axivion

stopping software erosion

Clone Management in Practice

IWSC 2016 Osaka

Stefan Bellon
Axivion GmbH

Who am I?

- 1997-2002: Degree in computer science from University of Stuttgart, Germany
- Diploma Thesis „Vergleich von Techniken zur Erkennung duplizierten Quellcodes“ (engl. Comparison of techniques for detecting duplicated source code)
- 2003-2005: Researcher at Programming Languages and Compiler Group of the Institute of Software Technology at University of Stuttgart, Germany
- Research Topics: Architecture Checking and Clone Detection
- 2006-2016: Co-Founder and managing director of Axivion GmbH

Introduction: My Diploma Thesis

„Vergleich von Techniken zur Erkennung duplizierten Quellcodes“

- 6 researchers, 6 tools:

Researcher	Tool	Technique
Brenda S. Baker	Dup	Suffix tree, token based
Ira D. Baxer	CloneDR	Subtree matching in the AST
Toshihiro Kamiya	CCFinder	Input transformations, token based
Jens Krinke	Duplix	Program Dependence Graph
Ettore Merlo	CLAN	Function metrics and token based
Matthias Rieger	Duploc	Pattern matching, token based

Introduction: My Diploma Thesis

„Vergleich von Techniken zur Erkennung duplizierten Quellcodes“

- 4 differently sized C and 4 differently sized Java systems
- Human oracle to build reference corpus
- Very (!) short summary of results:

	Baker	Baxter	Kamiya	Krinke	Merlo	Rieger
Recall	+	-	+	-	-	+
Precision	-	+	-	-	+	-

- Benchmark details: http://www.bauhaus-stuttgart.de/clones/
- Result details (talk on 1st IWSC 2002 Montreal): http://www.sbellon.de/archives/clonesmontreal.pdf.gz

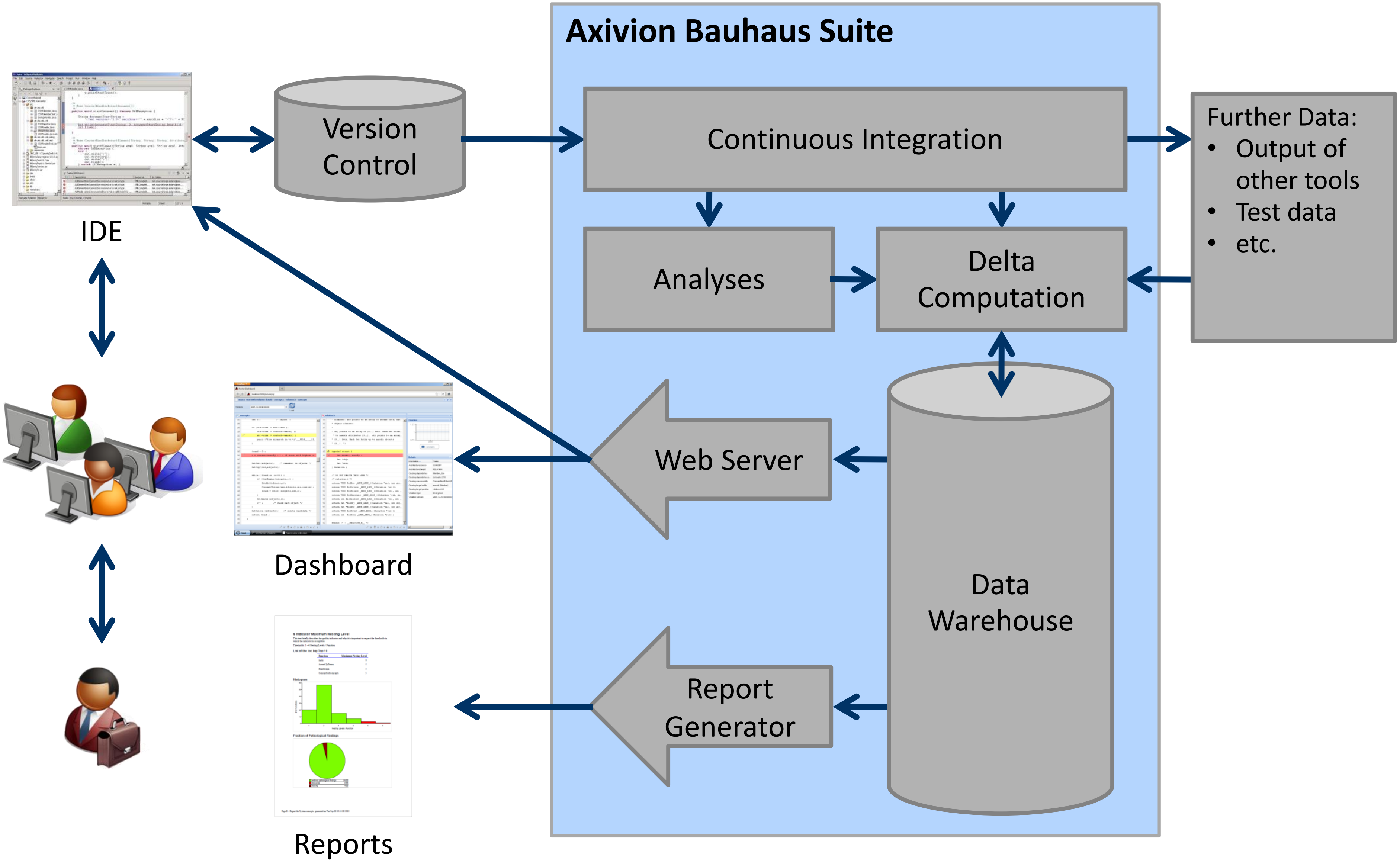
Motivation

- 1st IWSC 2002 Montreal
  - How good are tools at detecting software clones?
  - Pros and Cons of certain techniques
- 10th IWSC 2016 Osaka
  - Detection of software clones is „good enough“ in general
  - Usefulness of clones for the user is undecidable for a tool
  - User interface to the developer needs more attention now
  - New challenge: Managing software clones
    - What to do with the clone?
    - Who is responsible?

Overview

- Introduction of Axivion Bauhaus Suite wrt. clone management
- Clone Management to find errors and/or maintenance burden
- Clone Management in product lines
- Clone Detection to check for license compliance
- Demonstration of Axivion Bauhaus Suite

Introduction of Axivion Bauhaus Suite wrt. clone management



Axivion Bauhaus Suite

IDE

Version Control

Continuous Integration

Analyses

Delta Computation

Further Data:
- Output of other tools
- Test data
- etc.

