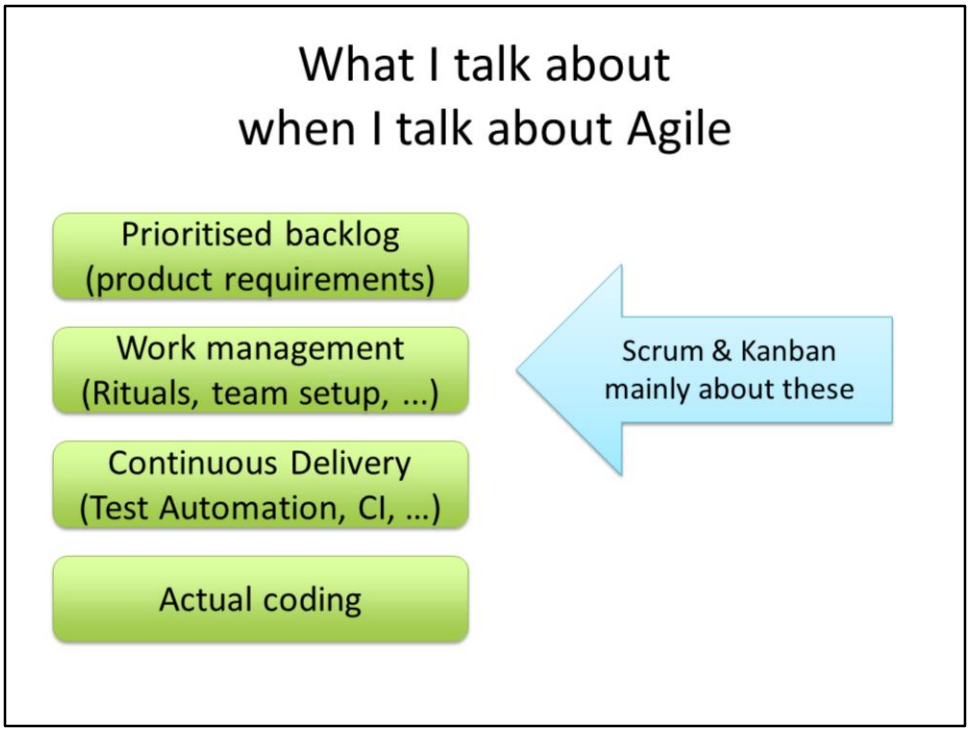
The Agile software security assessment method described at the end of the talk is joint work between me and Towo Toivola, Director, Quality of Operations.

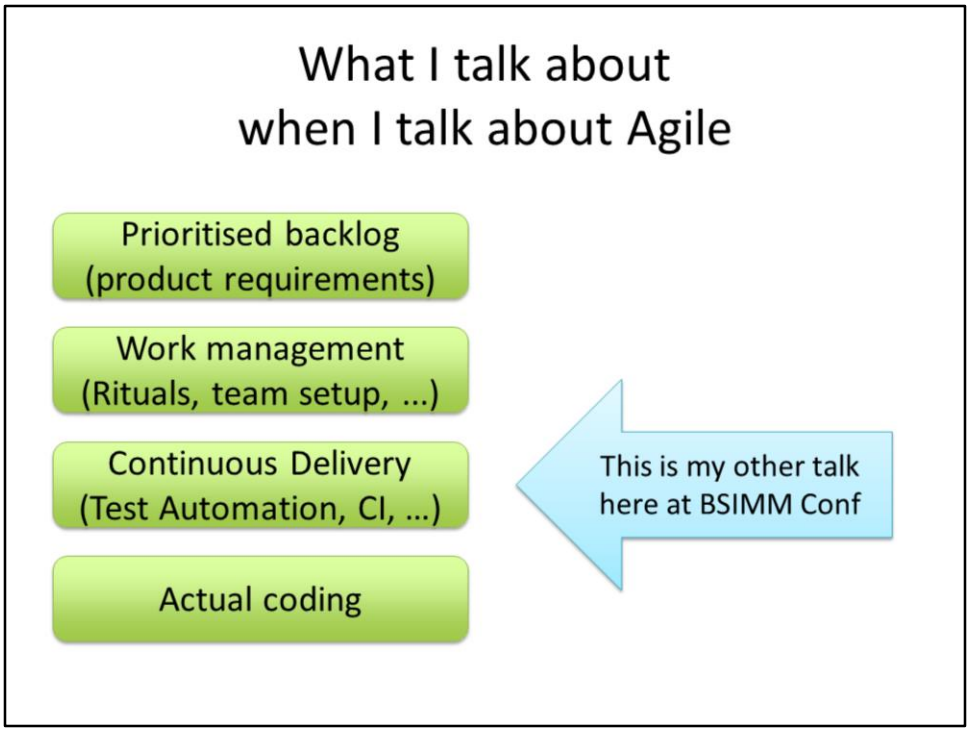
Software security & backlogs work was done partially under the umbrella of a Finnish industry & academia project, the Cloud Software Program (http://www.cloudsoftwareprogram.org/).

