

# Privacy by Design

Antti Vähä-Sipilä  
Nokia

This presentation takes a look at privacy from the *engineering* standpoint. This is not about privacy policies as such, governance of privacy, or reactive measures. That said, it does not address specific privacy *features*, but instead a generic approach of how privacy needs to be reflected in a software development lifecycle and organisation.

My viewpoint is from developing products through agile methods that handle consumer data. Whereas I do not believe that other types of private data would be much different in terms of engineering principles, there may be regulatory or compliance issues that force certain practices for other domains of data.

Privacy by Design principles are the “how”.

It guides the intent,  
business case creation,  
and spirit.

Privacy by Design is a collection of principles that guide the thinking around privacy. Originally coined by Ontario (Canada) Information and Privacy Commissioner Ann Cavoukian, it has become somewhat of a buzzword lately in privacy circles.

If you wish to learn more about PbD, have a look at the Privacy by Design portal at <http://www.privacybydesign.ca/>.

1. Proactive not Reactive; Preventative not Remedial
2. Privacy as the Default Setting
3. Privacy Embedded into Design
4. Full Functionality — Positive-Sum, not Zero-Sum
5. End-to-End Security — Full Lifecycle Protection
6. Visibility and Transparency — Keep it Open
7. Respect for User Privacy — Keep it User-Centric

PbD is based on these seven principles:

<http://www.ipc.on.ca/images/Resources/7foundationalprinciples.pdf>

## Kerry / McCain Commercial Privacy Bill of Rights

### **SEC. 103. PRIVACY BY DESIGN.**

Each covered entity shall [...] implement a comprehensive information privacy program by

—

- (1) incorporating necessary development processes and practices throughout the product life cycle [...]
- (2) maintaining appropriate management processes and practices throughout the data life cycle [...]

For more information, see

<http://kerry.senate.gov/commercialprivacy>

<http://kerry.senate.gov/imo/media/doc/Commercial%20Privacy%20Bill%20of%20Rights%20Text.pdf>

IAPP analysis: [https://www.privacyassociation.org/publications/experts\\_discuss\\_proposed\\_commercial\\_privacy\\_bill\\_of\\_rights/](https://www.privacyassociation.org/publications/experts_discuss_proposed_commercial_privacy_bill_of_rights/)

EFF analysis: <https://www.eff.org/deeplinks/2011/05/how-would-kerry-mccain-commercial-privacy-bill>

Microsoft, HP, Intel and eBay support: <http://www.ebaymainstreet.com/files/Joint-Statement-on-Commercial-Privacy-Bill-of-Rights-April-12-2011.pdf>

Covered entity: anyone that “collects, uses, transfers or stores ‘covered information’ on more than 5,000 individuals” over a consecutive 12-month period and is subject to FTC authority, the Communications Act or is a nonprofit. (IAPP) Leaves gaps. May take out some consumer rights, esp. related to class actions. (EFF)

The Fair Information Practices\* are the “what”.

They guide (or regulate)  
what may not be done,  
and what should be done.

\*Or an analogous set of practices

Fair Information Practices (in various forms and syntheses) are slightly more pragmatic and easier to map to actual engineering and features. I consider these to be the “what” axis of privacy engineering, whereas PbD is the “how” or “intent” axis.

These practices have several different reincarnations, for example:

USA: FTC Fair Information Practices -[https://secure.wikimedia.org/wikipedia/en/wiki/FTC\\_Fair\\_Information\\_Practice](https://secure.wikimedia.org/wikipedia/en/wiki/FTC_Fair_Information_Practice)

EU: Principles in the Data Protection Directive -  
[https://secure.wikimedia.org/wikipedia/en/wiki/Data\\_Protection\\_Directive#Principles](https://secure.wikimedia.org/wikipedia/en/wiki/Data_Protection_Directive#Principles)

OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data, one of the “classics” or privacy regulation: [http://www.oecd.org/document/18/0,2340,en\\_2649\\_34255\\_1815186\\_1\\_1\\_1\\_1,00.html](http://www.oecd.org/document/18/0,2340,en_2649_34255_1815186_1_1_1_1,00.html)

- a) Consent
- b) Accountability
- c) Purpose of collection
- d) Data minimisation (Collection limitation)
- e) Data use, disclosure and retention limitations
- f) Accuracy and correctness (Data quality)
- g) Security
- h) Transparency (Openness)
- i) Access and availability (Indiv. participation)
- j) Compliance and governance

This is one synthesis of Fair Information Practices. (See references on the previous slide.)

# Privacy Engineering

Having framed the context of "how" (Privacy by Design) and "what" (Fair Information Practices / Guidelines), let's now have a look of how these would pragmatically map to software engineering.

