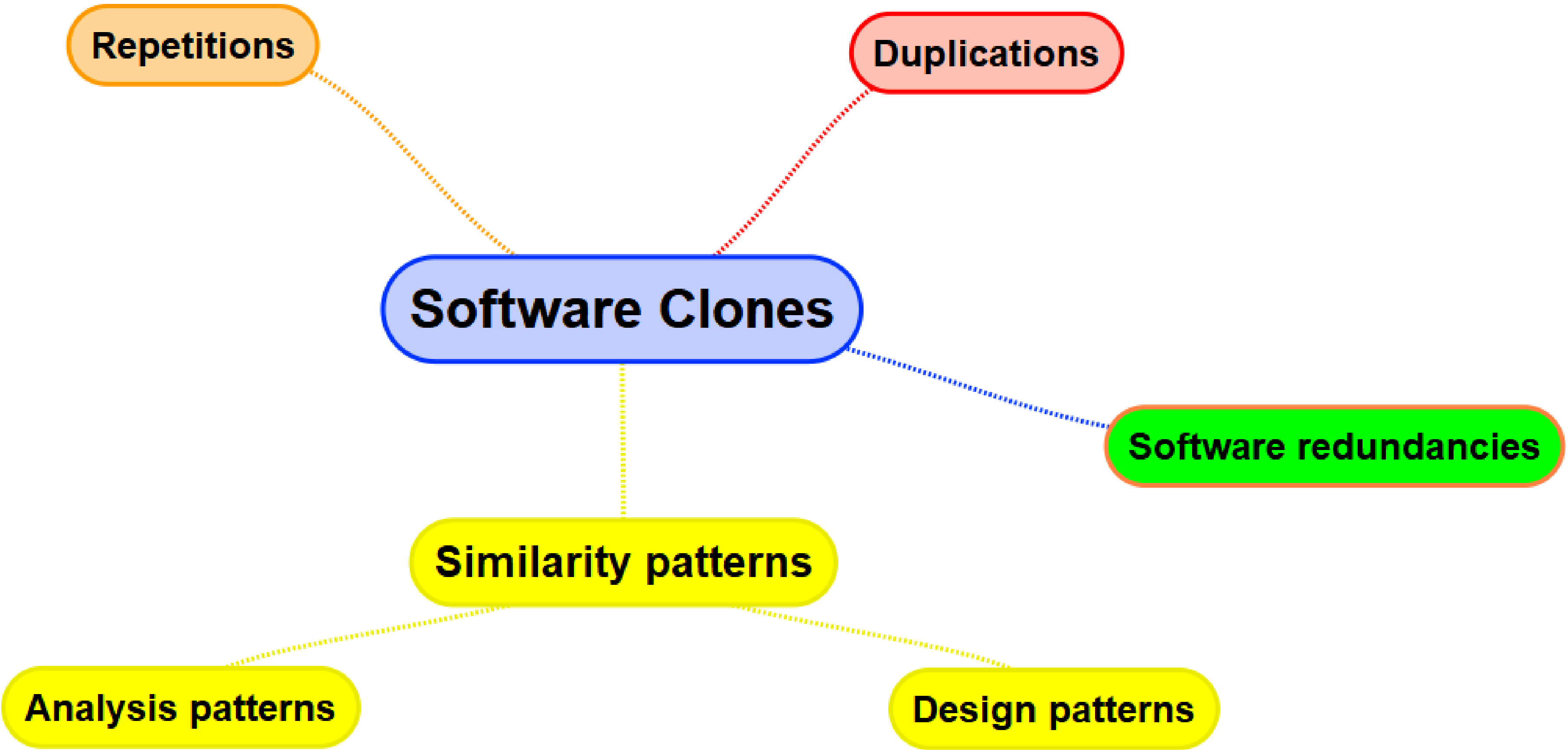
Software Similarities and Clones

A Curse or Blessing?

Stan Jarzabek
Bialystok University of Technology

Clone-related terms



Copy-paste-modify

Outlook

- What are software clones?
- Reasons for cloning

  Good clones – Bad clones?

- The quest for non-redundancy
- When we can’t avoid clones in a program?
- Software reuse and Software Product Lines
  - *Understanding software similarity is a key*

- How much *copy-paste-modify* have you used?

	1	2	3	4	5	
Not much						**A lot**

- Why programmers duplicate code?
- Are code clones generally good or bad?
  Or – cannot tell in general?
- “If I write code, there would be no clones! I could always parametrize code, use design patterns etc. to avoid clones”
  - *Is that what you think? Can programs be always clone-free?*
- Does it always make sense to remove clones?

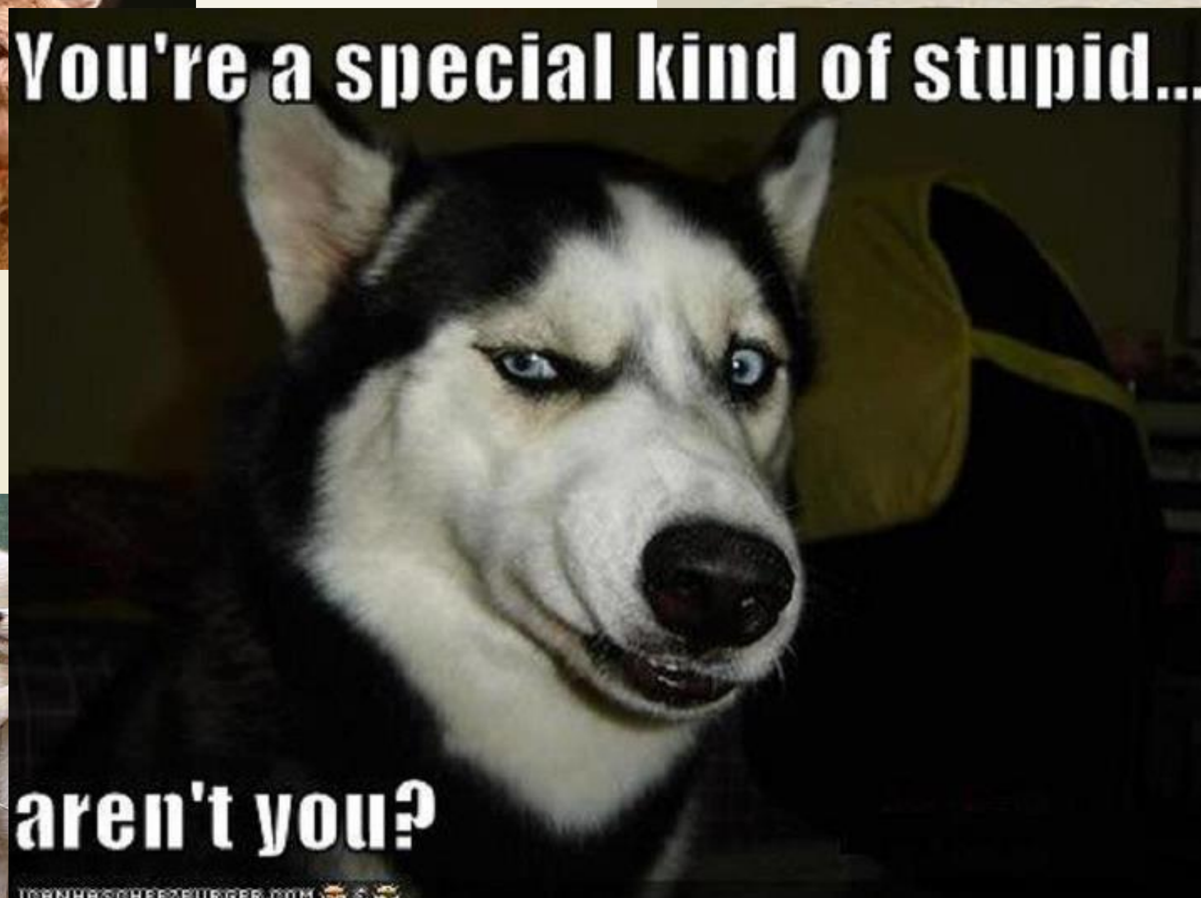
Software clones are not accidental

- Similarities are inherent in software
  - *Conceptual similarities are inherent in problem domains, program design*
  - *Conceptual similarities trigger clones in code*

Still, non-redundancy in programming is considered a virtue

- Generic programming, software reuse
- Similarities create opportunities for program simplification & productivity improvements

What are software clones?



Software Clones

- simple clones – recurring segments of code
  - *similar functions, class methods, any code fragments*

```
int f1(byte) {
int c;
  while (isalpha(c)) {
   if (p == buf) p=grow_buf(p);
    c = getc(finput);}
}
```

```
int f2() {
int d;
  while (isdigit(d)) {
   if (p == buf) p=grow_buf(p);
   skip();
   d = getc(finput); }
}
```

```
char f3(char) {
char c;
  while (isdigit(c)) {
    if (p == buf) p=grow_buf(p);
    if (c == '-') return;
    c = getc(finput);}
}
```

Simple clone types

```
int foo(byte) {
int c;
   While (isalpha(c)) {
       if (p == token_buffer)
       p = grow_token_buffer(p);
       c = getc(finput); }
}
```

Exact clones

```
int foo(byte) {
int c;
   While (isalpha(c)) {
        if (p == token_buffer)
       p =
     grow_token_buffer(p);
       c = getc(finput); }
}
```

Parametric clones

```
char bar(long)  {
char d;
   While (isdigit(d)) {
       if (p == token_buffer)
       p = grow_token_buffer(p);
      d = getc(finput); }
}
```

Gapped clones

```
char foo(long, float)  {
char d;
   While (isdigit(d)) {
        if (p == token_buffer)
       p =
     grow_token_buffer(p);
       if (d == '-') return;
       d = getc(finput); }
}
```

No common similarity metrics …

Multiple program representations

- Plain text
- Stream of tokens
- Abstract Syntax Trees
- Program Dependence Graphs
- Collection of software metrics
- Neural networks
- …

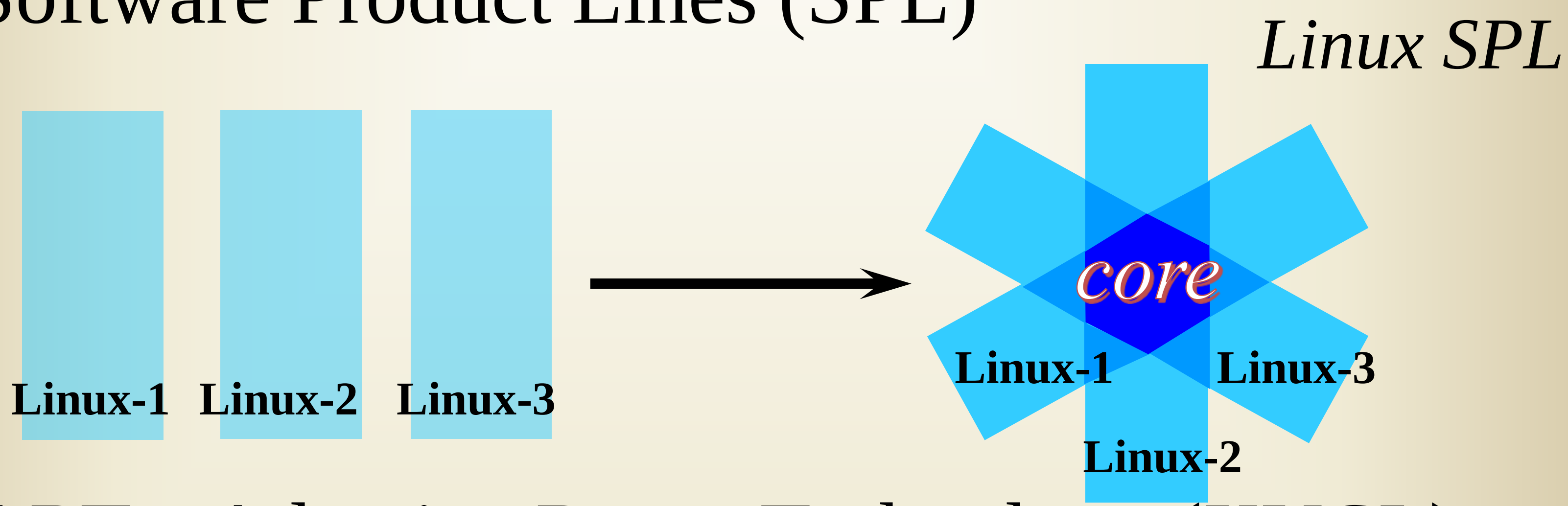
Code similarity metrics

- Clone similarity criteria and metrics:
  - *How big must be a fragment to be a clone?*
  - *Livenstein similarity measures*
  - *% of the same tokens in code fragments*
- The exact clone similarity criteria depend on the reasons why we are interested in clones:

"I am interested in clones at least 3 LOC and differing in no more that 15% tokens"

Why I got interested in clones?

