

**Victor Larios
Félix F. Ramos
Herwig Unger (Eds.)**

LNC3 3061

Advanced Distributed Systems

**Third International School and Symposium, ISSADS 2004
Guadalajara, Mexico, January 2004
Revised Selected Papers**



Springer

TP274-53

A244

2004

Victor Larios Félix F. Ramos

Herwig Unger (Eds.)

Advanced Distributed Systems

Third International School and Symposium, ISSADS 2004
Guadalajara, Mexico, January 24-30, 2004
Revised Selected Papers



E200404151



Springer

Volume Editors

Victor Larios

Universidad de Guadalajara

CUCEA, Dept. Sistemas de Informacion

799, Periferico Norte, Edif.L-308, Zappam, Jal., 45100, Mexico

E-mail: vlarios@acm.org

Félix F. Ramos

CINVESTAN

Prol. López Mateos Sur 590 Guadalajara, J 45090 A.P. 31-438, Mexico

E-mail: framos@gdl.cinvestav.mx

Herwig Unger

Universität Rostock

Fachbereich Informatik, Albert-Einstein-Str.23, 18051 Rostock, Germany

E-mail: hunger@informatik.uni-rostock.de

Library of Congress Control Number: 2004107500

CR Subject Classification (1998): C.2.4, I.2.11, D.2.12, D.1.3, D.4, H.3, H.4

ISSN 0302-9743

ISBN 3-540-22172-7 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable to prosecution under the German Copyright Law.

Springer-Verlag is a part of Springer Science+Business Media

springeronline.com

© Springer-Verlag Berlin Heidelberg 2004

Printed in Germany

Typesetting: Camera-ready by author, data conversion by DA-TeX Gerd Blumenstein

Printed on acid-free paper SPIN: 11011446 06/3142 5 4 3 2 1 0

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

New York University, NY, USA

Doug Tygar

University of California, Berkeley, CA, USA

Moshe Y. Vardi

Rice University, Houston, TX, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Preface

This volume contains the accepted papers from the 3rd International School and Symposium on Advanced Distributed Systems held in Guadalajara, Mexico, January 24–30, 2004. This event was organized by the teams made up of members of CINVESTAV Guadalajara, CUCEI, the Computer Science Department of the Centre of Research and Advances Studies at the CUCEA campus of the University of Guadalajara, Mexico, the University of Rostock, Germany and ITESO, Guadalajara. The ISSADS symposium provides a forum for scientists and people from industry to discuss the progress of applications and theory of distributed systems. This year there were over 300 participants from 3 continents, among which about 20 percent came from industry.

The conference program consisted of 25 accepted papers out of 46 submissions and covered several aspects of distributed systems from hardware and system level up to different applications. These papers were selected by a peer review process, in which each paper was evaluated by at least three members of the international program committee.

In addition, the three invited speakers, Adolfo Guzman Arenas, Yakup Parker and Joaquin Vila, presented interesting overviews to current development and research directions in distributed systems. Furthermore, eight tutorials and four industrial forums from IBM, INTEL, HP and SUN enabled the participants to extend their knowledge in selected areas. A panel, which was organized by a team composed of researchers from the Universidad de Guadalajara and focused on traffic control and simulation, also demonstrated the practical application of recent research in distributed systems to the problems of Guadalajara.

At this moment, we would like to say thank you to all the members of the program and organizing committees as well as their teams, and we would like to show our particular gratitude to all those who submitted their papers to ISSADS 2004. Furthermore, we would like to acknowledge the local support from the Council of Science and Research of Jalisco, Mexico and the Jalisco Software Industry. Special thanks are also given to Yuniva Gonzalez and Cynthia Guerrero for their organizational support. We hope that all the participants enjoyed their stay in Mexico and benefited from fruitful discussions and a good time. We look forward to more new participants at the next ISSADS conference to be held again in Guadalajara, Mexico, in January 2005.

May 2004

Félix F. Ramos C.
Herwig Unger
Victor Larios

Program Committee

Chair Félix Francisco Ramos Corchado, CINVESTAV Guadalajara

Co-chair Victor Manuel Larios Rosillo, CUCEA, Universidad de Guadalajara

Editorial chair Herwig Unger, Rostock University, Germany

Scientific Committee

Anbulagan	A. Gelbukh	P. de Saqui Sannes
F. Arbad	A.A. Guzmán	R.M. Pires
G. Babin	G. Juanolet	R. Rajkumar
H.R. Barradas	H. Kihl	F. Ren
J.P. Barthés	J.-L. Koning	G. Román
N. Bennani	P. Kropf	E.E. Scalabrin
T. Böhme	S. Lecomte	S. Tazi
P. Boulanger	A. López	H. Unger
M. Bui	R. Mandiau	J. Vila
L. Chen	E. Moreira	T. Villemur
M. Diaz	S. Murugesan	P. Young-Hwan
D. Donsez	A. N Tchernykh	A. Zekl
K. Drira	Y. Paker	
C.V. Estivill	E.P. Cortéz	

