Product Security Risk Management in Agile Product Management

Antti Vähä-Sipilä
Nokia

This talk was presented at OWASP AppSec Research 2010 in Stockholm, Sweden.

This talk builds on ideas from many people. In particular, I would like to acknowledge Vasco Duarte, Heikki Mäki, Martin von Weissenberg and Lauri Paatero of Nokia, Camillo Särs of F-Secure, and comments from participants of the workshop held in April 2010 (see last slide).

Agile & lean

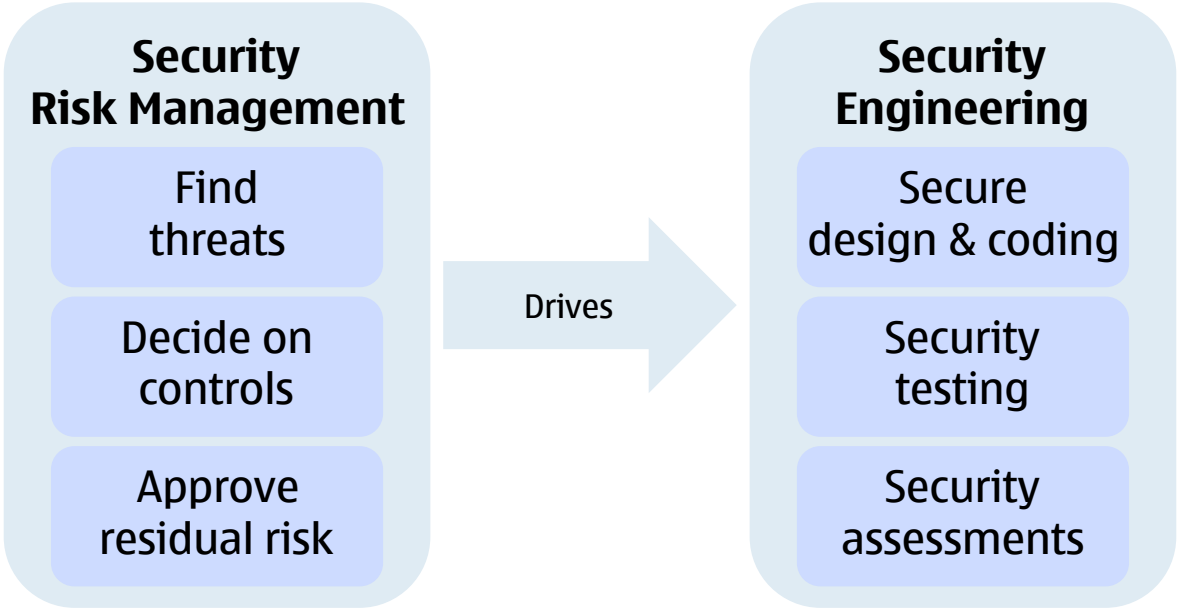
EARLY IS CHEAPER	Changes in direction are easier early on
AVOID QUEUES	Work items should not need to wait for a rubber stamp
REDUCE VARIABILITY	Make activities day-to-day, optimise for repetition
SMALL BATCHES	Small incremental additions reduce risk and do not clog the machine
RAPID FEEDBACK	Local feedback loops enable quicker reaction time to changes
DECENTRALISE	Build scalable capacity into the organisation

“Agile” is understood in many different ways. Here, an agile method is something that allows fast response to changing targets and environment. Many agile methods have aspects that are “lean”. Lean processes aim to produce a constant flow of work, which can be steered quickly towards new targets, and that aims to minimise any wasted work when the target changes.

Aspects that slow down response time are, for example, work queues, where a lot of work is waiting on a gate or a milestone, congesting the upstream and starving the downstream of work, high variability, which means that the flow is not smooth, causing congestion and ripple effects, and centralisation, which does not allow quick adjustment to locally changing conditions.

If you have a lean and agile process, adding security management into the process must also be lean. Otherwise it will destroy the lean properties. On the other hand, if the security management activities are lean, adding those activities to the development process is low-risk. This talk is exploring the choices we have looked into for introducing lean product security management practices.

If you want to know more about lean development, a recommended book is by Reinertsen (see the last slide).



3

“Security” can also be approached from different angles. Our view of product security is usually split into two spheres: managing the product security risk, which actually means making sure that the security-related residual business risk is on acceptable level, and security engineering.

Having said that, the risk management does cross into engineering:

- Finding threats involves technical security threat analysis (some call this threat modelling or risk analysis).
- Deciding on controls usually involves a level of technical design decisions.

Security engineering activities are driven by risk considerations. Not always explicitly though – for example, in clear cases, security testing and input validation can be done without requiring an explicit mandate from risk analysis.

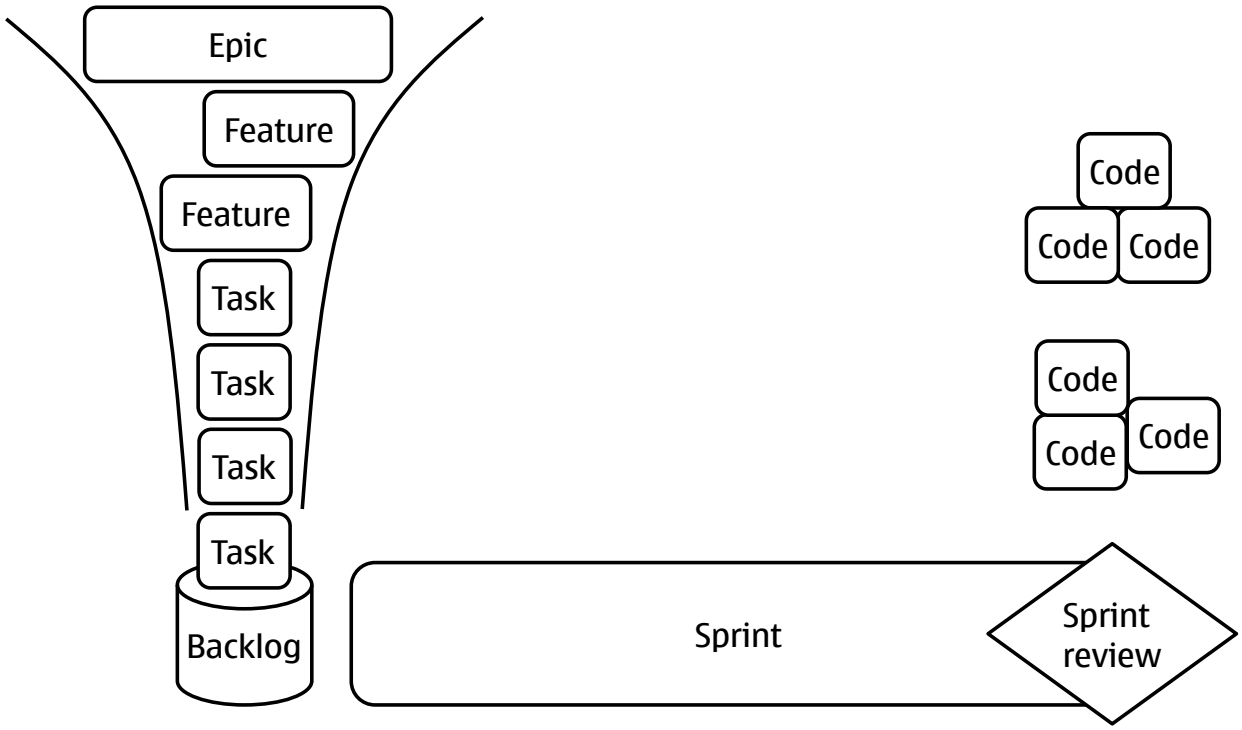
This talk is about the left side

But yes, you also need the secure coding & testing part

This talk concentrates on the risk management part, not the engineering part. In essence, we are looking at

- How to set up product management so that security risks and security feature needs are identified and addressed effectively and early, without destroying the lean properties,
- Who make the control decisions (requirements / design / coding / testing) in an agile product management setup, and
- Who can approve residual security risk in an agile product development setup.

