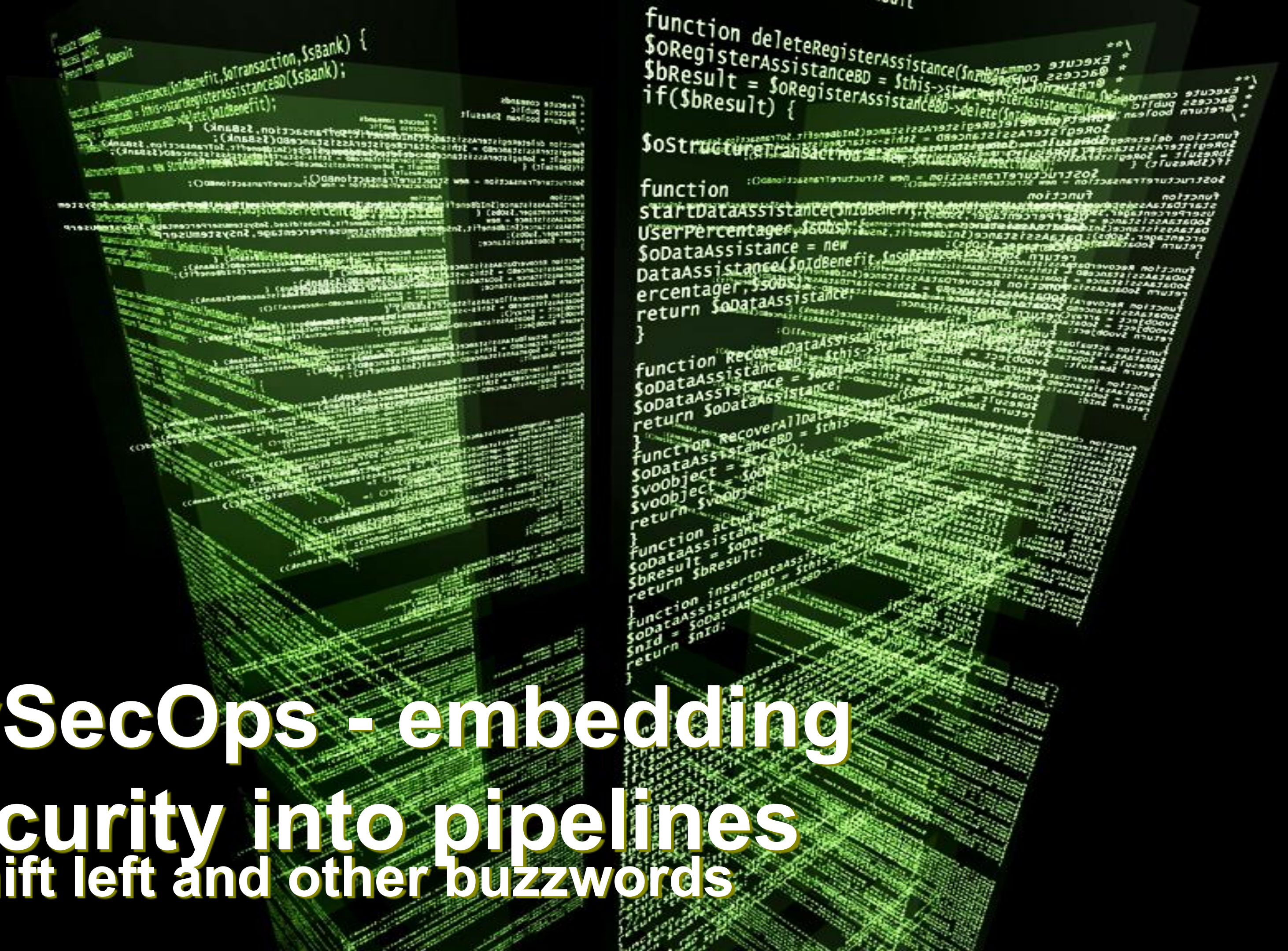


DevSecOps - embedding security into pipelines

shift left and other buzzwords

whoami

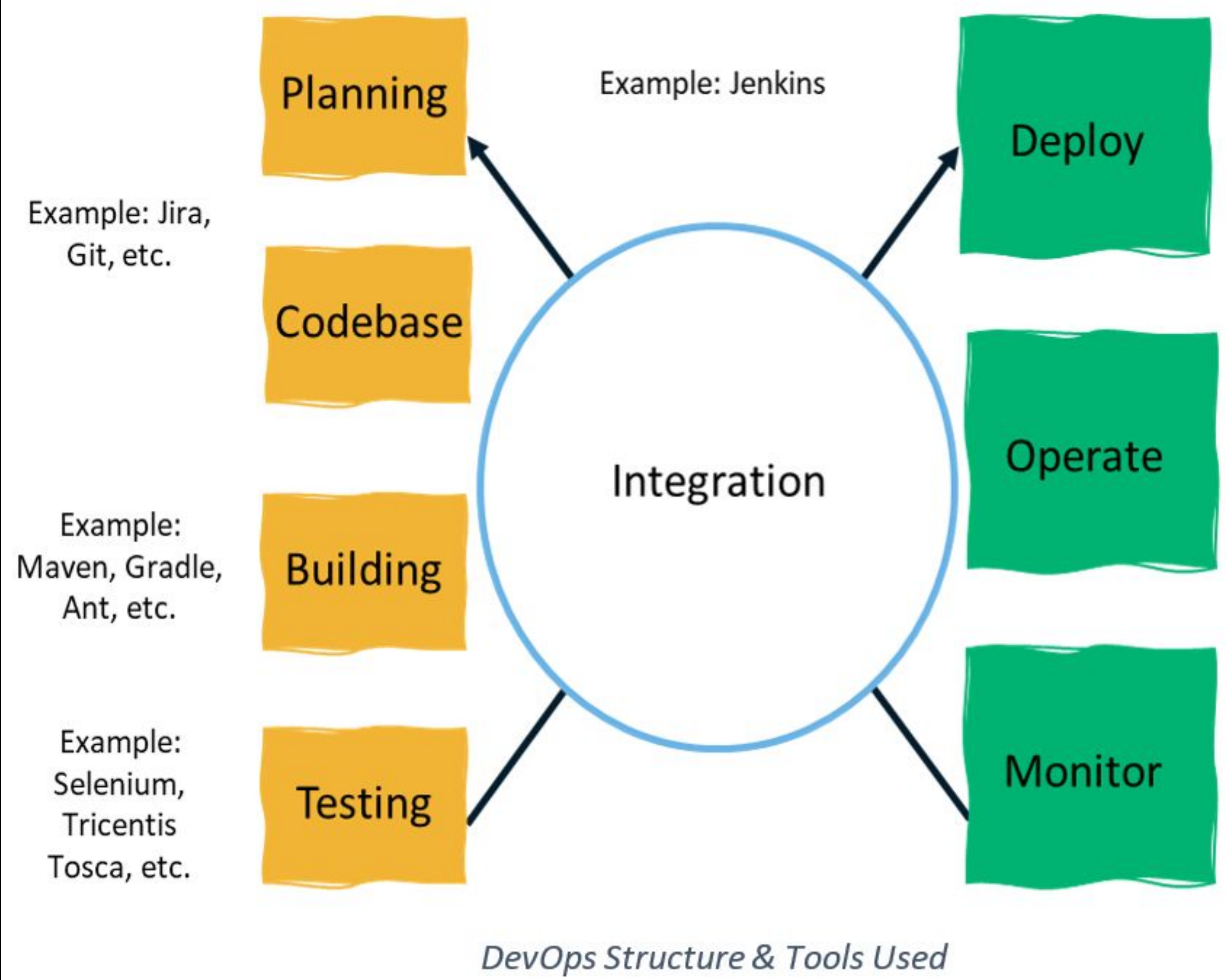
➜ ~ : whoami

Svet - DevSecOps engineer working for Natural Intelligence. For contacting me - please use sbalevski@protonmail.com

- 8 years in IT, 4 in security and counting. Worked as a security engineer/ architect/ consultant for some of the largest companies in Europe.

