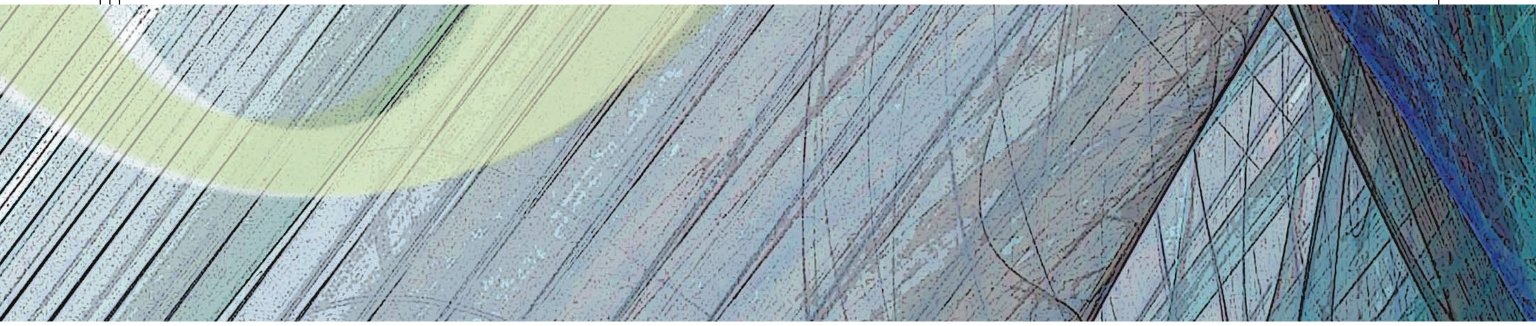
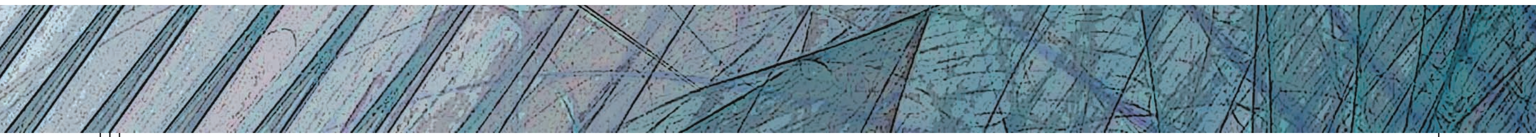


PROFESSIONAL AND ETHICAL DILEMMAS IN SOFTWARE ENGINEERING



Brian Berenbach, *Siemens Corporate Research*

Manfred Broy, *Technical University of Munich*

The authors identify, categorize, and name nine specific ethical and professional dilemmas in software engineering, placing them in the context of the IEEE code of conduct, with the hope that giving such behavior a name will increase awareness and decrease the frequency with which these dilemmas occur.

Humans have been engineering things for hundreds of years, and for all that time they have faced essentially the same ethical challenges we have outlined elsewhere. So what makes software engineering significantly different? Software engineering is a discipline with which many people are unfamiliar and where issues and problems are harder to spot in advance. Therefore, people must trust software engineering experts even more than experts in other fields of engineering.

Software engineers often engage in unprofessional or unethical behavior without realizing it.[1] In ethics courses, or through professional association codes, practitioners might be warned of situations or dilemmas that could eventually lead to unethical behavior, such as "Avoid harm to others."

A joint ACM/IEEE-CS task force has created a code of ethics for addressing issues of unethical behavior (www.nspe.org/Ethics/index.html). But too often the focus is on headline scenarios and not on the (initially) mundane situations that abound in our profession. Further, ethical training might not be given the needed emphasis during undergraduate education.[2-4]

Rather than an isolated ethical lapse, what typically makes the headlines is the result of a sequence of related ethical lapses. When such scenarios cascade, there tends to be a magnification effect.

An ethical dilemma occurs in software engineering when the professional must make a choice between competing values, such as personal versus professional. For example, a sales manager might sign a contract to deliver a software product knowing, or having been advised, that the product will take longer to deliver than the promised date. The sales manager's dilemma might be that his employer is under pressure to meet a financial target, or there might be job-related consequences.

