Victor R. Basili, H. Dieter Rombach,
Richard W. Selby (editors):

Experimental Software Engineering Issues

Dagstuhl-Seminar-Report; 47
14.09.-18.09.92 (9238)

ISSN 0940-1121

Tel.: +49-6871 - 2458
Fax: +49-6871 - 5942

Das Internationale Begegnungs- und Forschungszentrum für Informatik (IBFI) ist eine gemeinnützige GmbH. Sie veranstaltet regelmäßig wissenschaftliche Seminare, welche nach Antrag der Tagungsleiter und Begutachtung durch das wissenschaftliche Direktorium mit persönlich eingeladenen Gästen durchgeführt werden.

Bezugsadresse: Geschäftsstelle Schloß Dagstuhl
Informatik, Bau 36
Universität des Saarlandes
W - 6600 Saarbrücken
Germany
Tel.: +49 -681 - 302 - 4396
Fax: +49 -681 - 302 - 4397
e-mail: office@dag.uni-sb.de

Experimental Software Engineering Issues: A Critical Assessment and Future Directions

H. Dieter Rombach, Universität Kaiserslautern, Germany
Victor R. Basili, University of Maryland, USA
Richard R. Selby, University of California, USA

Since its inception in 1968, software engineering has struggled to find its identity. Today, we can identify three different approaches to study of the discipline of software engineering in the research community: the mathematical or formal methods approach, the system building approach, and the empirical studies group. Within the mathematical or formal methods camp, the emphasis is on finding better formal methods and languages and software development is viewed as a mathematical transformation process. Within the system building group, the emphasis is on finding better methods for structuring large systems and software development is viewed as a creative task which cannot be controlled other than through rigid constraints on the resulting product.

Within the empirical studies group, the emphasis is on understanding the strengths and weaknesses of methods and tools in order to tailor them to the specific goals of a particular software project.

The purpose of this workshop was to gather together those member of the software engineering community who support an engineering approach, based upon empirical studies to provide an interchange of ideas and paradigms for research.

This empirical approach is made difficult when one observes that in practical software organizations, project contexts (i.e., project goals and environmental characteristics) vary from project to project. Thus, no single technology or method can be expected to work well in all contexts and observing software phenomena out-of-context, seems to be doomed to fail. As part of the learning process, we need to characterize and understand the project context and understand the various phenomena relative to those context and learn in an incremental and evolutionary manner. We need to replicate experiments in different contexts to fully understand the nature of the various phenomena and be able to build models to facilitate learning.

Improvement oriented approaches, that take into account the evolutionary and experimental nature of software have recently been suggested as a framework for studying the relationship between product and knowledge engineering.

This framework bears the potential of integrating the efforts of the mathematical analysis, system building, and empirical studies approaches in a promising way. This improvement approach is based on the use of empirical technology for building models. Formal methods as well as system building technology can be elevated to the level of useful technologies from an engineering perspective if augmented with knowledge of their effectiveness based on empirical evidence.

We have only begun to understand the experimental nature of software engineering, the role of empirical studies and measurement within software engineering, and the mechanisms need-

ed to apply them successfully. Seminar discussion was focused on assessing past accomplishments within the experimental software engineering community and proposing necessary future steps. The discussion included various topics of interest to experimental software engineering:

(1) Identifying the appropriate paradigms for software engineering (2) Understanding the range of different contexts for empirical studies in software engineering

(3) Devising the appropriate procedures and mechanisms for empirical studies

(4) Guiding the use of empirical data to build or improve existing software models

(5) Identifying appropriate concepts and mechanisms for packaging existing models for reuse across projects

(6) Proposing appropriate means for distributing experimental ideas to practitioners and students

It was clear for the workshop discussion that a lot has been achieved in the past from both the practitioner's as well as the researcher's perspective.

Nevertheless, much more research and infusion of experimental software engineering technology into practice is needed. Advances in research and practice are highly interdependent.

Advances of the state-of-the-practice require more mature research results; advances in research - especially in the area of experimental software engineering - require early feedback from practical experience.

The following sections provide a snapshot of some of the most important achievements identified by the seminar attendees:

PAST ACHIEVEMENTS (Practitioner's Perspective):

Each workshop attendee was in a position to report about empirical studies and/or measurement programs and subjective results ranging from increased understanding of certain software engineering phenomena or the improvements of real-world software processes. Most of these achievements are not well documented or documentation is company-confidential. Achievements are typically based on empirical studies in very specific contexts and cannot be generalized due to the variability of contexts across different organizations.

The workshop attendees felt that in order to live up to the theme of the workshop, we needed to come up with "documented" results which can be analyzed by others and can be used to convey the potential value of empirical work in terms of measurable benefits. One outstanding sample of achievements possible through empirical studies within an experimental framework was reported from the Software Engineering Laboratory (SEL) at NASA's Goddard Space Flight Center and will be mentioned here as an example:

- People oriented technologies appear to be most effective (e.g., inspections, Cleanroom) as opposed to automated tools
- Commonly accepted complexity measures are not very meaningful in the SEL domain
- Ada software costs more to develop but less to deliver because of reuse (multiple experiments)

- Inspections by stepwise abstraction reading are more effective and more cost effective than basic functional or structural testing techniques (Basili, Selby)
- Models/relationships developed can be incorporated into the management process (e.g., manager's handbook) and supported (e.g., SME - an automated environment)
- The error rates in development projects were reduced significantly through the use of Cleanroom (e.g., 33% on a series of 3 projects)
- Environmental heritage/context/legacy is the dominant impact on processes and products (e.g., use of Ada over time)
- Productive reuse is driven by process reuse and packaging of design - NOT by code packaging (e.g., Ada/OOD experiments)
- Some effective processes in other contexts are inappropriate for the SEL (e.g., IV&V)
- Personnel variation in productivity are tremendous (factors of 3 or 4 for large systems; factors of up to 15 in small systems)
- Design structure (strength/compiling) is a good predictor of module defects (Card/Page/McGarry)

Examples from other local environments were reported but cannot be reported in this brief summary.

PAST ACHIEVEMENTS (Researcher's Perspective):

The most important lessons learned about empirical studies and measurement included a broad agreement regarding the experimental nature of software engineering. It was the common view that empirical studies are needed as a driver for learning and improvement, and that such studies (1) need to be performed with a goal and hypothesis in mind and (2) that context characteristics need to be taken into account when interpreting measurement data.

