

Laboratory for
Appled
Software
Engineering
Research

at ETH Zürich

July 2002

TABLE

| | |
|---|-----------|
| <u>OVERVIEW</u> | <u>3</u> |
| <u>WHY APPLIED SOFTWARE ENGINEERING RESEARCH?</u> | <u>4</u> |
| <u>COMPONENT CERTIFICATION CENTER</u> | <u>10</u> |
| <u>INFORMATION SECURITY CENTER</u> | <u>12</u> |
| <u>PROGRAMMING RESOURCES CENTER</u> | <u>13</u> |
| <u>PRACTICAL SETUP OF THE LABORATORY</u> | <u>15</u> |
| <u>RESOURCES AND CONTACTS</u> | <u>15</u> |
| <u>SUMMARY</u> | <u>18</u> |

Laboratory for Applied Software Engineering Research

at ETH Zürich

1 OVERVIEW

The Laboratory for Applied Software Engineering Research establishes at ETH Zürich a center of excellence in software technology, intended to improve the state of the art in this crucial field. The Laboratory benefits research, teaching, and local industry.

The Laboratory's mission is to explore and implement methods and techniques for producing software of much higher quality than is the norm today, hence addressing a key hindrance of modern societies. The work of the Laboratory helps the IT industry cope with the ever increasing size and sophistication of requested software applications, and the ever decreasing tolerance of users for malfunctioning or otherwise unsatisfactory software.

The word “*Applied*” in the Laboratory's name emphasizes the practical nature of its work, focused on developing solutions that can be applied to mission-critical projects. This includes theoretical work too, both to use existing theories when applicable to software practice, and to further the state of software engineering theory. The Laboratory relies on the most effective modern approaches to quality software construction, such as object technology, component-based development, formal software construction, proofs of program correctness.

Specific initiatives associated with the Laboratory include:

- *Component Certification Center*: process, standards and tools for producing high-quality reusable components and assessing the quality of commercial components.
- *Information Security Center*: knowledge, education and research facility devoted to the study and enforcement of IT security.
- *Programming Resources Center*: pooled development resources to support quality software development in an academic environment.

The laboratory provides a common structure for these developments, and remains open for future initiatives that may arise in connection with new developments in software technology.

The remaining sections of this report explore the various facets of the Laboratory's background and activity:

- State of the art in software engineering and need for a Laboratory of Applied Software Engineering Research. → [Section 2, page 4.](#)
- The Component Certification Center. → [3, page 10.](#)
- The Information Security Center. → [4, page 12.](#)
- The Programming Resources Center. → [5, page 13.](#)
- Practical setup for the Laboratory. → [6, page 15.](#)
- Resources and contacts. → [7, page 15.](#)

2 WHY APPLIED SOFTWARE ENGINEERING RESEARCH?

Keeping up

Software is at the center of most processes in modern societies. The extraordinary development of computer hardware over the past 50 years has made possible applications that no one would have imagined, such as detailed, on-time weather prediction. “Moore’s Law” — the doubling of computer power every 18 months, at constant or decreased price — has reigned supreme until now, and will continue to apply in the near future, leading to an ever growing appetite for new applications.

Software, for its part, is constrained by the human mind, which doesn’t improve at the speed of electronic technology. So it’s not surprising that while there have been great advances in software engineering, they haven’t kept up with those of hardware. The result is not only a discrepancy between the set of applications we would like to produce and those we complete; it’s a general state of dissatisfaction with the quality of the software that does get produced. Here we may just quote directly from a press report about a recent study by NIST:

Technology - Reuters

Software Errors Cost Billions

Fri Jun 28, 6:12 PM ET

(Reuters, June 2002)

NEW YORK (Reuters) - Software bugs are not just annoying or inconvenient. They're expensive.

According to a study by the U.S. Department of Commerce's National Institute of Standards and Technology (NIST), the bugs and glitches cost the U.S. economy about \$59.5 billion a year.

"The impact of software errors is enormous because virtually every business in the United States now depends on software for the development, production, distribution, and after-sales support of products and services," NIST Director Arden Bement said in a statement on Friday.