Organization

Public Relations Carolina Mata, CINVESTAV Guadalajara

Logistics Cynthia Guerrero, CINVESTAV Guadalajara

Logistics Jorge Hernández, CINVESTAV Guadalajara

Logistics Yuniva González, CINVESTAV Guadalajara

Lecture Notes in Computer Science

For information about Vols. 1–3028

please contact your bookseller or Springer-Verlag

- Vol. 3139: F. Iida, R. Pfeifer, L. Steels, Y. Kuniyoshi (Eds.), *Embodied Artificial Intelligence*. IX, 331 pages. 2004. (Subseries LNAI).
- Vol. 3133: A.D. Pimentel, S. Vassiliadis (Eds.), *Computer Systems, Architectures, Modeling, and Simulation*. XIII, 562 pages. 2004.
- Vol. 3125: D. Kozen (Ed.), *Mathematics of Program Construction*. X, 401 pages. 2004.
- Vol. 3123: A. Belz, R. Evans, P. Piwek (Eds.), *Generating Language*. X, 219 pages. 2004. (Subseries LNAI).
- Vol. 3120: J. Shawe-Taylor, Y. Singer (Eds.), *Learning Theory*. X, 648 pages. 2004. (Subseries LNAI).
- Vol. 3118: K. Miesenberger, J. Klaus, W. Zagler, D. Burger (Eds.), *Computer Helping People with Special Needs*. XXIII, 1191 pages. 2004.
- Vol. 3116: C. Rattray, S. Maharaj, C. Shankland (Eds.), *Algebraic Methodology and Software Technology*. XI, 569 pages. 2004.
- Vol. 3114: R. Alur, D.A. Peled (Eds.), *Computer Aided Verification*. XII, 536 pages. 2004.
- Vol. 3113: J. Karhumäki, H. Maurer, G. Paun, G. Rozenberg (Eds.), *Theory Is Forever*. X, 283 pages. 2004.
- Vol. 3112: H. Williams, L. MacKinnon (Eds.), *New Horizons in Information Management*. XII, 265 pages. 2004.
- Vol. 3111: T. Hagerup, J. Katajainen (Eds.), *Algorithm Theory - SWAT 2004*. XI, 506 pages. 2004.
- Vol. 3110: A. Juels (Ed.), *Financial Cryptography*. XI, 281 pages. 2004.
- Vol. 3109: S.C. Sahinalp, S. Muthukrishnan, U. Dogrusoz (Eds.), *Combinatorial Pattern Matching*. XII, 486 pages. 2004.
- Vol. 3108: H. Wang, J. Pieprzyk, V. Varadharajan (Eds.), *Information Security and Privacy*. XII, 494 pages. 2004.
- Vol. 3107: J. Bosch, C. Krueger (Eds.), *Software Reuse: Methods, Techniques and Tools*. XI, 339 pages. 2004.
- Vol. 3105: S. Göbel, U. Spierling, A. Hoffmann, I. Iurgel, O. Schneider, J. Dechau, A. Feix (Eds.), *Technologies for Interactive Digital Storytelling and Entertainment*. XVI, 304 pages. 2004.
- Vol. 3104: R. Kralovic, O. Sykora (Eds.), *Structural Information and Communication Complexity*. X, 303 pages. 2004.
- Vol. 3103: K. Deb (Ed.), *Genetic and Evolutionary Computation - GECCO 2004*. XLIX, 1439 pages. 2004.
- Vol. 3102: K. Deb (Ed.), *Genetic and Evolutionary Computation - GECCO 2004*. L, 1445 pages. 2004.
- Vol. 3101: M. Masoodian, S. Jones, B. Rogers (Eds.), *Computer Human Interaction*. XIV, 694 pages. 2004.
- Vol. 3100: J.F. Peters, A. Skowron, J.W. Grzymala-Busse, B. Kostek, R.W. Świniarski, M.S. Szczuka (Eds.), *Transactions on Rough Sets I*. X, 405 pages. 2004.
- Vol. 3099: J. Cortadella, W. Reisig (Eds.), *Applications and Theory of Petri Nets 2004*. XI, 505 pages. 2004.
- Vol. 3098: J. Desel, W. Reisig, G. Rozenberg (Eds.), *Lectures on Concurrency and Petri Nets*. VIII, 849 pages. 2004.
- Vol. 3097: D. Basin, M. Rusinowitch (Eds.), *Automated Reasoning*. XII, 493 pages. 2004. (Subseries LNAI).
- Vol. 3096: G. Melnik, H. Holz (Eds.), *Advances in Learning Software Organizations*. X, 173 pages. 2004.
- Vol. 3094: A. Nürnberger, M. Detyniecki (Eds.), *Adaptive Multimedia Retrieval*. VIII, 229 pages. 2004.
- Vol. 3093: S.K. Katsikas, S. Gritzalis, J. Lopez (Eds.), *Public Key Infrastructure*. XIII, 380 pages. 2004.
- Vol. 3092: J. Eckstein, H. Baumeister (Eds.), *Extreme Programming and Agile Processes in Software Engineering*. XVI, 358 pages. 2004.
- Vol. 3091: V. van Oostrom (Ed.), *Rewriting Techniques and Applications*. X, 313 pages. 2004.
- Vol. 3089: S. Berardi, M. Coppo, F. Damiani (Eds.), *Applied Cryptography and Network Security*. XIV, 510 pages. 2004.
- Vol. 3086: M. Odersky (Ed.), *ECOOP 2004 – Object-Oriented Programming*. XIII, 611 pages. 2004.
- Vol. 3085: S. Berardi, M. Coppo, F. Damiani (Eds.), *Types for Proofs and Programs*. X, 409 pages. 2004.
- Vol. 3084: A. Persson, J. Stirna (Eds.), *Advanced Information Systems Engineering*. XIV, 596 pages. 2004.
- Vol. 3083: W. Emmerich, A.L. Wolf (Eds.), *Component Deployment*. X, 249 pages. 2004.
- Vol. 3080: J. Desel, B. Pernici, M. Weske (Eds.), *Business Process Management*. X, 307 pages. 2004.
- Vol. 3079: Z. Mammeri, P. Lorenz (Eds.), *High Speed Networks and Multimedia Communications*. XVIII, 1103 pages. 2004.
- Vol. 3078: S. Cotin, D.N. Metaxas (Eds.), *Medical Simulation*. XVI, 296 pages. 2004.
- Vol. 3077: F. Roli, J. Kittler, T. Windeatt (Eds.), *Multiple Classifier Systems*. XII, 386 pages. 2004.
- Vol. 3076: D. Buell (Ed.), *Algorithmic-Number Theory*. XI, 451 pages. 2004.
- Vol. 3074: B. Kuijpers, P. Revesz (Eds.), *Constraint Databases and Applications*. XII, 181 pages. 2004.
- Vol. 3073: H. Chen, R. Moore, D.D. Zeng, J. Leavitt (Eds.), *Intelligence and Security Informatics*. XV, 536 pages. 2004.