We'll call this "privacy engineering".

|                                 |   |
|---------------------------------|---|
| Proactive, not reactive         | <ul style="list-style-type: none"> <li>Control data disclosure at data source</li> </ul>  |
| Privacy by default              | <ul style="list-style-type: none"> <li>The easiest way for the user must be privacy-preserving</li> <li>Default to opt-in, not opt-out</li> <li>Minimise data disclosure, collection and use</li> </ul> |
| Embed privacy into design       | <ul style="list-style-type: none"> <li>Provide user options out of the box</li> </ul>   |
| Positive-sum, not zero-sum      | <ul style="list-style-type: none"> <li>If a business case seems to contradict privacy, innovate on the business case</li> <li>Offer true customer value in exchange</li> </ul>                          |
| End-to-end lifecycle protection | <ul style="list-style-type: none"> <li>Provide user controls to change and delete data</li> <li>Ensure confidentiality and integrity</li> </ul>   |
| Visibility and transparency     | <ul style="list-style-type: none"> <li>Make data sharing obvious to the user</li> <li>Provide ways to see what data has been collected</li> </ul>   |
| Respect for user privacy        | <ul style="list-style-type: none"> <li>Do not bury privacy in technobabble or legalese</li> <li>If the user says so, respect the decision</li> </ul>  |

This is a (probably not exhaustive) look at how Privacy by Design could map to *requirements*, both *functional* and *non-functional*. If you think about an agile software development shop, Product Owners would necessarily have to address most of these in User Stories (or similar). Many of these aspects target the business case and business logic, and may actually have a large effect on business cases that are based on consumer data, for example.

It is also no very probable that implementation (design, coding) would be able to engineer around business case level privacy issues – at least without major disadvantages or reassessing the business case. PbD therefore provides a good way of addressing privacy *before* code is written.

For further information on the meaning of these principles:

[http://www.privacybydesign.ca/content/uploads/2010/11/PbD\\_Curriculum.zip](http://www.privacybydesign.ca/content/uploads/2010/11/PbD_Curriculum.zip)

In systems design, Privacy by Design:

1. Cuts all the way from business case down to actual implementation and coding
  - How the requirements (product) management is done is critical
2. Has a significant effect and reliance on user interface
3. Can be completely pre-empted by lack of security

This means that business case creation and product ownership (product management) must have a true connection with both privacy folks (usually legal), user experience people, and development.

Disconnect between any of these functions has a great potential of causing privacy issues.

You can have security without privacy, but you cannot have privacy without security.

- Encryption for data in transit
- Vulnerabilities that allow exfiltration of private data

Having a secure software development process is therefore a prerequisite for privacy engineering.

Security is a requirement for privacy (as an example, just consider how hard it would be to keep data private without confidentiality and authentication).

However, [technical] security does not guarantee privacy. This is because privacy has more human aspects and information has a property of once being freed, you cannot get the genie back into the bottle.

I posit that fielding successful privacy engineering process throughout a company is more difficult than fielding a security engineering process (for human and organisational reasons; privacy cuts across more organisational boundaries and different incentive structures). On the other hand, the closer you get to the code, the more these overlap. (For example, unique uncorrelated identifier generation is a cryptographic problem, squarely in security engineering domain.)

|  |   |
|--|---|
| Consent                                  | User interface (usage flow),<br>user documentation (prominence)   |
| Accountability                           | Set-up of SW dev. responsibilities  |
| Purpose of collection                    | User interface (obviousness),<br>user documentation (prominence)  |
| Data minimisation                        | Genuine technical need for usage,<br>identifier and pseudonym usage,<br>anti-correlation design strategies,<br>log and database scrubbing |
| Use, retention and disclosure limitation | User interface (“right to be forgotten”:<br>clear history, reset identifiers, account<br>deletion)  |
| Accuracy                                 | User interface (“right to lie”, account data<br>editing)  |
| Security                                 | (Discussed separately)  |
| Transparency                             | User interface (obviousness),<br>user documentation (clarity of wording)  |
| Access                                   | User data access,<br>User documentation (contact points)  |
| Compliance                               | Logging, monitoring and access control  |

If we look at the Fair Information Practices, the practical (functional!) requirements start to be clearer.

This is again just skimming the surface, but you will notice that a significant amount of these functional requirements are related to the user interface or user experience.

“Security” is one of the Practices, which just reaffirms its position as a prerequisite.

(Notes on the terminology – “right to be forgotten” is now a buzzword in Europe; “right to lie” is meant in the positive way: whether customer data is accurate is in the customer’s decision; if a customer wants to be known as John Doe, in many cases, that is “accurate” unless there is a real legal or contractual reason of knowing the customer’s real name. For a good example of “right to lie” done right, have a look how Google Latitude allows the user to define his/her position without regard to the true position.)

