Yandex

Yandex

How to implement SDL and don’t turn gray

Andrey Kovalev, Security Engineer

Agenda

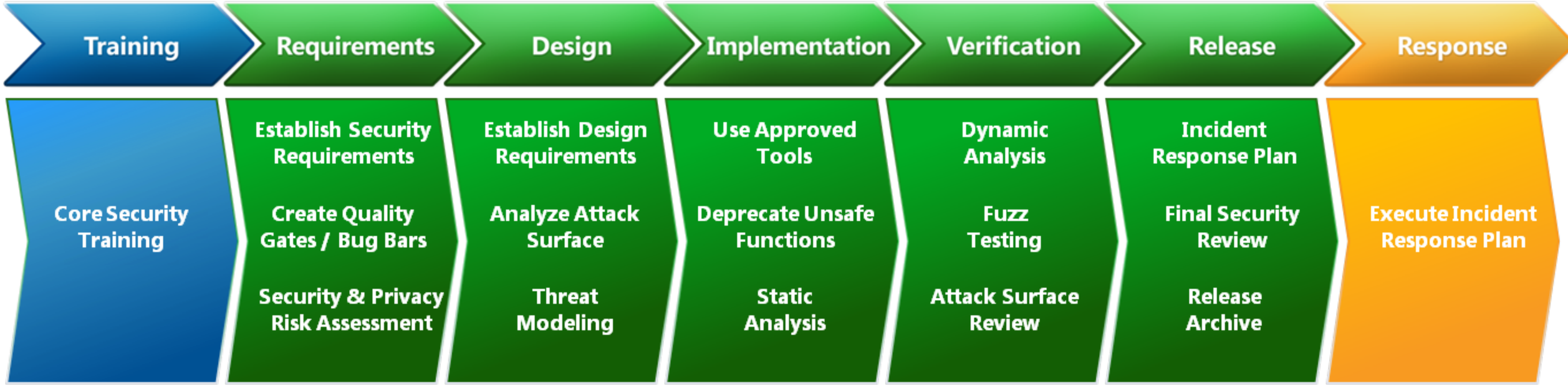
- SDL 101
- Yandex approach
- SAST, DAST, FSR: drawbacks and solutions
- Summary