- Vol. 3072: D. Zhang, A.K. Jain (Eds.), *Biometric Authentication*. XVII, 800 pages. 2004.
- Vol. 3071: A. Omicini, P. Petta, J. Pitt (Eds.), *Engineering Societies in the Agents World*. XIII, 409 pages. 2004. (Subseries LNAI).
- Vol. 3070: L. Rutkowski, J. Siekmann, R. Tadeusiewicz, L.A. Zadeh (Eds.), *Artificial Intelligence and Soft Computing - ICAISC 2004*. XXV, 1208 pages. 2004. (Subseries LNAI).
- Vol. 3068: E. André, L. Dybkjær, W. Minker, P. Heisterkamp (Eds.), *Affective Dialogue Systems*. XII, 324 pages. 2004. (Subseries LNAI).
- Vol. 3067: M. Dastani, J. Dix, A. El Fallah-Seghrouchni (Eds.), *Programming Multi-Agent Systems*. X, 221 pages. 2004. (Subseries LNAI).
- Vol. 3066: S. Tsumoto, R. Słowiński, J. Komorowski, J.W. Grzymała-Busse (Eds.), *Rough Sets and Current Trends in Computing*. XX, 853 pages. 2004. (Subseries LNAI).
- Vol. 3065: A. Lomuscio, D. Nute (Eds.), *Deontic Logic in Computer Science*. X, 275 pages. 2004. (Subseries LNAI).
- Vol. 3064: D. Bienstock, G. Nemhauser (Eds.), *Integer Programming and Combinatorial Optimization*. XI, 445 pages. 2004.
- Vol. 3063: A. Llamas, A. Strohmeier (Eds.), *Reliable Software Technologies - Ada-Europe 2004*. XIII, 333 pages. 2004.
- Vol. 3062: J.L. Pfaltz, M. Nagl, B. Böhlen (Eds.), *Applications of Graph Transformations with Industrial Relevance*. XV, 500 pages. 2004.
- Vol. 3061: F.F. Ramos, H. Unger, V. Larios (Eds.), *Advanced Distributed Systems*. VIII, 285 pages. 2004.
- Vol. 3060: A.Y. Tawfik, S.D. Goodwin (Eds.), *Advances in Artificial Intelligence*. XIII, 582 pages. 2004. (Subseries LNAI).
- Vol. 3059: C.C. Ribeiro, S.L. Martins (Eds.), *Experimental and Efficient Algorithms*. X, 586 pages. 2004.
- Vol. 3058: N. Sebe, M.S. Lew, T.S. Huang (Eds.), *Computer Vision in Human-Computer Interaction*. X, 233 pages. 2004.
- Vol. 3057: B. Jayaraman (Ed.), *Practical Aspects of Declarative Languages*. VIII, 255 pages. 2004.
- Vol. 3056: H. Dai, R. Srikant, C. Zhang (Eds.), *Advances in Knowledge Discovery and Data Mining*. XIX, 713 pages. 2004. (Subseries LNAI).
- Vol. 3055: H. Christiansen, M.-S. Hacid, T. Andreassen, H.L. Larsen (Eds.), *Flexible Query Answering Systems*. X, 500 pages. 2004. (Subseries LNAI).
- Vol. 3054: I. Crnkovic, J.A. Stafford, H.W. Schmidt, K. Wallnau (Eds.), *Component-Based Software Engineering*. XI, 311 pages. 2004.
- Vol. 3053: C. Bussler, J. Davies, D. Fensel, R. Studer (Eds.), *The Semantic Web: Research and Applications*. XIII, 490 pages. 2004.
- Vol. 3052: W. Zimmermann, B. Thalheim (Eds.), *Abstract State Machines 2004*. Advances in Theory and Practice. XII, 235 pages. 2004.
- Vol. 3051: R. Berghammer, B. Möller, G. Struth (Eds.), *Relational and Kleene-Algebraic Methods in Computer Science*. X, 279 pages. 2004.
