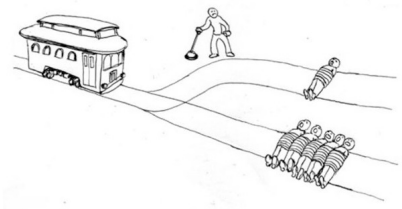
Ethics in Software Development: Time Traveler Edition

Back...from the future

Moshe Zadka – https://cobordism.com

2019



Who am I

- Born: July 10th, 2258
- Certified Software Developer (passed exams Jan 22nd, 2287)
- No formal higher education

I have to let you all in on a little secret. I was born in the future. 2258, to be precise. Until I travelled back in time, I was working as a certified software developer ever since I passed my exams in 2287.

I don't want to go into the details of my time travel. I want to talk about the differences in software development. I want to talk about software developer ethics. A code of ethics. But what is a code of ethics?

Ethic codes are hard choices

Ultimately, a code of ethics consists of "solutions" to ethical dilemmas: a case where, regardless of what we do, some damage will be done. A code of ethics tells us who to damage. In the most of extreme of cases, a code of ethics tells us who to kill.

So why do we want a code of ethics? Why do we want to kill? It is because in some professions, you are going to be responsible for the life of people. This responsibility for life is part of what you get when you choose the profession. This is something even you 21st century folks know: some professions have accepted that part of their responsibilities is making life and death decisions.

Ethics in 21st century: civil engineers

- Example: Act as faithful agent
- Example: Issue true statements
- Example: Service with competence

Close to software developers, civil engineers is also a profession where a lot of technical knowledge and understanding of abstract principles is used to create things people use every day. Since they have been building things since the dawn of time, and occasionally those collapse killing thousands, civil engineers had to already think about what their ethics should look like.

Ethics in 21st century: lawyers

- Example: Confidentiality of information
- Example: Competence
- Example: No counsel to criminal actions

Lawyers, as welll, have their code of ethics. Part of their job is dealing with people who broke the law, or possibly broke the law, or who plan to have very interesting relationships with the law. Regardless of whether the law is just or not, a lawyer is not allowed to give advice to violate the law, though they can explain the consequences of violating the law. Regardless of how bad the secrets confided in them, a lawyer cannot divulge them, unless it is to stop a future consequence.

Ethics in 21st century: doctors

- Example: Responsibility to patient is paramount
- Example: Commitment to medical education
- Example: Report physicians engaging in fraud

Another profession who can see their responsibility for life and death, since they deal, often, with patients that are near death, is doctors. They have their own unique code of ethics, that is applicable for their constraints.

However, in the 21st century, we have no widely accepted code of ethics. The ACM has a code that has none of the rigor that the other codes have, and with none of the training that back them. Of course, the other thing the ACM code lacks is enforcement. Doctors who violate ethics cannot practice medicine. Lawyers cannot practice law. There are no consequences for software developers.

The Incident

This, in part, is what led to the incident. We do not like to talk about the incident. I am not going to talk about it. I'll just mention, it was bad. Really bad. And there was no doubt that this was a direct cause of software development being completely unregulated.

The Incident: Causes

There was a lot of public conversation about what led to the incident. The first thing was that people realized that nobody is ever accountable for software. Everything comes with broad liability disclaimers. Moreover, nobody feels accountable for the final result. Senior management off-loads to product managers, product managers off-load to developers, and they in turn off-load back to product managers, or sometimes to QA. Nobody feels responsible. Nobody is in charge of managing user expectations, and making sure they align with what the software does. Most programmers work on one small part of the project, and often have no visibility into what the project as a whole is supposed to do. They often do not even know how this is intended to make money.

The Incident: Consequences

What happened? Trillions of dollars were lost, either directly or as a result of something breaking. People died. Business went bankrupt. It was pretty bad. In 21st century term, it was worse than 2008 financial crisis bad. More like...world-war level of bad.

The Incident: Aftermath

In the aftermath of the incident, finally, the political will to internationally regulate software existed. We went back to the basics. What would make software fit to be used? Who is allowed to write software that people will use? How do we ensure that?

23rd century

It took a while...

Figuring an entire regulatory structure for an existing industry took a while. Imagine if driving regulation started a couple of hundred years after cars were invented. We had to figure out how to enact the regulation in a way that would allow the software industry to adapt. It probably would not have worked, but the incident was so bad, that whenever political will faltered, someone would remind people how bad it can get.

Treaties were made. Laws were confirmed. We finally had a regulatory regime that allowed software to be safe for people.

OS Updates

- Signed
- Mandatory

We started with the basics: what the operating system that runs on devices is supposed to look like. OS updates were signed and mandatory. A non-updated device would be disfunctional until it could get updated. Device vendors had to commit to sending critical and security updates for the lifetime of the device.

