**APPLICATION SECURITY PROGRAM: DEFINITION OF SUCCESS IN AN AGILE WORLD**

**TL;DR:** How to design metrics and goals for an Application Security Program, embracing agile methodology and surviving to tell the story.

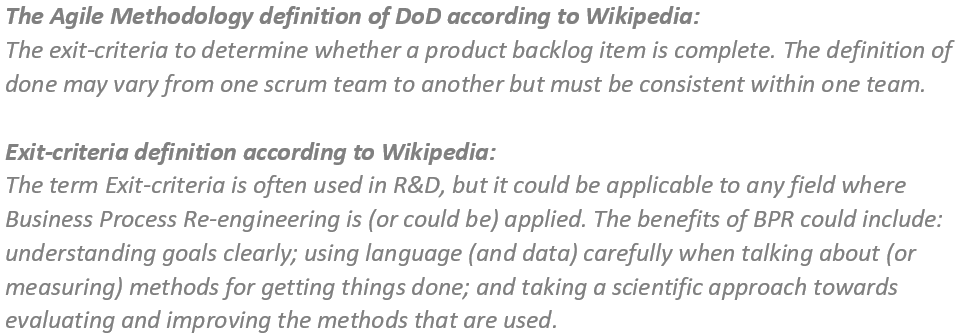
### .....

The Application Security program is not something that will ever be complete, it's not something that has an end: it's built upon a process and it injects security principles into the development process.

By nature it is a service, always changing to adapt to new challenges like new vulnerabilities, new languages, new frameworks, new libraries, new versions of existing libraries, new ways of developing, and new ways of delivering software in production.

We must adapt yet again to a relatively new way of delivering software development, and we need to define the parameters of success for our new application security initiative.

Can we apply some of the Agile methodology concepts to our Application Security program? For instance, can we use the Definition of Done/Condition of Satisfaction to help us with the Definition of Success?



So, to determine if "something" is complete we need to define/identify the properties of the completeness of this "something", which in our case should be translated in success of this "something. These properties will create the exit-criteria against which any instances of the implementation of this "something" will be measured against.

In a matter of speaking we need to define a metric system which will have the purpose of telling us if the "something" we created is fit for its PURPOSE.

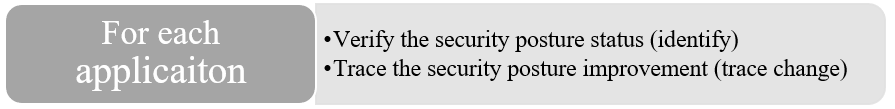


## Now what's the purpose of an Application Security Program?

Using broad strokes, we can summarize the purpose with:

Verifying and improving the security posture of any application implemented by the organization […] in a cost-efficient way.

Let's focus on the first part and simplify the problem (the second part will be discussed on a later blog article on cost-efficiency):



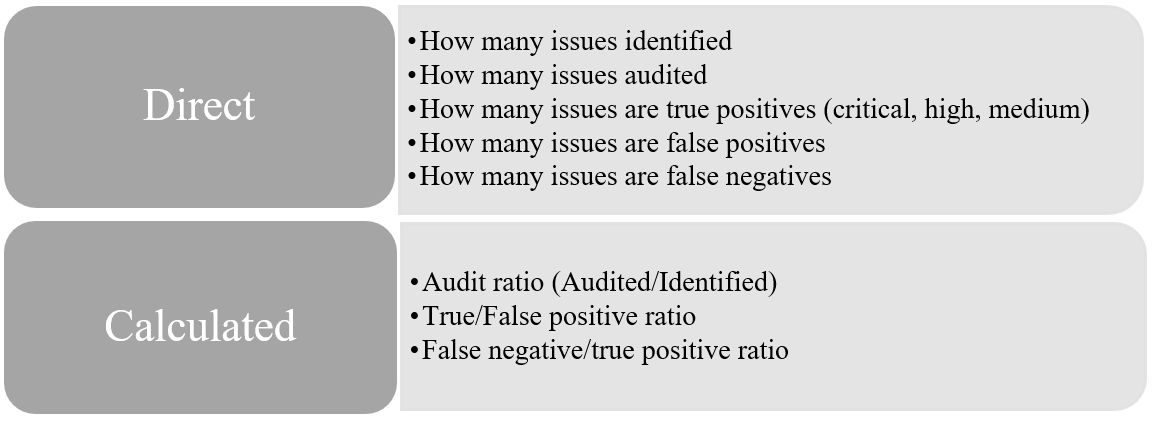
The simplest way of measuring the security posture of a component is by counting the number of issues for that particular component. This can be enriched with the properties of the issues like severity and category.

