CS446
Cloning and Software Design

Wei Wang

Materials adopted from:

Michael Godfrey’s

“We all like sheep”

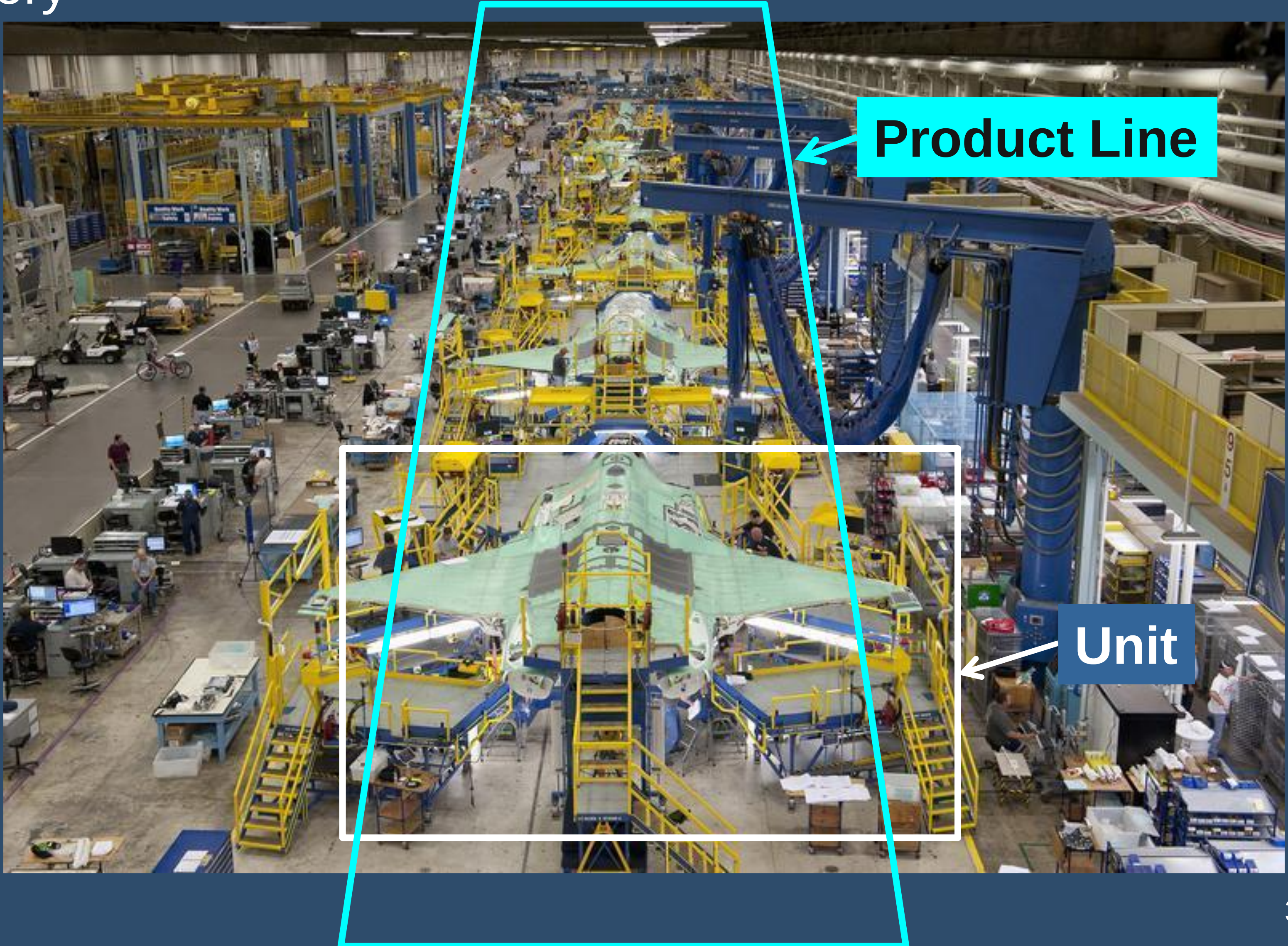
Deliverable #4

- the first thing you would give a new employee to get them up to speed on the low-level structure of your system

- **Rationale** must be provided documenting why you selected your design

Design patterns

Factory



Which design pattern is applicable here?