Software users contribute about half the problem, while developers and vendors are to blame for the rest, the study said. The study also found that better testing could expose the bugs and remove bugs at the early development stage could reduce about \$22.2 billion of the cost.

"Currently, over half of all errors are not found until 'downstream' in the development process or during post-sale software use," the study said.

The study, conducted by the Research Triangle Institute in North Carolina and the software industry was conducted to identify and assess technical needs to improve software-testing capabilities.

Software is error-ridden, in part because of the complexity inherent in millions of lines of code. About 80 percent of the cost of developing software programs goes to identifying and correcting defects. Yet, few products of any type other than software are shipped with such high levels of errors, the study found.

Other factors contributing the problem include marketing strategies, limited liability by software vendors, and decreasing returns on testing and debugging, according to the study.

In January, the National Academy of Sciences ([news](#) - [web sites](#)) issued a report urging lawmakers to consider adopting legislation to hold software vendors liable for security breaches.

If software makers were held liable, the cost to consumers would rise dramatically, said Marc E. Brown, a partner at the Los Angeles law firm of McDermott, Will & Emery.

However, Europe already has begun addressing the issue.

A Dutch judge in September convicted Exact Holding of malpractice for selling buggy software, rejecting the argument that early versions of software are traditionally unstable.

Unless one assumes that the rest of the world uses techniques vastly superior to the Americans' (it doesn't), the \$60 billion figure transposes to at least \$150 billion worldwide. An article about Swiss incidents, whose beginning is reproduced on the next page, provides evidence that the situation described in the Reuters article is by no means specific to the USA.

Taking judicial actions such as the one cited at the end of the Reuters article may encourage suppliers to take a more systematic attitude towards validating their software. But the technical problems remain enormous. The report's comments that "*Better testing could remove the bugs at an early stage*", while not totally incorrect (better testing can remove *some* bugs) is naïve in its generality, since testing techniques can only address a small part of the issue. Unlike hardware devices, software systems have such huge numbers of possible input combinations that testing, even with the best modern techniques, can do no more than increase confidence in the software by a moderate amount — find a few more needles in the haystack. Testing is just one in a battery of approaches of which others — better development languages, formal (mathematical) software construction, formal proofs of correctness, object technology, component-based development — hold considerable promise.

NEWS

[Inhalt Print](#)
[Schweiz](#)
[Gesellschaft](#)
[Wirtschaft](#)
[Medien](#)
[Computer](#)[Internet](#)
[Wissen](#)
[Sport](#)
[Ausland](#)
[Kultur](#)
[Demontage](#)
[Wetter](#)
[Money](#)

SERVICE

[FastSearch](#)
[Archiv](#)
[Bookmarks](#)
[Desktopnews](#)
[Newsletter](#)
[SMS-Dienst](#)
[Übersicht/Site-Map](#)
[Abonnement](#)
[Guides Bestellen](#)
[Mediadaten](#)
[Impressum](#)

COMMUNITY

[Openchat](#)
[Forum](#)
[Games](#)
[Casino](#)
[Tango](#)
[Kontakt](#)

RUBRIKINSERATE

[Immowinner](#)
[Jobwinner](#)
[Carwinner](#)
[Partnerwinner](#)

Absturz mit System

Computer-Software wird unter höchstem Zeitdruck entwickelt. Der Swisscom-GAU wird nicht der letzte Totalabsturz sein.

Von Serge Hediger

Beim Blitzkurier in Kloten ZH ging letzten Freitag gar nichts mehr. Die Natel-Panne der Swisscom hatte den Dreimann-Betrieb komplett lahm gelegt. «Wenn wir gewusst hätten, was los ist, hätten wir dichtgemacht und uns einen Tag freigenommen», sagt Dani Trochsler. Der Geschäftsführer des Kurier- und Kleintransportunternehmens rechnet nun mit einer Umsatzeinbusse von 500 bis 1000 Franken – dank Sommerflaute vergleichsweise wenig. Firmen beklagen Verdienstaufschübe. Geschäftsleute konnten Anrufe nicht tätigen. Liebende fanden einander nicht.