So, the security posture of an application/service (architecture) is represented by the security posture of each software component used to implement the application/service.

It follows that from an Application Security Program perspective, the security posture for an organization is represented by the security posture of each application/service hosted/used by the organization.

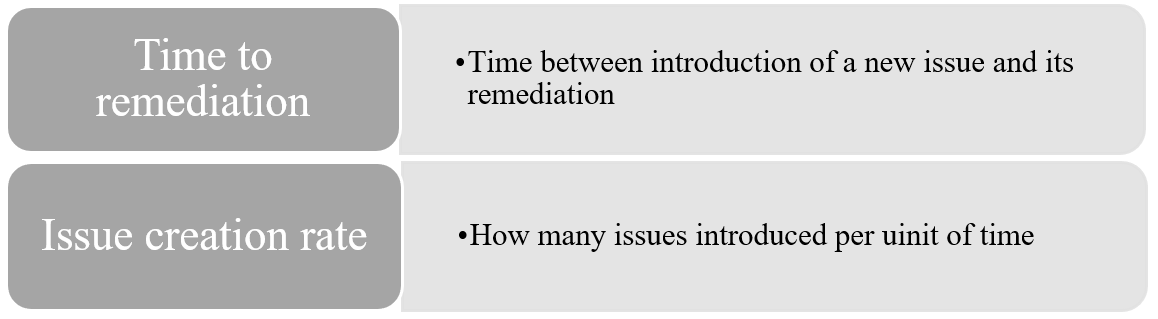
### How to verify the security posture

The first part of the purpose requires the identification of the security posture, so we need to find a way to measure "how well" the Application Security Program is able intercept/identify that set of information.



### How to trace the security posture improvements

The second part of the purpose is about tracing the evolution of the system, so how those measures that define the security posture change over time. In other words, we need to find a way to measure "how well" the Application Security Program performs, so what is the effectiveness of the effort spent.

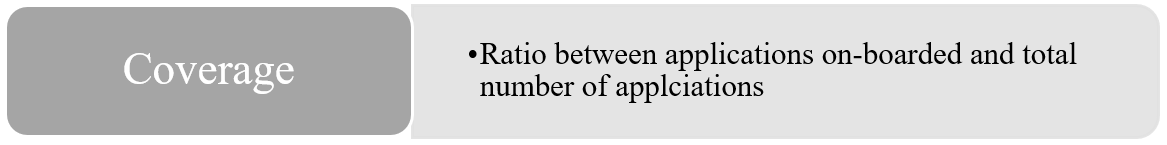


Time to remediation: We need to know how much time goes between the introduction of a new issue and its remediation. So, the architecture should be able to trace when an issue is introduced for the first time and when it was fixed.

Issue creation rate for each stage of the development life-cycle (design, development, implementation, execution): Since flaws are unavoidable, it's very important to measure the rate at which vulnerabilities are introduced and at which stage they are intercepted (design [threat modelling, security architecture design review], development [sast, dast, iast, sca], implementation[pt, config review], execution [pt, va, patch management, siem]).

### How to define the Application Security scope

The purpose of an Application Security program is not just limited to a single application or a single software component, instead it applies to every application/service (architecture) hosted/used/delivered by the organization. So, we need to find a way to measure "how well" the Application Security Program represents the organization in terms of coverage.