The most frequently occurring themes during the discussion and on the questionnaires were

- Software engineering research needs to be driven by empirical studies
- Metrics and data in isolation (i.e., without context) are useless
- No single set of metrics is universally best
- Sound empirical approaches are essential
- Empirical studies have produced results in local contexts; we have not been able to generalize local results

Again, the example lessons learned within the SEL are given as an example:

- The purpose of experimentation should be for self-improvement and self-understanding rather than inter-organization and inter-country comparison
- Expectation of N to 1 improvement in productivity over finite time (5 to 10 years) is baseless and won't happen
- We are most effective when using multiple processes based on context (e.g., the SEL uses Fortran/Functional-Decomposition-based design/reuse-oriented waterfall, Ada/OOD/reuse-oriented waterfall, and Cleanroom)

- Each of the above technologies and processes had to be tailored to the environment
- Understanding (baselining) is absolutely mandatory as a first step (before planning, controlling, technology transfer)
- Process definition and clarification of the empirical process is mandatory for successful experimentation and measurement (i.e., Doctor heal thyself)
- The process improvement paradigm is equally important for the software development task and the experimentation/data collection task
- The process for empirical studies has to be well defined and improved
- There must be a goal/rational for data collection
- Data by itself provides minimal, most likely erroneous or detrimental insights
- Measurement data has intrinsic imprecision and inconsistency, and incompletely represented context. This drives the need to study trends.
- More data does not necessarily mean better results (i.e., National databases for measurement data are a waste of time and resources)
- Packaging of experience is the key to success - but is rarely done effectively.
- Packaging (i.e., development of local standards) needs to be experience driven (e.g, 2167A is incomplete approach if it is not based upon experience)
- Effective cookbooks can be developed for particular domains (e.g., SEL measurement handbook, SEL management handbook)
- Experimentation requires two identifiable, separate (but cooperating) organizational infrastructure components, which both involve cost
  - overhead to project (noise -- 2%)
  - analysis and synthesis of data (8-10%)
  - support (quality assurance, databases,...)
- Developers treat data collection/experimentation as an annoyance only
- Infusion of significant process change (e.g., Ada, Cleanroom, OOD) requires 5 to 10 years to mature within an organization.

The attendees agreed, during the discussion of important topics for future consideration that there is a need for the continued development of experimental, empirical and infrastructure technology. There is also a need for more wide-spread practical applications of existing empirical methods in order to build models of interesting software product and process aspects, and characterize the potential and limits of existing technologies as a basis for systematic software engineering.

In summary, important accomplishments towards the manifestation of empirical studies as a major subdiscipline of software engineering, and its influence in moving software engineering as a whole towards a true engineering discipline were identified. By the end of the workshop, most of the attendees acknowledged that now they feel part of a true community of empirically oriented software engineers. In order to foster that sense of community, future gatherings like this were suggested, and a e-mail list for people interested in empirical software engineering research was suggested.

NOTE: Since October 1992 such an e-mail list (empirical-se@informatik.uni-kl.de) exists!

The Experimental Paradigm in Software Engineering

Victor R. Basili, University of Maryland, College Park

There is a need to understand software and engineering. Software is inherently complex; and there is a lack of well defined primitives or components of the artifact and the discipline. We need good inductive and deductive paradigms to build, analyze and evaluate models of the software process and products, various aspects of the environment, e.g. people, organization, and the interactions of these models. The experimental paradigm requires the building of laboratory environments for experimenters, which are useful to developers, and allow us to package models. This can be done and has been done at NASA/GSFC in the Software Engineering Laboratory (SE) and has been effective.

Profile of an Artifact Assessment Capability

William W. Agresti, The MITRE Corporation, McLean, Virginia

We outline features of an engineering software artifact assessment capability to analyze evolving Ada designs. Experience developing this capability have led to our speculation about the role and practice of empirical software engineering. We discuss the role of identifying leading indicators and "speaking with data." Viewing our industrial software projects as "naturally" occurring experiments and possibly using them in large, simple trials study are suggested.

Problems in Modeling the Software Development Process as an Adventure Game

Jochen Ludewig, Institut für Informatik, Universität Stuttgart

In our group at Stuttgart University, we are working on a project named SESAM (= Software Engineering Simulation using Animated Models.)

The goal of SESAM is to provide a Simulator which can be used like an adventure game, or a flight simulator. Its user plays the role of a software project manager, who has to control his or her project so that it is finished within schedule and budget, delivering software of the required level of quality.

For such a simulator, we need a model (e.g. a set of differential equations) describing the change of state variables over time, which may be influenced by other state variables, and by the user's actions. Such a model (which is actually a theory of the software development process) does not exist today.

Building a series of prototypes, which are based on increasingly sophisticated hypotheses, we try to get to a model (= a simulator) which can be used for teaching Software Engineering.

Support of Experimentation by Measurement Theory

Horst Zuse, Technische Universität Berlin

During the last years much attention has been directed toward the measurement of the properties and the complexity of software. The major goal using software measures is to get

reliable software. an objective representation of the properties of software and the software development process by numbers, and a prediction which factors of software complexity have been developed in order to determine the static complexity of single programs (intra-modular complexity) and of entire software systems (inter-modular complexity) during the phases of the software life-cycle.

Since software complexity cannot be defined mathematically, it is necessary to make experiments in order to get correlations between software metrics and errors or metrics and factors of software maintenance. Validation of software metrics is another important topic which is close connected to correlations between attributes of objects.

However, to validate metrics and to interpret correlations between attributes of software is not an easy task. We think, that measurement theory can help here to get better results. The reason for recommending the use of measurement theory for a support of experiments, correlations and validation is that measurement theory deals with the "connection" of the empirical world with the numerical world. Measurement theory gives hypotheses about reality.

Software Engineering as an Organizational Challenge

Günther R. Koch, 2i Industrial Informatics GmbH Freiburg

The corresponding European project to SEI's Assessment Methodology Project is BOOTSTRAP (funded under ESPRIT). The basic hypotheses we constructed from "market research" inputs is that the software (engineering) exists in the first phase is a managerial and organizational crisis, then comes methodology and only in third line technology. Thus our investigation method is to assess software engineering units in these three dimensions. Different from SEI BOOTSTRAP conclude a "quality profile" per assessed organizational unit and uses this profile for stimulating self improvement processes. From this experience we are motivated to start in depth research on how to model maturity of organizations avoiding "absolute" ordinal scales (as by SEI).

