SAMSUNG Research

Ivannikov ISP RAS Open Conference 2019

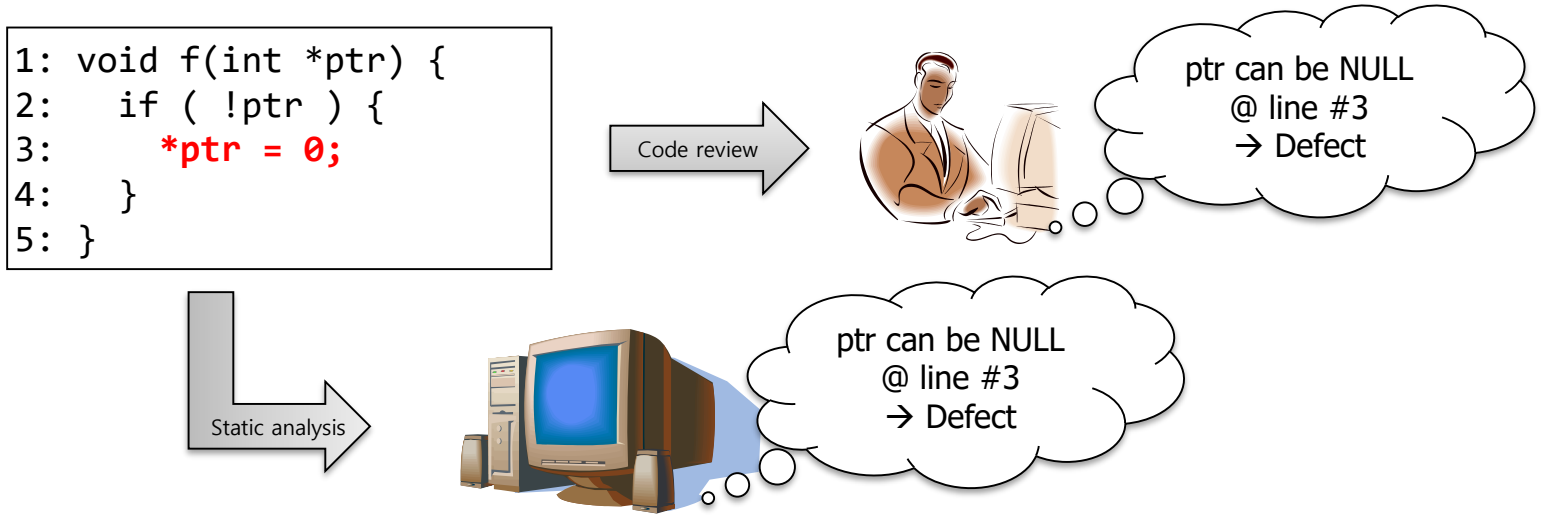
Deploying SVACE to Samsung

Youil Kim (Samsung Research)

Shape the Future with Innovation and Intelligence

Background

- **SVACE is a static analysis tool for early detection of SW bugs.**
  - The name stands for "Security Vulnerabilities And Critical Errors."
- **Static analysis estimates a program's runtime behavior without actual execution by analyzing the source code.**
  - It does not require runtime environments or test inputs.
  - It is easily applicable to large-scale SW.
  - We may consider it as an automated code review.