Web Server

Data Warehouse

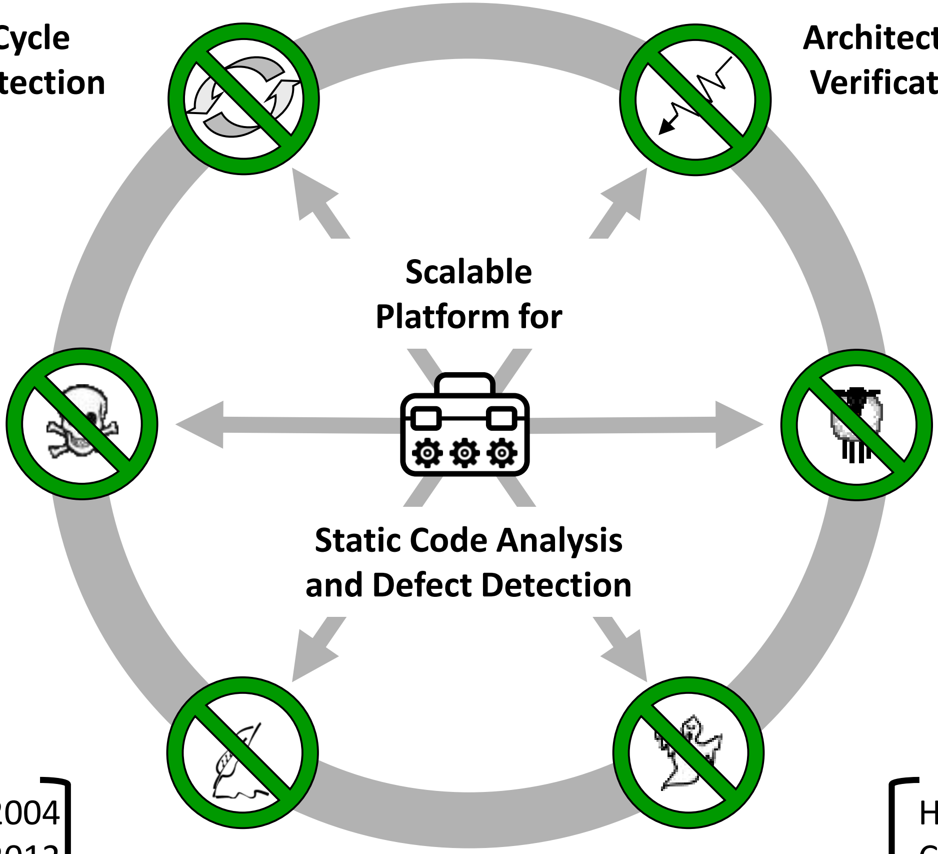
Report Generator

Dashboard

Reports

Introduction of Axivion Bauhaus Suite wrt. clone management

axivion
stopping software erosion



Scalable Platform for Static Code Analysis and Defect Detection

Cycle Detection

- Calls
- Module Dependencies
- Includes/Imports
- Customer specific
- ...

Architecture Verification

- Integrated modeler and XMI import interface

Dead Code Analysis

- Reachability Analysis
- Libraries

Clone Detection and Clone Management

- Type I (1:1)
- Type II (Changes)
- Type III (Changes + Add/Delete)

Stylechecks

- Misra C:2004
- Misra C:2012
- Misra C++:2008
- Best Practise
- Customer specific

Metrics Monitoring

- HIS
- Chidamber & Kemerer OOD
- Best Practise
- Customer specific
- ...

Introduction of Axivion Bauhaus Suite wrt. clone management

Clone detection tools in the Axivion Bauhaus Suite

- Token-based detection using (P-)suffix trees (for C, C++, C#, Java, and Ada)
- Syntax-tree-based detection using subtree hashing (for C, C++, and C#)

All examples and numbers later on are gathered using the syntax-tree-based clone detection with a minimum clone fragment length of 30 lines and a minimum clone fragment weight of 30 nodes in the syntax tree.

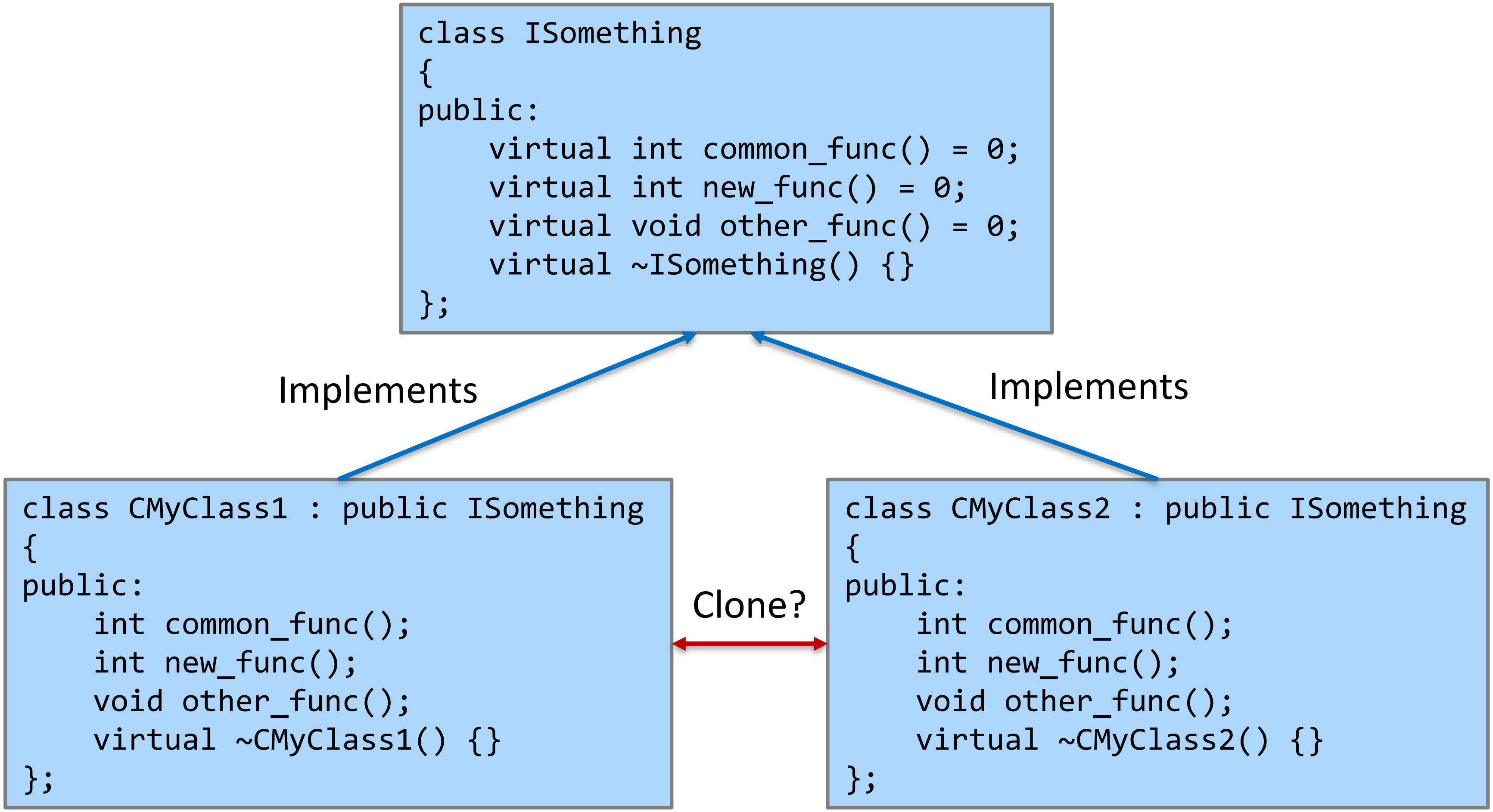
Clone Management to find errors and/or maintenance burden

