Possible Implications of Design Decisions Based on Predictions

Magnus Larsson[1], Ivica Crnkovic[2]
[1]*ABB Corporate Research, 721 67 Västerås, Sweden*
magnus.larsson@mdh.se
[2]*Mälardalen University, Department of Computer Science and Engineering*
PO Box 883, 721 23 Västerås, Sweden
ivica.crnkovic@mdh.se, www.itd.mdh.se/~icc

Abstract. *Software systems and applications are increasingly constructed as assemblies of pre-existing components. This makes software development cheaper and faster, and results in more favorable preconditions for achieving higher quality. This approach, however, introduces several problems, most of them originating from the fact that pre-existing software components behave as black boxes. One problem is that it is difficult to analyze the properties of systems in which they are incorporated. To simplify the evaluation of system properties, different techniques have been developed to predict the behavior of systems on the basis of the properties of the constituent components. Because many cannot be formally specified, these techniques make use of statistical terms such as probability or mean value to express system properties. This paper discusses ethical aspects of the interpretation of such predictions. This problem is characteristic of many domains (data mining, safety-critical systems, etc.) but it is inherent in component-based software development.*

Keywords. Component-based development, predictability, ethics.

1. Introduction

Software, misunderstood, or even known to contain faults, is often deployed in computer systems and this may have serious consequences. This paper analyzes the ethical grounds on which decisions which might have such unintended results are made. In particular, we consider the prediction of quality attributes. Certain quality attributes, performance, for example, are difficult to determine even with thorough testing. A better approach then pure testing is to develop techniques for predicting such attributes. These techniques are not yet fully proven and decisions based on using the values they predict acquire an added moral aspect. If critical decisions are based on an unsound moral reasoning and only benefit the software company or developer, there is a risk that catastrophic consequences such as an incorrect execution time leading to the missing of a critical deadline may follow. In the business world, company decisions are commonly based on self-interest, the purpose of the company being to make money. From the business point of view, such a decision might be acceptable but there are other points of view and consequences which should be considered when making decisions.

The aim of this paper is to point out the risks of using predicted data when making software design decisions and to convey the importance of understanding the risks involved in using software. The paper discusses in particular the component-based development approach. In this approach software systems are built from software components already existing, possibly developed by third parties. The behavior of such a system is dependent on, inter alia, the properties of the components, but these properties are very often imprecisely defined and uncertainly validated.

The paper is organized as follows. Section 2 is a short survey of predictability in component-based software development and discusses the problems related to the predictability of quality attributes of a software system or a software component. Section 3 outlines software risks and the moral aspects of making decisions in which these risks are accepted. In section 4 the limitations to the prediction of software quality are considered, with an example in which the importance of a moral decision is discussed. Section 5 discusses several examples of decision importance. The paper concludes with a summary of the advantages and challenges of prediction-enabled technologies from an ethical point of view.

2. Overview of predictable component-based development

To improve the software development process, a new approach, component-based development (CBD) has been introduced during recent years and is now widely used in many engineering and different application domains. The major goals of CBD are [3]:

- To provide support for the development of systems as assemblies of components;
- To support the development of components as reusable entities;
- To facilitate system maintenance by replacing components and to enable system upgrading by customizing components.

The basic idea of CBD is to make possible the development of software systems in the same way as hardware systems are developed - by using components already available on the market. In a similar way as a car is assembled from different parts developed by subcontracting suppliers, CBD supports as far as possible, the separation of system development from software component development. This basic approach has many advantages but is not without problems. The main problem is the unpredictability of the total system behavior; in particular that of quality attributes, such as reliability, availability, performance, robustness, etc. Quality attributes are, in general, difficult to predict in software systems because of inadequate specifications and validations. This problem is exacerbated in component-based systems, since components are developed independently of systems and the system developer must rely on the manufacturers' specifications. These generally remain inadequate today as there are still no adequate formal means to express them. For this reason researchers continue to develop *prediction-enabled component-based technologies* which will be able to justifiably predict the behavior of component-based systems [4,5,7,10].

