WHY IS BAKING PRIVACY INTO SOFTWARE DESIGN HARD?

Dr Inah Omoronyia, Lecturer in Software Engineering and Information Security at the University of Glasgow, takes a dystopic view of the five burdens placed on software designers in the hope of provoking amenable and sustainable solutions.

In an era where privacy regulatory compliance is key, organisations are mitigating their risk by pushing software builders to consider privacy early in software design. Indeed, the design phase offers the greatest value for incorporating privacy in software, since issues discovered later in the software lifecycle become more difficult and expensive to fix. Also, products that are developed following this approach can leverage the 'privacy-by-design' badge as a selling advantage. But, these benefits can quickly be overshadowed by the burden it brings to the software designer / developer.

'At the same time, it is important to reduce the burden on those team members to know, hold and work with privacy regulations / principles in depth.'

Designers are not generally privacy experts

Software designers are not normally employed as privacy experts, yet they carry the ultimate responsibility of privacy-preserving design. Core in the designer's mindset is how to translate functional requirements into engineering actions in a design. If privacy is considered at all, it is usually later when the core design decisions are already made. Very few developers can comfortably say that at the design stage, they successfully used the foundational principles of privacy-by-design, OECD Privacy Principles or Fair Information Practice Principles (FIPPs) as a reference framework.

Why is using a framework important?

Well, these principles form the basis for existing and emerging privacy and data protection laws (e.g. GDPR and the Data Protection Act). They also serve as the foundation for the creation of leading practice privacy programs. As such, it will be difficult to suggest that a design is privacy compliant when the designer does not consider the foundational principles upon which the privacy regulations are built.

As part of the agenda for realising privacy preserving software, there is the need for tools and techniques that better enable software designers to incorporate privacy at the earliest stages. At the same time, it is important to reduce the burden on those team members to know, hold and work with privacy regulations / principles in depth.

Inconsistencies between privacy and functional requirements

The conflict and tension between privacy and functional requirements is often made worse by organisational pressure for shorter time to market cycles. Of course, this is important in today's world where the only way to beat your fellow competitor is by being the first to get the product to the consumer. But this also has a consequence on how the designer may approach their work.

The easiest approach to deal with the conflict between privacy and functional requirements, in the shortest possible time, is to design the software in a zero-sum manner. In such an approach, the resulting design solution often forces end-users to make uncomfortable trade-offs to forfeit privacy in order to gain functionality, or vice versa. This outcome tends to counteract the benefit of being the first to ship the product.

At the very least, the inappropriate usage of privacy becomes a barrier to technology adoption; there is a loss of trust as the brand becomes compromised; and the organisation becomes more vulnerable to regulatory fines and sanctions. Indeed, the alternative to this conundrum is to design the software in a win-win positive-sum manner, where end-users are not required to consider trade-offs.

A structured, privacy-centric design approach often takes time, requiring detailed consideration of the nature of inconsistencies, as well as having to navigate the difficult path of comparing design alternatives. This option is at odds with the realities of deadline-driven

doi:10.1093/itnow/bwz077

deployment cycles. See Bashar Nuseibeh's review of living with inconsistencies in software development[1] for a deeper perspective of this problem.

Ideological nature of privacy regulations and principles

Although it is easy to see privacy regulations as they apply to software engineering, they are not necessarily created with software engineering in mind. This is because the manner in which existing regulations and laws are written seldom helps the software designer. Often, regulations are written abstractly to cover a wider audience, making it difficult to measure disclosure risk in a more intuitive way. In other cases, they are formulated as slogans that offer useful explanations of the meaning of privacy, but don't have the in-depth systematic and analytic lines of action to achieve such privacy.

In short, the designer needs to ask substantive privacy questions to result in concrete engineering actions that comply with the regulatory requirement of the design. If regulations, principles and laws provide no guidelines on how such questions may be asked, designers may find themselves unable to translate privacy requirements of end-users into concrete and verifiable evidence in technology.

Changing requirements and evolutionary design

In his publication *Is Design Dead?* Martin Fowler argues that changing requirements are the number one big issue that causes headaches in software projects. You may attest to this claim if you are the business analyst who frequently has to interact with the customer. The software designer has to understand these requirements well enough to separate volatile from stable aspects of the design, to ensure that the design is able to evolve and be maintained. Now, imagine adding to this already hard problem, frequently changing privacy needs of end-users, or end-users that only know that they have a privacy need when they hear of bad press in mainstream media...

Well, the real question here is this: how can a designer plan to deal with volatile (privacy) requirements when they have no clear insights on what kind of changes to expect?

Design time vs runtime distinction

Traditionally, all software engineering processes are organised into design time and runtime activities. These categories are highly sequential and dictate that design time activities are completed before runtime, with little or no transfer of top line ideas between the two worlds. For this reason, some design bugs are normally not discovered until the software is tested in a production environment with real data, despite sophisticated design-time checking and pre-release testing.

This distinction can throw the design of privacy-preserving software into a dilemma: at one end, using real data at runtime has stronger implications for privacy and privacy harm, as it is often costly and difficult to recover from. At the other end of the scale, privacy requirements of end-users often emerge at runtime. Since these regulations are not watertight, the design will always be open to privacy vulnerabilities introduced during the design process, only to manifest themselves at runtime, when the software is already deployed to end-users.

Conclusion

As previously mentioned, this dystopic view of realising privacy-preserving software is intentional, aimed at provoking thoughtful discussions and solutions. It's not to say there are no success stories of good design. For example, see Ann Cavoukian's case study of successfully applying privacy-by-design in a facial recognition system (**https://bit.ly/2FmpBP7**). But there is a need for more case studies, particularly in a world where new technologies are being introduced into the market at a much faster pace relative to the privacy techniques necessary to ensure privacy in them.

[1] Bashar Nuseibeh. 1996. *To be and not to be: On managing inconsistency in software development.* In IWSSD '96.