Sci Eng Ethics (2016) 22:417–434
DOI 10.1007/s11948-015-9665-x

ORIGINAL PAPER

Professional Ethics of Software Engineers: An Ethical Framework

Yotam Lurie[1] · Shlomo Mark[2]

Received: 10 January 2015 / Accepted: 29 May 2015 / Published online: 6 June 2015

Abstract The purpose of this article is to propose an ***ethical framework*** for software engineers that connects software developers' ethical responsibilities directly to their professional standards. The implementation of such an ***ethical framework*** can overcome the traditional dichotomy between professional skills and ethical skills, which plagues the engineering professions, by proposing an approach to the fundamental tasks of the practitioner, i.e., software development, in which the professional standards are intrinsically connected to the ethical responsibilities. In so doing, the ***ethical framework*** improves the practitioner's professionalism and ethics. We call this approach Ethical-Driven Software Development (EDSD), as an approach to software development. EDSD manifests the advantages of an ***ethical framework*** as an alternative to the all too familiar approach in professional ethics that advocates "stand-alone codes of ethics". We believe that one outcome of this synergy between professional and ethical skills is simply better engineers. Moreover, since there are often different software solutions, which the engineer can provide to an issue at stake, the ethical framework provides a guiding principle, within the process of software development, that helps the engineer evaluate the advantages and disadvantages of different software solutions. It does not and cannot affect the end-product in and of-itself. However, it can and should, make the software engineer more conscious and aware of the ethical ramifications of certain engineering decisions within the process.

✉ Yotam Lurie
yotam@som.bgu.ac.il

Shlomo Mark
MarkS@sce.ac.il

1 Department of Management, Ben-Gurion University of the Negev, P.O. Box 653, Beersheba 84105, Israel

2 Department of Software Engineering, SCE- Shamoon College of Engineering, Bialik/Basel Street, Beersheba 84100, Israel

Keywords Code of ethics · Professional ethics · Software engineer

Introduction

Familiar cases of faulty engineering, like the Ford Pinto gasoline tank (De George 1981), the Space Shuttle Challenger disaster caused by an O-ring seal that failed at liftoff (Boisjoly et al. 1989), or the 1997 collapse of the pedestrian bridge erected especially for the Israeli Maccabiah games (Schmemann 1997), have given rise to numerous discussions about engineering ethics (Dodig-Crnkovic and Feldt 2009). These cases pertain to the classical engineering branches of electrical, mechanical and civil engineering. However, as technology shapes and drives the world in which we live, software has become an integral element in many of our day to day activities. Indeed, the ubiquity of software is such that it can be found in almost every aspect of our lives, including, but not limited to our children's electronic toys, our smartphones, our HD plasma or LED televisions, and our cars' onboard computer systems, to name but a few examples. Software also plays an essential role in providing business information (BI) solutions for enterprise resource planning (ERP), enabling, organizing, and protecting the activities of large, medium and small organizations alike, including, among others, department stores, banks, government agencies and the military.

As a result, the impact of software errors and software bugs can be dramatic and can often have catastrophic consequences. In fact, it is increasingly being recognized that the source of many technological failures originates from software issues. Software plays an essential role in both our private and professional lives. Basically, people have become increasingly dependent in their daily lives (indeed, for their very survival) on computers and software, which together provide a service we cannot do without—for example, it is quite simply impossible to manually gather data and perform the calculations needed to process all the data we have without the proper software. In fact, as the foundation of any computer-based system or product, the software package is practically the "mind" of the product, without which computer systems would be rendered useless heaps of metal and plastic. Moreover, what this means is that the quality and dependability of the software package determines, in turn, the quality, usability, reliability, accuracy, serviceability, and safety of the product in which it is installed.

At the 1968 NATO convention the notion of ***Software Crisis*** was coined to express the gap between the ability to systematically (i.e., from an engineering perspective) develop a "quality software" product—correct, understandable, reliable, stable and verifiable—and the rapid expansion in computing power. The ***software crisis*** is reflected in irregular schedules, budget overruns, software that is inefficient or of low quality, non-compliance, and programs that are not delivered. It was found that 75 % of large software products sent to customers are considered failures in the sense that they are either not used or they do not meet customer requirements (Mullet 1999). The cost of repairing software bugs in the U.S. each year is estimated at $59.5 billion (NIST 2002). Published findings of the consulting firm "The Standish Group", which systematically and continuously surveys the field of software engineering and IT, show that over time, only one-third of software development projects end successfully and on time. About 15 % of projects fail almost immediately with the start of development, and almost half of the projects run into problems and end with significant deviations from both forecasted budgets and schedules. In addition, more than half of the projects that end successfully and are delivered to customers require significant post-delivery changes, since typically only about half of the required features purported to be supported by the software were actually functioning. The 2009 Standish report on the success of IT projects shows an increase over previous years in the incidence of IT project failure:

> This year's results show a marked decrease in project success rates, with 32 % of all projects succeeding which are delivered on time, on budget, with required features and functions" says Jim Johnson, chairman of The Standish Group, 44 % were challenged which are late, over budget, and/or with less than the required features and functions and 24 % failed which are cancelled prior to completion or delivered and never used. (StandishGroup n.d.)

