

Essentials of Software Process

Hakan Erdogmus

In “The Challenge of Good Enough Software” (*American Programmer*, Oct. 1995), James Bach introduced the utilitarian view of software process. Well over a decade has passed since this article was published, but I find myself increasingly subscribing to this idea. It’s likely to dominate the post-big-process, post-agile, post-model-driven-development era to come. Whatever is emerging from grassroots and craft-oriented process trends colliding and converging with traditional research- and engineering-based approaches has a lot to do with the utilitarian view. Here, I attempt to deconstruct and extend this view in terms of seven essential characteristics that mutually reinforce and balance each other.



I define “process” like Bach does (see www.satisfice.com for an updated version of the 1995 article) but with a slight adaptation. A *process* is a collection of patterns and activities for solving a class of problems. We can place process trends inside a triangular map according to their emphasis relative to three aspects, represented by the vertices: people, technology, and rigor. Plan-oriented, engineering, and research-based approaches tend to view software as a rigid artifact that’s difficult to change, so they stress technology and rigor aspects over the people aspect. Evolutionary approaches tend to view software development as an organic,

skills-driven technical activity, so they stress people and technology over rigor. But this scheme of positioning process approaches is rather rough. A more complete scheme requires a dissection in more than three dimensions. The new dimensions comprise the seven essential characteristics mentioned earlier: human-centricity, technical orientation, discipline, pragmatism, empiricism, experimentation, and value orientation.

Human-centricity

Human-centricity in a nutshell is peopleware, after Tom DeMarco and Tim Lister’s famed book of compelling narratives. People’s role is so evident that I first wondered whether to bother listing it. On the other hand, this role is still notorious for receiving lip service. Clearly, software development is an intellectual endeavor, and serious software development is a team activity. A human-centric process emphasizes collaboration, recognizes the importance of effective leadership, and caters to the needs of creative professionals who take pride in their work.

Human-centricity requires understanding what motivates knowledge workers and how they perform their work and interact with others. It also demands an appreciation of how non-software professionals perceive software development, what customers want, and, when human users are involved, how people use the software. Thus, human-centricity is about valuing soft skills as much as hard skills.

Mission Statement: To be the best source of reliable, useful, peer-reviewed information for leading software practitioners—the developers and managers who want to keep up with rapid technology change.

Software Boards

The Computer Society has recently launched its new online face, Computing Now. Our new associate editor in chief Maurizio Morisio will act as liaison to Computing Now in addition to handling submissions in our high-volume coverage areas. Maurizio is an associate professor in Politecnico di Torino's Department of Automation and Computer Science, where he manages a research group. His expertise spans object-oriented analysis, software design, programming paradigms, software reuse, software process, open source development, and experimental software engineering.

Many thanks to departing members David Dorenbos, Jeffrey Voas, and Stephen Mellor. David has served in the Advisory Board since 2003, undertaking special projects and contributing to editorial content. Jeffrey Voas joined the Advisory Board in 2004 after having served as an associate editor in chief. Stephen Mellor served on the Editorial Board from 1997 to 2001, after which he joined the Advisory Board and later performed an outstanding job leading it. Frances Paulisch replaces Stephen as the new chair of the Advisory Board.

But can peopeware singly guarantee success?

Technical orientation

It's rather hard to get useful software by throwing together a bunch of people with wonderful soft skills but not enough individual technical competencies to do the job. Here, competence in both fundamentals and software technology matters. Fundamentals include base knowledge of such things as algorithms, data structures, programming paradigms, design patterns, testing, requirements, user interfaces, and reference architectures. Technology includes languages, tools, environments, frameworks, platforms, and other relevant infrastructure. A technically oriented process fosters possession and acquisition of teamwide competence in fundamentals and technology.

The trademark of any sensible process is continuous attention to technical excellence, in which a solid infrastructure fit for the purpose is primary. A technically oriented process leverages tools and automation so that people are liberated from performing mechanical, error-prone tasks and can focus their energies on more intellectually demanding ones.

A technical orientation perceives as disingenuous failure to automate mundane, repetitive activities, such as the build or deployment process (see "Software Builders," *IEEE Software*, May/June 2008). But it also advocates not aspiring to mechanize activities that require creative, skillful hu-

man intervention. The definitive artifacts of a technically oriented process are the ones subject to human manipulation at the lowest abstraction level.

