Nicolas Guelfi
Anthony Savidis (Eds.)

LNCS 3943

Rapid Integration of Software Engineering Techniques

Second International Workshop, RISE 2005
Heraklion, Crete, Greece, September 2005
Revised Selected Papers

Springer

Nicolas Guelfi Anthony Savidis (Eds.)

Rapid Integration of Software Engineering Techniques

Second International Workshop, RISE 2005
Heraklion, Crete, Greece, September 8-9, 2005
Revised Selected Papers

Springer

Volume Editors

Nicolas Guelfi
University of Luxembourg
Faculty of Science, Technology and Communication
1359 Luxembourg, Luxembourg
E-mail: nicolas.guelfi@uni.lu

Anthony Savidis
Foundation for Research and Technology - Hellas (FORTH)
Institute of Computer Science
GR-70013 Heraklion, Crete, Greece
E-mail: as@ics.forth.gr

Library of Congress Control Number: 2006925116

CR Subject Classification (1998): D.2, F.3, K.6.1, K.6.3

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-34063-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-34063-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11751113 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3943

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Preface

RISE 2005 (http://rise2005.ics.forth.gr/) was the second annual inter-national workshop of the ERCIM (European Research Consortium for Informatics and Mathematics - http://www.ercim.org/) Working Group on Rapid Integration of Software Engineering techniques (RISE - http://rise.uni.lu/). RISE is an international forum for researchers and practitioners interested in the advancement and rapid application of novel, integrated, or practical software engineering approaches being part of a methodological framework, which apply to the development of new or evolving applications and systems. RISE provides an opportunity to present and discuss the latest research results and ideas in the rapid and effective integration of software engineering techniques. Target application domains of interest to RISE include:

- Web-based software systems
- Mobile communication systems
- High-availability or mission-critical systems
- Resilient business and grid applications
- Ambient intelligence environments
- Embedded systems and applications
- User interface development
- Development environments
- Electronic entertainment
- Enterprise computing and applications

In particular, RISE 2005 focused on an open and inclusive set of key software engineering domains, which formed the focal point of the workshop, including, but not limited to:

- Software and system architectures
- Software reuse
- Software testing
- Software model checking
- Model-driven design and testing techniques
- Model transformation
- Requirements engineering
- Lightweight or practice-oriented formal methods
- Software processes and software metrics
- Automated software engineering
- Design patterns
- Design by contract
- Defensive programming
- Software entropy and software re-factoring

- Extreme programming
- Agile software development
- Programming languages
- Software dependability and trustworthiness

This second RISE workshop was, like RISE 2004, very successful and fruitful, almost doubling the number of submissions and of accepted papers, and leading to a high-quality two-day technical programme. Overall, RISE 2005 accomplished its objective to move a step forward in setting the ground towards a targeted, prestigious and open inter-national forum of high-quality research and development results in the arena of rapid integration of software engineering techniques.

All papers submitted to this workshop were peer reviewed by at least two members of the International Programme Committee. Acceptance was based primarily on originality and scientific contribution. Out of the 43 submissions received, 19 were selected for inclusion in the workshop technical programme. Six chaired thematic sessions were organized over a single workshop track, covering many aspects of the integration of complementary mature software engineering techniques. This year, the submissions addressed areas such as modelling safety case evolution, practical approaches in model mapping, context-aware service composition, techniques for representing product line core assets for automation, formal development of reactive fault-tolerant systems, stepwise feature introduction in practice, programming languages, aspects and contracts. The technical discussions that followed the paper presentations were tape recorded, and the transcripts can be found on the RISE 2005 website (http://rise2005.ics.forth.gr/).

The keynote speech was delivered by Bertrand Meyer, Chair of Software Engineering at ETH Zurich, Founder and Chief Architect of Eiffel Software in California, and inventor of the Design by Contract™ method. His inspiring talk as well as active participation during the discussion sessions contributed to the overall success of the workshop.