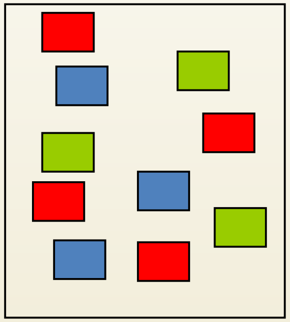
- Software Product Lines (SPL)



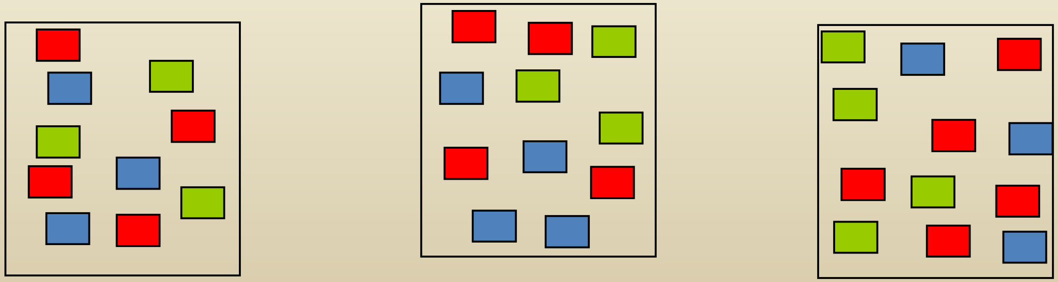
- ART – Adaptive Reuse Technology (XVCL)
  - *Variability management technique for SPL*
- How to migrate existing systems into SPL?
  - *Obviously, for that we need to find larger similarities than simple clones*

How are Linux versions similar and how they are different?

- Find similarities and differences within a single Linux version



- Find similarities and *differences* across Linux



Software Clones - def

- similar program structures, recurring in the same or similar form
- simple clones – recurring code fragments
  - *similar functions, class methods, any code fragments*
- structural clones – higher-level, larger similarities
  - *any similar program structures*
  - *similar classes, components, files, directories*
  - *patterns of collaborating components*

Structural Clone Classes

```
class A1 {
  f1 ()
  g1 ()
  h1 () }
```

```
class A2 {
  f2 ()
  g2 ()
  h2 () }
```

```
class A3 {
  g3 ()
  h3 ()
  f3 () }
```

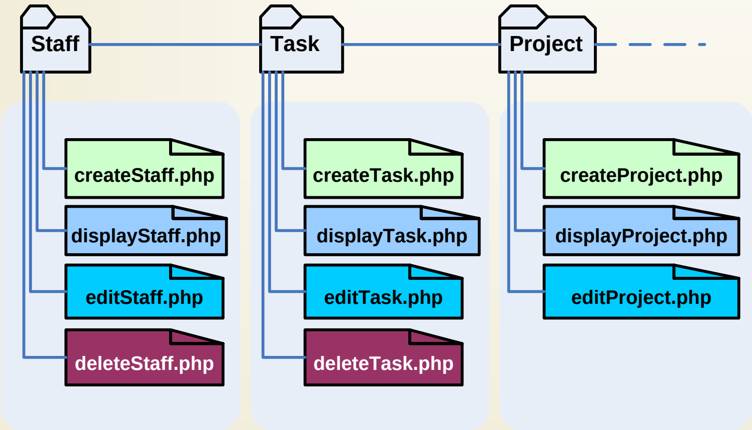
```
class A4 {
  f1 ()
  g1 ()
// h() is missing
```

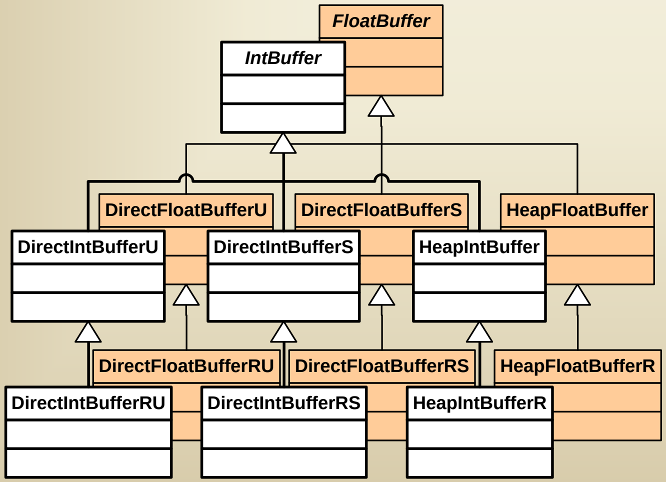
```
class A5 {
  f1 ()
  g1 ()
// h() is missing
  extraMethod ()
```

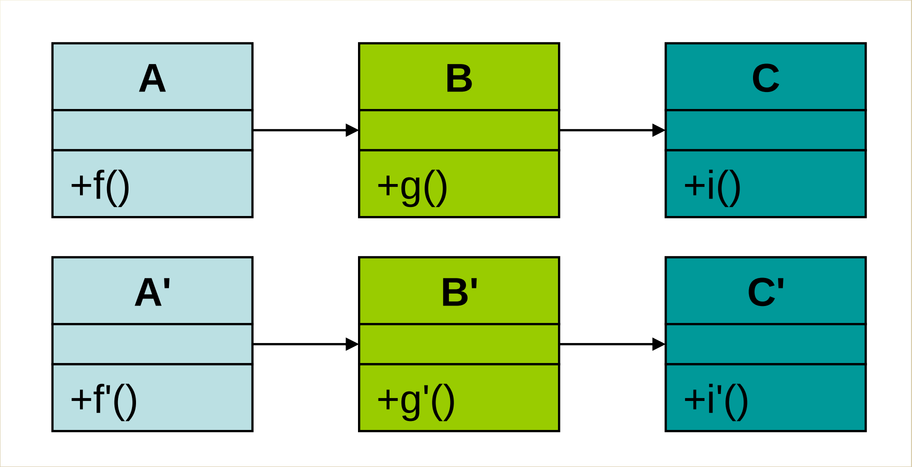
- Similarity of code and similarity of structure matter

Other structural clone types

Large-granularity repetitions, configurations of components

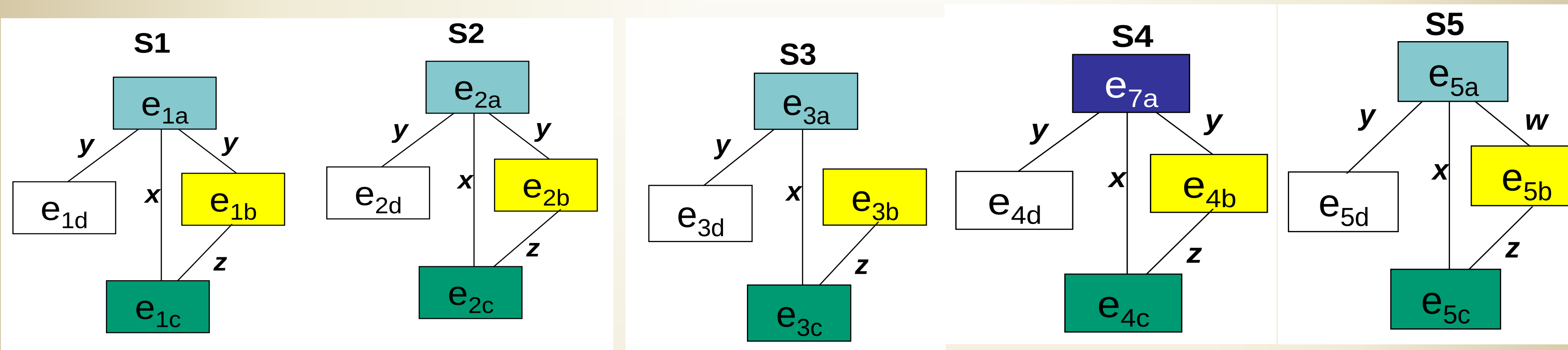






Collaborative structural clones

Structural clones are graphs



Bottom level:

- Recurring configurations of simple clones

Moving up the hierarchy (abstraction step):

- Recurring configurations of entities that have been recognized as clones of each other

Clone Miner (CM)

- Finds simple clones first
- Uses data mining techniques to find recurring configurations of simple clones

Basit, A.H. and Jarzabek, S. "Detecting Higher-level Similarity Patterns in Programs," *ESEC-FSE'05*, September 2005, Lisbon, pp. 156-165

Basit, H. A., Jarzabek, S. "Data Mining Approach for Detecting Higher-level Clones in Software," *IEEE TSE*, July/August 2009 (vol. 35 no. 4) pp. 497-514

Good clones? Bad clones?

No general answer

Questions to ask

- Why did programmers clone the code?
- What’s the role of clones in a program?
- Can we remove clones?

What’s the impact of clones on

program qualities that matter?

Clones and program maintainability

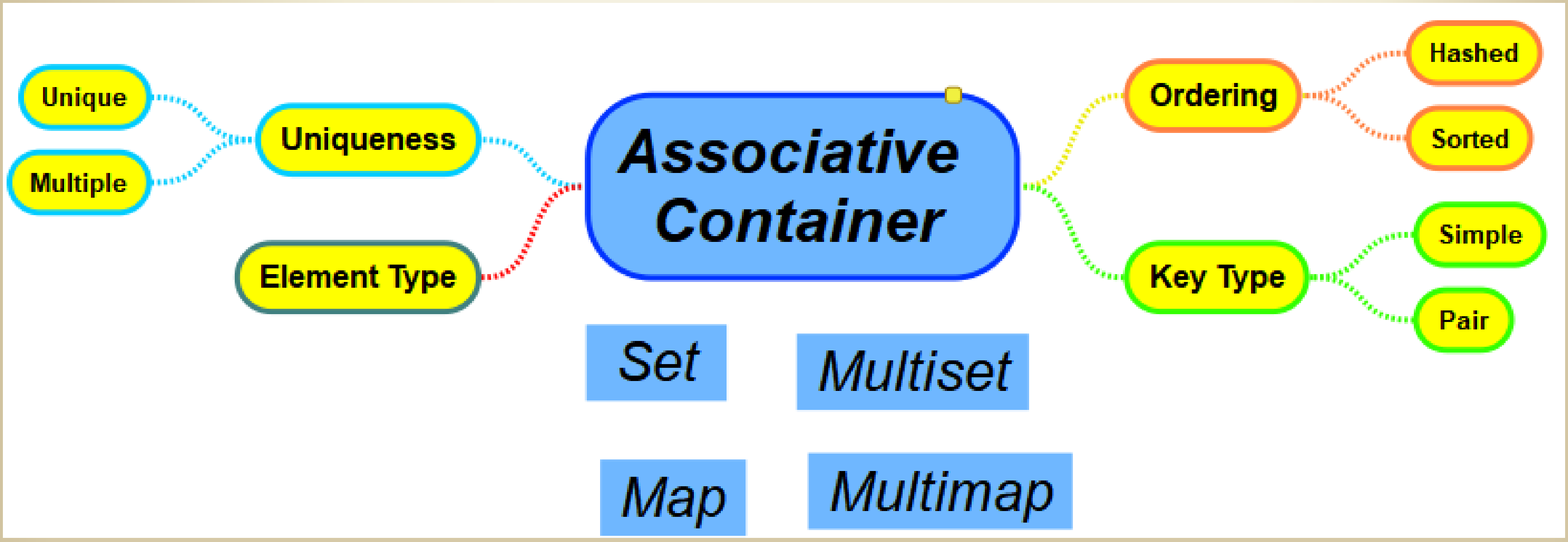
- Katsuro Inoue
  - *A precursor of clone research*
  - *Author of a popular clone detector CCFinder*
- Thousands of companies use *CCFinder* and cloning rates to assess software quality
- So why could clones be bad?
  - *More code to maintain*
  - *Increased risk of updates anomalies*
- On the other hand, clones due to standardization could be beneficial for maintenance

Reasons for cloning code

- C*opy-paste-modify*
  - *Ad hoc reuse practice for quick productivity gains*
  - *Reuse of proven solutions*
  - *Ad hoc maintenance, under pressure*
- Improve program performance
- Improve program reliability
- Standardization of program solutions
- Patter-driven development
- Limitations of programing techniques

Sources of cloning

- Clones in generated programs
  - *Do I have to maintain generated code?*
- “Feature combinatorics” phenomenon
  - *Observed by Batory, observed in many programs*



How much cloning?

High rates of cloning

percentage of repetitions	observed in:
20% – 50%	reengineering projects
68%	Java Buffer library
40% - 60%	STL in C++: containers and some functions
68% 50%	C# .NET; J2EE, command and control appl.
17% - 63%	17 Web Applications (simple, exact clones)
60% - 90%	Web portals ASP (simple + structural) clones
80% - 95%	business applications in COBOL (reuse with frame technology)

Potential benefits of non-redundancy

- Conceptual clarity
- Uniformity, conceptual integrity of the design (Brooks)
- Reduced risk of update anomalies
- Overall program simplification

The quest for non-redundancy is almost as old as programming …

- J. Goguen 1986 – parameterized programming
- Generics, templates – STL
- OO inheritance, application frameworks
- SAP – generic, parameterized ERP apps
- Refactoring (M. Fowler)
- Generative techniques, macros, AoP, ART
- Software reuse, Software Product Lines (SPL) – generic arch. for a family of similar products

Summary: clone categories

- Desirable clones
  - *Intentional clones: Play useful role in a program*
- Avoidable clones
  - *Caused by careless programming, poor design*
- Essential clones
  - *Induced by conceptual similarities in application domain or design technique*
  - *Difficult to refactor from programs, but*
  - *Can be handled with generative approaches*

Essential clones

Clones in STL?