How to implement SDL and don’t turn gray

SDL 101

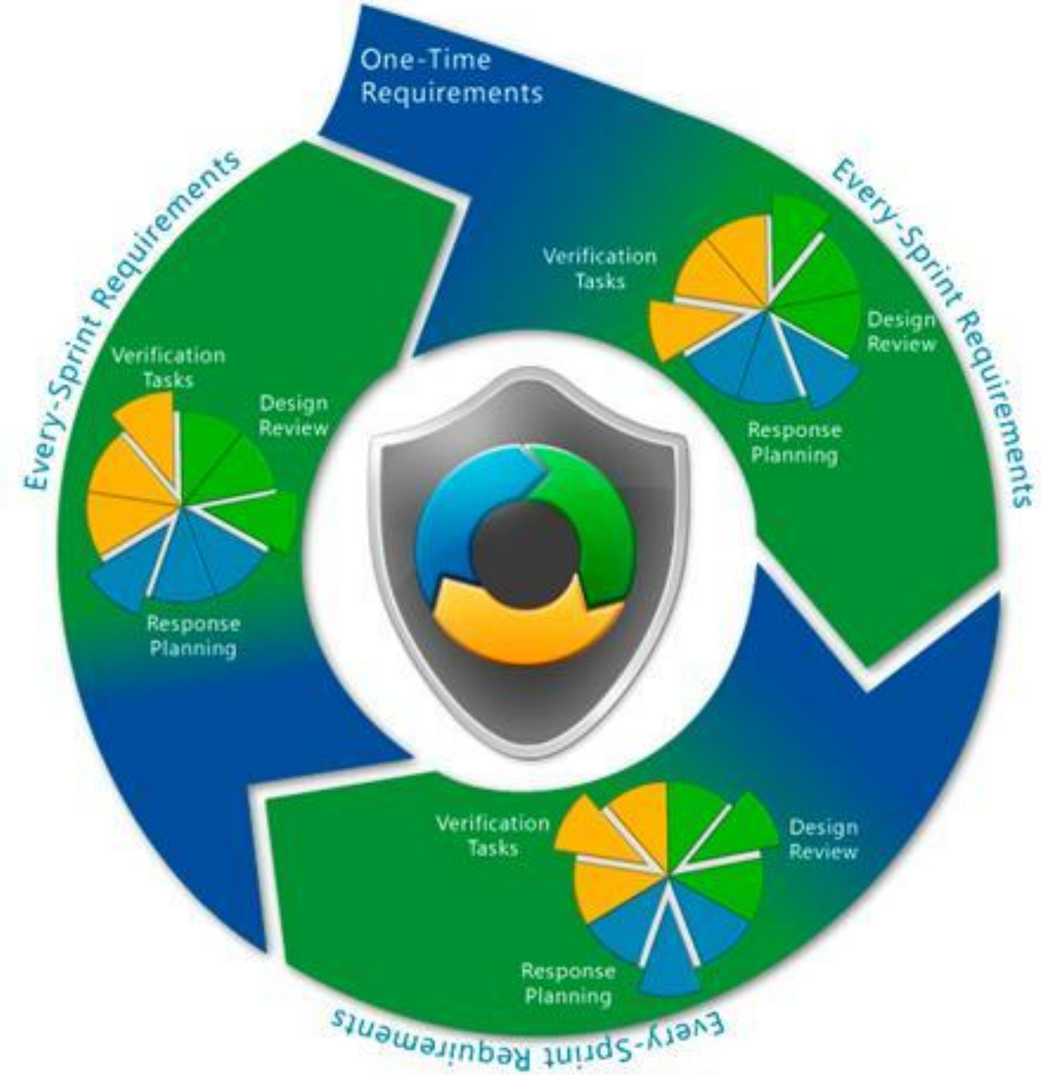
Canonical waterfall SDL

Invented and used in Microsoft since 2004, published in 2012



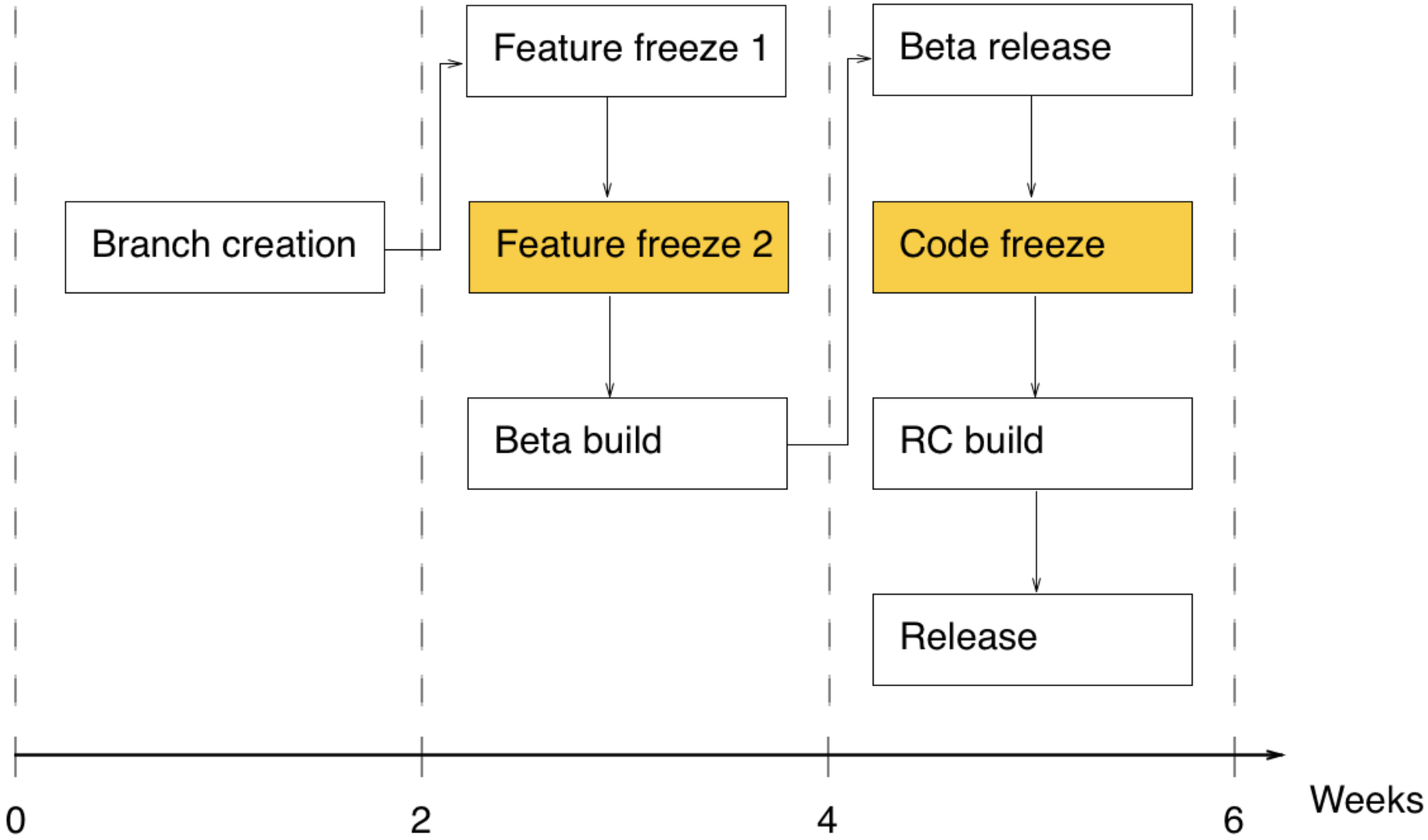
Agile SDL variant from Microsoft

- Controls: one-time, bucket and sprint
- Actionable points: threat modeling, fuzz testing, code analysis, final security review
- Flexible scope, exceptions are not forbidden