The editors wish to thank the keynote speaker and the authors for their valuable contribution in making RISE 2005 a truly outstanding event of high-quality contributions, as well as all the Programme Committee members who actively participated in the review process in a timely and quality fashion. Finally, we are thankful to Springer LNCS for publishing the RISE 2005 proceedings under this volume (the RISE 2004 post-proceedings are published as Springer LNCS Vol. 3475).

September 2005

Nicolas Guelfi
Anthony Savidis

Organization

RISE 2005 was organized by the Institute of Computer Science (ICS), Foundation for Research and Technology Hellas (FORTH).

Programme Chairs

Anthony Savidis	ICS-FORTH - *Programme and Organization Chair*
Nicolas Guelfi	FNR, Univ. Luxembourg - *General Workshop Chair*

International Programme Committee

Arve Aagesen Finn	NTNU, Norway
Avgeriou Paris	Fraunhofer IPSI Concert, Germany
Bertolino Antonia	CNR-ISTI, Italy
Bicarregui Juan	CCLRC, UK
Bolognesi Tommaso	CNR-ISTI, Italy
Born Marc	Fraunhofer FOKUS, Germany University of Geneva, Switzerland
Carrez Cyril	NTNU, Norway
Dony Christophe	LIRMM, France
Fitzgerald John	DCS, Newcastle, UK
Greenough Chris	CCLRC-CSE CG, UK
Guelfi Nicolas	FNR, Luxembourg
Haajanen Jyrki	VTT, Finland
Issarny Valérie	INRIA, France
Klint Paul	CWI, The Netherlands
Mistrik Ivan	Fraunhofer IPSI IM, Germany
Mens Tom	FWO-Mons-Hainaut University TM, Belgium
Moeller Eckhard	Fraunhofer FOKUS, Germany
Monostori Laszlo	SZTAKI, Hungary
Pimentel Ernesto	SpaRCIM, Spain
Reggio Gianna	DISI, Genoa, Italy
Romanovsky Alexander	DCS, Newcastle, UK
Rosener Vincent	FNR, Luxembourg
Savidis Anthony	CS-FORTH, Greece
Schieferdecker Ina	Fraunhofer FOKUS, Germany

Local Organization

Maria Papadopoulou	ICS-FORTH
George Paparoulis	ICS-FORTH
Maria Bouhli	ICS-FORTH
Yannis Georgalis	ICS-FORTH

Sponsoring Institutions

This workshop was supported by ERCIM and by the ERCIM Working Group (WG) on Rapid Integration of Software Engineering Techniques (RISE).

Lecture Notes in Computer Science

For information about Vols. 1–3882

please contact your bookseller or Springer

Vol. 3987: M. Hazas, J. Krumm, T. Strang (Eds.), Location- and Context-Awareness. X, 289 pages. 2006.

Vol. 3984: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part V. XXV, 1045 pages. 2006.

Vol. 3983: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part IV. XXVI, 1191 pages. 2006.

Vol. 3982: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part III. XXV, 1243 pages. 2006.

Vol. 3981: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part II. XXVI, 1255 pages. 2006.

Vol. 3980: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part I. LXXV, 1199 pages. 2006.

Vol. 3979: T.S. Huang, N. Sebe, M.S. Lew, V. Pavlović, T. Kölsch, A. Galata, B. Kisačanin (Eds.), Computer Vision in Human-Computer Interaction. XII, 121 pages. 2006.

Vol. 3978: B. Hnich, M. Carlsson, F. Fages, F. Rossi (Eds.), Recent Advances in Constraints. VIII, 179 pages. 2006. (Sublibrary LNAI).

Vol. 3970: T. Braun, G. Carle, S. Fahmy, Y. Kocheryavy (Eds.), Wired/Wireless Internet Communications. XIV, 350 pages. 2006.

Vol. 3968: K.P. Fishkin, B. Schiele, P. Nixon, A. Quigley (Eds.), Pervasive Computing. XV, 402 pages. 2006.

