Software Metrics And Ignominy

“How to Win Friends And Influence People”

Lassi A. Tuura
Northeastern University, Boston
Overview

Introduction to Ignominy

Metrics
- Project metrics table
- Metrics defined
- Modularity vs. Quality

Ignominy dependency data and diagrams

Geant4 analysis
- Findings
- Recommendations

Drilling into Geant4 packages (demo)
These tools were developed in CMS IGUANA project
- Initially to control dependencies in our own project
- Later used to analyse potential external products

We have analysed several large software projects:
Anaphe, ATLAS, CMS, Geant4, ROOT

CMS has positive experience with this type of QA
- Significant improvements in release process (release layering)
- Has helped developers a lot to guide design and simply to clean up
- Systematic analysis and action on most CMS projects

I am not a Geant4 developer
- I wrote CMS G4 visualisation in IGUANA so I know some parts intimately
- Hopefully this material will be useful for improving quality of Geant4
Introduction

The diagram illustrates the IGUANA system's components and their relationships:

- **Source Code**: The input for the system.
- **Build Products**: The output of the system.
- **Dependency Database**: Stores logical dependencies.
- **Metrics**, **Graphs**, **Tables**: Outputs of the system analysis.
- **User-defined logical dependencies**: Custom defined dependencies.

**Model**

- Examines direct and transitive **source** and **binary** dependencies.
- Creates reports of the collected results:
  - As a set of **web pages**
  - Numerically
  - Graphically
  - As **tables**

*ignominy*: dishonour, disgrace, shame; infamy; the condition of being in disgrace, etc. (Oxford English Dictionary)

*ignominy*: a suite of perl and shell scripts plus a number of configuration files (IGUANA)
Analysis Results

Anaphe 3.6.1 Dependency

Dependency data

Dependency statistics

Metrics

Log | Warnings only

Extra detail: All | Includes | Includes (details)

C: Concentrated; L: Logical; M: Maximal Logical
B: Binary Only; PP: PACKAGE Dependencies
P: By Package Purpose; C: By Package Origin

<table>
<thead>
<tr>
<th>Project</th>
<th>Release</th>
<th>Packages</th>
<th>Average # of direct dependencies</th>
<th>Cycles (Packages Involved)</th>
<th># of levels</th>
<th>ACD*</th>
<th>CCD*</th>
<th>NCCD*</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anaphe</td>
<td>3.6.1</td>
<td>31</td>
<td>2.6</td>
<td>8</td>
<td>5.4</td>
<td>167</td>
<td>6.3</td>
<td>2 (92)</td>
<td>5.4</td>
</tr>
<tr>
<td>ATLAS</td>
<td>1.3.2</td>
<td>230</td>
<td>6.3</td>
<td>96</td>
<td>70</td>
<td>16211</td>
<td>7.0</td>
<td>2 (92)</td>
<td>7.0</td>
</tr>
<tr>
<td>CMS/COBRA</td>
<td>5.2.0</td>
<td>87</td>
<td>6.7</td>
<td>19</td>
<td>15</td>
<td>1312</td>
<td>4.9</td>
<td>4 (10)</td>
<td>1350k</td>
</tr>
<tr>
<td>CMS/IGUANA</td>
<td>2.4.2</td>
<td>35</td>
<td>3.9</td>
<td>6</td>
<td>5.0</td>
<td>176</td>
<td>2.9</td>
<td>4 (8)</td>
<td>200k</td>
</tr>
<tr>
<td>ROOT</td>
<td>2.25/05</td>
<td>30</td>
<td>6.4</td>
<td>22</td>
<td>19</td>
<td>580</td>
<td>4.7</td>
<td>1 (19)</td>
<td>660k</td>
</tr>
</tbody>
</table>

*) John Lakos, Large-Scale C++ Programming
Dependency Analysis

Ignominy scans...
- Make dependency data produced by the compilers (*.d files)
- Source code for #includes (resolved against the ones actually seen)
- Shared library dependencies (“ldd” output)
- Defined and required symbols (“nm” output)

And maps...
- Source code and binaries into packages
- #include dependencies into package dependencies
- Unresolved/defined symbols into package dependencies

And warns... about problems and ambiguities (e.g. multiply defined symbols or dependent shared libraries not found)