Analyses of the software crisis have led researchers to associate it with a wide variety of causes ultimately rooted in the software company's abilities and organization. Among the typical drivers are the complexity of the software package, exaggerated expectations, poor planning, a lack of clear goals and objectives or objectives that are suddenly changed mid-project, changing requirements, unrealistic time and cost estimates, lack of cooperation, impaired communication and faulty teamwork, and lack of skill (Tilmann and Weinberger 2004). Over time, the software crisis has been dealt with, to varying degrees of success, using a range of approaches—the development of dedicated software tools, computer aided software engineering (CASE), frameworks, tools, programming paradigms, and programming languages; the implementation of improved modeling and design; changes to process and project management and software architecture; augmented software development and testing methodologies, principles and maintenance—but the inevitable conclusion is that there is no "silver bullet" solution (Brooks 1986) to the software crisis, and software development continues and grows, while facing future challenges, including various failures. Meanwhile, software engineers and producers must internalize the lessons learned from past failures and recognize where and how to implement the myriad of minor changes and improvements required to bridge the gap between increasing computation power and lagging software development.

This paper suggests the notion of an ***ethical framework***, adopted from the concept of a framework in software engineering, to deal with challenges, non-successes and failures in software development processes. Although it is a professional challenge, it is related to the engineer's interconnectedness with clients and other stakeholders. A viable approach to combatting the effects of the crisis entails increasing the ethical professionalism of software engineers through the use of a software development approach termed Ethical-Driven Software Development (EDSD). This approach constitutes an alternative to the classical model expressed by the traditional stand-alone engineering codes of ethics.

Engineering and Software Engineering

Software engineering is a young sub-discipline that strives to belong to the engineering professions. To fully understand the challenges beyond the software development process, it is relevant to re-contextualize software engineering within the broader context of the engineering professions. The American Engineers Council for Professional Development (ECPD) defines the engineering field as a "creative application of scientific principles…all as respects an intended function, economics of operation and safety to life and property".

Another definition, from the Oxford English Dictionary (OED), defines "engineering" as:

1. The branch of science and technology concerned with the design, building and use of engines, machines, and structures.
2. A field of study or activity concerned with modification or development in a particular area.

Basically, an engineer is an expert at creatively applying scientific principles in the design of products—i.e., structures, engines, machines, apparatuses, manufacturing processes, etc.—for specific functions. As such, the engineer is intimately familiar with the ins-and-outs of the "black box", or the software product being developed, and how to adapt that product for each specific environment where the software will be applied. Moreover, engineers, as opposed to technicians, are expected to have the creative ability to deal with new technological challenges in an innovative and creative manner.

As software engineering is a relative newcomer to the field of engineering, there is still much debate about whether it conforms to the classical definition of engineering. To resolve this issue, we have to answer the question: what is software engineering? This issue was addressed for the first time at the NATO convention in 1968 (Naur and Randell 1968),[1] where software engineering was defined as "…the establishment and use of sound engineering principles in order to obtain economical software that is reliable and works efficiently on real machines". In 1990, the IEEE-the Institute of Electrical Engineers, which is a professional association of engineers, (IEEE 1993) was required to address this question, and accordingly defined software engineering as:

> (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)

[1] The eight major engineering principles (Hambling 1995) are: "plan before building" Planning requires knowledge, experience and availability of resources, planning tools and finances/cost. If these are the constraints on planning, subsequently the quality of planning is predetermined. Assure compatibility; the idea is that all producers are working according to the same standards, Design testing procedures before building; check designs before commitment; configuration management; quality assurance and quality control—learn from mistakes (reuse); know where you're going.

In 1993, the Association for Computing Machinery (ACM) and the Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS), two leading international computer societies, established the IEEE-CS/ACM Joint Steering Committee for the Establishment of Software Engineering as a Profession.

Professionalism: Engineering and Software Engineering as Professions

The academic debate about the definition of a profession, and the attempt to identify the specific criteria an occupation must fulfill in order to be regarded as a profession, is a broad issue. Some have argued that just about any expert (knowledge-based practitioner) within any type of occupation (or practice) can be understood to be part of a professional practice (Kasher 2005). In the same vein, Parsons (1939) portrayed professionalism as a natural accompaniment to the forces of modernization, and as such, the existence of professionalism benefits both practitioners and the public. In contrast, a narrower approach to professionalism maintains that not all occupations are professional occupations. Professions, according to this narrower paradigm, are a special sub-set of occupations characterized by certain features or traits. Thus, an occupation must meet certain requirements to be considered a ***professional occupation.***

Possibly the earliest attempt to set criteria for professionalism was Abraham Flexner's infamous 1915 speech "Is social work a profession?" in which he proclaimed:

> Professions involve essentially intellectual operations with large individual responsibility; they derive their raw material from science and learning; this material they work up to a practical and definite end; they possess an educationally communicable technique; they tend to self-organization; they are becoming increasingly altruistic in motivation.

Ernest Greenwood (1957) cited five attributes of a profession: systematic theory, authority, community sanction, ethical code, and culture. Bayles (1982) identified three necessary features of a professional practice: extensive training that involves a significant intellectual component and that places one in a position to provide an important service to society. Within the realm of professional ethics, the professional ethics of engineering is an important sub-field. In this part of the paper we will first give an overview of the professional ethics of engineers.

