ics-fd-qa project

• Objective
• Present scope
• QA calling sequence
• Present status & tools
• Integration into projects & kobe
• Some issues
• Conclusion

• Your input, discussion
Objective

- Provide integrated code quality assessment
- High quality code scans, try to find “fishy” pieces automatically
- Report only false-true, keep noise down
- Graphical output with sonarqube, fancy, motivating
- Support and encourage easy documentation writing (sphinx, ...)

- Provide clear code fix proposals if available
- Provide q.a. history

- Have minimal impact on projects
- Existing build chains are sacred
- Be able to digest lots of legacy code and give orientation for “hot spots”
Present Scope

- ICS-FD section internal
- C/C++ code, no java, no control+
- Cmake build chain (qmake/Qt/bear come later)
- Static code analysis: clang-tidy, cppcheck (more tools can be easily added, but clang-tidy is the upshot)
- Push results to tmp-sonarqube, will have own fd instance later (dixit ics-in)
- Integrated with docker, gitlab-CI and uses a bit of nexus
- Therefore is kobe-compatible, see later
- Can update existing sphinx-doc “almost for free”: tools are included in the docker image
**Calling sequence**

**Push new sources**

**C/C++**
- project
  - gitlab
  - cmake

**Trigger Pipeline**

**Sonarqube**
Ics-fd qa webpage
- Code issues
- Proposed corrections
- Statistics
- Graphs

**Sonar-scanner transmission**

**Trigger with payload**
- Artifact URL
- Project name
- Control flags

**Ics-fd-qa (gitlab, nexus)**
Docker: clang 7.0.1
Python 3, Sphinx, Doxygen, Breathe, Sonar-scanner, break, cmake 3, ...

**stages:**
1. Fetch sources URL & unpack
2. analysis: clang-tidy, cppcheck, aux results, output & logs artifacts
3. Push to sonarqube using sonar-scanner

**Build:**
- all sources are available
- Produce a build command
- json-db

**ADDED stage:**
- Pull qa scripts from nexus
- Analyze json-db for full source dependencies
- Pack sources into artifact (URL)
- Trigger with payload

**Nexus Temp repo**
QA calling sequence

• works mostly blind. Nexus as a simple fix for dl qa-scripts
• Well isolated in extra project and docker, but some project adaption is needed
• Need to add
  • SET(CMAKE_EXPORT_COMPILE_COMMANDS “ON”) in CMakeLists.txt, for building
  • a “QA pack&trigger” stage to your project .gitlab-ci.yml, right after the build
• Kobe integration and refinements seem nicely possible, elegant trigger with payload
• Get hold of ALL code if possible including external/system headers...
• Somehow heavy:
  • Large docker image, complex
  • Source artifact potentially large =~whole OS, when complete sources
  • Hitting gitlab limits (log size) already, but can be fixed
• Enlarge scope by using break: create json-db for multiple build chains, part of standard cc7 ;-) have not tried yet
• Clang-tidy is much happier and performant with json-db
Integration into projects and kobe

- Project artifact “packed sources & json db” only available at the end of job, but trigger is earlier still during the job. Replace delay with another sequential pipelined job, pass project params in a file like kobe/riku do already
- Resolve detailed URL in the receiver (ics-fd-qa), not in the sender
- Adapt/refine trigger payload and switches, use defaults:
  - Defined no _WIN32 : only linux sources
- Project integration
  - ics-fd-qa scripts have to be pulled from nexus, maybe use gitlab as well (versioning...?!)  
  - Gitlab-ci.yml has to be updated: “QA pack & trigger” stage  
  - “do it the kobe way”: global stages, names and error codes  
  - Cmake/gcc -> clang-tidy: create a compile_command.json database:  
    - Add CMAKE_EXPORT_COMPILE_COMMANDS=“ON”  
  - should use bear for other (all) build chain types: i.e. qmake/gcc -> clang-tidy  
  - Building natively with clang would be ideal but is a long shot & complex project  
- All projects must provide a download token  
- Rename jobs, pipelines, variables to taste and standards
Some Issues

- Gitlab projects need to be public + repo token for download to work
- Minimize false-positives “where is the line?” – it should be low for start
  - configure clang-tidy for useful C++ checks
  - Configure sonarqube quality profiles, gates and dashboards
  - Only go for real problems in code
  - Exclude “coding style” in a large sense
  - Converge to a smallest common denominator, select rules from ~10k rules available
  - Messy cleanup & config business... “once and for all for everyone” .. iterative
- Sonarqube has some CTL rules – is this what I think it is ?
- Sonar Eye Candy: dashboards... keep it simple (80% is useless)
- Run “badCodeExample” for verification & test
Rules - status

• Profile ics-fd-qa has rules 3671 activated/10410 totally available=\sim35%

• C++/C language + pre-processor + some cognitivity/complexity rules + a very few good style rules to avoid bugprone code

• Rather “acceptable” quality threshold & gate, only find real issues

• Disabled things like:
  • Missing public API docs
  • Code repetitions
  • Naming conventions

• A single rule can make a big difference: careful backup & management ;-

• [http://slc6sboychen:9000](http://slc6sboychen:9000) (takes \sim5..10mins for one run)

• Rule: “if it is red, do something” ... non-red != “all is fine”, though
Conclusion

- Gitlab + docker can be well adapted to run QA as “appendix” to projects
- Maintenance of ics-fd-qa project in gitlab: essentially scripts & Dockerfile
- Report to own ics-fd sonarqube instance: no problem
- Setup, config, fine-tuning etc. is going on
- You are looking at a ics-fd-qa alpha-version
- Only linux sources analysed for now because
  - no “native copy”, need gitlab runners@windows, docker@windows is not available
  - Otherwise a cross OS solution might be possible:
    - Produce a json-db for windows build (break works only for linux)
    - statically analyse windows sources under linux... just pack the **complete** sources
    - Or run linux inside windows and share the disk... haha...
- Ready for your input!
- Valgrind, dr.memory, pc-lint, RATS, vera++, .... Seem all supported by sonar