Produces a simple text file database for the dependency data
## Package Metrics

<table>
<thead>
<tr>
<th>Project</th>
<th>Release</th>
<th>Packages</th>
<th>Average # of direct dependencies</th>
<th>Cycles (Packages Involved)</th>
<th># of levels</th>
<th>ACD*</th>
<th>CCD*</th>
<th>NCCD*</th>
<th>Size</th>
<th>Own Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anaphe</td>
<td>3.6.1</td>
<td>31</td>
<td>2.6</td>
<td>--</td>
<td>8</td>
<td>5.4</td>
<td>167</td>
<td>1.3</td>
<td>630/170k</td>
<td>1.3</td>
</tr>
<tr>
<td>ATLAS</td>
<td>1.3.2</td>
<td>230</td>
<td>6.3</td>
<td>2 (92)</td>
<td>96</td>
<td>70</td>
<td>1621</td>
<td>10</td>
<td>1350k</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>1.3.7</td>
<td>236</td>
<td>7.0</td>
<td>2 (92)</td>
<td>97</td>
<td>77</td>
<td>18263</td>
<td>11</td>
<td>1350k</td>
<td>11</td>
</tr>
<tr>
<td>CMS: ORCA</td>
<td>4.6.0</td>
<td>199</td>
<td>7.4</td>
<td>7 (22)</td>
<td>35</td>
<td>24</td>
<td>4815</td>
<td>3.6</td>
<td>420k</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.1.0</td>
<td>385</td>
<td>10.1</td>
<td>4 (9)</td>
<td>29</td>
<td>37</td>
<td>14286</td>
<td>4.9</td>
<td>580k/57%</td>
<td></td>
</tr>
<tr>
<td>CMS: COBRA</td>
<td>5.2.0</td>
<td>87</td>
<td>6.7</td>
<td>4 (10)</td>
<td>19</td>
<td>15</td>
<td>1312</td>
<td>2.7</td>
<td>180k/24%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.1.0</td>
<td>99</td>
<td>7.0</td>
<td>4 (8)</td>
<td>20</td>
<td>17</td>
<td>1646</td>
<td>2.9</td>
<td>200k/29%</td>
<td></td>
</tr>
<tr>
<td>CMS: IGUANA</td>
<td>2.4.2</td>
<td>35</td>
<td>3.9</td>
<td>--</td>
<td>6</td>
<td>5.0</td>
<td>174</td>
<td>1.2</td>
<td>150/38k/49%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.1.0</td>
<td>45</td>
<td>3.3</td>
<td>1 (2)</td>
<td>8</td>
<td>6.1</td>
<td>275</td>
<td>1.3</td>
<td>150/60k/48%</td>
<td></td>
</tr>
<tr>
<td>Geant4</td>
<td>3.2</td>
<td>108</td>
<td>7.0</td>
<td>3 (12)</td>
<td>21</td>
<td>16</td>
<td>1765</td>
<td>2.8</td>
<td>680k</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.2</td>
<td>135</td>
<td>6.4</td>
<td>4 (26)</td>
<td>31</td>
<td>20</td>
<td>2728</td>
<td>3.3</td>
<td>710k/55%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.0p2</td>
<td>135</td>
<td>6.4</td>
<td>3 (25)</td>
<td>33</td>
<td>22</td>
<td>2936</td>
<td>3.5</td>
<td>770k/55%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.1</td>
<td>137</td>
<td>6.6</td>
<td>3 (25)</td>
<td>34</td>
<td>22</td>
<td>3058</td>
<td>3.6</td>
<td>870k/54%</td>
<td></td>
</tr>
<tr>
<td>ROOT</td>
<td>2.25/05</td>
<td>30</td>
<td>6.4</td>
<td>1 (19)</td>
<td>22</td>
<td>19</td>
<td>580</td>
<td>4.7</td>
<td>660k</td>
<td></td>
</tr>
</tbody>
</table>

*) John Lakos, Large-Scale C++ Programming

- ACD = average component dependency (~ libraries linked in per package)
- CCD = sum of single-package component dependencies over whole release: test cost
- **NCCD = Measure of CCD compared to a balanced binary tree**
- Size = total amount of source code (roughly—not normalised across projects!)
- Own = percentage of own code (size minus comments, white space, generated code)
What’s This NCCD?

Defined in John Lakos’ “Large Scale C++ Programming”
  • A “must read” for all developers!

