Organizational Leadership with Modern C++

By : Kevin Kostrzewa kevin.kostrzewa@thomsonreuters.com
and John Wyman john.wyman@thomsonreuters.com
cppcon15

What is organizational leadership?

How do I know if I am doing it?

Some background on us...

- We started in C… in MS-DOS
- We develop on the Microsoft stack
- We have about 75 developers with long tenure, and 175 total members of our development staff.
- ~11M LOC.
- Ann Arbor Michigan, a great development community, but not quite costal.

C++ changes just keep getting faster.

1983 we get… C++
1998 we get… C++ 98
2003 we get… C++ 03
2007 we get… C++ 07/TR1
2011 we get… C++ 11
2014 we get… C++ 14

2017 we get… C++ 17

This is great. I don't want to live in a world without lambdas… However

- How do we stay current?
- How do we keep my development organization current?
- How do we keep managment current?
- Which language features/libraries are relevant to our project(s)?

Mo' features mo' problems...

Making changes can be like...

1. Educate Yourself

- This is a bit of preaching to the choir.
- Keep your personal knowledge current.
- Stay plugged in.
  - Reddit/r/cpp
  - Channel 9
  - Attending a convention.
- Keeping yourself current is the first step in keeping your organization current.
- Having current leadership is the first step to making good design decisions and allocating development resources wisely.

Educate Yourself

- **Education cannot be in a vacuum**
  - **What we learn means much more if it is shared**
  - **How do we accomplish that?**

Educate Yourself

- **As a tangible example, this conference. This conference provides access to some of the luminaries of the C++ language. How do we use this?**
  - **Ensure that Senior developers are aware of the conference schedule.**
  - **Ask for input regarding session attendance. Perhaps they see value in sessions that we do not.**
  - **Even if there is no input, you still have raised topic awareness.**

Educate Yourself

- **How do we bring this home?**
  - **Trip Report.**
  - **Require notes from each session attended. Does it pertain to what we are doing now? Will it be useful in the future? If so, how?**
  - **Don’t always attend the same sessions. There are two of us.**
  - **Present report to lead developers at next scheduled leads meeting.**

Educate Yourself

2. Encourage Self Education

- Raise awareness of the dynamic nature of the C++ standard and it’s libraries.
- Access to computing educational resources is available to anyone with an internet connection.
- Self educated developers become self starters.
- Able to contribute knowledge that others in the development department may not have.

Encourage Self Education

- Appealing to management
  - Generally inexpensive.
  - Improves developer engagement
  - Improves software quality.
- Allows for individual specialization
- Huge number of free resources, especially around C++
- Be respectful our products shipdates

Encourage Self Education

- Weekly meetings of peers
- Opportunity for idea sharing, open to all staff.
- Open access for all, we don’t want a “keeper of the tomes of knowledge” culture.
- Non-critical listening inside of meetings. Keep the tone professional.

Encourage Self Education

- Educational micro-assignments
  - Utilize cyclical nature of our development cycle.
  - A great opportunity for individual discovery
  - Video watching and reporting.
    - Work in pairs, a lead and a junior member.
    - Youtube cppcon channel. MS Build.
    - Does this map to our solution space right now? Could it in the future.
  - Every assignment ends with an internally published write up. This information must be share.

Encourage Self Education

- Educational macro-assignments
  - Utilize cyclical nature of our development cycle.
  - Upgrading compiler / tooling / library versions.
  - Developers making our last compiler migration refactored mutable containers to const containers created from initializer list.
  - Developers self educated and improved code quality

Encourage Self Education

- Advanced concepts aren’t always easily mapped to your problem space.
  - Think globally, act virally
  - SFINAE anyone?
  - We have “virally” started off developers with an implementation to demonstrate value.

Self Education

3. Encourage use of best practices

Best Practices

- **Aligning solution space with problem space**
  - Developers know the correct tools
  - Developers understand the tools
  - Create a staff that is self sufficient
  - Have similar problems solved in similar ways by all developers
  - Easier for developers to switch tasks or pass
  - Consistent approach makes it easy to onboard new developers

Your style guide.

- **Adopt a style guide.**
- **Make the style guide your own.**
- **Make your style guide a continuing conversation.**

Refactoring across 11 million LOC was difficult.

- **Most developers don’t have the entire codebase local to their machine.**
- **Our old solution was network share with with our whole codebase on it.**
- **We just did brute force… Until**

We added indexing to our code repository. It is hard to refactor 11m LOC w/out an indexer.

- **Now we use OpenGrok. Now a global search is in the order of a second.**
- **Other products available.**

Email, where good information goes to die.

- Explore email alternatives.
- We use an internal social media site for conversation.
- We also use an internal wiki to persist documentation.
- Critical decisions must be transparent and publicly documented. This raises awareness and spurs conversation.

Best Practices

4. Limit use of worst practices.

Worst practices

Worst practices as code.

- Prevent unwanted libraries.
- Prevent std::auto_ptr.
- Prevent header injected namespace pollution.
- Stop bad code as early as you can. static_assert is your friend.
- static_assert + type_traits is great for enforcing preconditions with templates.

Worst practices as process.

- **CI/builds catches mistakes early.**
- **Gated builds are even better.**

Technical debt is inevitable. How do we own it?

- **static_assert in your code against compiler and library versions (external forces)**
- **This risks breaking the build when the external forces change**
- **The value of good comments around these static_asserts. These will change someday. What happens with a broken build. Including use cases for investigating.**

Worst practices

5. Lead through API

Lead through API

- **Why use API?**
  - This is for internal consumers. This is not a public API
  - Consumers are forced to deal with/understand new parameters and return types.
  - The downside is that the API owner is on the hook for documentation. Nobody likes writing documentation.

Lead through API

- Changes have varying need for changes by the consumer.
  - FOO &bar(FOO *)
  - FOO bar()

Lead through API

Use case

- **Legacy callback from C days**
- **Domain model creation along with product callback**
- **Function pointers used like this lead to global state**

Lead through API

Use case

- **AddInSh(void (*)())**
- **AddInSh(std::function<void ()>);**

Lead through API

Conclusion

- Keeping development skills current is a part of organizational success.
- The velocity of the industry is only increasing.
- This includes product leaders down to junior developers.
- There are deliberately no right answers here, these are not one size fits all problems.