What is DevOps

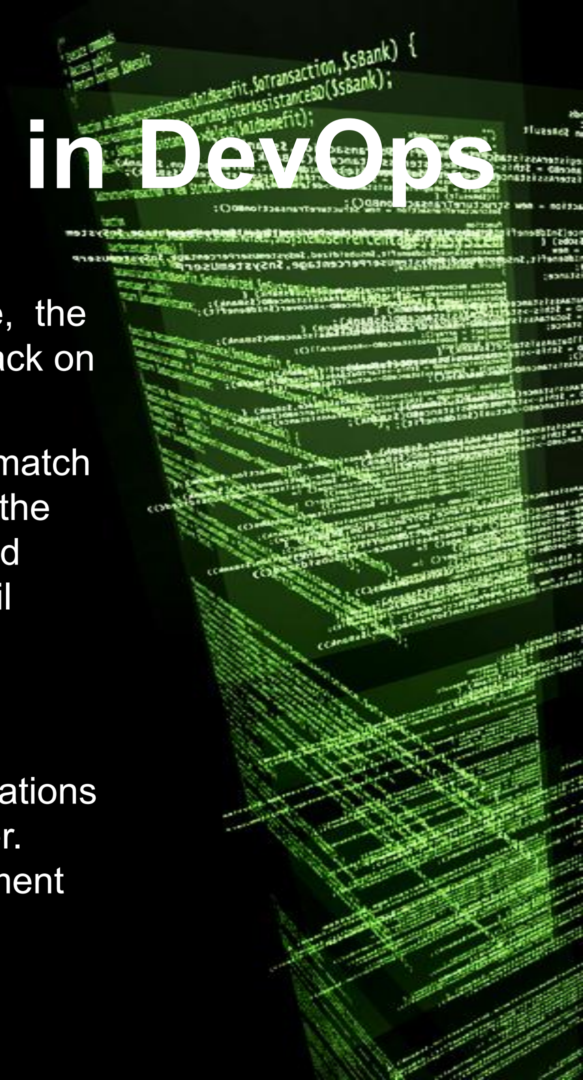
DevOps is a **philosophy or practice** that brought together all the processes in software development from coding to deployment in one cycle, so now, instead of working in silos, they work towards the same end goal of moving into production at a faster rate, with a feedback mechanism incorporated into the whole process.