- Show status of each level *uniformly*
- function: countOperaters() – return the number of works (of a unit, of a line, of a factory)

PART ONE OF TWO

Clones and clone detection

Overview

- Some motivating examples
- Kinds of clones, by structure
- Approaches and tools for clone detection
- The software engineering dimension:
  - Just how bad are clones? How do we know?
- A taxonomy of clones, by design intent

Some examples of code clones

Consider this code…

```
const char *err = ap_check_cmd_context(cmd, GLOBAL_ONLY);
if (err != NULL) {
    return err;
}
ap_threads_per_child = atoi(arg);
if (ap_threads_per_child > thread_limit) {
    ap_log_error(APLOG_MARK, APLOG_STARTUP, 0, NULL,
                 "WARNING: ThreadsPerChild of %d exceeds ThreadLimit "
                 "value of %d", ap_threads_per_child,
                 thread_limit);
    ….
    ap_threads_per_child = thread_limit;
}
else if (ap_threads_per_child < 1) {
    ap_log_error(APLOG_MARK, APLOG_STARTUP, 0, NULL,
                 "WARNING: Require ThreadsPerChild > 0, setting to 1");
    ap_threads_per_child = 1;
}
return NULL;
```

and this code …

```
    const char *err = ap_check_cmd_context(cmd, GLOBAL_ONLY);
    if (err != NULL) {
        return err;
    }
    ap_threads_per_child = atoi(arg);
    if (ap_threads_per_child > thread_limit) {
        ap_log_error(APLOG_MARK, APLOG_STARTUP, 0, NULL,
                     "WARNING: ThreadsPerChild of %d exceeds ThreadLimit "
                     "value of %d threads,", ap_threads_per_child,
                     thread_limit);
        ….
        ap_threads_per_child = thread_limit;
    }
    else if (ap_threads_per_child < 1) {
ap_log_error(APLOG_MARK, APLOG_STARTUP, 0, NULL,
                     "WARNING: Require ThreadsPerChild > 0, setting to 1");
ap_threads_per_child = 1;
    }
    return NULL;
```

… or these two functions

```
static GnmValue *
gnumeric_oct2bin (FunctionEvalInfo *ei, GnmValue const * const *argv)
{
    return val_to_base (ei, argv[0], argv[1],
            8, 2,
            0, GNM_const(7777777777.0),
            V2B_STRINGS_MAXLEN | V2B_STRINGS_BLANK_ZERO);
}

static GnmValue *
gnumeric_hex2bin (FunctionEvalInfo *ei, GnmValue const * const *argv)
{
    return val_to_base (ei, argv[0], argv[1],
            16, 2,
            0, GNM_const(9999999999.0),
            V2B_STRINGS_MAXLEN | V2B_STRINGS_BLANK_ZERO);
}
```

Or this …

```
static PyObject *
py_new_RangeRef_object (const GnmRangeRef *range_ref){
    py_RangeRef_object *self;
    self = PyObject_NEW py_RangeRef_object,
            &py_RangeRef_object_type);
    if (self == NULL) {
            return NULL;
    }
    self->range_ref = *range_ref;
    return (PyObject *) self;
}
```

… and this

```
static PyObject *
py_new_Range_object (GnmRange const *range) {
    py_Range_object *self;
    self = PyObject_NEW (py_Range_object,
            &py_Range_object_type);
    if (self == NULL) {
            return NULL;
    }
    self->range = *range;
    return (PyObject *) self;
}
```

An overview of clone detection

What’s a clone?

“Software clones are segments of code that are similar according to some definition of similarity.”

– Ira Baxter, 2002

- No universally agreed upon definition
- Often use “what my tool found” as ground truth
  - Algorithms, thresholds may vary greatly
  - Could hand examine subset of results to guess false positive rate
  - False negatives? … and no ground truth from experts typically.
- Hard to compare results!