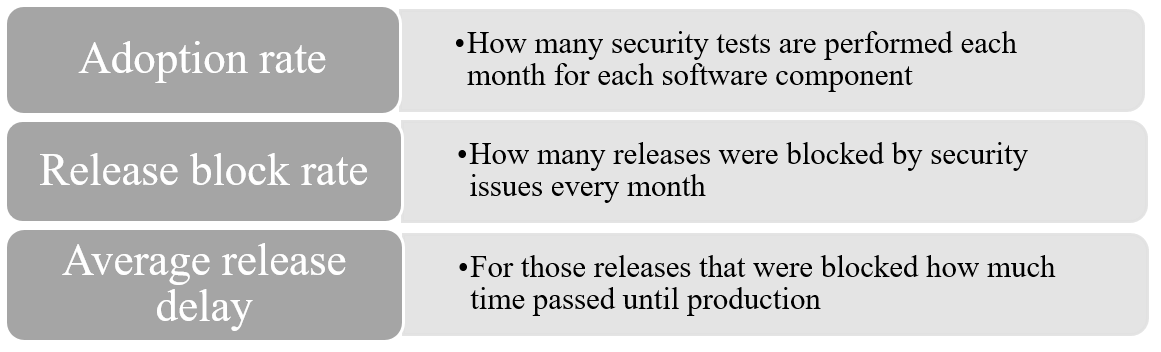


We need to be able to know how many applications/software components are in scope, so that we can answer any coverage questions like: what is the program/application/service coverage level? 0%? - 100%? Something in between? Is everything relevant?

This metric is tightly coupled with the existence of an application inventory, but this doesn’t mean the Application Security Program should not start until an inventory is completed.

## How to deliver a holistic view

To bind the tree aspects of the Application Security Program purpose and to measure the performance in an aggregated way we can define two more metrics:



We need to know how often the systems that implement the Application Security program are used. To do that we can measure the number of automated security tests per month for each software component.

Release block rate and average delay: We need to verify how many releases were delayed or blocked by security issues. These metrics are supposed to be an alarm bell and will help in identifying any disconnections between business needs and security policies that must be acknowledged and dealt with.

## How to define the Application Security Program goals

Now that we know the purpose and what information we need to collect to measure the level of our success, we need to decide where to put the measuring stick. We need to set the goals for each metric.

The goals, as commonly used, must be:

 SMART: Specific, Measurable, Achievable, Realistic, and Timely.

We already defined what can be measured, therefore we got Specific and Measurable covered already. We must take care of Achievable, Realistic and Timely, simple! Probably not easy though. We need to define what can be Realistically Achieved in an arbitrary Time interval.

This is just a blog post, so we cannot examine all the different shades and nuances of possible ways to define the goals, so we will explore two different ways of starting that process. The Realistic and Achievable aspects of the goals imply knowledge of the current state.

* Finger in the air based on years of experience with the SDLC implemented by the organization.
* SDLC maturity assessment ([OpenSAMM](https://www.opensamm.org/" \t "_blank))

The first method is as valid as the second, as long as we already have a good and solid idea of the Software Development Life-cycle (SDLC or SDL) maturity level.

The point is not how we gather the information that build our knowledge about the organisation's security processes. The point is that the knowledge must be as complete as possible, and as unbiased as possible.

The only problem my inner architect/engineer has with the first approach is around the unbiased aspect. I guess everything boils down to scientific method and the need for evidence supporting the claim that the knowledge itself is good in the first place. As long as we can prove the quality of that knowledge, how we got there is not very relevant.