In particular, for devices that were to be used for real work, the OS and device maker were considered the same entity. While the device maker could outsource the development of the OS, they had final responsibility.

Application stores

However, a device maker was explicitly barred from making decisions as to what software is allowed to run on the device. App stores would do that. Specific rules for competitions between app stores were enacted. A device maker

could pre-load any app store, but had to give a way for a user to add *any* licensed app store that supported the device.

Licensing an app store is non-trivial. Among other things, an app store has to have significant financial backing. Since it is ultimately responsible for rogue apps, it has to be able to pay out in case of a problem. All big companies run an app store: I will not give names, since things have changed.

Translated to the 21st century, every big company has an app store: Google, Microsoft, Apple, Samsung, Facebook, Disney and more. There are about twenty "popular" app stores, plus some niche stores. Some stores specialize in devices from one company. Some stores specialize in "ratings": you can know that if an app is rated "G", it really is safe for kids.

Some stores specialize in the exact opposite: since most stores do not allow porn or porn-centric apps, there is a market for porn-first app stores, which explicitly allow for those apps.

Finally, there is no difference between an "app store" and a "certificate authority". All apps use the 23rd century equivalent of TLS to connect to servers when they do. Apps will verify that the certificate authority of the server is store they were installed through. For general browsing, installing a store is the same as installing a certificate authority, and will allow connecting to web sites that were signed by this authority.

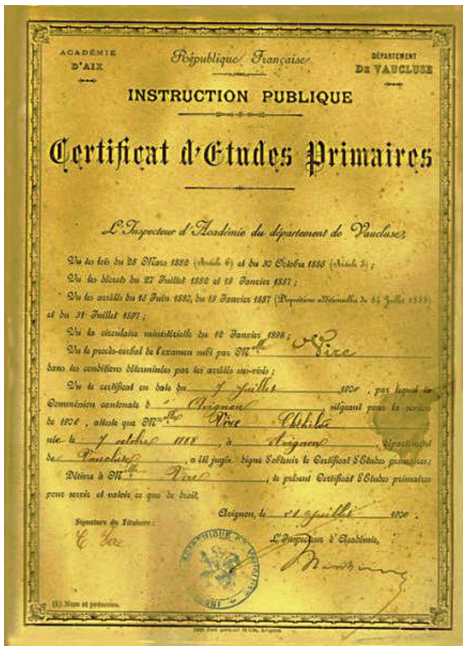
In some sense, it does not matter to the store which part of the app runs on the device, and which part runs on the server: the store certifies the whole thing is both fit for use, and complies with whatever guidelines the store has.

Publishing on application stores

Software Development Studio is a legal category

One regulation that is universal to all app stores, and is part of the qualification for them, is that they must ascertain that each app is build by a licensed software development studio. Registering a business as a software development studio is straightforward. After registration, you get a "license number" that must accompany all apps. If the app store has a problem with an app, the license number is essentially a way to contract the studio.

One thing a software development studio must do is to make sure that all software that makes it into "production" (including any build software) is writ-


ACADÉMIE D'AIX

République Française

DÉPARTEMENT DE VAUCLUSE

INSTRUCTION PUBLIQUE

Certificat d'Études Primaires

L'Inspecteur d'Académie du département de Vaucluse,

ten by certified software developers. There are some accomodations for "interns", but every intern must have a mentor who is a certified developer.

Certified software developers

In order to be certified as a software developer, several things have to happen. First, there is a "competence exam", where sufficient competency in areas of software development must be demonstrated. There is both a practical test, implementing working software and fixing bugs, as well as a theoretical test, including concepts like "what is build", and "what are tests".

Next, there is the "ethics exam": sufficient understanding of the software developer code of ethics must be demonstrated. This does not stop bad actors, but it does prevent the excuse of "I did not know it was bad". This test has examples of real ethical dilemmas, and asks what to do.

There is also a test about the laws and regulations surrounding software.

Finally, after passing all these tests, there is a ceremony where the certificates are given, and the newly certified developers swear (or commit) to uphold the code of ethics.

This is not a one-way process. A certification can be suspended, or revoked, by a special licensed tribunal. They do so on code of ethics breaches, after hearing evidence and arguments.

Hobbyist coding

Any device has an obvious switch to "hobbyist mode". In this mode, the device will run whatever software the user puts on it. However, flipping this

switch will cause most apps to avoid running or run in minimal mode, and most web sites will not accept connections from the device.

Hobbyist mode is a fun way to learn how to program: you can write programs on one device, which runs an IDE, and test them on a "hobby" device. Most developer tools support this scheme, and this is a common way to develop software.

Note that many web sites will have a "sandbox" which has a similar API to the real site, but without real data, and allows connections from hobbyist devices.