Bellon's taxonomy

Type 1	Program text (token stream) identical … but white space / comments may differ
Type 2	… and literals + identifiers may be different
Type 3	… and gaps allowed (can add/delete sections)
Type 4	Two code segments have same semantics (Undecidable in general, not sought often)

–There are other kinds of "clones" that don't fit well here

–Note that type 1, 2, and 4 clones form equivalence classes, but type 3 clones do not

Bellon’s taxonomy

- Type 1 clones are fairly easy to detect
  - Tokenize the source code, remove comments
  - Simple approach:

    ```
    % tokenize file1.c > f1.c
    % tokenize file2.c > f2.c
    % diff -w f1.c f2.c
    ```

  - Scalable approach:
    - Progressively build a suffix tree / array to store *all* known partial sequences of tokens

Bellon’s taxonomy

- Type 2 clones are almost as easy
  - Extra step in tokenization:
    - All identifiers mapped to special token <ID>
    - All explicit string values mapped to <STRING>
    - All explicit numerical values mapped to <NUM>

Bellon’s taxonomy

- Type 3 clones
  - Look for type 2 clones, but allow “gaps” up to some threshold of lines/tokens
  - Notes:
    - Given a big enough threshold, any two pieces of code are type 3 clones!
    - *“is-a-type-3-clone-of”* is not transitive

Bellon’s taxonomy

- Type 4 (semantically identical) clones
  - *“Does P1 have same semantics as P2”* is undecidable in the general case
  - Typically not done, no general purpose detector exists
    - Type 4 category is included for sake of completeness
  - But if we are interested, we can make guesses using various tricks

    e.g., common test suites, dynamic traces

Spot the clone type!

```
const char *err = ap_check_cmd_context(cmd, GLOBAL_ONLY);
if (err != NULL) {
    return err;
}
ap_threads_per_child = atoi(arg);
if (ap_threads_per_child > thread_limit) {
    ap_log_error(APLOG_MARK, APLOG_STARTUP, 0, NULL,
                 "WARNING: ThreadsPerChild of %d exceeds ThreadLimit "
                 "value of %d", ap_threads_per_child,
                 thread_limit);
    ....
    ap_threads_per_child = thread_limit;
}
else if (ap_threads_per_child < 1) {
    ap_log_error(APLOG_MARK, APLOG_STARTUP, 0, NULL,
                 "WARNING: Require ThreadsPerChild > 0, setting to 1");
    ap_threads_per_child = 1;
}
return NULL;
```

Spot the clone type!

```
    const char *err = ap_check_cmd_context(cmd, GLOBAL_ONLY);
    if (err != NULL) {
        return err;
    }
    ap_threads_per_child = atoi(arg);
    if (ap_threads_per_child > thread_limit) {
        ap_log_error(APLOG_MARK, APLOG_STARTUP, 0, NULL,
                     "WARNING: ThreadsPerChild of %d exceeds ThreadLimit "
                     "value of %d threads,", ap_threads_per_child,
                     thread_limit);
        ....
        ap_threads_per_child = thread_limit;
    }
    else if (ap_threads_per_child < 1) {
ap_log_error(APLOG_MARK, APLOG_STARTUP, 0, NULL,
                     "WARNING: Require ThreadsPerChild > 0, setting to 1");
ap_threads_per_child = 1;
    }
    return NULL;
```

string constant different

white space different

Type 1 clones

```
const char *err = ap_check_cmd_context(cmd, GLOBAL_ONLY);
if (err != NULL) {
    return err;
}
ap_threads_per_child = atoi(arg);
if (ap_threads_per_child > thread_limit) {
    ap_log_error(APLOG_MARK, APLOG_STARTUP, 0, NULL,
                 "WARNING: ThreadsPerChild of %d exceeds ThreadLimit "
                 "value of %d threads,", ap_threads_per_child,
                 thread_limit);
    ….
    ap_threads_per_child = thread_limit;
}
else if (ap_threads_per_child < 1) {
ap_log_error(APLOG_MARK, APLOG_STARTUP, 0, NULL,
                 "WARNING: Require ThreadsPerChild > 0, setting to 1");
ap_threads_per_child = 1;
}
return NULL;
```