Basit, H.A., Rajapakse, D.C., and Jarzabek, S. "Beyond Templates: a Study of Clones in the STL and Some General Implications," Int. Conf. Software Engineering, ICSE'05, St. Louis, USA, May 2005, pp. 451-459

STL: Data structures &Algo

- Sequences
  - *Vector*
  - *List*
  - *Deque*
- Associative Containers
  - *Set*
  - *Multiset*
  - *Map*
  - *Multimap*
  - *Hash Set*
  - *Hash Multiset*
  - *Hash Map*
  - *Hash Multimap*

- Container Adapters
  - Stack
  - Queue
  - Priority Queue

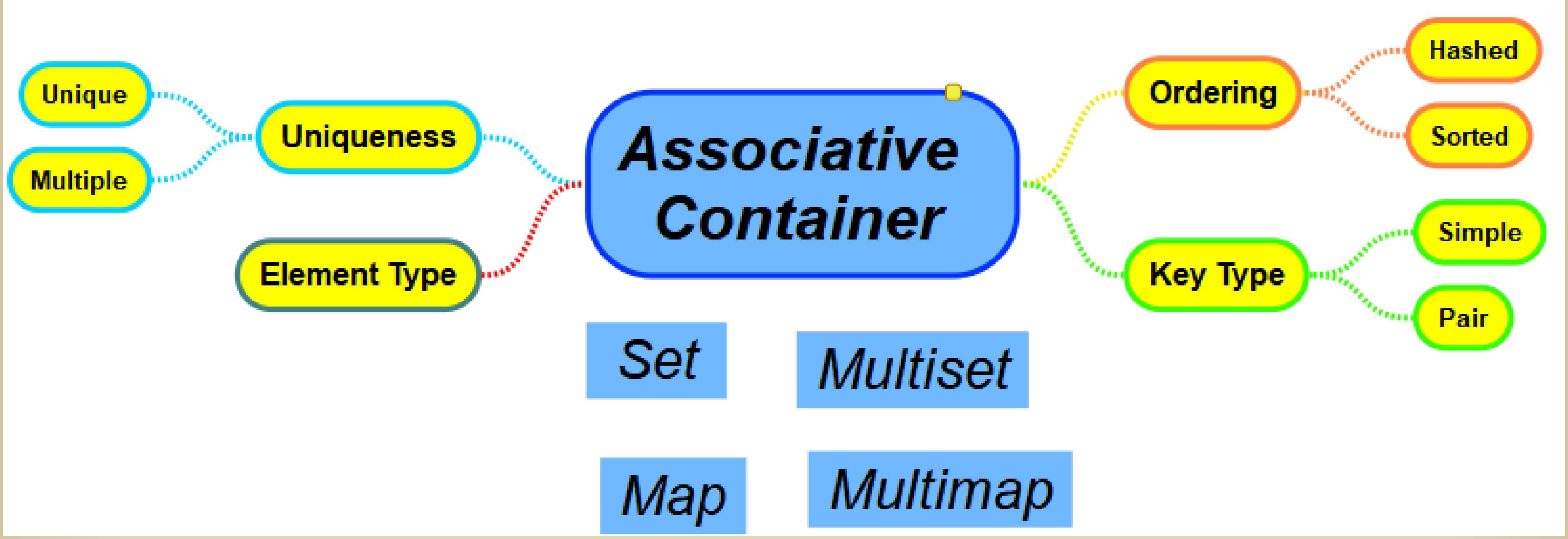
Sort()
Count()
Select()

STL

- A powerful example of non-redundant, generic template programming
- A collective effort of many smart people AT&T and HP Labs
- A range of OO techniques such as templates, adapters and iterators were invented/advanced
- STL: replace groups of similar classes with templates:
  - *intStack, floatStack, charStack* *Stack < T >*

Associative containers

- Variable-sized containers supporting efficient retrieval of elements via keys
- Features of associative containers:



Feature combinations

Class	Features		
	Storage	Uniqueness	Key Type
Set	sorted	unique	simple
Multiset	sorted	multiple	simple
Map	sorted	unique	pair
Multimap	sorted	multiple	pair
Hash Set	hashed	unique	simple
Hash Multiset	hashed	multiple	simple
Hash Map	hashed	unique	pair
Hash Multi	hashed	multiple	pair

Sample parametric clones in containers

```
Template
< class _Key, class _Compare,
  class _Alloc >
inline bool operator @op
(
 const
   set<_Key,_Compare,_Alloc>& __x,
 const
   set<_Key,_Compare,_Alloc>& __y
)
{
 return __x._M_t @op __y._M_t;
}
```

@op : `==`, `<`, `>`, etc.

Sample clones in iterators

```
_Self& operator++() {
    --current;
    return *this;
  }
  _Self operator++(int) {
    _Self __tmp = *this;
    --current;
    return __tmp;
  }
  _Self& operator--() {
    ++current;
    return *this;
  }
  _Self operator--(int) {
    _Self __tmp = *this;
    ++current;
    return __tmp;
  }
```

Sample clones in Helpers

```
template <class _Key, class _Compare, class _Alloc>
inline bool operator ==  (
     const set<_Key,_Compare,_Alloc>& __x,
      const set<_Key,_Compare,_Alloc>& __y) {
  return __x._M_t ==__y._M_t;
}
---------------------------------------------------------------------------
-----------
template <class _Key, class _Compare, class _Alloc>
inline bool operator <  (
     const set<_Key,_Compare,_Alloc>& __x,
      const set<_Key,_Compare,_Alloc>& __y) {
  return __x._M_t  < __y._M_t;
}
```

Sample clones

```
template <class _Tp>
inline valarray<_Tp> operator+(
   const valarray<_Tp>& __x, const _Tp& __c) {
  typedef typename valarray<_Tp>::_NoInit _NoInit;
  valarray<_Tp> __tmp(__x.size(), _NoInit());
  for (size_t __i = 0; __i < __x.size(); ++__i)
    __tmp[__i] = __x[__i]  + __c;
  return __tmp; }
------------------------------------------------------------------
template <class _Tp>
inline valarray<_Tp> operator+(
  const _Tp& __c, const valarray<_Tp>& __x) {
  typedef typename valarray<_Tp>::_NoInit _NoInit;
  valarray<_Tp> __tmp(__x.size(), _NoInit());
  for (size_t __i = 0; __i < __x.size(); ++__i)
    __tmp[__i] = __c + __x[__i];
  return __tmp; }
```

Cloning level in STL

templates	**cloned code**
8 assoc. containers	50% of cloned code, *can be unified by 2 power-generics*
stack and queue	40% of cloned code
set operations: union, intersection, etc.	50% of cloned code, can be unified by one generic operation

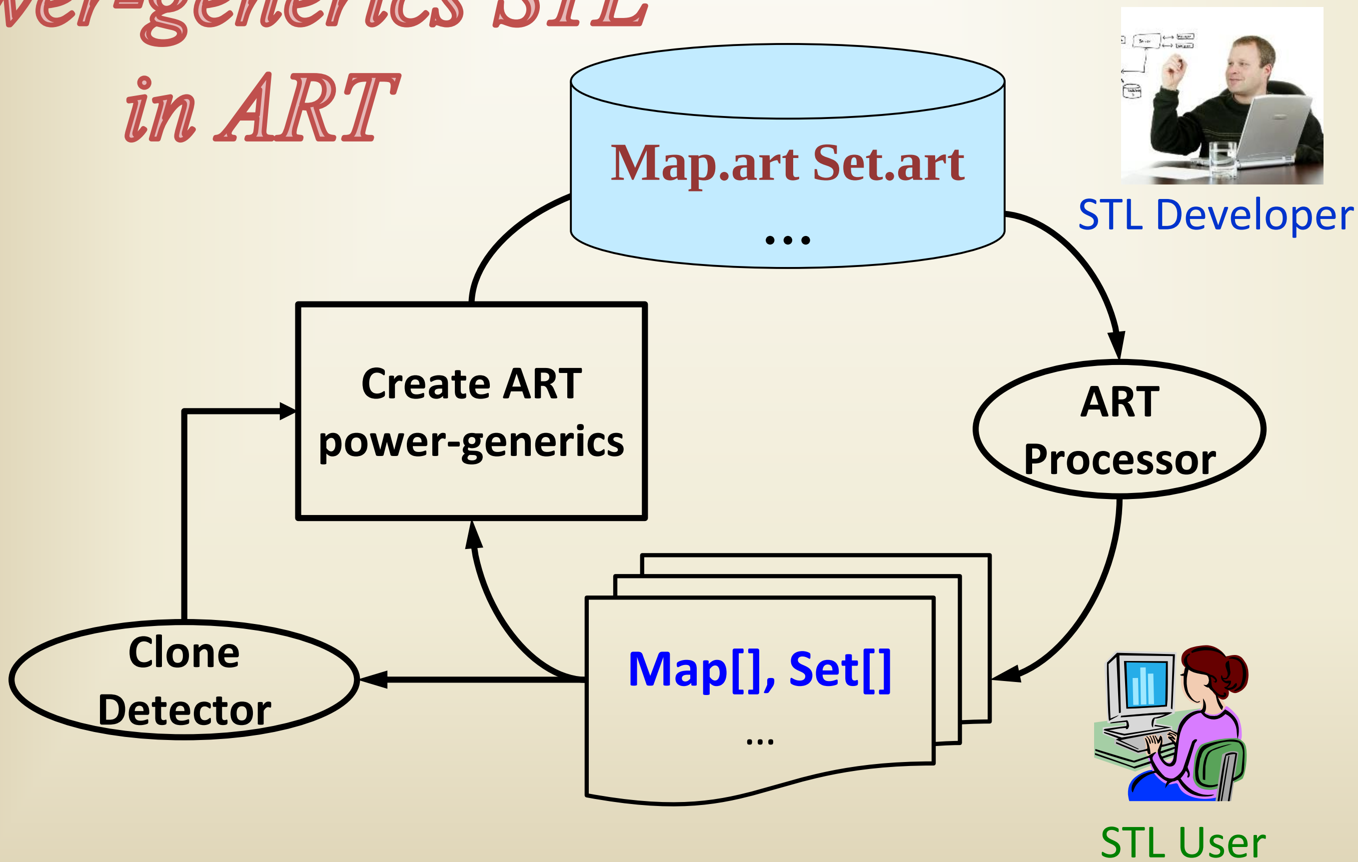
Differences between clones

- Different names for cloned classes and methods
- Different return types for cloned methods
- Extra parameters in cloned templates
- Extra **typedef**s in cloned code
- Type variations in cloned **typedef**s
- Extra methods in cloned classes
- Small algorithmic variations in cloned methods

It's all in details

Clone-free STL with *power-generics*

Power-generics STL in ART



Original STL

Clones in Buffer Library

JDK

Jarzabek, S. and Li, S. "Eliminating Redundancies with a "Composition with Adaptation" Meta-programming Technique," *Proc. ESEC-FSE'03, European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ACM Press, September 2003, Helsinki, pp. 237-246 (ACM Distinguished Paper award)

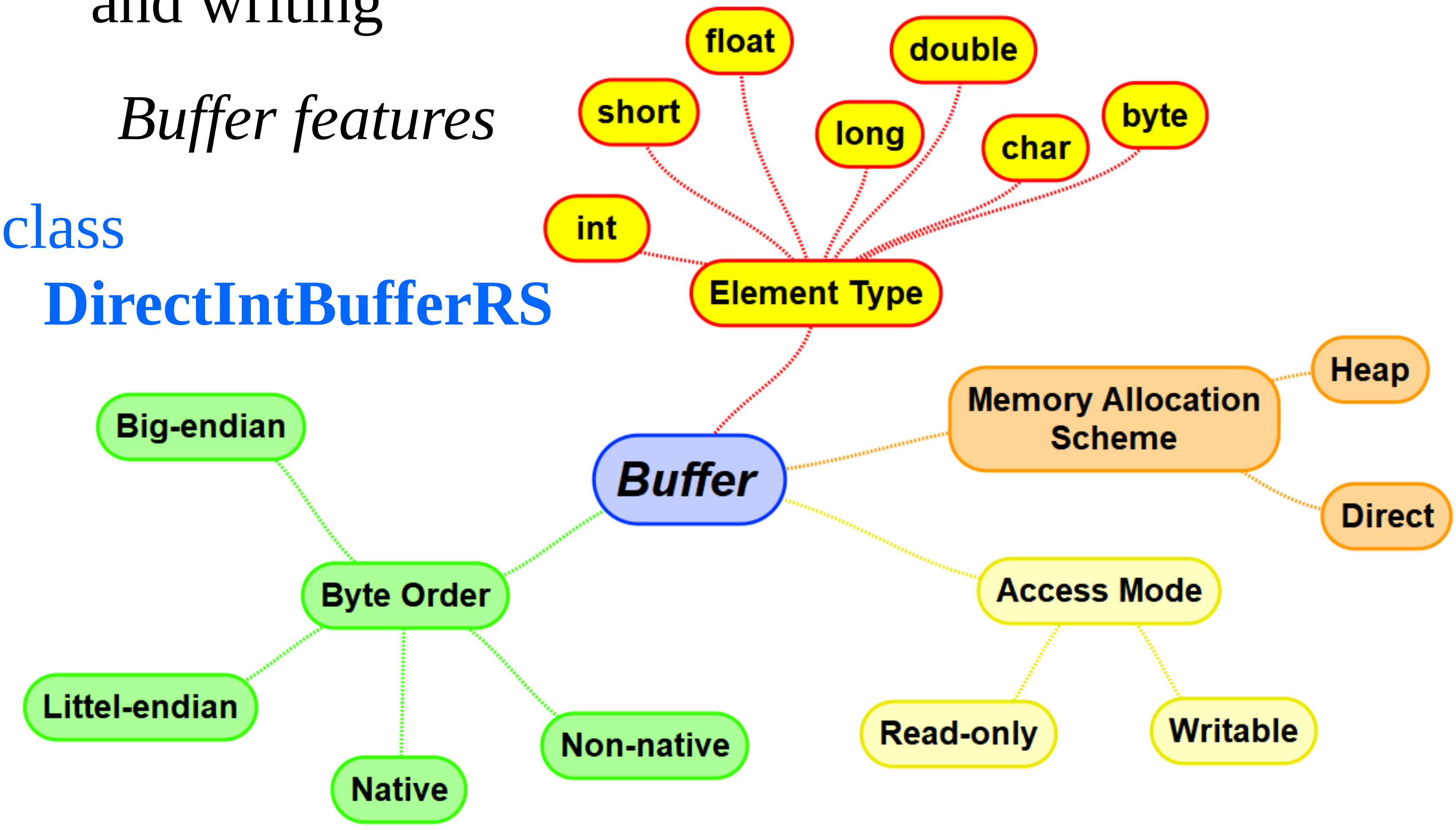
Jarzabek, S. and Li, S. "Unifying Clones at the Meta-Level for Enhanced Changeability: A Case Study and General Implications," *Journal of Software Maintenance and Evolution: Research and Practice* John Wiley & Sons, Volume 18, Issue 4, July/August 2006, pp. 267-292