The prediction theories are often validated empirically, using standard statistical methods. The theories and results are similar to methods such as data mining or knowledge-management in other research areas such as medicine or economics. One of the main challenges in developing prediction-enabled component-based technologies is to obtain trustworthiness of systems and the components of which they are built. An underestimated trustworthiness may result in excessive precautionary measures and high development costs, while the consequences of overestimated trustworthiness may be the non-delivery of services expected by the customers and users, or such serious results as large economic losses or even a threat to the environment or human life.

As software becomes more complex, it is increasingly difficult to assess the quality of the functions required. It is even more difficult to verify system's quality attributes. It is not possible to test exhaustively large software systems since the potential number of possible states and execution paths increases exponentially. Quality attributes such as scalability, performance, memory consumption and reliability, are often not considered to be part of the functional requirements of a software system and are therefore often given inadequate attention during the design process. Purchasers are most often interested in a particular functionality when they buy software and, frequently, particular quality attributes are not specifically requested. The incorporation of these attributes is most likely taken for granted by the end customer. This reasoning implies, however, that functionality requirements are given higher priority than the non-functional quality requirements in the development process. Further, the lower priority given to quality attributes means that they may not be adequately considered until the system is implemented and tested.

The actual quality of a software product is determined in the testing phase and only then is appropriate action taken to achieve the quality required. Often, only the quality attributes which obviously do not meet the requirements are observed. Other attributes, not directly visible during the development process, but very important for the product's lifecycle, such as those related to reliability and safety issues, are not given the attention their importance warrants.

3. Morality and software risks

There are several reasons why the quality attributes are not given proper attention. *Ignorance* can be one, *high pressure* to keep costs down and to meet time-to-market requirements, another. As the inadequacy of the quality attributes might not be directly apparent, the line of least resistance is to "forget" them and leave the solution of any problems to the future.

If there is a risk that a particular software design decision could lead to events that might harm people, it is questionable if such a decision is morally sound. What are the moral standards relating to such decisions? Customers in other countries and cultures than those of the software developing company may have different concepts of what is right or wrong. A better understanding of whether the behavior of software products is acceptable or unacceptable can be attained from a study of ethics, the theory of morals, in relation to computing.

Figure 1, which shows ethical areas and issues related to computing, illustrates the complexity of the problem of making proper decisions. The most important areas in which computing and ethics are jointly concerned today are commerce, computer abuse, privacy, speech issues, social-justice issues, intellectual properties.

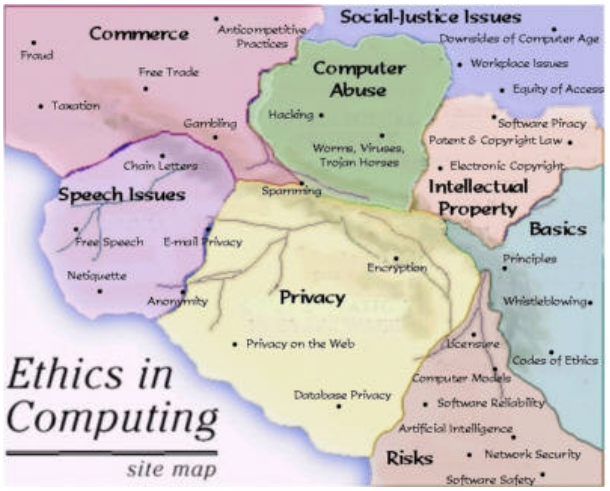

Figure 1. A landscape of ethical issues in computing [11]

Quality attributes are mainly associated with the area Risks (of computing) but also with the area Commerce. For instance, a software vendor might certify that the quality attribute requirement of software supplied has been met. This certification can be absolute or presented with a certain degree of confidence and it is of interest to the end user of the software to know the degree of confidence placed in the software components concerned. The customer must consider if the level of confidence quoted is acceptable for use in a particular system.

A force driving software developers is the belief that the software produced will solve the problems or satisfy the needs of its users. [2]. The quality attributes of software become critical in the sense that its behavior must be known before it is used in practice but testing cannot cover all possible eventualities. The software or system designer must therefore make certain decisions based on estimated quality attributes. In most cases, software technologies cannot prove and guarantee the correct behavior of software in all circumstances. For example, replacing a well-established technology with a new but not fully proven technology, knowing that it may cause injury or loss of life to human beings is morally unsound but on the other hand, there is a moral dilemma if, with the introduction of the same technology, it is possible to prevent damage or save the lives of human beings.