In the conclusion of his 1996 paper, Michael Davis (1996) took a very critical position of software engineers. He asked two questions with respect to software engineers: first, as an occupation, is "software engineering" really ***engineering***? And second, is it, ethically speaking, a ***professional occupation***? He concluded by answering both questions in the negative. As he put it, "there is nothing […] to suggest that professional engineering is taking place". Davis' criteria (1999) for professionalism are normative rather than ontological. From the normative perspective, he argued that the activities of software engineers constitute neither an engineering occupation nor, ethically speaking, a professional occupation:

> The history of a profession is the history of how a certain occupation organized itself to hold its members to certain standards, beyond what law, market and morality demand. The history of a profession is the history of organizations, standards of competence, and standards of conduct.

The statement suggests that for a certain occupation to gain recognition as a profession, it is not sufficient that it operate like a guild—"granting membership in their associations (at a professional level) based on technical achievements" (p. 7). Davis specifies two, more demanding requirements: He argues that a professional occupation must be based on specific knowledge, which in software engineering today is commonly identified with a degree in software engineering. His other requirement, which is fundamental to the present discussion, is the commitment to use this knowledge in a certain way, that is, in a professional manner according to certain codes of conduct. Professionals have a special responsibility to adhere to a certain spirit, a calling which is beyond just following the rules (Davis 1996). This "commitment to use the knowledge according to certain codes…and adhere to a certain spirit" has been achieved by the use of a code of ethics.

Since Davis' (1996) article, much has changed in the world of software engineering. The Guide to the Software Engineering Body of Knowledge (SWEBOK) 2004 Version contributes three of the five components that, according to Ford and Gibbs (1996), characterize the engineering profession. The first is the knowledge component, and includes professional education and accreditation, certification or mandatory licensing, skill development and continuing professional education. The other two components are communal support via a professional society (a guild) and a code of ethics for the profession.[2]

Before discussing the central component of this paper, the code of ethics, we look briefly at the knowledge component. With respect to the **knowledge requirement**, in today's world, with novel programming languages such as Java and C# and powerful development tools and platforms such as Eclipse and .Net, code development, or coding, is no longer limited to a small cadre of computing professionals. In fact, the ability to "code" is global and within the reach of people almost everywhere. However, as coding skills proliferate in tandem with the breadth of human knowledge and initiative, the need for increasingly complex and powerful software products grows even faster, and the software creation community is confronted with problems related to the process of software creation, of which raw coding is but one relatively simple part (Sommerville 2004). Indeed, software development (process) is much more complicated than mere code writing. As a related set of activities and processes that are involved in developing and evolving a software system (Sommerville 2004), a software development (process) is defined as "Software is engineered" (Pressman 2010), which dictates the need for greater

[2] In 2004 they further clarified this definition (IEEE 2004): "Software engineering is about creating high-quality software in a systematic, controlled, and efficient manner. Consequently, there are important emphases on analysis and evaluation, specification, design, and evolution of software. In addition, there are issues related to management and quality, to novelty and creativity, to standards, to individual skills, and to teamwork and professional practice that play a vital role in software engineering".

expertise in the development of software and in the development of the relevant technology, methods and tools.

Software engineering is an occupation that builds practical skills based on the theoretical foundations of computer science together with practical expertise in the engineering methodologies aimed at the development of software systems. Genealogically, the field of software engineering has grown out of computer programmers/computer scientists who develop software, often without the ability to consider some of the other, equally important (technological, human and organizational) aspects of the system. The difference between software engineers and computer scientists is that while computer scientists suggest computational methods and solutions, software engineers design and implement the processes and methods that will engender quality and economic benefit.

Another condition required of a profession is the existence of a professional code of ethics. Identified in Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering SE2004, the condition indicates the importance of ethics in the profession of software engineering: "…graduates need to gain an understanding and appreciation of professional issues related to ethics and professional conduct, economics, and the societal needs" (IEEE 2009).

Most sets of codes of ethics in science and engineering are very similar in their ethical standards and topics. The Software Engineering Code of Ethics, jointly approved by the ACM and the IEEE-CS, has undergone major revisions over the years, up to the production of version 5.2, which was approved in 1999 and has been accepted as the standard for teaching and practicing software engineering (Gotterbarn 1999). The current code includes eight principles pertaining to the chief stakeholders: Public, Client and Employer, Product, Judgments, Management, Profession, Colleagues, and Self. The underlying guiding principle within the code in terms of all ethical decisions is a primary concern for the health, safety and welfare of the public. The preamble to this code summarizes that:

> Because of their roles in developing software systems, software engineers have significant opportunities to do good or cause harm, to enable others to do good or cause harm, or to influence others to do good or cause harm. To ensure, as much as possible, that their efforts will be used for good, software engineers must commit themselves to making software engineering a beneficial and respected profession. In accordance with that commitment, software engineers shall adhere to the following Code of Ethics and Professional Practice.

Summarizing the definitions of software engineering reviewed above, as a profession, it must abide by certain standards in terms of quality, processes, methods, and tools, among other issues that include scheduling, economics, risk and coding. The codes eight principles pertain to the engineer's relationship with the following stakeholders: Public, Client and Employer, Product, Judgement, Management, Profession, Colleagues and Self.