You can also find some older material on this from my personal project page at https://fokkusu.fi/agile-security/ and https://fokkusu.fi/agile-security-slides/. The latter one also has further pointers on this topic.



In order to get everyone on the map, I want to give a short definition of what I'm going to talk about in this presentation.

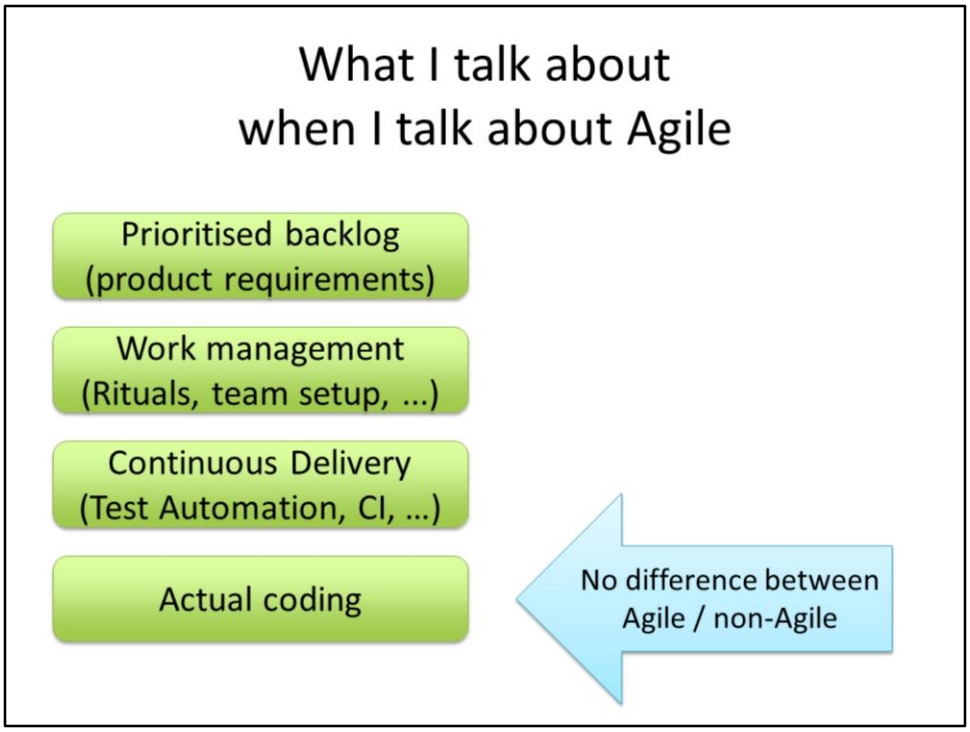
First, Scrum and Kanban are mostly about work management and daily rituals. Especially Scrum is rather ritual-heavy. We aren't going into the specifics of these, and everything in this presentation should apply to either. I am using the term "sprint" at times, that's because I've got a Scrummy background.



Automation (particularly test automation and Continuous Integration) are key factors in an Agile development process. However, this area (the larger context of Continuous Delivery, which also includes DevOps as a team concept) is discussed in my other BSIMM Conference 2013 talk. Have a look.

Using test automation doesn't mean that the development process is agile; there have been many examples of successful test automation in a completely Big-Design-Up-Front traditional processes. It's just that often agile folks also subscribe to Continuous Integration / Continuous Delivery ideals.

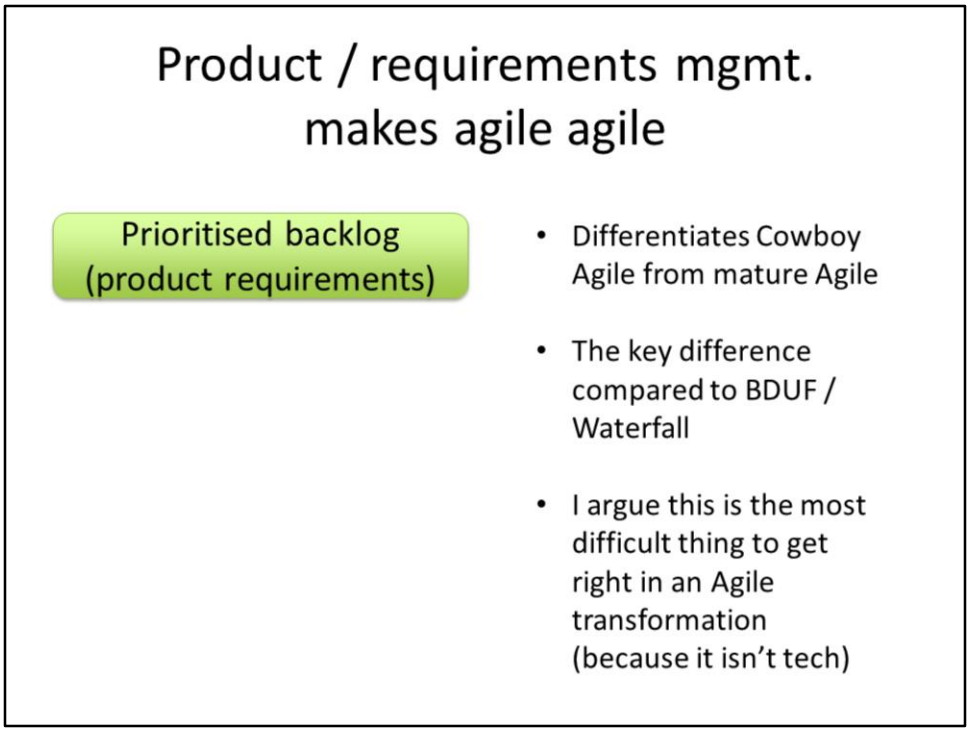
There certainly are things in this talk that refer to automation.