- Vol. 3050: J. Domingo-Ferrer, V. Torra (Eds.), *Privacy in Statistical Databases*. IX, 367 pages. 2004.
- Vol. 3049: M. Bruynooghe, K.-K. Lau (Eds.), *Program Development in Computational Logic*. VIII, 539 pages. 2004.
- Vol. 3047: F. Oquendo, B. Warboys, R. Morrison (Eds.), *Software Architecture*. X, 279 pages. 2004.
- Vol. 3046: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.K. Tan, O. Gervasi (Eds.), *Computational Science and Its Applications - ICCSA 2004*. LIII, 1016 pages. 2004.
- Vol. 3045: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.K. Tan, O. Gervasi (Eds.), *Computational Science and Its Applications - ICCSA 2004*. LIII, 1040 pages. 2004.
- Vol. 3044: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.K. Tan, O. Gervasi (Eds.), *Computational Science and Its Applications - ICCSA 2004*. LIII, 1140 pages. 2004.
- Vol. 3043: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.K. Tan, O. Gervasi (Eds.), *Computational Science and Its Applications - ICCSA 2004*. LIII, 1180 pages. 2004.
- Vol. 3042: N. Mitrou, K. Kontovasilis, G.N. Rouskas, I. Iliadis, L. Merakos (Eds.), *NETWORKING 2004, Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications*. XXXIII, 1519 pages. 2004.
- Vol. 3040: R. Conejo, M. Urretavizcaya, J.-L. Pérez-de-la-Cruz (Eds.), *Current Topics in Artificial Intelligence*. XIV, 689 pages. 2004. (Subseries LNAI).
- Vol. 3039: M. Bubak, G.D.v. Albada, P.M. Slood, J.J. Dongarra (Eds.), *Computational Science - ICCS 2004*. LXVI, 1271 pages. 2004.
- Vol. 3038: M. Bubak, G.D.v. Albada, P.M. Slood, J.J. Dongarra (Eds.), *Computational Science - ICCS 2004*. LXVI, 1311 pages. 2004.
- Vol. 3037: M. Bubak, G.D.v. Albada, P.M. Slood, J.J. Dongarra (Eds.), *Computational Science - ICCS 2004*. LXVI, 745 pages. 2004.
- Vol. 3036: M. Bubak, G.D.v. Albada, P.M. Slood, J.J. Dongarra (Eds.), *Computational Science - ICCS 2004*. LXVI, 713 pages. 2004.
- Vol. 3035: M.A. Wimmer (Ed.), *Knowledge Management in Electronic Government*. XII, 326 pages. 2004. (Subseries LNAI).
- Vol. 3034: J. Favela, E. Menasalvas, E. Chávez (Eds.), *Advances in Web Intelligence*. XIII, 227 pages. 2004. (Subseries LNAI).
- Vol. 3033: M. Li, X.-H. Sun, Q. Deng, J. Ni (Eds.), *Grid and Cooperative Computing*. XXXVIII, 1076 pages. 2004.
- Vol. 3032: M. Li, X.-H. Sun, Q. Deng, J. Ni (Eds.), *Grid and Cooperative Computing*. XXXVIII, 1112 pages. 2004.
- Vol. 3031: A. Butz, A. Krüger, P. Olivier (Eds.), *Smart Graphics*. X, 165 pages. 2004.
- Vol. 3030: P. Giorgini, B. Henderson-Sellers, M. Winikoff (Eds.), *Agent-Oriented Information Systems*. XIV, 207 pages. 2004. (Subseries LNAI).
- Vol. 3029: B. Orchard, C. Yang, M. Ali (Eds.), *Innovations in Applied Artificial Intelligence*. XXI, 1272 pages. 2004. (Subseries LNAI).