DevOps Structure & Tools Used

Source: https://sectigostore.com/blog/wp-content/uploads/2020/04/devops-structure-and-tools.png

Famous Buzzwords in DevOps



Continuous Integration - each time a developer checks in a code change, the system is automatically built and tested, providing fast and frequent feedback on the health of the code base.

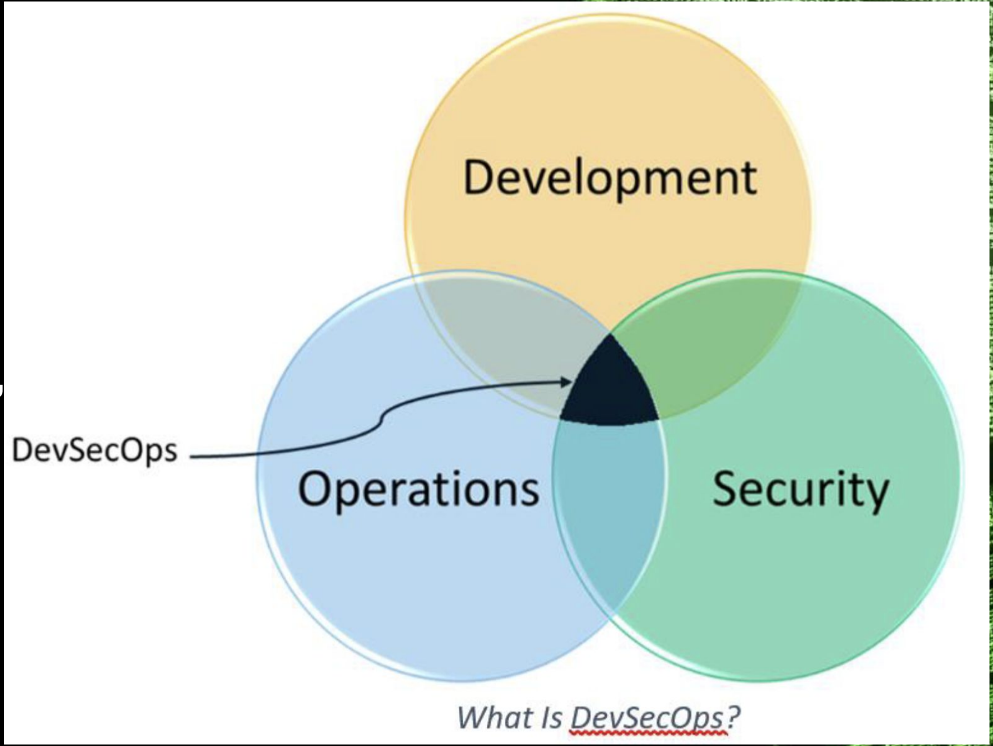
Continuous Delivery - provisioning and configuring test environments to match production as closely as possible—automatically. This includes packaging the code and deploying it to test environments; running acceptance, stress, and performance tests, as well as security tests and other checks, with pass/fail feedback to the team, all automatically; and auditing all of these steps and communicating status to a dashboard. Later, you use the same pipeline to deploy the changes to production.

DevOps pipeline - a set of practices that the development (Dev) and operations (Ops) teams implement to build, test, and deploy software faster and easier. One of the primary purposes of a pipeline is to keep the software development process organized and focused.

But what is DevSecOps?