A follow-up to Davis' challenge (1999), this paper demonstrates how an ***ethical framework*** for software engineers can be interwoven into the life cycle of any software. By bridging the gap between the ethical and professional elements of the practice of software engineering, the implementation of an ethical framework will eventually produce engineers who work according to the professional standards of the occupation. Those standards, by definition, will entail certain ethical considerations.

The synergy Between Ethical and Professional Skills in Software Engineering

A software package is in many respects like a "black box" to most of its end users, who have no idea how it works or how to interact with it. Often the software package includes infrastructure, features and functions that are far beyond what the end user needs or comprehends. However, its black-box status notwithstanding, in practical terms the fundamental quality of the software product and of its design will determine whether it will fulfill the needs of the consumer. Our increasing **dependence** on computers and software packages in essential infrastructures, on the one hand, together with the profound **lack of understanding** (i.e., a knowledge gap) among most end users regarding the operation of the software package, on the other hand, entails a critical dependence of the end-user/consumer on the professionalism of the software experts. Due to this dependence, it makes sense to instill in, and promote among, software professionals a professional code of ethics. Because software engineers provide an essential service to end-users, their work resembles that of the other service professions in the sense that the service they provide is more important than the tangible good delivered, and consequently, there should be no separation between the software professional's practical and ethical skills. In other words, the ethical and professional skills associated with software engineering should synergies (Fig. 1).

Approaching the issue from a slightly different angle, the relationship between the software developer and the consumer can be characterized as asymmetrical in the sense that the end-user/consumer is dependent and relies on a software product

Ethical
skills
Profesional
skills

Fig. 1 Synergistic relationship between ethical and professional skills

s/he cannot understand. The software developer, who holds the keys to this black box, i.e., software package, must have sufficient knowledge, expertise and skills to control the product upon which the consumer is so dependent. Such a relationship, commonly termed a relationship of power, which in this case is between the software developer (the professional) and the end-user (the customer), necessarily requires ethical checks and balances to regulate it and assure that it is not abused.

More specifically, what appear to be primarily technical features of a software package, pertaining to the design of a software product—such as its quality, usability, reliability, accuracy, serviceability, and safety—in fact determine what the product can do and how it serves the end-user. In other words, these primarily *technical features* of a software product actually have important *ethical ramifications*. For example, consider that an engine computer installed in one's motor vehicle at times malfunctions, occasionally failing to warn the driver when the brake pedal (or the clutch pedal or the speed/distance sensor) is not functioning properly. The potential of this software system to return unreliable or inaccurate information represents risk to the customer and can result in serious harm. Ethically, we would say that a malfunctioning engine computer is the result of carelessness, and its installment in automobiles borders on reckless negligence due to the potential danger to the customer. The manufacturer of this engine computer can be blamed for being irresponsible, causing harm or simply acting unethically and unprofessionally. In addition, the engineer who failed to identify the flaw that led to the system's malfunction or to notify the proper authorities in the factory about the potential for malfunction and possibly rectify it, is indirectly (if not directly) responsible for any related injuries sustained by the users of the vehicle.

However, one need not talk about the risk and danger of harm in order to demonstrate that the technical features of software systems have an ethical dimension to them. In making choices in the design of information systems, software engineers impart the system with social and moral values. Once a system has been put into use, it affects its stakeholders to the extent in which it supports their values and expectations. Examples of stakeholders' values that may be either hindered or supported by technology are privacy, autonomy, openness, identity, human welfare, ownership and property, freedom from bias, and trust. For example, consider the requirements of a software package installed on the computer of a health care provider. Numerous issues arise, such as to what information about the availability and costs of treatments should the health provider have access when designing a treatment protocol for a patent? What information about prior visits to the clinic, e.g., personal medical data and data pertaining to one's family medical history, should be accessible to the health provider when treating a patient? These questions ultimately affect the acceptability of an information system. In this respect software systems affect the way people act and interact, for example, the types of treatment that will be offered to the patient, and thus the system ultimately has an effect on values and social norms.

The design methodologies of software engineers explicitly take ethical values into account in the design process of a software product in the form of ***soft constraints*** that are identified in the ***requirements phase*** of the engineers' software development. It is the designer's prerogative to choose the requirements that need to be considered during the design phase and where, due to the designer's professional experience, shortcuts can be implemented. Such decisions have ethical ramifications. For example, in the modeling and design of object-oriented software development, the professional designer has the ability to choose, based on his/her experience, which features or functionality requires a more intense design investment depending on the number of different points of view the architecture has to visualize or, in other words, the number of unified modeling language (UML) based design diagrams that must be applied. However, by representing values as informal soft constraints, designers run the risk of making the impact values have on the design too implicit, thus failing to sufficiently consider these factors. To properly account for ethical values in the design of a software system, they must be explicit and formally accounted for to have an identifiable and justifiable effect on the design. Indeed, ethical values can and should play an important role in the early requirements phase.

