<table>
<thead>
<tr>
<th>ID</th>
<th>Article Title</th>
<th>Author(s)</th>
<th>Advantages (RG1)</th>
<th>Disadvantages (RG2)</th>
<th>Static tools (RG3)</th>
<th>Solutions (RG4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>STATIC CODE ANALYSIS TOOLS: A SYSTEMATIC LITERATURE REVIEW.</td>
<td>Stefanovic, Darko</td>
<td>Manual code review is time-consuming. Optimizing the operation of the compiler. Detecting irregularities. Help understand the behaviour of a program without execution.</td>
<td>Detects must be visible in the source code. Not all programming languages are supported. A single tool might not be able to detect all defects.</td>
<td>Cppcheck, FindBugs, Splint, SonarQube, PMD, Flawfinder</td>
<td>Range of programming languages supported is continuously expanding. Combining tools can help detect all defects.</td>
</tr>
<tr>
<td>2</td>
<td>Probing into code analysis tools: A comparison of C# supporting static code analyzers</td>
<td>Shaikut, Rida</td>
<td>Saves enormous amount of time and money in a development process. Some tools are highly customizable.</td>
<td>Tool can give vague explanations and unwanted checks. High memory consumption (RedShaper, Paracsof).</td>
<td>FxCop, NDepend, Nitin, RedShaper, Covertly Scan, PVS-studio, Paracsof dotTest, Visual Code Grepper</td>
<td>Correct tools have technical and accurate description. Some tools can be customized to fix vague problems.</td>
</tr>
<tr>
<td>3</td>
<td>A Comparison of Open-Source Static Analysis Tools for Vulnerability Detection in C/C++ Code</td>
<td>Arusoaie, Andrei</td>
<td>Some tools do not detect all defects. Difficult on install/compile some tools because they rely on older versions of libraries of compilers.</td>
<td>For some defect types there is no good tool.</td>
<td>Clang, Frama-C, CILint, Cppcheck, Splint, Infer, Uno, Flawfinder, Sparse, Flin++</td>
<td>Certain tools are better at detecting certain defect types than others.</td>
</tr>
<tr>
<td>4</td>
<td>How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines</td>
<td>Zampetti, F.</td>
<td>Early detection of potential faults, vulnerabilities, code smells.</td>
<td>Tools help with the quality of the code as an engineer might not know the best methods (especially security-wise) and therefore also give insight to the user. Vulnerabilities can be found without executing.</td>
<td>CheckStyle, FindBugs, PMD, Apache-rt, Clirr, JDepend</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Static Code Analysis Tools with the Taint Analysis Method for Detecting Web Application Vulnerability</td>
<td>Maskur, Achmad</td>
<td>Tools help with the quality of the code as an engineer might not know the best methods (especially security-wise) and therefore also give insight to the user. Vulnerabilities can be found without executing.</td>
<td>Some vulnerabilities are not found. False negatives.</td>
<td>Detecting vulnerabilities using taint analysis can be improved by supporting OOP. Tools are to be configured based on false negatives and manually discovered defects that were not found by a tool.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Using Machine Learning Techniques to Classify and Predict Static Code Analysis Tool Warnings</td>
<td>Alkhashashneh,</td>
<td>Examine code without execution.</td>
<td>False positives. False negatives.</td>
<td>Machine learning (particularly Random Forests technique) can help reduce false positives. Also, techniques like KNN, SVM and RIPPER. False positives can also be avoided if developers rewrite their code in a way that reduces source code complexity, coupling and usage of global variables.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Comparative study on static code analysis tools for C/C++</td>
<td>Fatima, Anum</td>
<td>Efficient system to check on software coding scheme. Defects (safety, security bugs, quality standards, dereferences, buffer overruns, injection problems, memory leaks, dataflow problems, language implementation errors, inconsistencies) are detected without execution. Helps build long-lasting software without bugs and vulnerabilities.</td>
<td>No tool can give 100% certainty that software will never halt, crash, misperform. False negatives. Some tools are too specific and lack basic checks, others are just too basic.</td>
<td>FlawFinder, VCG, CppCheck, Splint, Polyspace, CESTAT (was the best of this bunch), QAC, Covertly, Astrea, XCode, Paracsof, VASSA, Sparix, IT54, Goanna, RATS</td>
<td>Tools can be improved as lacking functionality is discovered. Tools can be used together to cover all important code vulnerabilities.</td>
</tr>
<tr>
<td>8</td>
<td>Evaluating how static analysis tools can reduce code review effort</td>
<td>Singh, Devanshi</td>
<td>Tools can help to reduce code review effort. Code reviewer or developer does not need to know the best practice patterns. Tools automatically find defects and style issues. No need for reviewers to use comments in comments. noisy.</td>
<td>False positives.</td>
<td>PMD, FindBugs, CheckStyle</td>
<td>Often the best tools (etc. PMD) do not have high amount of false positives. For the remaining false positives, configuring a tool’s (PMD’s) ruleset cannot limit the warnings.</td>
</tr>
<tr>
<td>9</td>
<td>Identifying and Documenting False Positive Patterns Generated by Static Code Analysis Tools</td>
<td>Raynolds, Z.P.</td>
<td>Tools help developers find defects automatically. Manual code analysis is time-consuming.</td>
<td>False positives. False negatives.</td>
<td>CATNET, FindBugs</td>
