Component-Based Software

David S. Rosenblum
ICS 221
Winter 2001

Components and Reuse
- Develop systems of components of a reasonable size and reuse them
- Further use or repeated use of a component
- Adapting components for use outside their original context
- Extend the idea beyond code to other development artifacts

Goals of Reuse
- Goals of reuse are primarily economic
  - Save cost/time/effort of redundant work, increase productivity
  - Decrease time to market
  - Improve systems by reusing both the artifact and the underlying engineering experience
- Economic goals achieved only when units of reuse reach critical mass in size, capability and uniformity

Historical Origins
- Idea originally due to Doug McIlroy “Mass Produced Software Components”, 1968 NATO Conference on Software Engineering
  - Reusable components, component libraries
- Named as a potential “silver bullet” by Fred Brooks (1987)
  - Much research interest in the ‘80s and ‘90s
- Technical and managerial barriers have prevented widespread success
  - This led McIlroy to believe he had been wrong!

From Reuse to Component-Based Development
- The term reuse is a misnomer
  - No other engineering discipline uses the term
  - Systematic design and use of standard components is accepted practice in other engineering disciplines
  - The term will (eventually) become obsolete
- The important ideas behind reuse are centered on the notion of components
  - Design of components for use in multiple contexts
  - Design of families of related components
  - Design of components with standardized packaging

Different Flavors of Components
- (Reusable) Third-Party Software Pieces
- Plug-ins/Add-ins
- Applets
- Frameworks
- Open Systems
- Distributed Object Infrastructures
- Compound Documents
- Legacy Systems
Software Engineering Implications

- Traditional software systems
  - are developed by a single organization
  - undergo a phased development process
  - have a synchronized release schedule
  - have a proprietary design and proprietary component interfaces
  - have a monolithic code base
  - go through a painful evolution

Lifecycle Model of Traditional Systems

Component-Based Development

A Possible Lifecycle Model for Component-Based Software

Another Possible Lifecycle Model

Implications of the Lifecycle Models

- Integrate-Then-Deploy
  - Integration of shrink-wrapped off-the-shelf components
  - System validation carried out prior to system deployment
  - Possibly limited access to component development artifacts
- Deploy-Then-Integrate
  - Integration of “live objects”
  - System Integration = System Deployment
  - Possibly limited opportunity for pre-deployment validation
A Challenge for Component-Based Software: Testing

- Unit testing alone won't cut it
- Nor will static analysis techniques
- New dynamic analysis methods are needed

Another View of the Problem

Old-Style Development
- Single vendor
- White-box artifacts
  - code, specs, test cases, analysis support, docs

Component-Based Development
- Multiple vendors
- Many black-box artifacts
  - code, internal specs, test cases, analysis support

A (Partial) Solution: Component Metadata

- Metadata = “data about data”
  - Abstracted information about component internals and development history
  - Can be accessed via metamethods
  - Component developer supplies metadata
  - Application builder exploits metadata
  - Design time and runtime

Kinds of Metadata for SE Tasks (I)

- Information on customizing the component
  - Component properties
  - Constraints on properties
- Information to integrate the component
  - Interface syntax
    - Java reflection, COM QueryInterface, CORBA DII
  - Generated and consumed events
  - Interface semantics
    - Pre/post conditions and invariants
    - Protocol specs

  Many of these are “traditional” kinds of component metadata

Kinds of Metadata for SE Tasks (II)

- Information to evaluate the component
  - Static and dynamic metrics
    - Cyclomatic complexity
    - Test coverage achieved by developer
  - QoS information
  - Pricing/leasing information
- Information to test and debug the component
  - Exported state machine representation
  - Embedded test suite with coverage information
  - Input/output dependencies at interface
  - Dynamically computed coverage information

Kinds of Metadata for SE Tasks (III)

- Information to analyze the component
  - Summary flow information
  - Control dependencies
  - Data dependencies
  - Graph models
    - Call graph
    - Control-flow graph
- Other information to support software engineering tasks
An Example: Program Slicing

```java
public boolean checkingToSavings(String cAccountCode, String sAccountCode, float amount) {
    BankingAccount checking = new BankingAccount(cAccountCode);
    BankingAccount saving = new BankingAccount(sAccountCode);
    float balance, total;
    checking.open();
    saving.open();
    balance = checking.moveFunds(saving, amount);  // A
    total = balance + additionalFunds;  // B
}
```

Suppose we want backward slice w.r.t. `total` at B
- Do `saving`, `amount`, and/or state of `checking` influence `balance` at A?
- Dependency metadata for `BankingAccount` could tell us!

Implementation Issues:
Metadata Format and Naming

- Need uniform format for text and non-text metadata
  - XML
  - DTDs specify format
- Need uniform way of identifying purpose of metadata to users
  - MIME-like tags describe purpose
  - Example: `analysis/data-dependency` for data flow information
- Who establishes naming scheme?
- How do new metadata get established?

Implementation Issues:
Metadata Addition & Retrieval

- Need uniform way for a component to expose its particular collection of metadata
- Two metamethods
  - QueryMetadata
    - Like `QueryInterface` in COM
  - GetMetadata(tag, parameters)
    - Selects metadata according to "tag"
    - Returns statically-embedded or dynamically computed value
    - Could operate as an iterator for complex piecewise metadata

Metadata and Testing of Distributed Components

- Metadata can be used to aid application of existing testing techniques in distributed object systems
- But how should existing testing techniques be changed for distributed components and distributed object systems (and how can metadata help)?
  - Coverage criteria, reliability models
  - Testing infrastructure
  - Test monitoring and oracles

Conclusion

- Component-based software is the wave of the future
- But there are many software engineering challenges to address
- Metadata may provide a solution