University of Hertfordshire

7COM1025
Programming for Software Engineers
Lecture 22

Dr. Renato C. de Amorim

THE ONE HUNDRED UNDER FIFTY

REQUIRED READING

This class relates to chapter 8: versioning
Book:

Reddy, M. (2011). API Design for C++. Elsevier.

Pages: 241 - 265

University of Hertfordshire

THE ONE HUNDRED UNDER FIFTY

VERSION NUMBERS

There are indeed many different schemes, most commonly two or three separate integers. Eg: “1.2”, or “2.1.3” (major version, minor version, patch number).

Version information for your API should be accessible from the code. This would allow clients to write code that is conditional to your API's version number.

- It may call a method that only exists in a new version.
- It may work around a bug in an older version.

It may be useful for programmers to have access to this version number at compiler time, so that they can use preprocessor directives (those starting with #).

```
#define API_MAJOR 1
#define API_MINOR 2
#define API_PATCH 0
```

Another solution for the above is to provide a bool HasFeature(string name) method. For run-time information, you could have a .GetVersion() method.

API LIFECYCLE

Maintaining an API is not necessarily the same as maintaining a normal software product.

There is an extra constraint not to break existing clients.

In normal end-user software the change of a method or class does not affect the user-visible features. Such a change in an API may break the code of all existing clients.

An API is a contract, you must make sure you uphold your end of the contract.

We will now see the four general stages of API design

API LIFECYCLE - STAGES

1) Prerelease

Before the initial release an API can progress through a standard software lifecycle

Requirements gathering, planning, design, implementation and testing.

The interface can go through major changes and redesigns during this period.

You may release these early versions to users for feedback and suggestions. If you do so, the version should be 0.x

This way it is clear to the users that the API may change radically before version 1.0 is delivered.

University of Hertfordshire

THE ONE HUNDRED UNDER FIFTY

API LIFECYCLE - STAGES

2) Maintenance

An API can still be modified after release., but in order to maintain backward compatibility these changes must be restricted to:

- Adding new methods.
- Adding new classes.
- Fix bugs in the implementation of existing methods.

You should seek to evolve the API.

Not change it so it becomes incompatible.

API LIFECYCLE - STAGES

3) Completion

At some point the API will reach maturity and no further changes should be made to the interface.

Stability is the most important quality at this point in the life spam. So only bug fixes will be generally considered.

API reviews could still be run at this stage, but if changes are restricted to implementation code and not public headers then they may not be necessary.

Ultimately, the API will reach the point where it is considered to be complete and no further changes will be made.

API LIFECYCLE - STAGES

4) Deprecation

Some APIs eventually reach an end-of-life stage.

They are deprecated and then removed from circulation.

This means the API should not be used for any new development and that existing clients should migrate away from the API.

This can happen if the API no longer serves a useful purpose or if a newer, incompatible API has been developed to take its place.

COMPATIBILITY

Backward compatibility: An API that provides the same functionality as a previous version of the API. It can fully take the place of a previous version without requiring the user to make changes.

There are three different types of API backward compatibility, including:

Functional compatibility

This is concerned with the run-time behaviour of the implementation. An API is functionally compatible if it behaves exactly the same as its previous version.

Source compatibility

It is a looser definition of backward compatibility. It basically states that users can recompile their programs using the new version of the API without making changes to their code. It says nothing about the behaviour of the resulting program.

Binary compatibility

Implies clients only need to re-link their programs with a newer version of a static library or simply drop a new shared library into the install directory of their end-user application.

This contrasts with source compatibility in which users must recompile their programs.

University of Hertfordshire
THE ONE HUNDRED UNDER FIFTY

API REVIEW

Backwards compatibility doesn't just happen. It requires effort to ensure that no new changes to an API have silently broken existing code.

There are two models for API review:

Pre-release API reviews
- Hold a single pre-release meeting to review all changes since the previous release.
- Should focus on the interface being delivered, not the specifics of the code.
- You should be attended (at least!) by:
(i)Product owner (person with overall responsibility for product planning and clients needs) (ii) Technical lead; (iii) Documentation lead.

Pre-commit API reviews
You don't need to wait until right before release to catch problems!
- It requires a formal change request process
Engineers wanting to make changes to the public API must formally request permission to do so.
- Particularly useful in open source projects as patches can be submitted from many engineers, with different backgrounds and skills.

(You can also do both)

API REVIEW - PURPOSE

Maintain backward compatibility
This is the primary purpose of API reviews.

Maintain design consistency
Changes that do not fit into the API design should be caught and recast. Otherwise you will end up an API with no cohesion or consistency.

Change control
Manages change (including why changes were made!). You should be able to reject changes that are inappropriate.

Allow future evolution
A single change can be implemented in several ways. You should accept only those that are “future-proof”.

Revisit solutions
After release, you may receive feedback from your clients that will allow you to come up with better solutions.

University of Hertfordshire
THE ONE HUNDRED UNDER FIFTY