In response, the project manager creates an unrealistic project schedule, which lead development staff choose to defer to so that they can achieve political or organizational goals. A chain of unethical or unprofessional behaviors could then take place that eventually leads to massive penalty payments, lawsuits, layoffs, or even bankruptcy. Yet none of the players in this process will admit to or recognize that their behavior was unethical or unprofessional.

CATEGORIZING DILEMMAS

To shine some light on these kinds of subtle but nonetheless unethical or unprofessional situations, we have given each dilemma a name. Readers might disagree with our choice of labels, but we doubt they will disagree that the behavior is inappropriate.

This list is not and cannot be comprehensive. We cover the most significant instances of ethical dilemmas. All those we describe involve common occurrences. We cannot be sure that simply *naming* them will solve anything, but it will help us discuss them.

Keep in mind that not every wrong behavior is unethical. If people do not know better and behave wrongly, they are not acting unethically. It certainly is unethical, however, for people to make decisions when they know they lack the knowledge needed to make sound professional decisions.

The term *ethical behavior* refers to how an individual or an organization ensures that all its decisions, actions, and stakeholder interactions conform to the individual's or organization's moral and professional principles. These principles should support all applicable laws and regulations and are the foundation for the individual's or organization's culture and values. They define right from wrong.

Typically, incompetence, unprofessional behavior, personal misconduct, mismanagement or, more commonly, a seemingly inconsequential chain of small ethical or professional lapses brings about the situations that make headlines.

For example, if a project is running late, the project manager might be tempted to cut short the requirements definition phase, hoping to make up for some lost time. In order to get the product out the door, developers base their testing not on the requirements, but on developer descriptions of how their code will work. The team then delivers the result to the customer with possibly catastrophic consequences, such as an unusable product, contract cancellation, or lawsuits.

This behavior is shortsighted at best and certainly unprofessional. Wrong decisions lead to bad results. If a person who should know better makes wrong decisions, and if personal interests motivated those decisions, the behavior becomes unethical.

When students enroll in introductory ethics courses, they learn about clear and extreme situations. This environment makes it relatively easy to distinguish when behavior crosses the line or is unethical. However, real life is not so simple, and the following dilemmas come from scenarios that occur all too frequently.

Mission impossible

This dilemma occurs when an individual is asked to create or accept a schedule that is obviously impossible to meet. Because of perceived pressure, or for other reasons, the person creates or accepts the schedule knowing it is unrealistic.

The mea culpa dilemma occurs when staff members must deliver a product that still lacks key functionality or has known software defects.

The consequences of this lapse in judgment can range from loss of qualified staff to significant loss of revenue. Overwork and burnout cause loss of staff. Loss of revenue can derive from the premature announcement of a product's availability, which then reaches the marketplace later than anticipated. Meanwhile, customers stop buying the current product in anticipation of the new product's arrival.

Mea culpa

This dilemma occurs when staff members must deliver a product that still lacks key functionality or has known software defects. The market's anticipation can create pressure to release the product prematurely, before a competitor does or before contractual obligations—possibly associated with a penalty clause—come due.

A risk assessment is worthwhile and might reveal that, under certain circumstances, releasing early could be beneficial. However, problems might arise if no one performs a risk assessment or the actual risks prove much greater than perceived.

In the short term, delivering incomplete software products causes customer dissatisfaction. Long-term repercussions might include a bad reputation and loss of market share or sales. If incomplete deliveries happen often enough, the company could go out of business. In a worst-case scenario, company staff could be exposed to civil or criminal penalties.

Rush job

Occasions can arise in which either a poor work ethic or perceived pressure to deliver compromises quality. A developer working on a software product delivers working code, but the quality of the product is shoddy, with minimal or no rationale and little or no documentation. The programmer might feel under pressure to deliver, becoming more concerned about meeting milestones than ensuring quality.

The rush job and mea culpa dilemmas differ markedly. In the mea culpa mode developers still deliver a product, although one missing functionality. However, in the rush job scenario, full functionality can be present, but the resulting low-quality product does not meet set standards because developers intentionally traded quality for speed of implementation.

Nondiligence occurs when important documentation such as requests for proposals, requirements documents, or contracts does not receive a thorough review.

