ACCU Spring Conference 2004

14.-17. April 2004, The Randolph Hotel, Oxford

About this document

I was asked by my employer to give a presentation on the ACCU 2004 Conference and the presentations I attended. This is a slightly edited version of that presentation.

These slides were mostly compiled from my notes. Any factual or spelling errors you may find are almost certainly my fault - please report them.

An updated version of this document may or may not be available at

http://www.haun-online.de/accu/

Dirk Haun
dirk@haun-online.de
2004-05-08

Day 1: Wednesday, 14 April

- *World Domination - The First Five Years*
- *Code Craft*
- *Report from the C and C++ Standards Committees*
- *The Economics Of The Design Process*
- *Beyond Methodology*
- *The IT View*
- *Successful Open Source Advocacy*

World Domination - The First Five Years

Eric S. Raymond

- *“First they ignore you, ...”*
- *there is a phase in SW development where secrecy actually makes sense - when you’re ahead of the competition*
- *could do without the GPL - stealing OSw will not pay in the long run*
- *“use-value” vs. “sale-value” (boxed) SW*
- *separate business logic from the engine*
- *competition will have to figure out how things work - wasting their time*
- *you’ll never know who will look at your SW - and it will change the behaviour of your developers to know this*

Code Craft (1)

Pete Goodliffe

- *What makes a good programmer?*
  - *Attitude: "The orientation (of an aircraft's axes) relative to a reference"*
- *Code Monkeys*
  - *Programming is a social activity - programmer's attitudes matter*

Code Craft (2)

Pete Goodliffe

- *The ideal programmer*
  - *Technical genius*
  - *Artist*
  - *Relational*
  - *Politician*
- ☑ *P.R.A.T.*
- *Since we work in teams, individual attitudes affect the quality of the team*
  - *to improve, you must change*

Code Craft (3)

Pete Goodliffe

- *Goal:*
  - *Team player*
  - *Honest and Humble*
  - *Improving constantly*
  - *Considerate*
  - *Keen*
- ☑ *T.H.I.C.K.*
- ☑ *T.H.I.C.K. P.R.A.T.* ☺

From the C / C++ Committees

Francis Glassborow

- *next C standard: C2004 (or C2005)*
- *Defect Reports only, no other changes*
- *C will be put into "maintenance mode" after that, i.e. no changes for the forseeable future*
- *changes planned for C++ 2009*
- *e.g. unified initialiser syntax*
  int x = 0; vs. int x(0);

The Economics of the Design Process

Hubert Matthews

- *Constraints*
- *Variation*
  - *Planned (flexibility) vs. unplanned (deviation from specs)*
  - *no economic value in reinventing the wheel*
  - *manufacturing: variation is unplanned*
  - *sw development: variation is what we're selling*

Beyond Methodology (1)

Alan Kelly

- *Do we like to use methodologies?*
- *300 methodologies?!*
- *Budgets are methodologies but make things more difficult*
- *Methodologies undermine commitments*
- *People hide behind methodologies*
- *Beyond methodology: People, Communication, Change*

Beyond Methodology (2)

Alan Kelly

- *Finding People*
  - *Good people have more than just technical skills*
  - *In an interview, aren't they asking all the wrong questions?*
- *Communication*
  - *Referring people to documents instead of talking to them!?*
  - *Build in feedback loops*
- *Change*
  - *Embrace change and work for improvement*
  - *Success is dangerous: no change - failure encourages change*

Day 2: Thursday, 15 April

- *Is C++ Relevant on Modern Environments?*
- *Funding Open Source*
- *Open Source licenses 101*
- *Linux at IBM*
- *ACCU Journals online*
- *Test Driven Development Workshop*
- *Open Source in Education*

Is C++ Relevant on Modern Environments?

Herb Sutter

- *Windows XP SP1 already ships with .NET*
- *as of Longhorn, .NET will be the OS API ("WinFX")*
- *any language not supporting this API will be marginalized*
- *hence: C++/CLR*
- *extensions to C++*
- *without breaking existing code*
- *in the process of being standardized*
- *C++ people and organisations involved (e.g. Bjarne Stroustrup, EDG, ...)*

Funding Open Source

- *Some foundations post so-called bounties as a reward for developing Open Source Software (e.g. Mark Shuttleworth)*
- *The EU also has funds (for new software only) but it involves some amount of paperwork*
- *“Sponsoring” of bugfixes / implementation of new features*

Open Source licenses 101

David Ascher

- *desire to share vs. desire to make money*
- *license is an implicit contract*
- *4 types: GPL, LGPL, "liberal" (BSD, Apache), "corporate" (Apple, Netscape)*
- *GPL is often chosen for simplicity, not because the author really meant it (or agreed with all of RMS views)*

Linux at IBM

Moore

- *adding the things businesses are looking for to Linux*
- *running benchmarks, looking for bottlenecks*
- *“no use for us if it’s not accepted by the community”*
- *Linux widely used within IBM now*
- *z-Series (390, 60’s technology) revived*
- *several 100 Linux projects at IBM*
- *Linux allows you to ”move sideways”*

Test Driven Development Workshop

Ivan Moore, Duncan Pierce, Rachel Davies

- *don't write code for something unless you have a test for it*
- *writing the test will help you identify what is expected (from the code / class / appl.)*
- *design evolves with the tests*
- *refactoring*
- *tests should be self-contained*

Day 3: Friday, 16 April

- *ISO9001:2000 - Something Old, Something New..*
- *Class Design - a checklist approach*
- *Revision Control - Leave branching to the trees*
- *Visual C++ Presentation*
- *Python as a testing tool*
- *Scripting C from Python*
- *Honey, I Shrunk the Threads*
- *ACCU Accreditation*