Technical orientation adds meat to human-centricity, but doesn't any serious, sustainable, scalable business need managerial and procedural rigor?

Discipline

Many view a process's methodical, managerial, and operational dimensions as the highway to predictability, sustainability, and repeatability. These dimensions constitute the rigor aspect of a process. But discipline isn't equivalent to formal procedures, top-down management, maturity levels, unrelenting compliance, and a forever-orderly, sterile development. Discipline is the systematic use of synergistic practices appropriate for the context. It's about robust structuring of workflow and division of labor as well as accepting, expecting, and accordingly managing chaos.

A disciplined process applies its chosen technical practices consistently, if not blindly. These practices may include coding standards, frequent builds, requirements modeling, architecting, informal design, code inspections, regressive unit testing, and exploratory testing. A disciplined process also consistently applies management practices to coordinate activities, share knowledge, and control product artifacts. These might entail business analysis, requirements management, documentation as needed,

EDITOR IN CHIEF

Hakan Erdogmus

hakan.erdogmus@computer.org

EDITOR IN CHIEF EMERITUS:

Warren Harrison, Portland State University

ASSOCIATE EDITORS IN CHIEF

Computing Now: Maurizio Morisio, Politecnico di Torino; maurizio.morisio@polito.it

Design/Architecture: Uwe Zdun, Vienna Univ. of Technology; zdun@infosys.tuwien.ac.at

Development Infrastructures: Martin Robillard, McGill University; martin@cs.mcgill.ca

Distributed and Enterprise Software:

John Grundy, University of Auckland; john-g@cs.auckland.ac.nz

Empirical Results: Forrest Shull, Fraunhofer Center for Experimental Software Engineering, Maryland; fshull@fc-md.umd.edu

Human and Social Aspects: Helen Sharp, The Open University, London; h.c.sharp@open.ac.uk

Management: John Favaro, Consulenza Informatica; john@favaro.net

Processes and Practices: Frank Maurer, University of Calgary; maurer@cpsc.ucalgary.ca

Programming Languages and Paradigms:

Sophia Drossopoulou, Imperial College London; s.drossopoulou@imperial.ac.uk

Quality: Annie Combelles, DNV/Q-Labs; annie.combelles@dnv.com

Requirements: Ann Hickey, University of Colorado at Colorado Springs; ahickey@uccs.edu

DEPARTMENT EDITORS

On Architecture: Grady Booch, IBM

Bookshelf: Art Sedighi, SoftModule

Design: Rebecca J. Wirfs-Brock, Wirfs-Brock Associates

Loyal Opposition: Robert Glass, Computing Trends

Not Just Coding: J.B. Rainsberger, Diaspar Software Services

Professional Development: Philippe Kruchten, University of British Columbia

Requirements: Neil Maiden, City University, London

Software Technology: Christof Ebert, Vector

Tools of the Trade: Diomidis Spinellis, Athens Univ. of Economics and Business

User Centric: Jeff Patton, consultant

Voice of Evidence: Forrest Shull, Fraunhofer Center for Experimental Software Engineering

ADVISORY BOARD

Frances Paulisch, Siemens (Chair)
 Jennitta Andrea, ClearStream Consulting
 Elisa Baniassad, Chinese University of Hong Kong
 J. David Blaine, consultant
 Kaoru Hayashi, SRA
 Simon Helsen, SAP
 Juliana Herbert, Herbert Consulting
 Gargi Keeni, Tata Consultancy Services
 Karen Mackey, Cisco Systems
 Steve McConnell, Construx Software
 Erik Meijer, Microsoft
 Grigori Melnik, Microsoft
 Bret Michael, Naval Postgraduate School
 Ann Miller, University of Missouri, Rolla
 Deependra Moitra, Infosys Technologies, India
 Linda Rising, consultant
 Wolfgang Strigel, consultant
 Dave Thomas, Bedarra Research Labs
 Laurence Tratt, Bournemouth University
 Markus Völter, consultant