Not my problem

Occasionally, a project team or staff will concern itself with day-to-day activities, accepting the development culture's status quo and showing no inclination to improve productivity or quality. For example, error codes might be hard-coded in the software rather than placed in a table. When developers ignore best practices, they can leave the door open for civil, and in some rare cases criminal, liability. We call this dilemma *not my problem* because team members frequently will state that quality, productivity, and best-practice issues are someone else's responsibility.

Red lies

Red lies occur during meetings with clients or management, when representatives make statements about a product or project that are known to be untrue—such as stating that a project's delivery is on schedule when the team already knows they cannot deliver it on time.

There is a little bit of not my problem in red lies. For example, rather than admit that a project is behind schedule, a project manager could rationalize that the burden of making up the lost time will fall on the development team. This lets the manager report an idealized schedule. If the development team does not meet this schedule, that failure becomes *their* problem.

The use of the color red is significant in this case because it indicates what might happen to the company's bottom line if this behavior becomes pervasive or ongoing.

Fictionware versus vaporware

The *fictionware* dilemma occurs when an organization or individual promises or contracts to deliver a system for which some agreed-on features are infeasible. Fictionware and the frequently used term *vaporware* differ in that a fictionware product exists but lacks a variable amount of the specified functionality. In the case of vaporware, the product simply does not exist. This situation typically occurs when people feel under intense pressure to meet sales targets; denial can make it difficult to read a request for proposal objectively.

Fictionware contracts are endemic in contracting organizations that decouple sales commissions from delivery. The sales representative might have only a vague understanding that the contracted-for project is infeasible, but that person really does not care because the commission is contingent on the contract award, not on long-term profitability.

Mitigating problems with fictionware can best be achieved by coupling merit or bonus payments to *after-completion* project profit, and by giving engineering professionals significant upfront responsibility and authority to influence the bidding or quotation process.

Nondiligence

This behavior occurs when important documentation such as requests for proposals, requirements documents, or contracts does not receive a thorough review. In the case of nondiligence, agreements might be made without a careful understanding of what is being agreed to, either because of failure to carefully evaluate a specification or failure to pay close attention to staff when they voice their concerns.

Canceled vacation

A canceled vacation syndrome can arise when managers pressure staff members at the last minute to cancel planned trips or otherwise sacrifice their personal time—and possibly money through, for example, nonrefundable trip reservations—to meet a short-term deadline.

While working at a consulting company, one of us observed several consultants being told to cancel their vacation plans so that a project milestone could be met. In one case, the employee's parents were flying in from overseas, and the trip plans had been finalized nearly a year before the trip date. The forced cancellation indicated a lack of planning on the part of project management, and while potentially solving a short-term problem, in this particular situation it caused an even more serious long-term staffing and morale problem. Every employee asked to cancel a vacation left the company within a year. Moreover, management killed the project shortly after the trip cancellations occurred. So the company that fostered this canceled vacation syndrome gained nothing and lost several valuable employees.

Table 1. Mitigation strategies: Cross-referencing dilemmas with imperatives.

Ethical dilemma	Applicable ACM–IEEE imperatives	Comment
Mission impossible	Honor contracts, agreements, and assigned responsibilities—"a computing professional has a responsibility to request a change in any assignment that he or she feels cannot be completed as defined."	The difficulty with honoring agreements and not accepting impossible assignments is that often in the organizational culture acceptance of any assignment is the norm when the assignment comes from a supervisor.
Mea culpa	Strive to achieve the highest quality, effectiveness, and dignity in both the process and products of professional work—"The computing professional must strive to achieve quality and to be cognizant of the serious negative consequences that may result from a poor quality system."	The imperative is too broad to allow the professional to recognize when it applies in routine situations.
Rush job	See mea culpa	See mea culpa
Not my problem	See mea culpa	See mea culpa
Nondiligence	Give comprehensive and thorough evaluations of computer systems and their impacts, including possible risks.	Mixed teams of project management, marketing, and sales can make it difficult to achieve this objective, especially if the opinions given do not coincide with senior management's goals.
Fictionware/Vaporware	Be honest and trustworthy.	Honesty and trustworthiness are much more difficult to achieve with organizational dynamics than as an individual. Nonetheless, per the ACM imperatives, there are times when a professional should take a stand or walk away from an assignment.
Canceled vacation	Not covered by the ACM code of ethics. The ACM imperatives deal with fairness and discrimination, not the mistreatment of staff.	The ACM code deals only with generic fairness and nondiscrimination.
Sweep it under the rug	Strive to achieve the highest quality, effectiveness, and dignity in both the process and products of professional work; also, honor contracts.	Management often resolves problems that occur during construction and testing of software; unfortunately, many managers are unaware of or consider themselves not bound by ACM ethical codes.

