Source Code Rejuvenation is not Refactoring

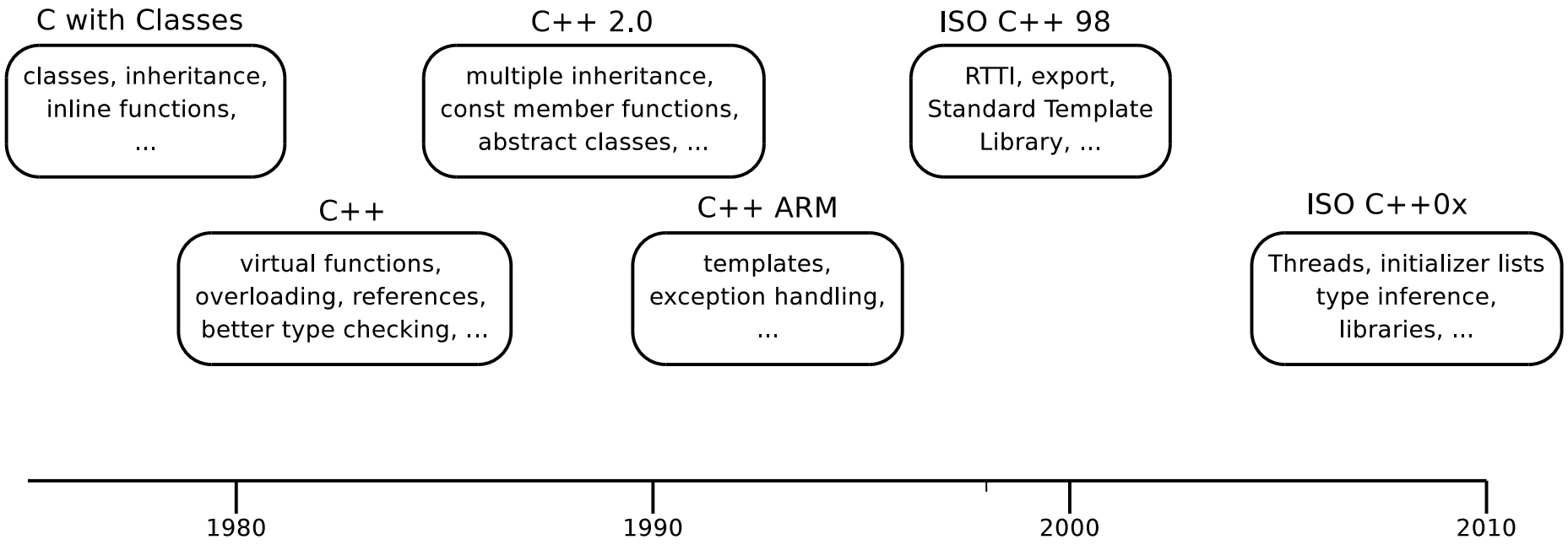
Peter Pirkelbauer Damian Dechev Bjarne Stroustrup

Texas A&M University

SOFSEM 2010

Parasol
Smarter computing.
Texas A&M University

Programming Language Evolution



Evolving a language in and for the real world: C++ 1991-2006. [Str07]

Parasol
Smarter computing.
Texas A&M University

Overview

Parasol
Smarter computing.
Texas A&M University

What is Source Code Rejuvenation?

Definition

- Detection of outdated coding styles and idioms
- Automatic replacement of old code with modern language features or libraries.

Parasol
Smarter computing.
Texas A&M University

Source Code Rejuvenation

Goals

- Preserve or *improve* a program's behavior
- Raise the level of abstraction in source code

Parasol
Smarter computing.
Texas A&M University

Container Initialization in C++0x

C++0x Initializer List

```
// initializes a vector with 3 elements: 1, 2, 3
vector<int> vec = {1, 2, 3};
```

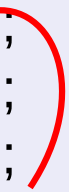
concise, non-redundant code

Parasol
Smarter computing.
Texas A&M University

Container Initialization in C++

Consecutive push_backs

```
vector<int> vec;
vec.push_back(1);
vec.push_back(2);
vec.push_back(3);
```



repetitive code

Copying from an array

```
static const int a[] = {1, 2, 3};
vector<int> vec(a, a+sizeof(a)/sizeof(int));
```

Parasol
Smarter computing.
Texas A&M University

Container Initialization in C++

Consecutive push_backs

```
vector<int> vec;
vec.push_back(1);
vec.push_back(2);
vec.push_back(3);
```

potentially inefficient code
(might trigger resize of the vector)

Copying from an array

```
static const int a[] = {1, 2, 3};
vector<int> vec(a, a+sizeof(a)/sizeof(int));
```