Der Netzausstieg hat 3,4 Millionen Natelbenutzer betroffen. Schlagartig wurde ihnen am Mittag des 27. Juli klar, wie abhängig sie von der Technik sind. Nichts ging mehr auf dem Natel-Netz von Marktführerin Swisscom mobile. Ein Software-Fehler hatte den Zentralcomputer in Lausanne abstürzen lassen. «Kein Netz», hiess es über zehn Stunden lang auf den Handy-Displays. Software-Fehler – der Begriff taucht immer häufiger auf, wenn Geräte, Einrichtungen oder Dienste nicht funktionieren. Beispiel SBB. Erst Mitte Juni war das neue Stellwerk Basel zweimal ausgefallen. Ein Software-Fehler im Betriebszentrum legte den Zugverkehr für mehrere Stunden lahm. Das Reserve-Computersystem – der so genannte Back-up-Rechner – schaltete sich beide Male zwar automatisch ein, stürzte aber kurze Zeit darauf ebenfalls ab – wegen des gleichen Software-Fehlers.

Den verärgerten Zugspassagieren entstanden Verspätungen bis zu 90 Minuten. Beispiel Handy. Ende Mai erst musste die Herstellerin Sony in Japan über eine halbe Million Mobiltelefongeräte zurückrufen, da ihre Software fehlerhaft war. Die Handys wurden kostenlos umgetauscht. Und das sehnlichst erwartete WAP-Handy 7110 von Nokia war in der Schweiz lange Zeit nicht lieferbar – Software-Fehler. Beispiel EC-direct. Ausgerechnet am verkaufsintensiven 24. Dezember letzten Jahres legte ein computergesteuerter Roboter ein Speichermedium nicht richtig ein. Zehntausende konnten an den Terminals in den Warenhäusern ihre Weihnachtsgeschenke nicht bezahlen.

Weiterer Artikel

[Interview mit CEO von Swisscom mobile](#)

FACTS Archiv

2001/31 [«Relativ fassungslos»](#)
 2001/24 [Terror-SMS aufs Handy](#)
 2000/07 [WAP ohne Publikum](#)

weitere Artikel

Scheissshandy



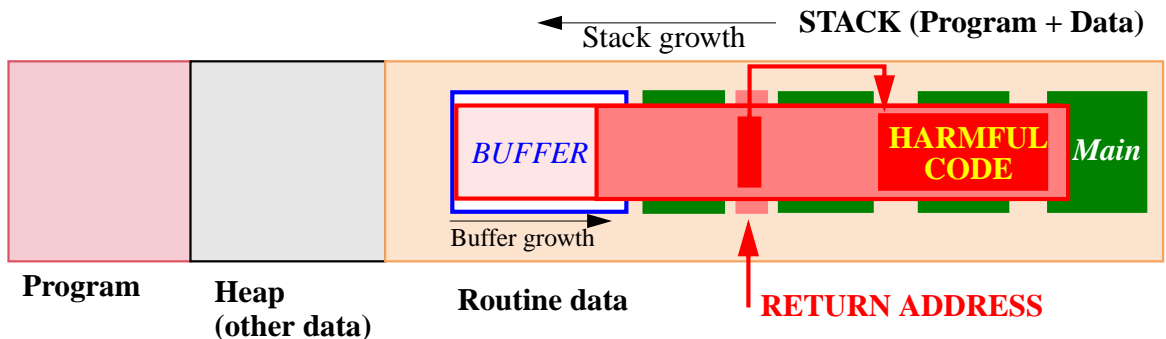
Bänz Friedli
 FACTS-Kulturredaktor

Am Morgen die Rechnung in der Post: 1209 SMS hab ich letzten Monat verschickt, mal 241.80 Franken, Himmel, das doch bestimmt Schweizer Rekord, wann spendet mir die Swisscom endlich mal eine Prämie? Zum Beispiel eine Reise nach, sagen wir mal, Alaska? Dort wärs jetzt kühl.