Sweep it under the rug

This syndrome occurs when unforeseen issues arise that could potentially damage a project or company but, to keep things running smoothly, developers ignore the issues in the futile hope they will vanish. For example, a tester uncovers a flaw in a communication system and calls it to the attention of his supervisors. They determine that while the flaw is real, the odds of its impacting the delivered product are relatively small and, besides, once the customer starts using it, the responsible parties will have moved to another project.

Sweep it under the rug differs from not my problem in that it deals with mishandling or ignoring infrequently occurring unique problems, whereas not my problem occurs when developers fail to address systemic infrastructure or process problems.

ACM AND IEEE ETHICS CODES

The Software Engineering Code of Ethics and Professional Practice contains a set of 24 imperatives that deal with professionalism, the interaction between professionals and society, and leadership (www.ieee.org/portal/pages/iportals/aboutus/ethics/code.html); the ACM Code of Ethics and Professional Conduct contains a set of 10 imperatives that deal with honesty, responsibility, conflicts of interest, technical competence, and fairness (www.acm.org/about/code-of-ethics).[5,6] We have cross-referenced the dilemmas listed with their relevant imperatives in the ACM-IEEE codes, as Table 1 shows. The imperatives are well-crafted and comprehensive.

These imperatives have not served the professional software community as well as they might for a variety of reasons:

- A large percentage of software professionals do not belong to the IEEE or the ACM.
- Many individuals working on projects might not be software professionals, but instead are product or project managers.
- Many ACM and IEEE members are unfamiliar with these ethics codes.

- Even when somewhat familiar with the imperatives, peer, organizational, or other pressures might be brought to bear.
- In some cases, the imperatives are vague and require study to understand when they apply to a particular situation.

We have selected several imperatives relevant to the ethical dilemmas described here to highlight how they might not provide adequate guidance to the software professional during daily activities.

Software professionals should be cognizant of the financial, legal, and political repercussions of irresponsible behavior.

Be honest and trustworthy

Determining what honesty entails might be open to question. Just as a heavy gravitational field can bend light, heavy organizational or financial pressure can bend the truth. For example,

- telling a client that software is operational when, in fact, it is under construction;
- forecasting a delivery date that is achievable only if the staff works 24-hour days; or
- stating that there are no known problems with software when, in fact, testing or development have reported serious problems.

The problem is not only that honesty might be open to interpretation, but that intense organizational and financial pressure might be applied to cast issues in a particularly biased light.

Quality, effectiveness, and dignity

As conscientious developers, we should strive to achieve the highest quality and greatest effectiveness in both the processes and products of our professional work—and we must do so with dignity.

We learn in requirements engineering that terms like "highest quality" are inherently ambiguous. We also know that quality comes at a price. There comes a point at which the cost to find a product's last few defects outweighs the benefits of finding them. Sometimes recognizing that achieving the highest quality might not be feasible renders the whole issue of quality moot.

For example, because of a shortage of professional staff, or for other reasons, there might be no peer reviews on a project. Code reviews are one of the most effective mechanisms for finding software defects; without peer reviews, an organization might be asking for trouble. Staff might not know that reviews are missing from their process or they might recognize that the reviews are missing but accept management's position that there is no time to conduct them properly.

If an organization is not diligent, its process can easily degenerate into an anarchic hacking environment.

CRIMINAL VERSUS UNETHICAL BEHAVIOR

Sometimes an individual or organization engages in practices that go beyond unethical and stray into the outright criminal.[7] The individual involved might not realize that the practice or the lack of best practice is criminal. In the US, there can be variances in every state in how the laws are interpreted. Outside North America, laws can vary widely, and a practice that is not criminal in one locale can be in another. Thus, individuals might break the law without realizing it.