Technical issues for users

- Dealing with technically correct type-2 clones, however, completely uninteresting ones for the user
- Dealing with type-3 clones that even indicate errors
- Responsibility of a clone
- Refactoring of clones
- Visualization of clones

Clone Management to find errors and/or maintenance burden

Dealing with technically correct type-2 clones, however, completely uninteresting ones for the user



```
class ISomething
{
public:
    virtual int common_func() = 0;
    virtual int new_func() = 0;
    virtual void other_func() = 0;
    virtual ~ISomething() {}
};
```

```
class CMyClass1 : public ISomething
{
public:
    int common_func();
    int new_func();
    void other_func();
    virtual ~CMyClass1() {}
};
```

```
class CMyClass2 : public ISomething
{
public:
    int common_func();
    int new_func();
    void other_func();
    virtual ~CMyClass2() {}
};
```

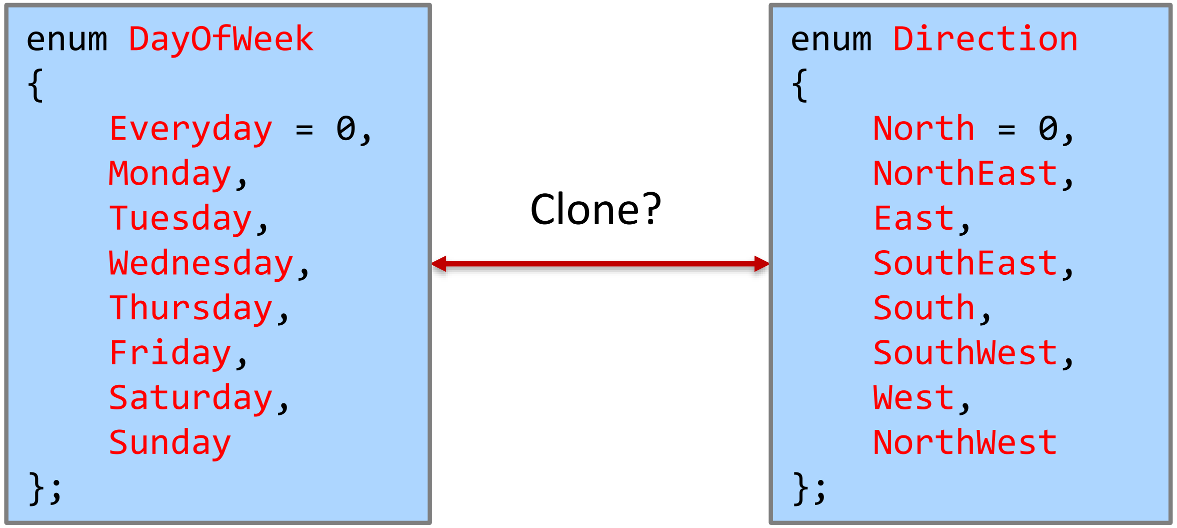
Clone Management to find errors and/or maintenance burden

Dealing with technically correct type-2 clones,
however, completely uninteresting ones for the user

- Diploma thesis: „No difference regarding cloning and clone detection between programming languages [C and Java]“
- Revised experience today: C++ is different than Java (and C)
  - Separate class definition and method implementations (.hpp, .cpp)
  - Class definitions of common base classes are type-2 clones, but that is due to the language, you cannot avoid it!
  - Often not regarded as helpful to get them reported as the compiler itself enforces consistent changes to child classes if base class changes
  - Inconsistent changes may still occur → valuable in some cases anyway!
  - (Clones in method implementations are interesting → pull up refactoring)

Clone Management to find errors and/or maintenance burden

Dealing with technically correct type-2 clones, however, completely uninteresting ones for the user



```
enum DayOfWeek
{
    Everyday = 0,
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday
};
```

Clone?

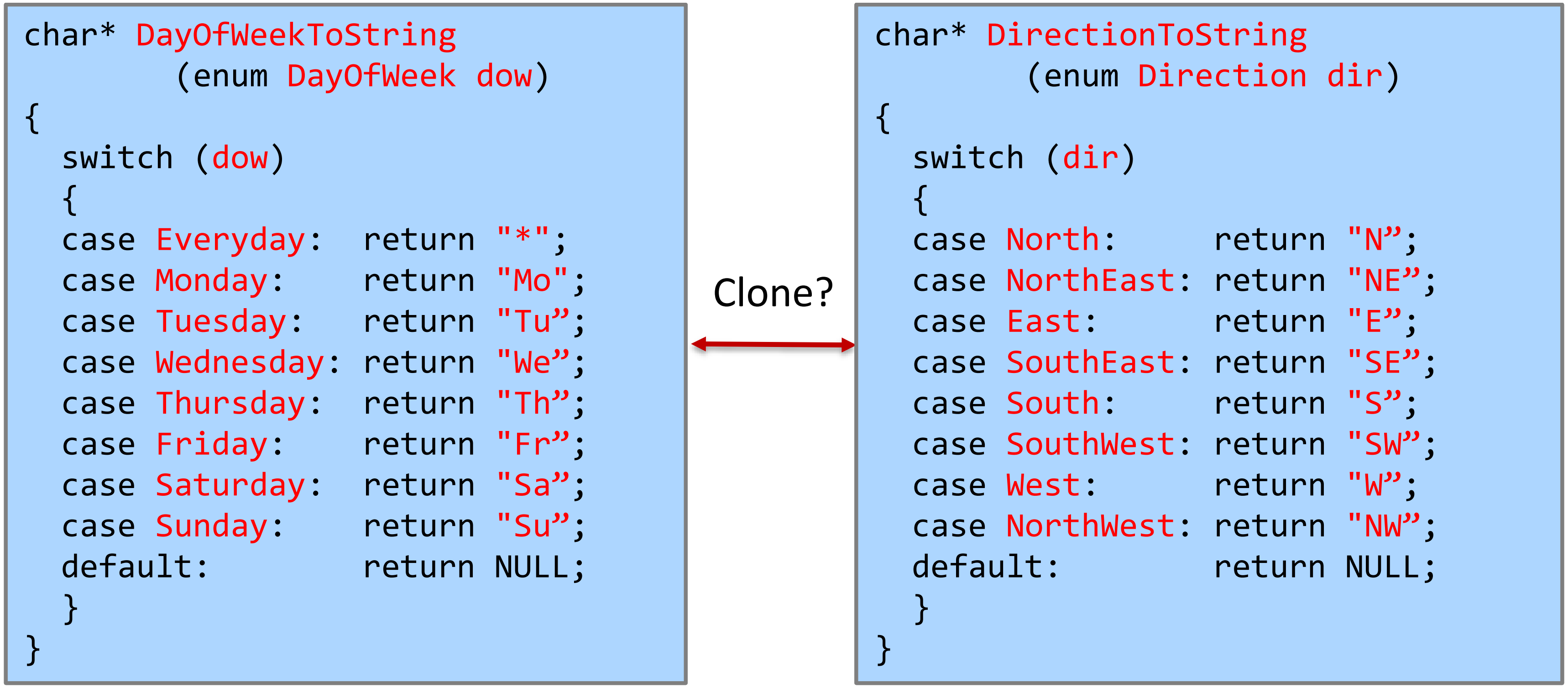
```
enum Direction
{
    North = 0,
    NorthEast,
    East,
    SouthEast,
    South,
    SouthWest,
    West,
    NorthWest
};
```