Kurz nach zwölf, vor einem Teller Äpfel/Magronen, will meinem Kollegen Thomas eine Kurznachricht schicken. Nicht Besonderes. Dass ich das selbste net gefunden hätte, was er vorhin im Büro gesagt habe, dem Display erscheint NACHRICHT NICHT

From software engineering to information security

One particular kind of non-quality has achieved global fame: security violations, through viruses, worms, Denials of Service and other attacks. Although security is a research field in itself, it is closely connected to the work on other software quality issues and software engineering in general; most attacks exploit a deficiency in some software product, due to poor software engineering practice. The most striking example is the *buffer overflow* technique, present (usually along with other mechanisms) in most recent viruses and attacks. Buffer overflow uses the typical memory layout of applications to change the return address of some seemingly innocuous, publicly available facility, such as the program that reads user input in a Web browser:



The data entered into the program goes into the “*BUFFER*”. With a poorly written program, a hostile user could artfully contrive such data to overrun the size normally attributed to the buffer, so as to change the “**RETURN ADDRESS**” of the program and make it point to some “**HARMFUL CODE**” that the attacker has deposited (as part of that very data), hence giving him full access to the machine. No wonder a 20-year-old from the Philippines can, in a few moments, disrupt the global IT infrastructure, and the global economy.

Buffer overflow fundamentally relies on bad software construction: a well-written utility, especially one that is available to any user such as an outside visitor to the Web site, should ensure that references to the buffer stay within its official limit (the blue rectangle on the figure), and reject any input that would extend beyond that limit. This close connection between IT security issues and general software engineering issues explains why the Laboratory addresses both aspects and includes an Information Security Center as one of its constituent entities.

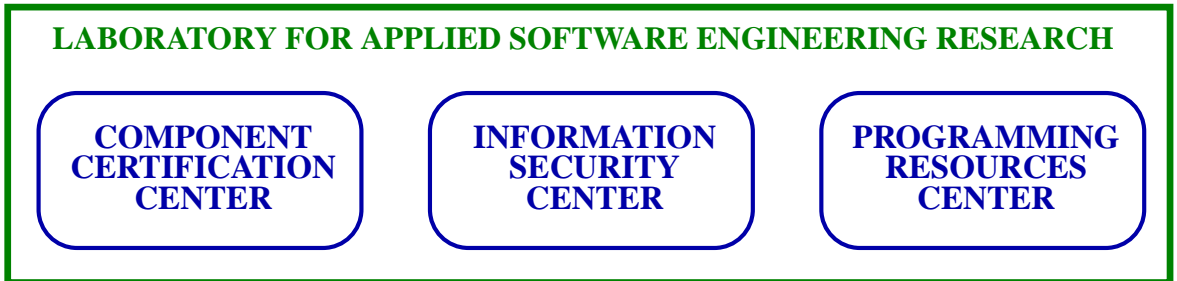
The Laboratory and ETH

The Laboratory takes advantage of the unique situation of ETH Zürich:

- The techniques that can help address the most pressing issues of software technology involve a delicate combination of theory and practice. This benefits from the engineering tradition and spirit of ETH.
- This tradition is particularly strong in computer science, where ETH has achieved worldwide fame through pioneering practical work on languages, programming methodology and compilers.
- The financial industries, in particular banking and insurance, are in dire need of better solutions, making them primary recipients for the results of the Laboratory's work.
- Existing collaborations, in particular with Germany, the UK, France and the US, support the development of the Laboratory.
- Recent faculty recruitment has strengthened software engineering at ETH, although more is needed.

Overall organization

As noted in the Overview, the Laboratory initially hosts three Centers:



The three Centers benefit from a common infrastructure and shared overall goals, while pursuing specific projects. The Laboratory's structure is designed with enough flexibility to accommodate other such Centers in the future, as long as their scope and spirit fit within those of the organization.

The next three sections describe the current Centers in turn.

3 COMPONENT CERTIFICATION CENTER

One of the most promising developments in the software engineering field, viewed by some experts as the only hope for coping with the ever-increasing sophistication of society's demands, is "reuse", or "component-based development": the reliance on pre-written software elements, or "components", covering patterns that systematically recur in applications. By systematically reusing such components, each application development can focus on its truly innovative and competitive contributions, relying on best-of-breed existing solutions for all the rest.

The idea is similar to how the electronics and computer industry has reengineered itself to take advantage of reusable solutions — VLSI chips. Reuse is, however, far more difficult in software because of the higher degree of variability of the basic components. It is not sufficient in software to reuse components applying well-defined transformations to simple sets of inputs: the reuser must be able to *adapt* the components to more specific needs of the application at hand. This need to combine reuse with adaptation is what makes the reusability issue a major challenge of software engineering.