A Short Collaboration History

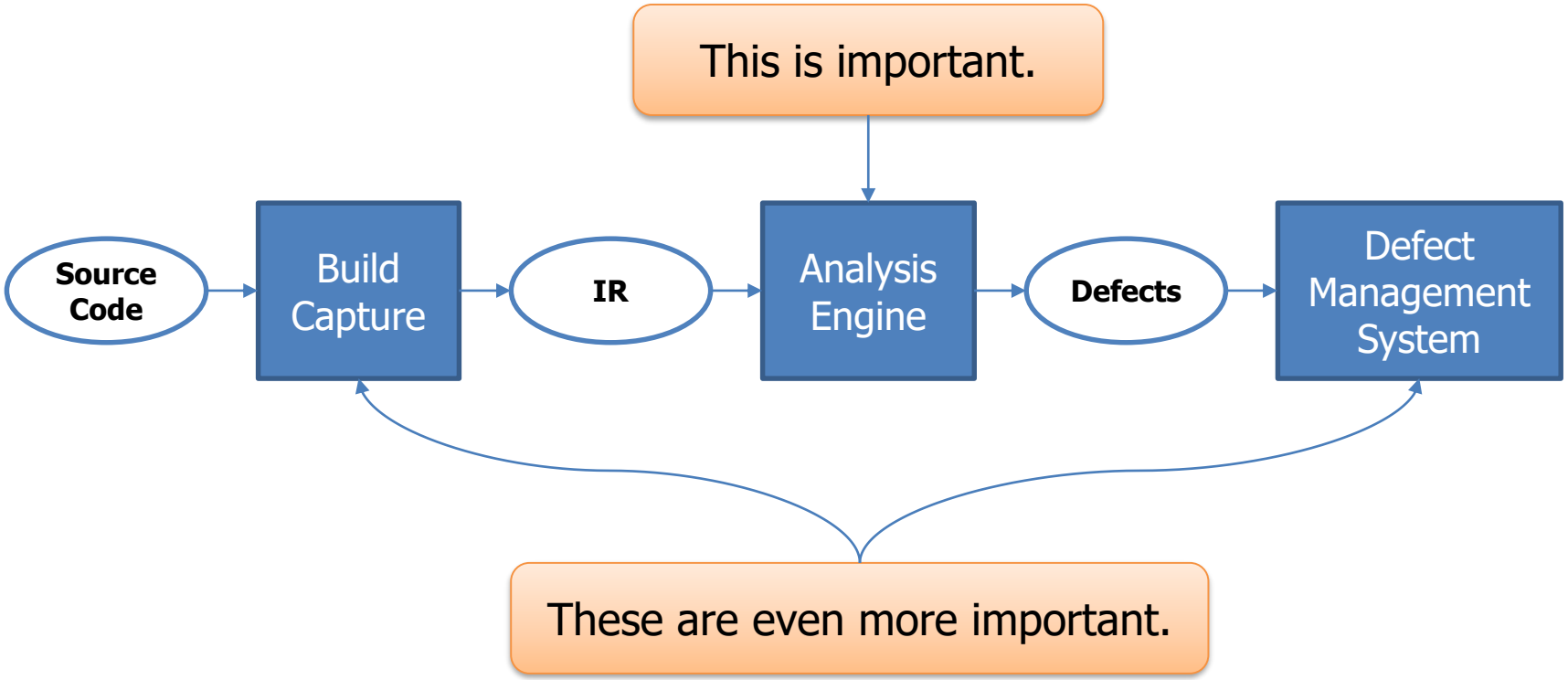
- **In 2009, we started a joint project on SVACE of ISP RAS.**
  - To introduce SVACE as an auxiliary tool in addition to commercial tools
- **In 2012, we decided to develop SVACE for the purpose of deploying it as the main static analyzer in our SW development process.**
  - We put a lot of time and effort in improving SVACE from 2013 to 2015.
  - ISP RAS colleagues also had very tough time.
- **At the end of 2015, we successfully deployed SVACE!**
  - We improved SVACE every year and released SVACE 3.0 in this year.
- **Our collaboration is to be continued in 2020.**

Our Current Position

- **SVACE has been deployed to most business divisions in the company.**
  - Samsung Research
  - Digital Appliances Business
  - Health & Medical Equipment Business
  - Mobile Communications Business
  - Network Business
  - Visual Display Business
- **Most of C, C++, C#, and Java source code is regularly checked with SVACE.**
  - Including Tizen TVs, Samsung mobile devices, FamilyHub, etc.
  - More than 10,000 users (developers)
  - More than 300 billion lines code has been checked (since 2015.)

From Research to the Field

- **There were several challenges.**
  - We knew that it would be never easy, but we underestimated the effort.