Vol. 3967: D. Grigoriev (Ed.), Computer Science – Theory and Applications. XVI, 684 pages. 2006.

Vol. 3964: M. Ü. Uyar, A.Y. Duale, M.A. Fecko (Eds.), Testing of Communicating Systems. XI, 373 pages. 2006.

Vol. 3960: R. Vieira, P. Quaresma, M.d.G.V. Nunes, N.J. Mamede, C. Oliveira, M.C. Dias (Eds.), Computational Processing of the Portuguese Language. XII, 274 pages. 2006. (Sublibrary LNAI).

Vol. 3959: J.-Y. Cai, S. B. Cooper, A. Li (Eds.), Theory and Applications of Models of Computation. XV, 794 pages. 2006.

Vol. 3958: M. Yung, Y. Dodis, A. Kiayias, T. Malkin (Eds.), Public Key Cryptography - PKC 2006. XIV, 543 pages. 2006.

Vol. 3956: G. Barthe, B. Gregoire, M. Huisman, J.-L. Lanet (Eds.), Construction and Analysis of Safe, Secure, and Interoperable Smart Devices. IX, 175 pages. 2006.

Vol. 3955: G. Antoniou, G. Potamias, C. Spyropoulos, D. Plexousakis (Eds.), Advances in Artificial Intelligence. XVII, 611 pages. 2006. (Sublibrary LNAI).

Vol. 3954: A. Leonardis, H. Bischof, A. Pinz (Eds.), Computer Vision – ECCV 2006, Part IV. XVII, 613 pages. 2006.

Vol. 3953: A. Leonardis, H. Bischof, A. Pinz (Eds.), Computer Vision – ECCV 2006, Part III. XVII, 649 pages. 2006.

Vol. 3952: A. Leonardis, H. Bischof, A. Pinz (Eds.), Computer Vision – ECCV 2006, Part II. XVII, 661 pages. 2006.

Vol. 3951: A. Leonardis, H. Bischof, A. Pinz (Eds.), Computer Vision – ECCV 2006, Part I. XXXV, 639 pages. 2006.

Vol. 3950: J.P. Müller, F. Zambonelli (Eds.), Agent-Oriented Software Engineering VI. XVI, 249 pages. 2006.

Vol. 3947: Y.-C. Chung, J.E. Moreira (Eds.), Advances in Grid and Pervasive Computing. XXI, 667 pages. 2006.

Vol. 3946: T.R. Roth-Berghofer, S. Schulz, D.B. Leake (Eds.), Modeling and Retrieval of Context. XI, 149 pages. 2006. (Sublibrary LNAI).

Vol. 3945: M. Hagiya, P. Wadler (Eds.), Functional and Logic Programming. X, 295 pages. 2006.

Vol. 3944: J. Quiñonero-Candela, I. Dagan, B. Magnini, F. d'Alché-Buc (Eds.), Machine Learning Challenges. XIII, 462 pages. 2006. (Sublibrary LNAI).

Vol. 3943: N. Guelfi, A. Savidis (Eds.), Rapid Integration of Software Engineering Techniques. X, 289 pages. 2006.

Vol. 3942: Z. Pan, R. Aylett, H. Diener, X. Jin, S. Göbel, L. Li (Eds.), Technologies for E-Learning and Digital Entertainment. XXV, 1396 pages. 2006.

Vol. 3939: C. Priami, L. Cardelli, S. Emmott (Eds.), Transactions on Computational Systems Biology IV. VII, 141 pages. 2006. (Sublibrary LNBI).

Vol. 3936: M. Lalmas, A. MacFarlane, S. Rüger, A. Tombros, T. Tsikrika, A. Yavlinsky (Eds.), Advances in Information Retrieval. XIX, 584 pages. 2006.

Vol. 3935: D. Won, S. Kim (Eds.), Information Security and Cryptology - ICISC 2005. XIV, 458 pages. 2006.