Yandex approach

Development lifecycle



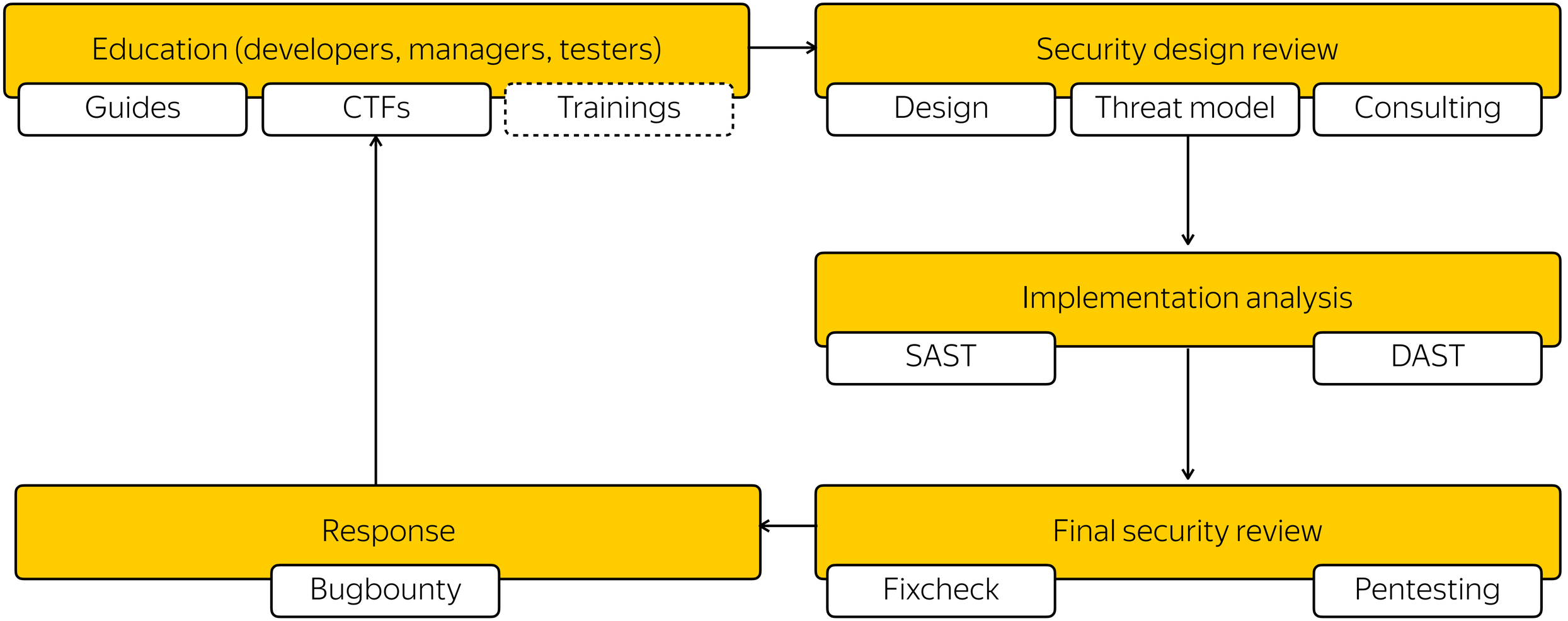
Development paradigm

- Scrum-based agile paradigm
- Sprint lasts 2 weeks
- Story point - approximate 1 day
- Tasks type: product feature, technologic task, infrastructure
- Product feature team: product manager, project manager and development team

Codebase and CI

- One repository for Desktop and mobile versions
- Languages: C++ 14, JavaScript, Java, ObjectiveC (for mobile)
- Size of code : approximately 10 GB
- Our codebase is about 20% and Chromium is about 80%
- CI system: TeamCity with own modifications

Yandex SDL approach



Main scope

- Security-related features
- New key and complicated features
- Yandex browser Web-API enhancements
- All features, that potentially could reduce security or privacy of our users

Examples, solutions and drawbacks

Security design review

Design & consulting

- Product requirements analysis
- Make architecture modification roadmap (if needed)
- Consulting in choosing right security solutions or algorithms (if needed)
- Provide special recommendations for testers

Threat modeling

- 2 types of models: general and feature-based
- General threat model is based on typical threats and created (and updated) by security team
- Feature based model is created for feature during security design review with managers and developers (one release control)
- Feature based model is the scope for final security review or for special fuzzers

General threat model of the Yandex Browser

Attack vector	Typical vulnerabilities	Severity level
Sandbox escape	Sandbox policy flaws, lpe, etc	P1
RCE	Memory corruption, logic flaws	P2
SOP bypass	Uxss, cache issues, api issues, etc	P2
HTTPS bypass	Bugs in the TLS realization	P3
CSP bypass	Any except proxy- or extension-based	P3
Protect bypass	Secure wifi, safe browsing bypass	P3
Built-in extensions flaws	Xss, xsrf, xxe, sensetive info leakage	P3
UI Redressing and evil manipulations	Full screen imitation, browlocking, etc	P4
DoS	Local or remote hangs	P4