Challenges #1 – Build Capture

- **What happened to us when we tried to analyze C/C++ code in the company**
  - Build capturing is essential.
  - Clang was not enough to support 90 different C/C++ compilers in the company.
  - We had to support various systems including an old legacy OS.
  - Conventional build capture implementation may not work with Git Build System (GBS) or Open Build Service (OBS).
- **Some achievements**
  - SVACE provides quite stable build capture feature.
  - We extended Clang so that it can handle most of the company code, with a small workaround to ignore unknown constructs.
  - We made a special RPM package to install SVACE into GBS/OBS build roots and enable SVACE's build capture.

Challenges #2 – Analysis Engines (1/2)

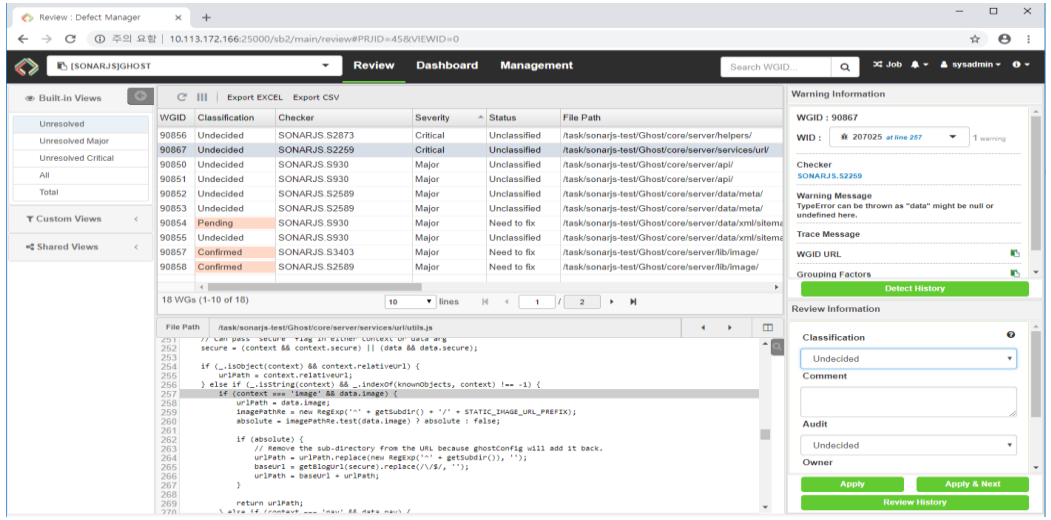
- **IR level analysis was not enough.**
  - At the beginning, SVACE was a static analyzer for LLVM bitcode.
  - Some coding errors could not be detected after we translate the code into LLVM bitcode.
- **Path-sensitive analysis was not enough.**
  - Path-sensitive analysis was essential but was not enough to satisfy our quality goals.
  - There is no one silver bullet to address many different false positive patterns.
  - Some warnings are technically true positives, but developers don't want to see them.
- **Some achievements**
  - SVACE has lightweight checkers on AST. (e.g., Clang Static Analyzer for C/C++ language)
  - SVACE performs path-sensitive analysis using an SMT solver.
  - In this year, we removes 47% of useless alarms that developers marked as 'Won't Fix' and 'False Alarm' in 52 C/C++ checkers.

Challenges #2 – Analysis Engines (2/2)

- **Analysis results should be deterministic and stable.**
  - We should not rely on hardware timer, random values, or any non-deterministic components.
  - A build process is often not deterministic, due to timestamps, temporary files, and so on.
  - Even if some part of a project has been changed, developers expect to get the same set of defects for the rest (unchanged part) of the project.
- **The performance is always not enough.**
  - Developers expect to get daily analysis results even for the largest codebase.
  - Our codebase becomes larger and larger.
  - Developers even expect to get analysis results within several minutes after code review integration.
- **Some achievements**
  - It takes 11 hours for SVACE to build and analyze Tizen 5.0 platform (17M LOC).
  - SVACE produces the same results for each trial, when we build and analyze the Tizen platform 5 times.