Still, every team must have competences to do secure software development on the design, implementation and testing level. However, this is irrespective of the software development lifecycle methodology that is used; it holds true in waterfall and agile.



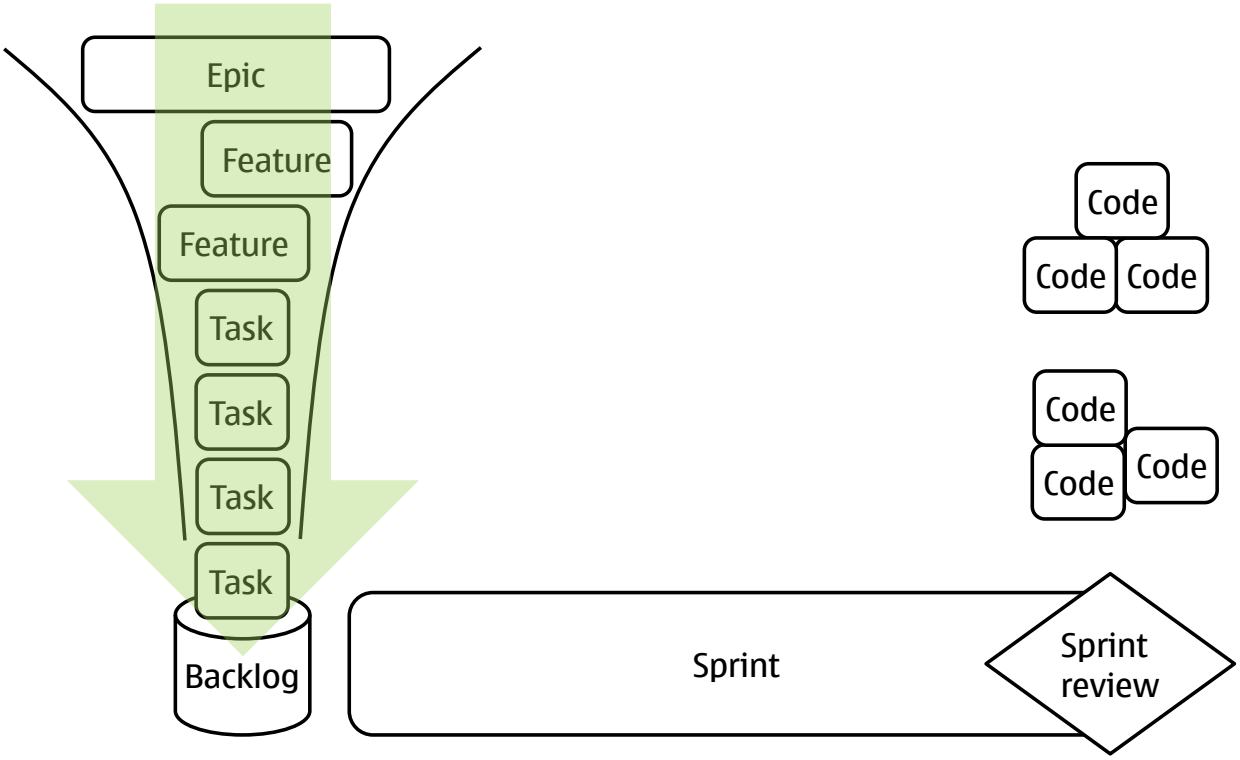
5

This is a generic picture of a typical agile product management setup. The flow is from top left to right, in a U-shaped curve.

It has three parts: Requirements management on the left, development at the bottom, and releasing on the right. Let's consider all three parts in more detail on the following slides.

Note that these are not phases. All parts are running concurrently. Think of it like a picture of a machine, where the requirements management "funnel" constantly populates the prioritised backlog, and the development teams consume items from the backlog and integrate code into the end product.

Although this model seems to be popular, it is possible that your agile product management model looks nothing like this. The lean principles should, however, be applicable.



6

The requirements management part is also known as requirements decomposition.

It contains a “requirements funnel” on the left. A customer (aided by a *Product Owner*) feeds the funnel with high-level user stories, often called *epics*. These are typically of the form

“as a user, I want *foo* so that *bar*”,

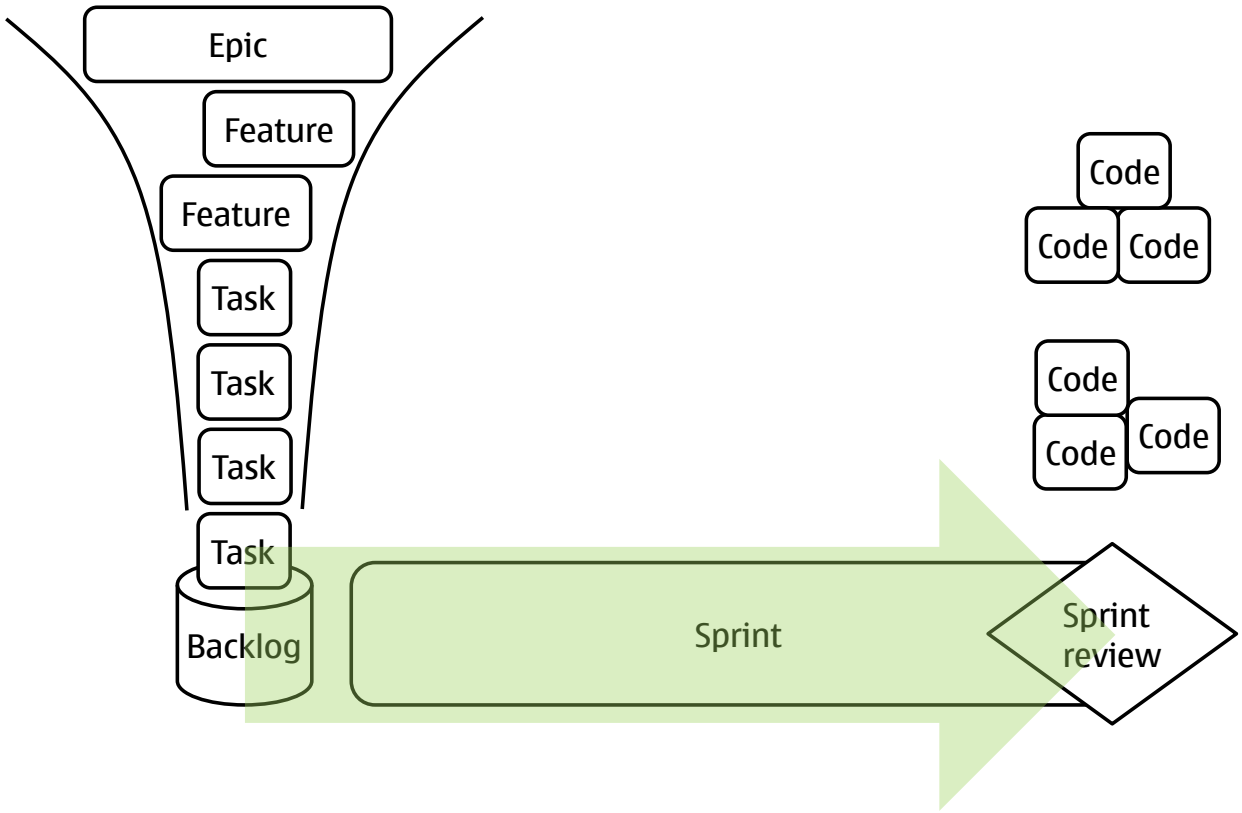
where *foo* is functionality or a more general user experience and *bar* is a value-adding rationale.

