Software Similarities and Clones

A Curse or Blessing?

Stan Jarzabek
Bialystok University of Technology
Clone-related terms

- Repetitions
- Duplications
- Software redundancies
- Similarity patterns
- Analysis patterns
- Design patterns

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?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Not much</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A lot</td>
</tr>
</tbody>
</table>

• 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

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

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

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

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

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

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

   if (d == '-') return;
   d = getc(finput); }
}
```

```c
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 than 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

ICEIS'2020
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
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


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

• Copy-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

<table>
<thead>
<tr>
<th>Percentage of Repetitions</th>
<th>Observed in:</th>
</tr>
</thead>
<tbody>
<tr>
<td>20% – 50%</td>
<td>Reengineering projects</td>
</tr>
<tr>
<td>68%</td>
<td>Java Buffer library</td>
</tr>
<tr>
<td>40% - 60%</td>
<td>STL in C++: containers and some functions</td>
</tr>
<tr>
<td>68% 50%</td>
<td>C# .NET; J2EE, command and control appl.</td>
</tr>
<tr>
<td>17% - 63%</td>
<td>17 Web Applications (simple, exact clones)</td>
</tr>
<tr>
<td>60% - 90%</td>
<td>Web portals ASP (simple + structural) clones</td>
</tr>
<tr>
<td>80% - 95%</td>
<td>Business applications in COBOL (reuse with frame technology)</td>
</tr>
</tbody>
</table>
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?

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`  \textit{Stack} < \textit{T} >
Associative containers

• Variable-sized containers supporting efficient retrieval of elements via keys

• Features of associative containers:
# Feature combinations

<table>
<thead>
<tr>
<th>Class</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Storage</td>
</tr>
<tr>
<td>Set</td>
<td>sorted</td>
</tr>
<tr>
<td>Multiset</td>
<td>sorted</td>
</tr>
<tr>
<td>Map</td>
<td>sorted</td>
</tr>
<tr>
<td>Multimap</td>
<td>sorted</td>
</tr>
<tr>
<td>Hash Set</td>
<td>hashed</td>
</tr>
<tr>
<td>Hash Multiset</td>
<td>hashed</td>
</tr>
<tr>
<td>Hash Map</td>
<td>hashed</td>
</tr>
<tr>
<td>Hash Multi</td>
<td>hashed</td>
</tr>
</tbody>
</table>
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

```cpp
_Self& operator++() {
    --current;
    return *this;
}

_OPERATOR доходе(int) {
    _Self __tmp = *this;
    --current;
    return __tmp;
}

_Self& operator--() {
    ++current;
    return *this;
}

_OPERATOR доходе(int) {
    _Self __tmp = *this;
    ++current;
    return __tmp;
}
```
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;
}
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

<table>
<thead>
<tr>
<th>templates</th>
<th>cloned code</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 assoc. containers</td>
<td>50% of cloned code, <em>can be unified by 2 power-generics</em></td>
</tr>
<tr>
<td>stack and queue</td>
<td>40% of cloned code</td>
</tr>
<tr>
<td>set operations: union, intersection, etc.</td>
<td>50% of cloned code, <em>can be unified by one generic operation</em></td>
</tr>
</tbody>
</table>


Differences between clones

• Different names for cloned classes and methods
• Different return types for cloned methods
• Extra parameters in cloned templates
• Extra typedefs in cloned code
• Type variations in cloned typedefs
• 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*

![Diagram](image)

- Map.art, Set.art
- Create ART power-generics
- ART Processor
- Clone Detector
- Original STL

ICEIS'2020
Clones in Buffer Library

JDK


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

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

```java
class IntBuffer {
    int hb;
    public final boolean hasArray() {
        return (hb != null) && !isReadOnly ; }
}
```
Inheritance: example 1

```
A
  X
  Y
  Z

foo()
```

undo foo()
Inheritance: example 2

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

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

protected int doSomething(Double op1, Double op2) {
    Double retval = doSomethingElse(op1, op2);
    print(retval);
    return retval;
}
```
Generics-unfriendly variations

```java
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());
    }

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

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

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

    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

Heap[T]Buffer

Direct[T]Buffer[S|U]

ByteBufferAs[T]Buffer[B|L]

Heap[T]BufferR

Direct[T]BufferR[S|U]

ByteBufferAs[T]BufferR[B|L]

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

R: read-only

byte ordering: S – non-native U – native

byte ordering: B – BigEndian L – LittleEndiand
Clone-free Buffer lib with power-generics

Power-generics Buffer lib

[T]Buf.art Heap[T]Buf.art

Create ART power-generics for Buffer classes

Clone Detector

IntBuffer CharBuffer

ART Processor

Buffer-lib Developer

Buffer-lib User

Original Buffer lib
The extent of cloning in Buffer classes

LOC comparison

- Original: 100%
- power-generics: 32%
- Generics: 73%

ICEIS'2020
Evaluation

Clone-free is smaller - but is it better?
Maintenance effort: clone-free vs. the original Buffer lib