Insomuch as the products designed by software engineers affect the lives and livelihoods of the clients and end users of those products, software engineering as a profession has an obligation to society, which dictates that software engineers act ethically and professionally. The professional activities of engineers in general and software engineers in particular are defined in the life cycle of a software product and include the requirements, specifications, analysis, design, development, evaluation and maintenance of the software package. But the ethical dimension of the activity of software engineers has mostly been left implicit or, at most, relegated to stand-alone ethical codes. There are indeed several familiar codes of ethics for engineers in general and for software engineers in particular.

In contrast, this paper adopts a somewhat different direction. The idea is to sketch out an ethical framework within which software engineers make use of EDSD: Ethical–Driven Software Development. The concept of a framework has been defined as following: "a framework generally provides a skeletal abstraction of a solution to a number of problems that have similarities. […] A framework generally outlines the steps or phases that must be followed in the implementation of a solution without getting into the details of what activities are done in each phase" (Mnkandla 2009). In other words, the framework has to be a reusable, extendable and abstract set of basic components (objects) that characterizes the area of the problem. The user can adopt the object as-is, can change it, ignore it or add new objects to it.

By proposing an ***ethical framework***, which is inherently connected to the life cycle of a software product, rather than suggesting another "stand alone ethics code" for software engineers, the paper breaks down the dichotomy between the technical and ethical aspects of the profession, thereby explicitly linking the technical and ethical aspects of software engineering. Through the development of an ***ethical framework for software engineers***, the occupation moves one step closer to becoming a professional occupation. The goal of this paper, therefore, is to articulate the details of an ***ethical framework*** that will be woven into the life cycle of the software development process and become an integral part of the professional life of software engineers. This approach is unique in that the proposed ***ethical framework*** is inherently connected to the day-to-day professional activity of software engineers and the software development process, regardless of the development life cycle, e.g. Agile (scrum, XP, RUP etc.) iterative waterfall and others, which is adopted. Indeed, the efficacy of this framework depends on it being incorporated as part of the daily work routine of the software engineer. By adopting the concept "ethical framework" as a skeleton abstraction of a solution to integrate ethical concepts in the software development process we enable the user to tailor ethical objects for his/her project, as well as for her/his development process and team.

By way of analogy, much of the corporate world has turned in the last two decades from stand-alone voluntary ethics programs to a compliance based model. The FSGO (Federal Sentencing Guidelines for Organizations), put into effect in 1991 and further amended in 2004 (after Sarbanes–Oxley) and 2010 (after Dodd–Frank), established a framework for punishing corporations that broke the law, but more significant is that it created incentives for companies to self-police themselves. The Guidelines created a "carrot and stick" regime giving credit (in the form of punishment mitigation) to companies with effective compliance and ethics programs, and on the other hand severe penalties for companies that supposedly tolerated, encouraged or condoned improper behavior by not having effective compliance and ethics programs. More specifically, the FSGO created a detailed compliance framework based on a detailed and stringent "seven steps" program that organizations are expected to comply with.

Ethical-Driven Software Development (EDSD): An Ethical Framework

Engineers and ethicists are "programmed" to think in vastly different ways. Whereas ethicists strive for meaningful "codes of ethics" that are often inspirational but that lack standardized procedures, engineers approach the world via protocols and checklists. (Farrell 2002; Gaumnitz 2004) As a result, the engineering codes of ethics, which are not ingrained into the day-to-day practice of the profession, have only random and contingent effects on the day-to-day practice of engineering. All too often the professional code of ethics is conceived merely as an inspirational-normative ideal without practical bearings.

This matter bears upon a deeper philosophical point which should not be overlooked. David Hume (1711–1776) and much of modern moral philosophy contrast between factual judgments and value judgments in what has come to be known as the fact-value distinction, maintaining that these are two separate realms of judgment and that one cannot infer value judgments based on fact judgments and vice versa. The fact-value distinction is closely related to the naturalistic fallacy, according to which it is a fallacy to make claims about what ought to be on the basis of statements about what is. Insofar as this relates to the ways codes of ethics supposedly function, the profession has rules (protocols and standard procedures) of rational judgment regarding factual matters, but these are distinct from the value judgments expected of professional software engineers.

Contemporary moral philosophers (Foot 1978; MacIntyre 1981; Putnam 2002) have challenged these distinctions, arguing that facts and values are deeply entangled and that the dichotomy is based on a misunderstanding of the nature of fact. As an alternative to this typical separation between the ethical values of the profession and the practical protocols and checklists according to which the engineering profession operates, it is possible to connect the professional ethics of software engineers as a ***practical tool*** inherently linked to their day to day work. The EDSD approach allows for this by introducing an ethical framework into the development process. The notion of an ***ethical framework*** is a rich metaphor. Within the world of software development, a framework is a "sub-system design made up of a collection of abstract and concrete classes and the interfaces between them…frameworks are generic and extended to create a more specific application or subsystem" (Sommerville 2004). In other words, a framework can be presented as an engineering-aided tool to support or guide the developer under the rules of the development approach. On the other hand, within the world of applied ethics, an ***ethical framework*** denotes both the ethical limits/boundaries of a normative system as well as its situational constraints and directives.