These user stories (an example, “as a user, I want to have a user account so my settings are persistent”) get decomposed or split into smaller things, such as functional features (“system must have a registration page”), and finally they decompose into *product backlog items* or tasks (“implement email validation”). Typically, a larger story decomposes into a number of smaller stories or tasks.

The number of intermediate levels for the user stories varies. The decomposition is usually driven by a Product Owner, aided by architects and designers (typically, the closer it gets to actual task definition, the more the implementation team may be consulted). A product owner is the customer-facing representative and also the business owner for the product.

In larger setups, there may be several Product Owners for parts of the complete product (in some cases, one Product Owner per team).

More specific explanation of this can be found, for example, in *A Lean and Scalable Requirements Model for Agile Enterprises* by Dean Leffingwell and Juha-Markus Aalto, and in a book by Pichler (see the last slide).

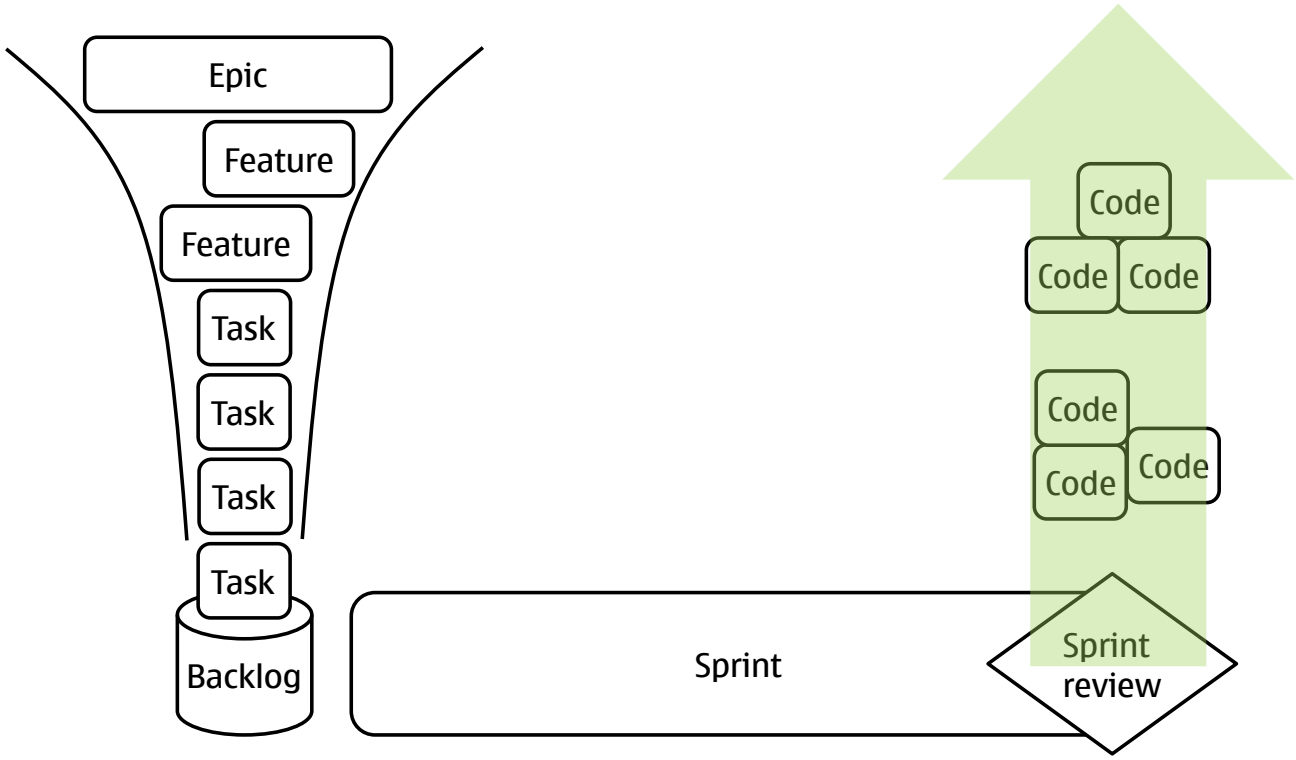


7

In the implementation part, the development team takes the (currently) highest-priority backlog items, as prioritised by the Product Owner, and implements them.

This does not need to be a Scrum sprint – other methods, such as kanban, can probably be used here as well. It's just that we have tried this out with Scrum.

A Scrum sprint ends in a *Sprint Review*, which is a quality gate for the sprint. It has a very important role in the product security risk management, which we are going to see later.