Table of Contents

International School and Symposium on Advanced Distributed Systems

Myths, Beliefs and Superstitions about the Quality of Software and of Its Teaching <i>Adolfo Guzman Arenas</i>	1
Enhancing a Telerobotics Java Tool with Augmented Reality <i>Nancy Rodriguez, Luis Jose Pulido, and Jean-Pierre Jessel</i>	9
VIBES: Bringing Autonomy to Virtual Characters <i>Stéphane Sanchez, Hervé Luga, Yves Duthen, and Olivier Balet</i>	19
An Overview of the VIRTUOSI Toolkit <i>Alcides Calsavara, Agnaldo K. Noda, and Juarez da Costa Cesar Filho</i>	31
Assessing the Impact of Rapid Serial Visual Presentation (RSVP): A Reading Technique <i>Barbara Beccue and Joaquin Vila</i>	42
An Open Multiagent Architecture to Improve Reliability and Adaptability of Systems <i>Edson Scalabrin, Deborah Carvalho, Elaini Angelotti, Hilton de Azevedo, and Milton Ramos</i>	54
Toward a Generic MAS Test Bed <i>Juan Salvador Gómez Álvarez, Gerardo Chavarín Rodríguez, and Victor Hugo Zaldivar Carrillo</i>	67
State Controlled Execution for Agent-Object Hybrid Languages <i>Ivan Romero Hernandez and Jean-Luc Koning</i>	78
Cognitive Agents and Paraconsistent Logic <i>Elaini Simoni Angelotti and Edson Emílio Scalabrin</i>	91
A Multiagent Infrastructure for Self-organized Physical Embodied Systems: An Application to Wireless Communication Management <i>Jean-Paul Jamont and Michel Occello</i>	105
Tlachтли: A Framework for Soccer Agents Based on GeDa-3D <i>Francisco Ocegueda, Roberto Sánchez, and Félix Ramos</i>	118
Evaluating Location Dependent Queries Using ISLANDS <i>Marie Thilliez and Thierry Delot</i>	125

Conceptual Information Retrieval

*Emerson L. dos Santos, Fabiano M. Hasegawa, Bráulio C. Ávila,
and Fabrício Enembreck* 137

Semantic Search Engines

Alcides Calsavara and Glauco Schmidt 145

The Internal-Local-Remote Dependency Model for Generic Coordination
in Distributed Collaboration Sessions

José Martín Molina Espinosa, Jean Fanchon, and Khalil Drira 158

About the Value of Virtual Communities in P2P Networks

German Sakaryan, Herwig Unger, and Ulrike Lechner 170

Search in Communities: An Approach Derived from the Physic Analogue
of Thermal Fields

Herwig Unger and Markus Wulff 186

A Component-Based Design Approach
for Collaborative Distributed Systems

Francisco Moo-Mena and Khalil Drira 197

Architecture for Locating Mobile CORBA Objects
in Wireless Mobile Environment

Mayank Mishra 207

Integration of Load Balancing into a Parallel Evolutionary Algorithm

*Miguel Castro, Graciela Román, Jorge Buenabad, Alma Martínez,
and John Goddard* 219

Random Distributed Self-stabilizing Structures Maintenance

Thibault Bernard, Alain Bui, and Olivier Flauzac 231

A New On-Line Scheduling Algorithm for Distributed Real-Time System

Mourad Hakem and Franck Butelle 241

Facing Combinatory Explosion in NAC Networks

Jérôme Leboeuf Pasquier 252

Multiple Correspondences and Log-linear Adjustment in E-commerce

María Beatriz Bernábe Loranca and Luis Antonio Olsina Santos 261

A Distributed Digital Text Accessing and Acquisition System

Adolfo Guzmán Arenas and Victor-Polo de Gyves 274

Author Index 285

Myths, Beliefs and Superstitions about the Quality of Software and of Its Teaching

Adolfo Guzman Arenas

Centro de Investigacion en Computacion (CIC)
Instituto Politecnico Nacional, Mexico
a.guzman@acm.org

Abstract. It is a surprise to see how, as years go by, two activities so germane to our discipline, (1) the creation of quality software, and (2) the quality teaching of software construction, and more generally of Computer Science, are surrounded or covered, little by little, by beliefs, attitudes, “schools of thought,” superstitions and fetishes rarely seen in a scientific endeavor. Each day, more people question them less frequently, so that they become “everyday truths” or “standards to observe and demand.” I have the feeling that I am minority in this wave of believers and beliefs, and that my viewpoints are highly unpopular. I dare to express them because I fail to see enough faults in my reasoning and reasons, and because perhaps there exist other “believers” not so convinced about these viewpoints, so that, perhaps, we will discover that “the imperator had no clothes, he was naked.”

1 Myths and Beliefs about the Production of Quality Software

This section lists several “general truths,” labeled A, B..., G concerning quality of software, and tries to ascertain whether they are reasonable assertions (“facts,” sustainable opinions) or myths.

1.1 About Measuring Software Quality

A. It Is Possible to Measure the Main Attributes that Characterize Good Quality Software. The idea here is that software quality can be characterized by certain attributes: reliability, flexibility, robustness, comprehension, adaptability, modularity, complexity, portability, usability, reuse, efficiency... and that it is possible to measure each of these, and therefore, characterize or measure the quality of the software under examination. To ascertain whether point A is a fact or a myth, let us analyze three facets of it.

