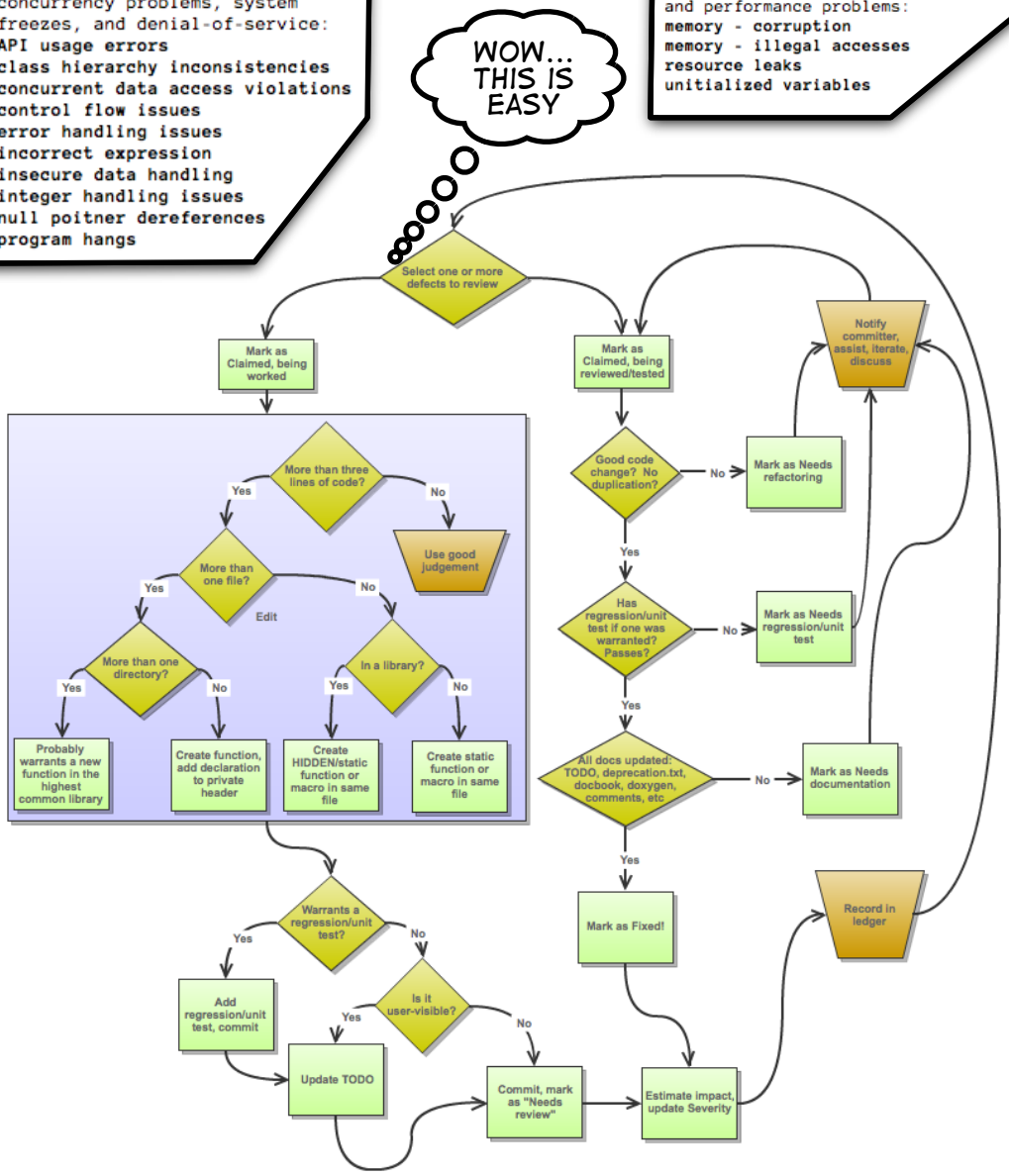
OFFICIAL BRL-CAD CODE REVIEW OFFSITE WORKSHOP CHEAT SHEET

lower-risk, medium-impact defects that cuase incorrect results, concurrency problems, system freezes, and denial-of-service:
API usage errors
class hierarchy inconsistencies
concurrent data access violations
control flow issues
error handling issues
incorrect expression
insecure data handling
integer handling issues
null poitner dereferences
program hangs

higher-risk, high-impact defects cause crashes, program instability, and performance problems:
memory - corruption
memory - illegal accesses
resource leaks
unitialized variables



Refactoring (noun): "a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior." Refactoring (verb): "to restructure software by applying a series of refactorings without changing its observable behavior." http://en.wikipedia.org/wiki/Refactoring

Zero One or Infinity (ZOI): "An entity should either be forbidden entirely, one should be allowed, or any number of them should be allowed." http://en.wikipedia.org/wiki/Zero_one_infinity_rule

Don't Repeat Yourself (DRY): "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." http://en.wikipedia.org/wiki/Don%27t_repeat_yourself

Rule of Three "The first time you do something, you just do it. The second time you do something similar, you wince at the duplication, but you can do the duplicate thing anyway. The third time you do something similar, you refactor." http://en.wikipedia.org/wiki/Rule_of_three_(computer_programming)

Cargo Cult Programming Using patterns and methods without understanding why. Examples include applying a design pattern without understanding the reasons behind that design principle in the first place, adding unnecessary comments to self-explanatory code, http://en.wikipedia.org/wiki/Cargo_cult_programming

THINGS TO WATCH OUT FOR

Dependency changes. Introduces new/cyclic dependencies? Push down.
API changes. Minimally impacting only. Update doc/deprecation.txt
Technical debt. No new debt. Do it right, refactor accordingly.
Duplicate code. Copy-paste is evil. More than 3, rule of three.
Bombs. Use sparingly, try returning empty/null/zero instead.
Missing tests. multiple defects + rule of three = new test
Contrived complexity. KISS.
Comments. Use them.

INSPECT, FIX, TEST, REVIEW, DOCS

UNIT TEST...

```
  cd src/libbu
Copy an existing test:
  cp test_progname.c test_bomb.c
Edit test to exercise function
  # create test cases for each input, boundary conditions,
  # invalid/null inputs, etc.  test one thing at a time.
Edit CMakeLists.txt, add new executable.  Example:
  add_executable(test_bomb test_bomb.c)
  target_link_libraries(test_bomb libbu)
Edit Makefile.am, add file to EXTRA_DIST (alphabetical):
      test_bomb.c \
Rebuild and run test
      cd ../.build
      make test_bomb
      test_bomb
```

REGRESSION TEST...

```
  cd regress
Copy an existing test:
  cp iges.sh step.sh # simple example
     or
  cp red.sh step.sh # robust example
Edit script to exercise application
  # try to not rely on printed output or whitespace-sensitive testing.
  # create and feed inputs to tool, test expected behavior or outputs.
Edit CMakeLists.txt, add new target.  Example:
  ADD_CUSTOM_TARGET(step-regress
    ${SH_EXEC} ${CMAKE_SOURCE_DIR}/regress/step.sh ${CMAKE_SOURCE_DIR}
    DEPENDS step-g
  )
Edit Makefile.am, add script to EXTRA_DIST:
      step.sh \
Rebuild and run test
      cd ../.build
      make step-regress
```