Lessons from the C++11 Standard

Who Am I?

- Alisdair Meredith
- ISO committee member since 2003
- Current Library Working Group Chair

What is this session?

- War Stories
- An appreciation of the standard specification
- Some useful library design guidelines

First Standard Meeting

- Oxford 2003
- Meeting specified the majority of TR1
- Bjarne opened Evolution for C++0x submissions
- Attempt to remove ‘export’ from the language

My Proposal

- N1479 A proposal for ‘array’
- Assumed I should have something to show and tell
- Wrote up the simplest class I could imagine, citing 3 prior versions in print, plus a Boost library!
- Now published in the C++11 standard

What did I learn?

- Proposal is not necessary to attend!
- Proposal is probably the best way to get involved
- The standard has a distinct way of saying things - expect to rewrite!
- Specification is all that matters
  - implementation provides useful feedback

Library TR 1

- Number of disparate library extensions
- Many drawn from Boost
  - New committee members
  - Well tested and documented libraries
  - User doc is not a library specification
- Other contributions from existing members

Towards a new Standard

- TR1 a valuable experience
  - useful libraries
  - training for standards process
- Most of TR1 adopted directly into C++0x
- Math functions became a separate standard

What is the C++ Standard Library?

- A collection of classes and functions?
- useful code shipped with the compiler?
- A vocabulary?
- A fundamental part of the language?

Vocabulary

- Container
- iterator
- CopyConstructible and other requirements
- string
- streams

The C++ Standard Library is

- A specification, not an implementation
- All public interfaces are documented
  - Implementation details may be hinted
- Contract between users and implementers
  - Specification must be clear and unambiguous
- Implementations may make distinct choices

A standard Library for C++11

- Provide More facilities
  - Concurrency
- Support and exploit new language features
- Resolve bug reports
- Clearer and more consistent text

A Better Specification

- From Concepts to Requirements
  - Consistent and simplified wording
- Eliminating weasel words
- Better organization
  - Several clauses moved around
  - `bitset` is no longer an associative container!

Concepts

- Key language feature proposed by evolution
- A ‘metatype system’ constraining template parameters
- A complex feature to solve a lot of problems
- Fundamental to a successful C++0x library
- See Larisse Voufo’s session tomorrow

Concepts in C++03

- Many concepts implied by existing library
- Iterators provide a good model
- Algorithms make good use
- Containers less clearly specified
- C++03 describe code by valid syntax
  - concepts describe semantics too

Concepts Mismatch

- Syntactic concepts for backwards compatibility
- Semantic concepts for new code without backwards compatibility concerns
- Doubled the number of concepts
- Many more fine-grained concepts
- Design space no longer simple

What Did We Learn?

- Many existing library 'concepts' are underspecified
- Too many requirements have exemptions
- Named requirements clauses support clear and consistent specifications
- Requirements vocabulary a useful product in its own right

The End Result

- A growing number of named requirements
- Many requirements gathered together into the library introduction, to be referenced throughout the library
- Requirements are specified much more precisely, with fewer escape clauses
- Container requirements are still special...

Concept Based Overloading

- A key concept feature to direct overload resolution based on matching concepts
- Syntactic emulation possible with SFINAE
- SFINAE is user to (ab)use with library utilities
  - `enable_if`
  - type traits
  - `declval`

SFINAE in the Library

- Many function templates are required to use SFINAE to match only compatible arguments
- Several library components use SFINAE-based detection techniques to reduce their set of requirements, by providing defaults for missing features
  - C++11 allocators much simpler to write

Weasel Words

- Is ‘size’ always a constant time operation?
- Can calling ‘begin’ invalidate an iterator?
- Is a default constructed container empty?
- Are all containers EqualityComparable?
- Are all random access iterators mutable?

Allocator Weasels

- An implementation may assume:
  - All instances of a given allocator type are required to be interchangeable and always compare equal to each other.
  - The typedef members pointer, `const_pointer`, `size_type`, and `difference_type` are required to be `T*`, `T const*`, `size_t`, and `ptrdiff_t`, respectively.