Vol. 3934: J.A. Clark, R.F. Paige, F.A. C. Polack, P.J. Brooke (Eds.), Security in Pervasive Computing. X, 243 pages. 2006.

Vol. 3933: F. Bonchi, J.-F. Boulicaut (Eds.), Knowledge Discovery in Inductive Databases. VIII, 251 pages. 2006.

Vol. 3931: B. Apolloni, M. Marinaro, G. Nicosia, R. Tagliaferri (Eds.), Neural Nets. XIII, 370 pages. 2006.

Vol. 3930: D.S. Yeung, Z.-Q. Liu, X.-Z. Wang, H. Yan (Eds.), Advances in Machine Learning and Cybernetics. XXI, 1110 pages. 2006. (Sublibrary LNAI).

Vol. 3929: W. MacCaull, M. Winter, I. Düntsch (Eds.), Relational Methods in Computer Science. VIII, 263 pages. 2006.

Vol. 3928: J. Domingo-Ferrer, J. Posegga, D. Schreckling (Eds.), Smart Card Research and Advanced Applications. XI, 359 pages. 2006.

Vol. 3927: J. Hespanha, A. Tiwari (Eds.), Hybrid Systems: Computation and Control. XII, 584 pages. 2006.

Vol. 3925: A. Valmari (Ed.), Model Checking Software. X, 307 pages. 2006.

Vol. 3924: P. Sestoft (Ed.), Programming Languages and Systems. XII, 343 pages. 2006.

Vol. 3923: A. Mycroft, A. Zeller (Eds.), Compiler Construction. XIII, 277 pages. 2006.

Vol. 3922: L. Baresi, R. Heckel (Eds.), Fundamental Approaches to Software Engineering. XIII, 427 pages. 2006.

Vol. 3921: L. Aceto, A. Ingólfsdóttir (Eds.), Foundations of Software Science and Computation Structures. XV, 447 pages. 2006.

Vol. 3920: H. Hermanns, J. Palsberg (Eds.), Tools and Algorithms for the Construction and Analysis of Systems. XIV, 506 pages. 2006.

Vol. 3918: W.K. Ng, M. Kitsuregawa, J. Li, K. Chang (Eds.), Advances in Knowledge Discovery and Data Mining. XXIV, 879 pages. 2006. (Sublibrary LNAI).

Vol. 3917: H. Chen, F.Y. Wang, C.C. Yang, D. Zeng, M. Chau, K. Chang (Eds.), Intelligence and Security Informatics. XII, 186 pages. 2006.

Vol. 3916: J. Li, Q. Yang, A.-H. Tan (Eds.), Data Mining for Biomedical Applications. VIII, 155 pages. 2006. (Sublibrary LNBI).

Vol. 3915: R. Nayak, M.J. Zaki (Eds.), Knowledge Discovery from XML Documents. VIII, 105 pages. 2006.

Vol. 3914: A. Garcia, R. Choren, C. Lucena, P. Giorgini, T. Holvoet, A. Romanovsky (Eds.), Software Engineering for Multi-Agent Systems IV. XIV, 255 pages. 2006.

Vol. 3910: S.A. Brueckner, G.D.M. Serugendo, D. Hales, F. Zambonelli (Eds.), Engineering Self-Organising Systems. XII, 245 pages. 2006. (Sublibrary LNAI).

Vol. 3909: A. Apostolico, C. Guerra, S. Istrail, P. Pevzner, M. Waterman (Eds.), Research in Computational Molecular Biology. XVII, 612 pages. 2006. (Sublibrary LNBI).

Vol. 3908: A. Bui, M. Bui, T. Böhme, H. Unger (Eds.), Innovative Internet Community Systems. VIII, 207 pages. 2006.

Vol. 3907: F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, C. Cotta, R. Drechsler, E. Lutton, P. Machado, J.H. Moore, J. Romero, G.D. Smith, G. Squillero, H. Takagi (Eds.), Applications of Evolutionary Computing. XXIV, 813 pages. 2006.