Important developments have occurred in this area. The spread of *object technology* has established a strong technical basis for reusable components. A number of *component models* have emerged, such as Borland's Delphi, Microsoft's COM, Sun's Enterprise Java Beans (EJB), CORBA, and Microsoft's most recent offering, .NET (pronounced *dotnet*). As a result, a small industry of components is thriving, and a number of large companies see the future of their software developments residing entirely in component reuse; they expect major progress in quality and productivity through effects of scale.

This development, however promising, has largely occurred without much attention being paid to the *quality* of the components, in spite of the potential dangers of reusing unsatisfactory components. The scaling effect scales up everything, good and bad; in a mission-critical application built out of components, the quality of the weakest link determines the quality of the application as a whole. The component-based reengineering of the electronic industry could only occur thanks to a fanatical approach to quality at every stage of the process; the IT industry is still far from sharing this attitude. For all its enthusiasm about reuse, it has been slow to realize that moving to component-based development without a meticulous attention to component quality could bring as many dangers as benefits.

The Component Certification Center provides a central resource to address component quality. It requires the combination of a *low-road* approach, addressing immediate needs of component qualification, and a *high-road* requiring more research and paving the way for the fully guaranteed components of tomorrow:

- The “low road” is essential to help the industry assess the quality of today’s EJB, COM or .NET components, of which, for the most part, no one has any good grasp. One cannot expect to prove such components correct, or otherwise guarantee them fully; but it is still possible to perform credible assessments of some of their quality-relevant properties. A **Component Quality Model** under development is the central effort supporting this goal.
- The “high road” leads to future components developed with far more rigorous techniques than today. In particular, considerable advances have occurred in recent years in the area of *mathematical proofs* of program correctness, leading to the full proofs of systems such as the security mechanism of the new driver-less metro line in Paris (Meteor). The potential of applying such proofs to components, where the development and proof effort is justified by the scaling-up effect of reuse, is particularly attractive, leading to some of the central tasks of the Component Certification Center.

To pursue both of these goals, the Center organizes its activities along three main lines: *normative*, *analytic* and *innovative*.

The **normative** work establishes the basis for component development and assessment: methods, techniques, standards, best practices, assessment rules for today’s and tomorrow’s components. The development of the Component Quality Standard is at the center of this effort.

In its **analytic** work, the Component Certification Center is available for evaluations of existing components, useful to both the producers of these components, and to potential consumers (companies that may use these components for their own developments), for which no independent agency is available today to assess the quality of commercial or open-source components.

This analytic effort, devoted to assessing practical components, may be viewed as a service activity. As such it may ultimately outgrow its placement in a university. When and if this happens, it will be a sign of success; the component assessment task may then have to be outsourced to a new entity, either non-profit or commercial. In the near term, however, there is no cause for concern about possible incompatibilities with the usual goals of university research: the goal of component assessment is so new as to require

several years of research and experimentation before anyone is ready to turn the effort into a routine service activity.

The **innovative** part of the Center's work is research work to develop new techniques for producing better components and evaluating their quality. A key part of this effort is the application of proof techniques to components, a possibility that after many years of research appears within reach but still requires considerable effort, theoretical and practical. Since "do as I do" is a welcome complement to "do as I say", the Center also develops model components of guaranteed quality, intended to serve as a reference for the industry.

Surprisingly, although a considerable literature has already been devoted to component quality, no Center of the kind described here has to our knowledge been set up so far. Having such a Component Certification Center in place at ETH is a way to achieve strategic leadership in software engineering, with benefits not only through research and the worldwide IT industry — it is not unrealistic to assume that the Center will help make at least a small dent into the 150 billions' waste cited above — but also on the local industry and on the whole process of teaching software at ETH.

4 INFORMATION SECURITY CENTER

Security violations have become headline news around the world, and are among the most serious obstacles to the development of Information Technology. The ease with which individuals can bring down systems and penetrate computer installations is a sword of Damocles hanging over every move of the industry.