DevSecOps is the integration and automation of security as at every phase of the software development lifecycle, from initial design through integration, testing, deployment, and software delivery. Security

becames shared responsibility since the beginning (instead of the usual - “We will take care on a later stage”)

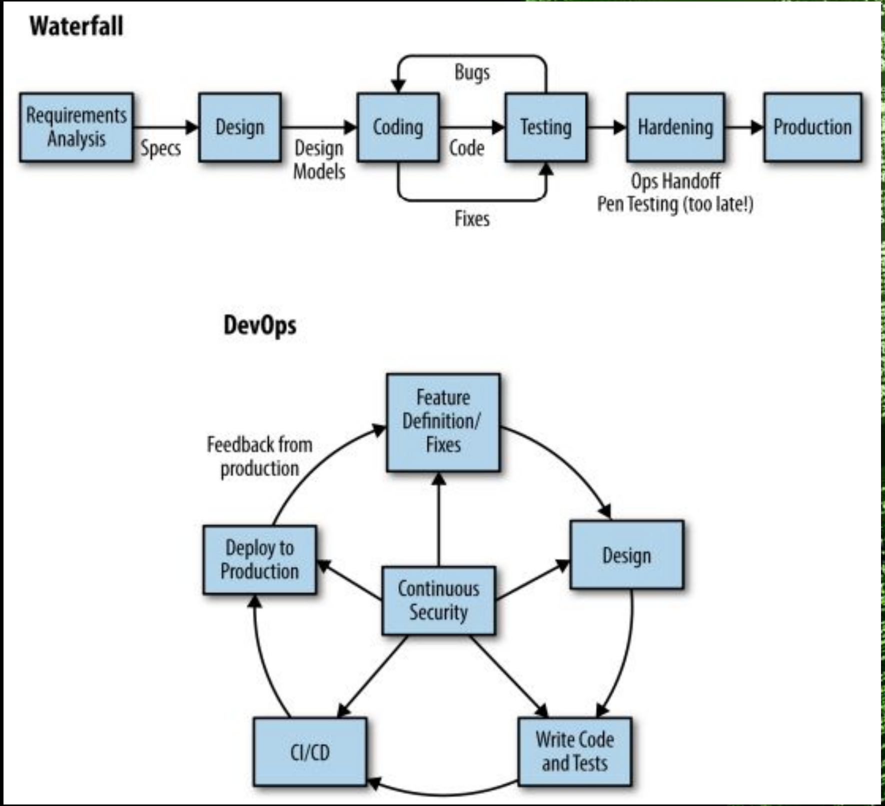


Key Challenges

1. **Deployment speed** - in 2020 Amazon have deployed **23000** times per day (or 1 deploy / 1.3 seconds)
2. **Lack of clear design considerations** - Often design is at the code. Starting with MVP and adding incremental changes is great from product perspective but we don’t have a clear design to apply shiny security design reviews.
3. **Fail early, fail fast** - why we should test code for security if it will be thrown away eventually?
4. **Microservices** - code is code in the end and microservices are not the problem. But the communication between them makes the connections far more complex leading to increased attack surface.

Another buzzword - shift left. What is this??!

In the fast moving DevOps world, for security there is only one chance to be in a same plain with Continuous Delivery. We need to start with security in the same time we are starting with the development. A.K.A. **Continuous security**



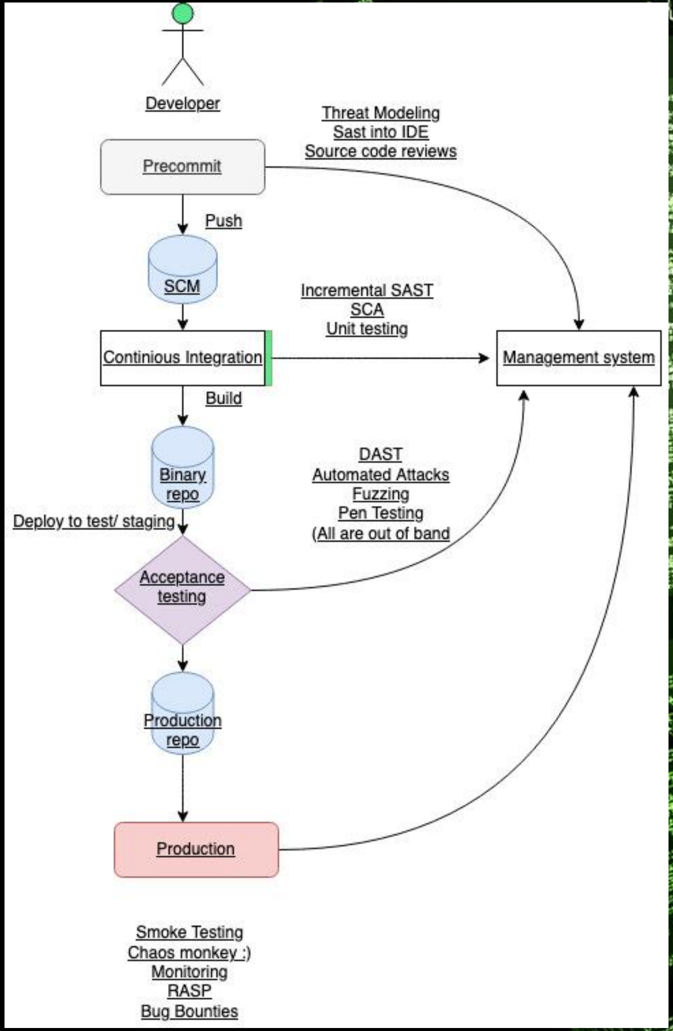
shell-b22[illegible]86c30f6

This sounds great! But it is only a picture, isn’t it? How to achieve it??