Fundamental Role of Measurement in Software Engineering

Norm Fenton, Centre for Software Reliability, City University London

There are many reasons why we need to use measurement in software engineering. For example, we need to use measurement if we wish to assess the reliability of our products, the productivity of personnel, the relative costs of different process methods, or to make accurate predictions of costs/schedules. However, one of the most important uses of measurement is to assess the efficacy of software engineering methods/techniques/tools. The whole profession has been plagued by the constant introduction of new proposed "technological fixes", some of whose efficacy is demonstrated by any more than anecdotal evidence. Thus software practitioners are at the mercy of "salesmen" or self-appointed experts when they come to choose new methods/tools. We believe it is possible to assess objectively the efficacy of methods; using case studies this is done by measuring the extent to which the method has been applied, the 'quality' (notably reliability, maintainability) of the product which result, and the cost of applying the method. We are currently using this approach to assess the efficacy of standards. (SMARTIE project)

Manny Lehman, Imperial College, London

The first issue must be to define Software engineering. These is a whole spectrum of viewpoints. Mine, which I regard of fundamental importance is based on a distinction between broad definition of programming and programmers on the one hand and software engineering and software engineers on the other. The former are product engineers responsible for the development and evolution of specific systems or system elements. The latter on the other hand, are concerned with the processes by which software is developed, maintained satisfactory and evolved and the support of those processes is methods and tools. They are process engineers. Clear the terminology is not, of itself important. But the existence of these two roles the different but complementary contribution that they wake to software development and to the increasing exploitation of computer technology must be recognized, understood and its implication applied. The "end" in software technology is the software product which must be satisfactory in application and remain satisfactory as (as a consequence of feedback and system exogenous change) the software must be changed and evolved. Software maintenance is the maintenance of user satisfaction, and of the validity of the assumption set embedded in software. Maintaining satisfaction, the quality of the product in relation to its application domain is the need and the good. The evolution process is the means whereby we achieve the goal. The quality of a product can, in general, be no better than the quality by the process that produced it.

Reuse of Models

Stuart Feldman Bellcore, Morristown, NJ

On the main assigned topic of packaging for reuse, my main point is that models should be created and packaged for a purpose; more artifacts (models, process descriptions, etc.) are thought to be reusable than are reused. The first step in reuse is finding something that meets the need and deciding to investigate it further. Later steps include detailed checking of suitability and actual instantiation.

The modelable components can be products, processes, or mixtures (such as executable data).

The most important problem is that first step: making clear why a major part should be utilized. If a process model does not come with appropriate descriptive information, references to earlier uses and limitations, and perhaps tools to simplify application, it will probably not be chosen. Pragmatists tend to look for strong evidence that application will be of direct value, leading to results that are better, soon, cheaper, or surer. If product data do not come with some sort of certification or indication of usability, they will also be ignored.

Later, more detailed information is needed to verify that a component actually meets the current need or can be modified to do so. Process models are unlikely to be so numerous in the next few years that sophisticated query or storage facilities will be needed. Libraries of objects and other code components do demand such facilities.

There are many examples of successfully reused product data, including mathematics libraries, widget collections, and other GUI interfaces. Major subsystems and other libraries will typically be reused in specific industrial settings. Processes are commonly reused when embedded in a tool; many people do not even think of these applications as process reuse. Much rarer is reuse of descriptions of processes, except those embedded in textbooks or local rule books.

Software Product Research

There is a crying need for credible information on software artifacts. At this late date, we do not have information on statistics of static and dynamic structure, architecture and design relations, failure data, and so forth, for a documented variety of contexts. Without such information, tool research is seriously hampered and software research is difficult. An effort should be initiated to organize such data from a variety of industrial sources, with uniform definitions and statistics conventions. Bellcore would contribute to such an effort.

Process-and-Context-Model Based Measurements

Nazim H. Madhavji, School of Computer Science, McGill University, Montreal, Canada

Measurement of a software development process is an important way to understand and improve its quality. Ther is an overwhelming evidence, however, that measurements were carried out at all, are predominantly applied to entities of informally defined or undefined software processes. In addition, the context of the process is rarely documented. Consequently, there are: inconsistencies in data gathered in processes; difficulties in determining reasons for such inconsistencies; concerns regarding the validity of the interpretations, especially when the context is not clear; invalid measurement points (w.r.t. project objectives) especially when processes have changed; and other such problems in software projects.

Our hypothesis is that underlying a process measurement programme should be defined process models so that measurements have their rationale made explicit in the models. The relationship between process models and measurements is analogous to the relationship between software requirements and code. There are many anticipated benefits of the model-based measurement programme. However, new problems that need to be solved to make this approach viable include: management of process models (w.r.t. fruitful representation of enacted processes); construction and use of an effective change mechanism to manage changes in process models and processes, and assess the impact of changes on measurements.

Model Reuse and Technology Transfer

Albert Endres, IBM Development Laboratory, Böblingen, Germany

Quantitative models are a widely accepted means to predict and to control software quality and development productivity. If these models are updated to reflect the positive effect of new processes or new technologies, and if the same models are reused by several projects, this will make the advantage of the new technology visible and will create pressure to adopt it. Thus, reuse of models becomes a vehicle for technology transfer.

In my talk, I illustrated the above using a defect removal model as an example.

Norman F. Schneidewind, Naval Postgraduate School, Monterey, CA

We explain why it is important to validate metrics for use on multiple projects and how the risk of doing so can be assessed. We also point out that by prototyping the measurement plan on a project, we can eliminate the risk inherent in the process of validating metrics on one project and applying them on another project, where the two projects may have significant differences in application and domain environment characteristics. In order to support the application of metrics across multiple projects, there must be reuse of methodologies, metrics, and metrics processes. A metrics methodology is reusable when there is a process associated with the

methodology that can be applied across projects. Both the metrics and the process for applying metrics must be reusable Metrics are reusable when validated metrics can be applied across projects, using the methodology. An example is given of assessing risk by using the confidence limits of a metric to evaluate the consequences of best-case and worst-case outcomes of using metrics on multiple projects.

Kevin Wentzel, Hewlett Packard Laboratories, Palo Alto, CA

Models for use in software engineering include analytical, process, organizational and technological models. These models are best developed, tested and tuned in an experimental environment which mixes investigators with practical users of the models. Models should be linked or integrated into domain specific kits oriented toward user's goals to be transferred and used effectively. Application of the experimental paradigm to software engineering research is difficult. Controls are almost impossible to establish in realistic situations. The involvement of humans with their varying interests, abilities and motivations make experiments difficult to evaluate. But, the mix of experiments and practitioners is essential if we want our results to be applicable to more than a research laboratory.