Static analysis

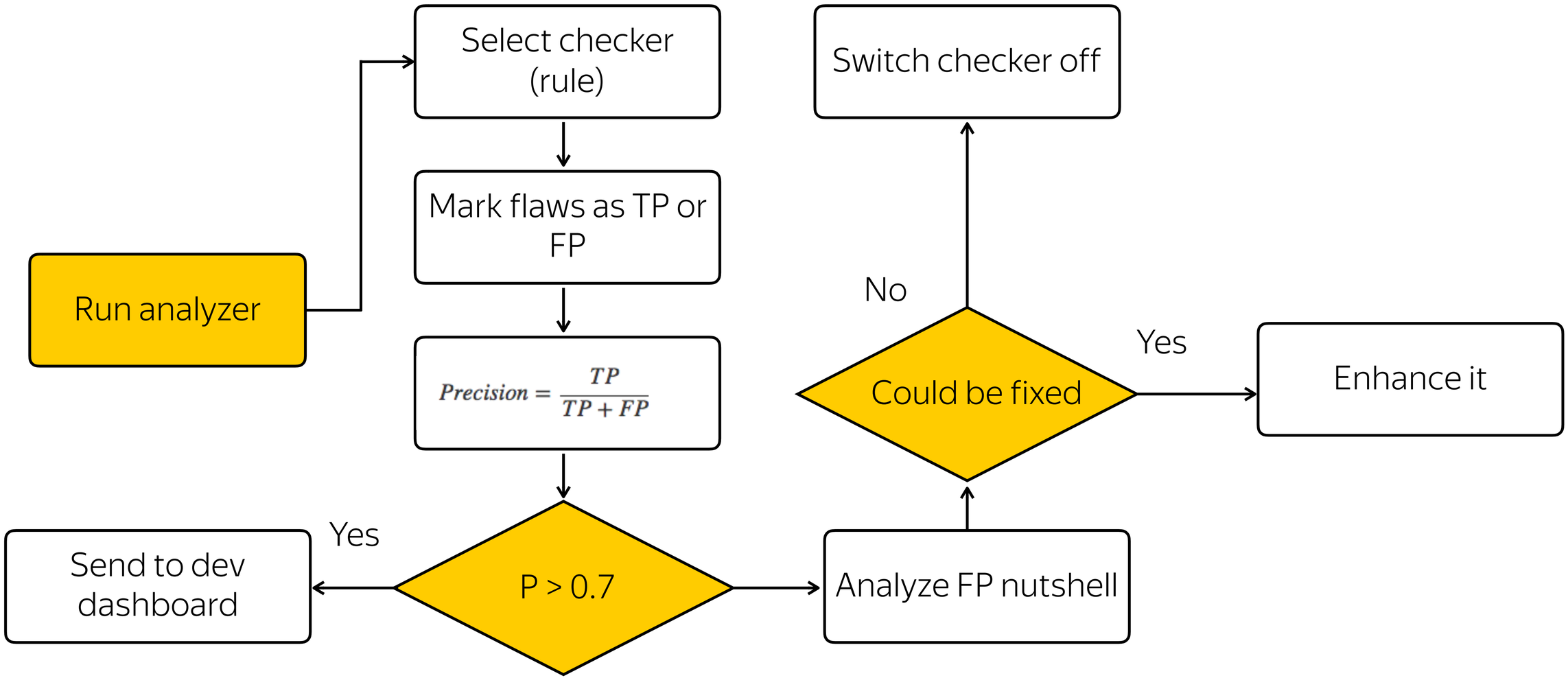
Why do we need SAST?

- We have platform specific code (Windows, Android, etc), there are lack of modern DAST tools (sanitizers, fuzzers, etc.)
- We do not have special test cases (or any test at all) to use it
- It’s easy to create own checkers, which could help to find potentially dangerous function calls
- High code coverage from scratch (particulary there are some exceptions)