Parasol
Smarter computing.
Texas A&M University

Container Initialization in C++

Consecutive push_backs

```
vector<int> vec;
vec.push_back(1);
vec.push_back(2);
vec.push_back(3);
```

Copying from an array

```
static const int a[] = {1, 2, 3};
vector<int> vec(a, a+sizeof(a)/sizeof(int));
```

redundant type specification

Parasol
Smarter computing.
Texas A&M University

Container Initialization in C++

Consecutive push_backs

```
vector<int> vec;
vec.push_back(1);
vec.push_back(2);
vec.push_back(3);
```

Copying from an array

```
static const int a[] = {1, 2, 3};
vector<int> vec(a, a+sizeof(a)/sizeof(int));
```

cumbersome range specification

Parasol
Smarter computing.
Texas A&M University

Source Code Rejuvenation

Goals

- Preserve or *improve* a program's behavior
- Raise the level of abstraction in source code

Parasol
Smarter computing.
Texas A&M University

C++ Concepts - A Quick Introduction

Unconstrained code

```
template <class Iterator>
Iterator random_elem(Iterator first, Iterator last) {
   typename Iterator::difference len = distance(first, last);
   advance(first, rand()%len);
   return first;
}
```

C++ concepts specify syntactic and semantic properties

```
concept RandomElemIter <typename Iterator> {
   Iterator::Iterator(const Iterator&); // copy constructor
   typename Iterator::difference; // associated type
   Iterator::distance distance(Iterator&, Iterator&);
   void advance(Iterator&, Iterator::distance&);
   ...
}

   template<RandomElemIter Iterator>
   Iterator random_elem(Iterator first, Iterator last);
```

Parasol
Smarter computing.
Texas A&M University

C++ Concepts - A Quick Introduction

Unconstrained code

```
template <class Iterator>
Iterator random_elem(Iterator first, Iterator last) {
   typename Iterator::difference len = distance(first, last);
   advance(first, rand()%len);
   return first;
}
```

C++ concepts specify syntactic and semantic properties

```
concept RandomElemIter <typename Iterator> {
   Iterator::Iterator(const Iterator&); // copy constructor
   typename Iterator::difference; // associated type
   Iterator::distance distance(Iterator&, Iterator&);
   void advance(Iterator&, Iterator::distance&);
   ...
}
```

```
template<RandomElemIter Iterator>
Iterator random_elem(Iterator first, Iterator last);
```

Parasol
Smarter computing.
Texas A&M University

C++ Concepts - A Quick Introduction

Unconstrained code

```
template <class Iterator>
Iterator random_elem(Iterator first, Iterator last) {
  typename Iterator::difference len = distance(first, last);
  advance(first, rand()%len);
  return first;
}
```

C++ concepts specify syntactic and semantic properties

```
concept RandomElemIter <typename Iterator> {
  Iterator::Iterator(const Iterator&); // copy constructor
  typename Iterator::difference; // associated type
  Iterator::distance distance(Iterator&, Iterator&);
  void advance(Iterator&, Iterator::distance&);
  ...
}
```

```
template<RandomElemIter Iterator>
Iterator random_elem(Iterator first, Iterator last);
```