Establishing the Fundamentals of Software Engineering

Dan Hoffman, University of Victoria BC Canada

There is a crippling lack of agreement as to what constitutes the fundamental techniques of Software Engineering. In industry, there is little use of explicitly defined methods. Academic proposals, though promising, have seen little industrial use and evaluation. Pilot projects in industry can help establish the fundamentals by showing which techniques are effective in various situations. The critical question is: how should these projects be organized and instrumental so that the results are convincing and widely applicable?

Software Engineering is not Engineering - and maybe never will be

Bev Littlewood, Centre for Software Reliability, City University London

It seems necessary (albeit perhaps not sufficient) that we are able to predict and control before we can claim that we are engineering our systems. Our ability to do this is presently limited, and seems not to have improved significantly in the past ten years. Some reasons for this are

- dearth of adequate metrics for attributes of interest (e.g. complexity)

- very incomplete identification of the attributes that may be important - resulting in confounding of factors and invalid extrapolation of results to moved circumstances

- poor models for the relationships between metrics/attributes

- generally poor validation: most s/w metrics research is presented and justified to potential users via mere anecdote

One exception to this dismal picture is software reliability growth modeling. In particular, here there are now sophisticated validation techniques that allow a user to know whether (or not) the reliability predictness are trustworthy on his/her particular project.

The downside is that the scope of these reliability models is very restricted. The lesson may be that, if we learn to walk before we run, we may be able to obtain a scientifically respectable (but limited) theory.

Case studies

Barbara Kitchenham, NCC

I believe that experimentation in software engineering covers a variety of approaches. These can be characterized as

- formal experiments based on the hypothetico-deductive paradigm
- case studies which are specific projects aimed at trialling some method/tool
- survey which are analyses of large database aimed at detecting trends

Case studies are attractive to industry because they allow a technology/tool/method to be investigated in the context of the company. I am interested in whether we can improve the confidence we can have in the results of case studies.

The most important issue is to have some means of interpreting results i.e. a basis for comparison. In my experience, a basis for comparison could be one of three types:

i) comparison with a baseline where a baseline could be constructed from measurements derived from other similar projects using current methods (i.e. a baseline must be quite large 7+ projects)

ii) comparison with a matched control project, where the control project uses the current method.

iii) comparison within a project. This can be done if the technology under investigation applies to components (documents/modules). In this case the technology/method can be applied to some components (preferably selected at random) and the remaining components can be developed in the "normal" way.

I think we should also use the standard terminology and techniques of experimental design to assist planning, running and analyzing of case studies. We must state our hypothesis in order to determine what response variables we should measure. We should also use state variable to describe the environment in which the case study in being conducted. Such a characterisation can help with the selection of an appropriate project for a case study and the interpretation of the case study result.

An Axiomatic Model of Program Complexity

Marvin Zelkowitz, University of Maryland, NIST, USA

We have developed an axiomatic model of program complexity that provides minimal criteria that a valid program complexity must adhere to. The basic model differentiates between the relationship between programs and a complexity measure. Complexity is a relation between programs (some, not all programs) and a complexity measure is a real number applied to a given program. So if a relation exists between two programs then their complexity can be compared. All other models generate models generate numbers on a given program, so the complexity between any two programs can be compound.

We have 5 axioms defining this model. Axioms 1 + 2 tell when this complexity relationship must exist, axiom 3 shows that larger programs generally (but not always) get more complex. Axiom 4 is the relationship between complexity and complexity measures and axiom 5 is the distribution on complexity measure values.

We experimented with this model using data from the NASA SEL using the Selby-Porter classification tree model. By using the axioms we reduced the 74 potential measures to 18 effective ones, and by applying classification tree process to these 18 only, we can improve on the original classification tree process and identify high cost modules.

Collection and Distribution of Experimental Software Engineering Data

Warren Harrison, Portland State University, Oregon

A major problem in experimental software engineering is the availability of consistent, reliable and general empirical data. Very few studies are able to use data from other studies due to inconsistent counting rules, data collection procedures etc. .

In order to advance beyond its current state, it appears, these issues must be addressed. One way this can be done is by identifying a common set of tools and processes for all to use which will maintain information on the basic elements and relationships that make up the software object. This information, when maintained in a data base, can be used to produce the metric of the researcher's choice through the use of a standard query script.

The information to be retained must be balanced with the need to be unable to reproduce the source code from the data base to encourage industry to contribute data. The specific information relationships and security needs those affected.

Measurement - A point of view

John Marciniak, CTA, Rockville, Maryland

Measurement ist the most important technical area with respect to software engineering in the 1990's. This is so, in my opinion, because the practice of software developed has advanced to a state of discipline that cannot be advanced without quantitative understanding. The emphasis on process improvement and quality demands the use of measurement.

From a practitioner's viewpoint, there are two aspects of measurement: from a practitioner's view and from a researcher's view.

From the research view, the researcher needs to put into place the scientific basis for measurement, and deliver specific mechanisms that should be used in engineering practice.

From a practitioner's view, treasurement must be integrated into engineering practice. A software engineer needs to have practical measures and the means to apply them to understand and control both the process he/she is using, and to predict the attributes of the product that will be achieved.

Both views are important and necessary. Research provides the basis to provide practical engineering application.

On Experimental Computer Science

Walter F. Tichy, Universität Karlsruhe

Experimental Computer science appears to be in its infancy, and software engineering research is no exception. Consider the following data: a) TOPLAS published a total of 89 papers in the four years starting with 1988. Theory papers accounted for 61% and design papers for 24%. Only 13 papers (15%) had a non-trivial, quantitative evaluation. There were zero papers describing an experiment for testing stated hypothesis.

b) IEEE Transaction on Software Engineering published 92 full-length papers between June 91 an June 92. These were about 35 papers on theory and 40 on design and case studies. Only 17 (18%) included quantitative evaluation or experimental results.

The following conclusions arer suggested: 1) Researchers in the "systems" area may be confusing experimental apparatus (their designs) with results. 2) Computer Scientists produce

too many designs and too few experiments and experimentally verified results. 3) Too few students are being trained in experimentation.

The need for corrective action is great and goes beyond Software Engineering. Computer Scientists should no longer develop systems for their own sake but always perform a careful evaluation of their systems.