Vol. 3906: J. Gottlieb, G.R. Raidl (Eds.), Evolutionary Computation in Combinatorial Optimization. XI, 293 pages. 2006.

Vol. 3905: P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt (Eds.), Genetic Programming. XI, 361 pages. 2006.

Vol. 3904: M. Baldoni, U. Endriss, A. Omicini, P. Torroni (Eds.), Declarative Agent Languages and Technologies III. XII, 245 pages. 2006. (Sublibrary LNAI).

Vol. 3903: K. Chen, R. Deng, X. Lai, J. Zhou (Eds.), Information Security Practice and Experience. XIV, 392 pages. 2006.

Vol. 3902: R. Kronland-Martinet, T. Voinier, S. Ystad (Eds.), Computer Music Modeling and Retrival. XI, 275 pages. 2006.

Vol. 3901: P.M. Hill (Ed.), Logic Based Program Synthesis and Transformation. X, 179 pages. 2006.

Vol. 3900: F. Toni, P. Torroni (Eds.), Computational Logic in Multi-Agent Systems. XVII, 427 pages. 2006. (Sublibrary LNAI).

Vol. 3899: S. Frintrop, VOCUS: A Visual Attention System for Object Detection and Goal-Directed Search. XIV, 216 pages. 2006. (Sublibrary LNAI).

Vol. 3898: K. Tuyls, P.J. 't Hoen, K. Verbeeck, S. Sen (Eds.), Learning and Adaption in Multi-Agent Systems. X, 217 pages. 2006. (Sublibrary LNAI).

Vol. 3897: B. Preneel, S. Tavares (Eds.), Selected Areas in Cryptography. XI, 371 pages. 2006.

Vol. 3896: Y. Ioannidis, M.H. Scholl, J.W. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, C. Boehm (Eds.), Advances in Database Technology - EDBT 2006. XIV, 1208 pages. 2006.

Vol. 3895: O. Goldreich, A.L. Rosenberg, A.L. Selman (Eds.), Theoretical Computer Science. XII, 399 pages. 2006.

Vol. 3894: W. Grass, B. Sick, K. Waldschmidt (Eds.), Architecture of Computing Systems - ARCS 2006. XII, 496 pages. 2006.

Vol. 3893: L. Atzori, D.D. Giusto, R. Leonardi, F. Pereira (Eds.), Visual Content Processing and Representation. IX, 224 pages. 2006.

Vol. 3892: A. Carbone, N.A. Pierce (Eds.), DNA Computing. XI, 440 pages. 2006.

Vol. 3891: J.S. Sichman, L. Antunes (Eds.), Multi-Agent-Based Simulation VI. X, 191 pages. 2006. (Sublibrary LNAI).

Vol. 3890: S.G. Thompson, R. Ghanea-Hercock (Eds.), Defence Applications of Multi-Agent Systems. XII, 141 pages. 2006. (Sublibrary LNAI).

Vol. 3889: J. Rosca, D. Erdogmus, J.C. Príncipe, S. Haykin (Eds.), Independent Component Analysis and Blind Signal Separation. XXI, 980 pages. 2006.

Vol. 3888: D. Draheim, G. Weber (Eds.), Trends in Enterprise Application Architecture. IX, 145 pages. 2006.

Vol. 3887: J.R. Correa, A. Hevia, M. Kiwi (Eds.), LATIN 2006: Theoretical Informatics. XVI, 814 pages. 2006.

Vol. 3886: E.G. Bremer, J. Hakenberg, E.-H.(S.) Han, D. Berrar, W. Dubitzky (Eds.), Knowledge Discovery in Life Science Literature. XIV, 147 pages. 2006. (Sublibrary LNBI).

Vol. 3885: V. Torra, Y. Narukawa, A. Valls, J. Domingo-Ferrer (Eds.), Modeling Decisions for Artificial Intelligence. XII, 374 pages. 2006. (Sublibrary LNAI).