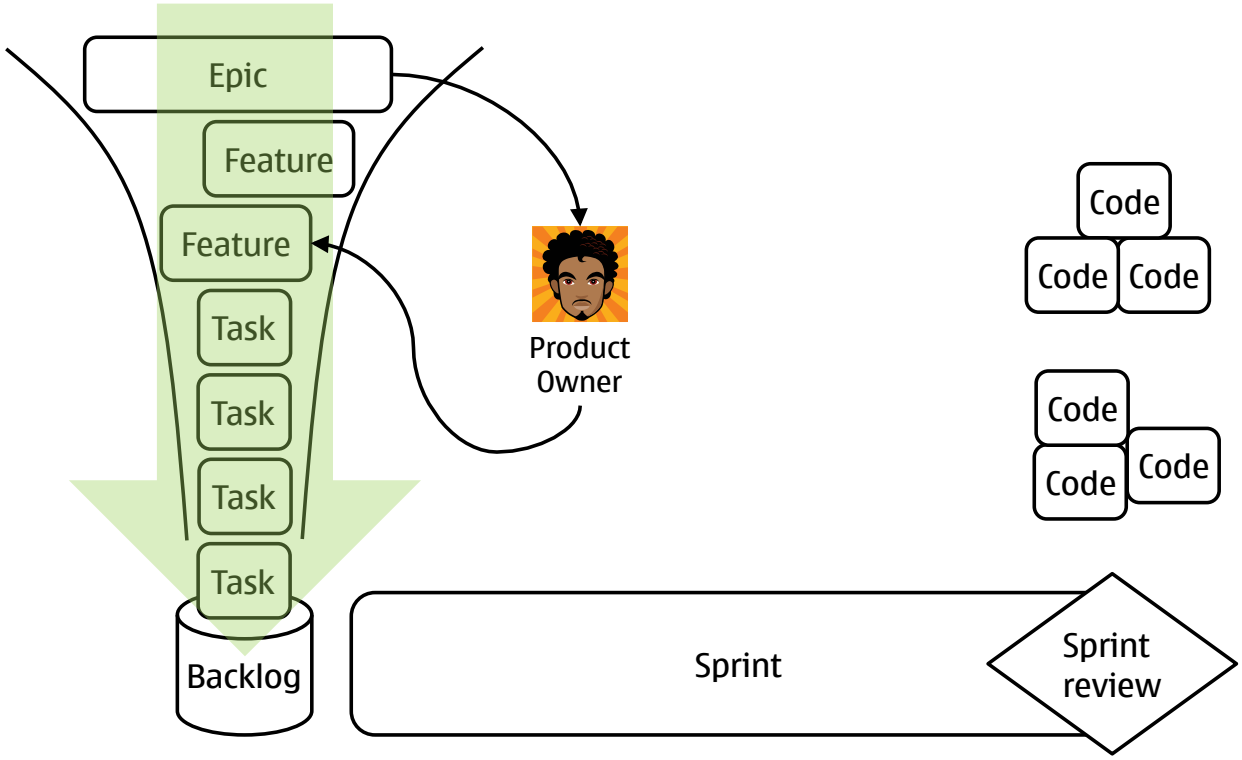


8

The third part is the releasing part. Code gets integrated (possibly in Continuous Integration) and released in internal and/or external releases.

The aspects in this talk mainly cover the first two parts – decomposition and implementation. However, the Product Owner has a critical role in approving the residual risk based on the actual delivered code.

Requirements Engineering



10

Just a reminder – this is the requirements decomposition “funnel”.

A Product Owner is there, hard at work, decomposing higher-level user stories into more specific features and tasks.

The customer is not shown here, but is assumed to be represented by the Product Owner.

Tools for requirements engineering phase

- **Tool:** "Security story" templates
  - Generic prototype user stories on security aspects.
  - Should mirror your (client's) threat landscape.
  - Provided with interpretation examples.
  - Short list – perhaps a laminated one-pager.

- **Tool:** Abuse cases ('attacker stories') allowed as requirements
  - The Product Owner does not need to understand *how*.
  - "As an attacker, I should not be able to…"
  - Translated into positive requirements by designers later.

We have two tools for the Product Owners:

1. *Prototype user stories* that address security aspects. These are kind of templates expressed in the form of generic security-related user stories, such as "As a user, I want my data to be secure so that nobody can misuse it". These should be written so that they fit your application's or client's business area. Each story is provided with interpretation examples for specific features – for example, a specific functional requirement that would fulfill that user story for a credit card payment usage scenario. These examples are not meant to be copied directly into the requirements but they are instead intended to lead the Product Owner by example.The idea is that this list is short (because the number of requirements is large and time is scarce), and that it is balanced between being a checklist and a completely free-form list of threats. The main idea is that the security experts have made this list short and actionable, but it still contains templated stories that are generic enough so that they cannot just be copied without thinking.
2. A second tool is actually just a permission and a policy decision for the requirements: the Product Owners must be permitted to create *negative "attacker stories"* (abuse cases, as they are often known in the literature) in addition to positive requirements. This is important, because the Product Owners may lack the security expertise to be actually able to say how to solve the problem; they can, however, probably say what should not happen.As the stories are decomposed into more specific features and tasks later with the designers and architects, the technical experts can interpret them in as technical positive controls.

A note on *non-functional requirements*: In many cases, a security requirement identified in decomposition is naturally a non-functional requirement, such as a testing need or a quality need. These should be translated into functional requirements whenever possible. For example, a testing need can be translated into a task like "create 100.000 fuzz test cases and make them run in the unit test framework". This treatment of non-functional requirements is related to making work visible. Further discussion on that follows later.

An example

A generic security story template

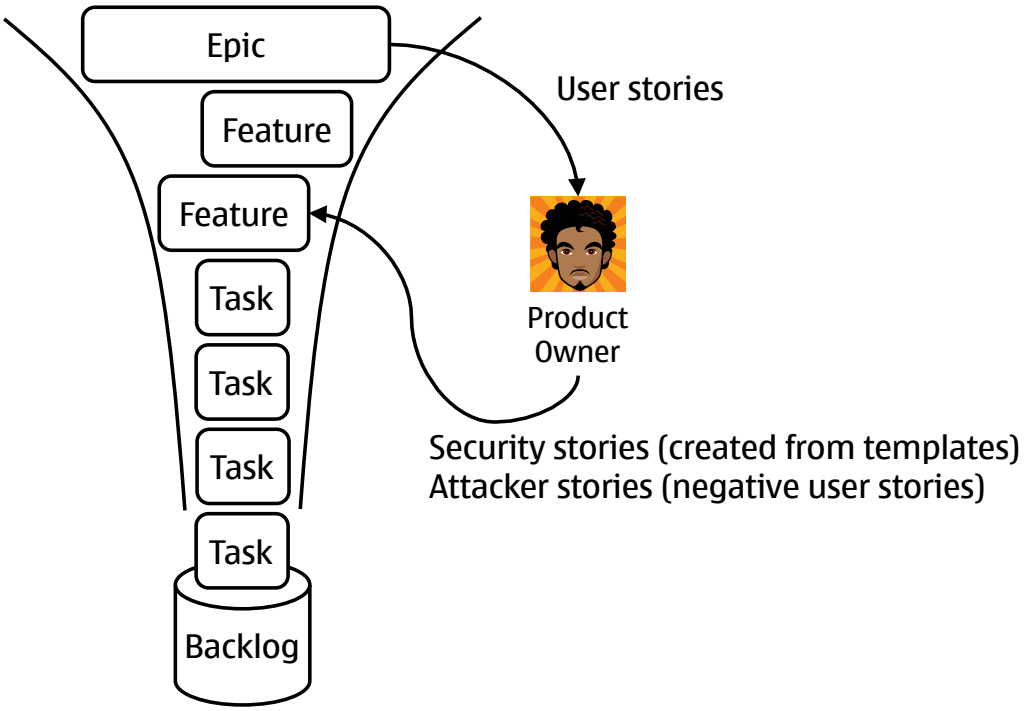
**“As a user,
I want to be protected from unintentionally or accidentally sharing personal information.”**

would be transformed, for example, in a photo sharing application into

**“As a user,
I want that geolocation information is removed from EXIF metadata unless I explicitly indicate that I want to share it.”**

This is an example of a privacy-related security story and how it could get adapted by the Product Owner when decomposing requirements for a sharing application.