Privacy engineering can be introduced into security threat modelling.

A security engineering setup that only does code-level activities (testing, static analysis) is not mature enough to address privacy.

“Proactive, not reactive” at work here, too.

Threat modelling (a.k.a. threat analysis, architectural analysis) is a key design/coding level security practice to identify privacy issues, because it addresses user experience and business logic in addition to pure code.

The following slides look at how a data flow based threat modelling can be adjusted to handle many of the privacy requirements.

### Data flow based threat modelling with STRIDE:

1. List all software components
2. List all data flows between the components
3. For each data flow, answer whether each of the STRIDE security considerations needs addressing, and if yes, how

(**S**poofing, **T**ampering, [non-]**R**epudiation, **I**nformation disclosure, **D**enial of service, **E**levation of privilege)

Data flow diagram (DFD) based threat modelling using Microsoft's STRIDE is a very accessible and actionable form of threat modelling. For a longer description of it, have a look at the following Microsoft resources:

<http://msdn.microsoft.com/en-us/magazine/cc163519.aspx>  
<http://www.microsoft.com/security/sdl/adopt/eop.aspx>

STRIDE incorporates the traditional CIA (confidentiality / integrity / availability) information security model.

However, STRIDE basically just addresses *security*, not as much *privacy* (except for those areas where these overlap, such as confidentiality).

Extending STRIDE to privacy – for each data flow:

- Do we have consent to send the data?
- Do we have a technical or business justification to send the data?
- Does the data flow cross a jurisdictional boundary?
- If the flow is to a data storage, do we adhere to a retention or deletion policy?

This “STRIDE+4” would catch most design level privacy issues (except for the user experience level).

This is what I call (for a lack of a better acronym and in memory of Commodore) STRIDE+4.

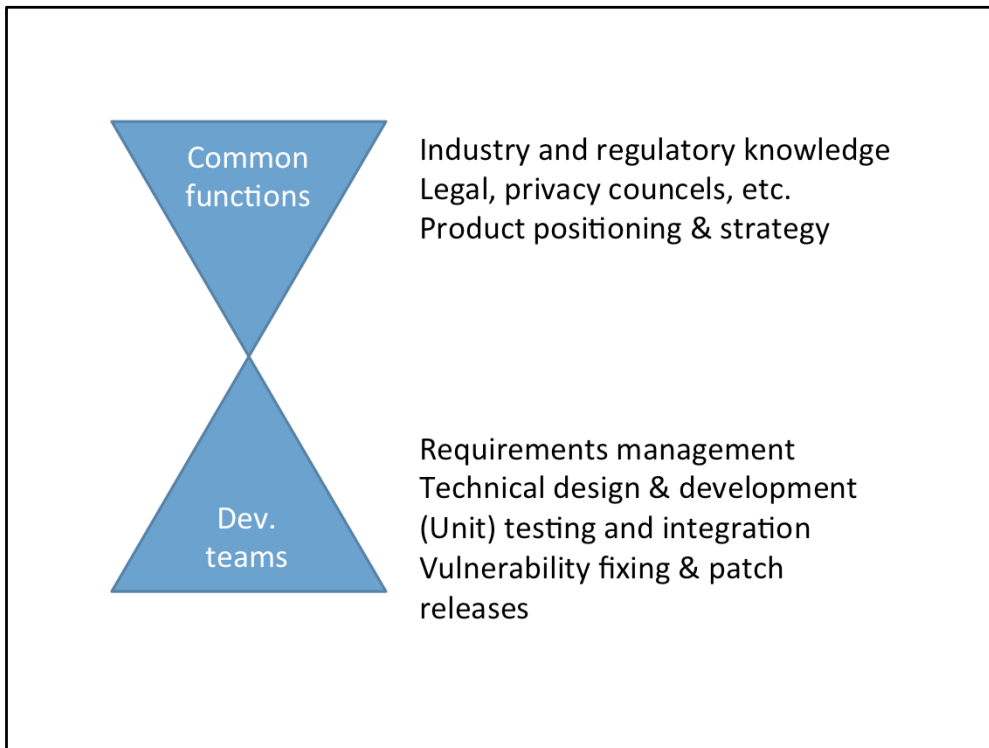
This adds four privacy-specific considerations to STRIDE, intended to be determined for each identified data flow in the system:

- *Consent*: Do we have an explicit or implicit consent to send the data (irrespective of whether the data transfer is secure or not)
- *Justification*: Do we have a business case and/or technical justification to send all the data we are sending (irrespective of whether we have a consent or whether the transfer is secure) – with the intent that if data transfer is not expressly required, it should not be done in the spirit of data minimisation.
- *Jurisdictional boundaries*: Is the data flow transborder, for example, out of EU/EEC, or out of a Safe Harbour? Could it ever happen that it is (for example, in the case of mobile clients)? If so, is there a need for legal review, additional consent, or additional security?
- *Retention and deletion considerations*: If the data flow is to/from a data store, such as a file system or a database, the component that owns that stored data needs to analyse the retention policy.