Vol. 3884: B. Durand, W. Thomas (Eds.), STACS 2006. XIV, 714 pages. 2006.

Vol. 3883: M. Cesana, L. Fratta (Eds.), Wireless Systems and Network Architectures in Next Generation Internet. IX, 281 pages. 2006.

Table of Contents

Invited Keynote Speech (Abstract)

Doing More with Contracts: Towards Automatic Tests and Proofs

Bertrand Meyer

ETH Zurich (Swiss Federal Institute of Technology),
Department of Computer Science,
CH-8092 Zürich, Switzerland
Bertrand.Meyer@inf.ethz.ch

Abstract. Equipping software with contracts, especially in the case of library components, opens up a whole range of applications. I will describe two of them, part of current work in the chair of software engineering at ETH. The first is automatic, "push-button" testing of contract-equipped components. The second is mathematical proof that such components satisfy their contracts. In both cases the effort is made more interesting by the existence of library versions that are fully contracted" thanks to the use of model classes based on set-theoretical concepts. Both the tests and the proofs apply to actual libraries as used in practical software development.

N. Guelfi and A. Savidis (Eds.): RISE 2005, LNCS 3943, p. 1, 2006.

Using Stepwise Feature Introduction in Practice: An Experience Report

Ralph-Johan Back, Johannes Eriksson, and Luka Milovanov

Turku Centre for Computer Science,
Åbo Akademi University, Department of Computer Science,
Lemminkäisenkatu 14, FIN-20520 Turku, Finland
{backrj, joheriks, lmilovan}@abo.fi

Abstract. Stepwise Feature Introduction is an incremental method and software architecture for building object-oriented system in thin layers of functionality, and is based on the Refinement Calculus logical framework. We have evaluated this method in a series of real software projects. The method works quite well on small to medium sized software projects, and provides a nice fit with agile software processes like Extreme Programming. The evaluations also allowed us to identify a number of places where the method could be improved, most of these related to the way inheritance is used in Stepwise Feature Introduction. Three of these issues are analyzed in more detail here: diamond inheritance, complexity of layering and unit testing of layered software.

1 Introduction

Stepwise Feature Introduction (SFI) [1] is a bottom-up software development methodology based on incremental extension of the object-oriented system with a single new feature at a time. It proposes a layered software architecture and uses Refinement Calculus [2, 3] as the logical framework.

Software is constructed in SFI in thin layers, where each layer implements a specific feature or a set of closely related features. The bottom layer provides the most basic functionality, with each subsequent layer adding more and more functionality to the system. The layers are implemented as class hierarchies, where a new layer inherits all functionality of previous layers by sub-classing existing classes, and adds new features by overriding methods and implementing new methods. Each layer, together with its ancestors, constitutes a fully executable software system.

Layers are added as new features are needed. However, in practice we cannot build the system in this purely *incremental* way, by just adding layer after layer. Features may interact in unforeseen ways, and a new feature may not fit into the current design of the software. In such cases, one must *refactor* the software so that the new feature fits better into the overall design. Large refactorings may also modify the layer structure, e.g. by changing the order of layers, splitting layers or removing layers altogether.

An important design principle of SFI is that each extension should preserve the functionality of all previous layers. This is known as *superposition refinement* [4]. A superposition refinement can add new operations and attributes to a class, and may override

N. Guelfi and A. Savidis (Eds.): RISE 2005, LNCS 3943, pp. 2–17, 2006.

old operations. However, when overriding an old operation, the effect of the old operation on the old attributes has to be preserved (but new attributes can be updated freely). No operations or attributes can be removed or renamed.

Consider as an example a class that provides a simple text widget in a graphical user interface. The widget works only with simple ASCII text. A new feature that could be added as an extension to this widget could be, e.g., formatted text (boldface, italics, underlined, etc). Another possible extension could be a clipboard to support cut and paste. We could carry out both these extensions in parallel and then construct a new class that inherits from both the clipboard text widget and the styles text widget using multiple inheritance (this is called a *feature combination*), possibly overriding some of the operations to avoid undesirable feature interaction. Or, we could first implement the clipboard functionality as an extension of the simple text widget, being careful to preserve all the old features, and then introduce styles as a new layer on top of this. Alternatively, we could first add styles and then implement a clipboard on top of the styles layer. The three approaches are illustrated in Figure 1.