There are several bases for making a moral decision. Moral decisions can be based on different ethical theories [6,9]. These include:

- *Divine command theories*, i.e. obedience to some sacred text, or the manifested will of some undisputed power, e.g. the will of God.
- *Utilitarianism or Consequentialism*, i.e. the belief that the best action to take is that which procures the greatest happiness or good for the greatest number.
- *Virtue ethics*, i.e. the taking of action that maximizes accepted virtue and minimizes vices.
- *The ethics of duty or deontological*[1] *ethics*. The basing of a decision on the duty of the decision maker. Do your duty.
- *Ethical egoism*, the taking of action leading to ones' own benefit only. This is directly contrary to Utilitarianism. Ethical egoism maximizes the benefit of an action to one person instead of the greatest number.
- *The ethics of natural and human rights*. Decisions which acknowledge that all people are created with certain unalienable rights.

When a condition, for example happiness, richness or quality of life is considered from the utilitarian point of view, it is only the number of those on the positive side which is of importance. Virtue ethics is related to utilitarianism but considers the entire group. Let us take happiness as one example. Utilitarianism maximizes the number of happy people but accepts that there are some who could be very unhappy. A decision could be made which makes many people extremely happy but some very sad.

Virtue ethics involves both the bad and the good aspects. Making a decision which makes fewer people extremely happy might make more people less unhappy. One can illustrate this with

[1] 'Deon' = Latin for duty

an example (shown in Figure 2). At the cost of having some very unhappy people we can get more very happy people. On the contrary, trying to have fewer very happy people will result in fewer who are miserable. This example shows the differences between utilitarianism and virtue ethics. Note that this relation might not apply for other virtues, such as empathy or self-control.

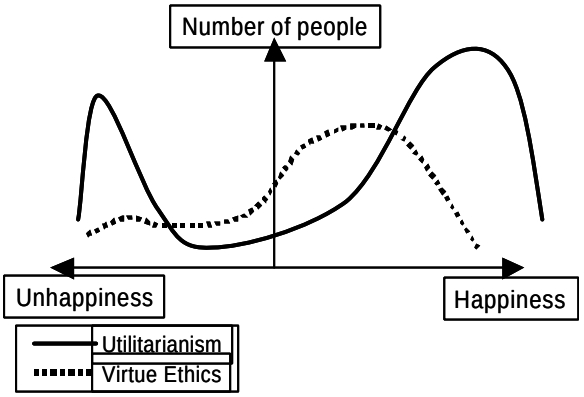

Figure 2. An example of the relation between utilitarianism and virtue ethics

Decision makers are frequently unaware of the ethical theory on which they base their decisions. They live in a culture with more or less fixed moral standards and make decisions in accordance with the ethical norm on which these moral standards are based. Awareness of the ethical basis of their decision-making and understanding the moral standards of the customer could reduce the number of incorrect decisions.

The risks associated with using software must also be understood and risk assessment should be a natural part of software development. Today, risk assessment in software development is more often than not, only the checking of the risks of not delivering the expected results on time. Software risks should not be confused with software process risks. For example, there is a *software risk* when software replaces hardware safety equipment such as an emergency brake. There is a *software process risk* when technology used in a project becomes obsolete during its development.

Apart from these underlying theories on which decisions are based, ethical norms or codes of ethics have been defined for use in software engineering specifically. There are the software engineering codes of ethics and professional practice issued by IEEE-CS and ACM [8] and The Ten Commandments of Computer Ethics [1]. These codes of ethics are defined in the form of a policy and state general principles about how the software developer should behave. For instance the first commandment from Computer Professionals for Social Responsibility (CPSR) [1]: *"Thou shalt not use a computer to harm other people"*, or the 3rd principle of the IEEE/ACM code of ethics [8], *"Software engineers shall ensure that their products and related modifications meet the highest professional standards possible*".

Neither of the two codes of ethics addresses the making of decisions relating to the use of software that might cause damage to property or injury to persons.