Software Engineering Still on the way to an Engineering Discipline

Norber Fuchs, Alcatel Austria - ELIN Research Centre

All areas of business and industry are used to have quantitative support to management decision making. They use measurement in estimation, evaluation and controlling. It is part of the normal way doing measurement.

Different things in software engineering are not really accepted as a support for managers and engineers. Measurements still are used out of context. "Magic number" are still used for proving everything. What we need is a broader acceptance of measurements in a context, for a purpose, under a certain viewpoint, related to goals. To use measurement in that sense we need a defined process with activities and relations defined between them.

Experimental software engineering seems to be one (the only?) way to come there. But it still seems to be a very long way.

Karl Heinrich Möller, Siemens AG, München

Fortschritte in Software Engineering sind nicht mehr möglich, indem einzelne Gebiete noch tiefer erforscht werden, erforderlich ist vielmehr, die Zusammenarbeit über "Grenzen" hinweg: Universitäten, Sozialwissenschaften, Psychologie etc. .

Dieser Workshop war eine ausgezeichnete Gelegenheit auf diesem Weg fortzuschreiten.

Experimental Software Engineering Should Concentrate on Software Evolution

Hausi A. Mueller, University of Victoria, Victoria, B.C. Canada

For the future the software engineering discipline it is critical that we devote sufficient energies to software evolution commensurate with its socio-economic implications. We propose that the area of experimental software engineering focus more on software analysis and understanding to reflect the needs of large, evolving software systems property. Since the results 0f evolution experiments don't necessarily scale up, we argue that the experiments should be performed in sites using large systems, such as telephone switching systems, banking systems, or health information systems which evolve naturally over decades.

Thesis 1: Shift experimental software engineering research efforts from software construction to software analysis and under standing. Thesis 2: Concentrate on sufficiently large evolution experiments using real-world software systems; experiments with toy examples do not scale up. Thesis 3: There will always be old software.

Effective Use of Measurement and Experimentation in Computing Curricula

Stu Zweben, Ohio State University, Columbus, Ohio

Empirical activities are fundamental to the computing discipline and hence must be an integral part of any computing curriculum. The study of empirical paradigms of science and engineering can and should be part of the requirements of an undergraduate computing degree. Courses in basic science, social science, engineering, and statistics can introduce or complement studies of these paradigms within the major. In the major not only courses in software engineering, but also standard courses in algorithms and data structures and advanced courses in areas such as AI and parallel computing, provide opportunities to apply the paradigms. However, one must be careful that students' backgrounds are consistent with the educational objectives expected in courses that employ these paradigms. For example, we should not expect students to be able to evaluate different design methods if they've never really applied them. Educators should be guided in developing appropriate curricula by analyzing the educational objectives, perhaps using a taxonomy such as proposed by bloom.

Qualitative Techniques and Tools for Modeling, Analyzing, and Simulating Software Production

Walt Scacchi, University of Southern California

Empirical studies of software production can benefit in a fundamental way through the use of qualitative research methods and tools. Qualitative methods rely upon observational case studies, field studies, and comparative case/field studies as part of their experimental design. These methods can be used to empirically determine the nature, composition, present and historical context of software products and processes within an organizational setting. Computational tools supporting qualitative research methods are now beginning to appear. Since qualitative research methods are complementary to quantitative measures and traditional experimental research designs, then these computational tools will need to support both knowledge-based techniques for modeling, analyzing and simulating software production as well as statistical techniques. At USC, we have applied qualitative research methods to study more than 20 industrial software projects using computation tools such as these.

The Role of Simulation in Software Engineering Experimentation

A. von Mayrhauser, Colorado State University, Fort Collins, Colorado

Software engineering experimentation must develop beyond statistical analysis of data and analytic models. We must become more active in developing a comprehensive discipline of simulation to aid experimental software engineering. This will require developing standard modeling components, connections between them, tools, and selection rules for them.

We are on the right track with the framework our process modeling advances provide. Unless we learn, however, more about how lower level components in the framework operate and when to use, what accuracy they provide, etc. process models will become a frame without a picture.

Ultimately we must come up with a "motherboard" architecture for software process and product with a well-defined, validated set of models and selection mechanisms for them. An

example of a family of models that warrants further work is reliability models. We often do not know why they work as well as they do (or not as the case may be). Simulation can help to answer some of these questions and provide predictable, high-quality components to populate a process framework.

Building Quantitative Models of the Software Development Process

Lionel Briand, SEL - CSD, University of Maryland, College Park

In order to make of software development an engineering discipline, technologies must be assessed and processes must be optimized based on quantitative analysis.

In order to do so, data must be collected, validated and transformed into models. These models should have the property of being predictive, but also interpretable in order to allow for process control and improvement.

Also, in order to increase the potential impact of software measurement on the software process, more metrics collectible early in the phases of software development should be developed and validated. Thus testability, maintainability, cost, schedule, reliability of software systems could be controlled effectively based on corporate past experience.

Early in the 70's, the SEL (NASA GSFC - U. Md. - CSC) started addressing these issues. Successes, but also failures were the results of 15 years of effort and commitment. Thus, a real expertise has been acquired and has shown with strength that measurement may lead to software process improvement.

Rethinking Measurement to Support Incremental Process Improvement

Adam Porter, University of Maryland, College Park

Software engineering has been unable to provide a detailed mechanism for systematic, incremental process improvement. As a discipline, we lack the knowledge necessary to select the right processes, methods and tools to combat specific goals. We have also not developed models that illuminate that link between local actions and global process. In the absence of such information and models, we cannot reason about tradeoffs associated with alternate processes, risking arbitrary side-effects. We are developing formal models of process improvement to compare alternative processes. The goal of such a model is to quantify the value of candidate process improvements.

Appropriate Research Philosophies in Software Engineering

Ross Jeffery, University of New South Wales

Evidence from recent empirical research shows that software requirements identification and post-delivery evolution are the major software process concerns. They are also lifecycle sub-processes which have high social components in contrast with the construction process and concern the conceptual construct underlying the system rather than the contracted product. Three classes of research which can be used in software engineering are the positivist, interpretive, and empirical methods. The positivist approach includes traditional theory/test experimentation. This has been criticized for having insufficient consideration of the context

and history of the events under study and limited to stimulus/response experimentation. Interpretive studies do not search for determinism or universal lessons, but rather a understanding of the phenomena behind the behavior. Critical studies aim to change the status quo through whention. The empirical evidence suggests that greater use of interpretive research will be necessary to provide knowledge and improvement in the software process.