Buffer library

- Buffer contains data in a linear sequence for reading and writing

Buffer features

class **DirectIntBufferRS**



Ideal solution for Buffer classes

A template:

[MS] [T] Buffer [AM] [BO]

MS – memory scheme: Direct, Heap

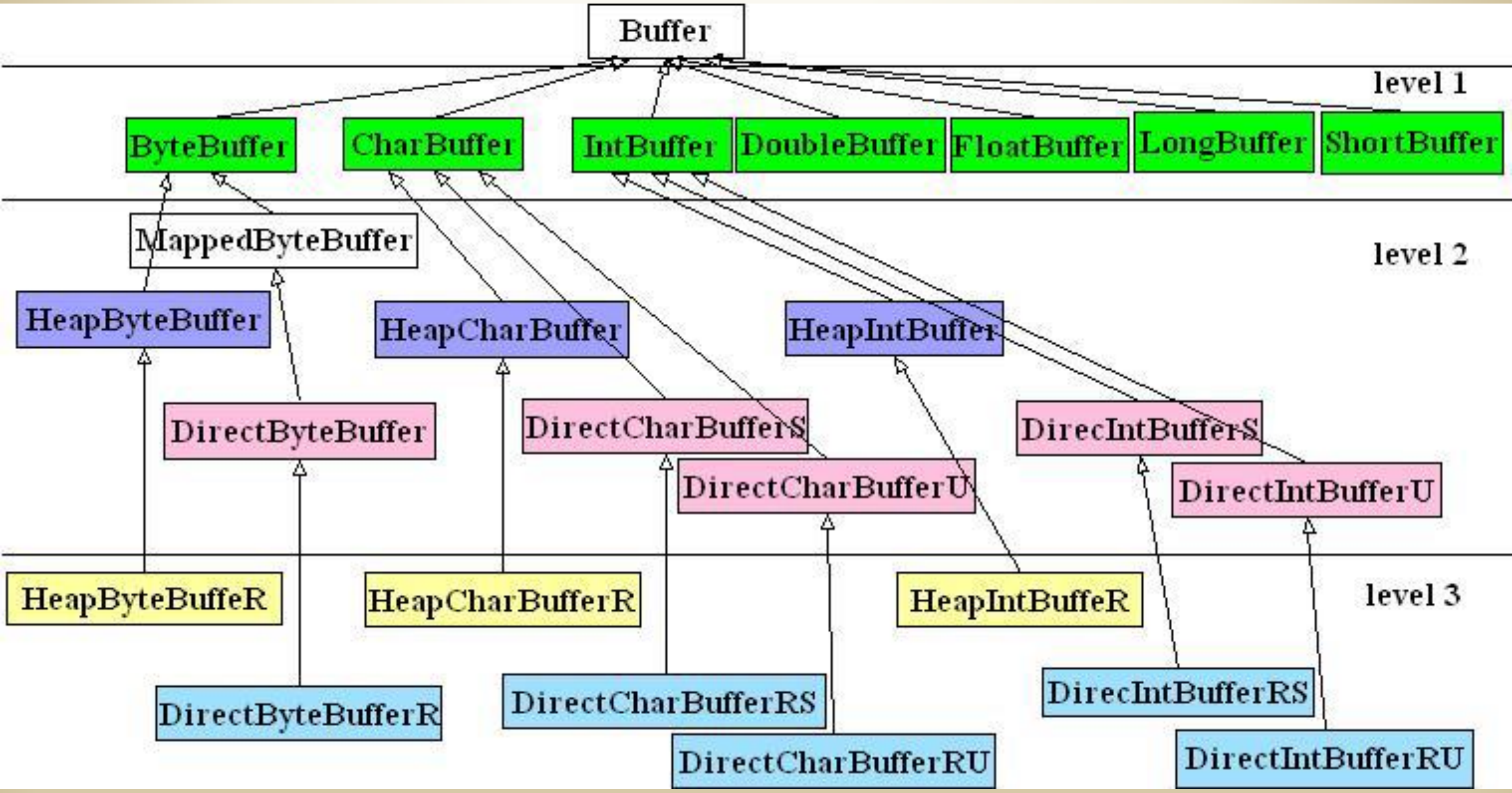
T – type: Byte, Char, Int, Double, Float, Long, Short

AM – access mode: R - read-only, W - writable

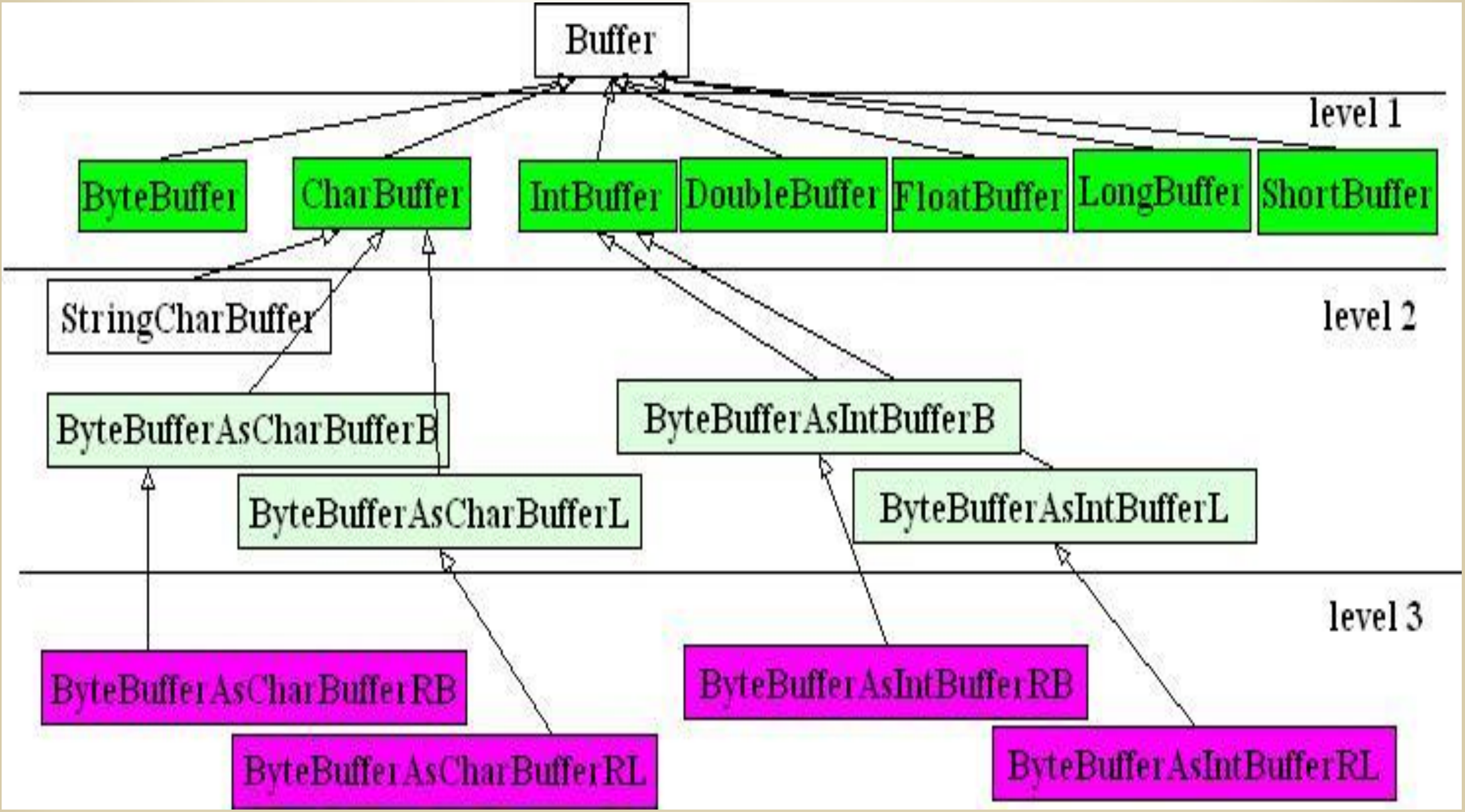
BO – byte ordering: S – non-native U – native,

B – BigEndian, L - LittleEndian

Slice of the Buffer library

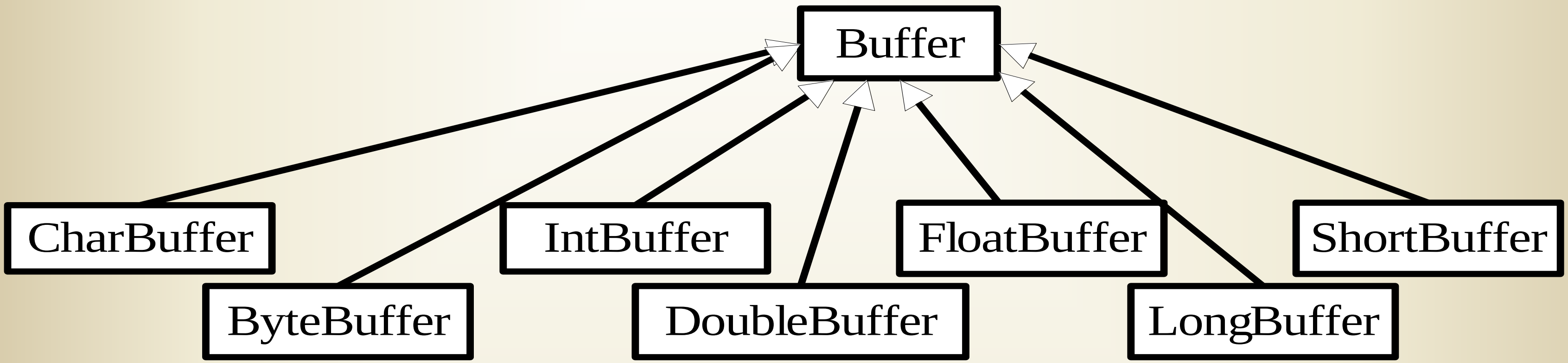


Buffer library



Could we refactor classes with inheritance or generics to avoid cloning?

Method *hasArray() repeated in 7 classes*

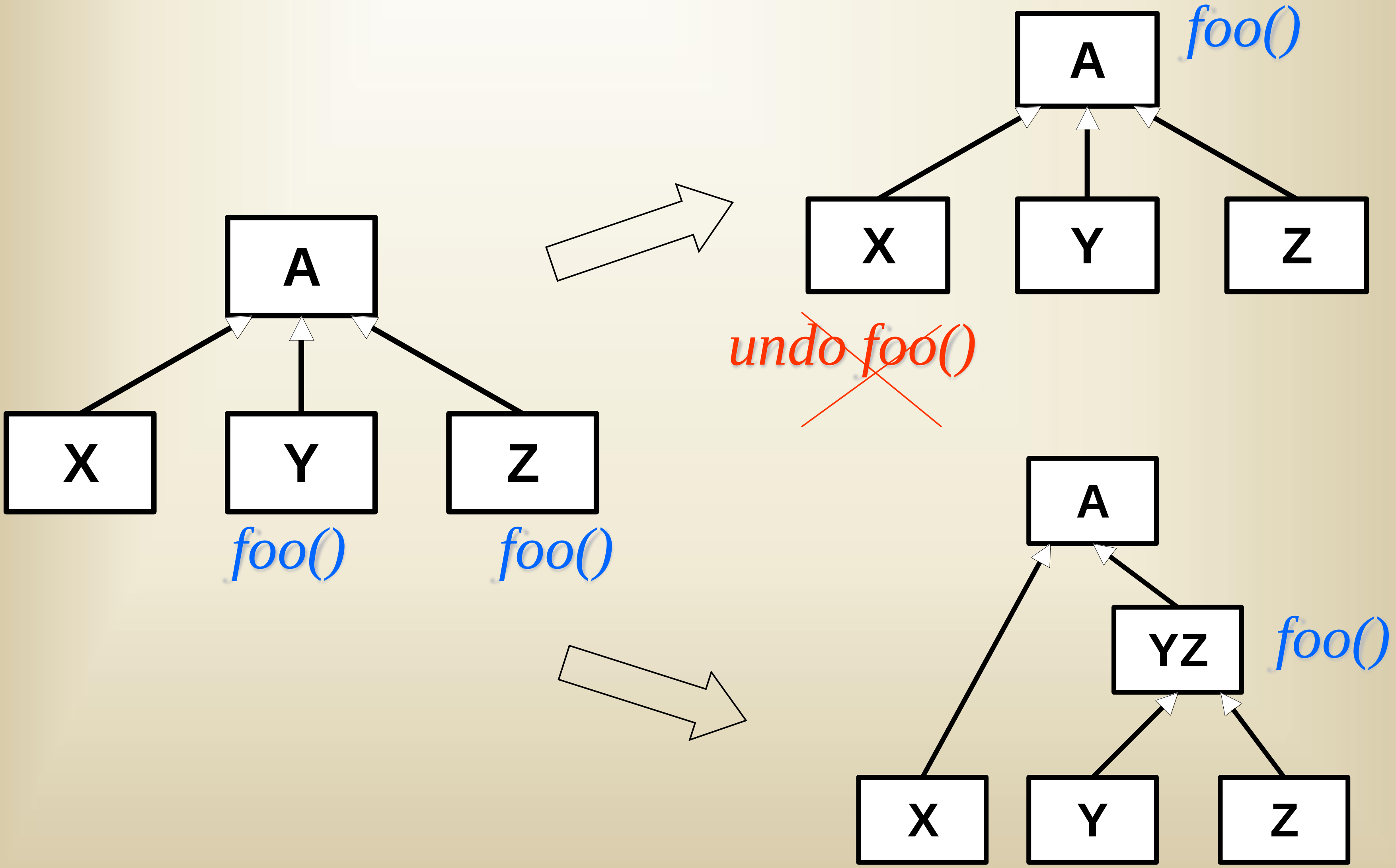


class CharBuffer {
  char **hb**;
  public final boolean ***hasArray*** () {
  return (**hb** != null) && !isReadOnly ; } }