In all cases at all times, software professionals should be cognizant of the financial, legal, and political repercussions of irresponsible behavior, including condoning behavior of which they themselves might not take part.

Negligent homicide

Negligent homicide involves the killing of another person through gross negligence or without malice. Usually, this sort of unintentional killing involves actors who should have known they were creating substantial and unjustified risks of death by conduct that grossly deviates from ordinary care.[8] The IEEE Code of Ethics commits "to accept responsibility in making decisions consistent with the safety, health and welfare of the public, and to disclose promptly factors that might endanger the public or the environment."

Reckless endangerment

Reckless endangerment occurs when a person engages in conduct that creates a grave risk of death to another. Those engaging in this behavior might not be aware they are endangering others' welfare. Reckless behavior itself is sufficient.

Depraved indifference

Depending on the laws of the jurisdiction, in a more serious kind of reckless endangerment the person engaging in behavior that took the life of another did so under circumstances that evinced a depraved indifference to human life—fully aware that those actions might lead to another's injury or death, but indifferent to that outcome.

Unethical versus criminal behavior

A professional might engage in criminal behavior without realizing that he or she had done so by acting as follows:

- *Not tracing requirements to test cases.* A requirement might state that a radiation dosage for an x-ray machine can, under no circumstances, exceed 5 RAD. If the developer permits larger doses to occur, and the tests do not pick this up, the organization might be criminally liable for failure to follow best practices such as end-to-end traceability.
- *Hard-coding error codes.* A rail signaling system hard-codes errors. Without a table containing the error codes, it becomes impossible to effectively find and test each error condition. This makes it possible for an untested error that occurs during operation to cause a signal to freeze in the go position, resulting in a train collision. The company that created the signaling system can be held criminally liable because basic software engineering texts can show that error management during software development is a fundamental best practice.

One ACM-IEEE imperative is "Know and respect existing laws pertaining to professional work." Unfortunately, in some infrequent situations, professionals learn after the fact that they might have broken laws or are subject to criminal or civil penalties. In the eyes of the law, it is still the individual's responsibility to ensure that best practices are followed, regardless of perceived organizational or management pressure.

DILEMMA MAGNIFICATION EFFECTS

When ethical dilemmas are coupled or chained together, the results can be more damaging than any one dilemma occurring alone.[9] For example, nondiligence might result in late delivery. At that time, it might be possible to renegotiate a viable schedule. However, to make up for lost time, a mission impossible dilemma results in difficult schedules with organization pressure applied. The probability of project failure now increases as things become more chaotic and process suffers.

Delivery pressure causes the rush job dilemma as developers abandon best practices to get the software out the door. The probability of project failure increases again as failure to follow best practices might result in an untenable product or delivery that never works. In general, the dilemmas tend to cascade: The earlier in the process the dilemmas are recognized, the easier it might be to alter behavior and steer toward a positive or at least less negative outcome.

MITIGATION STRATEGIES

When faced with a potential ethical dilemma, one of the best mitigation strategies is to perform a risk analysis before deciding on a course of action. An effective preventive strategy involves providing a working code of conduct and holding ethics training sessions for all staff. Table 1 lists additional preventive strategies.

While most companies and organizations have ethical codes of conduct, software professionals might not recognize that such codes apply to everyday practices as well. All the dilemmas we have described occur commonly, but often the participants do not recognize that their behavior is unethical.[10] Perhaps by naming the dilemmas as developers do with software patterns, it will be easier to recognize their occurrence and take corrective action.

Ethical dilemmas can cascade, with an increased probability of project failure with each misstep. Unfortunately, we lack the data to quantify the contribution of each dilemma to the probability of project failure.[11] Such information can usually be found buried in the failed-projects file cabinet.

One possible mechanism for preventing such behavior is professional or corporate education.[12] Clearly, it is not enough to reach out to members of the IEEE or the ACM as the initial or causal dilemma in a chain might occur further upstream—during contract negotiation or product definition, for example.

