Abraxas Software

Understanding YOUR GOALS & Using CodeCheck

Implementing Corporate Source Code Guidelines

C/C++ Source Code GuideLine Automation

The goals of CodeCheck are:

1 To create a standard which will enable you to provide the customer with a quality product in a timely manner.

2 To promote standardization of software development among programmers.

3 To act as a learning tool for new programmers.

4 To act as a reference tool for all levels of programmers.

5 To promote a team approach among programmers.

6 To help programmers create readable, re-usable and maintainable software.

7 To help programmers write portable software.

8 To improve programmer productivity.

HOW TO ORGANIZE RULE FILE(S) for Corporate QA:

It's a bad idea to deliver too much information. Also normally the simple comment checking can be ran & fixed by anyone, but the deep control logic must only be done by your most expert programmer, its best to split your rule into standard & advanced.

Normally rule files are broken into the following "families"

// STANDARD - Non expert code modification allowable

1.) environment // verify code is "ANSI" compliant?

2.) char sets // legal chars?

3.) comments // header checking

4.) identifiers // hungarian, & other naming convention

5.) types // correct type checking

6.) constants // legal constant checking

7.) declaration & definitions // consistent decl/defn

8.) Initialization // legal intialization & formatting, ...

// ADVANCED - Requires careful code modification

9.) operators // mixed operators & complexity

10.) conversion // casting and mixed type's

11.) expression

12.) control flow

13.) function

14.) pre-processing

15.) pointer & array

16.) structure & union [class / template]

17.) standard library checking

Each rule file should be written for each of the above sections, and then each family concatenated into std/adv.

Most preferable is to check code by family if the total number of rules exceeds 1/2 dozen per family.

** TECHNIQUE

Its best to organize rule files by whom is going to analyze the results, normally the most number of errors will be in the most simple areas, e.g. standard rules. The most complex problems are less frequent.

It's best to organize by **metrics** so you can calculate the weight of all rules by family, and generate the results as CSV or DBF or HTML files. Then you can use a browser or database to view the hot-spots, then apply the rules by family to generate the textual solution to the problems.

So for the first pass you could have a total rule file that generated a database enumerating all errors by module, author, error-number, line-number. Then analyze the results to determine where the effort should go, and apply the rules that help solve each problem.

CodeCheck is capable of producing most common metric algorithms which are most useful for finding 'hotspots' in projects, and or modules. Lastly quantifying programmers by product complexity and/or problem generation.

** Standard Approach

CodeCheck-ing requires different approaches.

1.) repairing old code

    a.) study the old code by family

2.) keeping new code clean

    a.) have programmers run rules on their code daily to prevent the errors

3.) help management understand both of the above

    a.) management can have HTML or database data generated showing where the problems are.

** PROJECTS

When looking at the #include problems you need to use the 'project' approach, like make-files you need to look at everything. Anytime your looking at 'global' data you apply the project mode of CodeCheck. All other stuff is module based, you don't need to run by project.

Basically you need to break things down into three parts,

1.) Use database analysis of all rules on entire project to generate visual data on entire problem. Then decide plan of attack.

2.) Have programmers run family rule file on each module and fix problems in order of easiest to hardest.

3.) Have developers apply rules files daily for code enhancement & QA controls.

** DEVELOPMENT

* Development/Test Procedure

check file.c // make sure your code is compilable

check -rrule.cc // compile your rule file / develop - this generates rule.cco

check -rrule.cc file.c // test your rule file on real code

check -rrule.cco file.c // turn your rule file over to production/QA - they don't need the source for the rule file

.CCO file is an object file just like a .obj file,

CodeCheck doesn't use the .CC file when it runs on CODE, it uses the .CCO file, which is a OBJ-TREE.

GETTING TO WORK

If anything is not clear, if anything you can think of is not available, please contact support@abraxas-software.com and we'll send you what you need.