Challenges #3 – Defect Management

- **It is crucial to have a good defect management system.**
  - Some of key features are supports for code navigation and grouping similar warnings.
  - It requires considerable efforts to develop such system as it should be able to handle a lot of data.
- **Some achievements**
  - SVACE provides some necessary information (e.g., warning grouping factors, relations among code tokens) for defect management systems.
  - We developed our own defect management system and improved it for several years.



One More Challenge – Competitiveness

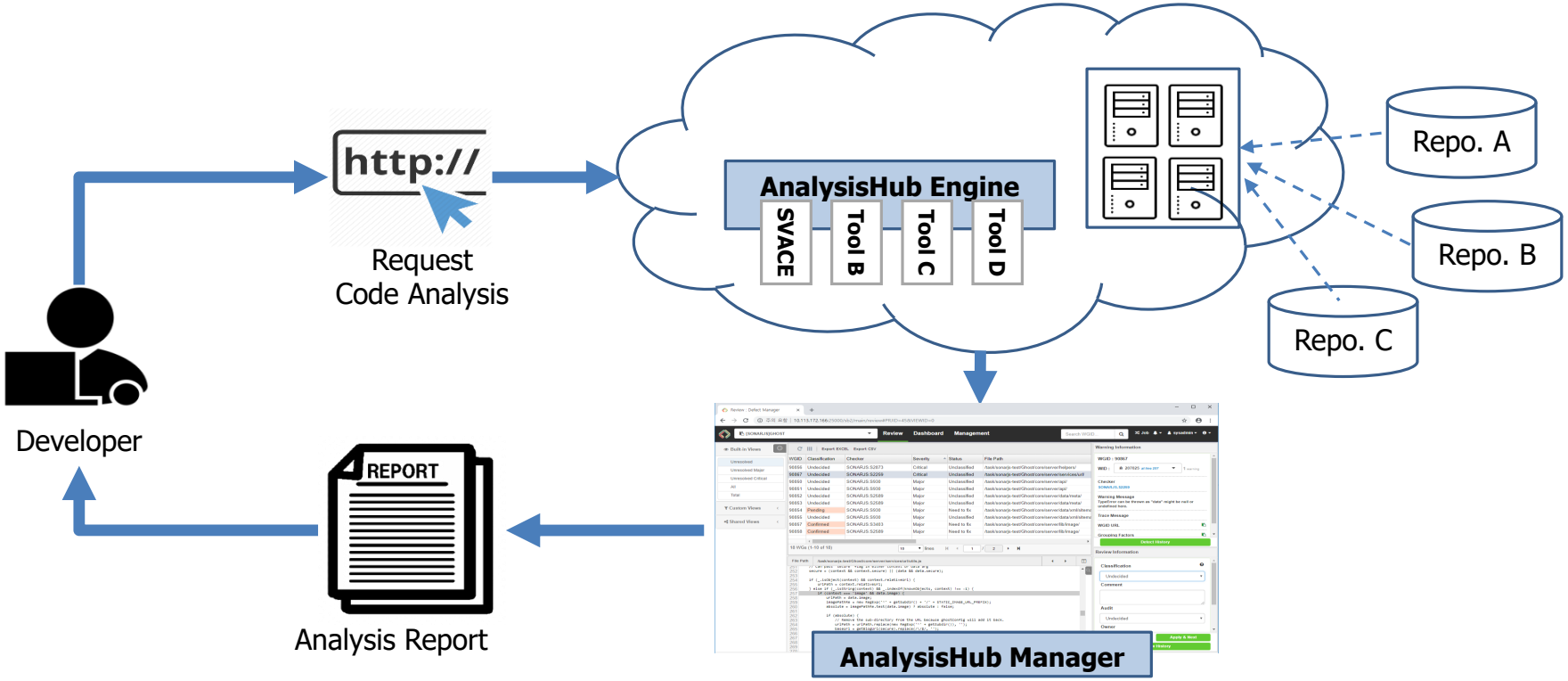
- **We should keep our competitiveness against commercial tools.**
  - By meeting company-specific requirements better
  - By tight integration into our SW development infrastructure
- **Some achievements**
  - In 2015, we provided SVACE for an old legacy RHEL 3 system, which a commercial tool stopped to support.
  - In 2016, SVACE enabled the analysis of a new Android platform using Jack toolchain, 6 months earlier than a commercial tool.
  - SVACE also has a number of checkers developed by the requests from our business divisions.