Language Changes are Disruptive

- New language features affect the library
- Especially those that affect interface design
  - rvalue references vs. lambda expressions
- Early adoption hurts when the language feature evolves
- Early adoption gives feedback for language designers, in order to evolve

rvalue references

- Two key applications
  - move semantics
  - perfect forwarding
- Library utilities for users to exploit feature
  - std::move
  - std::forward<T>

Applying Rvalues

- Large, disruptive change affecting many library clauses
- Happened as TR1 integrated, so many missed opportunities (caught later)
- Subsequent proposals evaluated on the expectation of an rvalue-enabled interface

Rvalue Idioms

- Move support: add two overloads
  - `const & value_type` to copy lvalues
  - && value_type to move from rvalues
- Perfect forwarding: a single signature
  - ```
    template<typename T>
    void func(T && argument);
    ```
    ```

Problems!!

- By initial rules, a function taking a && does not know if it is passed an lvalue or an rvalue
  - relies on an lvalue overload being in the overload set to ‘steal’ lvalues
- New rules!
  - lvalues never bind to rvalue references
  - only library impact : move/forward utilities

Late Problems!!

- Implicitly generated move constructors
- Explicit move constructor deletes implicit copy constructor
  - many library types became ‘move-only’
- Language feature became contentious and unstable
- Not resolved until the final meeting!

Compatibility Problems

- vector move constructor could not offer the strong exception safety guarantee
  - yet will be called when some existing code recompiled unchanged!
- Problem: copy constructor called by overload resolution if no move defined
- `vector<pair<string, user_type>>`

No easy solution

- Invent a new language feature long after the final deadline!! (CD1 ballot resolution)
- `noexcept` exception specifications
- `noexcept` operator
- Test for a `noexcept` move, otherwise use safe-but-expensive copy

How Does noexcept apply to the library?

- A new language feature to adorn all our interfaces!
- Potential of many late-breaking edits
- Many library APIs are documented as not throwing an exception
- Only a few *need* the feature to solve the container problem

Library Guideline for `noexcept`

- Library is a specification, not an implementation
- Wide vs. Narrow contracts
- Specify only those places that noexcept is needed, or is guaranteed and always defined
- Vendors may use in implementations as an optimization

Experience is Vital

- Late changes often due to late experience with a feature
- No (core) specification was unchanged following implementation
- Library TR1 features had notably less churn than other parts of the library
- Concurrency library churned every meeting

Sometime Late Invention is Necessary

- Concurrency is possible *the* key feature of C++11
- Lack of a concurrency library would have been tragic
- No single clear library API, or even semantic
- thread cancellation particularly contentious

Proceed with Caution

- Threads library modeled after Boost, utilizing new language features
- Try to establish a clear set of goals to know when ‘done’
- Library design tracked almost live by the working paper!
- Boost tracking the standard invaluable

ABI Matters

- Vendors represent many customers, and breaking their code is not an option
- ABI breakage is far more subtle than API breakage - not everyone can recompile
- Not all ABI breakages are equal
  - Loss of CoW string broke HP
  - `ios_base::failure` less of an issue

Type Erasure is Good!

- std::function
  - target functor
  - allocator
- std::shared_ptr
  - deleter
  - allocator

More on Type Erasure

- unique_ptr vs. shared_ptr
  - do not always want to pay the cost
- boost::any
  - Someone should write this up!
- boost::filepath
  - a different kind of erasure

Key Lessons

- Library is a specification
- No substitute for experience
  - Sometimes we must proceed anyway
- Language changes affect the library
  - Late changes break things!
- Not all language changes are equal

C++11 Library

- Move Semantics
- Type erasure
- Widespread use of SFINAE
- Support new language
- TR1
- Concurrency

The Future!

- More work delivered working in parallel
- New Study Groups focus on specific areas
  - Deliver more frequent specifications as study groups complete projects
- New proposals spawn new Study Groups
- Work on main standard continues

Study Groups

- SG1 Concurrency
- SG2 Modules
- SG3 Filesytem
- SG4 Networking
- ... Numeric facilities