Type 2 clones

```
static GnmValue *
gnumeric_oct2bin (FunctionEvalInfo *ei, GnmValue const * const *argv)
{
    return val_to_base (ei, argv[0], argv[1],
            8, 2,
            0, GNM_const(7777777777.0),
            V2B_STRINGS_MAXLEN | V2B_STRINGS_BLANK_ZERO);
}
```

numerical constant different

identifier different

```
static GnmValue *
gnumeric_hex2bin (FunctionEvalInfo *ei, GnmValue const * const *argv)
{
    return val_to_base (ei, argv[0], argv[1],
            16, 2,
            0, GNM_const(9999999999.0),
            V2B_STRINGS_MAXLEN | V2B_STRINGS_BLANK_ZERO);
}
```

Type 3 clone

```
static PyObject *
py_new_RangeRef_object (const GnmRangeRef *range_ref){
    py_RangeRef_object *self;
    self = PyObject_NEW py_RangeRef_object,
            &py_RangeRef_object_type);
    if (self == NULL) {
            return NULL;
    }
    self->range_ref = *range_ref;
    return (PyObject *) self;
}
```

Type 3 clone

```
static PyObject *
py_new_Range_object (GnmRange const *range) {
    py_Range_object *self;
    self = PyObject_NEW (py_Range_object,
     &py_Range_object_type);
    if (self == NULL) {
           return NULL;
    }
    self->range = *range;
    return (PyObject *) self;
}
```

Type 3 clone

```
static PyObject *
py_new_Range_object (GnmRange const *range) {
    py_Range_object *self;
    self = PyObject_NEW (py_Range_object,
     &py_Range_object_type);
    if (self == NULL) {
            return NULL;
    }
    self->range = *range;
    return (PyObject *) self;
}
```

A more common type 3 clone

```
static PyObject *
py_new_Range_object (GnmRange const *range) {
    if (!DEBUG) {
        py_Range_object *self;
        self = PyObject_NEW (py_Range_object,
        &py_Range_object_type);
        if (self == NULL) {
            return NULL;
        }
    } else {
        return NULL;
    }
    self->range = *range;
    return (PyObject *) self;
}
```

Measuring detection effectiveness

- We borrow these terms from IR:
  - Precision: How many of the answers you find are real?
  - Recall: How many of the real answers do you find?

  … but we usually lack "ground truth"

- False positives and filtering:
  - Most detection tools are highly tunable
  - Often set tool for "more hits", then perform customized filtering to remove common false positives

More of the same, only different

- Problems related to software clone detection
  - Plagiarism detection, IP theft
  - DNA sequence analysis
  - Data compression
  - SPAM analysis, malware detection

Code clone detection methods

Structural

- Sequences
  - Strings
  - Tokens
- Graphs
  - ASTs
  - PDGs

Time, complexity, prog lang dependence

Others / hybrids

- Metrics
- Lightweight semantics
- Normalization
- Analyzing assembler

See also Roy & Cordy’s tech report

Sequence-based approaches

- Atomic unit of comparison?
  - … could be char string (LOC), token, assembler instruction, SHA1 hash code, …
  - Comparison between atoms could be exact or approximate

- To find clones of sequences of atoms:
  - Longest exact sequential match
    - Use suffix tree/array
  - Compute Levenshtein edit distance
  - Use n-grams and a moving window to allow for gaps [Baker, Johnson, MOSS]