<td>Static code analysis tools can be integrated into frameworks for evaluating the tools which can help identify potential false positives. Reduce usage of some techniques like global variables to avoid some false positives. False positives can still be “fixed.” False positive patterns can be studied and fixed. Multiple tools can be used together.</td>
</tr>
<tr>
<td>11</td>
<td>Prioritizing Alerts from Multiple Static Analysis Tools, Using Classification Models</td>
<td>Flynn, Lori Shavely, William Sivoboda, David Vankouskos, Nathan Qin, Richard Burns, Jennifer Zubrow, David Stoddard, Robert Marc-Santurio, Guiberto</td>
<td>Tools can help determine flaws without executing the code. False positives. False negatives. Complex control flow or data flow constructs significantly reduce tools’ success rate. Machine learning can help with false positives. Tools can be used together for improved efficiency.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Software quality through the eyes of the end-user and static analysis tools</td>
<td>Kamonhop Sriroopha, Reem Alfayez</td>
<td>The analysis tools can reveal possible vulnerabilities, defects, or design issues at an early stage in the development phase.</td>
<td>PMD, FindBugs, SonarQube</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Scientific Developers v/s Static Analysis Tools: Vision and Position Paper</td>
<td>Rohan Krishnamurthy, Thomas S. Henze, Carina Haupt, Andreas Schreiber, and Michael Meinl</td>
<td>They help identify defects and code smells or enforce common coding standards. High false positives rates, low comprehensibility of analysis results, and missing process integration restrain the use and acceptance of static analysis tools.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>A Practical Approach for Ranking Software Warnings from Multiple Static Code Analysis Reports</td>
<td>Binh Hy Dang</td>
<td>The problem of using multiple bugs finding tools is they not only detect similar software defects but also generate new warning messages. The excessive warnings make code analysis time consuming and expensive. High number of false positive.</td>
<td>FindBugs, PMD, Cppcheck, Undertstand</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Automatically Generating Fix Suggestions in Response to Static Code Analysis Warnings</td>
<td>Diego Marcillo, Carlo A. Furia, Rodrigo Bonfácio, Gustavo Pinto</td>
<td>They report warnings that may always not correspond to an actual mistake. Developers normally fix only a small fraction (typically, less than 10%) of the reported issues.</td>
<td>SonarQube, FindBugs, SpotBug</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Source Code Analysis for Secure Programming Practices</td>
<td>Christian Barrientes, Jeong Yang, Joshua Sanchez, and Young Rae Kim</td>
<td>Static analysis of source code can help mitigate the common coding errors such as buffer overflow, memory leaks, unused variables, and various race conditions. Tools can help determine flaws without executing the code. False positives. False negatives.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Challenges with Responding to Static Analysis Tool Alerts</td>
<td>Nafis Imtiaz, Akond Rahman, Eirat Farhana, and Laurie Williams</td>
<td>Help developers detect potential defects in the code early in the development cycle. False positive alerts, the way in which the alerts are presented, incomprehensible and untrustworthy alerts and a lack of customizability.</td>
<td>FindBugs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>An Empirical Assessment of Machine Learning Approaches for Triage Reports of a Java Static Analysis Tool</td>
<td>Ugur Koc, Shyli Wei, Jeffrey S. Foster, Marine Carpuat, Adam A. Porter</td>
<td>High false positive rates.</td>
<td>FindBugs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>How Developers Diagnose Potential Security Vulnerabilities with a Static Analysis Tool</td>
<td>Justin Smith, Brittany Johnson, Emerson Murphy-Hill, Bill Chu, and Heather Richter Lipford</td>
<td>These tools locate and report on potential software security vulnerabilities, such as SQL injection and cross-site scripting even before the code executes. Researchers cite several related reasons why these tools do not help developers resolve defects, for instance, the tools “may not give enough information”, produce “bad warning messages”, and “miscommunicate” with developers.</td>
<td>Find Security Bugs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>The Use and Limitations of Static Analysis Tools to Improve Software Quality</td>
<td>Paul Anderson</td>
<td>The latest static analysis tools are capable of finding serious errors in programs such as null-pointer dereferences, buffer overruns, race conditions, resource leaks, and other errors. Static analysis can be used very early in the development cycle, its use can reduce the cost of development. They also make it easier to achieve full code coverage. There are path limitations, most tools ignore recursive calls and functioncalls that are made through function pointers. Tool's won't give accurate results when parts of the source code are missing or where the fundamental rules of the language aren't being violated. Tools should be used only to compliment other review and testing techniques.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Static analysis tools are very useful for carrying out initial verification and validation activities compared to other quality assurance procedures, especially due to their low implementation cost. The main disadvantage of static analysis tools is their high false positive rates. False positive alerts lead to an increase in process costs, because their detection is usually done by human intervention. This consumes precious time that could be used for the correction of real faults.

