not have had to hire programmers and extensively train its employees in order to make use of the system.
3. DG incurred considerable expense for the system's modification, employee training, additional staffing and consultations.

EW claimed that:

1. EW was hired for a limited purpose, that of evaluating DG's needs, and preparing a "Request for Proposal" to be distributed to potential vendors of hardware and software and to recommend a vendor. EW had no further involvement in the project, except for providing a representative who acted for DG in a very limited advisory role.
2. EW claims that any difficulties DG experienced with the system's implementation were a direct result of unwise decisions by DG's management.

The court decided that a consultant hired by a customer, who holds himself out as having special skill and knowledge to give advice relating to the products or services being considered, may be liable both in contract and for the tort of negligence to the customer for giving advice that falls below the standard of reasonable skill and care of the consultant's profession. EW argued that it should have been held to an ordinary, rather than a professional, standard of care. The jury decided in favour of DG and awarded \$150,000 on the negligence claim and assessed fault as 55% to EW and 45% to DG. The final award for DG was \$82,500 for negligence and \$50,000 for breach of fiduciary duty. It was held on appeal that DG was not entitled to two separate damage awards for the same harm, and only the negligence judgment was affirmed.

This case is interesting as it illustrates the application of a professional rather than an ordinary standard of care, based on applicable standards and the stated expertise which a consultant claimed to have. The decision reached in this lawsuit is likely to have an effect on the liability standards that could be applied to software consultants and vendors and so have implications for software development as a profession.[43] An analogy could be drawn to software engineering, where, in a similar situation, the court could measure the software engineer's behaviour against voluntarily imposed standards created by professional associations.[8]

4.2.2 Data Processing Services v. L.H. Smith Oil Corporation; Court of Appeals, Indiana, 1986

Data Processing Services (DPS) were a consulting company developing customized software applications. They entered into an oral agreement with Smith Oil (SO) to develop specific software for SO's IBM computer. After the payment of several billings, SO refused to continue payments because it found the system to be inoperable.

DPS initiated a lawsuit against SO, alleging breach of contract. SO counterclaimed for the tort of negligence. The judge ruled in favor of SO. The measure of damages for a breach of contract is the loss actually flowing from the breach. SO paid $28,000 to DPS for services in the development of the system. SO also had to hire an additional employee due to the failure of DPS's program to perform, for a cost of $9,000. The total judgment of $33,000 plus costs was found to correspond to the evidence presented at the trial. The court found that SO contracted for the development and delivery of a "program" by DPS. This program was considered to be a specially manufactured good and the transaction thus was considered to fall within the meaning of the Uniform Commercial Code (UCC).

DPS claimed the contract was not for a "good", but rather for a "service" and appealed the decision. The Court of Appeals noted that the transaction, unlike many cases reported in other jurisdictions, did not involve any hardware or pre-packaged software. DPS was hired to design, develop and implement a data processing system and thus to *act* to meet SO's specific needs. SO bargained for DPS's skill, knowledge and ability. The predominant factor in this transaction was considered to be the sale of services, the situation being analogous to a client seeking the advice of a lawyer, and so the UCC was not applicable. DPS was instead found to have breached the implied promise of having reasonable skill and ability to perform the job for which it contracted.

The Court of Appeals held that the principles of negligence, applicable to the established professions, applied with equal force to those who contract to develop computer software.

4.2.3 Hawaiian Telephone Co. v. Microform Data Systems, Inc.; USA Court of Appeals, 1987

In this case, a telephone company (HTC) contracted with the manufacturer of computerized directory systems (MDS) to manufacture and install a system. The software development was behind schedule and the system was not fully operational and could not perform critical functions by the date stated in the contract. It was later determined that MDS would require up to two years and at least three people working full-time to bring the system up to specifications. HT notified MDS that it was cancelling the contract because of the delays and MDS accepted the cancellation. HT then sued MDS for breach of contract. The court decided that the language used in the manufacturing and supply agreement providing for daily penalties for each day that MDS delayed the installation did not allow

MDS to take indefinite time to satisfy the contract. MDS was required to perform in a reasonable time. The contract agreement warranted the fitness of the system that MDS was to install, and stated that this warranty was to be instead of any liability of MDS for economic damages. This fact, however, did not prevent HTC from recovering economic damages ($600,872, plus costs) when MDS failed to deliver the system. These contractual provisions would only have applied if the system was delivered and installed as required by HTC. Because no system was delivered in the form contracted for, the breach was so fundamental that the exclusion of consequential damages by the contract was found to be unconscionable.