ISO9001:2000

Neil Martin

- *quality has become associated with procedure*
- *not much evidence of improvement in software quality*
- *too much emphasis on process, resulting in a negative image of quality*
- *ISO9001:1994 vs. ISO9001:2000 - emphasis on written procedures is gone*
- *ISO9001 is a standard for management, not necessarily for quality management*
- *we still don't know how to produce software of a predictable quality*

Class Design - a checklist approach

Alan Bellingham

- *“Pilots use checklists - programers tend to just jump in”*
- *checklist too C++ specific ...*
  - *are the compiler-generated c’tors / d’tors / etc. sufficient?*
- *following the checklist ensures that we don’t have unwanted behaviour of our class*

Leave branching to trees (1)

Stewart Brodie

- *"HEAD case": check out the latest version and ship it*
  - *hard to reproduce faults*
  - *this is not source control*
- *"One Big Project model": repository tagged*
  - *so you can get back and reproduce problems*
  - *only suitable for small teams or projects*

Leave branching to trees (2)

Stewart Brodie

- *"One branch per customer"*
  - *bad: have to apply changes to all branches*
  - *good: avoids customer-specific modifications "infecting" the code*
- *developing on the branches*
  - *difficult to keep track what has been merged back*
  - *but you may have to do this if a lot of programers work on the same sorts of files*

Leave branching to trees (3)

Stewart Brodie

- *split up software into meaningful components*
  - *avoids people making changes to the same files*
  - *requires strong interface design*
  - *leads to replaceable components, ideally*

Leave branching to trees (4)

Stewart Brodie

- *interface becomes important*
  - *i/f changes must be thought through*
  - *minimize version dependencies*
  - *simplifies design of unit tests*
  - *you can assign ownership of code*
- *management of version numbers*
  - *which version of which component go into the build?*
  - *requires strong tagging*

Leave branching to trees (5)

Stewart Brodie

- *Configuration management*
  - *list of components (and their versions) that go into the build*
  - *controlled by the "build master"*
  - *ongoing development can't break the build*
  - *simple text file with (module, tag) pairs*
  - *under CVS (Subversion, ...) control*
  - *needs to be actively changed to get new versions into the build*

Python as a testing tool

Chris Withers

- *tests help define the problem you are solving*
- *automate tests*
- *ideally, tests are written before the actual code or when you find a bug*
- *unit tests - smallest unit of functionality as possible*
- *test edge cases + some typical cases*
- *if the test doesn't buy you anything - don't do it*

Scripting C from Python

Duncan Booth

- *Python C API*
- *ctypes lets you import C functions from a DLL or .so*
- *Pyrex: Python syntax with extensions*
  - *define classes implemented in C*
  - *define C functions*
- *Boost::Python for C++*

Honey, I Shrunk the Threads

Andrei Alexandrescu

- *writing multi-threaded code is hard*
- *requires skill, discipline, and attention*
- *but you can have the compiler help you with ensuring correctness of your MT code*
- *library tools make it easier to write clear code*
- *efficient - no spurious locking, not much overhead*
- *still no free lunch, though - requires discipline in using the tools*

ACCU Accreditation?

Neil Martin, Alan Griffiths, et al.

- *there is a need for an independent certifcation of good programmers*
- *obviously, we do not want this to be bound to some company (MCSE, ...)*
- *is there a role for the ACCU here?*

Day 4: Saturday, 17 April

- *Practical Excellence: A Perspective on Software Quality*
- *Writing Exception Safe Code*
- *ACCU AGM*
- *Interviewing Skills: a panel*
- *All Heap No Leaks*
- *(C++ Templates in Depth)*

A Perspective on Software Quality (1)

Chuck Allison

- *"Software Entropy": instead of refactoring, it's on to the next project*
- *"Technical debts" are summing up (exponentially), make changes harder*
- *"Technology du jour" doesn't help much if you can't handle the basic principles*

A Perspective on Software Quality (2)

Chuck Allison

- *don't repeat yourself (One Definition Rule, ODR)*
- *find invariants and assert them*
- *separate interface and implementation*
- *Test Driven Development*
- *pair programming (review as you go)*
- *automate builds, test, deployment*

A Perspective on Software Quality (3)

Chuck Allison

- *study good code (→ Code Reading)*
- *write to be read*
- *avoid premature optimizing*
- *design in public places (whiteboard)*
- *take pride in your work*

Writing Exception Safe Code (1)

Andrei Alexandrescu

- *Coding with exceptions is coding transactionally*
  - *i.e. need to ensure operations are done atomically*
  - *avoid "meta-stable" state*
  - *undo everything that has been done before the exception occured*
- *exceptions are assumption-violations*
- *assert for things you can't recover from*

Writing Exception Safe Code (2)

Andrei Alexandrescu

- *Brute Force: try {} catch { undo; throw; }*
  - *doesn't scale; nesting*
- *Politically Correct: Resource aquisition*
  - *efficient, elegant, scales*
  - *needs a class for every operation*
- *Real Life Approach: no exception handling*
- *ScopeGuard class*

All Heap No Leaks (1)

Paul Grenyer

- *What is a Memory Leak?*
- *How do we prevent Memory Leaks?*
- *Smart Pointers*
  - *a class maintaining a pointer to a dynamically allocated object and acting as if it was that object*
  - *automatically destroys the object when it goes out of scope*

All Heap No Leaks (2)

Paul Grenyer

- *Heap-only objects*
  - *objects that are created in one part of a program and deleted in another*
  - *objects that have no real concept of copying*
- *Forcing Heap creation*
  - *a Smart Pointer class can do this for us*

All Heap No Leaks (3)

Paul Grenyer

- *Enforcing Smart Pointer usage*
  - *make the heap-only object's constructor private*
  - *add a factory method that returns a smart pointer*
- ☑ *Voilà - no more memory leaks*