STAFF

Senior Lead Editor
Dale C. Strok
 dstrok@computer.org

Senior Editorial Services Manager
Crystal Shif

Magazine Editorial Manager
Steve Woods

Staff Editors
**Rebecca Deuel, Shani Murray,
 Dennis Taylor, Linda World**

Assoc. Peer Review Manager
Hilda Carman

Publications Coordinator
Kathleen Henry
 software@computer.org

Production Editor
Jennie Zhu

Technical Illustrators
Sarah Attwood, Ann Monn, Alex Torres

Director, Products & Services
Evan Butterfield

Digital Library Marketing Manager
Georgann Carter

Senior Business Development Manager
Sandra Brown

Senior Advertising Coordinator
Marian Anderson

CONTRIBUTING EDITORS

**Thomas Centrella, Molly Gamborg, Robert Glass,
 Keri Schreiner, Joan Taylor**

CS PUBLICATIONS BOARD

Sorel Reisman (chair), Angela Burgess,
 Chita R. Das, Van Eden, Frank E. Ferrante,
 David A. Grier, Pamela Jones, Phillip A. Laplante,
 Simon Liu, Paolo Montuschi, Jon Rokne,
 Linda I. Shafer, Steven L. Tanimoto

MAGAZINE OPERATIONS COMMITTEE

David A. Grier (chair), David H. Albonesi,
 Arnold (Jay) Bragg, Carl K. Chang,
 Kwang-Ting (Tim) Cheng, Norman Chonacky,
 Fred Douglas, Hakan Erdogmus, James A. Hendler,
 Carl E. Landwehr, Dejan Milojicic,
 Sethuraman (Panch) Panchanathan,
 Crystal R. Shif, Maureen Stone,
 Roy Want, Jeffrey Yost

Editorial: All submissions are subject to editing for clarity, style, and space. Unless otherwise stated, bylined articles and departments, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in IEEE Software does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society.

To Submit: Access the IEEE Computer Society's Web-based system, Manuscript Central, at <http://cs-ieee.manuscriptcentral.com/index.html>. Be sure to select the right manuscript type when submitting. Articles must be original and not exceed 5,400 words including figures and tables, which count for 200 words each.

Are You Running a Software Engineering Research Lab?



We're offering complementary print copies for a period of two years to select university research groups with student members. Let us know if you're responsible for a software engineering research group at a reputable academic department and wish to receive two complementary copies of *IEEE Software* to be made available to your group in a common area. Tell us a bit about your group and its research focus (not exceeding one paragraph). Don't forget to mention the group's size and composition as well as the contact person's name, email, and postal address. Indicate whether your institution is privately or publicly funded. This is a limited-time offer available to eligible research groups on a first-come, first-served basis as long as quotas last. Each geographical area has a quota. Send your request to software@computer.org, with subject line "research group comp copies."

time-boxed iterations, phasing, estimation, progress tracking, and retrospectives.

Discipline is essential, but it often needs to be moderated for specific needs.

Pragmatism

Pragmatism is the moderating counterforce to discipline. It's as important as discipline for making a process tick. To be pragmatic, a process activity should be continuously feasible, adapted to your context, scalable, and compatible with your organization's changing culture and risk-taking preferences. A process activity should serve a clear purpose, whether it's to meet customers', project teams', or users' needs or to satisfy regulatory requirements. So pragmatism is just common sense, which Bach's utilitarian view advocates.

It's possible to be pragmatic with process choices, but how does one know that the decisions, no matter how sensible they seem, are paying off as expected? Extra elements need to be in place in order to take calculated risks with a pragmatic but disciplined approach.

Empiricism

This brings us to empiricism—a concept not too popular with grassroots circles because of its traditional association with “big” process, quantitative management, and academic research. Empiricism at its heart is supporting decisions through evidence based on data, both observations and measurements. By observations I

mean occurrences that we can simply record. By measurements I mean things we can count, calculate, or quantify. Measurements have values, whereas observations have descriptions, possibly including contextual information. Observations provide deeper insight in areas in which measurements serve only as proxies for other constructs. Each kind of data has a place, and empiricism entails collection and use of both kinds.

The right data give a team the ability to preserve knowledge, connect actions to consequences, and relate inputs and outcomes. This awareness in turn helps the team figure out when it's failing or succeeding or whether it's improving or deteriorating, sense why things are going the way they are, and take corrective, preventative steps.