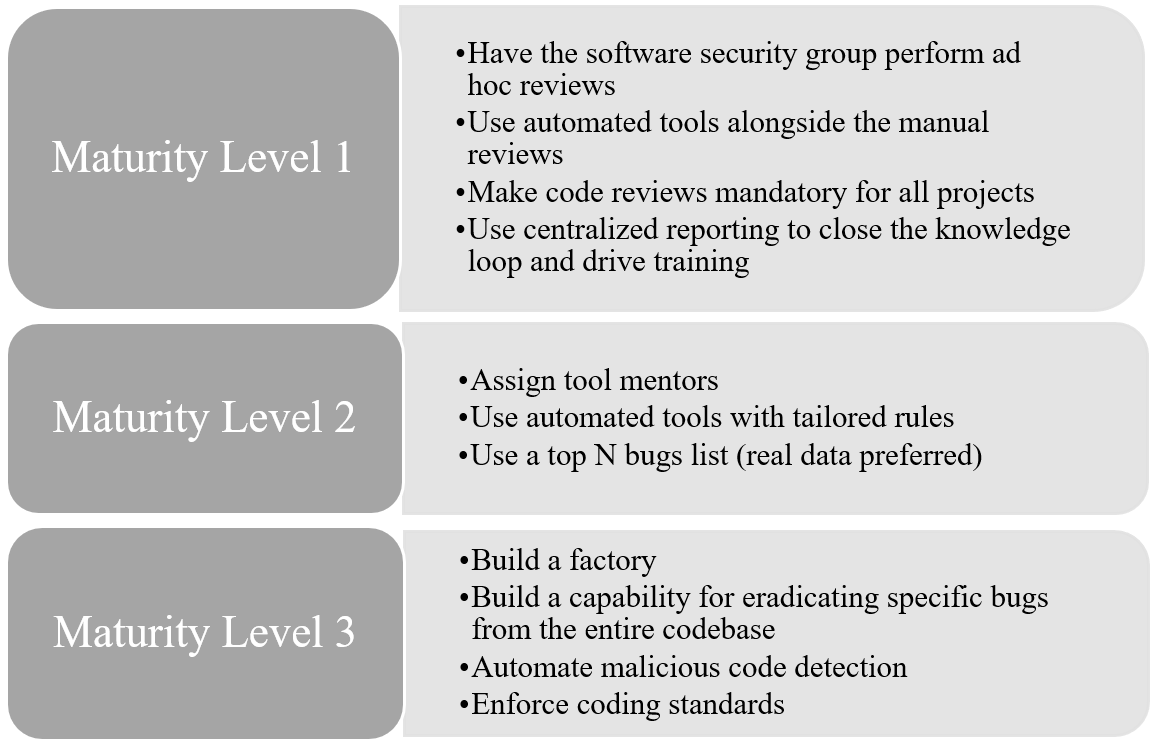
At this point we have an idea of which set of data is required to make an informed decision about the goal, but we are still missing the goal. This goal (or better, goals), to an extent, will be always arbitrary and self-referential.

To address the problem of being self-referential, we need to match our goals with what other organizations are doing, for example, using [BSIMM](https://www.bsimm.com/) report or other similar reports.

A good way to mitigate the arbitrariness is provided by the ability to correct the goal over time. We need to remember that the decision is based on data that change over time, so the goal (and the data that lead to that decision) must be reviewed over time to see if they are still relevant.

Depending on the organization maturity we can have different goals, and therefore the decisions pertaining the goals should not be cast in stone but re-evaluated on a regular basis.

We can consider the example provided by the latest BSIMM9 to identify the evolution of the goals that must be achieved to get to a specific maturity lever for Secure Software Code Review:



We need to consider that each of those achievements that would categorize our Appsec initiative maturity level must be evaluated in terms of effectiveness.

Let's examine two possible cases: one has a maturity level 2 with the following properties:

* We perform some code reviews (when we can)
* Sometimes we use automated tools
* We have in our policies that every project must have a code review
* We put the pdf reports in our SharePoint/Jira.

The other has the following properties instead:

* Every application prior go live is code reviewed --> application coverage 100%
* Every code review happens automatically on a weekly basis --> 100% automation
* Every project can go in production only if it has no security issue --> 100% issue free or 0 risk exposure
* Each scan result is delivered automatically to the developer group responsible for the scanned codebase. Aggregated security performances are reported monthly to the Tech leads and Project managers. --> 100% visibility

While the first example is probably very realistic and honest for any Application Security initiative that has just started, it's vague, so definitely neither specific nor measurable. The second example is, by and large, unrealistic and unachievable for an initiative that has just started.

What can be done to make the second example achievable, more specific, measurable, and realistic? How about adding time to the equation?



## Time changes everything

Let's review the goals with a realistic and achievable timeline and scope:

* Every business-critical application must be code reviewed prior go live - work towards a 100% coverage in 12 months;
* Every code review for business-critical applications must happen automatically multiple times during the development pipeline - work towards a business-critical application automated pipeline in 18 months;
* Every new Critical issue in production must be traced and remediated in 2 working weeks - work towards a remediation SLA;
* Each scan result must be delivered automatically to the developer group responsible for the scanned codebase - work towards report automation in 18 months.

Because of the complex and mutating nature of the problem of setting goals, to reduce it to the constituting components we can try to apply the AGILE methodology: split the bigger problems in Themes/Initiatives/Epics and Stories.