1. **Security by default** - first stop is to use frameworks and templates. Mandatory for everything related to encryption (but this is well known, isn’t it?)!
2. **Security is self-service**- the role of security engineer is not to block the development teams until something is secure. Our role is to enable development teams to do their job. Use as much self-service as possible - the tools should be simple, integrated in the IDE as much as possible.
3. **IaC**- the more, the merrier (of course it should be tested for security as well)
4. **Small incremental changes**- small scope and isolations makes them safer
5. **High speed of delivery?!** - Believe it or not, the fast changing target is much harder to being hit by the bad guys. All of those scanning, recoinance, … takes some time. And changes during that time could get potential attackers back to square one (a.k.a.“Honeymoon Effect”: older software that is more vulnerable is easier to attack than software that has recently been changed.)

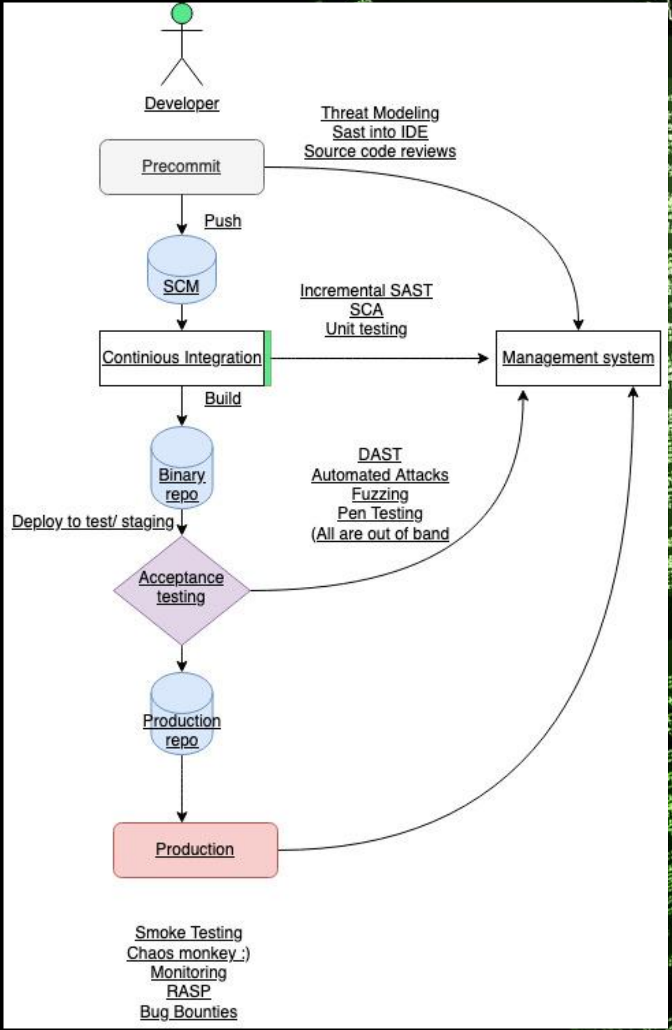
Security into CD 1/2

1. Before commit
   - **Threat Modeling (iterative) - architectural level**
   - **SAST / SCA integrated into IDE**
   - **Security code reviews**
2. Commit
   - **Full blown SCA**
   - **Incremental SAST**
   - **Security unit tests (if such exists)**
   - **Digital signing of the binaries**



