REUSE ECONOMICS – BARRIERS – TECHNOLOGY

(REUSE – WHAT'S THE USE?)

Will Tracz

Loral Federal Systems Company
Owego, NY

tracz@lfs.loral.com

http://www.sei.cmu.edu/arpa/dssa/dssa-adage/demo/lfs/tracz.html

15 October 1995

Presentation Outline

1. **Is it Use or Re-use?**:
   - **Re-usable Software**: software that was designed for re-use.
   - **Not** limited to code!
2. **Reuse Economics**: Motivator and Inhibitor
   - Survey of SOP
   - Cost Model
   - Metrics – apples and oranges
3. **Reuse Inhibitors**: Non-Technical and Technical
   - Chicken and Egg
   - Pay now or pay later
   - Interoperability (Design for Re-use) & Understandability (Design for Use)
4. **Reuse Technology**: (no) $urprises
5. **Q/A** – Fire at Will!

Reuse State of the Practice

1. **Reuse Comes of Age**[1]
    - Study by QSM Assoc.Inc. of Pittsfield, MA
        - 15 SW development projects at 9 companies
        - 84% reduction in programming costs
        - 70% reduction in schedule
        - 25% to 30% more time to produce good reusable code.
2. **Economics of Software Reuse**[2]
    - Initial construction of reusable resources is about:
        - 50% more expensive
        - 30% longer schedule
3. **Reusable Class Libraries** [3]
    - 1000 reusable classes
    - each reusable class written 3 times
    - reuse rate of 50% – 70% depending on product line
    - 90% of reusable classes produced by "parts" center
    - 3 year adoption time

[1] Info. Week 11/14/94
[2] Capers Jones 7/94 IEEE Computer
[3] Robert Troy – Verilog ICSE 16, May 1994

Japanese SW Factory Reuse [4]

- Toshiba (1976) 1390 SLOCS/mo
- Toshiba (1979) 13% reuse rate
- Toshiba (1981) 2870 SLOCS/mo – 65% reuse rate
  0.3 defects/KSLOC
- Toshiba (1985) 3100 SLOCS/mo – 48% reuse rate
- Toshiba (1990) 30% reuse outside domain
- Toshiba (1990) 60% -80% reuse inside domain

- Hitachi (1986) increase productivity 2.3X with SW Parts
- Hitachi (1986) increase productivity 4.5X with Program Generator
- Hitachi (1988) increase productivity 2-3X with reusable logic frames
- Hitachi (1988) 80% reuse coverage in small office applications.
- Hitachi (1989) FREICT reduced workload 1/2 to 1/8
  Parts Creation Tool reduced workload by 90%.
- Hitachi (1991) 80% generation using EAGLE (73-98%)

[4] See *Chapter 18: Software Reuse Motivators and Inhibitors* in **Confessions of a Used Program Salesman: Institutionalizing Software Reuse** for a fuller treatment of this topic.

More Productivity/Quality Numbers

- NEC (1986) increase productivity 1.1-1.2X with SW parts
- NEC (1986) increase productivity 1.2-10X with standard logic parts
- NEC (1992) 90% coverage with SEA/I generator
- NEC (1992) 0% coverage with SEA/I generator yields up to 4X improvement
- NEC (1992) 70% coverage with SEA/I generator yields 10X improvement

- NTT (1988) reuse ratio 16%
- NTT (1990) reuse ratio saturated 20% (not domain-specific)

- Fujitsu (1989) DESIGN work load reduction 30PM ⇒ 10PM ⇒2PM (interface design reuse)
- Fujitsu (1991) using YPS/APG - productivity improvement 1.3-1.5

The Cost of Reuse – Tracz

1. **Up-Front**
   - 30% Design for Reuse
   - 15% Additional Documentation
   - 15% Additional Testing
   - ??% Additional Support
2. **First-Time**
   - ??% learning curve
3. **Recurring**
   - 20% Integration

Reuse Metrics

• Classifying Code Reuse

Base Code:	application independent
Salvaged Software:	not designed for reuse but not developed
Carry-Over Code:	application dependent – not changed
Reused Code:	designed for reuse and supported

• What to Count?

1. Generated Code
2. Calls
3. Includes

• What to Measure?

Reuse Ratio:	Reuse SLOC/Total SLOC
Deposition Ratio:	Reuse SLOC put into Library/Total SLOC
Reuse Utility:	(Reuse SLOC * Number of Uses)
Cost Avoidance:	Supported Reuse SLOC * Factor

Reuse Payoff Threshold: Cost to Develop < (N * Cost Aviodance)

Reuse Maturity Milestones (Martin Griss – HP)

Level	**Reuse**	**Requirements**
No Reuse	-20% to +20%	– Business as usual – Smart people
Salvaging	10% to 50%	– Smart people – Depends on luck – Maintenance problems
Planned Reuse	30% to 40%	– Reuse Library – Management Support – Incentives
Systematic Reuse	50% to 70%	– Reuse Library – Reuse Process – Reuse Metrics – Education
Domain-Oriented Reuse	80% to 90%	– Domain Analysis – Application Generators – Architectures

Reuse Inhibitors

1. **Nontechnical**[5]
   - no usesable software to reuse
   - (perceived) too hard to understand how to reuse
   - NIH (not invented here)
   - ignorance or arrogance
   - no incentive (management or employee)
   - not a trained skill
   - legal issues (copyright, patent, trade secrets)

2. **Technical**
   - no integration tool (interface, language, etc.)
   - * catalog/search problem[6]

Bottom Line: Cost to design and use it with confidence.

[5] *Software Reuse: Motivators and Inhibitors*, Proceedings of COMPCON'87, Tracz, pages 358-363.
[6] The reuse community considers this a closed research area!

"Reuse" Inhibitors that Technology Can Address

- Lack of components to reuse
- Cost to make reusable components
- Lack of quality components
- Can't locate component to reuse (a non-issue to some)
- Don't understand how to use a component
- Hard to modify a component
- Generalized components too inefficient
- Hard to integrate/compose components
- Possible litigation

Reuse Inhibitors and Technology Facilitators [7]

Reuse Inhibitor	Technology Facilitator
Lack of components to reuse	Domain analysis Knowledge acquisition/representation Standardization of constructs Domain-specific components Multimedia collaboration
Cost to make reusable components	Re-engineering environments Prototyping languages and environments Application generator generators
Lack of quality components	Formal specification of interfaces Verification of implementations Standard terminology Compositional optimizers
Can't locate component to reuse	POB Distributed data bases High-speed networks Domain taxonomies

[7] See: *Chapter 19: Software Reuse Technical Opportunities* in **Confessions of a Used Program Salesman: Institutionalizing Software Reuse** for a fuller treatment of this topic.

Reuse Inhibitors and Technology Facilitators

Reuse Inhibitor	**Technology Facilitator**
Don't understand how to use a component	Design rationale capture Knowledge-based hypermedia Data visualization Algorithm animation
Hard to modify a component	Parameterization Polymorphism Composition constructs
Generalized components too inefficient	Caching Operating systems Extensible languages/compilers Cross-component optimization
Hard to integrate/compose component	Composition/prototyping languages Knowledge capture/manipulation Constraint-based reasoning Layered architectures Tool integration frameworks Verification
Possible litigation	Encryption Authentication Software safety

Questions and Answers

Fire at Will