The language of the contractual warranties was as follows:

"Microform warrants that the equipment, when delivered and installed, will conform to the Equipment Specifications attached hereto and will be in good working order...In the event any item of equipment does not perform as expressly warranted, Microform's sole obligation shall be to make necessary repairs, adjustments or replacements at no additional charge to the Customer...The foregoing warranties are in lieu of all other warranties, express or limited, including without limitation implied warranties of merchantability and fitness, and are in lieu of all obligations or liabilities on the part of Microform for any claims, damages or expenses of any kind, whether made or suffered by Customer or any other person, including without limitation consequential damages even if Microform has been advised of the possibility thereof."

4.2.4 Ottawa Strong & Strong v. McLeod Bishop Systems; USA District Court, 1987

In this case, the issue was whether stating that a computer program can be quickly and cheaply modified was a misrepresentation and constituted a valid claim in fraud, providing that the statement was made with the knowledge that the modifications would be very difficult and require significant time and expense.

The court concluded that the claimant's position in the negotiation of these agreements was not unusual in the world of computer software purchasing and leasing. People who purchase or lease software application packages have to rely on the vendor's representations that the products can be adequately modified to fit their particular requirements. If the product cannot be modified as represented, it can be useless to the purchaser or require significant delay and expense in making modifications. The court thus concluded that a person who purchases software based on such misleading representations has a valid claim against the vendor based upon fradulent misrepresentation.

4.3 Section summary

Software is not completely infallible, yet it is becomming more widespread in our society every day. The number of lawsuits is likely to grow, and the judgments reached in these cases will have an effect on the development and marketing of software as well as on standards and certification of software developers. In this way, the law will influence the direction and the future of software engineering. The bottom line is that the courts in software engineering cases, as in cases pertaining to other professions, will typically favor a claimant where safety-critical or economically-critical factors are involved, and the required degree of reasonable foreseeablity can be established.

5 Implications

While ethical considerations alone should provide sufficient motivation for a software engineer to do his best in order to develop safe and reliable systems, in the real world, the law is often a necessary tool in deriving this same result.

The law operates as a system to order society. As such, it influences all aspects of human endeavour, including the development of software. The question is, to what extent should civil law control software development? Research related to software is relatively recent and the fields of computer science and software engineering are expanding quickly. If the courts favour software users and the general public over the software manufacturers and impose huge damage awards on software development companies, it could result in a curtailing of research and development in the software field, which ultimately may not be in the public interest.

Any decisions that the courts make in this area will have an impact on the number of new claims that will be initiated, on the amount of damages awarded and on insurance costs for companies and individuals in the business of developing software.

The discussion about the nature of software and whether it should be classified as a product or as a service continues. This issue is still not solved and many court decisions have instead considered the nature of the accompanying transaction or looked at the source of the injury in an attempt to properly classify software.

In some US states, tort reform was suggested as a way of reducing frivolous lawsuits and large punitive damage awards that are increasing the costs of insurance to the industry.[34] For an engineer, idemnification and limitation of liability are forms of tort reform that are meant to provide a level of liability comparable with the compensation that the engineer is receiving for his professional services.[10]

What is the future of software engineering in light of these developments? Despite continuous efforts in the software industry, there is still no unique, widely agreed-upon standard applicable to software development and to the conduct of software engineers. If a standard that is too stringent and inflexible is encoded, it would likely have a negative impact on progress in the discipline.

Software companies, faced with such a standard, might be reluctant to try new technologies and implement new research findings. A standard that prescribes individual procedures and methods could also enable software manufacturers to blame the injuries caused by their product on the standard and the body that enacted it.

If, on the other hand, a flexible standard is developed, one that specifies the safety and reliability goals to be achieved, instead of specific procedures to be followed, it would give software companies more room for experimentation, while, at the same time, making them aware of their final responsibility for the safety and reliability of any products they produce.

There are many unresolved issues relating to the development of software systems which make it difficult to develop a unified standard and even decide whether or not software practitioners should be certified as professionals. It is still not known how to guarantee the safety of all the critical systems that are needed and being developed.