1) *It is possible to measure above attributes subjectively*, asking their opinion to people who have used the software in question.

Comment 1. Opinions by Experienced Users Are Reliable. That is, (1) is not a myth, but something real. It is easy to agree that a program can be characterized by above attributes (or similar list). Also, it is convincing that the opinions of a group of

qualified users respect to the quality, ergonomics, portability... of a given software are reliable and worth to be taken into account (subjective, but reliable opinions).

2) Another practice is to try to measure above attributes *objectively*, by measuring surrogate attributes if the real attribute is difficult to measure [Myth B below].

Comment 2. Measuring Surrogate Attributes. To measure the height of a water tank when one wishes to measure its volume, is risky. Objective (accurate) measurements of surrogate attributes may be possible, but to think that these measures are proportional to the real attribute, is risky. “If you can not measure beauty of a face, measure the length of the nose, the color of eyes...” If you can not measure the complexity of a program, measure the degree of nesting in its formulas and equations, and say that they are directly related. More in my comments to Myth B.

3) Finally, instead of measuring the quality of a piece of software, go ahead and measure the quality of the *manufacturing process* of such software: if the building process has quality, no doubt the resulting software should have quality, too (Discussed below as Myth C).

Comment 3. To Measure the Process, instead of Measuring the Product. In old disciplines (manufacturing of steel hinges, leather production, wine production, cooking...) where there are hundred of years of experience, and which are based in established disciplines (Physics, Chemistry...), it is possible to design a process that guarantees the quality of the product. A process to produce good leather, let us say. And it is also possible to (objectively) measure the quality of the resulting product. And to *adapt* the process, modifying it to fix errors (deviations) in the product quality: for instance, to obtain a more elastic leather. Our problem is that *it is not possible* to do that with software. We do not know what processes are good to produce good quality software. We do not know what part of the process to change in order, let us say, to produce software with less complexity, or with greater portability. More in my comments to Myth C.

B. There Exists a Reliable Measurement for Each Attribute. *For each attribute to be measured, there exists a reliable, objective measurement* that can be carried out. The idea is that, if the original attribute is difficult to measure,¹ measure another attribute, correlated to the first, and report the (second) measurement as proportional or a substitute for the measure of the original attribute.

1. Reliability (reliable software, few errors): measure instead the number of error messages in the code. The more, the less errors that software has.
2. Flexibility (malleability to different usage, to different environments) or adaptability: measure instead the number of standards to which that software adheres.
3. Robustness (few drastic failures, the system rarely goes down): measure through tests and long use (Subjective measurement).
4. Comprehension (ability to understand what the system does): measure instead the extent of comments in source code, and the size of its manuals.

¹ Or we do not know how to measure it.

5. The size of a program is measured in bytes, the space it occupies in memory (This measurement has no objection, we measure what we want to measure).
6. Speed of execution is measured in seconds (This measurement has no objection, we measure what we want to measure).
7. Modularity: count the number of source modules forming it.
8. Program complexity (how difficult it is to understand the code): measure instead the level of nesting in expressions and commands (“cyclomatic complexity”).
9. Portability (how easy it is to port a software to a different operating system): ask users that have done these portings (Subjective measurement).
10. Program usability (it is high when the program brings large added value to our work. “It is essential to have it.”): measure the percentage of our needs that this program covers (Subjective measurement).
11. Program reuse: measure how many times (parts of) this program have been used in other software development projects. (Objective measurement, but only obtained in hindsight).
12. Ease of use (ergonomics) characterizes programs that are easy to learn, tailored to our intuitive ways to carry out certain tasks. Measure instead the quantity of screens that interact with the user, and their sophistication.

Comment 4. Measuring Surrogate Attributes. These “surrogate measurements” can produce irrelevant figures for the quality that we are really trying to measure. For instance, the complexity of a program will be difficult to measure using point 8, for languages that use no parenthesis for nesting. For instance, it is not clear that a software with long manuals is easier to comprehend (point 4). To measure the temperature of a body when one wants to measure the amount of heat (calories) in it, is incorrect and will produce false results. A very hot needle has less heat than a lukewarm anvil.

Comment 5. It is true that in the production of other goods, say iron hinges, is easy to list the qualities that a good hinge must possess: hardness, resistance to corrosion... And it is also easy to objectively measure those qualities. Why is it difficult, then, to measure the equivalent quantities about software? Because hinges have been produced before Pharaohnic times, humankind has accumulated experience on this, and because its manufacture is based on Physics, which is a consolidated science more than 2,000 years old. Physics has defined units (mass, hardness, tensile strength...) capable of objective measurement. More over, Physics often gives us equations ($f = ma$) that these measurements need to obey. In contrast, Computer Science has existed only for 60 years, and thus almost all its dimensions (reliability, ease of use...) are not susceptible (yet) of objective measurements. Computer Science is not a science yet, it is an art or a craft.² Nevertheless, it is tempting to apply to software characterization (about its quality, say), methods that belong and are useful in these more mature disciplines, but that are not (yet) applicable in our emerging science. We are not aware that methods that work in leather production, do not work

² Remember the title of the book “The Art of Computer Programming” of Donald C. Knuth. In addition, we should not be afraid that our science begins as an art or a craft. Visualize Medicine when it was only 60 years old: properties of lemon tea were just being discovered. And physicians talked for a long time of fluids, effluvia, bad air, and witchcraft. With time, our discipline will become a science.

in software creation. Indeed, it is useful at times to talk of software *creation*, not of software production, to emphasize the fact that software building is an art, dominated by inspiration, good luck... (see Comment 7).