The most widely reported attacks, such as viruses, are only part of the issue. Most of the computer infrastructure — including its critical component, the Internet — has been built to maximize efficiency and convenience, not to protect against hostile acts. Attackers with time and resources can defeat many security measures; some of the techniques do not even require particular intelligence, as they are readily found on Web sites. The resulting dangers constantly hamper progress. As an example, it would technically be possible today, thanks to wireless technology, to let anyone connect a laptop computer or PDA to the Internet anywhere in an urban area such as Zürich without any hassle. The infrastructure is either there or easy to install. What has prevented such technology from becoming universally available is concern about security risks.

Universities have traditionally been a hotbed of security attacks. The game of cat and mouse, the intellectual challenges, the taste of the forbidden naturally attract some students, often among the most creative. The Information Security Center deflects such curiosity towards beneficial goals by providing a secure environment where students can be taught IT security, and experiment in depth with attacks and counter-attacks, without causing any harm, while being taught the ethics of computer use. In the protected setup of the Information Security Center, students will learn about vulnerabilities of operating systems and other products, about the anatomy of viruses and other attacks, about software engineering techniques that avoid such problems, about managerial and organizational responses to security threats, about the security measures that an enterprise must take, and about the proper professional attitudes towards computers and their security.

The Information Security Center is organized so that every computer science student at ETH gets an introduction to security as part of the standard curriculum, and that interested students can pursue more in-depth work for semester projects, Diplom (Master's) projects and advanced studies.

The Information Security Center provides a circumscribed environment within which students can give free rein to their astuteness and creativity without causing any risk to production systems, and learn to channel their skills towards beneficial purposes.

The Center's setup requires careful planning to ensure that it does not have any harmful effect on the security of the ETH infrastructure, and a precise definition of what is permissible. Cooperation with the Informatikdienste and other units responsible for computers and networks is an integral part of the Laboratory's organization.

The focus of the Information Security Center as described here is educational. The Center also offers a research potential, to be developed in cooperation with the forthcoming Professor in Information Security.

5 PROGRAMMING RESOURCES CENTER

Computer science research has not traditionally had the benefit of professional support for software development. The assumption has been that professors, researchers and PhD students would be responsible for producing their own software. This goes well with the tradition of computer science at ETH, which has resulted in a number of widely used academic products, in particular programming language implementations.

This scheme has the advantage of making programming an integral part of the research experience. Unlike some theory-only computer science departments, ETH Computer Science recognizes the intellectual and scientific value of software development.

Current developments in the software field limit, however, what this approach can achieve. Expectations on the part of the “market” are very high, even if this market is purely academic. It is no longer enough, for example, for a university project to deliver to the world an excellent compiler: even novice students have experienced modern GUI (graphical user interface) environments with debuggers, browsers, project management tools and other automated support, and expect that level of machinery as a minimum, whether in a commercial product or in an academic tool. In addition, the computer industry constantly brings in new operating system versions and new platforms; the resulting tasks of software porting and maintenance, essential to the practical success of tools, even those produced in an academic context, are hard to carry out properly if the people in charge are researchers with other priorities and PhD students who will go away once they have defended their thesis.

The Programming Resources Center, similar to an small internal software house, addresses this issue by providing a professional staff explicitly devoted to programming support work that helps make research products practical and usable: development of supporting software, quality assurance, maintenance, professional documentation, porting to new platforms, customer support, interfaces with other products. The presence of the Programming Resources Center also helps instill the software engineering culture — the emphasis on long-term viability and professional quality of the software being produced — that is not always present in a university environment but is an essential part of the teaching and research experience in computer science.

While the idea of the Programming Resources Center is new to computer science departments, it is hardly revolutionary if we relate it to the long-established practice of science and engineering departments in universities, which routinely employ technicians to help researchers set up experiments and develop devices. The members of the Programming Resources Center play a similar role for computer science researchers. Researchers will still be writing programs; the Programming Resources Center helps them by providing the professional infrastructure that can make the difference between an interesting prototype and a public-domain tool that is used worldwide and makes its mark on the field.

6 PRACTICAL SETUP OF THE LABORATORY

The Laboratory for Applied Software Engineering Research is managed by an executive committee consisting of the Laboratory's director and the heads of each of the Centers.