Parasol
Smarter computing.
Texas A&M University

C++ Concepts - A Quick Introduction

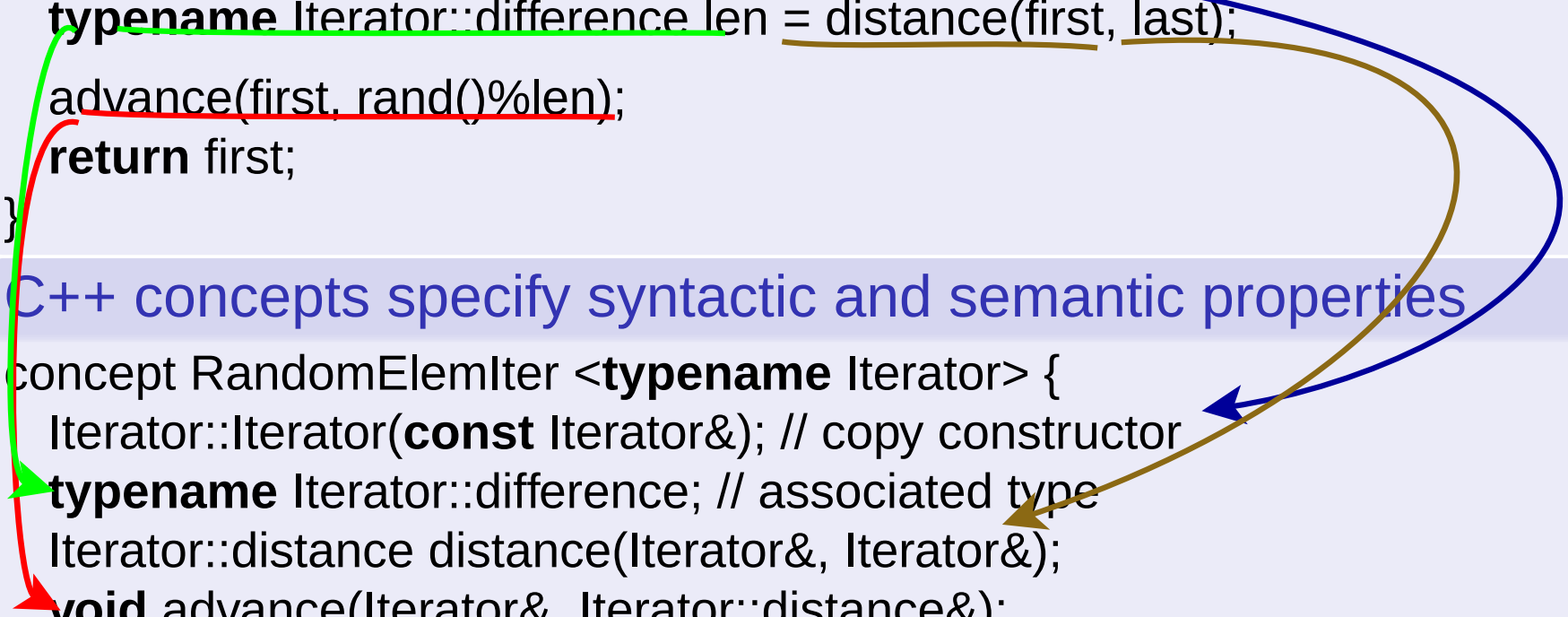
Unconstrained code

```
template <class Iterator>
Iterator random_elem(Iterator first, Iterator last) {
  typename Iterator::difference len = distance(first, last);
  advance(first, rand()%len);
  return first;
}
```

C++ concepts specify syntactic and semantic properties

```
concept RandomElemIter <typename Iterator> {
  Iterator::Iterator(const Iterator&); // copy constructor
  typename Iterator::difference; // associated type
  Iterator::distance distance(Iterator&, Iterator&);
  void advance(Iterator&, Iterator::distance&);
  ...
}
```

```
template<RandomElemIter Iterator>
Iterator random_elem(Iterator first, Iterator last);
```



Parasol
Smarter computing.
Texas A&M University

C++ Concepts - A Quick Introduction

Unconstrained code

```
template <class Iterator>
Iterator random_elem(Iterator first, Iterator last) {
  typename Iterator::difference len = distance(first, last);
  advance(first, rand()%len);
  return first;
}
```

C++ concepts specify syntactic and semantic properties

```
concept RandomElemIter <typename Iterator> {
  Iterator::Iterator(const Iterator&); // copy constructor
  typename Iterator::difference; // associated type
  Iterator::distance distance(Iterator&, Iterator&);
  void advance(Iterator&, Iterator::distance&);
  ...
}
```

```
template<RandomElemIter Iterator>
Iterator random_elem(Iterator first, Iterator last);
```

Parasol
Smarter computing.
Texas A&M University

Concept Recovery

advance — Input iterator

```
template<class Iter, class Dist>
void advance(Iter& iterator, Dist dist, input_iterator_tag) {
  while (dist--) ++iterator;
}
```

advance — Random access iterator

Disjoint sets of constraints

```
template<class Iter, class Dist>
void advance(Iter& iterator, Dist dist, random_access_iterator_tag) {
  iterator += dist;
}
```

Propagating requirements into random_elem

- Which advance exhibits minimal requirements?

Smarter computing.
Texas A&M University

Concept Recovery - w/ workaround analysis

advance — Input iterator

```
template<class Iter, class Dist>
void advance(Iter& iterator, Dist dist, input_iterator_tag) {
  while (dist--) ++iterator;
}
```

Workaround techniques provide semantic information

advance — Random access iterator

```
template<class Iter, class Dist>
void advance(Iter& iterator, Dist dist, random_access_iterator_tag) {
  iterator += dist;
}
```

Propagating requirements into random_elem

- Which advance exhibits minimal requirements?