Research suggests techniques and methods that, if properly employed, can guaranty compliance with stated requirements. Many heuristics for the development of safety-critical systems have also been suggested, and some sucessfully tried.[54]

There is still not enough research producing valid results relating to the selection and combining of these methods and heuristics for a software project, and to their implementation in the larger context of industrial software development in a way that would provide reasonable guaranties of safety and reliability.

5.1 Software Quality and Process Improvement

The legal issues reviewed in this paper suggest that the best way to avoid strict liability exposure is to make good quality products that are safe and reliable. The courts are predisposed to protect the public. If a product is judged to be "unreasonably dangerous", no contractual provisions purporting to exclude or limit liability will protect the software developer.

When developing a software project, a company has to necessarily make some tradeoffs. The deployment of best practices and cutting-edge methods and techniques might be desirable, but not always feasible because of insufficient budgets, inadequate expertise or a tight schedule.

Every software development company should strive to develop only safe and reliable products, not just to protect itself from legal liability. It should also seek to protect its reputation. In order to ascertain which projects to undertake in order to achieve this result, every software development company should perform a risk assessment before agreeing to take on a project.

Risk assessment should indicate whether a company and its employees have the level of capability and expertise necessary to develop the type of product under consideration, as well as indicating the types of processes, methods and standards to be used in the development process.(Figure 3.)

The quality and accuracy of such risk assessment will determine to what extent a company will mitigate both the risk of producing an unsafe product and the risks of legal liability, both in tort and in contract.

Having an assessment of a company's maturity, such as the assessment provided by SEI, can help a company to make a reasonable decision as to what types of projects it can safely undertake. For example, if a software development company is at Level 1 of the CMM model, it has no defined and repeatable software development processes. If such a company takes on the development of a system that is potentially safety-critical, claiming that it had previously developed similar systems and therefore has the required experience, it may incur substantial legal liability for negligent or fradulent misrepresentation of its expertise as well as potential liability for breach of contract in failing to produce a safe and reliable product which meets the client's stated requirements.

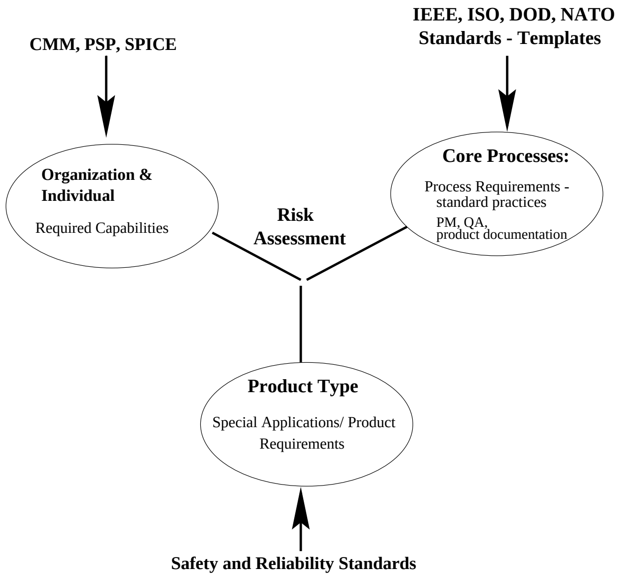

Figure 3: Risk Assessment in Software Development

The fact that a company previously developed similar systems which are operational, does not mean that the company is fit to develop such systems now, since being at Level 1 does not mean that it posesses repeatable processes, and so even if the process used to develop the previous system was the right one, there is no guaranty that the company will be able to repeat the process to develop a like system that would be safe.

Profit considerations should thus be weighed against liability and reputation considerations when proposing to undertake new projects.

5.2 Development of Product Liability Strategies

The control and prevention of liability for software systems produced should be seen as an integral part of the development and marketing process for computer products and services, and these efforts should be considered as an additional important factor in the software development process.

If, following the risk assessment, a company decides to proceed with a project, then special care should be devoted to drafting the contract with the client. The assistance of legal counsel is highly advisable, since a well writen contract can make a significant difference in helping to protect a software development company. Carefully drafted, explicit and understandable warnings are also important, so as to make every potential user aware of any potential dangers involved in using the product.

