

article summaries

MAY/JUNE 2004, VOLUME TWENTY-ONE, NUMBER THREE

RETURN ON INVESTMENT

Calculating ROI for Software Product Lines

by Günter Böckle, Paul Clements, John D. McGregor, Dirk Muthig, and Klaus Schmid, pp. 23–31. The product line engineering approach can improve the ROI from a set of products. Because product lines emphasize economic optimization, the need for ROI calculations is particularly important. This article describes a model for product line ROI and illustrates it in a prototypical industrial situation.

Measuring the ROI of Software Process Improvement

by Rini van Solingen, pp. 32–38. Software practitioners often hear “We cannot quantify the benefits of SPI, because it is not possible to express indirect benefits such as customer satisfaction or personnel motivation in financial numbers” as an argument against calculating the ROI for software process improvement. An approach is presented detailing how to measure benefits, as are case-study findings from pragmatic ROI analyses for SPI and an overview of literature on the ROI of SPI.

The Incremental Funding Method: Data-Driven Software Development

by Mark Denne and Jane Cleland-Huang, pp. 39–47. In the current IT environment, organizations are no longer willing to invest in software development without clear expectations for returns, and they’re demanding those returns in much less time. This clearly challenges the development community to change the way we do business. The Incremental Funding Methodology is a data-driven, financially informed approach to software development. IFM can analyze and sequence feature delivery to maximize net present value and to provide insight on how development decisions affect other financial metrics.

Value Creation and Capture: A Model of the Software Development Process

by Todd Little, pp. 48–53. Understanding software development dynamics can help managers and project team members maximize value delivery. Using functions and parameters applied in a spreadsheet, this process model facilitates this understanding by examining value creation and value capture in the presence of uncertainty. Items to explore include project staffing, optimal release dates, and potential impacts of process improvements such as pair programming.

The ROI of Software Dependability: The iDAVE Model

by Barry Boehm, LiGuo Huang, Apurva Jain, and Ray Madachy, pp. 54–61. The Information Dependability Attribute Value Estimation model helps users reason about the ROI of software dependability investments. iDAVE estimates the relative ROI for achieving the desired values for dependability attributes and helps select the most effective software dependability strategies. The authors describe its overall structure and illustrate its use in determining the ROI of dependability investments for a commercial order-processing system and a NASA planetary rover project.

Marketplace Issues in Software Planning and Design

by David G. Messerschmitt and Clemens Szyperski, pp. 62–70. For commercial software sold in the general marketplace, many software-planning and design decisions are based not only on meeting user needs but also on marketplace issues. Many of these issues fall into one of three standard ROI categories: revenue, cost, or risk. This article focuses on the uncertainties and strategic issues that suppliers encounter in selling or licensing software in the general marketplace.

FEATURES

A Task Allocation Optimizer for Software Construction

by Jim Duggan, Jason Byrne, and Gerard J. Lyons, pp. 76–82. The allocation of development tasks within a software project is a complex activity. There are many factors to consider, including the programmers’ skills and productivity levels. Multiobjective optimization is based on evolutionary algorithms and can generate a set of optimal solutions to problems with conflicting objectives. This article shows how to successfully apply this technique to allocate tasks within a software development team.

Quality-Evaluation Models and Measurements

by Jeff Tian, pp. 84–91. Quality can determine a software product’s success or failure in today’s competitive market. Among the many characteristics of quality, some aspects deal directly with functional correctness or the conformance to specifications, while others deal with usability, portability, and so on. Correctness is typically the most important aspect of quality. This article classifies several quality-evaluation models.

Righting Software

by James R. Larus, Thomas Ball, Manuvir Das, Robert DeLine, Manuel Fähndrich, Jon Pincus, Sriram K. Rajamani, and Ramanathan Venkatapathy, pp. 92–100. Correctness tools can improve software development by supplementing human shortcomings—detecting programming errors such as null-pointer references, API usage errors, and failing to close file descriptors. Microsoft Research has developed two generations of correctness tools. Together, these tools offer a systematic approach for finding and fixing errors early in the development process.