A component is divided into layers in SFI. Layers will often cut across components, so that the same layering structure is imposed on a number of related components. As an example, consider building an editor that displays the text widget. In the first layer we have a simple editor that only displays the simple ASCII text widget. Because of the superposition property of extension this simple editor can in fact also use the `CutAndPaste`, `Styles` or `BetterText` widgets, but it cannot make use of the new features. We need to add some features to the simple editor so that the functionality of the

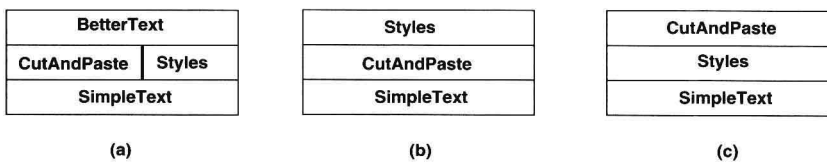

Fig. 1. Alternative extension orders

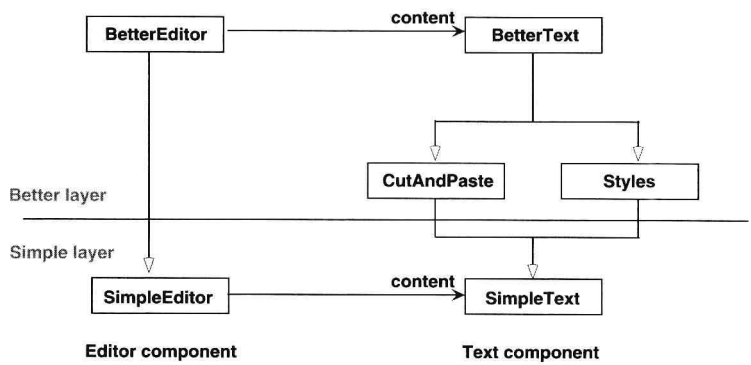

Fig. 2. Interacting components

extended widget can be accessed (menu items for cut and paste, or for formatting, or toolbar buttons for the same purpose). We do this by constructing a new an extension of the simple editor (a better editor), which uses the `BetterText` widget and gives the user access to the new functionality. The situation is illustrated in Figure 2.

The new editor is, however, restricted to only work on the better text widget, because the features it assumes are only available on this level. Hence, there are two layers in the design, the Simple layer and the Better layer.

Stepwise Feature Introduction has been tried out in a number of real software projects. This allows us now to evaluate the merits of this approach and to spot possible drawbacks as well as opportunities for improvement. Our purpose in this paper is to report on these case studies, and to provide a first evaluation of the approach, together with some suggestions on how to improve the method.

The paper is structured as following: Section 2 present the software projects where SFI was applied. We summarize our experience with the methodology in Section 3. In Sections 4–6 we then consider in more detail three interesting issues that arose from our experiments with Stepwise Feature Introduction. The problems with implementing feature combinations using multiple inheritance is discussed in Section 4. The problem of class proliferation is discussed in Section 5, where metaprogramming is considered as one possible way of avoiding unneccessary classes. In Section 6, we show how to adapt unit testing to also test for correct superposition refinement. We end with a short summary and some discussion on on-going and future work.

2 SFI Projects in Gaudi

The software projects where SFI was evauated were all carried out in the Gaudi Software Factory at Åbo Akademi University. The Gaudi Software Factory is an academic environment for building software for the research needs and for carrying out practical experiments in Software Engineering [5]. Our research group defines the setting, goals and methods to be used in the Factory, but actual construction of the software is done in the factory, following a well-defined software process. The work is closely monitored, and provides a lot of data and measures by wich the software process and its results can be evaluated. The software process used in Gaudi is based on agile methods, primarily *Extreme Programming* [6], together with our own extensions.

