To Reuse or Not To Reuse

Jim Estipona, California Institute of Technology

Measuring Software Reuse: Principles, Practices, and Economic Models *by Jeffrey S. Poulin, Addison Wesley Longman, Reading, Mass., 1997, ISBN 0-201-63413-9, 195 pp., $38*

This book gives you the tools to start quantifying the benefits obtained from developing and using reusable software. Poulin argues that "business decisions drive reuse." Thus, to encourage management support for reuse and measure the degree to which software is being reused, a metric must be available that is

- uniformly understood across different software systems,
- practical enough to start and maintain measurement through different stages of the software life cycle, and
- able to collect data simply.

Poulin presents a variety of metrics drawn from papers and economic models. He does a good job of pointing out the lack of clear definitions in many of the published experience reports that address what should be measured as reuse. Unless this is first clearly defined, the figures reported are useless. Poulin includes his own metrics suite—the "Reuse Metric Starter Set" (or Poulin and Caruso Reuse and Return on Investment Metrics)—that you can use to quantify software reuse. He also proposes some specific figures for use with this suite, including new-code error rate, cost of fixing an error, relative cost for writing reusable software, relative cost for using reusable software, and cost for new code.

Questionable Quirks

In defining reusable software as code developed "someplace else," Poulin conveniently excludes reuse of software within an organization, instead calling such recycled code just "good programming." With the advent of object orientation, which inherently encourages reuse, this definition seems too arbitrary.

The book focuses mainly on structured programming practices. OO people may find that some familiar terms, like abstract data type (ADT), do not coincide with how Poulin uses them. In particular, he seems to suggest that ADTs are generally small and domain independent. In OO, however, ADTs can be large and occur on any of several levels, including the application level.

In Chapter 7, "Measuring Software Reusability," Poulin suggests that software complexity or size is inversely proportional to software usability. This would seem to contradict the use of frameworks or foundation class libraries in OOP, whether you measure complexity using McCabe's Cyclomatic Complexity or Halstead's Software Science metrics. Foundation classes and frameworks are some of the largest and most complicated components in software, yet anyone who programs in C++ would be unlikely to claim low levels of reuse for them.

Reuse Resource

Overall, Poulin provides a readable account of measuring software reuse and estimating its cost and benefits. Further, he seeks to define what does and doesn't constitute reuse, then offers tools for gauging its effectiveness in your organization. Of particular interest is the Frakes and Terry Reuse Level Metric, which allows measurement of internal reuse. Finally, the book's extensive references and bibliography offer a solid starting point for further research into reuse. ❖

Narrow Economic Analysis of India's Software Industry

Don Shafer, Cirrus Logic

India's Software Industry: State Policy, Liberalisation, and Industrial Development *by Richard Heeks, Sage Publications, Thousand Oaks, Calif., 1996, ISBN 0-8039-9265-3, 397 pp.*

India's Software Industry addresses one very specific question: How has liberalization of governmental policy affected the development of the software industry in India? Heeks is an excellent economist and geoeconomist and his book reflects this. He is neither a computer scientist nor a software specialist; thus, while he elucidates export law, import economics, and the business of establishing a software industry, he provides no concrete information on life cycles, how and why specific software products are produced, or the