Lastly, the actual coding does not really differ between Agile and non-Agile.

Secure code looks just the same. Static analysis fits both worlds.

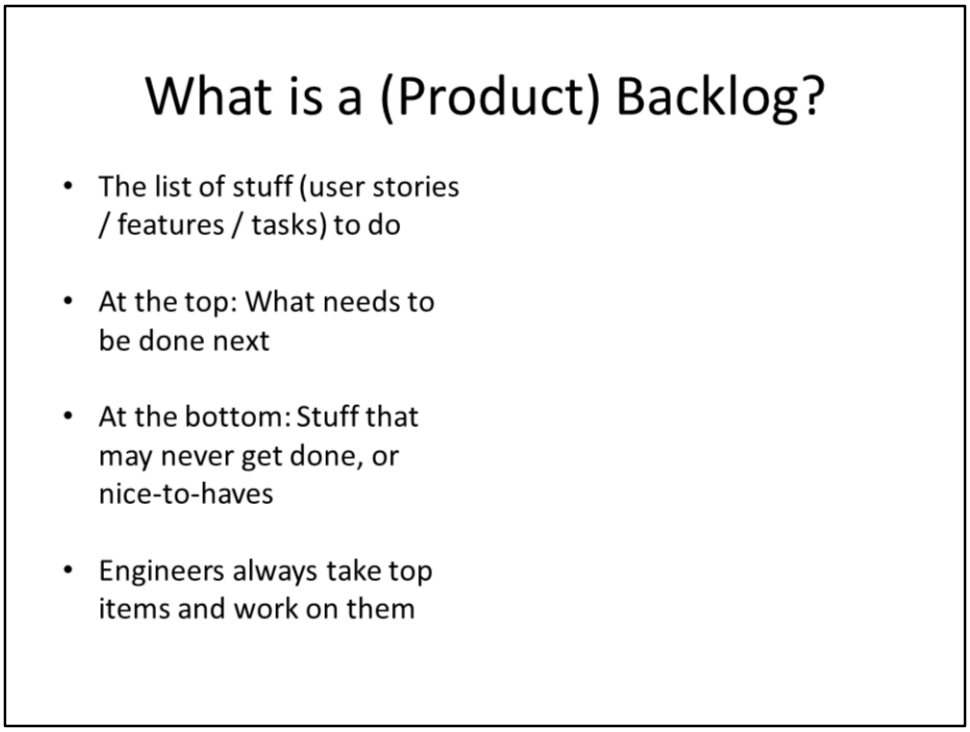
The difference here is how the developer's time is allocated for software security activities, and which aspects push the developer towards secure code. But the actual act of coding really doesn't change. (Except for pair programming, which often gets grouped in the "agile" bucket, but I'm not going there.)



What I think separates Agile from non-Agile is how the business drives the development: through a prioritised backlog (often a "Product Backlog", especially in the Scrum world).

In Agile, the business does not make a Big Design Up Front (BDUF); instead, they list stuff that seems worth doing right now, perhaps aiming for a Minimum Viable Product, or a next shippable incremental feature. The business is always ready to change its mind ("pivot").