Security stories seem to be well suited to privacy requirements as they are user-facing. Security features which are purely concerned with the under-the-hood plumbing are not as easy to write out as user stories, and would perhaps be better addressed in the design stage.



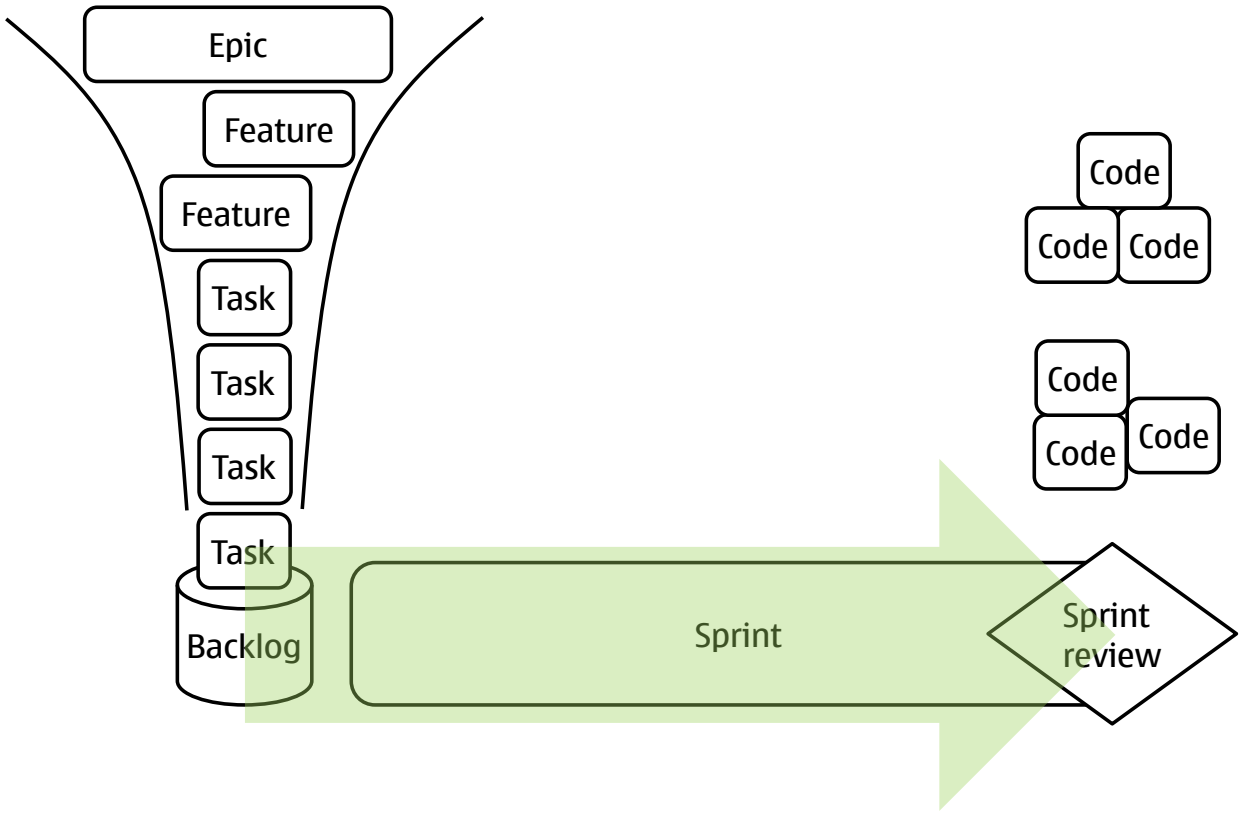
13

So, what is expected to happen in the decomposition phase is that:

1. The Product Owner gets user stories (of any level, they can be already decomposed);
2. The Product Owner uses the general security story templates to create new requirements specific to this specific feature;
3. The Product Owner has a permission to write the new requirements also as negative, “attacker” user stories.

The target is that this becomes business as usual for decomposition of any requirements. As the list of security user story templates is short, the Product Owner should, after a while, be able to do this naturally, without necessarily having to revisit the list any more.

Development



15

During development, the highest-priority product backlog items (tasks) are implemented by the team.

Even though we talk about Scrum sprint here, there is no reason why other models, such as kanban, couldn't be used instead.

However, one of the key control points in Scrum is the existence of a *Sprint Review*, which ends each sprint. A Sprint Review is a quality gate where the team has the option of deciding whether they are "done" or "not done" from the quality perspective.

In kanban, this quality gate could be modelled as one of the kanban "states".

Tools for development phase

- **Tool:** Security criteria in Sprint Review's Definition of Done
  - "Given the found threats, are we *done* from security p.o.v.?"
  - Team must have true power to say "no we're not".

- **Tool:** Introduce a security threat analysis into Scrum sprints
  - With training and guidelines, obviously.

- **Tool:** Have a simple filter which is able to flag potentially risky tasks
  - Should mirror your typical past or anticipated problems.

There are three tools that we've looked into for managing security in Scrum sprints.

1. If the team is using Scrum, the first tool is to *add security into the Definition of Done criteria* for a sprint. This set of criteria is evaluated by the team at every Sprint Review. If they think they are "not done", the team should have the option to declare that the tasks haven't yet been finished. This empowers the team to make quality (and hence security) decisions. Essentially this is the lowest level of residual risk acceptance.
   A suggested question for the Definition of Done is:
   "Given the security threats we have identified, can we say we're done from security perspective?"
   Note though, that it is important that the team is genuinely allowed to say they're "not done". If their financial incentives are tied to ticking things "done" in the sprints, do not expect any security issues to be addressed either.
2. The Scrum team must do *technical security threat analysis* (threat modelling). Actually how they do it is not discussed here – it could be data flow analysis with STRIDE, free-form brainstorming, or whatever works best for you. However, we will discuss the scheduling and resourcing such work on the next slides.
3. The Scrum team should also have a *task screening filter* which helps them to flag those aspects that really need threat analysis. This filter should – like the security story template list offered to Product Owners – be based on actual threat profile for your client / application / organisation. This filter is used to help to schedule threat analysis when planning the Scrum sprint.

Threat analysis tasks as *research spikes*

- For every potentially risky task, add an additional task to do threat analysis
- This screening can be done during sprint planning or during the sprint
- These additional tasks are sometimes called “research spikes” – scheduled as a task (a backlog item) but do not create deliverable code
- Add controls identified during the spike to the backlog as new tasks
- Those risks that are not controlled need to be accepted in Sprint Review, as residual risk – or the sprint is “not done”

This is a suggestion we have not yet had time to get much practical experience on, but this is what we are currently seeing as the best option.

