# COSC345 Week 24

# Reverse and Re-Engineering

# 23 September 2014

Richard A. O'Keefe

# Objectives

Understand re-engineering as a maintenance activity

Understand what reverse engineering is

Appreciate ethical issues of reverse engineering.

Appreciate legal issues of reverse engineering

# References:

~ok/COSC345/Copyright Act 1994.pdf

http://en.wikipedia.org/wiki/Reverse_engineering (Wikipedia)

http://www.acm.uiuc.edu/sigmil/RevEng/ (a book someone was writing, full of Windows characters — *sigh*)

Chapter 34 in the 5th edition of Sommerville

Chapter 28 in the 6th edition of Sommerville (+ 26)

Largely missing from the 7th edition, but see chapter 21.

# The NZ Copyright Act 1994

Making a backup copy of software is OK (s80)

Decompilation is OK **if** necessary to make a separate program to work with the original, **not** a clone (s80A)

Making a copy to fix it is OK **if** "a properly functioning and error-free copy of the program is not available ... within a reasonable time at an ordinary commercial price'' (s80B)

"observing, studying, or testing the functioning of [a] program'' is OK **if** you're doing something else OK at the time (s80C).

Contractual terms trying to stop you doing something legal have no effect (s80D).

# Copyright Act, again

You must be a **lawful** user of the program.

“Studying’’ a program is only allowed “while performing the acts of loading, displaying, running, transmitting, or storing the program’’ that you are otherwise entitled to do.

Decompilation to develop workalikes is forbidden.

Decompilation is forbidden if you could learn what you need some other way.

# Laws and ethics

The law doesn't always allow what is right.

The law sometimes allows what is wrong.

Different countries have different laws.

Laws change.

ACM code 3.13: Be careful to use only accurate data derived by ethical and lawful means, and use it only in ways properly authorized.

# Forward Engineering

1. Gather requirements
2. Produce specification
3. Write documentation
4. Write test cases
5. Write code
6. Test
7. Deliver

# Re-engineering

Each stage builds on products *and documents* of previous stage

Re-engineering starts from earlier stage and works forward again

Sometimes design documents are missing

One kind of improvement restructures code so that changes will be well localised and testable

Reverse engineering *reconstructs missing design documents* or cleans up existing ones so future changes have a solid base.

# Reverse Engineering

Travel backwards in "forward engineering time"

From object code, reconstruct assembly code
dis does this for .o files
otool does it for Mac OS X
javap does this for .class files.

From object or assembly, reconstruct source code
Java decompiler example (available for download).

# Reverse Engineering (2)

From source code, reconstruct test cases

From source code, reconstruct specification
Minimally, which arguments may not be NULL pointers?
Check reconstruction by testing.

Reconstruct requirements
No tools here, have to guess what humans were thinking.

# Beware!

Reverse engineering someone else's code is *illegal* in many parts of the world.

Only disassemble or decompile code that you have a legal right to.

# Source code translation I

There are dialect converters (for example, Classic C to C89, HTML to XHTML) and language converters (f2c, p2c)

Some are just for compiling (Stalin), some generate readable code (p2c).

Commercial translation services exist.

**Conversion always requires human judgement**

Sommerville says "in-line comments are lost". He is wrong.

# Source code translation II

With a foreign function interface, you may be able to convert one module or function at a time.

Dialect and language conversion doesn't help with libraries.

Beware of calling sequence changes (f77 to f95).

Old-to-new much easier than new-to-old.

# Why do we do reverse engineering?

Because we have to.

The purpose of maintenance is to preserve the value of a program to customers through time.

If it costs too much to make changes, we have to clean up the cost somehow.

This typically means moving back and then re-engineering.

# An example of dialect conversion

Old AWK had no `function` or `delete`

It was also missing some functions.

Therefore we need to replace newly reserved words.

```
sub(/N*(function|delete|...)/, "N&", id)
```

— but need at least lexical analysis to get this right (not inside comments, strings, or regular expressions)