Depending on the resources of each individual company and on the criticality of the projects developed by it, an assessment of the organization's capability (CMM model) and an assessment of the employees' capabilities (PSP model) should be considered, along with seeking the advice of a legal expert so as to facilitate the development of specific product liability strategies for the company.

5.3 Individual Protection

Personal liability exposure is often related to the employee's reputation and career as well as potential financial loss.[10]

Because a software company is vicariously responsible for the errors, omissions and negligent acts of its employees, it is easy to lose sight of the fact that individual employees can be sued in tort for their negligence in designing and implementing software products. This exposure can often be limited contractually in the relationship with the client by inserting provisions within the contract limiting the liability of the company and its employees, officers and directors to an agreed amount in both contract and tort.[34]

While this will limit the exposure of the employees to the client, third party claims initiated by end users that are outside of the contractual relationship will still be able to pierce the corporate veil if the individual employee who performed the work can be ascertained.

Ultimately, the exposure of both the company and its employees to these third party claims should provide strong motivation to adhere to high standards of quality control and process improvement as perhaps the only realistic way of reducing the incidence and quantum of future liability claims.

6 Conclusion

All things considered, how will future software engineering development practices likely be affected by the application of civil law principles? In our consumer-based society, the consumer is becoming increasingly sophisticated. With this sophistication, comes an expectation of performance according to functional criteria established between the software developer and the client. When a level of performance equal to those expectations is not achieved, the client or other ultimate user will have recourse to their legal remedies.

The overriding theme that arises out of this consideration as to how the law impacts upon and will likely control the future direction of software development, is that software engineering design must be responsive to the manner in which the courts perceive its place in modern society. It must be sensitive to potential danger to the public interest and it must develop a strategy for dealing with the need for standards of acceptable conduct in the industry which will constitute the future definition of the "reasonable software engineer". As with most disciplines, generic standards of acceptable conduct will inevitably be developed, notwithstanding the competing jurisdictional battles for control within the industry. It is the development of these standards which will allow the software industry to evolve and operate in a controlled economic environment borne of the certainty that these standards will engender.

References

[1] **Software Product Liability**, Jody Armour, Watts S. Humphrey, Technical Report, CMU/SEI-93-TR-13, ESC-TR-93-190.

[2] **The Boyle Case: Its Significance to the Air Traffic Control Industry**, Richard H. Jones, Deputy Administrator of the FAA 1984-86.

[3] **Case Study: Darlington Nuclear Generating Station**, Dan Craigen, Susan Gerhart, Ted Ralston, IEEE Software, January 1994.

[4] **Case Study: Paris Metro Signaling System**, Susan Gerhart, Dan Craigen, Ted Ralston, IEEE Software, January 1994.

[5] **Case Study: Traffic Alert and Collision-Avoidance System**, Dan Craigen, Susan Gerhart, Ted Ralston, IEEE Software, January 1994.

[6] **Case Study: Multinet Gateway System**, Susan Gerhart, Dan Craigen, Ted Ralston, IEEE Software, January 1994.

[7] **Critical Task of Writting Dependable Software**, John Knight, Bev Littlewood, IEEE Software, January 1994.

[8] **Software Engineering Malpractice and its Avoidance**, Christopher J. Palermo, Christie, Parker & Hale, Pasadena, Calif. 91109-7068, 4/92, IEEE.

[9] **An Investigation of the Therac-25 Accidents**, Nancy G. Leveson, Clark S. Turner, IEEE Computer, July 1993.

[10] **Liability and Professional Issues Facing Engineers in Industry**, W. H. Copenhaver, Textile, Fiber, and Film Industry, 1992 Technical Conference.

[11] **The Key Issue in Reducing Risk of Liability with Expert Systems: Product or Service**, Susan L. Rick, Developing and Managing Intelligent System Projects, 1993 Conference.

[12] **Product Liability and Malpractice**, Joseph Rigler, Ronald White, IEEE Potentials, December 1990, Vol. 9, Issue 4.

[13] **Negligent? Who? Me?**, Robert Gaitskell, Engineering Management Journal, June 1993.

[14] **Potential Theories of Legal Liability for Defective Expert System Software**, Robert D. Sprague, Leslie G. Berkowitz, Managing Expert System Programs and Projects, 1990.

