***Conclusions***

*Static Code Analysis*

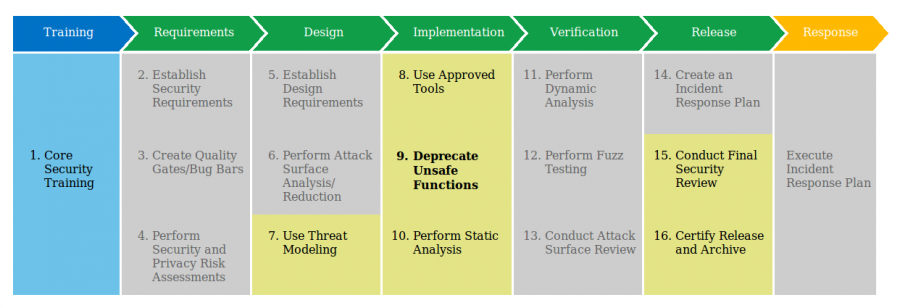
**Static analysis of the code** - analysis of the structure of the source code or the compiled code without its launch.

Static Analysis Software Testing (SAST) - tools for automatic static analysis.

Static analysis and code inspection.

*SDL for Agile*

Agile Development Using Microsoft Security Development Lifecycle - static analysis in every sprint.



Source: <http://www.microsoft.com/security/sdl/discover/sdlagile.aspx>.

*Analysis methods*

* **Lexicographic analysis** - searching for dangerous structures in the source code, usually based on a set of rules, different heuristics and matching to error patterns.
* **Formal methods** - based on the mathematical definition of program behaviour:
  + Mathematical models
  + Logic rules Hoare\_logic
  + Flow analysis Data-flow\_analysis
* **Source code metrics** - evaluating the quality of the source code based on statistical data.

*Benefits*

* **Speed of action** - quick detection of errors, the repair of which is simple and not expensive.
  + Do not require running the program
  + Easy to parallel
* Ease of use - easy to implement in the software development cycle
* Automation - integration with continius integrations tools
* Expandability options: own rules, plugins, ...
* Integration with other tools: automation servers, IDE, version control
* A lot of free tools

*Disadvantages*

* Required access to sources
* Rules usually detect simple errors and cannot eliminate manual code checks
* Lots of noise, too sensitive - high probability of classifying the correct fragment as an error (false positive)
* Each tool usually covers a range of tests (up to 14% of errors ?!). That's why it's worth using a few different code scanners. There are tools that integrate many SAST tools, e.g. CodeEx, Yast (open source)

*What static analysis is about*

* Syntax correctness analysis
* Security flaws, also errors that may occur with specific input data
* Backdoors detection, dangerous and outdated functions, memory leaks, buffer overflow, using uninitialized variables, SQL Injections
* Code quality, style rating, code repetition, unused code snippets
* Performance, bottleneck detection, inefficient constructions, suggestions for improving performance
* Compliance with good practices, maintaining standards, naming standards, problems with portability of the code

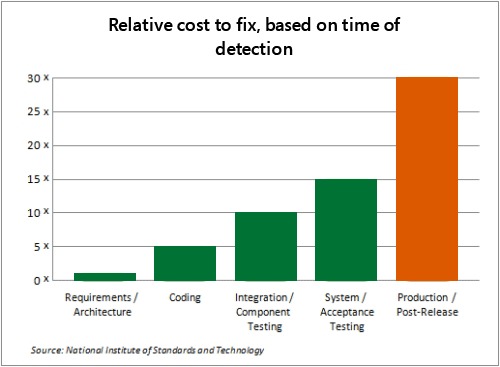
*Why analyze the code?*

The static analysis of the program allows you to:

* Increased efficiency and stability through rules based on good practices,
* Avoid common programming errors
* Providing a structure to manage code standards
* Forcing code writing rules and standards
* Increasing security through the next stage of testing
* By analysing the signalled errors, you can learn a lot about good practice in safe programming

*Why analyze the code?*

The cost of fixing the error depending on the time of detection.



Source: <http://www.microsoft.com/security/sdl/about/benefits.aspx>.