- suppose we add Complex buffer

<table>
<thead>
<tr>
<th>New classes</th>
<th>Changes in original Buffer library</th>
<th>Changes in clone-free lib</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No.</td>
<td>type</td>
</tr>
<tr>
<td>ComplexBuffer</td>
<td>25</td>
<td>automatic</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>manual</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>manual</td>
</tr>
<tr>
<td>HeapComplexBuffer</td>
<td>21</td>
<td>automatic</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>manual</td>
</tr>
<tr>
<td>HeapComplexBufferR</td>
<td>16</td>
<td>automatic</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>manual</td>
</tr>
</tbody>
</table>
Big Clones in Linux kernel

Linux OS runs on hundreds of computer brands, with varying architectures, processors

Linux OS SPL

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

![Diagram](image)

**Develops & evolves Linux kernel**

**Builds OS by customizing Linux kernel**

**Linux Expert**

**Linux SysAdmin**

**ICEIS'2020**
Cloned directories in Linux kernel

• Journaling Block Device (JBD):
**Sample cloned code in jbd files**

### Identical Code Fragments: ~554 LOC

```c
51: static inline void __buffer_unlink(struct journal_head *jh)
52: {
53: transaction_t *transaction = ...
54: __jbd2_update_log_tail(journal, first_tid, blocknr);
55: _update_sb_log_tail(journal, first_tid, blocknr);
56: ...
57: }
58: }
59: return
60: }
61: }
```

### Code Fragments with Parametric Changes: ~47 LOC

```c
128: while (__log_space_left(journal) < nb blocks) {
129: if (journal->j_flags & JFS_ABORT)
130: return;
131: spin_unlock(&journal->j_list_lock);
132: mutex_lock(&journal->j_checkpoint_mutex);
133: ...
134: }
```

### Code Modification: ~12 LOC

```c
333: set_buffer_iwrite(bh);
334: bhs[*batch_count] = bh;
335: __buffer_relink_io(jh);
336: jbd_unlock_bh_state(bh);
337: (*batch_count)++;
338: if (*batch_count == NR_BATCH) {
339: spin_unlock(&journal->j_list_lock);
340: __flush_batch(journal, bhs, batch_count);
341: }
342: }
343: ...
344: }
345: return;
346: }
347: write_unlock(&journal->j_state_lock);
348: mutex_lock(&journal->j_checkpoint_mutex);
349: }
```

```c
311: journal->j_chkpt_bhs[*batch_count] = bh;
312: __buffer_relink_io(jh);
313: transaction->t_chp_stats.cs_written++;
314: (*batch_count)++;
315: if (*batch_count == JBD2_NR_BATCH) {
316: spin_unlock(&journal->j_list_lock);
317: __flush_batch(journal, batch_count);
318: }
```
Clones in JBD

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

We analyzed 19,627 LOC of Linux kernel

<table>
<thead>
<tr>
<th>Type of clone</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Similar directories</td>
<td>Such as JBD; 6 dirs; 19-46 files per directory</td>
</tr>
<tr>
<td>Similar files</td>
<td>Drivers for various brands of touch screen devices</td>
</tr>
<tr>
<td>Similar code fragments</td>
<td>Queue handling code for different wireless network adapters</td>
</tr>
</tbody>
</table>
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
Clone-free Linux in ART

Clone Detector
Find big clones

Create ART templates of the kernel

Linux.art

Evolve Linux

ART Processor

Linux kernel cpp, Kconfig

cpp

Configure Linux

Kconfig

Linux OS for a target computer

ICEIS'2020
Clones in Web applications

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

Project Collaboration Portals, PCP

Conventional Development

1. People Tracking Portal SPL (business product - SARS)
2. PCP SPL (business product)
3. First PCP (office)

Personal Portal (home)

ICEIS’2020
Project Collaboration Portal (PCP)

PCP supports project teams in software development

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

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

- PCE Modules
  - Staff
  - Project
  - Task
  - ... ToDo

- Portal Foundation
  - User
  - Admin
  - Stats
  - ...
Patterns in PCP

• PCP involves entities and operations

entities: Staff Project Product

operations: Create Edit Delete Display Save

• 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

CreateStaff.BL
validateStaff()

CreateStaff.UI
createStaff()

Staff.DB
addStaff()
Staff Table

CreateProject.BL
validateProject()

CreateProject.UI
createProject()

Project.DB
addProject()
Project Table

CreateProduct.BL
validateProduct()

CreateProduct.UI
createProduct()

Product.DB
addProduct()
Product Table
Power-generic Create

Power generic Create[E].art in ART
E = Staff, Project, Task,…
PCP factory in power-generics

Clone-free PCP in ART

Create ART templates for PCP modules

Create.art
Edit.art

Clone Detector

ART Processor

PCP Developer

PCP User

PCE Modules

Portal Foundation

Staff
Project
Task
ToDo
User
Admin
Stats

Database
Web Server

ICEIS'2020
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 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

Novelty

Requires human creativity

Unique code

Code similar to what we have developed in the past

Routine, repetitive work

Can be reused
Should be automated

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 architecture 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