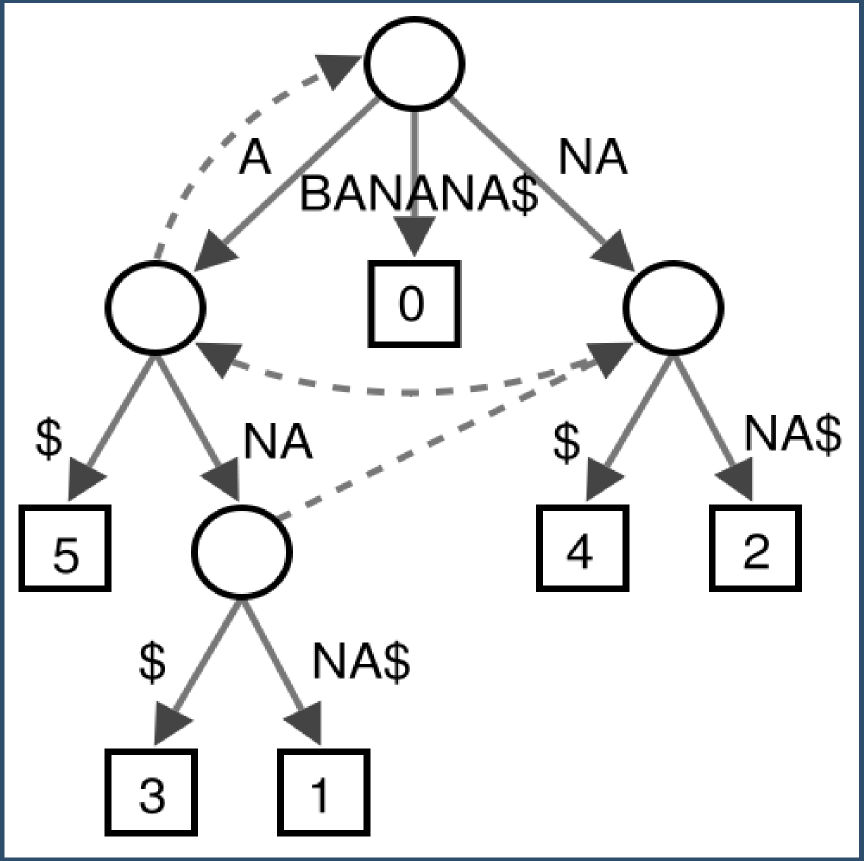
String-based clone detection

- Model:
  - Programs are *sequences* of *character strings* (i.e., LOC*)*
- Simple to implement
  - Look for exact textual matches of LOC
  - Mostly independent of prog lang
- Typical:
  - Pre-process to remove white space + comments
  - Use hashes to speed comparisons of strings
- Variants:
  - Allow gaps in sequences
  - Use Levenstein edit distance for near-misses

Token-based clone detection

- Model:
  - Programs are *sequences* of *tokens*
- Low dependence on program lang!
- Typical:
  - Use suffix trees/arrays to detect results

Suffix tree / array



[Diagram from Wikipedia]

- For each token stream ("string"), build a tree that represents all possible suffixes
  - Compare each new string to the set of existing trees
  - Comparisons are fast, but uses a lot of space

- Suffix array is a *sorted* list of all suffixes:
  1. a
  2. ana
  3. anana
  4. banana
  5. na
  6. nana

Token-based clone detection

- Variants
  - Naïve generalized token stream
    - Map ids to <ID>, string values to <STRING>, etc
    - So x = x + 1 becomes <id> = <id> + <NUM>, and will match these:

```
y = y + 1
a = b + 5
x = y + 3.14
```

but not these:

```
x = 1 + x
x++
x = x + y
```

PART TWO OF TWO

Clone analysis + sw eng concerns

- … to understand how a “thing” evolves, you must understand:
  - the thing and its programming,
  - its environment, and
  - how they can influence each other.

- … and “hard coding” can still lead to flexible, interesting run-time behaviors

So …

- … to understand software cloning, it’s not enough to study software clones in isolation. We have to study:
  - the systems from which they came as a whole
  - the (presumed) reasons for cloning
  - the short- and long-term effects of cloning on its (technical and social) environment
  - the longer term evolution of clones within the system

Cloning and Design

Just how bad is software cloning?

- Most early research concentrated on *detection* algorithms
- Recent focus has shifted to clone *analysis*
  - *What techniques are effective to study large systems?*
  - *What kinds of clones are there? What properties do they have?*
  - *Does cloning really hurt the design in the long run?*
- Case studies suggest that cloning is common practice in industrial software
  - 5-15% is common; up to 50% in some systems
  - … but it’s unclear how “bad” this is