Smarter computing.
Texas A&M University

Source Code Rejuvenation - Summary

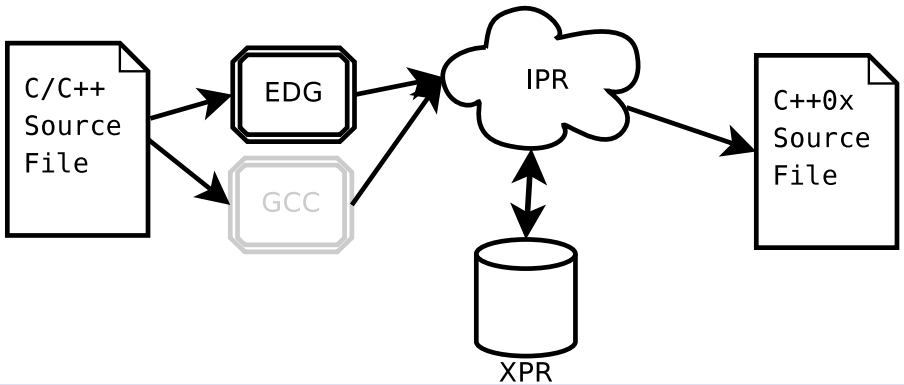
Key Points

- Raises level of abstractions (lowers software entropy)
- Modern code is more concise (easier to read and maintain)

- Code rejuvenation can improve program behavior
- Code rejuvenation is directed

Parasol
Smarter computing.
Texas A&M University

Tool support for Rejuvenation to C++0x - Pivot



The Pivot

- Industrial C++ Frontend
- Intermediate representation (IPR)
  - frontend independent
  - preserves high level details
  - ready for most C++0x features
- eXternal representation (XPR)
  - human read- and writable
  - 1:1 correspondence to IPR

What is refactoring?

Re ◇ factoring

- Factor multiple reoccurring code into a single function

Broader Definition

- An automatic and behavior preserving code transformation that improves source code that was subject to gradual structural deterioration over its life time. [OJ93]

Broader Definition

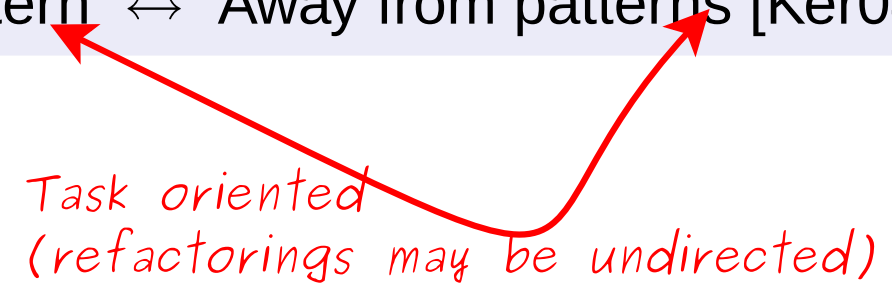
- Improves the design of existing code [FBB+99]

Parasol
Smarter computing.
Texas A&M University

Refactoring Examples

Repeated (user driven) maintenance tasks

- Class specification $\leftrightarrow$ Class generalization [Opd92]
- Towards pattern $\leftrightarrow$ Away from patterns [Ker04]



Parasol
Smarter computing.
Texas A&M University

Refactoring Examples (cont'd)

Elimination of bad code

- Code smells
- Anti-pattern

Parasol
Smarter computing.
Texas A&M University

Refactoring versus Rejuvenation

	Source Code Rejuvenation	**Refactoring**
Transformation	Source-to-source	Source-to-source
Behavior preserving	Behavior *improving*	Behavior preserving
Directed	yes Raises the level of abstraction	no
Drivers	Language / library evolution	Feature extensions Design changes
Indicators	Workaround techniques / idioms	Code smells Anti-patterns
Applications	One-time source code migration	Recurring maintenance tasks

Parasol
Smarter computing.
Texas A&M University

Thank You!

Parasol
Smarter computing.
Texas A&M University


Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts.

Refactoring: improving the design of existing code.
Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.


Joshua Kerievsky.

Refactoring to Patterns.
Pearson Higher Education, 2004.


William F. Opdyke and Ralph E. Johnson.

Creating abstract superclasses by refactoring.
In *CSC '93: Proceedings of the 1993 ACM conference on Computer science*, pages 66–73, New York, NY, USA, 1993. ACM.


William F. Opdyke.

Refactoring object-oriented frameworks.
PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1992.
UMI Order No. GAX93-05645.


Bjarne Stroustrup.

Evolving a language in and for the real world: C++ 1991-2006.
In *HOPL III: Proceedings of the third ACM SIGPLAN conference on History of programming languages*, pages 4–1, New York, NY, USA, 2007. ACM.

Parasol
Smarter computing.
Texas A&M University