Empiricism increases the quality of decision making and the value of planning. Management activities—estimation, quality assurance, release planning, resource allocation, progress tracking—become more meaningful and better grounded through empiricism. As Robert L. Glass states in *Facts and Fallacies of Software Engineering* (Addison-Wesley, 2003), “managing in the presence of data is far better and easier than managing in its absence.”

Empiricism also helps us test our theories about what works and what doesn't more objectively than with just gut feeling alone, hence fostering learning.

Experimentation

To learn, we need experimentation. Experimentation is possible in an environment in which alternative solutions to problems, viability and effectiveness of ideas, and hypotheses about prospective actions can be tested safely and cheaply to resolve uncertainty.

The concept of experimentation should be familiar to many. The Extreme Programming idea of a spike to assess a technical solution's feasibility is a form of experimentation. Prototyping, simulations, role playing, split runs, and early usability evaluations constitute other examples.

In the absence of uncertainty, experimentation is unnecessary. But in its presence, an unfavorable answer is obviously a possibility. Without a realistic expectation of a negative outcome, learning can't occur. Thus critical in effective experimentation is the acceptance of occasional, transient failure. Learning happens with deliberate trial and error. A process that's intolerant and punishing of unfavorable transient results will discourage experimentation.

We have almost come full circle. Sys-

tematic experimentation is actually a value-generating activity.

Value orientation

Value orientation goes hand in hand with pragmatism. As such, it's the next characteristic that closely relates to Bach's utilitarian approach. John Favaro provides a stereotypical manifestation of value orientation in "When the Pursuit of Quality Destroys Value" (*IEEE Software*, May 1996).

Value orientation is parsimonious purposefulness. Software projects operate under scarce resource and schedule constraints. Decisions and activities are subject to not only availability of time and money but also the unrealized benefits of future opportunities. In such environments, it's critical to reason about what ultimately matters, to whom, and why. Such reasoning entails resourceful consideration in cost-benefit terms of the trade-offs between productivity and quality, repeatability and flexibility, customer/user and employee satisfaction, predictability and

creativity, authority and autonomy, and short- and long-term goals.

The bottom line often constitutes the most important component of an activity's value to stakeholders. Early realization of benefits and deference of costs increase economic value. Obviously, so does minimization of overhead. According to financial theory, under conditions of uncertainty, economic value also increases through waiting, learning, flexibility, and risk management. This is a critical but less obvious component of value orientation. Value orientation thus implies perceiving the contribution of process activities in a new light of value maximization under uncertainty.

The seven essential characteristics I just listed aren't orthogonal. Rather, they resemble suspended blobs connected with invisible forces: you tug at one, and the others move in every direction. They help highlight the strengths and inevitable weaknesses of process approaches.

If your perspective differs, write me at hakan.erdogmus@computer.org. ☺

CALL FOR ARTICLES

Software Development for Embedded Systems

This special issue will share proven embedded-systems ideas and experiences from all phases of the development life cycle and from a range of industry domains. Of specific interest is how practices and methods from enterprise and desktop computing can be transferred into the embedded domain. Topics of particular interest include

- Design methods (so-called DFX)
- Domain-specific languages, including hardware related issues
- State of the practice in modeling, design, verification, and validation
- Communication protocols and middleware
- Modeling of quality attributes (for instance, where safety meets security)
- Concepts for maintenance, robustness, and remote diagnosable architectures
- Schemes for remote software maintenance and testing

PUBLICATION: May/June 2009

SUBMISSION DEADLINE: 1 Nov. 2008

GUEST EDITORS:

- Christof Ebert, Vector,
christof.ebert@vector-consulting.de
- Jürgen Salecker, Siemens,
juergen.salecker@siemens.com

COMPLETE CALL: [www.computer.org/
software/cfp3.htm](http://www.computer.org/software/cfp3.htm)

AUTHOR GUIDELINES: [www.computer.org/
software/author.htm](http://www.computer.org/software/author.htm)

SUBMISSION DETAILS: software@computer.org

IEEE Software [www.computer.org/
software](http://www.computer.org/software)