Component-based Software Engineering

References:

1. C.Szyperski, "Component Technology -- What, Where, and How?", ICSE'03 (from keynote talk)
2. E. Weyuker, "Testing Component-based Software: A Cautionary Tale", IEEE Software Sept/Oct 1998
3. J. Voas, "Maintaining Component-based Systems", IEEE Software July/Aug 1998
4. J. Voas, "Certifying Off-the-shelf SW Components", IEEE Computer, June 1998, Vol 31, No 6
5. N. Talbert, "The Cost of COTS", an interview with John McDermid in IEEE Computer, June 1998, Vol 31, No 6

Cf 1. Szyperski

Motivations for Components

- Development time: architectural, design, source code artifacts
- Build time: reusing partial design and implementation fragments
- Deployment time: allows last customization before installation
  - Deployment - act of readying a component for installation in a specific environment
- Evolution: dynamic servicing, upgrading, extension, integration into already deployed systems

What's a SW component?

- A unit of deployment
  - An executable deliverable for a (virtual) machine; executes w/o human intervention
- A unit of versioning and replacement
  - Remains invariant (code and data) as is installed
- May have static dependences, assumptions about environment
  - On platform
  - On other components

Complications

- Naming w/o collisions
- Versioning
  - Need version in the name
  - Side-by-side existence of diff versions of same component sometimes needed
    - Interferes with cross-component integration
    - Varies with degree of coupling

Cf 2. Weyuker

Testing

- Need for testing a component in its new context
  - Ariane 5 disaster
- Reuse of COTS, commercial off-the-shelf components, requires new approaches to testing to avoid this
  - Testing techniques cannot require source code
    - Not available in legacy codes nor off-the shelf comps

Phases of Testing

- Unit test - individual components
- Integration test - integrating individually tested components to test as an entity
- System test - entire system tested as one entity
- Additional - performance test, stress test, reliability tests

Testing Component-based SW

- Difficult to construct test suites
- Testing for reuse
  - Possibility of executing different parts of the component may lead into untested or lightly tested code
  - Even COTS components need retesting in situ
  - Debugging much more difficult w/o developer knowledge
  - W/o source code how to correct defects found?

Components

- In-house
  - Test various uses for component
    - Cannot envision all scenarios
  - Debugging and code modification difficult
  - Validation of quality difficult
  - Rethink repository design to include specs, modifics, test suite with ptrs to corresponding parts of code

- COTS
  - Lack of source code precludes modification for debugging or extension
  - Lack of detailed knowledge of design
  - No control over maintenance or support

Cf 3. Voas

Maintenance

- Longest stage in SW life cycle
- Components to be maintained are essentially black boxes
- Claim that OOPLs and componentn-based SE, turn SW development into SW manufacturing
  - Principle task: design & integration, not coding

Problems Maintaining COTS

- **Frozen functionality - no further vendor support**
  - **Implement yourself; obtain code and modify; get elsewhere**
- **Incompatible upgrades - customization**
  - **Build wrapper around incompat behavior or uninstall component**
- **Trojan horses - covertly malicious component**
  - **Avoidance may be impossible; detection is difficult**

Problems Maintaining COTS

- Unreliable components - no standards for reliability certification
- Wrappers - middleware that limits a components functionality
  - Middleware: SW that joins together, mediates between, or enhances 2 separate SW packages
  - Restricts input or output info
  - Reasonable approach to incompatibility, Trojan horses, dependability problems
  - Not foolproof

Shareware/Freeware

- Often useful, but can be used for malicious purposes
- Licensing restrictions can exist
- If SW in executable format, then like COTS
- If SW is source code, then may need domain knowledge for maintenance

Proprietory Repositories

- Functional structure
  - All source together, all analysis together, all designs together, all tests together
- Information class structure
  - Each component has source, analysis, design, tests stored together (easier to maintain)

Challenges

- Problem: ensuring that modifications are compatible with all clients
  - Control change process
    - Can add access rules about modification, but file locking creates problems for maintaining the applications using components
    - If do not lock, then change merging becomes a problem
  - Use promotion approach - levels of confidence in SW stability, managed by a person
    - Development/maintenance (low), test, release (high)

Voas' Advice

- Avoid using components for small systems
- Keep reqs documentation on each component; do not add too many features
- Use information class repository with promotion
- Allow 2 versions of component to be in repository when necessary for needs of 2 clients

Cf 4. Voas

Certifying Components

- Use of off-the-shelf (OTS) components require developer to know
  - Is component reliable?
  - Will system tolerate the component?
- Key questions:
  - Does component C fit the need?
  - Is the quality of component C high enough?
  - What impact will component C have on the system?
- Composing highly reliable components may not yield a reliable system!

Scenarios

	Fit the need?	High quality?	Positive impact?
1	Yes	Yes	Yes
2	Yes	Yes	No
3	Yes	No	Yes
4	Yes	No	No
5	No	N/A	N/A

How to certify a component?

- **Black-box component testing to ascertain if quality is high enough**
  - **Cannot use white-box approach w/o code**
  - **Use test suite distributed with component**
  - **Problems**
    - **Need an accurate oracle**
    - **May not execute enough of the code (e.g., possible malicious functionality -- Trojan horse)**

How to certify a component?

- **System-level fault injection to see if system can tolerate component failure**
  - **IPA - Interface Propagation Analysis - perturbs the state propagated through component interface**
    - **Can simulate failure of predecessor component**
    - **Modify data randomly**

How to certify a component?

- **Operational system testing to see if system works well with component**
  - Embed the component in the system and can see component failure in place
  - Problem - can take extensive testing to see failure happen
  - Solution - wrap component to limit its actions and modify its functionality
    - Can keep inputs from reaching component
    - Can keep outputs from reaching component clients

Cf 5. Talbert

COTS vs Custom Alternatives

- John McDermid, safety critical systems expert, Univ of York, UK
  - COTS- "standard commercial software developed w/o any particular application in mind"
  - Why use? To save money and allow interoperability and lessen risk
  - Mostly are providing GUIs, O/S, DBs

How to evaluate COTS SW?

- Stability of COTS SW and prior usage
- Need to learn what COTS SW does
  - Extensive testing
- Determine which fcns are safe for client use
- Examine reliability
  - Gather historical data (look at/analyze code if can get it)
- Demonstrate some certainty that SW cannot execute unwanted functionality
  - Use wrappers or code analysis or both

Cost of COTS SW

- Initial acquisition cost
- Keeping up with upgrades
- Problem:
  - What if vendor goes out of business?
    - SW escrow accounts - 3rd party keeps copy of SW to be turned over in event of disaster
  - What to do about bad support?
  - How to ensure safety of code?
    - Need to look at/analyze code which may be difficult
    - May need to deal with unwanted functionality