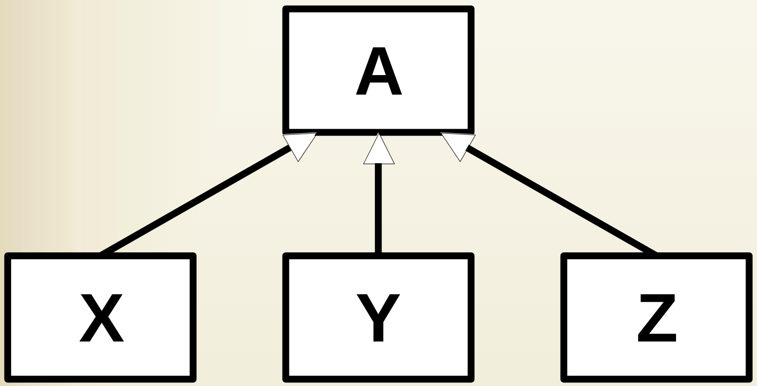
---

class IntBuffer {
  int **hb**;
  public final boolean ***hasArray*** () {
  return (**hb** != null) && !isReadOnly ; } }

Inheritance : example 1



Inheritance: example 2



foo(int) *foo()* *foo()*

int foo()

foo()

```
class X {
   int x;
   foo () {
          x = 2;  }
}
```

```
class Y {
   double y;
   foo () {
          y = 5;  }
}
```

An example of method clone

```
/*Creates a new byte buffer */
public ByteBuffer slice() {
        int pos = this.position();
        int lim = this.limit();
        assert (pos <= lim);
        int rem = (pos <= lim ? lim - pos : 0);
        int off = (pos << 0);
        return new DirectByteBuffer (this, -1, 0, rem, rem, off);
}
```

Java with generics

- Generics-unfriendly non-type parametric differences:
  - *constants, operators, keywords, names*

```
private int doSomething(Integer op1, Integer op2) {
      Integer retval = doSomethingElse(op1,op2);
      print(retval);
      return retlval;
}
```

```
protected int doSomething(Double op1, Double op2) {
      Double retval = doSomethingElse(op1,op2);
      print(retval);
      return retlval;
}
```

Generics-unfriendly variations

```
public abstract class CharBuffer
    extends Buffer implements Comparable, CharSequence{
```

```
public String toString() {
  StringBuffer sb = new StringBuffer();
  sb.append(getClass().getName());
            ...
  sb.append(capacity());
  sb.append("]");
  return sb.toString();     }
```

```
    public String toString() {
       return toString(position(), limit());}
```

- Extra methods in some of the classes

Differences among cloned classes in each group

- Type parameters in attribute declarations and methods
- Non-type parameters
  - *operators, keywords, constants, names*
- Minor or major editing changes
- Different implementation of the same method
- Extra methods in certain classes
- Details in method signatures, ‘implements’ clause, etc.

Golden Mean: Groups of similar classes

[T]Buffer

classes for T : Byte, Char, Int, Double, Float, Long, Short

Heap[T]Buffer

Heap[T]BufferR

R: read-only

byte ordering: S – non-native U – native

Direct[T]Buffer[S|U]

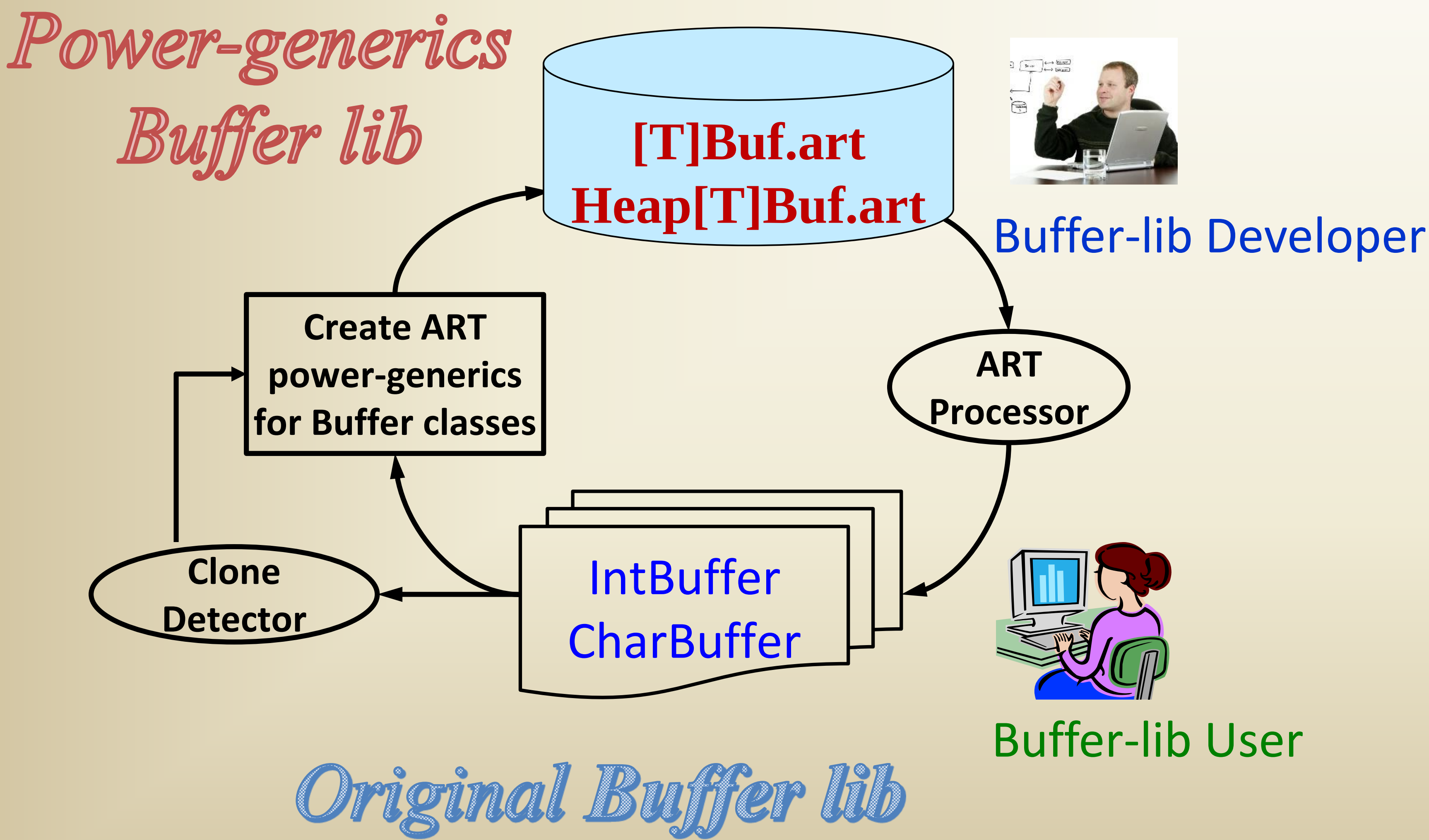
Direct[T]BufferR[S|U]

byte ordering: B – BigEndian L - LittleEndian

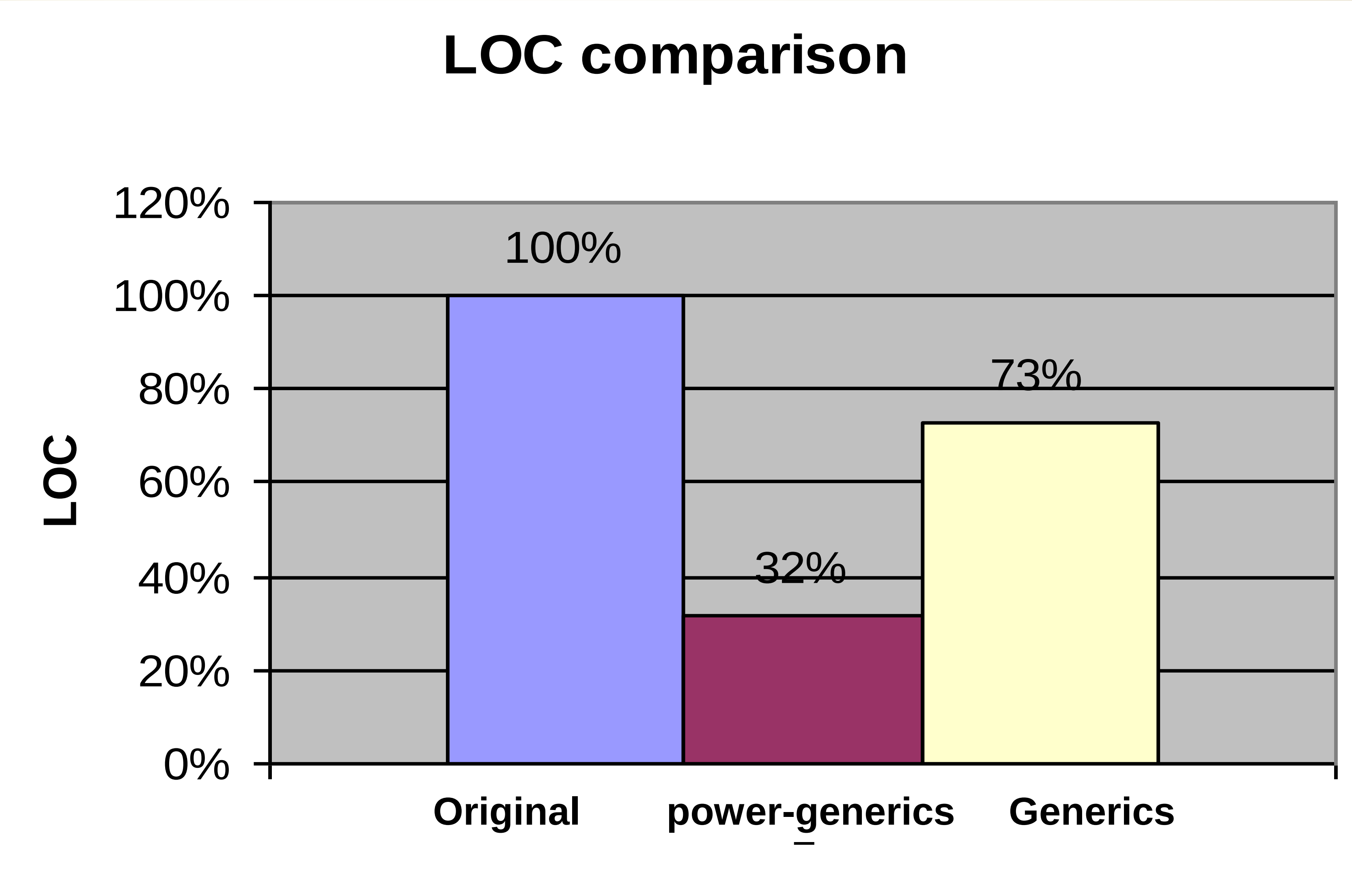
ByteBufferAs[T]Buffer[B|L]

ByteBufferAs[T]BufferR[B|L]

Clone-free Buffer lib with power-generics



The extent of cloning in Buffer classes



Evaluation

Clone-free is smaller - but is it better?