Security into CD 2/2

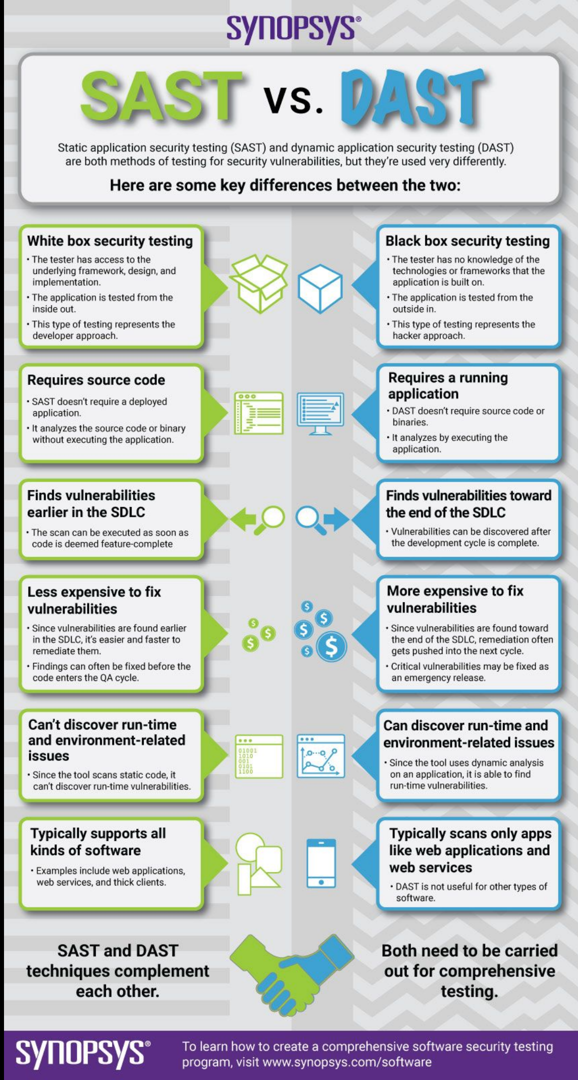
3. Acceptance
   - **Security smoke tests**
   - **Targeted DAST**
   - **IAST**
   - **Deep SAST (out of band)**
   - **Fuzzing (out of band)**
   - **Manual Pentest (out of band)**
4. Production
   - **Runtime defences (RASP, WAF)**
   - **Bug bounties**



Most well-known tools: SAST vs DAST

SAST - static application security testing. This is linter like white-box testing tool. Checks the source for vulnerabilities. Fast method with false positives. Can be part of SCM and don’t need deployed system

DAST - dynamic application security testing. I attempts to automate human pentester. Slow but without False-positives (by definition). Requires deployed code but it should **not be** used on production (a lot of DB queries with invalid data). Not suitable for all applications, for more complex apps with not standard authentication can be used in semi-automatic mode, with manual intervention. Mostly running out of band (due to time constraints).



SYNOPSYS®

SAST vs. DAST

Static application security testing (SAST) and dynamic application security testing (DAST) are both methods of testing for security vulnerabilities, but they’re used very differently.

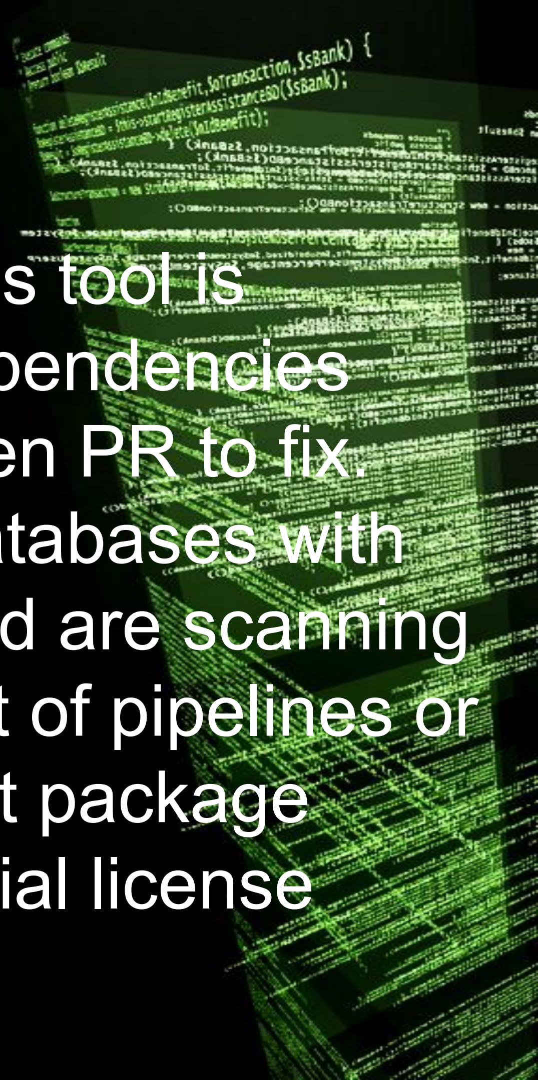
Here are some key differences between the two:

SAST	DAST
White box security testing • The tester has access to the underlying framework, design, and implementation. • The application is tested from the inside out. • This type of testing represents the developer approach.	**Black box security testing** • The tester has no knowledge of the technologies or frameworks that the application is built on. • The application is tested from the outside in. • This type of testing represents the hacker approach.
Requires source code • SAST doesn’t require a deployed application. • It analyzes the source code or binary without executing the application.	**Requires a running application** • DAST doesn’t require source code or binaries. • It analyzes by executing the application.
Finds vulnerabilities earlier in the SDLC • The scan can be executed as soon as code is deemed feature-complete	**Finds vulnerabilities toward the end of the SDLC** • Vulnerabilities can be discovered after the development cycle is complete.
Less expensive to fix vulnerabilities • Since vulnerabilities are found earlier in the SDLC, it’s easier and faster to remediate them. • Findings can often be fixed before the code enters the QA cycle.	**More expensive to fix vulnerabilities** • Since vulnerabilities are found toward the end of the SDLC, remediation often gets pushed into the next cycle. • Critical vulnerabilities may be fixed as an emergency release.
Can’t discover run-time and environment-related issues • Since the tool scans static code, it can’t discover run-time vulnerabilities.	**Can discover run-time and environment-related issues** • Since the tool uses dynamic analysis on an application, it is able to find run-time vulnerabilities.
Typically supports all kinds of software • Examples include web applications, web services, and thick clients.	**Typically scans only apps like web applications and web services** • DAST is not useful for other types of software.