Future Work

Vincent Y. Shen, Hong Kong University of Science and Technology

I think we have accomplished a lot in the past 10 - 20 years. There are a lot of agreements which we did not have before. I feel that we have gone past the "knee" of getting new and significant results - i.e., future efforts will bring fewer new discoveries. There is nothing wrong for a discipline that has reached maturity, which is the state I believe the metrics and measurements community has reached.

I think it is now time to move out of our subculture so that we can demonstrate our value to software practitioners. There were two proposals made at the workshop which are worth pursuing: the "benchmark" project and the "software engineering handbook" project. We can make the software engineering discipline more an "engineering" discipline.

Markku Oivo, VTT Technical Research Center of Finland

It is fairly easy to develop just another metric but the real challenge is to convince the practitioners of the value of experimental software engineering. We need to sell our ideas to the industry starting with companies which are mature enough for changing their environment to a constantly improving process which can be measured and guided based on both on-line and post mortem quantification feedback. With the good success stories we can spread out the word and get the ideas widely spread in the industry. For planning, packaging, and reusing information we need to build useful models which consistent internal representations. We need to offer different viewpoints to these models for different users.

A Reuse Culture for Software Construction

Claus Lewerentz, Forschungszentrum Informatik Karlsruhe

We advocate the idea of creating a new style of software construction that is based on the systematic reuse of application-specific collections of software components and design structures together with corresponding process models. On the structural level object-oriented approaches proved to be well suited for the specification and realization of such application frameworks. On the management level a whole set of measures has to be taken to foster the creation, propagation, and application of reusable software. Particularly there are different processes for constructing general or application-specific and reusable components and frameworks and for the development of a particular application software system.

Software Business, Concurrent Engineering, and Experience Factory Relationships

Giovanni Cantone, University of Rome at Tor Vergata

Relevant future investigations include relationships of both product quality versus process quality and the firm's business environment versus the firm's organization. A simple model of a business is the business life cycle (BLC). Each stage of the BLC is characterized by some kind of technology, possibly of workers, who fill roles, members and possibly of competitors and so on. What is the present main stage of the software BLC? How does it affect the organization of the software factory? What is the strategic position of a firm into the SBLC presently and in perspective. Does such a position affect the factory organization? How? Due to the invasive nature of software products and technology, have we to expect software firms necessarily located in different points of the SBLC? The experience factory and need for engineering and synchronization between software factory processes are briefly essential from the viewpoint of the SBLC.

Software Understanding

Frank McGarry, NASA Goddard Space Flight Center, Greenbelt, MD, USA

Software engineers are finally coming to the realization that we cannot produce any significant principles of software (engineering) without first observing and recording the existing relationships, rules, and de facto models of software. This is the first step of understanding. Software as currently used in the practitioner community has been one of the key elements previously overlooked by the researcher software community. The year 1992, as evidenced by this seminar in Dagstuhl, has witnessed the evolving acceptance and realization that efforts must be accelerated in creating the baseline understanding of software in practice.

Briding the gap between research and practice in software engineering measurement: reflections on the staffing factor paradox.

Chris F. Kemerer, MIT Sloan School of Management

While there are many areas of "disconnect" between industry and academia, none is perhaps so glaring as the differential attention paid to the staffing on software projects. Practicing project managers treat this as one of, if not the most, important variable under their control, while academics typically ignore this factor in their models and analyses. While difficult to study (for a number of reasons) it is essential that academia devote more attention to this issue.

Les Belady, Mitsubishi Research Laboratories, Cambridge, Mass.

Software engineering will probably split into many branches - systems engineering, applications engineering, etc - and become more involved with problems of the application domain, computer hardware and communication. Accordingly, whose who experiment (with models) and trying to quantify software, would include in their studies the entire enterprise, and its economics, for which a software controlled computer applications is developed. By the way, computer science - and along with it software engineering - will be an important part of the education of many subjects and not much a well defined, isolated discipline.

Yet Another Laboratory for Software Engineering

Eric Sumner, AT&T Bell Laboratories

The Software Production Research department is devoted to experimental research in large software engineering. It was formed in the Fall of 1990 at Indian Hill, the Illinois complex that houses many large software developments including that of the 5ESS switch.

Development projects at Indian Hill invest 5% of their budgets in process management teams (PMTs) which are responsible for the cost, interval, and quality of each development process (e.g., design, customer documentation). The research department engages in two types of activities. In the first, we attempt to improve the PMT methodology by inventing new approaches for observation, description, measurement, and analysis of large software system development. In the second, we attempt to directly improve the development methodology by inventing new processes and associated technologies for developing large software systems.

Objectives and Context of Software Measurement, Analysis, and Control

Michael A. Cusumano, MIT Sloan School of Management

This paper focuses on the what and why of measurement in software development and the impact of the context of the development organization or of specific projects or approaches to measurement, analysis, and control.

It is naive to expect a consensus to emerge easily within even a single organization, let alone within an entire industry or among a set of different actors or observers, regarding what to measure or why to invest the time and resources to collect and analyze data in software development. The perspective of senior managers, project managers, QA or inspection personnel, marketing and customer service groups, and researchers from companies or academics can be very different. Different types of systems and customers, as well as different company and division cultures, also may have an enormous impact on what projects can realistically measure and how they can use data for planning and control, learning, or benchmarking - three general reasons why we measure software development activities.

Software Measurement and Experimentation Frameworks, Mechanisms and Infrastructure

Richard W. Selby, University of California, Irvine

Software measurement continues to be an important and rapidly growing area in software engineering research and practice. Measurement enables the systematic analysis, understanding and improvement of software systems and processes. Measurement principles, techniques and systems provide a cross-cutting foundation for many aspects of software development and evolution from experimentation to empirically guided software synthesis.

This workshop was very effective in bringing together many leading researchers and practitioners in the measurement area to define points of consensus, controversy and future directions. I personally enjoyed the workshop very much, especially because of the opportunity to have deep interactions with the other participants. I hope this meeting is just one of many opportunities to have such synergistic relationships and I look forward to the next meeting.