- Technically type 2, but completely meaningless to the user!

Clone Management to find errors and/or maintenance burden

axivion
stopping software erosion

Dealing with technically correct type-2 clones, however, completely uninteresting ones for the user



```
char* DayOfWeekToString
        (enum DayOfWeek dow)
{
  switch (dow)
  {
  case Everyday:  return "*";
  case Monday:    return "Mo";
  case Tuesday:   return "Tu”;
  case Wednesday: return "We”;
  case Thursday:  return "Th”;
  case Friday:    return "Fr”;
  case Saturday:  return "Sa”;
  case Sunday:    return "Su”;
  default:        return NULL;
  }
}
```

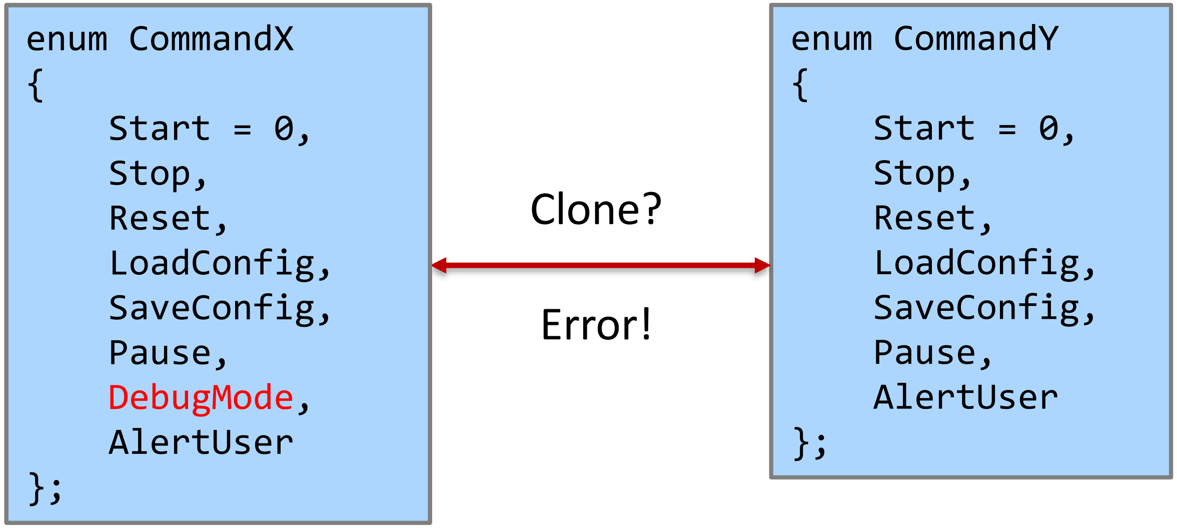
Clone?

```
char* DirectionToString
        (enum Direction dir)
{
  switch (dir)
  {
  case North:     return "N”;
  case NorthEast: return "NE”;
  case East:      return "E”;
  case SouthEast: return "SE”;
  case South:     return "S”;
  case SouthWest: return "SW”;
  case West:      return "W”;
  case NorthWest: return "NW”;
  default:        return NULL;
  }
}
```

- Technically type 2, but completely meaningless to the user!

Clone Management to find errors and/or maintenance burden

Dealing with type-3 clones that even indicate errors



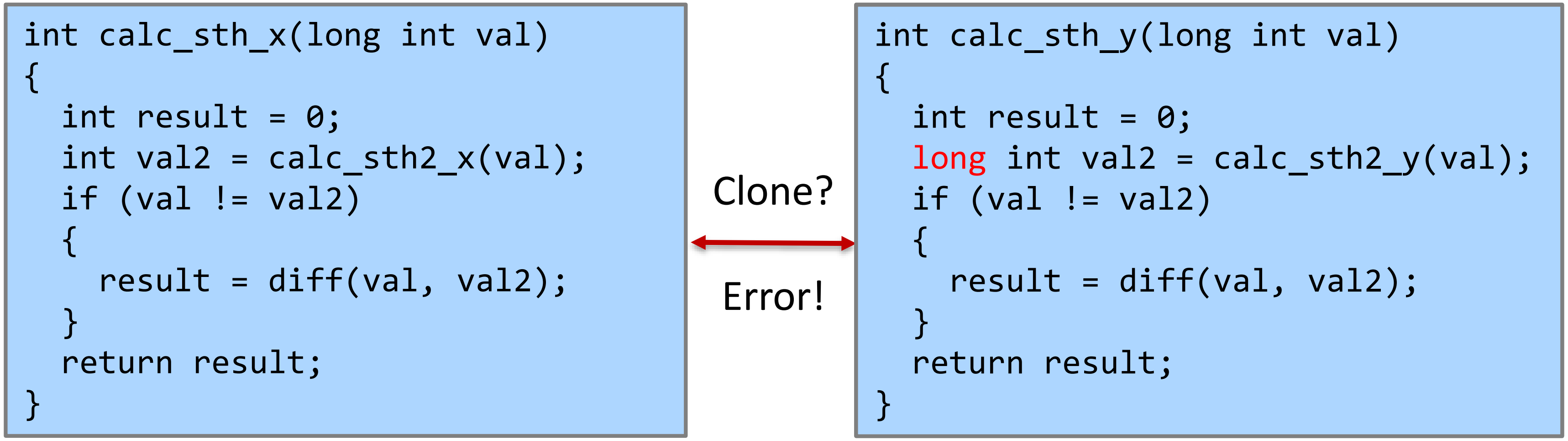
```
enum CommandX
{
    Start = 0,
    Stop,
    Reset,
    LoadConfig,
    SaveConfig,
    Pause,
    DebugMode,
    AlertUser
};
```

```
enum CommandY
{
    Start = 0,
    Stop,
    Reset,
    LoadConfig,
    SaveConfig,
    Pause,
    AlertUser
};
```

- Technically type 3, but quite certainly indicates an error!

