Perry, Humphreys, Kan, Cohen, Wilson, Fenton, *et alia.*

Reviewed by Dr. Robert Bruce Kelsey, Storage Management Software, DEC — robert.kelsey@cxo.mts.dec.com

How To Run Successful Projects

Fergus O'Connell

How To Run Successful Projects is written by Fergus O'Connell and is published by Prentice-Hall 1994, softbound, ISBN 0-13-138793-6, 170 pages, $44.95.

O'Connell wants to dispel the myth that software project management is a "black art." His Structured Project Management approach is not, however, a tedious, cumbersome, procedure-oriented workbook for those who lack telepathy or crystal balls. The book is a loosely connected, four part collection of (to use his words) "useful and practical ideas" presented "in an entertaining and stimulating way."

Parts I and II describe the Ten Steps of Structured Project Management. They are quick reads, and one doesn't notice that it isn't until chapter nine that O'Connell addresses software- specific planning. Structured Project Management isn't innovative - the steps include the ubiquitous visualizing the goal, setting the tasks, resource assignment, etc. What makes these sections worth reading is O'Connell's sense of humor and his emphasis on planning and people rather than on how to give good Gantt charts.

Part III begins with short summaries of techniques for interviewing, time management, conflict resolution, and accelerated analysis and design (a technique for brainstorming and documenting requirements). The final section offers a detailed project schedule using Boehm's 1981 data on the relationship between effort and schedule. Part IV describes O'Connell's Probability of Success Indicator and applies this rating system to some of the projects he has cited throughout the book.

Those not fond of militaristic similes and examples may not always find O'Connell entertaining, but those who think "project manager" and "tactician" are synonyms will certainly find a kindred spirit in O'Connell.

Reviewed by Dr. Robert Bruce Kelsey, Storage Management Software, DEC — robert.kelsey@cxo.mts.dec.com

A MAP For Software Acquisition

Jean E. Tardy

MAP For Software Acquisition is written by Jean E. Tardy and is published by Monterège Design 1991, softbound, ISBN 2-9802283-0-3, 243 pages, price not available.

"Most texts...do not have many items of value within their pages...."

So begins this self-styled "guide book" on how to acquire software. Tardy is the president of the consulting firm Monterège Design, MAP stands for Monterège Acquisition Process, and the book is a consulting support tool, as the course outline in Appendix C makes clear. But instead of describing how to acquire software, the book is little more than a tedious taxonomy of acquisition activities. **MAP** is full of verbiage but offers little practical advice about how to manage a project, how to develop and track requirements, how to monitor product development, how to perform acceptance tests, and how to determine support requirements.

But what a taxonomy! After oversimplifying "Software Engineering Basics" and "The Software Lifecycle," Tardy dissects "acquisition:" the acquisition process (of which there are four types, Extension Acquisition, Substitution Acquisition, Component Acquisition, and Autonomous Acquisition) comprises production activities and management activities; four production activities (definition, design, development, and delivery) each generate one of the four major deliverables (specifications, plans, program, and product) while the product itself matures through five distinct baselines; management activities include project initiation, direction, monitoring, and the project conclusion. Etc., etc., for another hundred or so pages.

In Tardy's view, "software quality refers loosely to an assessment of software based on features which are *not part of its specific functional requirements*" (21, italics mine). That statement may be disputable but it is not without use, for one could on the same logic separate the quality of a book from its content. For a book that omits object oriented technology from its description of software engineering, cites only one source and only four quality factors for software, and uses the equations for estimation from Boehm without proper calibration (221), that may be advantageous....

Reviewed by Dr. Robert Bruce Kelsey, Storage Management Software, DEC — robert.kelsey@cxo.mts.dec.com

Debugging the Development Process

Steve Macguire

Debugging the Development Process is written by Steve MacGuire and published by Microsoft Press 1994, (paperback), ISBN 1-55615-650-2, 183 pages, $24.95.

This book was a pleasure to read. It deals mostly with system delivery issues (of which programming is just a part) which are important to a technical contributor. It was easy to read and made a number of important and useful points, most of which are easy to implement in your day to day practice as a technical person.

