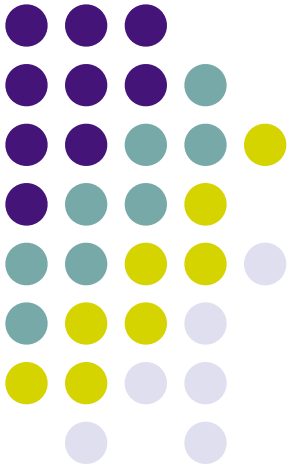
An Empirical Study of Function Overloading in C++

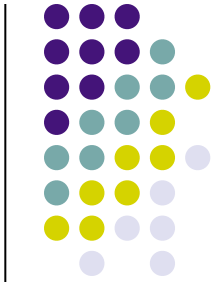
Daqing Hou

Electrical & Computer Engineering Dept.
Clarkson University
Potsdam, NY, USA

joint work with Cheng Wang

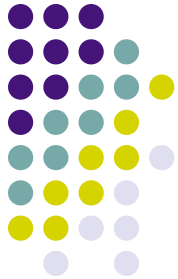


Motivation



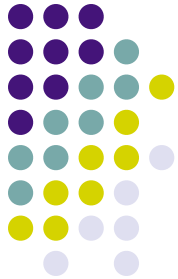
- Programmers are dealing with an increasingly rich set of tools and features in daily programming.
- To fully master these tools, in addition to learn what they can do individually, it is necessary to develop a methodology that provides the “big picture” view.
- Such a methodology should include an account of design rationale for tools and features, typical patterns of use, and usage guidelines and principles.
- This study tries to do this for C++ function overloading.

Example of Overloading



```
class Y;
class X {
public:
    operator char() const;

    void foo(int);                  //f1
    void foo(char);                 //f2
    void foo(double);               //f3
    void foo(X);                    //f4
    void foo(Y&);                   //f5
};

class Y: public X {};

void foo(double);                   //f6
void foo(int);                      //f7
```

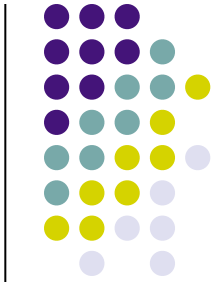
Example of Overloading



```
class Y;
class X {
public:
    operator char() const;

    void foo(int);                    //f1
    void foo(char);                   //f2
    void foo(double);                 //f3
    void foo(X);                      //f4
    void foo(Y&);                     //f5
};

class Y: public X {};

void foo(double);                     //f6
void foo(int);                        //f7
```

```
void bar(Y &aY)
{
    foo('c');          //C={f6,f7},V={f6,f7}, Best f7
    foo(aY);           //C={f6,f7},V={f6,f7}, Best f7
    aY.foo('a');       //C={f1…f5},V={f1,f2,f3}, Best f2
    aY.foo(aY);        //C={f1…f5},V={f1…f5}, Best f5
}
```

Type Conversion Rules for C++ Function Overloading



- Exact match
  - L-value to R-value conversion
  - Array-to-Pointer conversion
  - Function-to-Pointer conversion
  - Qualification conversion
- Promotion
- Standard conversion
- User-defined conversion
- Ellipsis

Format of Output Data

- Definition of Overloaded Functions

Function_Name: X::foo Definition_File: example.cpp Overload Times: 5

Function_Name: X::X Definition_File: example.cpp Overload Times: 2

Function_Name: ::foo Definition_File: example.cpp Overload Times: 2

- Calls of Overloaded Functions

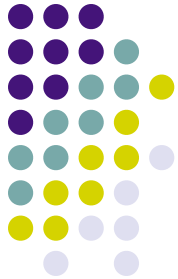
::foo #2 #2 #<2|3|char--int> #/Users/Wangc/Work/Test/example.cpp:20 #/Users/Wangc/Work/Test/example.cpp:17