Clone Management to find errors and/or maintenance burden

Dealing with type-3 clones that even indicate errors



```
int calc_sth_x(long int val)
{
  int result = 0;
  int val2 = calc_sth2_x(val);
  if (val != val2)
  {
    result = diff(val, val2);
  }
  return result;
}
```

Clone?

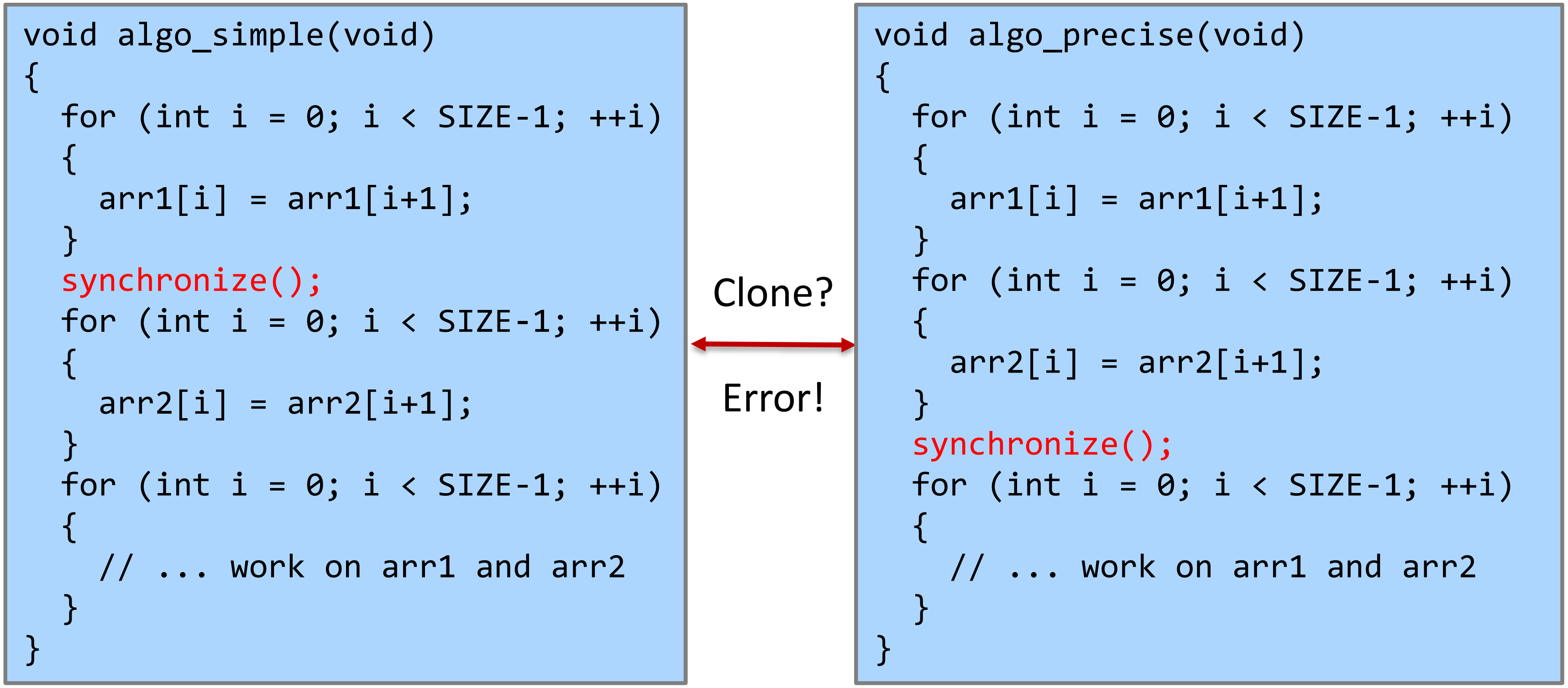
Error!

```
int calc_sth_y(long int val)
{
  int result = 0;
  long int val2 = calc_sth2_y(val);
  if (val != val2)
  {
    result = diff(val, val2);
  }
  return result;
}
```

- Technically type 3, but quite certainly indicates an error!

Clone Management to find errors and/or maintenance burden

Dealing with type-3 clones that even indicate errors



```
void algo_simple(void)
{
  for (int i = 0; i < SIZE-1; ++i)
  {
    arr1[i] = arr1[i+1];
  }
  synchronize();
  for (int i = 0; i < SIZE-1; ++i)
  {
    arr2[i] = arr2[i+1];
  }
  for (int i = 0; i < SIZE-1; ++i)
  {
    // ... work on arr1 and arr2
  }
}
```

Clone?