Conclusion – Beyond A Standalone Tool

- **We successfully deployed SVACE to the company through active collaboration with ISP RAS colleagues.**
- **We integrated SVACE into our SW development infrastructure.**
  - AnalysisHub provides SVACE as a service.
  - Code Review Bot provides SVACE results at code-review time.

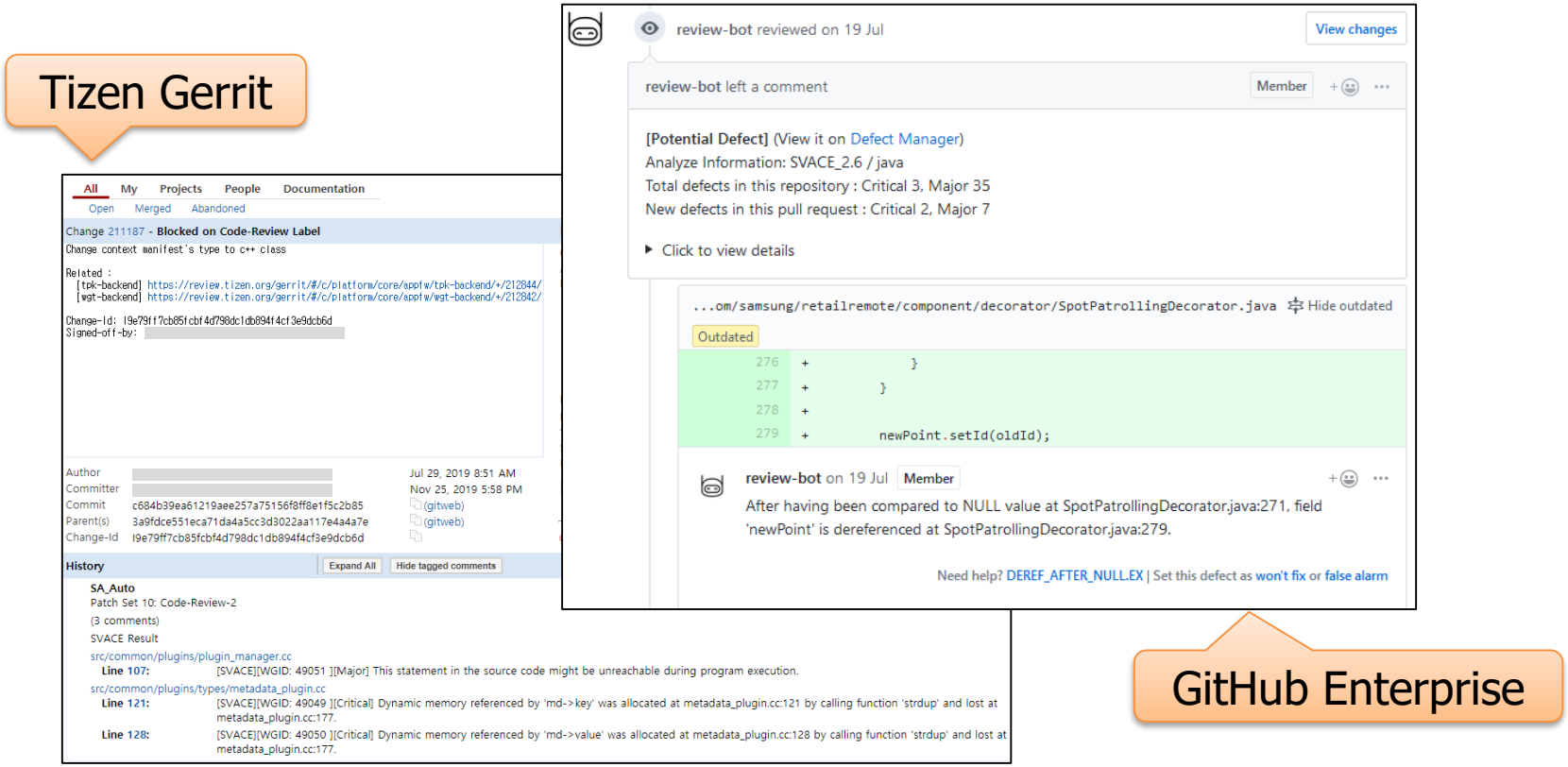
Integration #1 – Static Analysis as a Service