Systematic Software Technology Transfer

H. Dieter Rombach, FB Informatik, Universität Kaiserslautern

Software development is an engineering activity and need to be treated as such. In addition to technological methods and tools (e.g., languages, compilers, testing tools) we need to incorporate experimentation and measurement. One area in which the need for measurement is needed is technology transfer. We need to be able to quantify the needs for new technology, the effects and limitations of new candidate technology, and monitor the economic effectiveness of new technology. Furthermore, we need companies to treat technology improvement and transfer as an issue which requires specific attention. Transferring new technologies into an organization can be expected to change the development process used by this organization. It is unrealistic to expect project members to manage such changes on the fly and still fulfill their product-oriented project goals. We suggested a model which is based on sound scientific and engineering principles. It distinguishes between roles of technology improvement versus application development on the one hand, and process versus product engineering on the other. This model is currently being instantiated in the Software Technology Transfer Institute at Kaiserslautern. This seminar was very successful in bringing together the different groups (industry and academia) to exchange experiences gained in the past 10 years and to develop an agenda for the future

Dagstuhl-Seminar 9238:

List of Participants (update: 23.11.92)

William **Agresti**
MITRE Corp.
7525 Colshire Drive
McLean VA 22102
USA
agresti@mitre.org
tel.: +1-703-883-75 77

Victor R. **Basili**
Univ. of Maryland at College Park
Department of Computer Science
College Park MD 20742
USA
basili@cs.umd.edu
tel.: +01-301-405-26 68

Laszlo A. **Belady**
Mitsubishi Electric Research Labs
201 Broadway
Cambridge MA 02139
USA
belady@merl.com
tel.: +1-617-621-75 02

Lionel **Briand**
Univ. of Maryland at College Park
Department of Computer Science
College Park MD 20742
USA
lionel@tame.cs.umd.edu
tel.: +1-301-405-27 21

Giovanni **Cantone**
University of Rome - The Vergata
Dept. of Electronic Engineering
Via della Ricerca Scientifica
I-00133 Roma
Italy
cantone@tovvx1.ccd.utovrm.it
tel.: +39-6-72 59 44 95

Michael **Cusumano**
MIT
Sloan School of Management
Cambridge MA 02139
USA
mcusumano@sloan.mit.edu
tel.: +617-253-2574

Albert **Endres**
IBM Deutschland GmbH
Entwicklung und Forschung Abt. 32 75
Schönaicher Str. 220
W-7030 Böblingen
Germany
aendres@vnet.ibm.com
tel.: +49-7031-16-34 65

Stuart **Feldman**
Bellcore
MRE 2E-386
445 South Street
Morristown NJ 07960-1910
USA
sif@bellcore.com
tel.: +1-201-829-43 05

Norman **Fenton**
City University
Center for Software Reliability
Northhampton Square
London EC1V HB
Great Britain
n.e.fenton@city.ac.uk
tel.: +44-71-477-84 25

Norbert **Fuchs**
ALCATEL-Austria ELIN
Forschungszentrum
Ruthnergasse 1-7
A-1210 Wien
Austria
norbert.fuchs@rcvie.co.at
tel.: +43-1-39 16 21/2 64

Warren **Harrison**
Portland State University
Center for Software Quality Research
P.O. Box 751
Portland Oregon 97207-0751
USA
warren@cs.pdx.edu
tel.: +1-503-725-31 08

Dan **Hoffman**
University of Victoria
Department of Computer Science
P.O. Box 3055
Victoria B.C. V8W 3P6
Canada
dhoffman@uvunix.uvic.ca
tel.: +1-604-721-7222

Ross **Jeffery**
University of New Sourth Wales
School of Information Systems
P.O. Box 1
Kensington New South Wales 2033
Australia
rossj@cumulus.csd.unsw.oz.au
tel.: +61-2-6 97 44 13

Chris **Kemerer**
MIT
Sloan School of Management
E53-315
50 Memorial Drive
Cambridge MA 02139
USA
ckemerer@sloan.mit.edu
tel.: +1-617-253-29 71

Barbara **Kitchenham**
National Computer Centre Limited
Oxford House
Oxford Road
Manchester M1 7ED
Great Britain
tel.: +44-61-228-68 33

Günter R. **Koch**
2i Industrial Informatics GmbH
Haierweg 20e
W-7800 Freiburg
Germany
gk@dandsnx.uucp
tel.: +49-761-4 22 57

Meir **Lehman**
Imperial College of Science
Department of Computing
180 Queen's Gate
London SW7 2BZ
Great Britain
mml@iedoc.ic.ac.uk
tel.: +44-71-589-51 11

Claus **Lewerentz**
FZI Karlsruhe
Haid-und-Neu-Str. 10-14
W-7500 Karlsruhe
Germany
lewerentz@fzi.de
tel.: +49-721-9654-6 02

Bev **Littlewood**
City University
Center for Software Reliability
Northhampton Square
London EC1V HB
Great Britain
b.littlewood@city.ac.uk
tel.: +44-71-477-84 20

Jochen **Ludewig**
Universität Stuttgart
Institut für Informatik
Breitwiesenstraße 20-22
W-7000 Stuttgart 80
Germany
ludewig@informatik.uni-stuttgart.de
tel.: +49-711-7816-354

Nazim **Madhavji**
McGill University
Scholl of Computer Science
3480 University Street
Montreal Quebec H3A 2A7
Canada
madhavji@opus.cs.mcgill.ca
tel.: +1-514-398-37 40

John **Marciniak**
CTA Inc.
6116 Executive Blvd.
Rokville MD 20852
USA
marciniak@smtplink.cta.com
tel.: +1-301-816-14 39

Anneliese v. **Mayrhauser**
Colorado State University
Computer Science Department
Fort Collins CO 80523
USA
avm@cs.colostate.edu
tel.: +1-303-491-70 16

Frank **McGarry**
NASA/GSFC
Code 552
Romm E237 - Bldg. 23
Greenbelt MD 20771
USA
fmcgarry@GSfcmail.nasa.gov
tel.: +1-301-286-63 47

Karl Heinrich **Möller**
SIEMENS AG - ZFE ST AC5
Zentralabt. Forschung und Entwicklung
Otto-Hahn-Ring 6
W-8000 München 83
Germany
tel.: +49-89-636-4 76 60

Hausi A. **Muller**
University of Victoria
Department of Computer Science
P.O. Box 3055
Victoria B.C. V8W 3P6
Canada
hausi@csr.uvic.ca
tel.: +1-604-721-7630

Markku **Oivo**
Technical Research Center of Finland
Computer Technology Laboratory
Kaitovayla 1
SF-90571 Oulu
Finland
moi@tko.vtt.fi
tel.: +358-81-551-21 11

Adam **Porter**
Univ. of Maryland at College Park
Department of Computer Science
College Park MD 20742
USA
aporter@cs.umd.edu
tel.: +1-301-405-27 02