## Agile Application Security Program: Themes-Initiatives-Epics-Stories

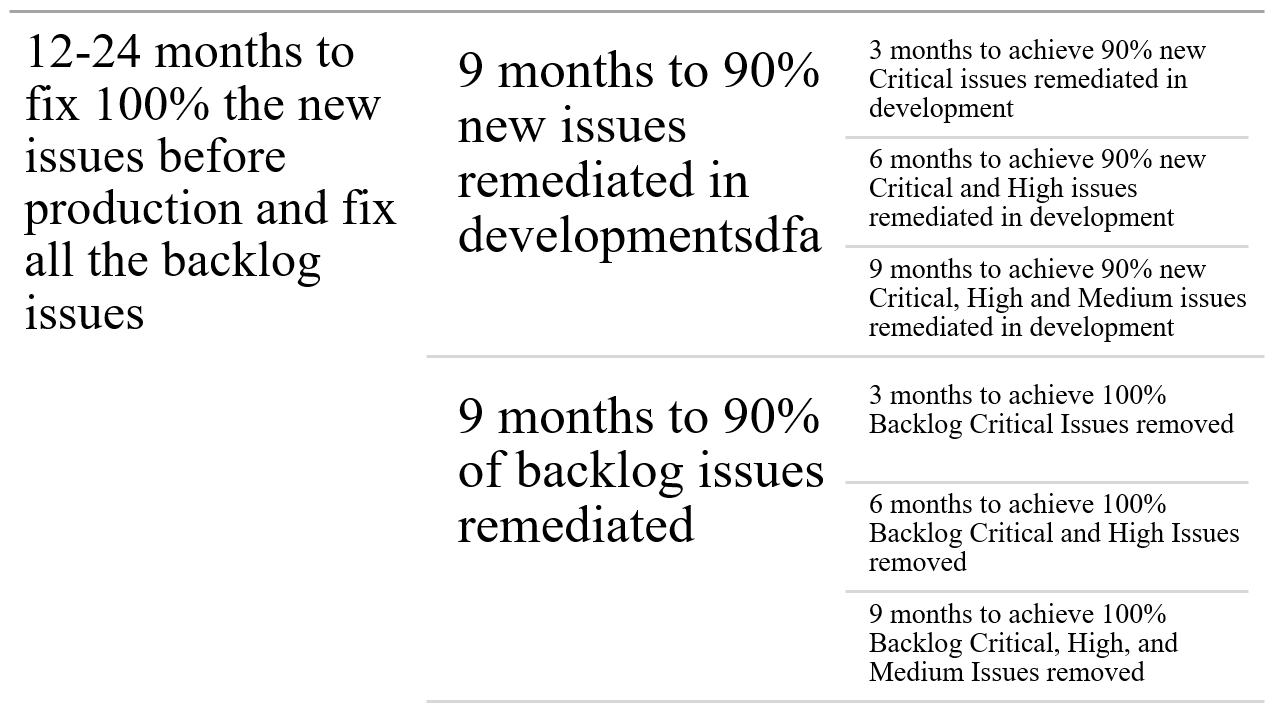
Themes are the super challenging goals like "100% automated appsec with 100% application coverage". The Themes comprise one or more Initiatives.

Initiatives are broad descriptions of aspects of a Theme, for example: "100% of business-critical application scanned automatically by the end of the year". The Initiatives are composed by Epics.

Epics describe the high-level view of some specific Initiative objective, for example: "build an automated code review system". The Epics are composed by Stories.

Stories represent the atomic properties that compose an Epic, for example: "implement a process that gets the code from a repository and trigger the static code analyzer engine".

Some examples of Themes/Initiatives/Epics goals could be:



Preparing a single set of goals that suites every organization is impossible. Each application security initiative will face unique challenges specific to the organization it is working within.

Security teams that are facing these challenges for the first time will find it difficult to reach a mature level immediately, therefore they should not rush through, but they should instead take the steps one at a time, sprint by sprint, to improve their approach to goal definition and implementation.

It is not only possible to hammer "the security ways of doing things" into an agile environment. **The best way of succeeding in doing that is embracing the methodology itself**. I will cover both challenges and rewards of adopting an agile approach in future articles.

I really hope that this first blog entry can be useful to the ones that just started their journey in Appsec.

Disclaimer: the views and opinion present on this post are those of the author and do not necessarily reflect the official policy or position of any other agency, organization, employer or company.