1.2 Measuring the Process instead of Measuring the Product

An indirect manner to ascertain the quality of a piece of software, is to review the quality of the process producing it.

C. Measuring the Quality of the Process, Not the Product Quality. Instead of measuring the quality of the software product, let us measure the quality of its construction process. To have a good process implies to produce quality software.

Comment 6. It is tempting to claim that a “good” process produces good quality software, and therefore, deviations of programmers with respect to the given process should be measured and corrected. The problem here is that it is not possible to say which process will produce good quality software. For instance, if I want to produce portable software, what process should I introduce, versus if what I want to emphasize is ease of use? Thus, the definition of the process becomes very subjective, an act of faith. Processes are used that sound and look reasonable, or that have been used in other places with some success. Or that are given by some standard or international committee. “If so many people use them, they must be good.” We need to recognize that our discipline is not (yet) a science nor an Engineering discipline, where one can design a process that guarantees certain properties in the resulting product, much in the same manner that the time and temperature of an oven can be selected to produce hinges of certain strength. Instead, our discipline is more of an art or a craft, where inspiration counts, “to see how others do it,” “to follow the school of Prof. Wirth,” to follow certain rites and traditions or tics that a programmer copied (perhaps unconsciously) from his teacher.

Comment 7. A more contrasting manner to see that certain measurement processes are not applicable to certain areas, is to examine an art, such as Painting or Symphony Composition. Following the rules of the hard disciplines (manufacturing of hinges), we would first characterize the quality symphonies as those having sonority, cadence, rhythm... Here, measuring those qualities becomes (as in software) subjective. Then, we would establish the rules that govern the *process* of fabrication of symphonies (by observing or asking notable composers, say Sergei Prokoffiev): the pen needs to have enough ink, use thick point; the paper must have a brightness no less than x, its thickness must be at least z; it must be placed on the desk forming an angle not bigger than 35 degrees. Light shall come from the left shoulder. Certainly, these rules will not hurt. But there is no guarantee that anybody that follows them will produce great quality symphonies, even if the very same rules in hands of Prokoffiev produce excellent results, over and over.

D. If You Have a Controlled Process, You Will Produce Good Quality Software.

It is easy to know when you have a “good” (reasonable) process. It is easy to design a “good” process to produce software.

Comment 8. On the other hand, it is not really known which processes will produce easy-to-use software, which other processes will produce portable software, or software with good real-time properties, etc. The “process design” bears thus little relation to the goal: to produce a software product with this and that features. The problem resembles that of hospital surgeons in the pre-Pasteurian period, when bacteria were not yet discovered. Many people who underwent surgery died, full of infection and pus, without anybody knowing why. Of course, it was easy to measure the quality of a product of a surgery process: “birth-giving woman died of septicemia.” Therefore, processes were designed to make sure that patients with surgery would not die: when coming to work, surgeons should pray to Saint Diego. Then, hang from your neck a chain of garlic bulbs. Then, wash your hands. You shall not commit surgery during days with nights having full moon... These rules certainly did not hurt (did not produce worse results), but they were not very related to the quality of the final results. Once bacteria were discovered, the rules were simplified, fine tuned and complemented with others: “wash your knives,” “disinfect your hands.” It is my impression that, in software creation, we are in a pre-Luis Pasteur epoch, and that we invent rules and processes “to have one at hand,” but that the results (the quality of the resulting software) of these processes have little to do with the invented process, and with its fulfillment or lack thereof.

E. It Is Necessary to Create “Quality Champions,” Quality Committees, and other human organizations whose goal is “to promote quality (of software).” Generally, a committee of this type (1) generates norms and rules saying how the construction of software is to be handled (regulations about the process; they *define* the process), including formats that certain intermediate and final documents (manuals, say) shall have, and (2) it observes if the programming team follows the rules (1), seeking to correct deviations.

Comment 9. These committees, since they do not know for sure neither how to measure the quality of the product (Myths A and B) nor how to alter the fabrication process if certain output attributes are unacceptable (Myth D), end up becoming hindrances and stereotyped bureaucracies. What they can demand (and they do) from the programming and design team is adherence to the process invented by said committee (or copied from an international organization). If they adhere and follow the process, “that is good,” and (by faith) “good quality software shall be the result.” If the team deviates, that is bad; offenders should be punished and be blamed for the bad quality of the results. This is equivalent to have, in a pre-Pasteurian hospital (see Comment 8) a committee that, watching that this week more patients died of general infection than in the previous week, strengthens its efforts and detects surgeons that did not pray to Saint Diego, while others hanged from their necks garlic bulbs that were not fresh. Let us reprehend these offenders, and less patients shall die.