# Organising Privacy Engineering

in a large software company

The following describes a challenge which I believe is fairly typical of both security and privacy in a (large) software company.



A very typical divide: legal, high-level product marketing, etc. is not necessarily very well connected with the development teams.

If every development team truly has resources to provide all the industry / legal / etc. privacy competence from within their organisation, that would most probably be a superior solution.

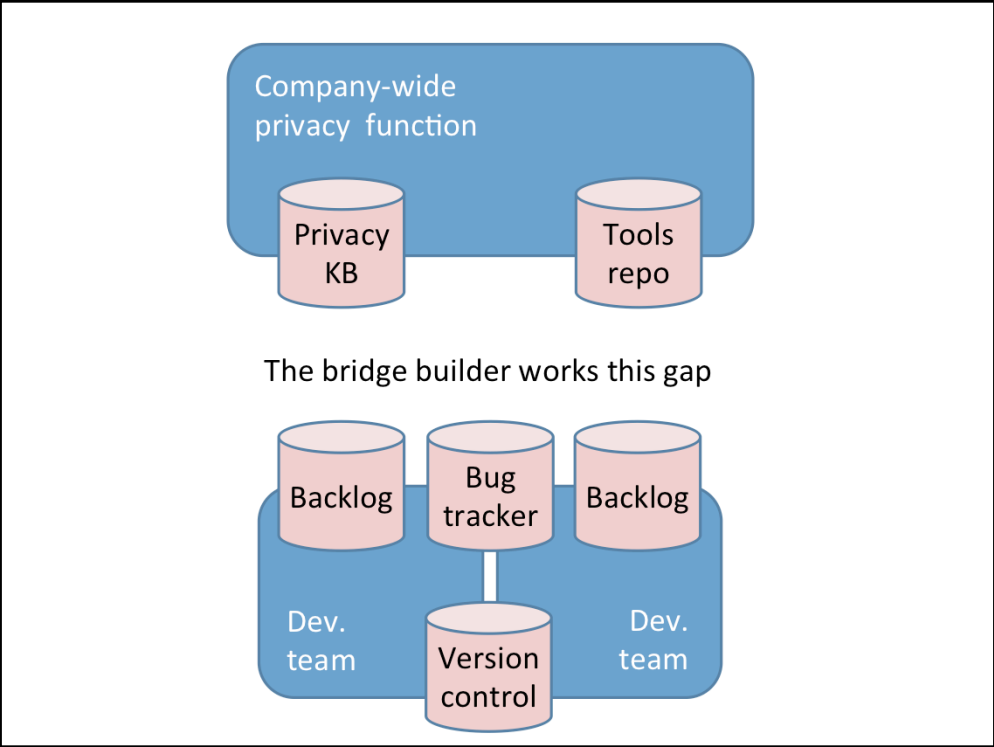
Towards a networked solution:

1. Each database, product backlog, repository, and version control system offers “outsider” access.
2. “Reporting up” replaced with “data mining down”, “pushing” replaced with “pulling”.
3. A bridge-builder – a manager/engineer interpreter – works the network.

Unless privacy competence can be built in into each of the dev organisations, creating a central function that is extensively networked would in my opinion be the next best thing. Security engineering may make use of a “software security group”, whose one reason to exist is to build bridges between the development and other functions. Privacy engineering would benefit from a similar role.

Addressing this type of execution gap in an agile organisation would try to form information flows without encumbering the development.

The key in this idea would be to enable information sharing without adding new reporting requirements. This means providing access to bug trackers, req management / backlog tools, knowledge bases, etc., in a way that facilitates “views” and “pulling” instead of “reporting” and “pushing”.



Qualities of a bridge-builder:

1. Can take a tool and actually deploy it in the development teams' environment.
2. Can take a guideline and reflect it in technical requirements and design.
3. Can help each side to find the right data, for measurement and tracking.

Feedback welcome.

antti.vaha-sipila@nokia.com (Work)

Twitter: anttivs (Personal)

Advertisement: Tietoturva ry (Finnish  
Information Security Association) privacy group  
[http://www.tietoturva.fi/jasenyys/henkilojasen/  
privacy-osy](http://www.tietoturva.fi/jasenyys/henkilojasen/privacy-osy)