The key points are summarized in larger, italic type centered on 3 inch lines. You can't miss them. For example:

> *Be sure that every report you ask for is worth the time it takes for the writer to prepare it.*

> *Don't implement features simply because they are technically challenging or "cool" or fun or...*

How many times are these popular folk wisdom, often neglected. Points like these are interspersed with war stories, and the writer uses a easy to read style to explain these points. Looking back, since the points are so poignant, most of the book's value could be gained be merely looking at and remembering these points. It probably would be valuable just to list the points (in hindsight, it could be a valuable appendix to the book). The text merely serves to give a foundation to justify these conclusions.

I really liked the typography and layout: lots of good, entertaining sidebars. It has an index and a brief bibliography (listing a number of business and computer science books; all the computer science books I've read and recommend if you haven't read them). The table of contents contains an abstract of each chapter. The book is of the same flavor of Brook's **Mythical Man Month**, on a smaller scale and somewhat updated. Definitely worthwhile to have on your library shelf and refer to in the future.

Reviewed by Marty Leisner, Xerox Corp., Rochester, New York — leisner@sdsp.mc.xerox.com

Managing Your Move to Object Technology: Guidelines and Strategies for a Smooth Transition

Barry McGibbon

Managing Your Move to Object Technology is written by Barry McGibbon and published by SIGS books 1995, in the Managing Object Technology Series, ISBN 1-884842-15-2, 269 pages, $35.

I have to admit: I'm not a big proponent of object technology. I am a big propopent of quality. Since quality is a elusive goal which has had debatable success over the last 40 years, the popular goal is now "Moving to Objects." I read this book because I wanted to gain insight into what made object oriented programming different than conventional programming. I was disappointed in this book.

It talks about managing software development with the McKinsey Seven S's (strategy, structure, systems, staff, styles, skills and shared values). One chapter is devoted to each "S." The analogy seemed interesting, but not unique.

The thrust of the book is taking a bland software software management text from the 1980s, and randomly inserting the word "object" once or twice each page and "class" every other page. Much like many object-oriented programmers I've met just rename their components without changing their methodology. I radically disagree with some of the points he makes. For example, talking about the Software Component Factory and reuse the author states:

> *The factory concept has been around for a long time...It has not always been successful as the languages used tended to inhibit effective reuse...Now, thanks to object-oriented lanagues, with their encapsulation and extension, the idea once more has merit.*

When did this the idea not have merit? And another good quote:

> *Object-oriented development is much more than the syntax of the language. &... For example, you can learn Smalltalk in roughly three days, but it may take up ot six months to fully comprehend the extensive library of classes.*

Why is this any different than a convential language like C? (of which I'm very partisan to). The common logic taught in schools is "a computer scientist can learn a language in two weeks". But after practice, I'd say a minimum of six months practice is necessary to use it effectively, with several years to master it.

There are pointers to other books (with an author and a date), but no bibliography. Which is too bad, since I like many of the references.

I read the book primarily to learn what was different about managing object technologies compares to structured technologies. The book did not fulfill this mission.

Reviewed by Marty Leisner, Xerox Corporation, Rochester, New York — leisner@sdsp.mc.xerox.com

Beyond Technology's Promise

Joseph B. Giacquinta, Jo Anne Bauer, & Jane E. Levin

Beyond Technology's Promise An Examination of Children's Educational Computing at Home is written by Joseph B. Giacquinta, Jo Anne Bauer, & Jane E. Levin, and published by Cambridge University Press 1993, ISBN 0-521-40447-9 ISBN 0-521-40784-2 (paperback) 244 pages. $16.95.

The personal computer promised to change education. It promised to help learning in the classrooms of public schools that were increasingly under attack for failing to teach useful skills. Moreover, it promised to help learning at home, where parents were told that for their children to be successful, the computer would be necessary. Educational Software promised to become the modern tutor.

Beyond Technology's Promise is the result of a three year study entitled "Studies of Interactive Technology in Education". The research traced computer usage both at home and in school among 70 families. The findings revealed a near-absence of the promised academic computing, only a modest amount of 'educational computing' –programming or word processing – and almost no telecommunicating. Instead, game playing took up most of the time these children spent on computers. While not terribly surprising, the reasons children did not use computers for academic learning are believed to be social: first, their parents didn't encourage or aid such use; second, schools emphasized other forms of use.