NCCD = Measure of CCD compared to a balanced binary tree
  • Measures the degree of coupling in the system
  • < 1.0: structure is flatter than a binary tree (= independent packages)
  • = 1.0: structure resembles fully balanced binary tree
  • > 1.0: structure is more strongly coupled (vertical or cyclic)

Aim: Minimise NCCD for given software/functionality
  • A good toolkit should have a value ~ 1.0
  • The aim is not to artificially reduce the NCCD
    ▪ Easy e.g. by copying code or with dubious obfuscating acrobatics
  ... but to design the same software (= functionality) with desired NCCD value
**Metrics vs. Quality**

- **NCCD measures mainly modularity**
  - The main benefit is that it is relatively easy to determine

- **Modularity is not quality, only a necessary ingredient**
  - The goal is not to achieve modularity but good design
  - A good toolkit is modular, but a modular system is not necessarily good
  - Should still observe traditional OO and non-OO metrics
    - # of methods per class, disjoint uses of classes, cyclomatic complexity etc.

- **In our experience NCCD is a good “first-line” indicator of the general quality of the software project, but it doesn’t measure**
  - How responsive the developers are
  - How good user interface it has
  - How feature-complete it is
  - How stable or buggy it is

*Use valgrind TODAY!*
In addition to NCCD Ignominy determines other variables

Package cross-dependency tables and charts
- From symbols, headers, user-defined, combined
- Against individual packages plus summarised
- Chart with packages against each other with user-defined sorting

Per-package data
- Forward and reverse directions
- Source, binary, user-defined dependencies
- Hierarchically for packages, subsystems, projects
- Package dependency diagrams with various options
- Detail: which symbols, headers caused dependency

Average number of dependencies per package, amount of code
Single Package Dependencies

Cmscan/IgCmscan

*Testing Level:* 5

*Outgoing edges:* 6
- from includes: 6 (145 files)
- from symbols: 4 (636 symbols)

*Incoming edges:* 1
- from includes: 1 (1 file)
- from symbols: 1 (1 symbol)
Domain Test Plan

Diagram: Cmscan
- Ig_Extensions
- Ig_Imports
  - CERNLIB
  - Ig Utilities
  - LHC++
- System
Package Impact Diagram

“Used-by” dependencies
An Extra Dependency

Bad dependency in prototype code; was resolved to be from bad class placement

1 IgSoReaderAppDriver ⇒ IgQtTwigBrowser via IgQtTwigModel.h
1 IgSoReaderAppDriver ⇒ IgQtTwigBrowser via IgQtTwigRep.h
Logical dependencies from packages used through “Interfaces”
Lack of a good configuration management tool
- Analysis of build or release areas by external tools is desperately difficult

Deep package structure is confusing

Two package dependency loops degrade quality significantly
- One central loop has significant influence (via ApplyCommand())
- Visualisation is tightly coupled but does not influence overall metrics much

Overall design seems relatively clear and clean, but...
- Number of package levels is high for a toolkit
- Average number of edges per package is high
- Some/many subsystems are in good shape!
- Thank you for clean design that lends itself easily to analysis (e.g. distribution of classes and in particular abstract interfaces to packages)
Dependency Hazard Zones

- Important to check
- Packages to check

Incoming Edges vs. Outgoing Edges graph with data points and zones highlighted.
Recommendations

Infrastructure
- Introduce a configuration management tool
- Reduce package structure to two levels

Better package levelisation
- (Remember to remove unnecessary includes first!)
- Analyse package cross-dependency tables
- Decide which parts can be cut off from each other
- Aim to reduce visibility across system
  - Splitting packages and/or more encapsulation?

Main design issues to tackle
- Get rid of the central dependency loop
  - May require rethinking some central object structures
- Redesign top-level visualisation structure (at least)
We’ve found this type of analysis useful for developers

Ignominy and related utilities are a part of IGUANA releases

• Analysis results for IGUANA are part of our release documentation
• IGUANA is open source—all you find is free to use
• Tools and IGUANA diagrams at http://iguana.cern.ch
• Questions and discussion at iguana-developers@cern.ch

Analysis results from various projects

• Explore the web pages at http://cern.ch/~lat/deps/
• Diagrams available according to my quota situation...
Drilling into Geant4 packages using the Ignominy-generated web output