[15] **Safety-Critical Systems - Legal Liability**, Dai Davis, Computing and Control Engineering Journal, February 1994, Vol. 5, Issue 1.

[16] **How Liable Are You For Your Software?**, George B. Trubow, IEEE Software, July 1991, Vol. 8, Issue 4.

[17] **At What Point Does Liability Hit Home?**, Robert D. Sprague, IEEE Software, July 1995, Vol. 12, Issue 4.

[18] **Product Liability and the Plant Engineer: An Introduction**, Lawrence K. English, IEEE Transactions on Industry Applications, Vol. 31, No. 6, Nov/Dec 1995.

[19] **Professional Competence in Safety-Related Software Engineering**, J. J. Rowland, D. Rowland, Software Engineering Journal, March 1995, Vol. 10, Issue 2.

[20] **Can You Exclude Liability? The Unfair Contract Terms Act 1997**, Robert Gaitskell, Engineering Management Journal, October 1992, Vol. 2, Issue 5.

[21] **A Methodology for Evaluating, Comparing, and Selecting Software Safety and Reliability Standards**, Debra S. Herrmann, IEEE Aerospace and Electronic Systems Magazine, January 1996, Vol. 11, Issue 1.

[22] **Product Liability in the UK - Issues for Developers of Safety Critical Software**, Ranald Robertson, COMPASS'90: Proceedings of Fifth Annual Conference on Computer Assuarance, Systems Integrity, Software Safety and Process Security, p. 178-81, IEEE, MD, USA, June 1992.

[23] **"Strategy for Damage Control - Litigation & Presentation of Evidence in Cases of Industrial Accident and Losses"**, Robert Gaitskell, IEE Colloquium on Managing Safety: Losses, Liabilities, Litigation and Law (1993) (Digest No. 066), p. 5/1-4, IEE London, UK, March 1993.

[24] **A Model for Assessing the Liability of Seemingly Correct Software**, Jeffrey M. Voas, Larry K. Voas, Keith W. Miller, Proc. of the IASTED Conf. on Reliability, Quality Control and Risk Assessment, pp.32-35, Tysons Corner, VA, November 1992.

[25] **Computer Law - Acquiring and Protecting Information Technology**, Barry B. Sookman, Carswell, Toronto, Calgary, Vancouver, 1989 - 1995.

[26] **Computer Law - second edition**, Chris Reed, Blackstone Press Limited.

[27] **Computer-Related Agreements: A Practical Guide**, C. Ian Kyer, Mark J. Fecenko, Butterworths, Toronto, Vancouver.

[28] **The Law of Contract**, 8th edition, G. C. Cheshire, C. H. S. Fifoot, M. P. Furmston, Butterworths, London, 1972, ISBN 0-406-56527-9.

[29] **Canadian Tort Law**, 4th edition, Allen M. Linden, Canadian Legal Text Series, Butterworths Toronto and Vancouver, 1988, ISBN 0-409-80191-7.

[30] **Black's Law Dictionary**: Definitions of the Terms and Phrases of American and English Jurisprudence, Ancient and Modern with Guide to Pronunciation, 4th edition, Henry Campbell Black, West Publishing Co. 1951.

[31] **Product Liability Issues for the Computer Industry**, Scott W. Fleming, Canadian Bar Association (B.C. Branch) Computer Law Subsection, 1994.

[32] **Globalization of the Computer Industry: World Computer Law Congress, 1991**, April 1991, Westin Bonaventure, Los Angeles, USA.

[33] **The Law of Information Technology in Europe 1992: A Comparison with the USA**, A. P. Meijboom, C. Prins, (eds.), Deventer, The Netherlands; Boston: Kluwer Law and Taxation Publishers, 1991.

[34] **Legal Consultants:** Mark S. Dwor, B.A, LL.B, ACIARB, CHARB, Stuart B. Hankinson, BSc.(Hons.), LL.B, William E. Knutson, LL.B, LLM, Bryan S. Shapiro, B.A, LL.B; Vancouver, BC, Canada.

[35] **Information Technology Law**
Ian J. Lloyd, Butterworths, London, Dublin, Edinburgh, 1993.

[36] **An Introduction to Product Liability in the Computer Industry**
Clive Davies, Computer Law and Practice, Vol. 9, No. 3, 1993.