F. Attitude Matters. The right mind-set towards quality shall permeate and impregnate each coder. The designer or programmer must be constantly thinking about quality, must have faith in that he will produce good quality software; he shall watch that the quality of his works be above a (high) minimum.

Comment 10. This is an act of faith, that certainly will not hurt. But it helps little. Software confection should not be based on faith or beliefs. Certainly, it helps somewhat that a programmer says each morning “today I am going to produce high quality software, I am sure of that,” much in the same manner as a pre-Pasteurian surgeon said “Today, no one of my patients undergoing surgery will die; today, no one of my patients undergoing surgery will die.” With respect to the idea that a programmer “shall watch the quality of his production,” this is commendable, but it is certainly difficult, since *it is difficult to measure software quality*, even if the person measuring is the software builder.

1.3 The Myth of Standards

G. Adhesion to Standards Means High Quality Software. “If we do software construction following the rules dictated by a standards organization, we will be producing good quality software.” “Following software construction standards ensures the quality of the process and the quality of the resulting software.” That is to say, of the many processes that we could follow when creating software, let us use one that is part of a norm or standard (preferably, an international one), or let us follow the process used by a company that produces good quality software (Oracle, say).

Comment 11. Nothing wrong can be perceived in this practice. It is as if the surgeons of a low quality hospital (of Comment 8) decide to copy the surgery process of The Hospital of the Holy Virgin Mary, which has low post-surgery mortality. Or if I want to use Prokoffiev's “rules for composing good symphonies” (Comment 7). No evil will come out of this. Nevertheless, subjectivity and the scanty relation between these “preferred” procedures and the quality of the resulting software must be clear, as Comment 8 explains.

2 Myths and Beliefs about the Quality of Teaching in Computer Science

We now examine how quality in Computer Science schools is measured.

2.1 It Is Enough to Measure the Product

It seems very reasonable and obvious (but let us examine it) to measure the quality of the product, in order to ascertain its quality.

H. Measure the Product and See if It Is of Good Quality. If I make a test or examination to two young undergraduate alumni of different Computer Science schools, and one of them knows more computer science than the other, certainly the person knowing more is of better quality. Example: the organism “Ceneval” in Mexico, who does just that.

Comment 12. This measurement is “almost right,” except that it does not measure the great *obsolescence* in our field. I estimate that the mean half-life³ of a computer science concept is about 5 years. That is, every five years, half of what we know becomes useless; not because we forget concepts or because they were inadequately learnt. Just because these concepts are no longer useful, they are obsolete: bubble memories, remote job entry, punch tape... The measurement of point H simply measures the *today* quality, the quality *as measured today*. What will happen in five or ten years with alumnus 1 versus alumnus 2? May be alumnus 1 still displays useful knowledge, while alumnus 2 (the more knowledgeable today) no longer has. One rusted faster than the other. That is, alumni formation (specially in high obsolescence fields, such as ours –this is due to its youth, scantily 60 years old) depends on two factors: (a) the basic, theoretical knowledge, which will enable our alumni to *keep acquiring knowledge* through his productive life, outside College; and (b) the “today” knowledge, the knowledge that is “fashion today” (in 2004, objects, UML, Java, say) that will allow them to become immediately productive. “To go out into the sugar cane field and start cutting cane.” Knowledge acquired in College is like the quality of a machete, which depends on two attributes: its *temper*, that permits it several resharpenings along its productive life (and we need to prepare alumni capably of a productive life of 40 years), and the *sharpness* of the cutting edge (which allows immediate productivity, “to go out and start cutting cane.” My problem with procedure H is that only measures the *sharpness* of the edge, the “today usefulness.” Add measurements every five years (longitudinal studies), or add measurements (today) about the degree of theory and basic subjects (those that can hardly change in 50 years, say) that the alumnus has; this knowledge is what renders him resistant to obsolescence.

I. Quality of a College Is Measured by the Quality of Its Alumni.

Comment 13. Again, this is “almost true.” Certainly, we tend to give high quality to those Colleges that produce high-quality alumni. But often these schools require the entering students to *have already high quality*. At entrance time. High entrance requirements. I only accept the best. Obviously, these people will exit school better prepared than students at another College who, due to incoming deficiencies, finish their studies less well prepared. To be fair, the quality of a school should be measured by the *added value*. That is, measure the student at entrance and exit times, and perform a subtraction.

2.2 To Measure Quality of the Alumni, It Is Sufficient to Measure Quality of the Process

The certification process of a College implies measuring the teaching process that it carries. There is a thought that good (certified) colleges produce high quality alumni.

³ The half-life of a radioactive substance is the time taken by that substance’s mass to decay to half its original mass.