By way of analogy, similar to how the World Medical Association has developed the ***Declaration of Helsinki*** that is the cornerstone for ethics in medical research involving human subjects, the ***ethical framework*** for software engineers should provide an ethical checklist that will be integral to the software's life cycle. The ***Declaration of Helsinki*** is an ethical guideline that requires the physician to follow a certain protocol in the research design. Similarly, the use of a practical tool, such as cue cards specifically designed for each stage of the development life cycle, can augment the awareness and commitment of the software engineer to think ethically about the stakeholders and risks involved. And when the client understands that the developer uses such tests, the tension that can potentially develop between client and developer is defused. In so doing, software engineers can provide the assurance needed that the development process abides by certain ethical standards.

More particularly, software life cycle models describe phases in the development of software and the order in which these phases are executed. There are many models, processes and methodologies for the software life cycle (waterfall, iterative, Agile etc.), and although each company (or project) adopts its own model and methodology, all are characterized by similar elements (artifacts). At each phase there are fundamental ethical requirements that must be met, which are derived from the specifics of the relevant stage. In all software development life cycles there are five major phases or, their incidence and terminology obviously depends on the specific rules of the life cycle methodology adopted: e.g., in waterfall methodology the phases appear one after the other as a sequence, while in Agile processes all five phases are concentrated to shorter time boxes depending on the methodology. (Thus, to look at two Agile processes: in SCRUM these appear as a tasks, while in the sprint, and in XP these appear as a user story in the iteration).

1. Initiation stage—in the **initiation phase,** the client articulates what s/he wants.
2. Setting requirements—in the **requirement phase,** the software engineer defines the client's needs. Much more than mere communication is at stake, the requirement phase entails a complex, interpretive process in which software engineers attempt to understand and match the needs of the client with the best

technical response. One must be aware that there are a variety of methods, depending on the lifecycle methodology, to achieve the goals of this phase.

3. Design stage—in the **design phase**, the clients "wants" that have been defined as "needs" receive a technical solution based on the technical abilities of the software engineer. Diagrams can be used to illustrate the functional operation of the system for the client. However, when obtaining the clients consent for a particular design, it is important to bear in mind that this is not ***informed consent***, which requires that the software engineer confirm the client indeed understands what is at stake, including matters pertaining to information security, information back up, etc.
4. Development stage—in the **Development phase**, the software engineer puts the technical solution into practice, after which the product enters the testing phase to be validated and verified, to determine where it succeeded and whether there were any failures in its operation.
5. **Testing and maintenance phase**—there are three types of maintenance: corrective, perfective and adaptive. At each phase, there are fundamental ethical considerations that must be taken into account, the resulting products, and deliverables that are required in the next phase in the life cycle. Requirements are translated into design. Code produced during Development is driven by the design. Testing verifies the deliverable of the Development phase in terms of whether it fulfills the requirements of that stage.

One must understand that the engineering of the development process is first and foremost connected to quality. Developing "quality software" is much more difficult than simple code development, especially when the definition of the term "software quality" is vague, equivocal and subject to different translations. In his book, Pressman (2010) preferred to define software quality by defining certain measures. Accordingly, software quality is measured by the software's degree of compliance with functional and performance requirements, development standards, and with all prerequisites required in professional software development. In other words, "how is the development process engineered?"

EDSD is a software engineering framework that is independent of the lifecycle development methodology, the specific project and the development team. Its goal is to obtain a quality engineering perspective by adopting ethical objects for all software development cycles, irrespective of specific development methodology and project. The ***ethical framework*** is a supportive software engineering tool that supports the EDSD approach and serves as a guide in the entire development process. There are several studies in order to determinate ethics interventions and rules for software developments, (Braude 2011; Miller 2011; Friedman 2012; Basart 2013) but in a study conducted in parallel to this work and continued and expanded these days (the results will be compiled and published shortly) in order to seek and explore the necessity of ethical interventions in a software development life cycle it was found (Amity 2014) that more than of 77.86 % the respondents (sample size over 200 people) agrees with (33.59 % strongly agree and 44.27 % agree) the statement that in a software development project, working in accordance to a defined set of rules and procedures is not enough to guarantee the success of the project, furthermore 76.17 % of the respondents agrees with (32.31 % strongly agree and 43.86 % agree) the statement that familiarity with the ramifications of every action taken within the development process is a necessary precondition for the process' success, And lack of awareness is one of the factors that hinder and harm the development process. Arguments that led us to the main idea beyond the EDSD approach, idea that emerged out of the claim that ethical processes are no different from any other development process and therefore adding ethical rules and procedures is not enough and in most cases software development decisions are made without considering the ethical effects and implications of the decision as part of the quality process. In other words, the status quo reflects a lack of awareness, which when present, will inevitably lead to proactive approaches that will ensure "quality software" and improve the quality of the final product. Therefore, all it takes is to raise the awareness and understanding of ethical interventions as a necessary component of the development process. Therefore, the ethical framework is a set of YES/NO ethical awareness questions that accompany each of the five phases in the development process, and of which every major stakeholder, including customer, administrator, team leader, designer, developer and quality member must be aware, before beginning any phase of software development. As such, the EDSD framework will not only increase the commitment of developers to produce high quality and dependable software, it may also substantially reduce the potential for tension developing between the stakeholders.

