Agenda

McAfee
An Intel Company

- Introduction: Software quality
- Adopting new programming standards
- Platform agnostic APIs
- Heavy but under or un-used code
- Blocking to non-blocking
- Conclusion

Introduction: Software Quality

We try to make a rock solid product and call it a quality one.

While that is true..it does not stop there

A software might be of high of quality at a given time. But we need to look at factors that would affect the quality in the long run. For example,

- Maintenance. A cleaner code is easier to maintain. Do we still write complex code to achieve something that could have been accomplished by other means?
- Scalability. A product that is not scalable may not win in the quality space forever.

Adopting New Programming Standards

This section focuses on software written in C++ programming language.

We strive to make our software cleaner, safer, and faster.

The new programming standard, C++11, has got a lot of features to get your software there.

Adopting New Programming Standards…

- If your software is based on older standards (say C++ 2003 or C++98) try adopting new standards.

- Agreed, there are challenges in adoption. Starting from upgrading your compiler to learning what is new in the latest standards and how to apply them.

- Even if you have achieved the functionalities provided by the new features in your own way, it is worth considering what the standard library has to offer. Remember standard library is better than handcrafted specialized code.

Adopting New Programming Standards…

- A simple use case

- To store the sum of two elements of vector of type ‘U’ and ‘V’ in a variable, say ‘sum’, and use ‘sum’ at some point later, we would write something like this

```
template <class U, class V>
void f(const vector<U>& v1, const vector<V>& v2) {
vector<U>::const_iterator v1_iter = v1.begin();
vector<V>::const_iterator v2_iter = v2.begin();
// ...
int sum = *v1_iter + *v2_iter;

// ...
// use ‘sum’ here.
}
```

Adopting New Programming Standards…

- If U is of type integer (say 10) and V of type double (say 20.5), then “sum” is 30. The summation result is lost. The type of “sum” should be double to hold the correct result. In other words the type of “sum” should be the one that you get from adding U with V.

- Changing the type to “long double” may not be desired if none of the vectors contain irrational numbers.

- One has to solve this problem the hard way today. Probably leading to hacks and workarounds.

Adopting New Programming Standards…

- The new standard has introduced “auto”(The C++ Standards Committee 2013) to solve this problem. “auto” deduces the type of a variable from its initializer. Now the code will look like

```
template <class U, class V>
void f(const vector<U>& v1, const vector<V>& v2) {
vector<U>::const_iterator v1_iter = v1.begin();
vector<V>::const_iterator v2_iter = v2.begin();
// ...
auto sum = *v1_iter + *v2_iter; // type of sum is
what you get from adding U with V
// ...
// use ‘sum’ here.
}
```

As you can see, it has not only solved the problem but also made the code cleaner and understandable.

Platform Agnostic APIs

Different OS(Operating system) provide different APIs to acquire and use system resources (creation of threads, for example).

Writing cross-platform software takes time. We will end up writing huge amount of code to build platform abstract library.

Platform Agnostic APIs…

- Maintaining and updating this library on addition of support to new platforms or on upgrade of target platform may be expensive.
- Resource allocation or any OS specific operation is outside the boundary of software's business logic. We should try to keep the direct dependency on OS specific API as minimal as possible.
- You could either see if the newer standards have got ways (features) to address this issue or use a third party library that provides cleaner abstraction.

Heavy But Under or Un-used Code

- Software when started from the grounds up stays lean and focused on the solving problem at hand. But over a period of time it might tend to attract more functionalities and features than required.

- Get the latest design document. Form a team with the people that have had worked on most part of the software. Trawl through the source code. Brainstorm among the team and check with the design document and find the chunks which are no longer needed.

Heavy But Under or Un-used Code…

- What to do with the unnecessary code?

- You can either remove the code or replace it with some lighter ones.

- How to verify if the software is still safe to use? In other words, how to verify the sanity of the software, post this operation?

- It is very important to see that the software's business logic functions the same way before and after carrying out the operation.

Heavy But Under or Un-used Code…

- You can follow the steps (this list not complete but are necessary) detailed below for the same.

1. Test the software thoroughly. Developers should be executing all the unit test cases, including the modules that are not impacted. This is to ensure the integrity of the software still remains intact.

2. Make sure there are no memory leaks. Run it through static analysis tools.

Heavy But Under or Un-used Code…

3. Regression: Involve the QA and get all the black box testing done. QA must be maintaining a repository of all the test cases executed since the birth of the software, meaning all the test cases created and executed across all the releases of the software.

4. Soak: There are few bugs which will surface only under certain conditions, say after few hours or days of run or under heavy load.

- To successfully complete this operation we require good amount of time and effort. It is where the software test automation gives us a big helping hand. If you have enough automation built around testing the complete functionality of the software, you could save a lot of time and effort.

Blocking To Non-blocking

VS

Producing responsive software is a challenge. One of the weakest links in the software is the blocking function call

It could be a blocking networking call, file system call, or individual I/O operations that take long time to complete

Blocking To Non-blocking…

- Implementation strategies such as thread-per-blocking request can degrade system performance, due to increased context switching, synchronization and data movement among CPUs.
- With asynchronous operations it is possible to avoid the cost of context switching by minimizing the number of operating system threads and only activating the logical threads of control that have events to process.
- Windows – iocp. Non-windows – select, poll, kqueue.
- Libevent, libuv, etc.

Conclusion

Adopting new standards may not be a panacea for all the software problems. But one can definitely improve the current quality of the software at various levels.

The new standards might provide cleaner, safer, and faster ways to solve your problem. Use caution while adopting the latest standards.

It is relatively difficult to develop applications using asynchronous mechanisms due to the separation in time and space between operation initiation and completion. Debugging asynchronous applications may also be harder due to the inverted flow of control. But the overall throughput of the software will rise sharply.

References

Kohlhoff Christopher, 2013,
http://www.boost.org/doc/libs/1_54_0/doc/html/boost_asio.html

The C++ Standards Committee, 2013,
http://www.open-std.org/JTC1/SC22/WG21/

Wikipedia, 2013,
http://en.wikipedia.org/wiki/Code_refactoring

Wikipedia, 2013,
http://en.wikipedia.org/wiki/Regression_testing

Wikipedia, 2013,
https://en.wikipedia.org/wiki/Test_automation

Note: All the images in this presentation are sourced from google images

Thanks!

Questions?