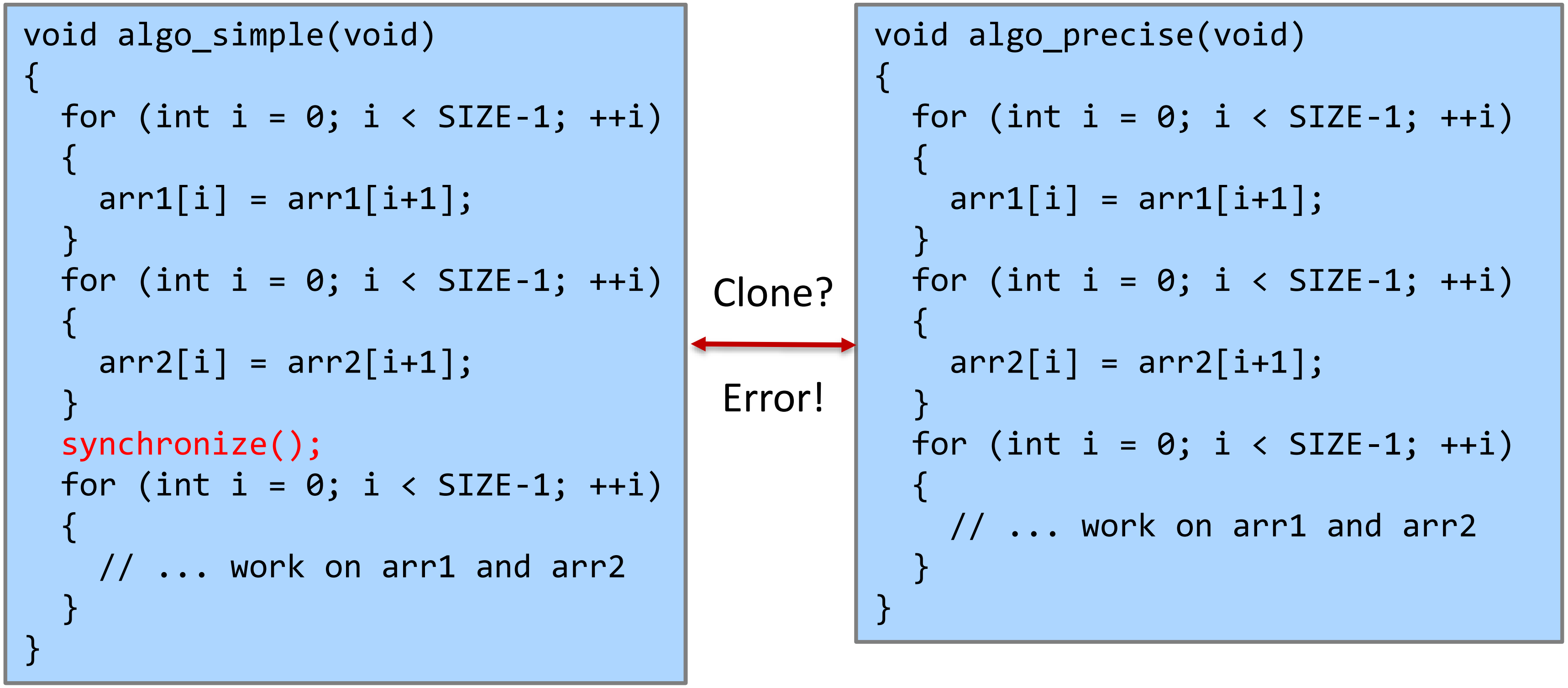
Error!

```
void algo_precise(void)
{
  for (int i = 0; i < SIZE-1; ++i)
  {
    arr1[i] = arr1[i+1];
  }
  for (int i = 0; i < SIZE-1; ++i)
  {
    arr2[i] = arr2[i+1];
  }
  synchronize();
  for (int i = 0; i < SIZE-1; ++i)
  {
    // ... work on arr1 and arr2
  }
}
```

- Technically type 3, but quite certainly indicates an error!

Clone Management to find errors and/or maintenance burden

Dealing with type-3 clones that even indicate errors



```
void algo_simple(void)
{
  for (int i = 0; i < SIZE-1; ++i)
  {
    arr1[i] = arr1[i+1];
  }
  for (int i = 0; i < SIZE-1; ++i)
  {
    arr2[i] = arr2[i+1];
  }
  synchronize();
  for (int i = 0; i < SIZE-1; ++i)
  {
    // ... work on arr1 and arr2
  }
}
```

Clone?

Error!

```
void algo_precise(void)
{
  for (int i = 0; i < SIZE-1; ++i)
  {
    arr1[i] = arr1[i+1];
  }
  for (int i = 0; i < SIZE-1; ++i)
  {
    arr2[i] = arr2[i+1];
  }
  for (int i = 0; i < SIZE-1; ++i)
  {
    // ... work on arr1 and arr2
  }
}
```

- Technically type 3, but quite certainly indicates an error!

Clone Management to find errors and/or maintenance burden

Technical issues for users

- Type-2 clones
  - Most of the time wanted to check for consistent changes in code
  - Some (technically correct!) clones however are annoying

- Solutions:
  - Threshold value N for number of allowed parametrization
    → still false positives for candidates <= N
    → and missing out valuable candidates > N
  - Report them all because tool cannot decide
    → let the user assess value of candidate

Clone Management to find errors and/or maintenance burden

Technical issues for users

- Type-3 clones
  - Most of the time unwanted because edit distance is too high and thus clone candidate far away from being helpful
  - Some clones however are strong indicators for coding bugs due to previous inconsistent changes to type-1/2 clones

- Solutions:
  - Threshold value N for number of allowed edit distance
    → still false positives for candidates <= N
    → and missing out valuable candidates > N
  - Report them all because tool cannot decide
    → let the user assess value of candidate

Responsibility of a clone

- The person who introduced the clone
  - He or she may be satisfied with his/her code, however, the author(s) of the cloned code may know why it is a bad idea to clone this code fragment

  → Notify all authors of the involved code fragments
- All authors of the involved code fragments
  - Some may not work in the team anymore
  - Some may be assigned to other tasks

  → Notify only the creator of the clone
- Solution: Configure per customer!

Refactoring of clones

- Pro
  - Maintenance easier in the future
- Contra
  - Possibility to introduce new bugs
  - Certified software may not be changed easily
  - Refactoring may not be done at the same time as normal development
  - Refactoring needs separate budget
- Solution: Do not refactor, but know where your clones are!

Visualization of clones

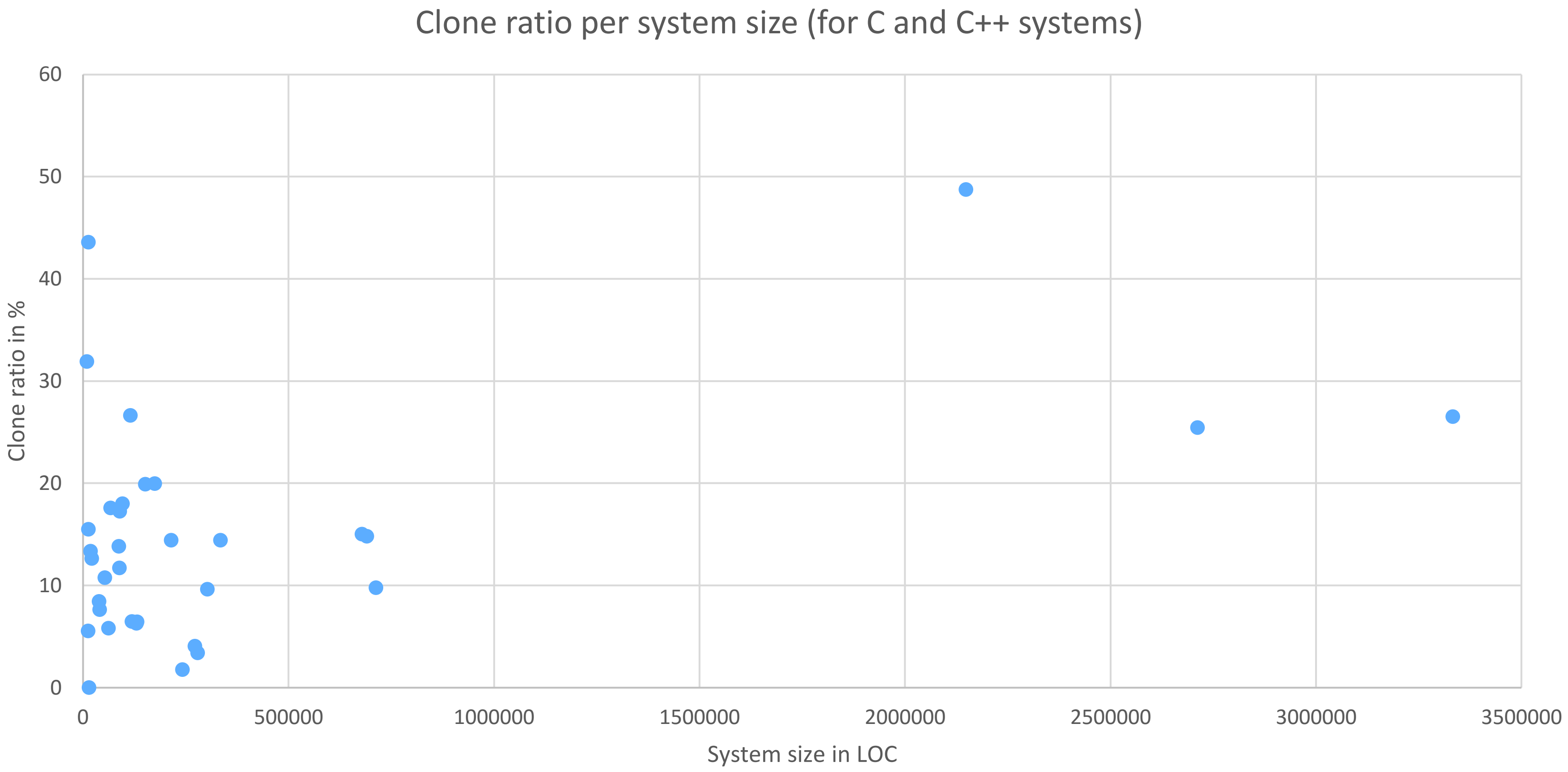
- Lots of papers propose fancy visualizations of software clones
- According to customer feedback, fancy visualization of clones does not help with the daily work of a software developer
- Developer needs clear instructions of what to do
- Developer needs information next to where he/she works anyway: the source code!