- **We have developed AnalysisHub, an extensible and scalable framework for providing static analysis as a service.**
  - SVACE is one of the core analysis engines in the service.



Integration #2 – Code Review Integration

- **On top of AnalysisHub, we developed Code Review Bot.**
  - It generates automated code review comments including SVACE analysis results.

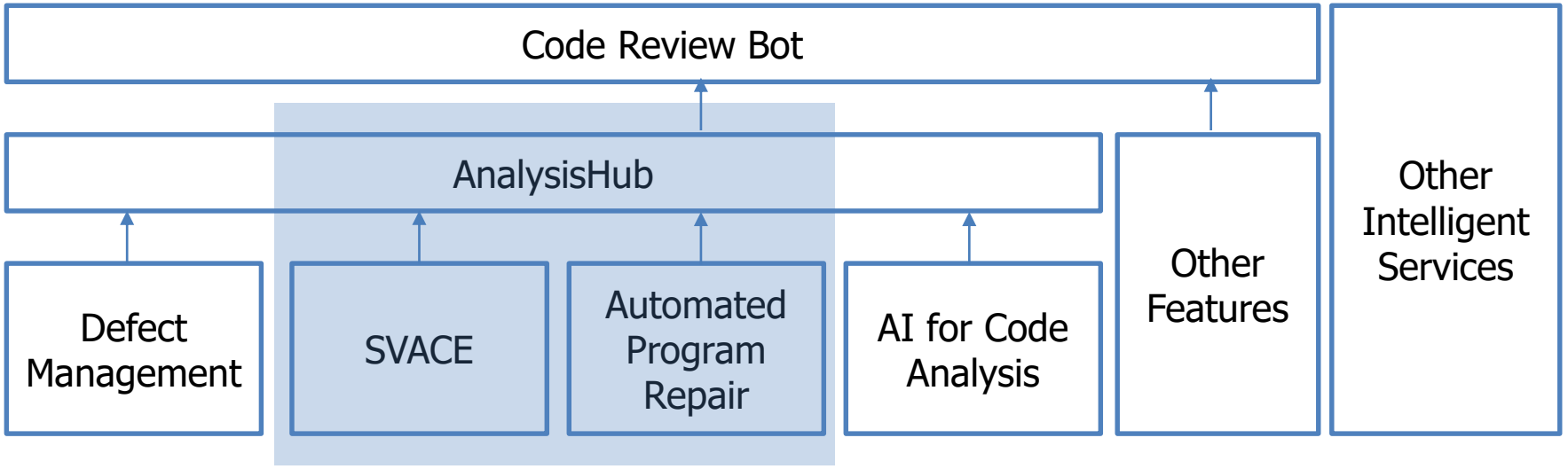


Future Work

- **Structural analysis as another important use case of static analysis techniques**
  - We developed SCRA, a static analysis tool for extracting a set of metrics and data/call relations, currently integrated into SVACE.
  - We're developing code analysis services to support refactoring work.
- **Combining AI techniques with code analysis techniques**
  - Further reducing false positives
  - Learning bug patterns from big code in repositories
- **Reducing code writing effort**
  - We started work on automated code fix tools in 2018.
  - SVACE can suggest how to fix bugs, for those detected by some C/C++ checkers.
  - We're extending this feature for other checkers.

Future Work – Collaboration with ISP RAS

- **Our collaboration is to be continued in the next years.**
- **We hope to extend our collaboration to other areas as well.**



Thank you

Shape the Future with Innovation and Intelligence