1. Generics

- Basic Theme: Reuse through parameterization
- Common Usage: Type-safe polymorphic containers
- Java Generics: planned inclusion in Java 1.5 (as proposed by JSR-14)
- Our case study shows:
  - situations in which Java generics fail to provide effective generic solutions
  - a meta-level extension of generics for optimal reuse

2. Buffer Library

- Part of java.nio.* package in JDK 1.4.1
- Variability / Feature dimensions in Buffer Library
- Each legal combination of these feature dimensions yields a unique buffer class
  - E.g., DirectIntBufferRS, represents the combination ET = int, AM = read-only, MS = direct, BO = non-native, VB = false
- Many classes in the Buffer library playing essentially the same role!
- Seven groups of similar buffer classes, much redundant code!
- Can we use generics to solve the problem?
  - Only three buffer groups were generics-friendly and could be replaced by 3 generic classes
  - This solution still relies on wrapper classes for primitive types (since parameterization of primitive types is not allowed)
  - only 27% of code could be eliminated from the Buffer library.
  - Many redundancies could not be removed due to various generic-unfriendly situations
  - Following types of generic-unfriendliness were noted
    - Restrictions on type parameters
    - Non-type parametric variations
    - Non-parametric variations
    - Coupling

3. Generics-unfriendly variations in buffer classes

- Only three buffer groups were generics-friendly and could be replaced by 3 generic classes
- This solution still relies on wrapper classes for primitive types (since parameterization of primitive types is not allowed)
- only 27% of code could be eliminated from the Buffer library.
- Many redundancies could not be removed due to various generic-unfriendly situations
- Following types of generic-unfriendliness were noted
  - Restrictions on type parameters
  - Non-type parametric variations
  - Non-parametric variations
  - Coupling

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

```java
public abstract class IntBuffer
extends Buffer implements Comparable, CharSequence
{
    ...
}
```

```java
public abstract class FloatBuffer
extends Buffer implements Comparable, CharSequence
{
    ...
}
```

```java
public abstract class List<T>
{
    ...
}
```

```java
public class IntList
{
    ...
}
```

```java
public class FloatList
{
    ...
}
```

```java
private long ix(int i) {
    return address + (i << 2);
}
```

```java
private long ix(int i) {
    return address + (i << 3);
}
```

```java
public int get(int i) {
    return Bits.swap(unsafe.getInt(ix(checkIndex(i))));
}
```

```java
public int get(int i) {
    return Bits.swap(unsafe.getInt(ix(checkIndex(i))));
}
```

```java
public byte get(int i) {
    return Bits.swap(unsafe.getByte(ix(checkIndex(i))));
}
```

```java
public double get(int i) {
    return Bits.swap(unsafe.getDouble(ix(checkIndex(i))));
}
```

```java
public class ByteOrder
{
    ...
}
```

```java
public class ByteOrder
{
    ...
}
```

```java
public class ByteOrder
{
    ...
}
```

```java
public class ByteOrder
{
    ...
}
```

```java
private long ix(int i) {
    return address + (i << 2);
}
```

```java
private long ix(int i) {
    return address + (i << 3);
}
```

```java
public int get(int i) {
    return Bits.swap(unsafe.getInt(ix(checkIndex(i))));
}
```

```java
public double get(int i) {
    return Bits.swap(unsafe.getDouble(ix(checkIndex(i))));
}
```

```java
public class DoubleBuffer
extends Buffer implements Comparable {
    ...
}
```
4. What is XVCL?
- XVCL is a meta-programming technique and tool for effective reuse
- XVCL uses “composition with adaptation” rules to generate a specific program from generic, reusable meta-components
- XVCL is based on Bassett’s frames, a technology that has achieved substantial gains in industry, therefore, the underlying principles of the XVCL have been thoroughly tested in practice
- unlike original frames, XVCL blends with contemporary programming paradigms and complements other design techniques

5. “Composition with adaptation”
- XVCL adapts meta-components into components of custom programs
- Any location or structure in a meta-component can be a designated variation point, available for adaptation by ancestor meta-components
- an x-framework is a meta-component architecture designed and “normalized” to eliminate redundancies and to enhance reuse
- program generation rules are 100% transparent to a programmer, who can fine-tune and re-generate code without losing prior customizations
- meta-components can evolve as needed without ever forcing retrofits

6. Extending generics using XVCL

7. XVCL solution for Buffer library
- composition via <adapt>
- insert code at break points
- select code among many given options
- code generation (loops)
- generic names: meta-variables and meta-expressions

8. Comparison of results

Other applications of XVCL achieved code reductions of:
- 68% in n-tier application (C#)
- 61% in MS ADO wrapper (Java)
- 59% in CORBA Activation IDL library (Java)

We envision many other applications of XVCL in software and non-software domains.