Practically speaking, the ethical framework suggests using EDSD index cards as a tool to aid and facilitate the development process, each phase of which has a specific set of such cards that could be color coded to distinguish between the phases. Each card contains only one yes/no ethical question, specific and focused,

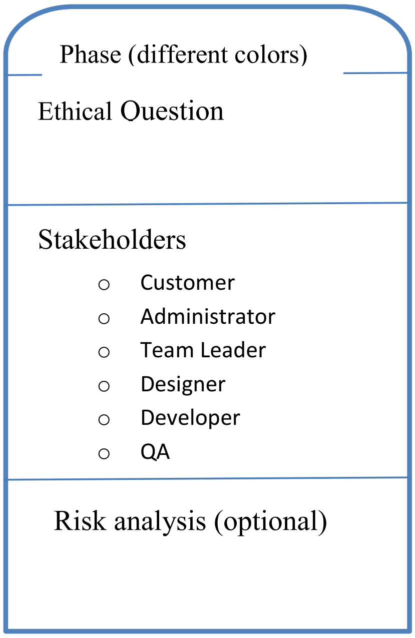

Fig. 2 Ethically-Driven Software Development Ethically—(EDSD) Index Card

relevant to the specific development phase at hand that all relevant stakeholders are asked to answer. In general, the questions raise stakeholder awareness about the ethical considerations and implications relevant to each development phase.

Two further comments about this matter. First, the collection of EDSD Index Cards is not to be taken as a final and closed collection of cards. The specific collection EDSD Index Cards used is dependent on: the development process, development life cycle, development team, development leader as well as complexity of the requirements and the software and as we learn about new needs, limitations and fuzziness in the development process, so do the specific ethical questions that compose the ethical framework develops. Also, second, one need not expect the EDSD Cards to provide the software engineers with answers, rather the cards require the software engineer to halt and reflect. They serve as a tool for raising awareness to the ethical ramifications of the process. The selected questions must be presented to the development team so that they are understandable by everyone, and everyone is able to answer the questions. The EDSD allows for the creation and adjustment of tailor made ethical questions (Fig. 2).

The EDSD Yes or No Questionnaire

1. **Initiation phase**
    1. Are there criteria for adjusting the organization to examine the initiation?
    2. Are there criteria for approval of product initiation, which should be particularly demanding for fields in which the organization or developer is not an expert?
    3. Are there criteria accordingly to which the supplier agrees to offer a product that is being developed for a new field?
    4. Is the supplier knowledgeable enough about the resources with which s/he must work?
    5. Is there a mechanism to ensure that supplier agreement pertaining to the project at hand was obtained only after determining whether the organization or developer has the ability to carry out the project?
    6. Has it been determined who makes decisions on behalf of the client?
    7. Has it been determined who approves the budgetary flexibility for project management?
    8. Has it been determined who is the authority who deals with abnormalities in the scope of work?

2. **Requirements phase**
    1. Has it been determined who defines the requirements?
    2. Are there mechanisms in place to determine the path of action when it is discovered that one or more of the functional requirements cannot be achieved?

3. Are there mechanisms in place to determine the path of action when the customer insists?
4. Is there a procedure in place to define the course of action if, during the planning requirements phases, it is discovered that fulfilling the customer's demand will lead to negative consequences and possibly contradictory and illegal outcomes?
5. Is there a mechanism in place to prioritize requirements?

3. **Design phase**

   1. Are conditions set to determine the level of commitment to developing a reuse approach?
   2. Are conditions set to determine the level of commitment to the development of all standards methodology—for example, Design Pattern in object-oriented analysis and design (OOAD)?
   3. Would it be easy to orient in the design diagrams?
   4. Do you think you should use more design diagrams?
   5. Can someone who is not an expert orientate the diagrams?
   6. Are the diagrams clear enough?
   7. Is there sufficient attention to details?
   8. Is the level of detail sufficient?

4. **Development phase**

   1. Are conditions set to select the encoding language?
   2. Are conditions in place for integrating young and less experienced developers?
   3. Are conditions in place for integrating the advanced capabilities of the code management? Timings? Environmental parameters?
   4. Is it easy to orientate the code you wrote?
   5. Do the comments have sufficient clarity?
   6. Is the code compatible with the design diagrams?

5. **Testing and verification phase**

   1. Has the level of allowed errors been determined before there is a transfer of the product to acceptance testing been set?
   2. Is a mechanism in place that distinguishes between a quality product versus a correct product?
   3. Is a mechanism in place that determines whether the required tests were performed?
   4. Is there a set of mandatory tests?
   5. Are the metrics for determining the level of required tester training defined?

Conclusion

From the perspective of professional software engineering, EDSD is a novel and practical approach to software development that bridges the gap and creates a link between ethical and professional skills, thus proposing a process of software development that is more transparent and brings to better awareness of the risks and limitations in the process of software development. Taking ethical questions into consideration thus becomes part of the process required for the development of "quality software" products. The ethical framework sketched out is very rudimentary and can be further adapted and expanded within the context of different software development protocols.

Philosophically, through the specific case of professional ethics in software engineering, a broader and more significant argument has been made regarding ethics in the professions. This approach is a ***process*** centered approach to ethics in the professions, which we demonstrated has advantages over principle and rule based approaches to ethics, as manifest in formal codes of ethics. Moreover, by integrating the ethical considerations with the professional considerations a more robust conception and manifestation of professionalization emerges.