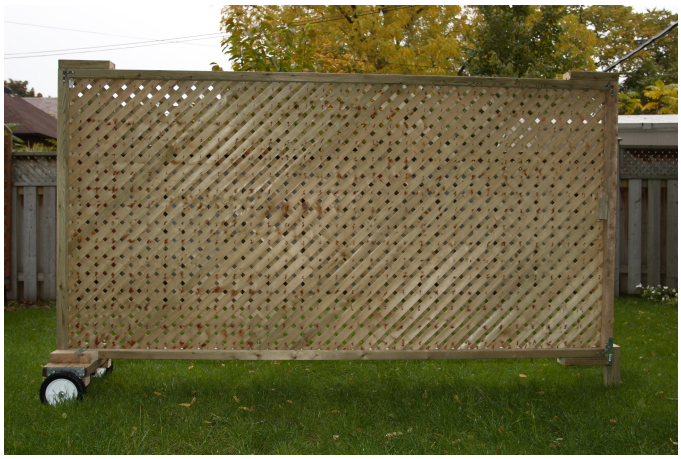
Another development option is to use a "sandbox" app. Sandbox apps emulate a device, and will run apps inside. They will have strict control on connections to the outside, preventing them entirely or sometimes supporting some minimal ability to approve specific connections. Sandbox apps are hard to write, but there is a great market, so there is an incentive to write them. App stores tend to be especially cautious of sandbox apps, and will sometimes have rules limiting who can publish those.

Privacy laws

We also have better regulation around privacy laws. The privacy policy must be easily accessible at all times, not just at install time. It must also be clear. There are specific guidelines on the length of it, and there are more vague, but still enforceable, guidelines on its language.

For example, a common misunderstanding, proven by surveying people on their expectations, will be considered as though it was part of the privacy policy.

Finally, privacy policies are *immutable*. Organizations cannot have a one-sided change of a privacy policy. There are two options that they can take when they need to change the privacy policy for business reasons. One is to *ask* people if they are willing. In that case, "no" must be respected as an answer.



The other option is to sunset the service, and encourage people to install a new service, with a new privacy policy, which will migrate the data.

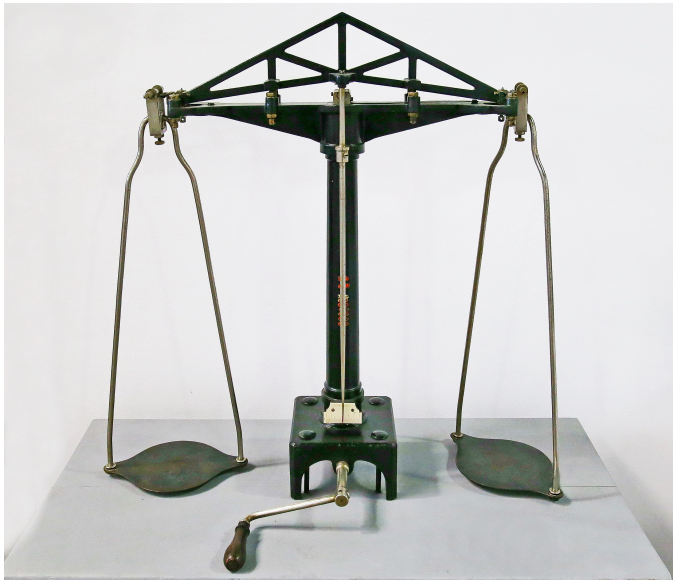
Security laws

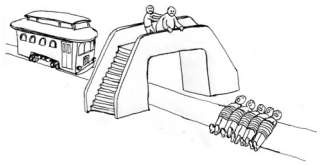
There are also laws governing security. Any breaches must be immediately reported. The only allowable delay is "for purposes of active investigation", in which case reporting to a law-enforcement authority is still required. The authority can grant delays to public disclosure on an as-needed basis, but in practice it is limited by internal procedures: a higher rank is needed to issue a longer delay.

Of course, once a breach is reported, the investigation will include whether the company took reasonable care to avoid it. This means using security best practices and measures. Not using those measures exposes a company to civil penalties, as well as potentially, in case of maliciousness or gross negligence, individuals in the company to criminal liability.

Fairness in competition laws

Finally, we have laws enforcing fair competition. These are a result of software being a "natural" monopoly, and monopolies in general being undesirable.





All devices must allow all application stores. As soon as an application store is vetted, it is not allowed to prevent it from being easily installable on a device. All web services must have documented APIs, and all local application data formats must be documented. As long as a user consents, any licensed application can access any service. This means that "importing data" is pretty much a standard feature of any service or application.

This also means that protection against a "Cambridge Analytica" type scandal, one of my favorite historical case studies from ethics class, does not come from limiting what an API can access. If a user can accesss their contacts' pictures, updates or details, so can a service which has the user's consent. However, that service must have a clear privacy policy and must come from a licensed development studio.

