article summaries

NOVEMBER/DECEMBER 2005, VOLUME TWENTY-TWO, NUMBER SIX

PREDICTOR MODELS

Building Effective Defect-Prediction Models in Practice

by A. Güneş Koru and Hongfang Liu, pp. 23–29. Successfully predicting defect-prone software modules can help developers improve product quality by focusing quality assurance activities on those modules. Emerging repositories of publicly available software engineering data sets support research in this area by providing static measures and defect data that developers can use to build prediction models and test their effectiveness. Stratifying NASA data sets from the PROMISE repository according to module size showed improved prediction performance in the subsets that included larger modules.

Improving After-the-Fact Tracing and Mapping: Supporting Software Quality Predictions

by Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram, pp. 30–37. The requirements traceability matrix can successfully predict quality before code is written. However, its tedious development process, requiring analysts to manually discover and vet links between artifact levels, has stymied widespread adoption. The authors' tracing toolkit, called RETRO (Requirements Tracing on Target), automates the information retrieval process, making RTM's development easier. Backed by empirical results, the authors describe RETRO's advantages over other information retrieval approaches.

Finding the Right Data for Software Cost Modeling

by Zhihao Chen, Tim Menzies, Daniel Port, and Barry Boehm, pp. 38–46. Strange to say, when building a software cost model, sometimes it's useful to ignore much of the available cost data. One way to do this is to perform data-pruning experiments after data collection and before model building. Experiments involving a set of Unix scripts that employ a variable-subtraction algorithm from the WEKA (Waikato Environment for Knowledge Analysis) data-mining toolkit illustrate this approach's effectiveness.

FEATURES

The Art and Science of Software Release Planning

by Günther Ruhe and Moshood Omolade Saliu, pp. 47–53. Poor release planning decisions can result in unsatisfied customers, missed deadlines, unmet constraints, and little value. The authors investigate the release planning process and propose a hybrid planning approach that integrates the knowledge and experience of human experts (the "art" of release planning) with the strength of computational intelligence (the "science" of release planning).

Using the OPC Standard for Real-Time Process Monitoring and Control

by Jun Liu, Khiang Wee Lim, Weng Khuen Ho, Kay Chen Tan, Arthur Tay, and Rajagopalan Srinivasan, pp. 54–59. The effort to develop this open, nonproprietary, plug-and-play system using the Object Linking and Embedding for Process Control (OPC) standard had two goals. The first was to make lab-based research and development work on process monitoring and control directly and easily applicable to industrial plants. The second was to provide a standard method for individual process monitoring and process control software to interact and share data. This article describes the system's design and implementation.

Opportunistic Problem Solving in Software Engineering

by Pierre N. Robillard, pp. 60–67. Software development is a complex, mainly cognitive endeavor. Studying human behaviors such as problem solving and opportunistic design can identify ways to improve our software engineering practices. Real and durable improvements will likely emerge from a scientific approach based on observing and measuring human behavior. One finding is that synchronization meetings can improve practices with significant opportunistic problem-solving content.

Software Process Improvement in Small Organizations: A Case Study

by Kathleen Coleman Dangle, Patricia Larsen, Michele Shaw, and Marvin V. Zelkowitz, pp. 68–75. A case study of a five-year-old startup company looks at the role of process improvement in the context of a small organization. It explores why the company realized that process improvement was necessary for continued growth. The company is implementing many CMM-related key practice areas, and they should be able to achieve their goal of CMM Level 3 in the near future.

The Economic Impact of Learning and Flexibility on Process Decisions

by Hakan Erdogmus, pp. 76–83. Understanding how high-level process characteristics alter a software project's economics can help organizations make informed decisions and improve value creation. Two central characteristics of iterative and incremental development—flexibility and learning—have much to do with whether a development process makes economic sense.

Managing Change in Software Process Improvement

by Lars Mathiassen, Ojelanki K. Ngwenyama, and Ivan Aaen, pp. 84–91. When software managers initiate software process improvement, most are ill prepared for the scale and complexity of the organizational change involved. Although they typically know how to deal with large software projects, few managers have sufficient experience with projects that transform organizations. For successful software process improvement, managers must understand the context of change, know the organizational elements involved, and master the tactics that facilitate successful change.

0740-7459/05/$20.00 © 2005 IEEE