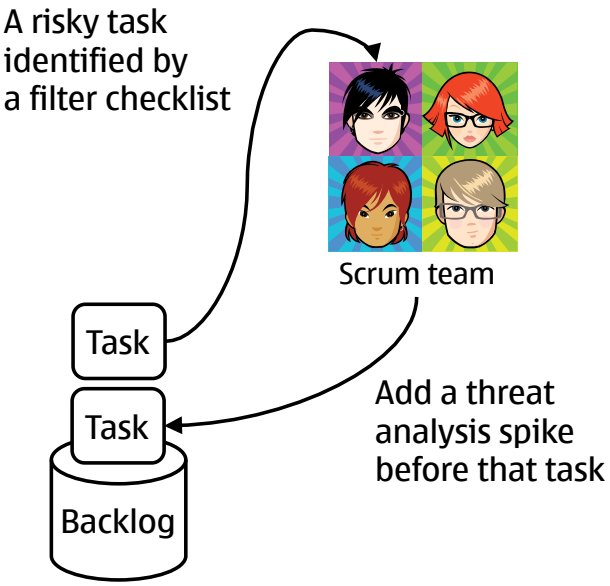
The challenge with threat analysis is that if it is treated as “extra” work, it gets easily optimised away. Also, assuming that every sprint would put time aside for security threat analysis moves this work creates overhead which is invisible from task scheduling viewpoint.

Therefore, the current assumption of how to do this is to make threat analysis visible as tasks on the backlog.

The agile literature offers a tool for this, called a *research spike*. A spike is usually a short task to validate a design decision, perhaps by creating a small, throwaway prototype.

So, if a task feels like it could use a threat analysis, a new task (a spike) can be added on the backlog.

Let’s see a picture on the next slide.

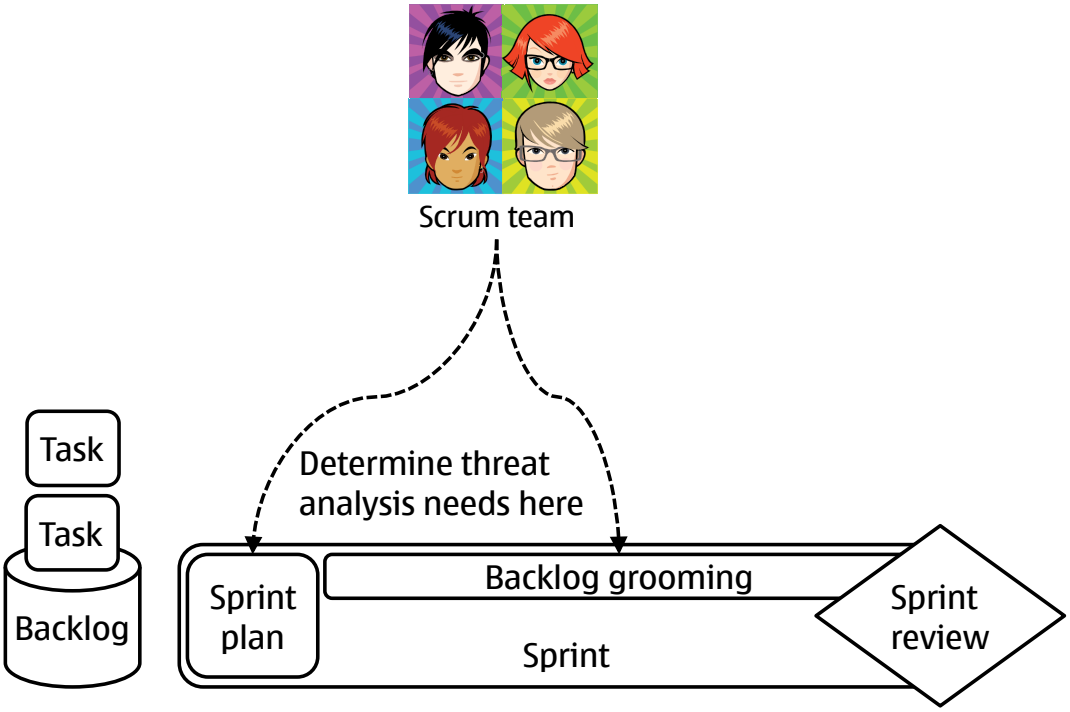


The Scrum team would have a look at each of the tasks they're about to do.

They would prescreen every task using a filter checklist, which was mentioned a couple of slides back as the third tool available to the Scrum team. If the task seems to be "security sensitive" by that list – perhaps it's about processing incoming information, creating a network connection, making a payment, and so on – the team just adds a new task on the backlog that precedes the original one.

As an example, if the task to be implemented is "email validation for registration", they would add a new task "security threat analysis for the email validation" and inject that on the backlog before the original task.

This way, the effort spent on threat analysis becomes visible on the backlog and it gets proper time allocated for it, instead of just assuming that threat analysis would just happen.



There are at least two options on when this filtering and adding threat analysis spikes can be done.

Sprint Planning is a session that starts each Sprint by populating the Sprint Backlog with content from the Product Backlog.

Backlog Grooming is an ongoing activity which can also be used for this purpose.

More discussion follows on the next slide.

When to add threat analysis "spikes"

- **During sprint planning:**
  - Deciding on research spikes can be done in sprint planning.
  - Can be based on a checklist-type filter that identifies risky tasks.

- **During backlog grooming:**
  - Some teams allocate ~10 % of time for backlog maintenance and enhancement – "grooming" the backlog.
  - Some of that time can be used to identify need for threat analysis spikes, or for actual threat analysis itself (if Backlog Grooming is actually being used for actual design).

There are at least two options on when this filtering and adding threat analysis spikes can be done.

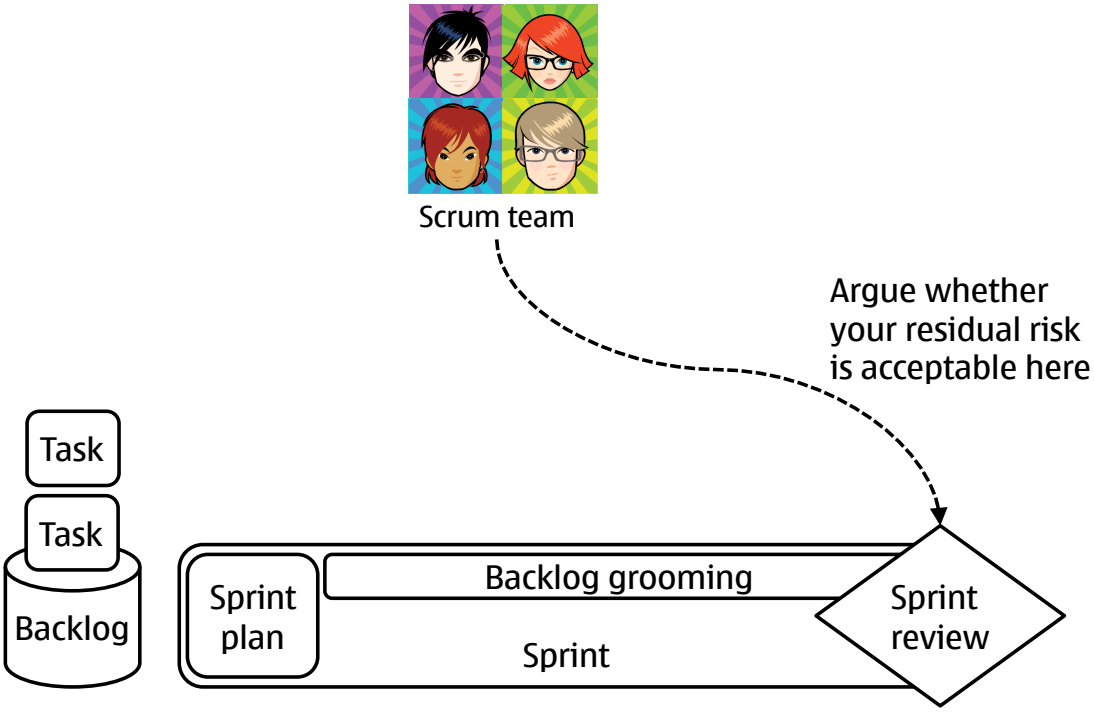
It can be done as a part of normal Sprint Planning session, where the *Sprint Backlog* (work to be done during the next sprint) is populated by taking in the highest priority items from the Product Backlog. For teams running vanilla Scrum, this is probably the most natural phase.