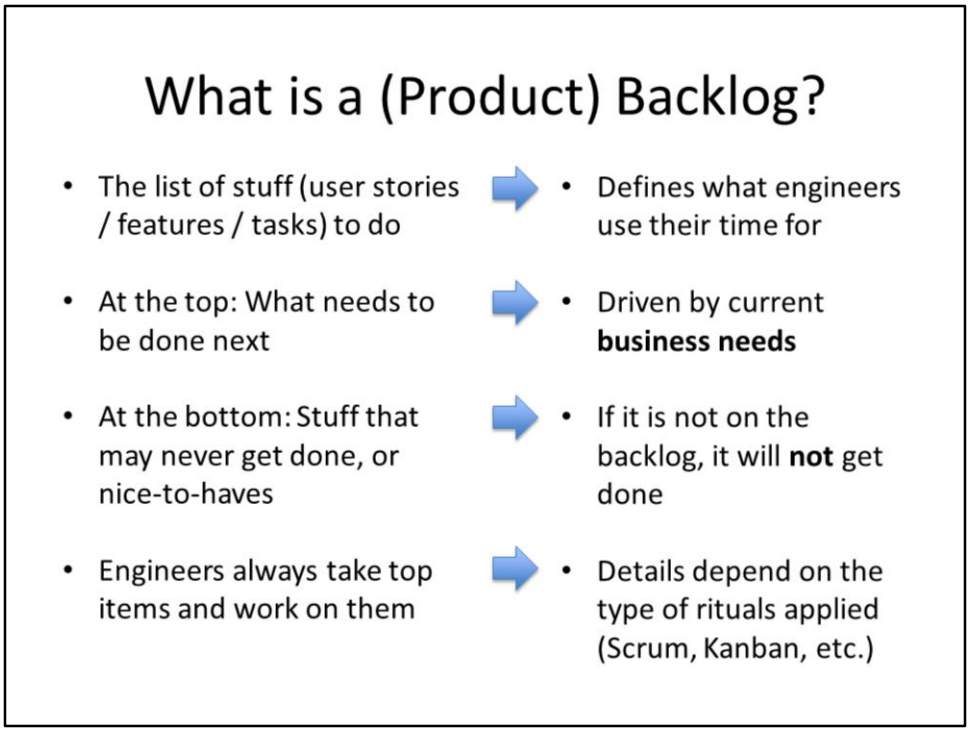
Cowboy Agile is characterised by the lack of rigour on the business side. For example, lack of documentation (which is how the Agile Manifesto is often mis-interpreted) could easily be remedied by requiring documentation just like you require features. It just means the product/requirements management needs to be explicit of the documentation need.



The backlog is a list of things to do. In many cases, people assume it's just functionality and features; however, in order to actually work for software security, the backlog needs to be in a position to drive *all* work. This includes so-called *non-functional* requirements, documentation, and testing work, too. (I'll get back later to why this is a good idea also for project management purposes.)

In some companies, the backlog is not the single source of work for a team, but the team has other inputs that affect their time allocation. *This is usually a bad thing*. Optimally a development team should have one, and only one, source of work tasks – a single backlog. If there are several sources of work, then those sources of work should have a priority order between themselves, and this quickly becomes pretty complex. A Senior Management Priority Override breaks agility.

In Scrum, the top items of the Product Backlog are extracted on a Sprint Backlog for each sprint. If you are a Scrummy person, just think about the Product Backlog during this talk. I won't refer to the Sprint Backlog at any point.



There is (should be) usually one person who decides the order of stuff on the backlog. This person is called the *Product Owner*. This is a kind of synthesis of a product manager (i.e., a requirements guy who knows what the market wants) and a business owner (i.e., someone who has enough authority to decide on what the company R&D uses its money for). In some cases this is a group of people.

But the main thing is that then Product Backlog is prioritised by business needs. Now you can already see that: 1) If Software Security activities must be on the backlog, and 2) It is always prioritised by business needs, it follows that Software Security activities must be explicitly prioritised by business owners. This is a good thing because: a) Awareness b) Correct level of investment c) Avoids SSG having to push activities from the sidelines with a rope.

Also, if something is not on the backlog, it won't get done. This lends visibility to software security activities: As Backlog items are marked as "done", software security activities also get checked off the list and evidence of them mounts; if something is at the bottom of the Backlog, everyone can see that the particular activity has not been done.

Getting SW security activities done: Sub-optimal solutions

- "Themed" sprints, "buckets"
  - Idea that time should be put aside for recurring (security) activities that happen in every (other) sprint
- Definition of Done
  - An agreed set of activities that "ought to be done"
  - Usually quality criteria

What's wrong?

There have been other suggestions of how security work ought to be driven. I, myself, have advocated using the Definition of Done too, but I've changed my mind.

What is common with these options is that they generate work that is *outside* the backlog.

(A side comment: A Definition of Done is a set of quality criteria that describes when a Backlog item is ready for delivery. Most agile coaches agree that a Definition of Done should be agreed between the Product Owner and the development team; and in many organisations, the Definition of Done is imposed as an external requirement set on the team. In the former case, is not guaranteed to contain software security activities and in the latter case, it is not guaranteed that the teams follow it.)

Getting SW security activities done: Sub-optimal solutions

- "Themed" sprints, "buckets"
  - Idea that time should be put aside for recurring (security) activities that happen in every (other) sprint
- Definition of Done
  - An agreed set of activities that "ought to be done"
  - Usually quality criteria

What's wrong?

- **Work not visible in the business-driven priority decisions**
- **Looks like a decrease in velocity (throughput)**
- **"Invisible" work – when pressure rises, corners get cut here**

Work that is outside the backlog is problematic because it doesn't get prioritised against all other backlog items.