::foo #2 #2 #<5|3:8|Y--int> #/Users/Wangc/Work/Test/example.cpp:22 #/Users/Wangc/Work/Test/example.cpp:17

X::foo #5 #3 #<3|4|Y*--X*> <0|0|char--char> #/Users/Wangc/Work/Test/example.cpp:24 #/Users/Wangc/Work/Test/example.cpp:9

X::foo #5 #5 #<3|4|Y*--X*> <0|7|Y--Y*> #/Users/Wangc/Work/Test/example.cpp:26 #/Users/Wangc/Work/Test/example.cpp:12

Case I: Mozilla



- Version 1.8b
- Some Size Metric

# header files	4679	# html files	2246
# cpp files	4442	# xul files	624
# c files	1515	# xml files	325

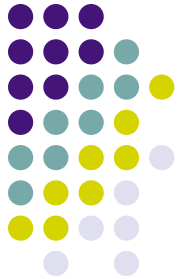
- Classes (5689)

Summary of Findings - Definition of Overloading Functions

- 13,817 names are overloaded. 42% are due to constructors. 47% due to template instantiations (11 names from 6 templates classes in xpcom).
- 6.6% of classes (375/5,869) in Mozilla overload member names.

- 85.6% of these classes overload 3 or less members.
- 92.6% of the 757 overloaded members are overloaded 2 or 3 times. 82.8% only 2 times.

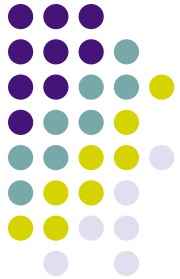
Summary of Findings - Definition of Overloading Functions



- By inspecting portion of 757 overloaded members, overloading is used in string and file operations, graphics, data, layout, db access API, and so on.

- Also found 3 patterns.
  - One is to overload getters and setters to provide different ways of setting and getting object attributes.
  - Another is to overload a core operation with several others that are reduced to the core.
  - Yet another is to provide two ways of retrieving object attributes, via return values and via a parameter, respectively.

Summary of Findings - Size of Candidate/Viable Set for Calls



- 71.8% of 39,012 calls have a viable set of size 1. And 81.6% have 4 or less.
- Calls with a large candidate/viable set are standard operations on string and character, file and stream, most defined in xpcom.
- Only 10 such names are from application modules.

Intra- and Inter- Module Calls (39,012)

	editor	layout	content	parser	network	gfx	widget	view	xpcom
editor	*248*	13	28	1	3		5		**3339**
layout		*1534*	149		14	478	13	9	**2553**
content		76	*1409*	9	60	7	6	1	**5947**
parser				*393*	4				**381**
network					*133*				**2299**
gfx		2				*334*			**496**
widget		1		1	3	12	*36*		**362**
view						27	6	*58*	**43**
xpcom									*3387*

Note

- #intra-module calls and #calls to xpcom dominate (96.6%).
- Only 1,332 (3.4%) inter-module calls.

Distribution of 3,812 Implicit Conversions for 1,332 Inter-module Calls

Conversion		Number
cr_identity		2568
cr_exact		100
cr_promotion		5
cr_std	ck_ctd	48 (int v.s unsigned int) 158 (0 to pointer)
	ck_ptr	238 (225 due to a template in xpcom)
	ck_base	315 (all due to string in xpcom)
cr_user		379 (all due to types defined in xpcom)

Conclusion

- This study is focused on discovering how C++'s function overloading is used in production code using an instrumented g++ compiler.

- Our principal finding for the systems studied (Mozilla and MySQL) is that the most "advanced" subset of function overloading tends to be defined in only a few utility modules, and the majority of application modules use only the "easy" subset of function overloading when overloading names.
- Most overloaded names are used locally within rather than across module interfaces.
- This study also contributes a set of concrete usage examples for C++ function overloading, which would be useful to guide future users in using this feature more effectively.

- Q: Perhaps the set of C++ conversion rules can be subset and controlled by developers rather than by only compilers.