Maintenance effort: clone-free vs. the original Buffer lib

- suppose we add Complex buffer

New classes	Changes in original Buffer library		Changes in clone-free lib	
	No.	type	No.	type
ComplexBuffer	25	automatic	3	manual
	17	manual		
	2	manual	2	manual
HeapComplexBuffer	21	automatic	3	manual
	10	manual		
HeapComplexBufferR	16	automatic	3	
	5	manual		

Big Clones in Linux kernel

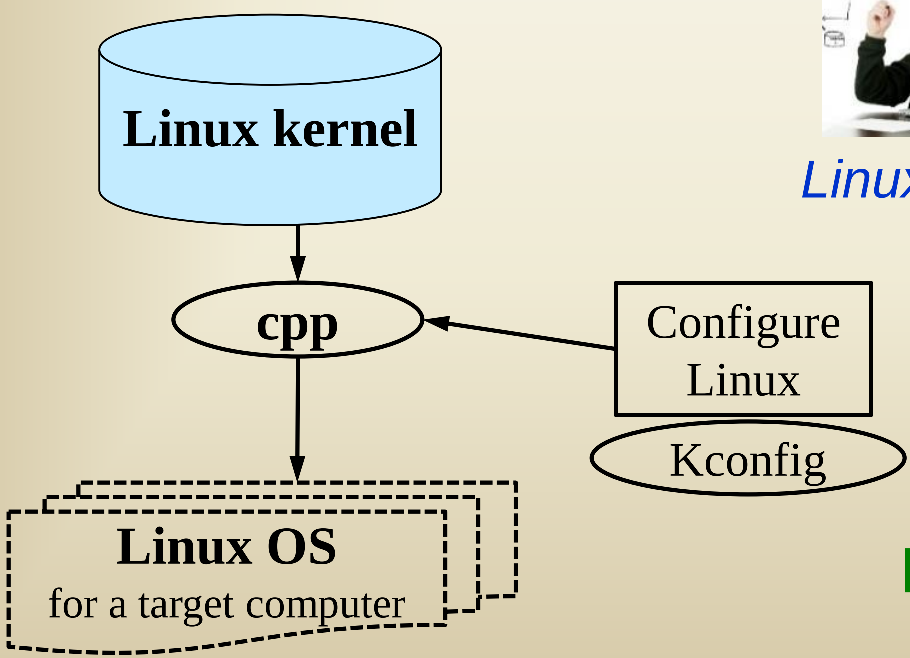
Linux OS runs on hundreds of computer brands,

with varying architectures, processors

Kuldeep Kumar, Stan Jarzabek and Daniel Dan „Managing Big Clones to Ease Evolution: Linux Kernel Example," *Federated Conference on Computer Science and Information Systems, FedCSIS, 36th IEEE Soft. Eng. Workshop,* Sept. 2016, pp. 1767 – 1776

Linux OS SPL

- ***Linux kernel***: A generic, parameterized code base from which to build Linix OSes for target computers
- 15 million of well-designed and documented code
- Variability managed with cpp, Kconfig, other tools



Develops & evolves Linux kernel

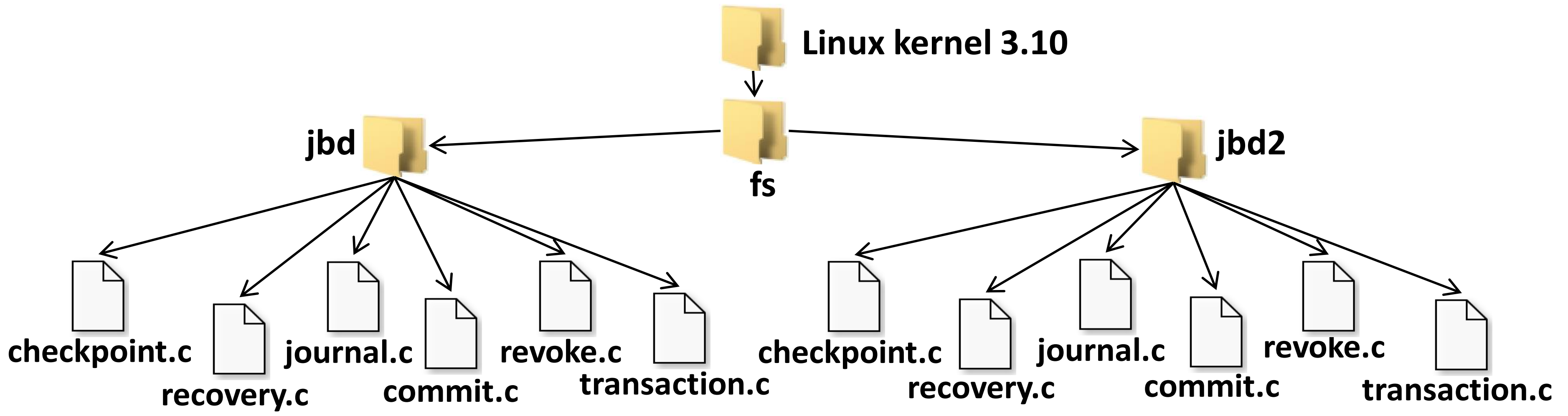
Linux Expert

Builds OS by customizing Linux kernel

Linux SysAdmin

Cloned directories in Linux kernel

- Journaling Block Device (JBD):



Sample cloned code in jbd files

Identical Code Fragments : ~554 LOC

```
static inline void __buffer_unlink(struct journal_head *jh)
{
        transaction_t *transaction = jh->b_cp_transaction;

        __buffer_unlink_first(jh);
        if (transaction->t_checkpoint_io_list == jh) {
                transaction->t_checkpoint_io_list = jh->b_cpnext;
                if (transaction->t_checkpoint_io_list == jh)
                        transaction->t_checkpoint_io_list = NULL;
        }
}
```

```
static inline void __buffer_unlink(struct journal_head *jh)
{
        transaction_t *transaction = jh->b_cp_transaction;

        __buffer_unlink_first(jh);
        if (transaction->t_checkpoint_io_list == jh) {
                transaction->t_checkpoint_io_list = jh->b_cpnext;
                if (transaction->t_checkpoint_io_list == jh)
                        transaction->t_checkpoint_io_list = NULL;
        }
}
```

Code Fragments with Parametric Changes: ~47 LOC

```
while (__log_space_left(journal) < nblocks) {
        if (journal->j_flags & JFS_ABORT)
                return;
        spin_unlock(&journal->j_state_lock);
        mutex_lock(&journal->j_checkpoint_mutex);
```

```
while (__jbd2_log_space_left(journal) < nblocks) {
        if (journal->j_flags & JBD2_ABORT)
                return;
        write_unlock(&journal->j_state_lock);
        mutex_lock(&journal->j_checkpoint_mutex);
```

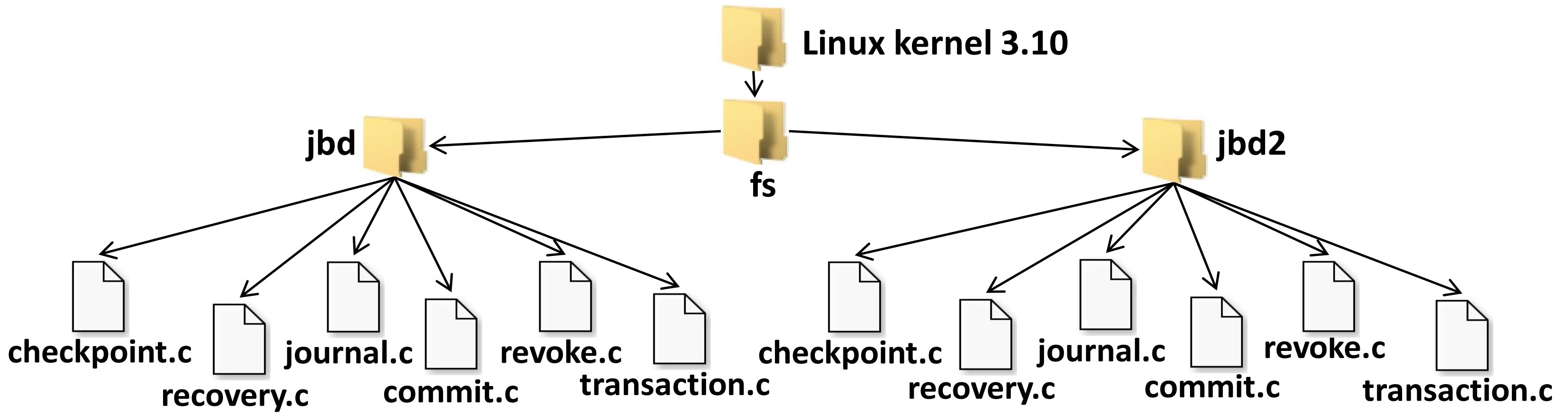
Code Modification: ~12 LOC

```
set_buffer_jwrite(bh);
bhs[*batch_count] = bh;
__buffer_relink_io(jh);
jbd_unlock_bh_state(bh);
(*batch_count)++;
if (*batch_count == NR_BATCH) {
        spin_unlock(&journal->j_list_lock);
        __flush_batch(journal, bhs, batch_count);
```

```
journal->j_chkpt_bhs[*batch_count] = bh;

__buffer_relink_io(jh);
transaction->t_chp_stats.cs_written++;
(*batch_count)++;
if (*batch_count == JBD2_NR_BATCH) {
        spin_unlock(&journal->j_list_lock);
        __flush_batch(journal, batch_count);
```

Clones in JBD



File Name	Total LOC	Identical LOC	LOC with param. Diff.	Modified LOC	Inserted LOC	Deleted LOC
checkpoint.c	700	554	47	12	29	95
commit.c	1000	523	93	35	364	218
journal.c	2100	1266	287	29	690	229
recovery.c	700	420	52	12	234	0
revoke.c	700	544	94	3	25	0
transaction.c	2300	1346	130	56	516	399

Other duplications

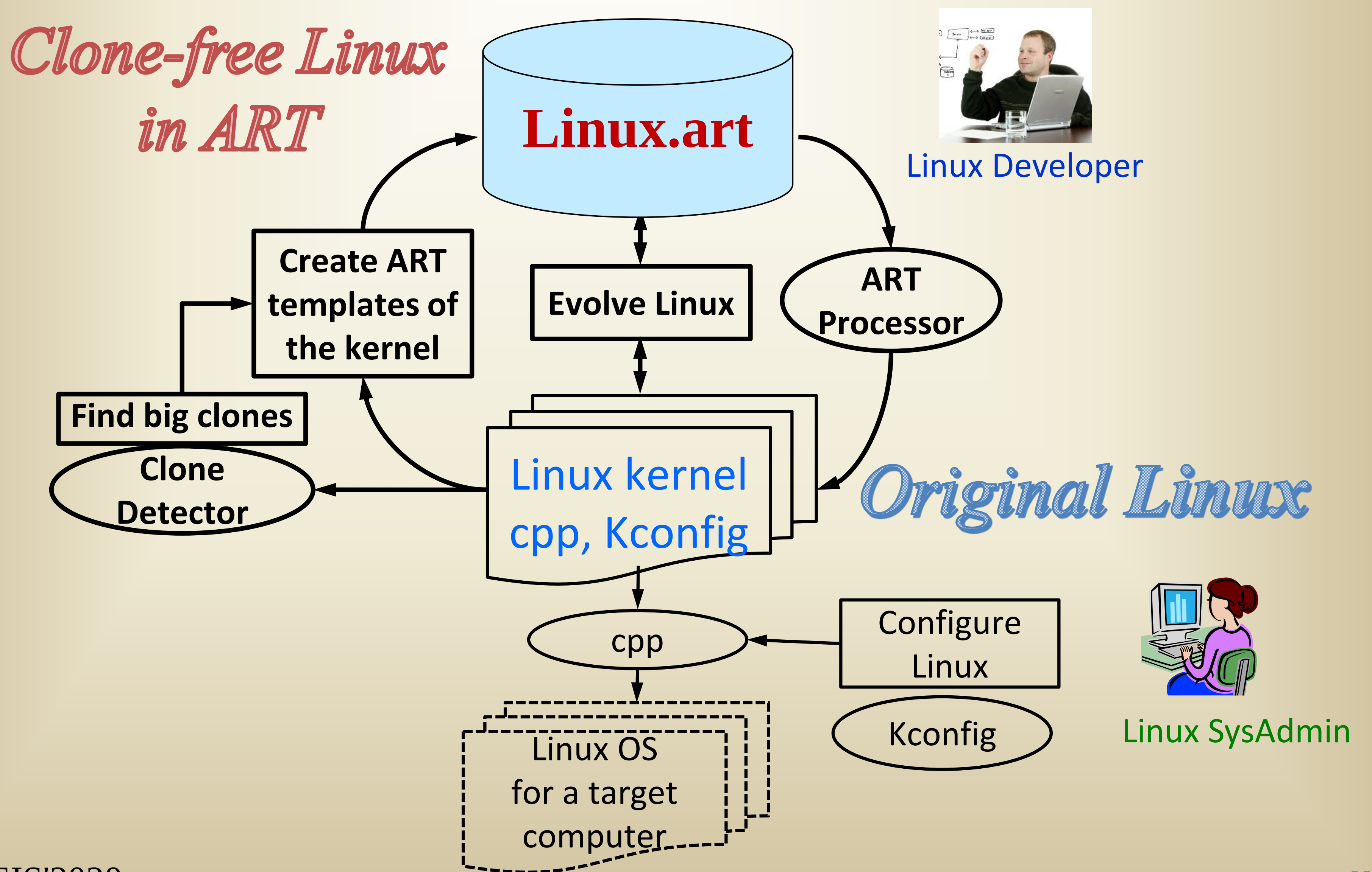
We analyzed 19,627 LOC of Linux kernel

Type of clone		Example
Similar directories		Such as JBD; 6 dirs; 19-46 files per directory
Similar files		Drivers for various brands of touch screen devices
Similar code fragments		Queue handling code for differnt wireless network adapters

Why clones occur?