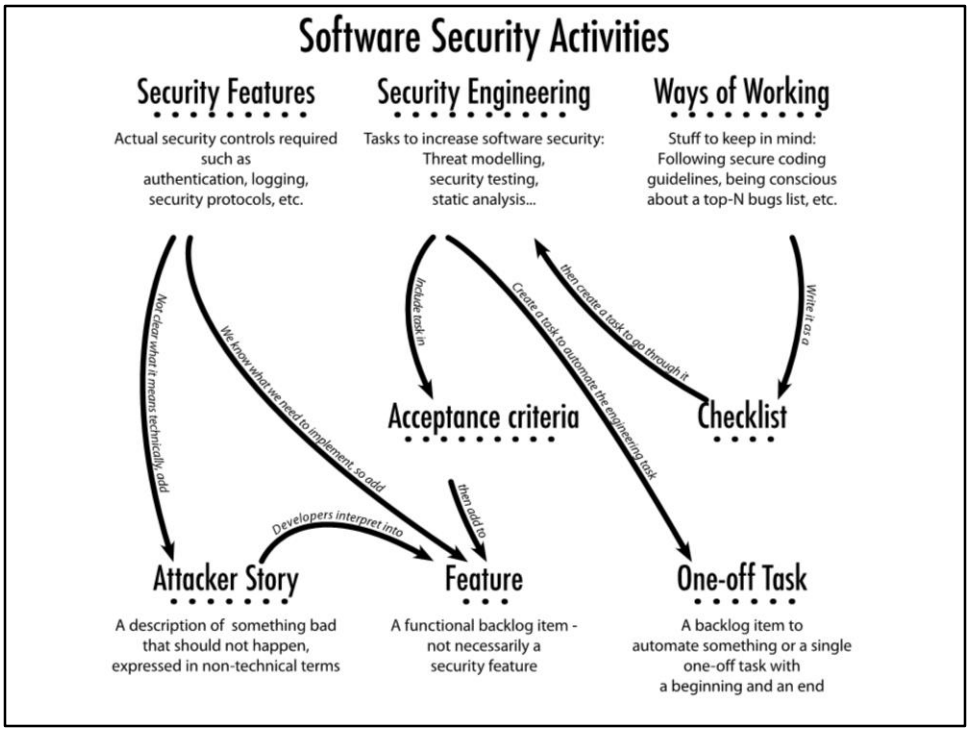
Even software security activities aren't sacred; sometimes something else takes priority (gasp!).

Also, this is a kind of "invisible work" or "dark matter" – for an external observer, it looks like the team wastes their time doing something else than the features on the backlog. The external observer doesn't see that the team is busy fuzzing or conducting ARA or whatever. In the worst case, they'll start to pressure the team to drop this invisible work and just get on with the features.

With schedule pressure, a less disciplined team might even cut corners in this "hidden" work by themselves. The team might get away with this as this sort of work might not be tracked externally. Until later, when the zero-day report lands in the SSG's inbox.

Using the prioritised backlog for all SW security work is often

A BETTER OPTION



This is an extract from a poster I made for a Finnish industry consortium event earlier. The original PDF is available from https://app.younited.com/?shareObject=24a03c09-3d43-bdf6-4110-7ca02251aa44

Basically, this picture has SW security activities at the top and Product Backlog items at the bottom. The small text on the arrows explains what is happening. To summarise:

1. If you have a (functional) security feature, just put it on the backlog as a feature.
2. If you have a security engineering task – something that has a beginning and an end (like a threat modelling session or a web security scan run):
    - Is it recurring? If it can be automated, create a one-off task to automate it, and put *that* task on the backlog. Automation will take care of the task from there on. [Assuming you have test automation!]
    - If it is not recurring, or cannot be automated, write it as an acceptance criterion and tag that with an existing backlog item. (Acceptance criteria are kind of mini-Definitions of Done, like specific quality criteria that need to be fulfilled for this feature to be deemed complete. But unlike Definition of Done, Acceptance Criteria are defined and live as a part of the Backlog item.)
3. If you have a way of working, which is not a task with a beginning and an end, create a guideline or a checklist and treat that according to 2), above.

Why better?

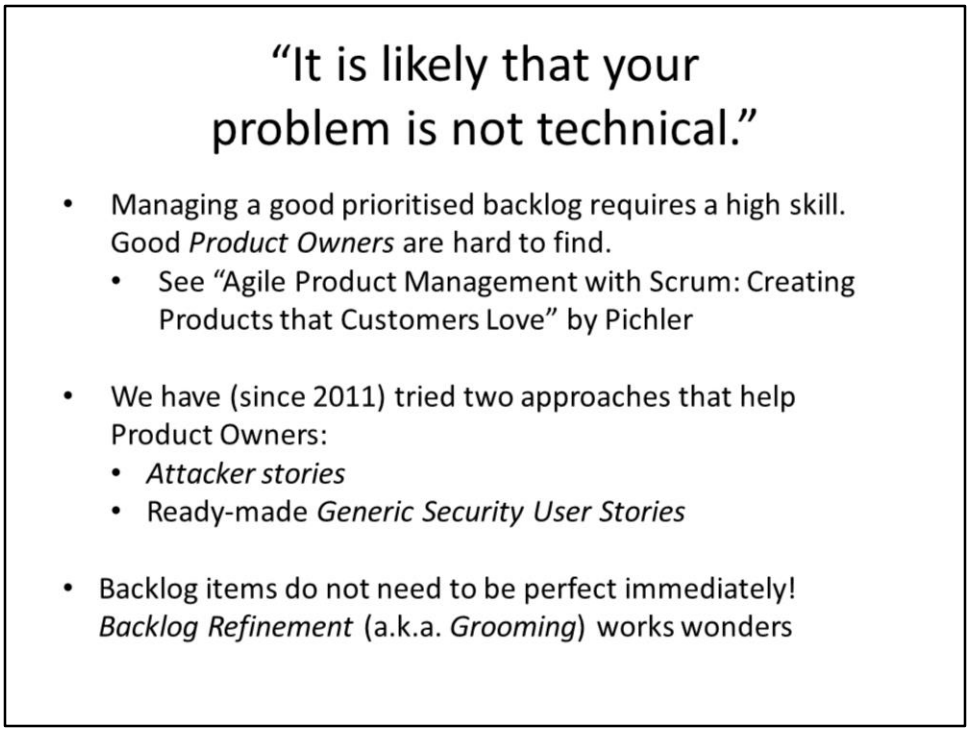
1. All security work is visible and gets prioritised against other business needs like functionality.
2. The security work that has *not* been done is visible, too. This is an indication of the residual risk, and the Product Owner must make a decision to accept it.
3. Security work that has been done gets put in the "backlog items that have been done" pile. Evidence!
4. Not all features require the same security effort. If your quality criteria are feature-specific, dropping a feature also drops the security effort.