Its supervision is ensured by a Board consisting of

- A representative of the Computer Science department, not himself a member of the Laboratory.
- Two representatives of Swiss industry.
- Two representatives of non-Swiss industry.
- Four representatives of academic organizations devoted to software engineering (for example laboratories devoted to software technology), two in Europe and two outside of Europe.
- The director of the laboratory.
- A representative of the laboratory's non-professorial academic personnel (Mittelbau).

The Board meets yearly to assess progress, and remains in frequent contact with the laboratory's management through email.

The Laboratory works in close cooperation with the Computer Science Department and provides its resources to researchers and students. It is closely involved in the software part of the computer science curriculum.

Every year, the Component Certification Center organizes a Workshop on Trusted Components, including the best international experts in the field.

7 RESOURCES AND CONTACTS

Resources: positions

The resources for the Laboratory, to be set up for five years, include the following positions:

- 6 Postdocs.
- 8 assistants.
- 2 positions of administrative/editorial staff.
- 5 professional software developers (for the Programming Resources Center).
- 1 software project manager (for the Programming Resources Center).
- 1 Web developer.

- 3 new professor positions, described next.

Operating budget

- Equipment budget: CHF 200,000 / year.
- Operational budget including travel and conference organization: CHF 100,000 / year.

Space

- Component Certification Center: 150 sq. m.
- Information Security Center: 150 sq. m.
- Programming Resources Center: 100 sq. m.

Professor positions

The following professor positions will be advertised in connection with the Laboratory. Two of them are being funded through other means.

1. Trusted components

One of the most promising advances in software engineering is the use of reusable components to build applications. Success in this area depends, however, on strict processes and tools to guarantee that the components are of unimpeachable quality. This position requires a thorough understanding of software technology in industry and a mastery of a variety of quality techniques, from formal approaches to testing and software composition.

2. Practical applications of formal methods

Mathematically-based techniques of software development, also known as “formal methods”, set out to build systems whose correctness can be proved mathematically. In recent years these techniques have made great practical advances and some of them have been successfully used to significant industrial systems. The candidate for this position should have not only a profound mastery of the theory of mathematically-proven software but also a strong practical experience and good understanding of the needs and constraints of software projects in industry.

3. Software project management and economics

Software engineering success involves not only technical solutions but also effective management and cost estimation. Software project management and software economics have developed into a discipline of their own, leading for example to the precise specification of process models, to elaborate standards for assessing the maturity level of organizations in software development, and to detailed software cost estimation model. Expertise in this area is required at ETH to complement the more strictly software- technical specialties.

4. Programming languages

The use of a programming language is an obligatory step along the path to realizing any software goal. Programming language research is a fascinating area of software engineering, with a rich history in which ETH played a major role, and many future developments to be expected. The candidate for this position should master several complementary topics: the theory of programming languages, in particular formal semantic specification techniques; the practical language needs of software engineers in industry; compiling techniques. The candidate should also have a demonstrated ability to come up with novel designs for languages and language constructs.

5. Software engineering for parallel and distributed systems

Some of the most practically important needs for good software engineering arise in distributed and concurrent system; these needs are made ever more pressing by the development of the Internet and other networks. This area of application also raises some of the most delicate problems, having to do with maintaining consistency between remote machines, handling many different computation speeds and resources, providing replication when needed, cope with intermittent failure. Scientific leadership in this area is part of any successful software engineering program.

Contacts

The following persons serve as contacts. All can be reached via the Computer Science Department.

- *Component Certification Center* and coordination of the Laboratory as a whole: Prof. Dr. Bertrand Meyer

- *Information Security Center*: Prof. Dr. Jürg Nievergelt; Mr. Michael Näf; Professor of Information Security (new faculty member, name not public yet).
- *Programming Resources Center*: Prof. Dr. Hans Hinterberger.

8 SUMMARY

The Laboratory for Applied Software Engineering Research provides a flexible yet focused structure for efforts to further the reputation of ETH in software technology, and establish a strong basis for education and research in this field. It combines long-term work on theoretical foundations with a major practical component intended to produce quickly visible results, of direct benefit to the university and to the IT world at large.