4. Prediction of quality attributes – examples of a moral dilemma

Better quality can be achieved by considering the quality attributes from the beginning of the development process. By having the architecture set rules for and limitations on the actual software produced in the architecture, it is possible to design the software so that certain quality attributes can be predicted with a certain degree of confidence in the design phase. This set of rules is determined by selection of a component-based technology. The predictability of quality attributes during the design phase permits explicit reasoning about quality and the determination of these attributes in advance. Predictions can very seldom be 100% accurate and there is usually a varying degree of confidence in the predictions. The question that then arises is: What if a software product is stated to have certain quality attributes and a customer makes decisions on that information in building a system which, if the quality requirements are not fulfilled, could harm the environment or, even worse, people?

We shall illustrate this with examples and cases.

4.1 A steel production control example

A control system monitors and controls a steel production mill. One of its tasks could be to control the pouring of molten steel into a cast. This task might be time-critical and if not fulfilled, molten steel might splash into the open, injuring personnel or damaging the plant. In this case, it is of great importance that the designer of the controller is certain that the task can be fulfilled within the specified period. In a component-based development the component providing that function should guarantee its

execution time. This information is necessary but not sufficient to guarantee the response time of the component in a running system.

The main problem is not to make predictions about the actual latency of a task but to prove that the predictions are correct. Having a theory of how to predict quality attributes, such as latency, in the development of a component-based software system is certainly advantageous, but the accuracy of the theory must be known. The verification of any theory can be formal or empirical, the formal verification actually proving its correctness and the empirical giving a degree of confidence in the theory. When the designer of a system has only a certain degree of confidence in the predictions of quality attributes, any decision to use the software, when and how, assumes a moral aspect. There are of course, other means of ensuring safety, even if the controller is not 100% proven. A decision to introduce safety equipment should be made if vital parts cannot be proven 100% reliable, or even if a proof indicates them to be 100% fail-safe since the confidence in the proof itself and the non-occurrence of unexpected events might be less than 100%.

A utilitarian approach to the decision to use or not to use certain control software in steel mill applications can lead to problems. The theory says that the decision should be made which procures the greatest happiness for the largest number of customers. The controller might deliver excellent functionality and contribute to producing the very best quality of steel, but there may be a known risk that the worst might happen, i.e. that hot steel could cause a serious, even fatal, accident. The decision can be reduced to choosing between the social and economic impact of the possible accident and the advantage of selling a system which satisfies many customers prepared to accept the risk.

4.2 Ariane 5 launcher case

On its maiden flight, the European Ariane 5 launcher crashed about 40 seconds after takeoff [12]. Fortunately there were no human casualties but the economic loss was half a billion dollars. The disaster was the result of a software error and particularly annoying was the fact that the error originated in a section of the software that was active unnecessarily after lift-off. The execution of a software component was intended to have ceased at 9 seconds before lift-off but the computation continued for 50 seconds. After lift-off this computation served no purpose but in the Ariane 5 flight it caused an exceptional situation which was not detected and the software and the launcher crashed. There has been much discussion regarding the root cause of such a banal error. The same software component functioned satisfactorily for Ariane 4 (although it included the same error) but was not tested for Ariane 5. What is interesting from an ethical point of view is the fact that due to budget reductions, there was a deliberate decision to omit certain tests. The quality was compromised for reasons of costs. Before every launch the quality manager decides if the launch is to be started or canceled. His final decision is based on the facts from different reports, his confidence in the reliability of the system, but also on pressures from different people who may be more ready to take the risks. The final decision is thus based on moral choices as well as on the technical facts.

4.3 Safety and probability

For safety-critical systems there is usually very little space left for uncertainty. The systems are designed so that their state is deterministic. For example, in real-time systems, it is assumed that the safety-critical services have available resources (memory, CPU, time). There are design methods that can achieve this determinism, but they require more resources. Finding a proper balance between safety requirements and cost constraints is the primary challenge in many software engineering domains. For example in the automotive industry the costs of electronics increase significantly year after year. To reduce production costs some car companies consider changing the principle of worst case execution time (WCET) to “most probable execution time”. An implication of this is that services (such as ABS-antilock braking system, or similar) function satisfactorily most of the time but that there is a risk of malfunction which might lead to catastrophic consequences. The companies calculate the probability of accidents, the probability of severe consequences and the resultant cost to the company (cost of possible compensation payments, bad image, and similar), and compare them with the costs of building systems which are 100% safe. Any decision to use a system less than 100% safe for economic reasons is morally questionable.