Quotes on source code cloning

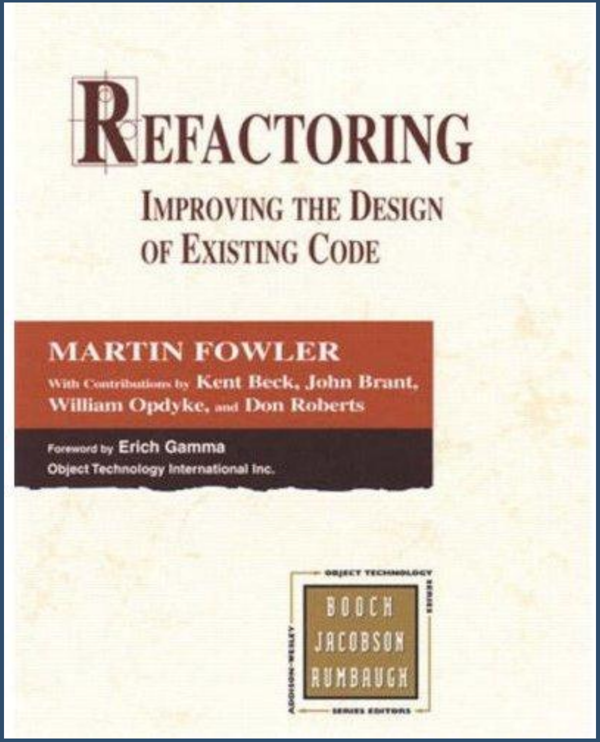
“Q: You want to stop developers cutting and pasting code? Why?

A: This is Software Engineering 101, for heaven's sake!! ”

– ACNP Software

`http://www.anticutandpaste.com/`

Quotes on source code cloning



REFACTORING

IMPROVING THE DESIGN OF EXISTING CODE

MARTIN FOWLER

With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts

Foreword by Erich Gamma

Object Technology International Inc.

OBJECT TECHNOLOGY SERIES

BOOCH JACOBSON RUMBAUGH

ADDISON-WESLEY

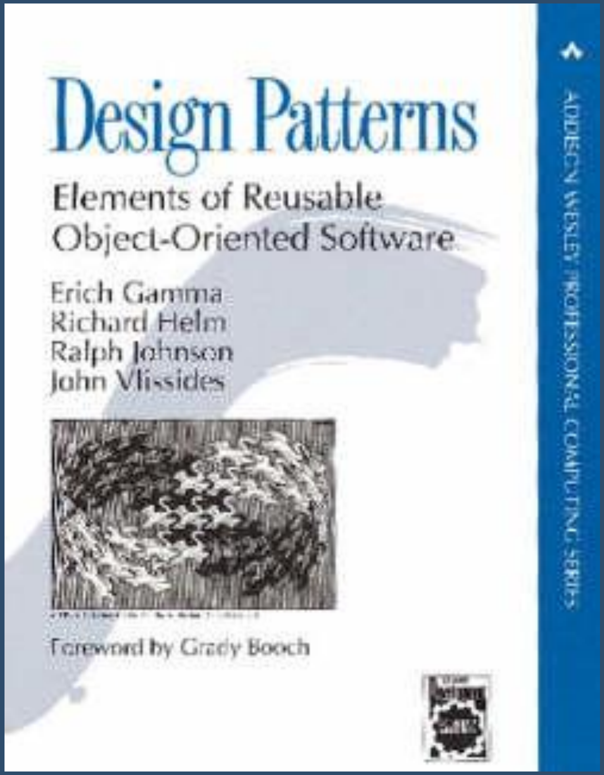
SERIES EDITORS

"Number one in the stink parade is duplicated code. If you see the same code structure in more than one place, <u>you can be sure</u> that your program will be better if you find a way to unify them."

– "Bad Smells"

[Beck/Fowler in *Refactoring*]

Quotes on source code cloning


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON WESLEY PROFESSIONAL COMPUTING SERIES

"So, copy-and-paste is not necessarily bad in the short run, if you are copying good code.

<u>But it is always bad in the long run.</u>"

– Ralph Johnson, 2004 blog [3]

Myth

Code cloning

is always bad

in the long run

Why cloning is supposed to be bad

- Duplicated code leads to bloat
  - Hard to understand, less “essential”
  - Inconsistent maintenance risk
    - Will all bug instances be fixed?