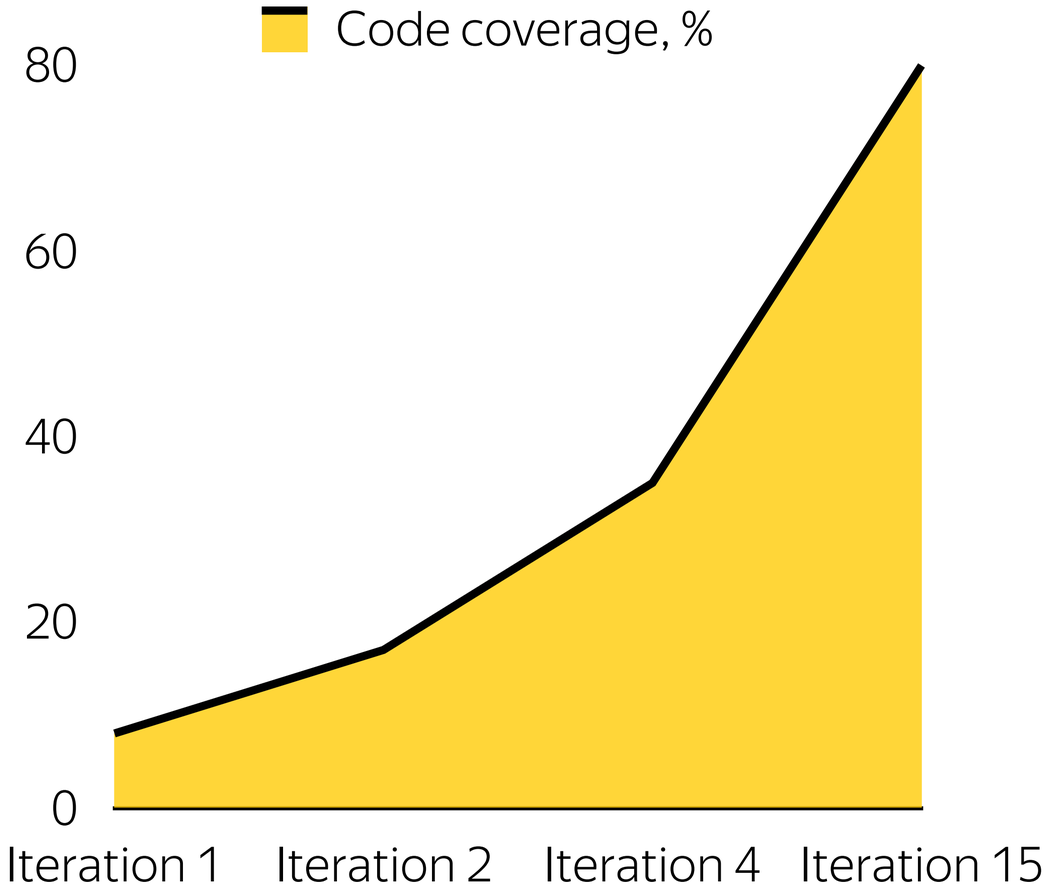
Main SAST drawbacks

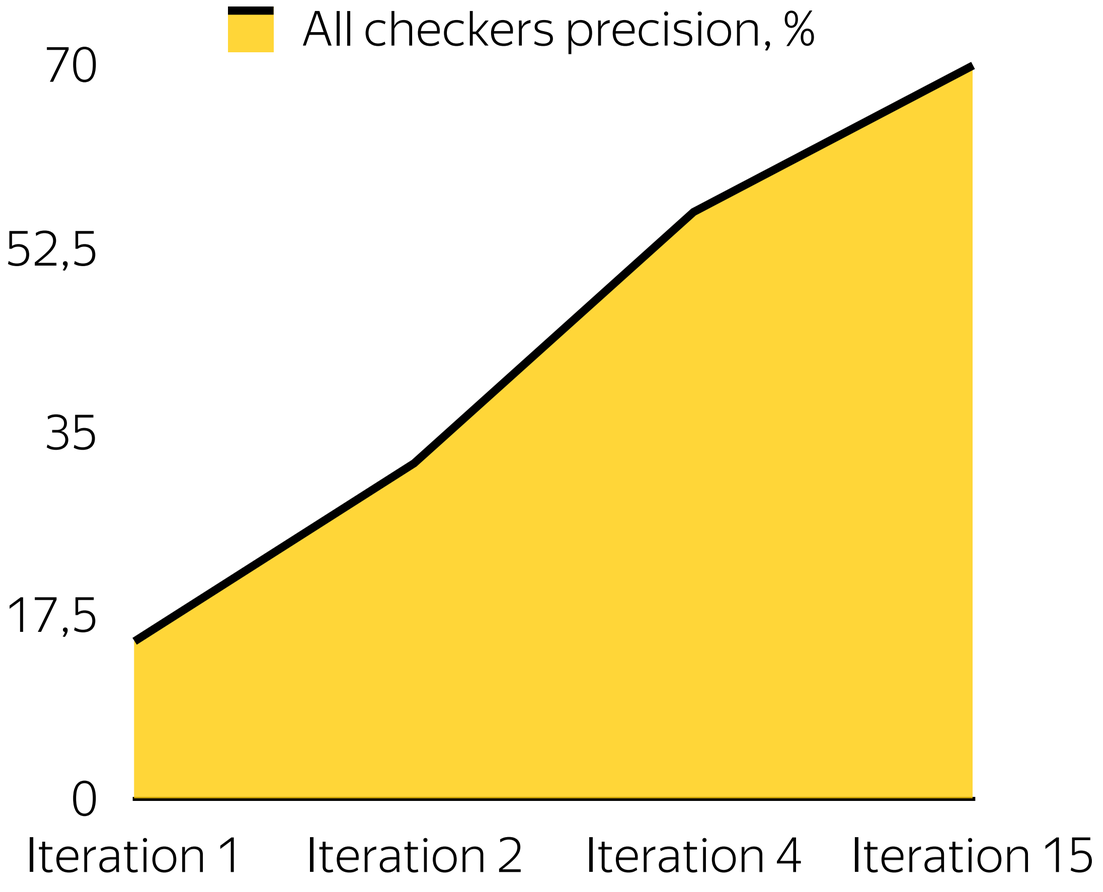
- Signal / Noise = 20 / 80 from scratch, developers don’t like it
- SAST tools have issues with modern language techniques, e.g. shared / weak pointers
- You should build your code (at least for C++) with SAST tool and have to wait support for modern compilers / IDE
- Powerful tools and solutions are not free