Further, IEEE and ACM ethical imperatives must be clearly communicated to computer science and software engineering students and professionals so that they can recognize unethical behavior, see the relevance to their work, and swiftly stop or mitigate it.[13-16]

References

1. T. Forester and P. Morrison, "Computer Ethics: Cautionary Tales and Ethical Dilemmas in Computing," *Harvard J. Law & Technology*, Spring 1991, pp. 300-305.
2. V. Flynn and T. Hall, "Ethical Issues in Software Engineering Research: A Survey of Current Practice," *Empirical Software Eng.*, Dec. 2001, pp. 305-317.
3. E. Towell, "Teaching Ethics in the Software Engineering Curriculum," *Proc. 16th Conf. Software Eng. Education and Training*, IEEE Press, 2003, pp. 150-157.
4. R. Godfrey, "The Compleat Software Engineering Professional—Doing the Right Thing as Well as Doing It Right: Five Steps on the Road to an Ethics Curriculum," *Proc. Int'l Conf. Software Engineering: Education and Practice* (SE:EP 96), IEEE CS Press, 1996, pp. 26-32.
5. D. Gotterbarn, K. Miller, and S. Rogerson, "Software Engineering Code of Ethics," *Comm. ACM*, Nov. 1997, pp. 110-118.
6. D. Gotterbarn, K. Miller, and S. Rogerson, "Computer Society and ACM Approve Software Engineering Code of Ethics," *Computer*, Oct. 1999, pp. 84-88.
7. J.R. Herkert and B.M. O'Connell, "Teaching Product Liability as an Ethical Issue in Engineering and Computer Science," *Proc. 33rd ASEE/IEEE Frontiers in Education Conf.* (IFIP 03), IEEE Press, 2003, pp. S2A-12.
8. J. Samaha, *Criminal Law*, 7th ed., Wadsworth Group/Thomson Learning, 2002.
9. R.P. Feynman, "Appendix to the Roger's Commission Report on the Space Shuttle Challenger Accident," *Report*

of the Presidential Commission on the Space Shuttle Challenger Accident, Washington, D.C., 6 June 1986; www.ralentz.com/old/space/feynman--report.html.

10. K. Brunnstein, "Why a Discussion on Ethical Issues in Software Engineering Is Overdue," *Ethics of Computing: Codes, Spaces for Discussion and Law*, Chapman & Hall, 1996, pp. 52-55.
11. J. Singer and G. Vinson, "Ethical Issues in Empirical Studies of Software Engineering," *IEEE Trans. Software Eng.*, Dec. 2002, pp. 1171-1180.
12. H. Pournaghshband and A. Salehnia, *Ethical Issues of Information Systems*, Idea Publishing Group, 2001, pp. 253-256.
13. D. Gotterbarn, "Ethical Considerations in Software Engineering," *Proc. 13th Int'l Conf. Software Eng.*, IEEE CS Press, 1991, pp. 266-274.
14. D. Gotterbarn, "Raising the Bar: A Software Engineering Code of Ethics and Professional Practice," *Proc. Ethics and Social Impact Component on Shaping Policy in the Information Age*, ACM Press, 1998, pp. 26-28.
15. J.B. Thompson and E. Towell, "A Further Exploration of Teaching Ethics in the Software Engineering Curriculum," *Proc. 17th Conf. Software Engineering Education and Training*, IEEE CS Press, 2004, pp. 39-44.
16. E. Georgiadou and P.K. Oriogun, "Professional Issues in Software Engineering Curricula: Case Studies on Ethical Decision Making," *Proc. Int'l Symp. Technology and Society*, IEEE CS Press, 2001, pp. 252-261.

***Brian Berenbach** is the technical manager of the Requirements Engineering Competency Center at Siemens Corporate Research, Princeton, N.J., and is an ACM Distinguished Engineer. Berenbach received an MSc in thermodynamics from Emory University. Contact him at brian.berenbach@siemens.com.*

***Manfred Broy** is a professor in the Department of Informatics of Technische Universität München, Germany. His research interests are software and systems engineering, comprising both theoretical and practical aspects. Broy received a Habilitation degree in informatics from the Technische Universität München. Contact him at broy@in.tum.de.*