A View On Three R's (3Rs): Reuse, Re-engineering, and Reverse-engineering

Don Yu
Unisys Defense Systems
8201 Greensboro Drive, #900
McLean, VA 22102

There is a need to define the meanings of and relationships among the three software technology terms, "Reuse", "Re-engineering", and "Reverse-engineering" ("3R"). The confusing use of these three terms is prevalent not only among software managers but also practitioners of software engineering. I offer my personal view on the terminology for these three engineering directions and invite your feedback. Some CASE technology advocates are defining 3Rs as "Repository", "Reuse", and "Re-engineering", but I maintain that the term, "Reverse-engineering" should have its place in 3Rs, replacing the "Repository." Since the repository concept is a means of implementing any one of these three engineering objectives, it should not be lumped into the other two Rs. Although a precise definition of my 3Rs (not CASE advocate's) does not exist in software engineering community, I will describe my understanding of these three inter-related yet distinctive software technologies.

Information systems engineering is largely conducted in four ways: Forward-engineering, Reverse-engineering, Transverse-engineering, and Re-engineering. The latter three have a common ground in the already-developed, existing systems as a source for their activities. A set of goal-oriented transformations usually applies to the source under these technologies: initially, reverse-engineering was directed toward deriving high level specifications from code; reverse engineering, towards derivation of specifications from existing code; and transverse-engineering, towards converting derived specifications into application-specific design specifications. This initial boundary got blurred as each engineering technology has become inter-dependent and has taken an expanded role in objectives and approaches. Thus, transverse-engineering activities have become reverse-engineering practices, and reverse-engineering activities are merging into re-engineering exercises.

The definition of 3Rs is presented as follows:

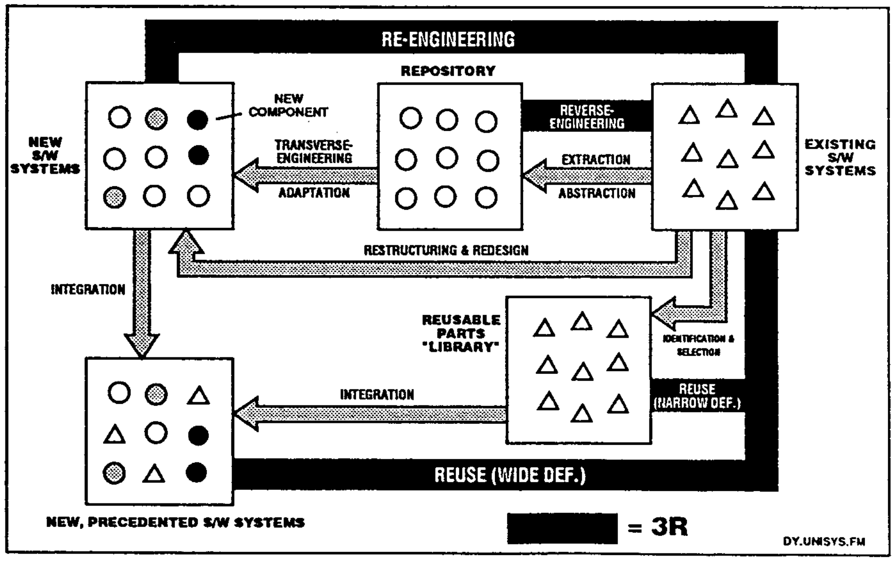
Reuse - Software engineering activities which focus on the identification of reusable software for straight import, reconfiguration, and adaptation for new computing system applications.

Re-engineering - Software engineering activities designed to effect the transformation of existing systems in order to achieve conformity with prevailing programming standards, to implement in high order languages for easier maintenance, to rehost to other hardware platforms, or to retarget to other computer system architectures. Re-engineering is usually initiated to transform existing "bad" systems to new "good" systems.

Reverse-engineering - Software activities pertaining to computer-aided extraction of specifications, design, and software components from existing software systems. Reverse-engineering implies derivation of abstract specifications from existing, "good" software systems and usually includes transverse-engineering steps.

Although many software engineering people may not agree with this definition, understanding of the central concepts for these three terminology will help one better relate to these activities. Re-engineering emphasizes re-design or re-development activities, whereas reverse-engineering focuses on transformation techniques. The majority of software reuse efforts relate to identification of reusable code or specifications and establishment of a resourceful library, based on such identification. Readers may note that the reverse-engineering supports the other two technologies or practices, whereas vice versa is not true. Also, re-engineering can be undertaken without any single reverse-engineered components. For instance, re-engineering of source code in the same implementation language as the one used in the original system reflect re-structuring and re-designing activities - a practice seen in "perfective" software maintenance process. Re-engineering of reverse-engineered components, however, would constitute much of the re-engineering activities today. It is done on the basis of cost-savings, recovery of design, re-documentation, and schedule savings. Once successfully reverse-engineered usable parts from existing software systems and re-engineered these parts for a project-specific adaptation, this success will then qualify for a case of apparent software reuse. Software reuse may depend on the reverse-engineering and re-engineering technologies, although software can be written such that it can be easily reused without the need of these two technologies.

Let's not misunderstand the meaning of "repository"; its meaning can vary, depending on the context in which the term repository is used. When it is used in the context of reuse, it usually means more like a library where directories and parts of reusable components are stored for efficient retrieval. When the term is, on the other hand, used in re-engineering (or less frequently in reverse-engineering), it means a database under a proprietary database management architecture which makes the computer-aided management of re-engineered specifications/code possible.The following figure displays the inter-relationships among these three concepts and technologies. A coordinated effort to 3Rs will bring a tremendous savings in software development time and monies, at least, for precedented systems.


RE-ENGINEERING
REPOSITORY
NEW COMPONENT
NEW S/W SYSTEMS
TRANSVERSE-ENGINEERING
ADAPTATION
REVERSE-ENGINEERING
EXTRACTION
ABSTRACTION
EXISTING S/W SYSTEMS
RESTRUCTURING & REDESIGN
INTEGRATION
REUSABLE PARTS "LIBRARY"
IDENTIFICATION & SELECTION
REUSE (NARROW DEF.)
INTEGRATION
REUSE (WIDE DEF.)
NEW, PRECEDENTED S/W SYSTEMS
= 3R
DY.UNISYS.FM