5. Summary and Conclusions

What are the consequences of a design decision? In many cases, the software engineer is more interested in meeting the technical challenge than in any non-technical consequences. Brooks stated already in the seventies that programming is fun and that the quality part of the work is not considered fun [2]. Every programmer wants to feel the sheer joy of making things.

The decision-making process in software engineering is often complicated by the fact that the designers are assigned multiple responsibilities. A software engineer is often responsible for requirements analysis, research, design, implementation, testing, error detection and correction, report and documentation writing and even project management. All these roles require that decisions be made, decisions which might have severe impact on the well-being of other human beings. This is not always clear and very often the designer have no time for engagement in philosophical discussions about the possible consequences of the decisions made.

In using the prediction-enabled technology, the objective is to determine the component behavior and predict the system behavior with a certain degree of accuracy. The positive aspect of this approach is that it provides explicit specifications and expresses them in statistical terms which indicate the degree to which they are correct. There is however a risk that the specifications may be mistakenly accepted as being absolutely correct and that the “high confidence numbers” guarantee that the system will work correctly.

Individuals may make decisions with the best of intentions but if a decision is incorrect, there is a risk that the consequences are a problematical situation which they cannot manage. They must instead be handled by a professional organization, making conscious and professional decisions based on policies known internally, as well as externally. Professionalism in these, and other, aspects provides a competitive advantage.

6. References

[1] Barquin, R., Computer Ethics Institute, *The Ten Commandments of Computer Ethics*,www.cpsr.org/program/ethics/cei.html, 2003.

[2] Brooks F. P., *The Mythical Man-Month - Essays On Software Engineering, 20th Anniversary Edition*, ISBN 0201835959, Addison-Wesley Longman, 1995.

[3] Crnkovic I., Larsson M., and Lüders F., "Implementation of a Software Engineering Course for Computer Science Students", In *Proceedings of 7th Asia-Pacific Software Engineering Conference (APSEC)*, 2000.

[4] Crnkovic I., Schmidt H., Stafford J., and Wallnau K. C., "4th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction", In *Software Engineering Notes*, volume 26, issue 6, pp. 33-40, 2001.

[5] Crnkovic I., Schmidt H., Stafford J., and Wallnau K. C., "5th Workshop on Component-Based Software Engineering: Benchmarks for Predictable Assembly", In *Software Engineering Notes*, volume 27, issue 5, 2002.

[6] Hinman, L. M., University of San Diego, *Lectures on Ethical Theory*, http://ethics.acusd.edu/video/Hinman/Theory/, 2001.

[7] Hissam S. A., Hudak J., Ivers J., Klein M., Larsson M., Moreno G. A., Northrop L., Plakosh D., Stafford J., Wallnau K. C., and Wood W., *Predictable Assembly of Substation Automation Systems: An Experience Report*, report CMU/SEI-2002-TR-031, Software Engineering Institute, Carnegie Mellon University, 2002.

[8] IEEE, IEEE-CS/ACM, *Software Engineering Code of Ethics and Professional Practice*, http://www.computer.org/tab/seprof/code.htm, 2003.

[9] Martin M. W. and Schinzinger R., *Ethics in Engineering*, ISBN 0-07-040849-1, McGraw-Hill, 1996.

[10] Moreno G. A., Hissam S. A., and Wallnau K. C., "Statistical Models for Empirical Component Properties and Assembly-Level Property Predictions: Toward Standard Labeling", In *Proceedings of 5th Workshop on component based software engineering*, 2002.

[11] NCSU, North Carolina State University, *Ethics in Computing*, http://legacy.eos.ncsu.edu/eos/info/computer_ethics/, 2003.

[12] Ariane 5 report, www.esrin.esa.it /htdocs/tidc/Press/Press96/ariane5rep.html