Note that the threat analysis itself shouldn't be attempted during Sprint Planning, as this would be just the kind of invisible overhead that we are trying to avoid. Instead, just add the new tasks on the Sprint Backlog.

Another option is to leverage a practice known as Backlog Grooming (a.k.a. Backlog Maintenance or Backlog Refinement), for which some books say the team should use 5 to 10 % of their time.

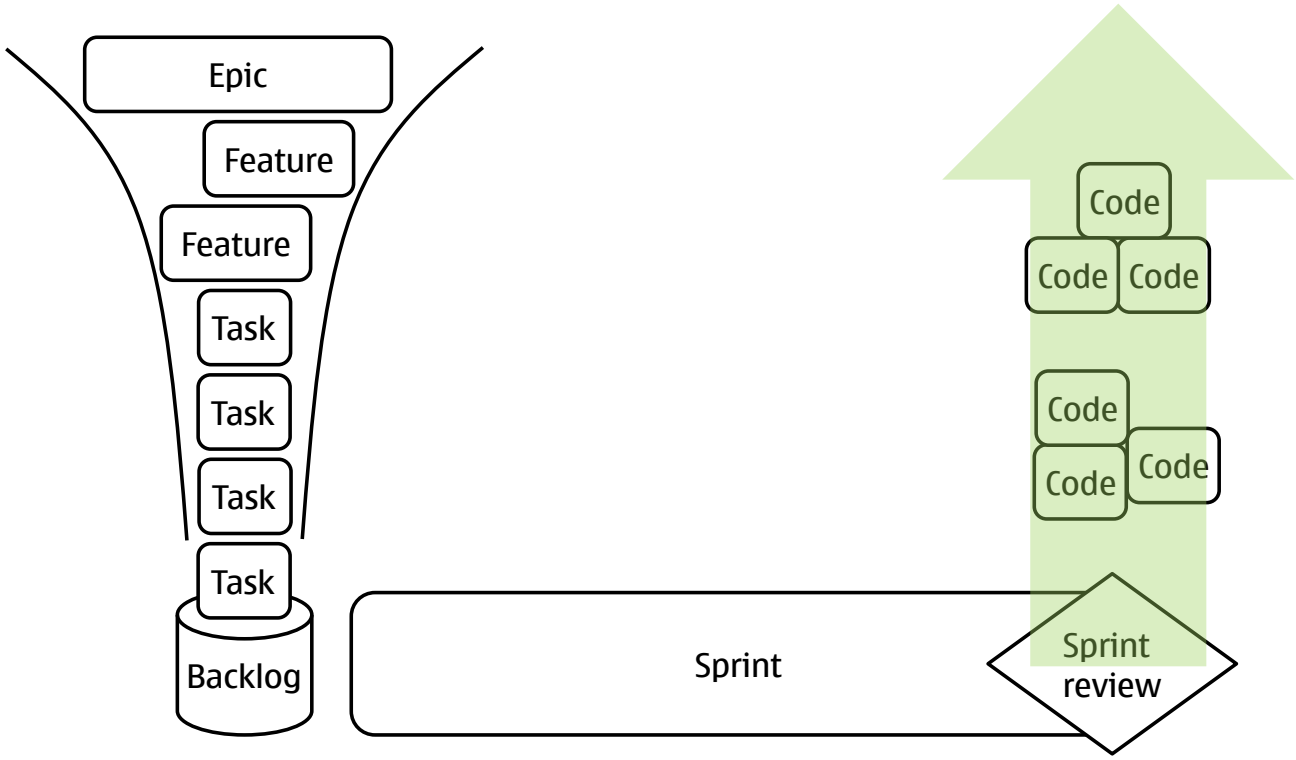
Backlog Grooming is an activity where the teams looks into the future Product Backlog items and to refine and enhance the upcoming tasks on the backlog. The idea is that the team can look further into the future than in Sprint Planning, where they are just considering the current Sprint that is about to begin.

Some teams have proposed that they would actually do threat analysis during the Grooming, but this is again something that should be really carefully considered as it will not make analysis explicitly visible and scheduled.



And finally, irrespective of how the threat analysis is scheduled, the Sprint Review should ask the big question – are we done from the security perspective.

Integration / Releasing


Epic
Feature
Feature
Task
Task
Task
Task
Backlog
Sprint
Code
Code
Code
Code
Code
Code
Sprint
review

Tools for release phase

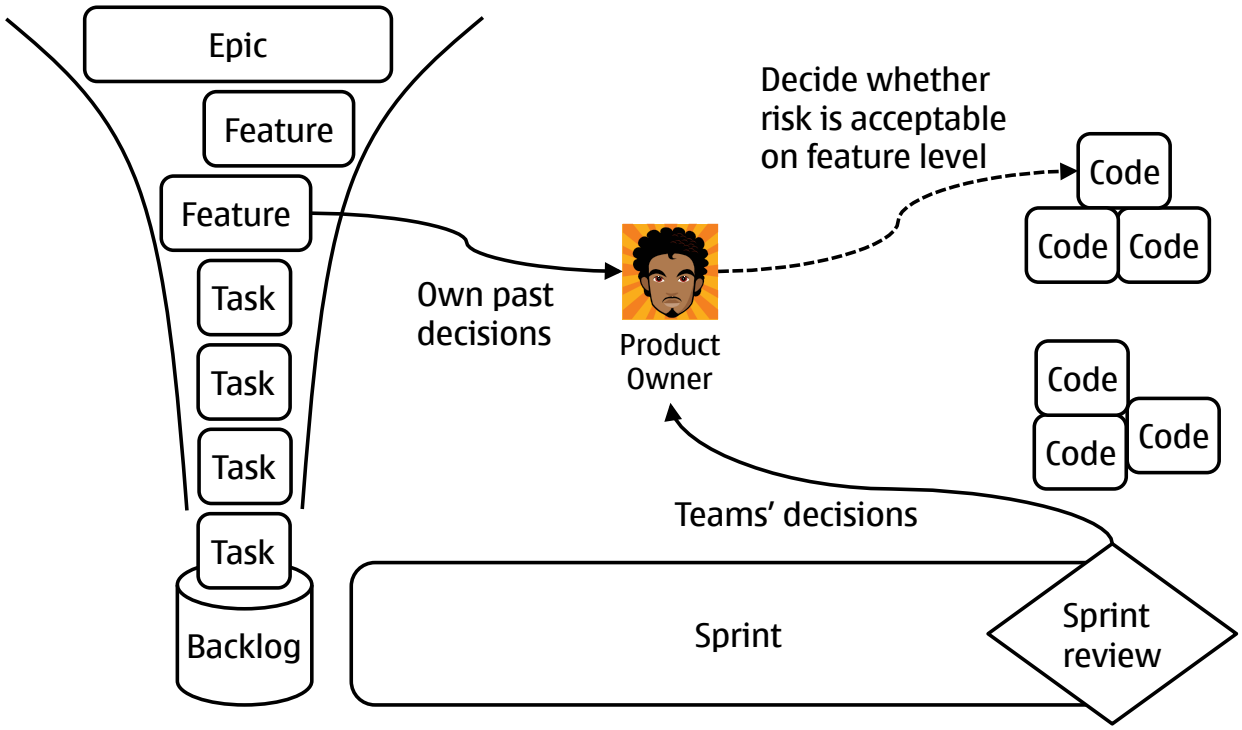
- **Tool:** If you have a regulatory requirement for an audit, try to separate those parts from the others
  - Allows for more rapid release cycle for those parts that do not need compulsory audits