We will here describe four software projects where Stepwise Feature Introduction was used throughout. The settings for all these projects were similar: the software had to be built with a tight schedule, and the Gaudi software process had to be followed. The programmers employed for these projects (4–6 persons) were third-fifth year students majoring in Computer Science or related areas. Each project had a customer who had final saying on the functionality to be implemented. The projects were also supervised by a coach (a Ph.D. student specializing in Software Engineering), whose main task was to guide the use of the software process and to control that the process was being followed. There has also been one industrial software project [7] with SFI, but this is outside the scope of this paper, as it was not carried out in the Gaudi Factory, and the software process used was not monitored in a sufficiently systematic manner.

All of the projects used SFI, but the ways in which the method was applied differed from project to project. We describe the projects in chronological order below. For each project, we present the goals: both for the software product that was to be built, and for the way in which SFI was to be evaluated in this project. We give a general overview of the software architecture, show how SFI was implemented, what went right and what went wrong, and discuss the lessons learned from the project.

2.1 Extreme Editor

The Extreme Editor project [8] was the first application of SFI in practice. It ran for three months during the summer 2001 and involved six programmers. The programming language of the project was Python [9]. The software product to be built was an outlining editor which became a predecessor for the Derivation Editor described in Section 2.2. The goal for the project was to obtain the first experience from practical application of SFI with a dynamically typed programming language. There were no technical guidelines for the application of SFI except that the extension mechanism for classes (the feature introduction—Section 1) should be inheritance.

Figure 3 shows the layered architecture of the Extreme Editor. There were eight layers in the system. Each layer introduced new functionality into the system, without breaking the old features. The software was structured into these layers in an ad hoc way. A new layer extended its predecessor by inheriting its corresponding classes and possibly introducing one or more new classes. There were no physical division of the

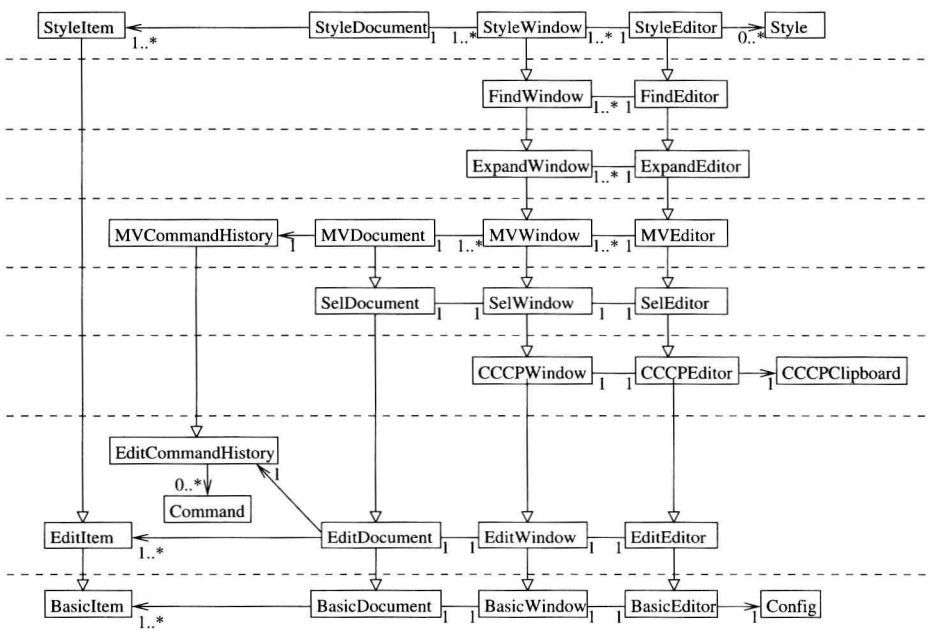

Fig. 3. The layers of the editor