The main advantages of static analysis tools include:
- Finding bugs early in the development process.
- Identifying potential security vulnerabilities.
- Enhancing code quality.
- Facilitating the maintenance process.

Some popular static analysis tools are:
- FindBugs
- PMD
- CheckStyle
- SonarQube

These tools should be combined together or with dynamic tools to reduce false positive rates. Another way would be assigning a weight or priority for each type of false positive and so bugs with higher priority would be given greater attention.

There is a wide range of tools today to choose from and they are becoming increasingly easier to use, especially in continuous integration pipelines. Precision is quite low on average. SonarQube, Better Code Hub, Coverity Scan, Findbugs, PMD, and CheckStyle are some of the tools that are frequently used.

Different tools should be combined to achieve better coverage. Most problems related to static code analysis tools could be avoided if current tools would be complemented with effective tools targeting automated refactoring and automatic test case generation.

The tools are fast, the code can be inspected much more frequently and they can contain the same level of knowledge as a human reviewer. They can help us find common software errors such as memory overruns, cross-site scripting attacks, injections and various other boundary cases. A tool will never find any error if this behavior has not been specified with rules or patterns. They cannot verify the design, architectural errors, poorly made cryptographic libraries, inappropriately selected algorithms, design problems which cause confusion. They can't find passwords or magic numbers in the code. They are prone to false positives. They are only as good as the rules they are using to scan with.

The main disadvantages of static analysis tools include:
- They are prone to false positives.
- They cannot verify the design, architectural errors.
- They are only as good as the rules they are using to scan with.

The tools should be used in correlation with manual code analysis and other review tools.

---

- **RQ1:** What are the main advantages to the usage of static tool analysers?
- **RQ2:** What are the main disadvantages to the usage of static tool analysers?
- **RQ3:** Which static analysis tools are more used in studies in the last 10 years?
- **RQ4:** How can the problems related to the static tool analysers be minimized?