Code of Ethics

Finally, now that I have introduced the regulatory regime, I can talk about some of the details in our code of ethics. Many of those would look about as weird as deciding to push the man off the bridge without having the regulatory context.

Whistleblowing

Violations of law must be reported

The first principle is that you must report any *suspected* violation of the law. Note that even if the law was never violated, but you had reason to suspect and not reported, it can be grounds for revocation of the software developer license. This is important: even if a company is exonerated because there was not enough evidence, the software developers involved in a court case might have their licenses stripped.

It is often easy to develop software in silence while breaking the law. Strong whistleblowing constraints means that it is dangerous for companies to try and do that: law enforcement is going to be involved, because people care more about their careers than their jobs.

Holistic understanding

Whence is the money?

A software developer is committed to understanding the whole product, at least in general terms, that they are buildling. In particular, they are expected to ask, and understand, what the business model of the software is: how can the company afford to pay them.

This avoids the "I didn't know" defense: it is your responsibility to know if the software you are making is used to support dictatorships or to kill people. Now, what you *do* with the knowledge depends on your own ethics, and is not really constrained. However, not *knowing* where the money is coming from *is* a breach of ethics.

Responsibility to principal

Except for violations of law, responsibility is to the development studio.

The ultimate responsibility of a developer is to the studio, unless instructed to violate the law or the code of ethics. While it is good to be concerned about the end user, or other impacted people, the only ethical way to avoid violating personal ethics is to quit and look for another job. Specifically, sabotaging the software, divulging trade-secrets, or violating NDAs is explicitly against the code of ethics.

This means that *employment* NDAs are more enforceable: violating those can be a career-ending move for a developer. This does not pretain in general to NDAs: unless you are signing on a software developer to a studio, this does not make an NDA breach to be an ethics violation.

Commitment to competence

Know your limits

A software developer is committed to working within their competence. This means two things. A developer must keep up to date on new developments and discoveries: for example, ongoing education on new kinds of attacks is crucial to knowing how to defend against them. In general, there is a thriving market in "continuing education" courses, in various formats, for software developers.

This also means that a developer must refuse to work outside their competence. If they do not understand a language well, for example, they must insist someone with understanding will review their code before going to production.

Process documentation

Document disagreements

Any disagreements among the development team must be documented in a non-deletable subpoenable format. This means that, as a matter of common practice, any trial about software development malpractice starts with a subpoena for all disagreements. This fullfills a similar function to whistleblowing: in cases where there is no clear legal violation, those documented disagreement still allow ferreting out whether any of the laws have been breached.

Again, note that documenting disagreements is an ethics matter: having a disagreement and *not* documenting it means it is grounds for license revocation, *regardless* of what the violation involved. There is no commitment to no compromise: merely one to documentation.

Discrimination and harassment

Beyond employment law

Finally, there are protections against discrimination, harassment, and retaliation. While most of those echo the law, they go beyond the law in some protected categories, but they also serve as a way of standardizing the law.

21st century: Incidents

- Cambridge analytica
- Zoom
- Uber

The 21st century had its share of incidents. While none of those rose to the level of "the incident", they were not insubstantial. Cambridge Analytica misused the data of many people. Zoom put people at risk by poor disclosure of vulnerabilities. Uber had a hostile work culture from the CEO down.

There is really no reason to wait for the incident to act.

If Not Now, When?

Can we change the timeline?

The ethics of time travel, naturally, are still in their infancy. So I propose something radical: let us change the timeline. Do not take my word for it: many professions where you are responsible for serious things, and require a high degree of technical competency, have codes of ethics.

Those codes help practicioners make better decisions, and helps them do the best work they can. They are not perfect, but they are better than nothing.

Improving the regulation around software, and having an enforceable code of ethics, might save the world... or at least, the lives of millions.

Picture credits:

- Picture of match: https://commons.wikimedia.org/wiki/File:Streichholz.JPG
- Picture of fire: https://commons.wikimedia.org/wiki/File:San_Francisco_Fire_Sacramento_Street_1906-04-18.jpg
- Picture of lock: https://commons.wikimedia.org/wiki/Category:Padlocks#/media/File:Candados_2012.JPG
- Picture of scale: https://commons.wikimedia.org/wiki/Category:Weighing_scales#/media/File:Vaga_MNT.jpg
- Picture of screen: https://www.flickr.com/photos/garyjwood/275644014/in/photolist-qmKn1-qmKBr-oYwMG-7UWHnT-EyyT4M-qmL3a-cjf8xs-5wx88t-caR8tq-7QR7pg-6BzSc1-
- Picture of red tape: https://commons.wikimedia.org/wiki/File:Redtape1.JPG
- Picture of certificate: https://commons.wikimedia.org/wiki/File:1901_Certificat_d%27%C3%A9tudes_primaires.jpg