Caveats:

1. If your organisation uses acceptance texting that has been segregated from your development (i.e., different people develop and test), it is hard to drive testing through the backlog. My other presentation at BSIMM Conference 2013 on DevOps SW security talks about this particular case.

2. What to do to underlying architectural decisions that pre-date the actual implementation, like selecting a platform or implementation language? I don't have a really good answer for this foundational architecture work in the context of an ongoing Agile project, but it would fit the concept of a "Sprint Zero" - a kind of bootstrapping sprint many teams use. But that would be a themed sprint which I already denounced earlier. So I need to think more about this. For now, I suggest doing a "Sprint Zero" and considering applying attack modeling activities (particularly AM2.2) in that context.

3. Also, high level "enterprise security architecture" work that has been built for BDUF is something is tricky. If the organisation has a separate enterprise architecture ivory^H^H^H^H^H team who does this sort of work outside the dev teams before actual development starts, it really isn't very Agile, and I don't have a good answer for that.

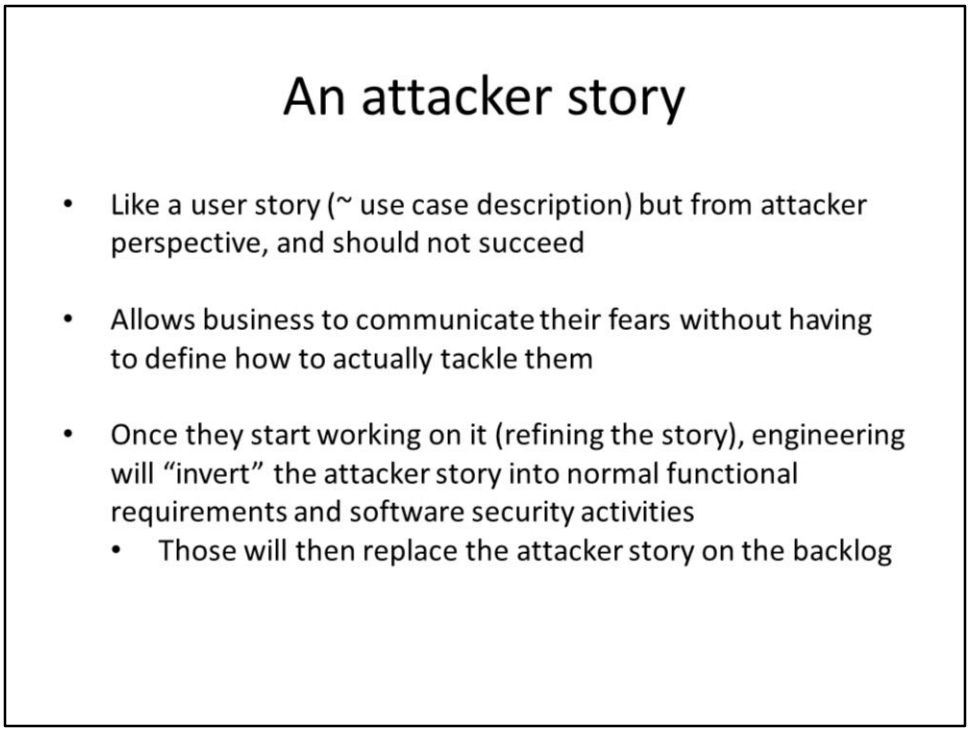


Now, doing this is easier on a PowerPoint than in real life.

Although a software security person might be able to create the appropriate backlog items, in reality the backlogs are maintained by Product Owners of various skill and interest in security, and some of them often bow to the one Executive that shouts the loudest.