Solution: tune your checkers



Results


Code coverage, %
80
60
40
20
0
Iteration 1
Iteration 2
Iteration 4
Iteration 15


All checkers precision, %
70
52,5
35
17,5
0
Iteration 1
Iteration 2
Iteration 4
Iteration 15

Dynamic analysis

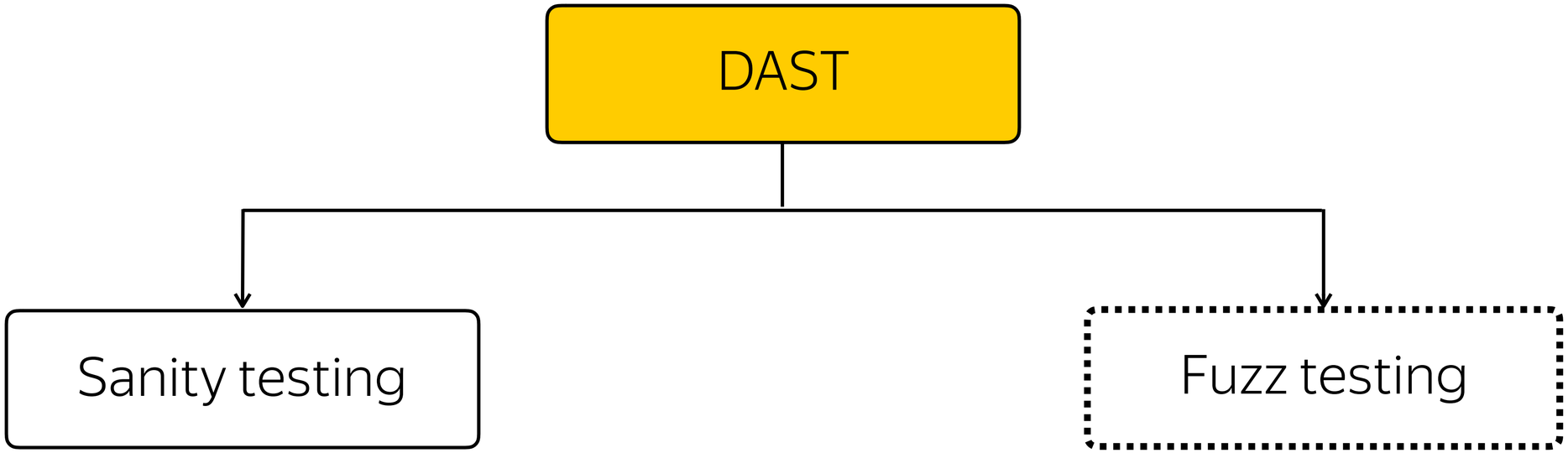
Why do we need DAST?

- Best accuracy, if it finds something developers have to fix
- Finds memory corruption or undefined behavior issues automatically (almost)
- Lots of powerful tools are free and open source

Main DAST drawbacks

- Multiplatform infrastructure for testing is required
- Developers have to write unit tests or fuzz tests and it’s hard to attain 80 - 100% coverage
- It’s not easy to implement complex solution on some platforms (Windows, Android)
- Doesn’t find issues, which were not executed during the test

Our dynamic security testing solution



Sanity testing

- Build your unit tests with sanitizers (Asan, Ubsan, Tsan, etc)
- Add additional testing step to your CI process
- Run sanitized tests regularly during this step on your testing infrastructure
- Collect crashes and analyze them using llvm-symbolizer