- Duplication is a sign of inexperienced developers
  - Copy/paste is often “easy”, JIT comprehension
  - Cruft will accumulate as developers fear changing working code

- Duplication is a sign of poor design / extensibility
  - Need to keep doing same kinds of things, but there’s no easy way to automate it

What you are supposed to do instead

1. Identify commonalities across code base

2. Refactor duplicate functionality to one place in the code
   - Functions with parameters
   - OO: Base class encapsulates commonalities, derived classes specialize peculiarities
   - Generics / templates for classes / functions / (aspects?)

Cloning is bad?

- Whoops!
- How did *that* happen?
- Have we been led astray?

Formula, repetition, duplication

- In the arts and life, we seek to explore the new through careful venturing away from the familiar
  - Watch a toddler exploring his/her world
- The familiar can be
  - a narrative structure (e.g., a fairy tale, a knock-knock joke),
  - a chorus (new to us, but repeated),
  - a theme (e.g., sacrifice for love), …
- Humans also seem to find comfort in ritual
  - We seek refuge in the familiar when the external world seems unpredictable and frightening

Formula, repetition, duplication

- But this is *engineering*!
  - And we have no need of ritual in a utilitarian design!
- Ritual no, but repetition yes!
  - In traditional engineering, we scale up by repeated instantiations of design elements

Formula, repetition, duplication

- But this is *software* engineering!
  - We don't need duplication in a *software* design, right?

  - Server farms
  - Map-reduce
  - Virtual machines
  - …

Formula, repetition, duplication

- Replication of trusted design elements works in software too!
  - We do need familiarity as a learning tool
  - And we can — and should! — employ duplication within a disciplined engineering process

Cloning as software design practice: Bronze an exemplar!

- The *Prototype* design pattern [GoF]
  - Create a copy of an existing complex object, often using a method called `clone()`
- The *Self* programming language [Ungar at al.]
  - Supports evolutionary designs better than trad. OO langs
  - Helps with the fragile base class problem

Cloning as software design practice

- The Rule of Three (eXtreme Programming)
  - Premature abstraction is the root of much evil!
  - Design the simplest thing that could possibly work.
- Boiler-plating is key to industrial-strength COBOL development [Cordy 03]
  - Reliable designs and working systems are golden!

Clone genealogies [Kim et al. 2005]

Q: How and why do clones change over time?

- Looked at two ~20 KLOC Java programs (CAROL+ dnsjava)

- Findings:
  – Some clones are “volatile”, are introduced as a means-to-an-end but get refactored and disappear quickly
  – Some clones are more long lived, often hard to refactor due to programming language limitations (e.g., lack of generics, aspects)
  – Many clones are maintained in parallel, but some are not
  – It’s common for clones to change in different ways over time
  – Some clones represent fundamental design decisions that can’t be refactored easily
  – Naïve aggressive refactoring is not the answer!

Consistency of change [Krinke 07]

Q: Is inconsistent maintenance of clones really a problem?

- Studied five large open source systems over time:
  - ArgoUML, CAROL, jdt.core, emacs, FileZilla

- Findings:
  - Clone groups are changed consistently about 50% of the time
  - Clone groups that are not changed consistently rarely become so later
    - So probably they were intended to diverge

Clones: What is that smell? [Rahman et al. 10]

Q: Is cloned code buggier than non-cloned code?

- Examined several large open source projects:
  - Evolution, Apache, Gimp, Nautilus
- Findings:
  - Most bugs have very little to do with clones,
    - Cloned code is typically *less* buggy than non-cloned code
  - Larger clone groups *don't* have more bugs than smaller groups
    - Making more copies of code *doesn't* introduce more defects,
    - Larger clone groups (# of members) have *lower* bug density per line than smaller clone groups.

Summary

- Cloning is common in industrial code!
- While cloning is sometimes due to laziness and causes problems, often it’s used as a principled design tool
  - So refactoring may be a *bad* idea
  - Need to consider context + design rationale before refactoring
- Empirical evidence from open source systems suggests:
  - There are many reasons to clone
  - Cloned code is often maintained appropriately
  - Principled cloning doesn’t seem to cause undue problems later on