Clone Management to find errors and/or maintenance burden

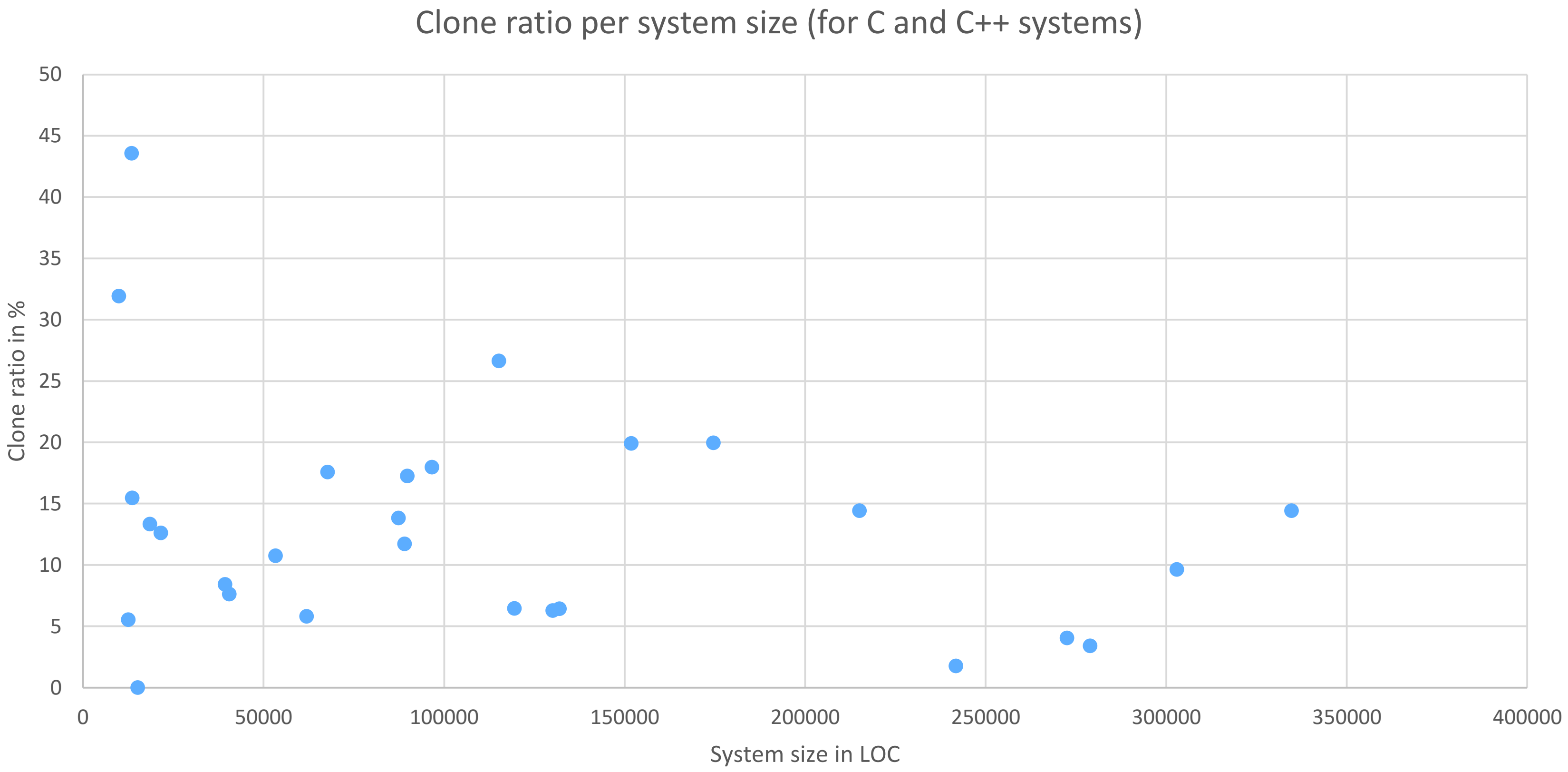
Advise

- Do not refactor clones in production code (unless you refactor anyway for other reasons)
- Refactor clones during development of new code
- Use clone management in the IDE for consistent bug removal (and for consistent changes)