Product Owners have been referred to as "anthropomorphic requirements", and the optimal Product Owner is really a mythical beast. In real life, they don't exist, and many companies approximate them through forming a group of people (usually a product manager, an architect, or a lead developer) that *refines* the backlog items.

We have tried two approaches to help Product Owners to take security on the backlog.



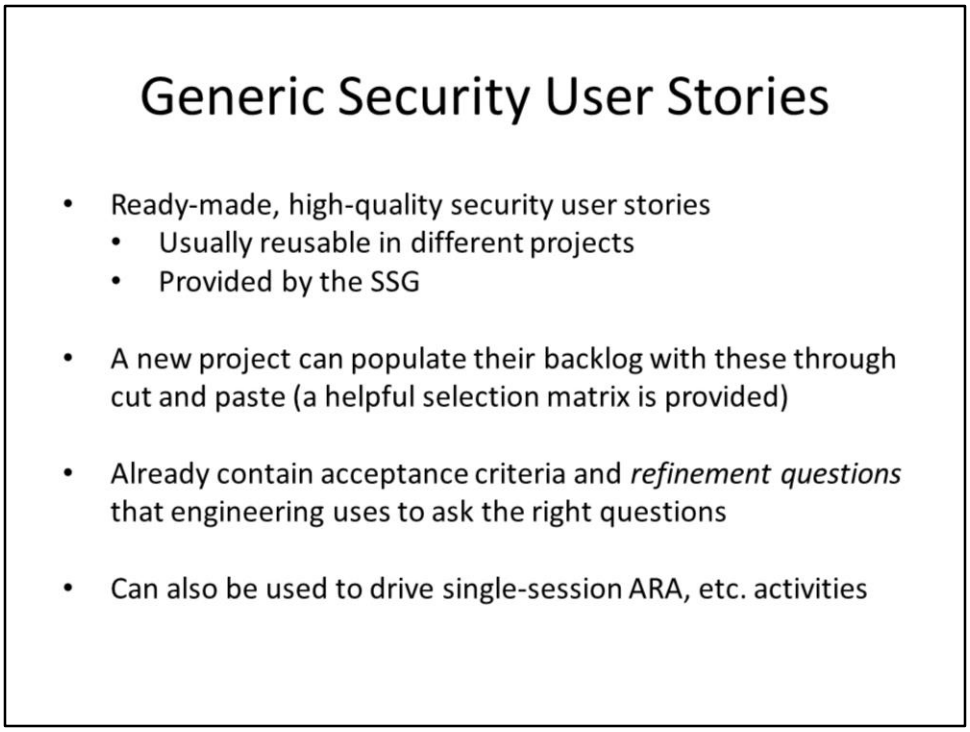
A "user story" is a sort of a canonical use case description:

As (someone) I want (something) so that (description of business value)

An "attacker story", analogously, is a misuse case description:

As (a threat actor) I must not be able to (something) so that (description of attacker's value proposition)

Attacker stories are invented just as placeholders to hold the main business-level risks and fears and to communicate them to the technical people.



We also have a set of user stories that are high quality and fulfil the typical needs of a project. The ones we use are a joint work between me, Camillo Särs and Tuuli Siiskonen from F-Secure's Information Security department.

The idea is that a Product Owner starting any new major development can populate the backlog with these stories just by cut'n'pasting them on the backlog. There's a selection matrix with a list of questions that guides the Product Owner to select the appropriate ones.

We could also have a library of typical software security related acceptance criteria, but currently we don't. Perhaps a future development idea.

With Towo Toivola, Director, Quality of Operations, F-Secure Corporation

Smells of Secure Agile

- F-Secure developed (2013) a qualitative method to determine whether a team's ways of working are conducive to secure software development
- A set of ~30 questions, phrased in 7 different ways, for developer, tester, architect, Scrum Master, Product Owner, project manager and an SSG / "satellite" role
- Questions try to detect *smells* in five categories
- Best conducted by an experienced agile coach + an SSG member riding shotgun