- Functional similarities among different modules and subsystems
- Over time, existing functionalities have been incrementally enhanced
- New subsystems were implemented by *copy-paste-modify* existing subsystems
- *Lack of a technique to handle such situations in generic way instead of duplicating the code*
- Decentralized development

Linux kernel with ART power-generics

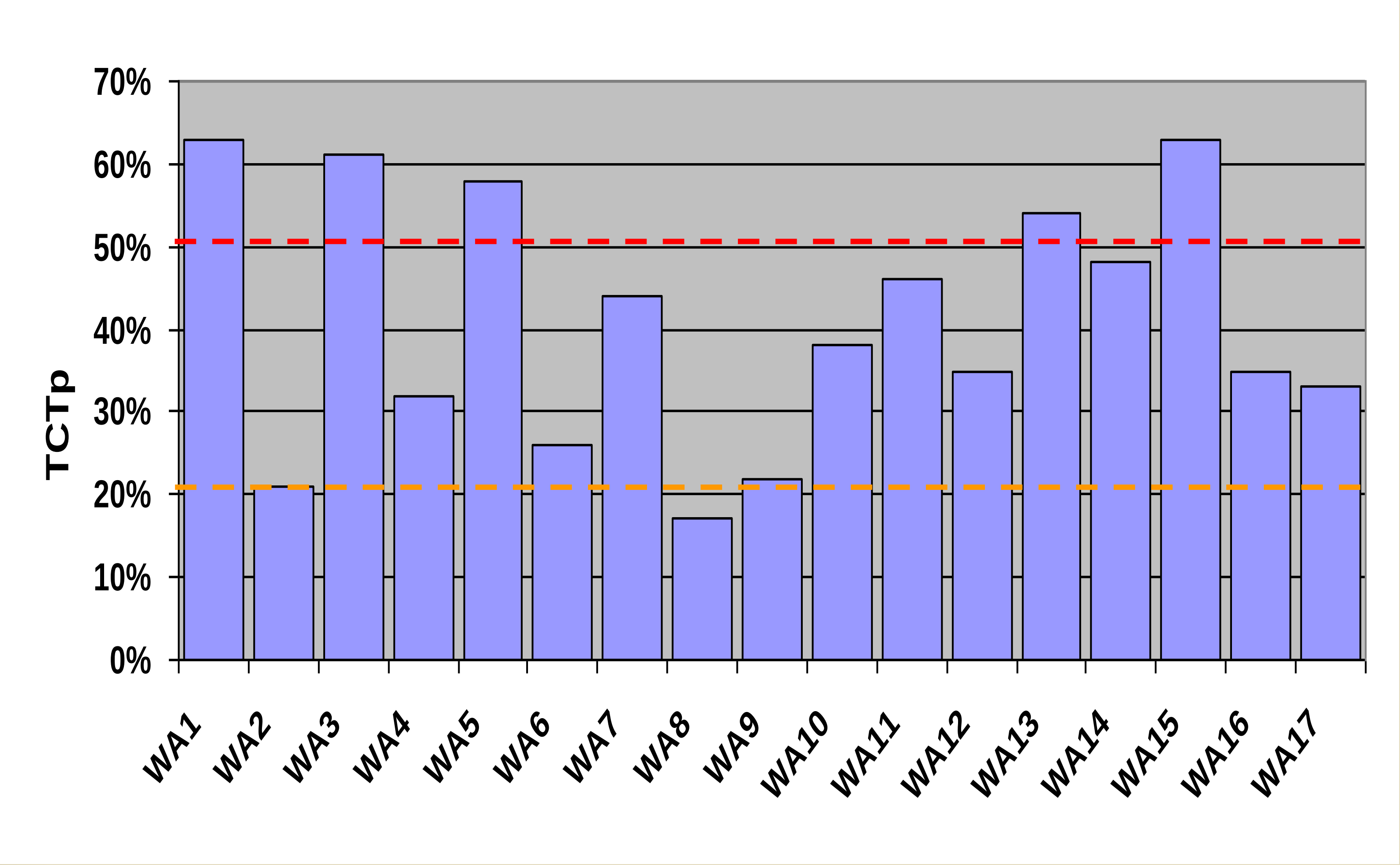


Clones in Web applications

Rajapakse, D.C and Jarzabek, S. “An Investigation of Cloning in Web Portals,” *Int. Conf. on Web Engineering, ICWE’05,* July 2005, Sydney, pp. 252-262

Cloning in 17 Web Applications

% of code is contained in clones



Project by ST Electronics (Info-Software Systems) Pte Ltd Singapore

Web portals for people tracking

in Singapore hospitals during

SARS outbreak in 2003

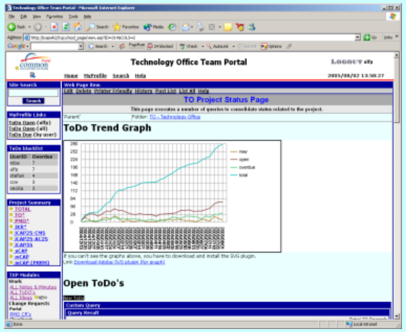
Pettersson, U., and Jarzabek, S. “Industrial Experience with Building a Web Portal Product Line using a Lightweight, Reactive Approach,” *ESEC-FSE'05, European Soft. Eng.Conf. and ACM SIGSOFT Symp.on the Foundations of Soft. Eng*, ACM Press, Sept. 2005, Lisbon, pp. 326-335

Project Collaboration Portals, PCP

Conventional Development

Personal Portal
(home)

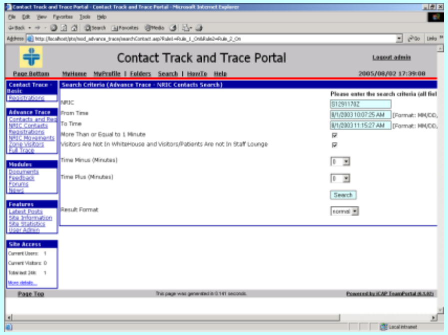
1



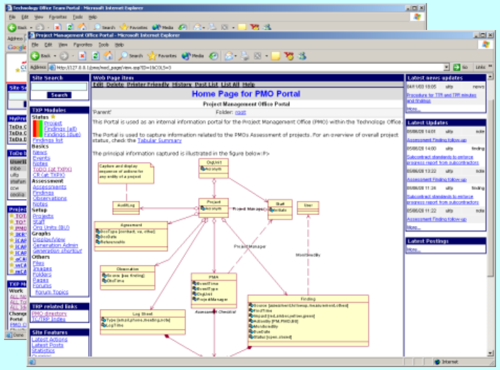
First PCP
(office)

2

3



People Tracking Portal SPL
(business product - SARS)

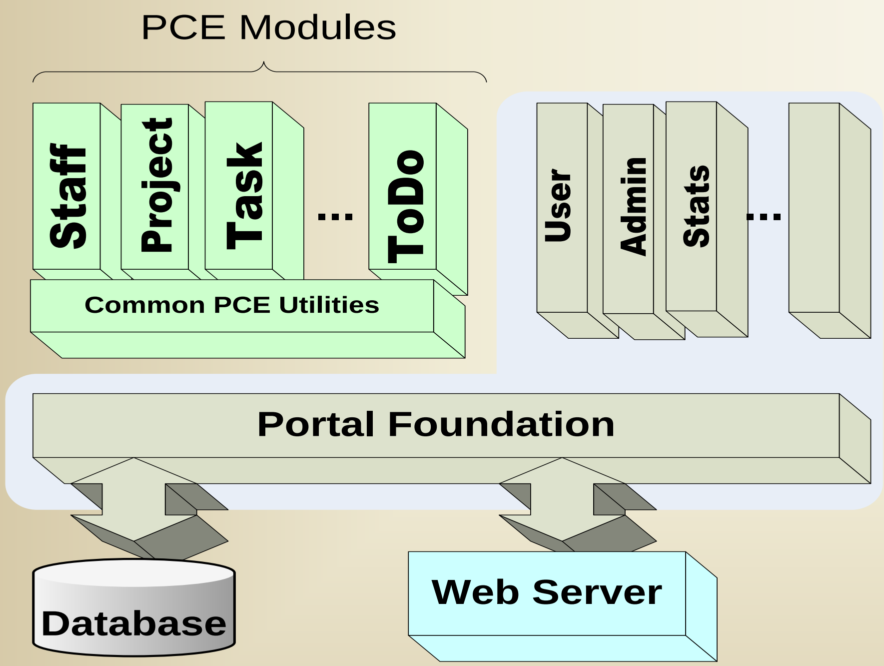


PCP SPL (business product)

SPL
in ART

Project Collaboration Portal (PCP)

PCP supports project teams in software development

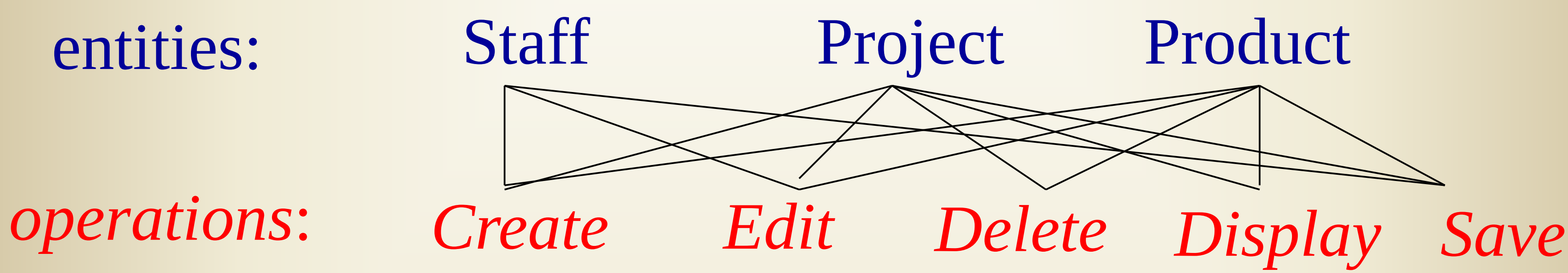


Stores staff, project data, monitors progress of work, supports communication in the team, etc.

- PCPs for Small or Big teams, Agile or Waterfall process, etc.

Patterns in PCP

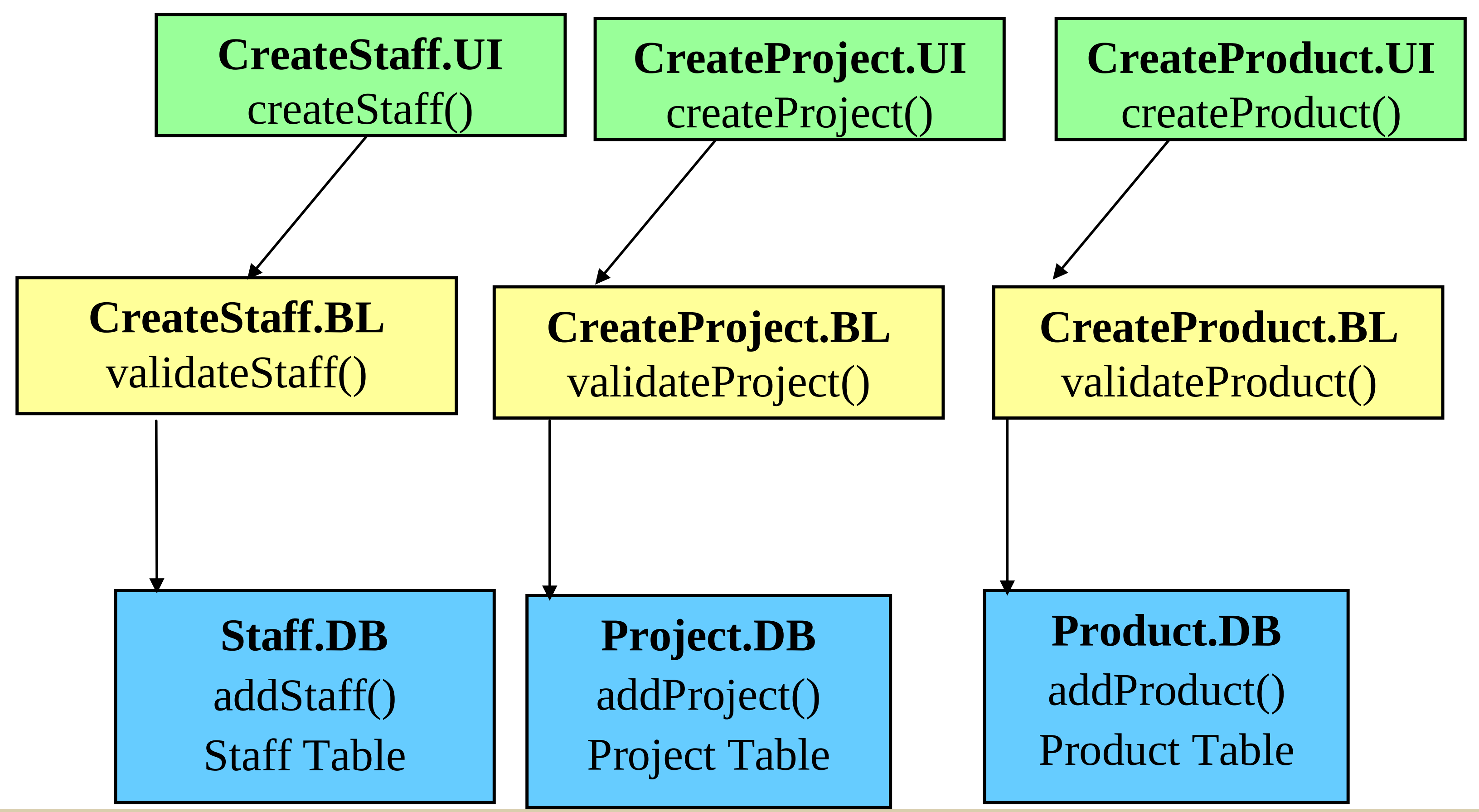
- PCP involves entities and operations



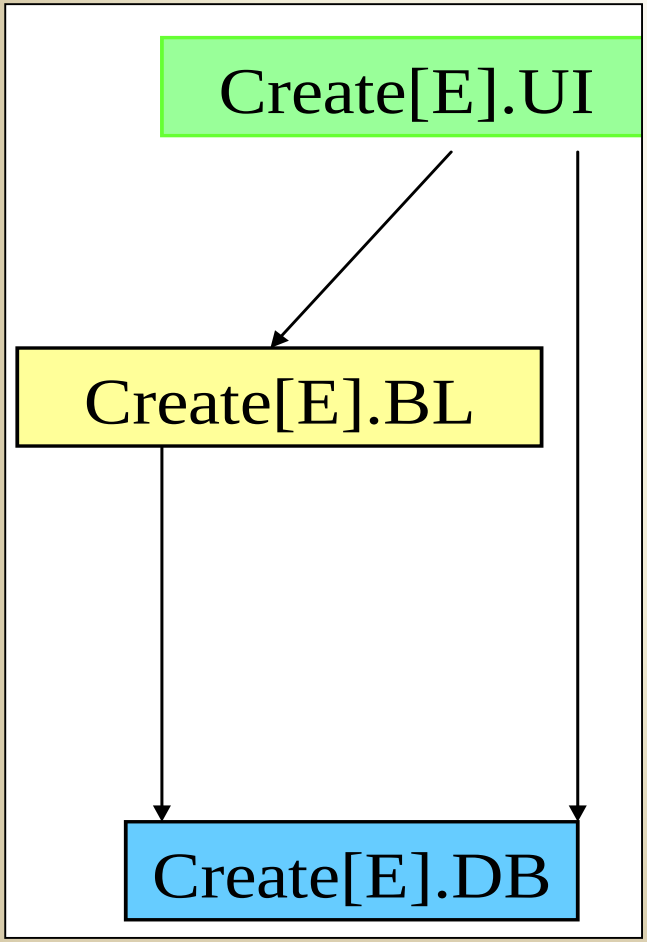
- PCP modules implement operations for entities
- Operations *Create* for different entities are similar but also different
- Less similarity in operations for Staff