References

Amity, E. (2014). *Agile and proffesional ethics*. M.Sc. Thesis in software engineering. SCE - Shamoon College of engineering, Israel.

Basart, J. M. (2013). Engineering ethics beyond engineers' ethics. *Science and Engineering Ethics, 19*(1), 179–187.

Bayles, M. D. (1982). *Professional ethics*. Belmont, CA: Wadsworth Pub. Co.

Boisjoly, R. P., Curtis, F. E., & Mellican, E. (1989). Roger Boisjoly and the challenger disaster: The ethical dimensions. *Journal of Business Ethics, 8*(4), 217–230.

Braude, E. J. (2011). *Software engineering: Modern approaches*. NewYork: Wiley.

Brooks, F. P. (1986). No silver bullet—Essence and accident in software engineering. In *Proceedings of the IFIP Tenth world computing conference* (pp. 1069–1076).

Davis, M. (1996). Defining 'engineer': How to do it and why it matters. *Journal of Engineering Education, 85*, 97–101.

Davis, M. (1999). Professional responsibility: Just following the rules? *Business and Professional Ethics Journal, 18*, 65–87.

De George, R. T. (1981). Ethical responsibilities of engineers in large organizations: The Pinto case. *Business and Professional Ethics Journal, 1*(1), 1–14.

Dodig-Crnkovic, G., & Feldt, R. (2009). Professional and ethical issues of software engineering curriculum applied in swedish academic context. In *HAoSE 2009 first workshop on human aspects of software engineering*. Orlando, Florida.

Farrell, B. C. (2002). Codes of ethics: Their evolution, development and other controversies. *Journal of Management Development, 21*(2), 152–163.

Foot, P. (1978). *Virtues and VICES and other essays in moral philosophy*. Berkeley and Oxford: University of California Press and Blackwell.

Ford, G., & Gibbs, N. (1996). *A mature profession of software engineering*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Friedman, B. (2012). The envisioning cards: A toolkit for catalyzing humanistic and technical imaginations. In *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM.

Gaumnitz, B. R. (2004). A classification scheme for codes of business ethics. *Journal of Business Ethics, 49*(4), 329–335.

Gotterbarn, D. M. (1999). Computer society and ACM approve software engineering code of ethics. Retrieved from http://www.computer.org/cms/Computer.org/Publications/code-of-ethics.pdf

Greenwood, E. (1957). Attributes of a profession. *Social Work, 2*, 44–55.

Hambling, B. (1995). *Managing software quality*. New York: McGraw Hill.

IEEE. (1993). IEEE standards collection. In *IEEE standard glossary of software engineering terminology*. IEEE Computer Society.

IEEE. (2004). *Software engineering 2004 curriculum guidelines for undergraduate degree programs in software engineering*. Washington, DC: IEEE computer society.

IEEE. (2009). *Curriculum guidelines for undergraduate degree programs in software engineering*. IEEE Computer Society. Retrieved from http://sites.computer.org/ccsepp15

Kasher, A. (2005). Professional ethics and collective professional autonomy. *Ethical Perspectives, 11*(1), 67–98.

MacIntyre, A. (1981). *After virtue: A study in moral theory*. Notre Dame, IN: University of Notre Dame Press.

Miller, K. W. (2011). Moral responsibility for computing artifacts: The rules. *IT Professional, 13*(3), 57–59.

Mnkandla, E. (2009). About software engineering frameworks and methodologies. In *AFRICON, 2009. AFRICON'09*. IEEE.

Mullet, D. (1999). The software crisis. *Benchmarks Online—A monthly publication of Academic Computing Services of the University of North Texas Computing Center*, *2*(7). https://www.unt.edu/benchmarks/archives/1999/july99/crisis.htm.

Naur, P. & Randell D. B. (1968). *Software engineering: Report of a conference sponsored by the NATO Science Committee*. Garmisch, Germany: The first NATO Software Engineering Conference in 1968 Garmisch, Germany. Retrieved October 7–11, 1968

NIST. (2002). *New Release of June 28, 2002*. The National Institute of Standards and Technology.

Parsons, T. (1939). The professions and social structure. In T. Parsons (Ed.), *Essays in sociological theory* (pp. 34–49). New York: The Free Press.

Pressman, R. (2010). *Software engineering: A practitioner's approach* (7th ed.). New York: McGraw Hill.

Putnam, H. (2002). *The collapse of the fact/value dichotomy and other essays*. Cambridge: Harvard University Press.

Schmemann, S. (1997). *2 Die at games in Israel as bridge collapses*. Retrieved from http://www.nytimes.com/1997/07/15/world/2-die-at-games-in-israel-as-bridge-collapses.html

Sommerville, I. (2004). *Software engineering, international computer science series* (7th ed.). Boston: Addison Wesley, Pearson Education.

StandishGroup. (n.d.). Retrieved from http://www.standishgroup.com/newsroom/chaos_2009.php

Tilmann, G., & Weinberger, J. (2004). Technology never fails, but project can. *Baseline, 1*(26), 28.