- **Tool:** Product Owner should review residual risk at feature level

There are two tools we have planned here, but I should disclose that the first one is really not very applicable in the very organisation I'm at, so that is only based on discussion with others, and we haven't actually tried that one out.

The second tool, however, is critical in actually tying all the loose ends together.

1. The first tool would be to *separate those parts that are subject to compulsory audit* and approval from the rest, so that the rest can be released with a tighter schedule. This may require architectural rework.
2. The second tool is that the Product Owner should consciously *approve any residual risk*. This is discussed on the next slide.



25

So, remember that when the requirements were being decomposed and split, the Product Owner identified new security needs.

Also, remember that during the implementation, the teams assessed whether they are "done" with the tasks from security perspective at each Sprint Review.

Now, the responsibility of the Product Owner is to determine:

1. Have all the security features identified in the requirements management phase actually been "done"?
2. Did the implementation teams raise any residual risks that they haven't been able to address?

Combining this information, the Product Owner should make the business decision of accepting any remaining residual risk. If it cannot be accepted, the Product Owner should add new tasks on the backlog to take care of those risks. (Residual risk can only be controlled or accepted. They cannot be rejected or ignored - that would equal to accepting the risk.)

Note that I have (on purpose) not talked about how the information of the risks and tasks flows in the system. This is because every organisation has a different set of tools to support this work. In the most simple form, this could be a wiki page with a list of risks and the controls or acceptance decision listed for each, which both the team and the Product Owner edit.

Reflection and Suggestions

So is it lean?

EARLY IS CHEAPER	Major security needs identified already in requirements decomposition
AVOID QUEUES	Control decisions and risk approval done locally, not in a committee
REDUCE VARIABILITY	Threat analysis is business as usual. It gets optimised and natural
SMALL BATCHES	Security issues are addressed one user story or task at the time
RAPID FEEDBACK	Two local (tight) risk acceptance loops – on team and Product Owner levels
DECENTRALISE	Every team & Product Owner does it. Scales naturally, no single failure point

Let's compare how this way of security management would fit the lean targets.

- Most systemic and large security features would be identified early on, because requirements management is made security-aware.
- There is no "security committee" that would need to be consulted on risk decisions. The Product Owner makes requirements-level decisions and the implementation team makes the implementation level decisions, with Product Owner then validating or vetoing those. Only if those persons feel they cannot make a decision, they should escalate it – and this should be towards the business owners. But in most cases, they can get rapid feedback locally.
- Threat analysis is done for every feature and every task, and spread out in the organisation to every Product Owner and team. However, it is heavily driven by templates that streamline the work, and it is done in small pieces. This means that the work gets optimised as the teams learn how to do it effectively.

Make work visible

- Think of agile and lean processes as a microkernel
  - It is only a task scheduler
- Everything that needs to be done should end up as a task on the backlog
- Or have other type of explicit time allocation
- Avoid encumbering the process with implicit work
- Makes security work and compliance costs visible and measurable

One of the core principles here is to make all security management (threat analysis, security documentation) work visible.

You could think of an agile process as a microkernel. It provides just task scheduling. A waterfall-style RUP could be compared a monolithic kernel that provides all kinds of services, but with overhead.

When threat analysis is treated as tasks, it won't get optimised away or just checked off as a checklist item. It also allows measurement of costs related to security.

Visibility principle applied

- You need a specific document for compliance or regulators?
  - Make that a task (a backlog item).
  - Do not expect them to implicitly materialise.

- You discover a non-functional requirement?
  - Express that as positive functional task (a backlog item).
  - Example: “Interface X needs to be robust” could translate into “Create fuzz test cases for interface X and add them to the Continuous Integration test system”.

A couple of examples of how this visibility principle can be applied.

If a regulator requires a specific security compliance document to be written, make that also a task on the backlog (and not something that the process should deliver “on the side”). The work needs to be done anyway, so why not make it visible.

Non-functional requirements should also be transformed into positive tasks (as much as it is possible – I have heard arguments that it is not always possible, but then again, I do not believe for a second that a note stuck on the monitor telling the developer to “remember security!” is going to help, either).

Dedicated security persons

- You can task one person in a team to be the "security conscience"
- For example, in Sprint Review, is responsible for asking aloud whether the team is really done in the security sense
- Should not, however, be single-handedly responsible for security
- Volunteers are probably the best for this role

One concept we've played with is a dedicated security person, which we did internally call a *"White Hat"*.

This person was not intended to be solely responsible for security, and especially threat analysis should be a teamwork. However, it is useful to have someone who asks the question whether the team is "done" from security perspective in Sprint Review, and generally someone who has a licence to pester people about security.

It seems that it would be best to pick a person who volunteers for this role as pestering others about security requires some sort of real commitment in the long term. If someone is interested in pursuing information security as their speciality, but such full-time jobs aren't immediately available in the company, this could also prove to be a useful career tool.

My statistics aren't hard data, but it currently feels like teams who did nominate this sort of person tend to find more issues earlier in their products. However, this may be just correlation (of being security-aware in general) and not a causal relationship.

It won't cost much, though, so probably useful to try out.

More to read

- A very good book on lean process development. No fluff. Applicable to almost any product creation process.
  - *The Principles of Product Development Flow: Second Generation Lean Product Development*, Donald G. Reinertsen. Celeritas Publishing, 2009.
- A concise summary on what Product Owners are – and should be, and what to look for when appointing or recruiting Product Owners.
  - *Agile Product Management with Scrum: Creating Products that Customers Love*. Roman Pichler. Addison-Wesley, 2010.
- Results of the secure software development and agile methods workshop held by Agile Finland and Finnish Information Security Association in April 2010, which crystallised many of these thoughts
  - http://confluence.agilefinland.com/display/af/Secure+software+development+and+agile+methods+-+notes

I have nothing to do with the books or authors listed here, they're just useful ones I've found.

If you want to share any experiences or feedback, I can be contacted at:

antti.vaha-sipila@nokia.com

Or, you can find full contact information including my private email address and GnuPG keys at

http://www.iki.fi/avs/contact.html