Collaborative structural clones

CreateStaff CreateProject CreateProduct



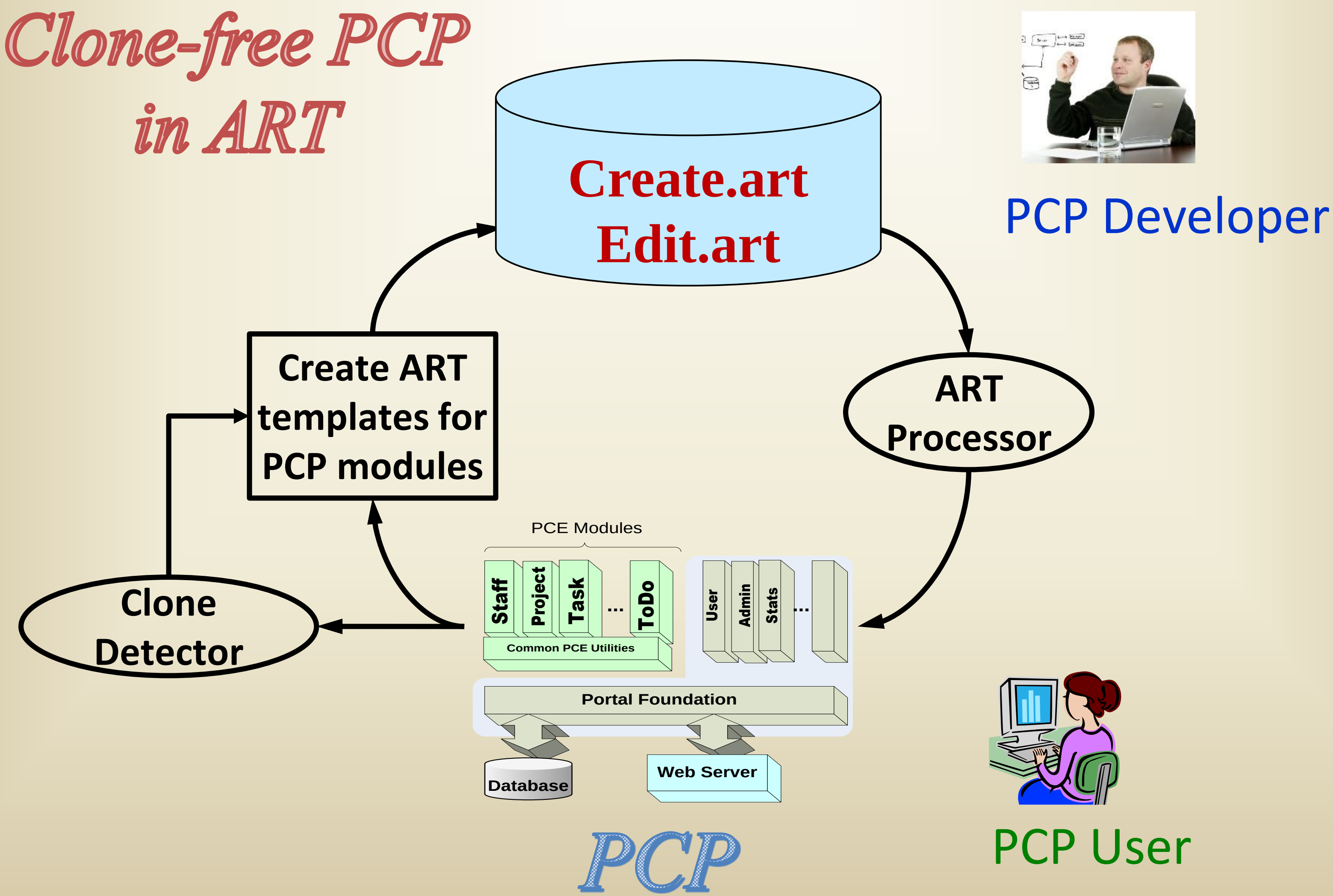
Power-generic *Create*



Power generic **Create[E].art** in ART

E = Staff, Project, Task,…

PCP factory in power-generics



Experiences from this project:

- STE has built and maintains over 100 different portals
  - *based on ART-enabled Software Product Line*
- Short time (less than 2 weeks) and small effort (2 persons) to start seeing the benefits
- Effort to build new portals with ART
  - ***60%*** **-** *90%* ***reduction of code*** *needed to build a new portal*
  - *estimated* ***eight-fold*** *reduction of effort*
- Effort to maintain already released portals
  - *for the for first* ***nine portals****, managed* ***code lines was 22% less*** *than the original* ***single portal***

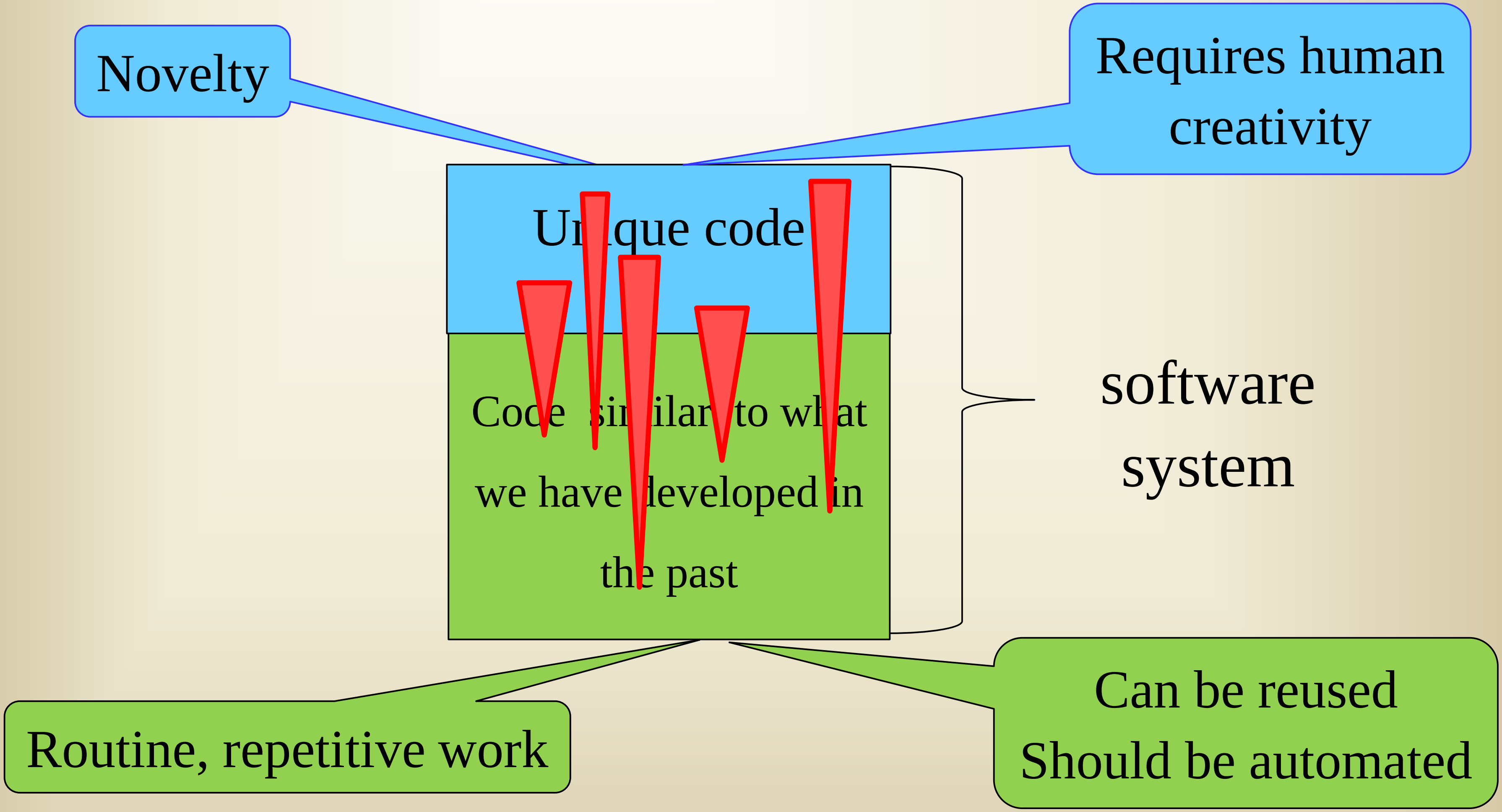
Research on software clones

- More than two decades of research on clones
- Enabling technology for migration to SPL
- Sessions on clones at top SE conferences: ESEC-FSE, ICSE, ICSME
- Int. Workshop on Software Clones IWSC since 2009
- SE journals TSE, TOSEM often publish clone research
- Surveys on software clones research, detection, visualization

Software reuse

- McIlroy M.D., 1968: *Mass Produced Software Components*
- *Premise*: There is much repetition in software processes and products
- What is repetitive we can reuse and automate

Software reuse, automation, productivity



Adaptable

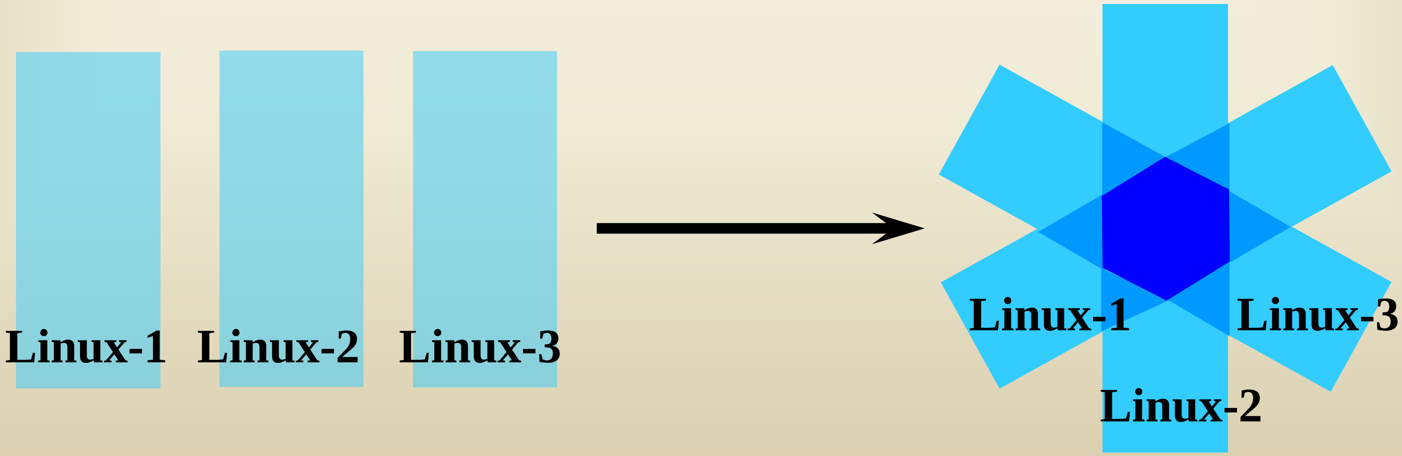
Components ⟶ Reuse ⟶ Automation ⟶ Productivity

Software Product Line (SPL)

1. A family of similar but also different software products
   - *Satisfy needs of a particular customer group*
   - *Share **common features** and differ in **variant**, customer-specific **features***
2. Products are managed from a shared architectuire and common base of *adaptable, reusable* **SPL components**

Software Product Lines SPL

- SPL: Transition from “one-of-the-kind” to reuse-based, automated development
- A key to SPL is to understand and take under control software similarities and differences



Conclusions

- Clones were, clones are, clones will be
- Similarities are opportunity for productivity improvements
- The amount of cloning depends on application
  - *Watch for "feature combinatorics"*
- Challenge - Integrate generative capabilities:
  - *into programming languages*
  - *into component- and architecture-centric software reuse*

Q & A