H. Dieter **Rombach**
Fachbereich Informatik
AG Software Engineering
Postfach 3049
W-6750 Kaiserslautern
Germany
rombach@informatik.uni-kl.de
tel.: +49-631-205-28 95

Walt **Scacchi**
University of Southern California
School of Business Administration
Decision Systems Department
Bridge Hall 401 V
Los Angeles CA 90089-1421
USA
scacchi@pollux.usc.edu
tel.: +1-213-740-47 82

Norm **Schneidewind**
U.N. Naval Postgraduate School
Code AS/SS
Monterey CA 93943-5100
USA
0442p@cc.nps.navy.mil
tel.: +1-408-646-27 19

Richard W. **Selby**
University of California
Department of Information and
Computer Science
Irvine CA 92717
USA
selby@ics.uci.edu
tel.: +01-714-856-63 26

Vincent **Shen**
The Hong Kong University of
Science and Technology
Department of Computer Science
Clear Water Bay
Kowloon
Hong Kong
shen@uxmail.ust.hk
tel.: (852) 358-7009

Eric **Sumner**
AT&T Bell Labs
1000 E Warrenville Rd
Naperville IL 60566
USA
ees@research.ih.att.com
tel.:+1-708-713-76 60

Walter **Tichy**
Universität Karlsruhe
Fakultät für Informatik
Im Fasanengarten 5
W-7500 Karlsruhe
Germany
tichy@ira.uka.de
tel.: +49-721-608-3934

Kevin **Wentzel**
Hewlett Packard Labs
P. O. Box 10490
Palo Alto CA 94303-0969
USA
wentzel@hpl.hp.com
tel.: 415-857-4018

Marvin **Zelkowitz**
Univ. of Maryland at College Park
Department of Computer Studies
College Park MD 20742
USA
mvz@cs.umd.edu
tel.: +1-301-405-26 90

Horst **Zuse**
TU Berlin
Fachbereich 20 Informatik
Franklinstr. 28-29
W-1000 Berlin 10
Germany
zuse@tubvm.cs.tu-berlin.de
tel.: +49-30-314-7 34 39

Stu **Zweben**
Ohio State University
Depart. of Computer &
Information Science
2036 Neil Avenue
Columbus OH 43210-1277
USA
zweben@cis.ohio-state.edu
tel.: +1-614-292-95 26

Zuletzt erschienene und geplante Titel:

N. Habermann, W.F. Tichy (editors):
Future Directions in Software Engineering, Dagstuhl-Seminar-Report; 32; 17.2.-21.2.92 (9208)

R. Cole, E.W. Mayr, F. Meyer auf der Heide (editors):
Parallel and Distributed Algorithms; Dagstuhl-Seminar-Report; 33; 2.3.-6.3.92 (9210)

P. Klint, T. Reps, G. Snelting (editors):
Programming Environments; Dagstuhl-Seminar-Report; 34; 9.3.-13.3.92 (9211)

H.-D. Ehrich, J.A. Goguen, A. Sernadas (editors):
Foundations of Information Systems Specification and Design; Dagstuhl-Seminar-Report; 35; 16.3.-19.3.9 (9212)

W. Damm, Ch. Hankin, J. Hughes (editors):
Functional Languages:
Compiler Technology and Parallelism; Dagstuhl-Seminar-Report; 36; 23.3.-27.3.92 (9213)

Th. Beth, W. Diffie, G.J. Simmons (editors):
System Security; Dagstuhl-Seminar-Report; 37; 30.3.-3.4.92 (9214)

C.A. Ellis, M. Jarke (editors):
Distributed Cooperation in Integrated Information Systems; Dagstuhl-Seminar-Report; 38; 5.4.-9.4.92 (9215)

J. Buchmann, H. Niederreiter, A.M. Odlyzko, H.G. Zimmer (editors):
Algorithms and Number Theory, Dagstuhl-Seminar-Report; 39; 22.06.-26.06.92 (9226)

E. Börger, Y. Gurevich, H. Kleine-Büning, M.M. Richter (editors):
Computer Science Logic, Dagstuhl-Seminar-Report; 40; 13.07.-17.07.92 (9229)

J. von zur Gathen, M. Karpinski, D. Kozen (editors):
Algebraic Complexity and Parallelism, Dagstuhl-Seminar-Report; 41; 20.07.-24.07.92 (9230)

F. Baader, J. Siekmann, W. Snyder (editors):
6th International Workshop on Unification, Dagstuhl-Seminar-Report; 42; 29.07.-31.07.92 (9231)

J.W. Davenport, F. Krückeberg, R.E. Moore, S. Rump (editors):
Symbolic, algebraic and validated numerical Computation, Dagstuhl-Seminar-Report; 43; 03.08.-07.08.92 (9232)

R. Cohen, R. Kass, C. Paris, W. Wahlster (editors):
Third International Workshop on User Modeling (UM'92), Dagstuhl-Seminar-Report; 44; 10.-13.8.92 (9233)

R. Reischuk, D. Uhlig (editors):
Complexity and Realization of Boolean Functions, Dagstuhl-Seminar-Report; 45; 24.08.-28.08.92 (9235)

Th. Lengauer, D. Schomburg, M.S. Waterman (editors):
Molecular Bioinformatics, Dagstuhl-Seminar-Report; 46; 07.09.-11.09.92 (9237)

V.R. Basili, H.D. Rombach, R.W. Selby (editors):
Experimental Software Engineering Issues, Dagstuhl-Seminar-Report; 47; 14.-18.09.92 (9238)

Y. Dittrich, H. Hastedt, P. Schefe (editors):
Computer Science and Philosophy, Dagstuhl-Seminar-Report; 48; 21.09.-25.09.92 (9239)

R.P. Daley, U. Furbach, K.P. Jantke (editors):
Analogical and Inductive Inference 1992 , Dagstuhl-Seminar-Report; 49; 05.10.-09.10.92 (9241)

E. Novak, St. Smale, J.F. Traub (editors):
Algorithms and Complexity of Continuous Problems, Dagstuhl-Seminar-Report; 50; 12.10.-16.10.92 (9242)

J. Encarnação, J. Foley (editors):
Multimedia - System Architectures and Applications, Dagstuhl-Seminar-Report; 51; 02.11.-06.11.92 (9245)

F.J. Rammig, J. Staunstrup, G. Zimmermann (editors):
Self-Timed Design, Dagstuhl-Seminar-Report; 52; 30.11.-04.12.92 (9249)