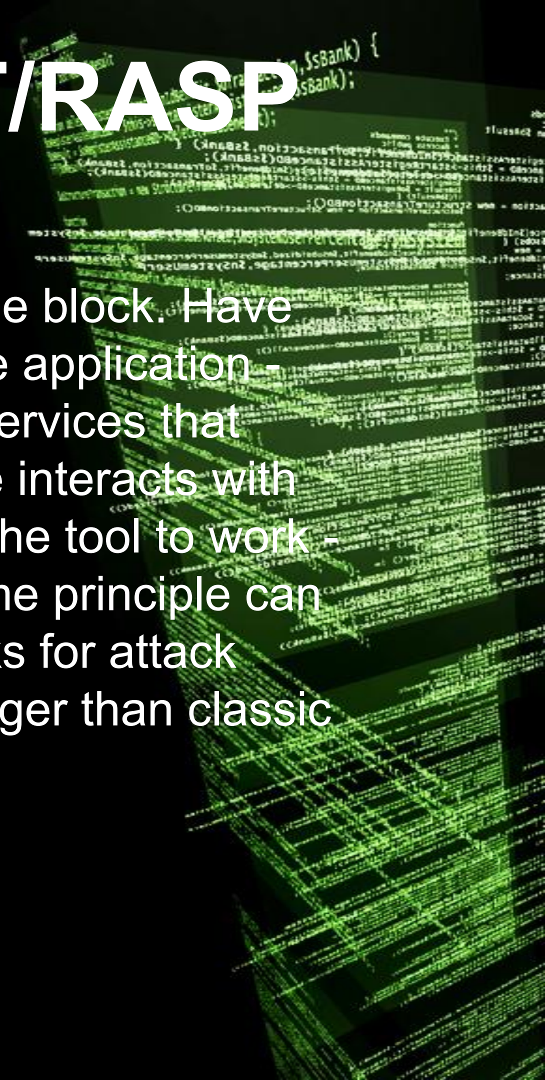
SAST and DAST techniques complement each other. **Both need to be carried out for comprehensive testing.**

SYNOPSYS® To learn how to create a comprehensive software security testing program, visit www.synopsys.com/software

Most well-known tools: SCA



SCA – Software composition analysis - this tool is scanning for known vulnerabilities into dependencies (hello node.js) and in some cases can open PR to fix. Commercial tools are using proprietary databases with vulnerabilities (or at least are claiming) and are scanning in real-time. Can be used into IDE, as part of pipelines or into the SCM. Tricky for languages without package manager. Some tools also scan for potential license issues.



Most well-known tools: IAST/RASP

IAST – Interactive application security testing - new kid on the block. Have the capability to replace DAST testing. IAST works inside the application - with agents deployed between different components/ microservices that looks for pattern into the traffic (inside traffic) when someone interacts with the application. Interaction with application is necessary for the tool to work - this is mainly done via e2e testing or with DAST testing. Same principle can be used with real time protection - it is called RASP and looks for attack patterns inside the application. Resource consumption is bigger than classic solutions (WAF)

Most well-known tools: IAST/RASP

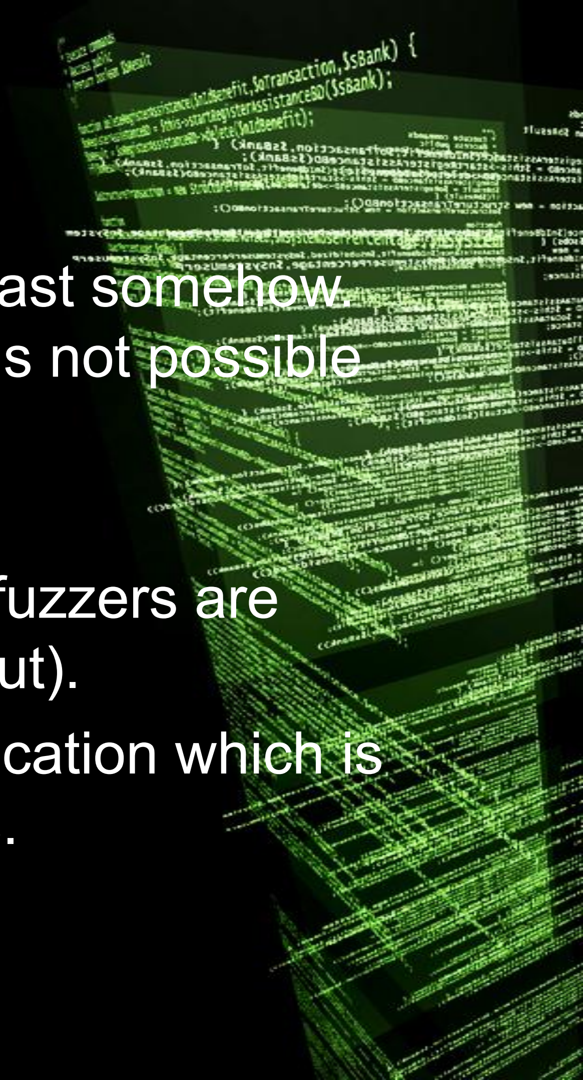
Positives:

- Small number of false positives
- Huge number of true positives
- Problem can be easily identified (compared to DAST)
- One the application have been instrumented is possible to combine IAST and RASP (but not always needed)

Negatives:

- Limited technology coverage
- Requires good coverage of test cases or using together with traffic generating tools (DAST).
- Integration is not straight-forward. At all.

Fuzzing



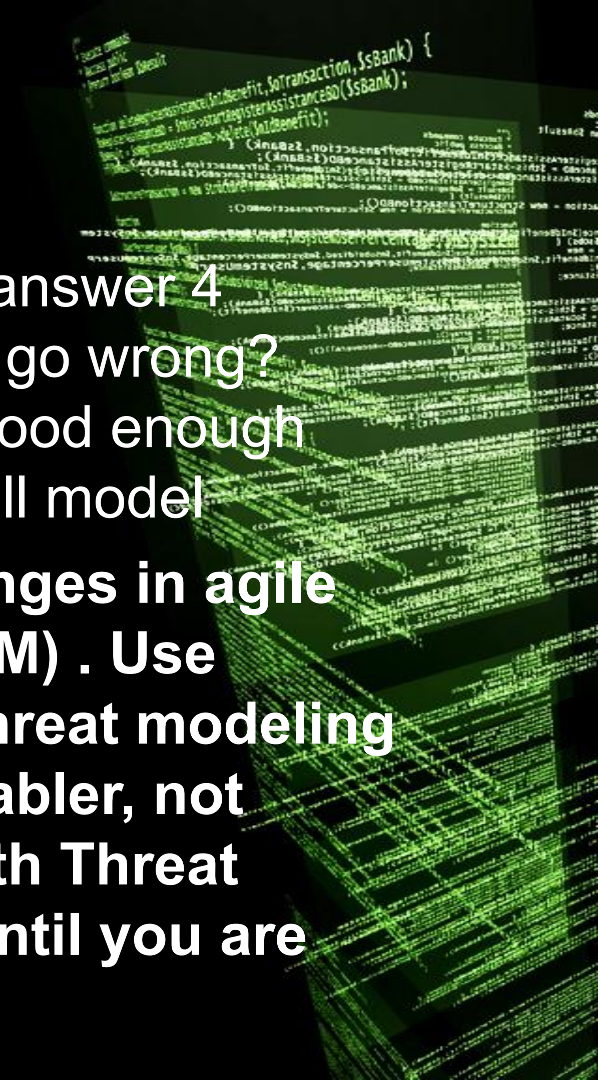
DAST tools can be integrated into pipelines at least somehow. But with fuzzing, as is with penetration testing it is not possible to be part of pipeline. Reasons:

- Fuzzing is slow.
- Tests are not predictable and not reputable (fuzzers are using random/pseudo random generated input).
- Result we are looking for is clash of the application which is basically damaging the next pipelines stages.
- Manual review of the results is needed.

Good old boring stuff

Threat modelling - It is series of interviews that answer 4 questions- What are we working on? What can go wrong? What are we going to do about it?Did we do a good enough job? Problem - it could takes weeks to build a full model

What to do? - Utilize iterative approach-changes in agile are small by definition and use RTM (rapid TM) . Use automatic threat modeling solutions (even threat modeling in code). And realize - security should be enabler, not gate-keeper. Work with dev teams to help with Threat modelling instead of keeping them waiting until you are ready with threat model.



Interesting links for reading

As the time is really short, there are some links you can use to review some not well-known concepts:

- Risk assesment in DevSecOps - https://infosec.mozilla.org/guidelines/risk/rapid_risk_assessment.html
- How to build own demo/ test lab - https://nullsweep.com/creating-a-secure-pipeline-jenkins-with-sonarqube-and-dependencycheck/

Credits:

- This presentation has been designed using resources from:
  https://poweredtemplate.com/06508/0/index.html
- Book – DevOpsSec by Jim Bird