Unit fuzz testing

- Implement after sanity testing
- For baseline ask developers to create fuzz tests with afl + libfuzzer with some input corpuses
- Add additional steps to your CI process: corpus collector and tests runner

Special fuzz testing

- Strange or complex solution requires special fuzzers, which works like standalone program
- Security team creates and runs such solutions

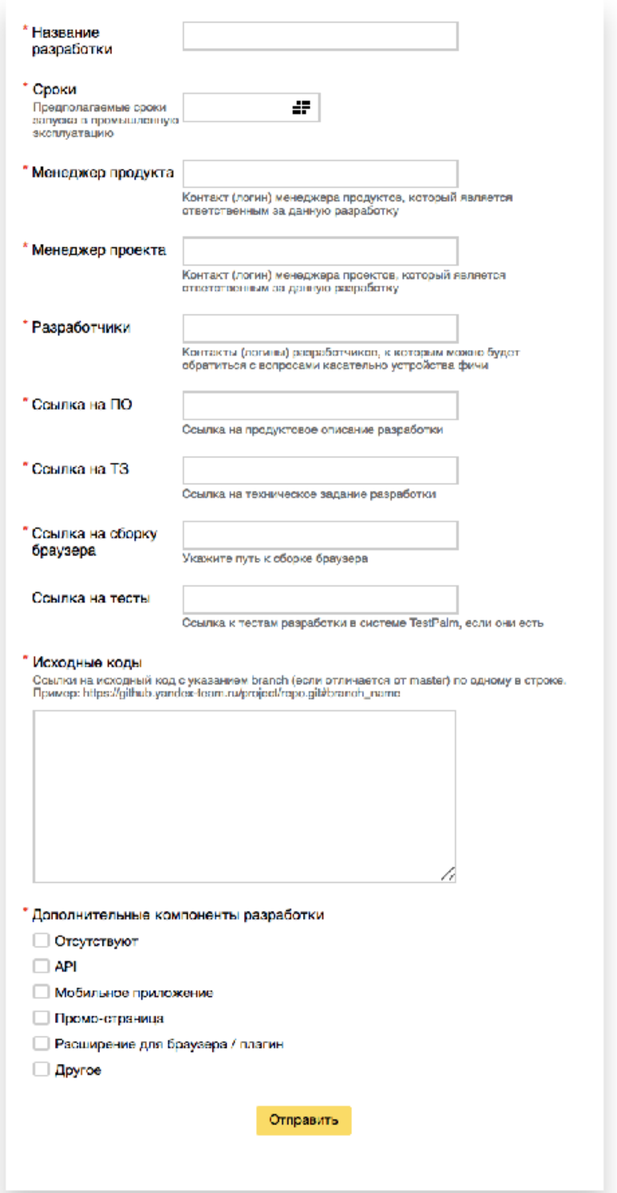
Final security review

Main FSR drawbacks

- It could become bottleneck if you don’t implement other steps of SDL
- In agile development it’s lack of time to recheck threat model, results of security tools, and bug fixes, etc

Final security review solution

- Per feature review process
- Non blocking for non-security features
- For some small features it’s just check, that all automation controls are passed
- If feature is key-feature security team focuses on feature-based threat model and performs additional pentesting



Summary

Process conclusions

- For huge products start SDL from their security related or most valuable features
- Work with all your team: managers are also your process’s actors
- Actualize your threat model as often as you can, think out of the box, create separate models for special features
- Work with your developers: actual guides, trainings, special hackathons or ctfs helps to increase code quality
- Want to measure effect? Start from the response stage

Methods conclusions

- There is no silver bullet, you have to use different approaches to implement security controls
- If you have good test cases coverage start DAST from building tests with sanitizers else try to write fuzzy tests with afl+ libfuzzer
- Using SAST watch your checkers, switch off false positive checkers, create 2 different filters: for developers and for security team
- Enhance tools, SAST or DAST analyzers form scratch are not good enough for you

Thank you!

Q &A

Andrey Kovalev
Security Engineer

 avkov@yandex-team.ru

 @L1kvID

 @L1kvID