[37] **If Disaster Strikes - Could You Be Liable?**
John Mawhood, Richard Raysman, Computer Law and Practice, Vol. 10, No. 5, 1994.

[38] **Product Liability, Computer Software and Insurance Issues - The St Albans and Salvage Association Cases**
E. Susan Singleton, Computer Law and Practice, Vol. 10, No. 5, 1994.

[39] **Strict Product Liability and Computer Software**
L. Nancy Birnbaum, Computer/Law Journal, Vol.8, No.2, Spring 1988.

[40] **Computer Software Defects: Should Computer Software Manufacturers Be Held Strictly Liable For Computer Software Defects?**
Patric T. Miyaki, Computer and High Technology Law Journal, Vol.8, No.1, May 1992.

[41] **Developing a New Set of Liability Rules for a New Generation of Technology: Assessing Liability for Computer-Related Injuries in the Health Care Field**
James N. Godes, Computer Law Journal, Vol.7, No.4, Fall 1987.

[42] **Design-Induced Errors in Computer Systems**
Bruce Lathrop, Computer/Law Journal, Vol.10, No.1, Winter 1990.

[43] **Small Business Reliance on Computer Software: There Should Be Protection**
Julie Delluomo, Computer/Law Journal, Vol.10, No.4, December 1990.

[44] **Computer Malpractice: Two Alternatives to the Traditional "Professional Negligence" Standard**
Joseph Condo, Computer/Law Journal, Vol.11, No.2, April 1991.

[45] **Computer Software: Should the U.N. Convention on Contracts for the International Sale of Goods Apply? A Contextual Approach to the Question**
L. Scott Primak, Computer/Law Journal, Vol.11, No.2, April 1991.

[46] **Information Liability: New Interpretations for the Electronic Age**
Blodwen Tarter, Computer/Law Journal, Vol.11, No.4, December 1992.

[47] **Tort Liability for Artificial Intelligence and Expert Systems**
George S. Cole, Computer/Law Journal, Vol.10, No.2, April 1990.

[48] **A Spiral Model of Software Development and Enhancement**
Barry W. Boehm, IEEE Computer, May 1988.

[49] **Models of Software Evolution: Life Cycle and Process**
Walt Scacchi, SEI-CM-10-1.0, October 1987.

[50] **Seven More Myths of Formal Methods**
Jonathan P. Bowen, Michael G. Hinchey, PRG-TR-7-94, Oxford University Computing Lab. technical report.

[51] **An International Survey of Industrial Application of Formal Methods Volume 1: Purpose, Approach, Analysis, and Conclusions**
Dan Craigen, Susan Gerhart, Ted Ralston, U.S. Department of Commerce, March 1993.

[52] **Safety-Critical Systems, Formal Methods and Standards**
Jonathan Bowen, Victoria Stavridou, Oxford University Computing Laboratory technical report, December 1992.

[53] **Risks to the Public in Computers and Related Systems**
Peter G. Neumann, ACM SIGSOFT, Software Engineering Notes, Vol.17, No.1, January 1992.

[54] **SafeWare: System Safety and Computers**
Nancy G. Leveson, Addison-Wesley, 1995.

[55] **Second International Software Engineering Standards Symposium**
sponsored by IEEE Computer Society, Montreal, Quebec, August 21-25, 1995.

[56] National Software Council Forum: Should US Software Engineers be Licenced?
25-26 June 1996, St. Louis, Missouri, http://www.reengineer.org/forum

[57] **What is the future of Software Engineering Standards?**
Leonard L. Tripp, ACM SIGSOFT, Software Engineering Notes, Vol.17, No.1, January 1992.

[58] **SEI Capability Maturity Model's Impact on Contractors**
Hossein Saiedian, Richard Kuzara, IEEE Computer Magazine, Vol.28, Issue 1, January 1995.

[59] **Software Process Improvement at Hughes Aircraft**
Watts S. Humphrey, Terry R. Snyder, Ronald R. Willis, IEEE Software, July 1991.

[60] **Capability Maturity Model for Software, Version 1.1**
M. Paulk et al., CMU-SEI-TR-93-24, February 1993.

[61] **Why Should You Use a Personal Software Process?**
Watts S. Humprey, ACM SIGSOFT, Software Engineering Notes, Vol.20, No.3, July 1995.