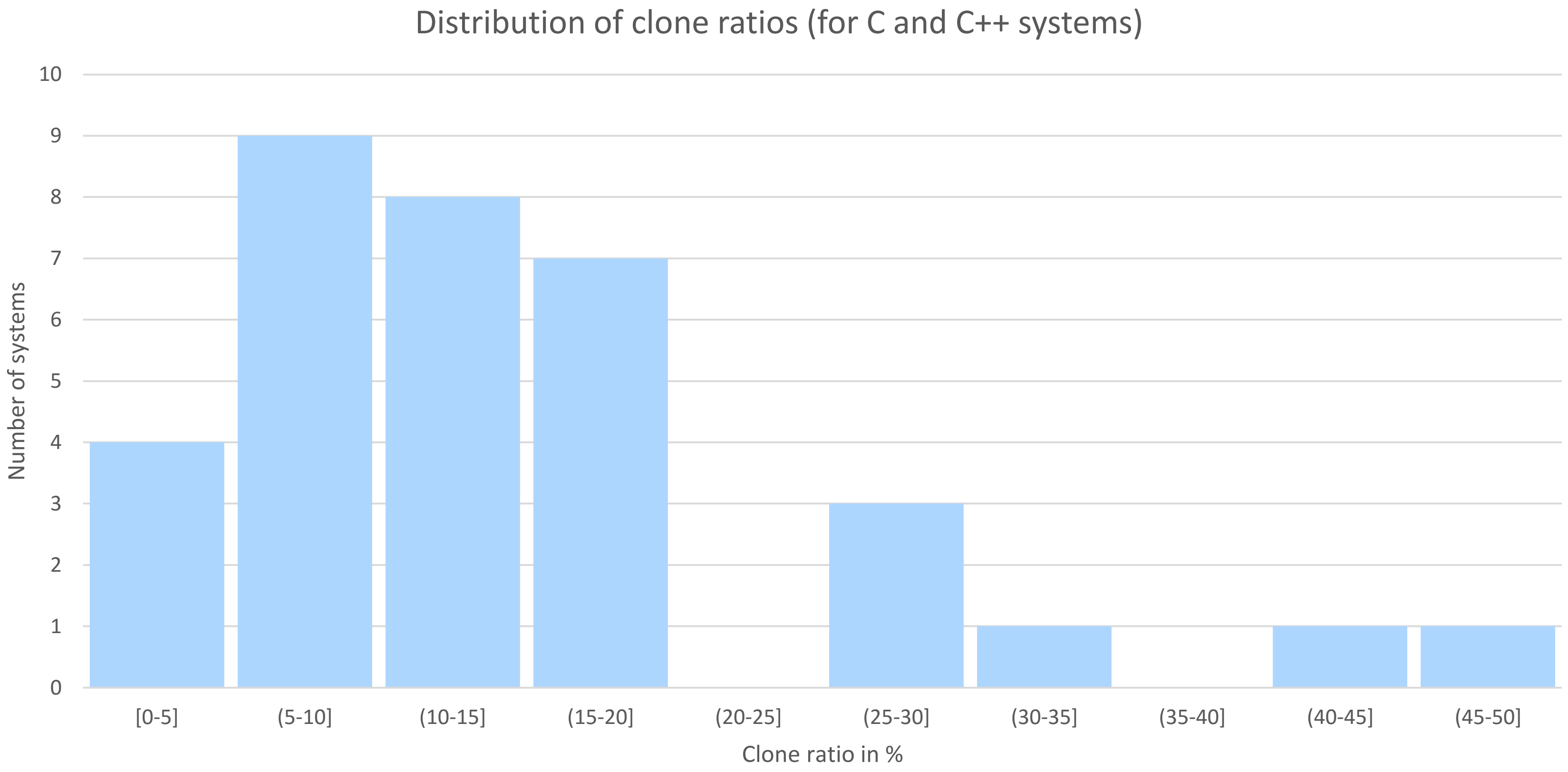
Clone ratios of industrial software systems



Clone ratios of industrial software systems (< 500 KLOC)



Clone ratios of industrial software systems



Clone Management to find errors and/or maintenance burden

Clone ratios of industrial software systems

- Systems between 10 KLOC and 3.3 MLOC
- Systems in C and C++
- Known generated code already filtered out
- Clones of at least 30 lines and 30 tree nodes per code fragment

• Smallest clone ratio encountered (15 KLOC system):	0.00 %
• Largest clone ratio encountered (2.1 MLOC system):	48.74 %
• Clone ratio of smallest system:	31.91 %
• Clone ratio of largest system:	26.49 %
• Average clone ratio:	14.90 %

How can clone detection help between product variants?

- Main interest in finding „clones of the past“, after all it is known that two variants are similar
- If products vary very little (if not just parameterization):
  - Not a task for clone detection, but rather a task for „diff“
- If products forked in the past, then developed separately:
  - Apply clone detection across the variants (i.e., filter out inner clones)
  - Filter out type 1 (i.e., common code across the variants)
  - Look at type 2 and type 3 to find potential code that needs to be synced

Clone Management in product lines

How can clone detection help between product variants?

- Reality in industrial environment
  - Systems in product lines often 100 KLOC or more
  - Often up to 40 variants or more of the same system
  - Variants developed by separate teams
  - No write access to code of other variants
  - No write access to code of shared core components

- Problems:
  - Cannot change code between variants in sync
  - Found bug fixes cannot easily be applied to other variants
  - Often developed as separate forks

- Clone Detection not suitable for daily usage

Clone Detection to check for license compliance

Technical aspects

- Create fingerprint for reference source code
- Feed fingerprint into reference corpus together with meta data (e.g., software name, version, license, etc.)
- Create fingerprint for candidate source code
- Emit meta data of matching fingerprints of reference corpus

Clone Detection to check for license compliance

Problems

- What to include in the reference corpus?
- How often to update?
- How to achieve big coverage?
- How to guarantee usefulness of reference corpus?

→ More legal problems than technical ones!

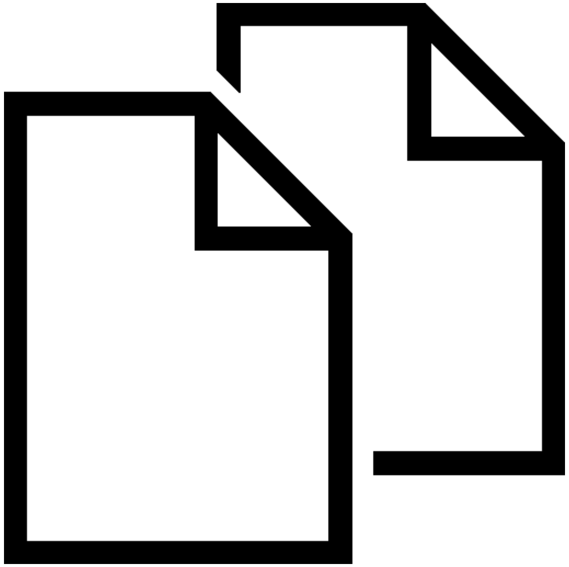
Demonstration of Axivion Bauhaus Suite

Conclusion

- Clones ARE common in industrial code and thus need attention
- Clone detection techniques are good enough “in general”
- More work needs to be done to distinguish wanted and unwanted clone candidates – but how?
- Seamless integration into development workflow/IDE essential → developers need precise instructions from the tool/workflow
- Managing product lines with clone detection mechanisms not straight forward
- License compliance checking has more legal problems than technical ones

One more thing …

One more thing …



… young people do not get the reference to Dolly any more!