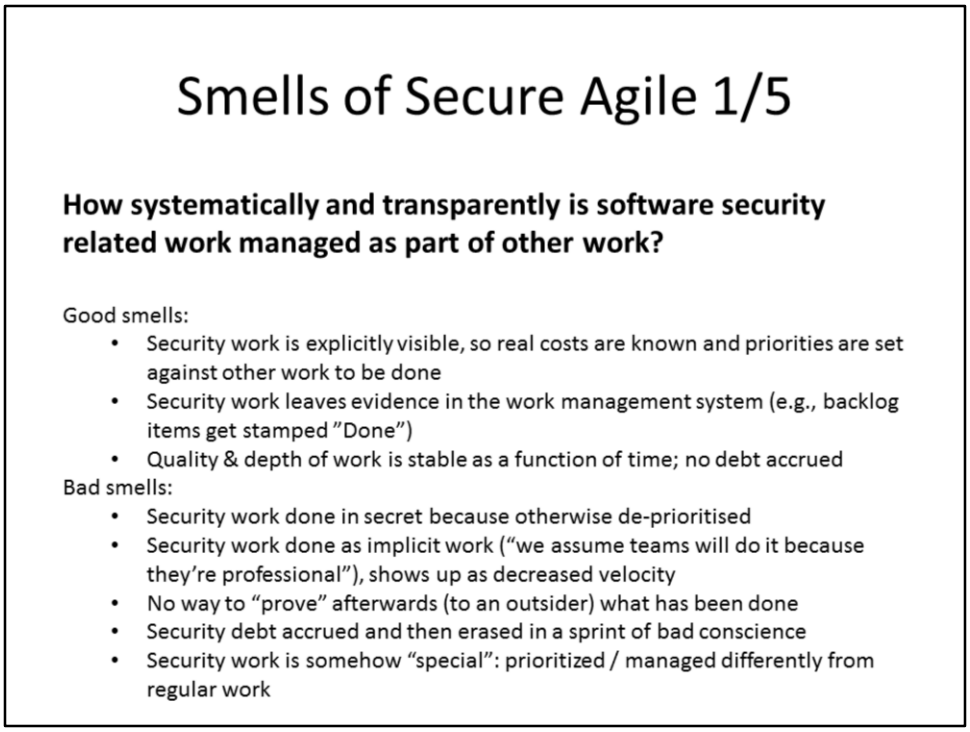
If you are running an program to enhance the Agile practices, that would be a very good opportunity to find allies from the process development / methods development folks.

We developed a method that we've used internally to determine whether the ways of working and the team's internal practices support secure software development. The following slides describe five areas which we try to figure out.

Doing the questionnaire would be best done by a very experienced agile methods person, preferably someone with experience from multiple organisations, and an SSG member. The persons should also have pretty strong quantitative interviewing skills so that they can coax the truth out. I don't believe the method would work as a checklist.

Some sort of qualitative research coding tool (ATLAS.ti comes to mind) would probably also come in handy. We've done it manually, and it's a pain.

The term "smell" originates from "code smells" that have been used to detect code refactoring needs.



Of course these aren't the questions asked from the team members. You need phrase them so that you get this information out of them, but you cannot just directly ask these.

Smells of Secure Agile 2/5

How well are flaws or vulnerabilities handled, and how well is the organization able to learn?

Good smells:

- Externally reported issues and patches are fed through the appropriate process (backlog, bug tracker, etc.), no hidden, secret, or ”hero” fixes
- Root cause analysis is performed & new tests added to catch regression
- Practices trying to actively create T-shaped people (”jacks of all trades, masters of one”)
- Established process for improving way-of-working that reduces future issues (e.g., retrospectives that actually drive change)

Bad smells:

- Issues fixed as special fly-by-night operations, firefighting
- Root cause analysis not done or only at immediate technical level
- ”Hero” / ”Rock Star” culture with certain indispensable individuals
- Improvements not actively and consistently followed up

A “T-shaped person” is a Toyota thing.

Smells of Secure Agile 3/5

How well does development-time test automation serve software security?

Good smells:

- Build & test automation contains static analysis and security testing
- Failure of security-related tests breaks build or stops build promotion; no new commits until the failure is resolved
- Ongoing process to increase coverage (e.g., evolving fuzz test sets)
- Innovative tests (e.g., the Netflix Chaos Monkey) taken into use in-house - innovation indicates that the automation platform is usable and accepted
- Test (or Behaviour) Driven Development held in high regard

Bad smells:

- No test automation, or it is getting stale (quality of test automation not a priority)
- Static analysis or robustness testing not performed automatically, or results not followed up
- Build / test errors not treated as stop-the-line

Smells of Secure Agile 4/5

Is it clear who accepts the residual risk, and based on what data?

Good smells:

- Those security activities and features that are *not done* (vs. would be good to do) are always visible (e.g., on the backlog)
- Architectural security risk analysis is regularly done (an up-to-date threat model exists), and analysis generates new backlog items, not desk-drawer reports
- People are aware of how much risk they are entitled to accept and who has authority to accept higher levels of risk

Bad smells:

- The person who controls developers' time never has to make decisions about security priorities - gets no escalations, or has no wiggle room
- The backlog, acceptance criteria, or Definition of Done make no mention of risk analysis or threat modelling
- Product Owner role is vague, weak, or lacks authority

Smells of Secure Agile 5/5

Does the (same) development team address/own security over the whole lifecycle?

Good smells:

- Operations infrastructure is managed as code, deployed automatically
- Team has development as well as operations competence
- Architects do design, testers read code, developers do environment hardening
- Team mentally takes responsibility for the quality of the entire system

Bad smells:

- Over-reliance on external pentests
- Developers make large assumptions of what operations will do to the product
- "Throw over the wall" mentality at any of the interfaces between business / architecture / development / test / operations activities
- Dev / ops transition has a large "go-live" control point
- Different parts of the organization use differing software development lifecycle methods (e.g., waterfall vs. agile) without compensating interfaces

Thanks

antti.vaha-sipila@f-secure.com (work)
avs@iki.fi (personal)
Twitter: @anttivs

If you have any comments or experiences about this topic, I'd gladly hear them!

Protecting the irreplaceable