

Radical Innovations of Software and Systems Engineering in the Future

Observations on Monterey Workshop 2002
Venice, Italy
October 7-11, 2002

What is the Monterey Workshop?

A by-invitation-only think tank sponsored by the U.S. military in order to anticipate future developments.

Large number of attendees from Europe

Keywords: Requirements, Formal Methods

Why this talk?

Observing what others are researching may suggest what areas are hot, and provide a cross-fertilization of ideas for use in our research here.

We finally find out that Java is *not* taking over the entire software engineering research community...especially not the avant garde

Some Things I Learned

There are too many dogs in Venice.

Venice has a smaller population than Las Cruces.

A lot of top researchers use Mac laptops.

Some Macintosh laptops' Apple logos are upside down compared with others.

Big name CS researchers are e-mail addicts.

Why is Software Productivity so Hard to Improve?

O. Nierstrasz

Berne

Software “engineering” is just a metaphor.
Software maintenance is 60% new features, 17%
fixing bugs, 18% porting.

What's the difference between software &
hardware? Hardware breaks if you *don't*
maintain it.

Post-Object-Oriented Programming

H. Hussman

Dresden

- OOP is really about bottom-up development
- Collaboration instead of functional decomposition
- “Service oriented programming”
 - extends component based programming
 - platform independent via network protocols
 - dynamic discovery of services

Software Components: the Challenges Ahead

M. Jazayeri

Vienna

Software Components

- how to compose?
- can they be trusted?
- how to handle component departures gracefully

Vienna Component Framework (VCF)

- facade covering COM+ CORBA EJB
- composition from different component models
- application compositing language
- component adaptation

Predicting Properties of Component Based Software Architectures

B. Kraemer

Hagen

Predictable Component Architectures using Dependable FSM's

Engineers prefer standardization & restriction over arbitrary flexibility

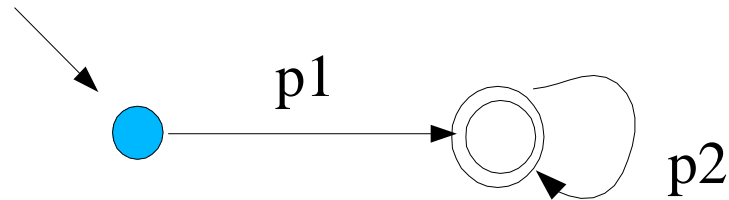
“Parameterized Contracts”

- environment dependent...provides interfaces
- usage dependent...requires interfaces

B. Kraemer

Hagen

```
provides {  
    ... p1(...);  
    ... p2(...);  
}
```



For each set of required interfaces,
a finite state machine.

Safety & Liveness Analysis of Component Systems

P. Inverardi
– connector free architecture

Aquila

Software Technology Available Today

B. Meyer

ETH/Eiffel Software

www.jot.fm Journal of Object Technology

- 6 million VB programmers, not 6M Comp Sci-ists
- want to improve their output? improve their compilers and components
- Software Quality is the biggest issue of software engineering, but decision makers don't care, except if there are major problems, or security
- low road = component certification center
- high road = theory for reasoning about programs with pointers, general framework for class provers
- bad components a major risk, perfectionism is good

Complexity of Systems

Jeanette Wing

CMU

Dimensions of Complexity

- functionality
- size and configuration
- lifetime
- openness
- physical look
- fault tolerance, security
- specificity
- communication & networking
- heterogeneity
- expectations of users

The build process
needs to be part of the
programming language

We need to make verification
(static and dynamic analysis)
truly lightweight, like compilation

The Inevitable Pain of Software Development

Dan Berry

Univ. of Waterloo

every major aspect of software development includes at least one step that is so painful that programmers habitually avoid it

biggest problem is remembering all the requirements.

you get more requirements while working on initial requirements. you forget to write them down in the excitement of coding, and feel guilt if you spend too much time on requirements, instead of coding.

changing requirements cause the most pain of all.

Future? of Object Orientation Panel

Much talk of POOP and POSD

e.foo

- if you allow assignments to it, OOP fails
- if no field by that name, invoke a method?

UML = the anathema of design patterns

- hype has displaced the lessons of decades

Objects are wired, not pluggable, in practice

MIT students all hate Swing, template classes

Services and Layered Architectures

M. Broy

TU Munich

Service-based software engineering paradigm

- Services everywhere, a la telecom/web
- Algebraic reasoning of services, components, and layers

Agile Modeling with UML

B. Rumpe

TU Munich

Software & Systems Modeling (Springer Journal)

Model based development

B. Rumpe

TU Munich

System (exe) generated from class diagram,
statecharts, + Java

Test code generated from object diagrams, sequence
diagrams, etc.

Refactoring = improving design while maintaining
functionality. Can be done systematically:

1. identify old structure
2. add new structure & queries, compile
3. identify invariants to relate old and new
4. add code for new, + invariant checks,

wherever

old was used

5. modify places where old was used
6. simplify

Domain Engineering

Denis Bjorner

Denmark

Railway system has its own logic, understand it fully
mathematically, before coding.

Use separate languages for rules, regulations, and
stimuli (observed behavior)

Reason about human behavior good/bad/criminal

High Confidence Systems of Embedded Systems

Luqi

Naval Postgrad Sch

High Confidence:

no failure due to errors in system, faults in environment, or attempts to compromise the system

General Resource Framework for Real-Time Systems

O. Sokolsky

U. of Pennsylvania

Resource modeling - oft-neglected, critical aspect of
real time systems

Resource classes - static/dynamic attributes

Model-Integrated Computing

Gabor

Vanderbilt

Many layers of middleware

Applications closely tied to whatever middleware
they use

From problem driven to platform specific models

M. Cerioli

University of Genoa

- mobility and decentralization
 - small devices rely on external sources
 - users need to access their data everywhere
 - middleware provides efficient correct off the shelf primitives
 - Architecture Specific Model (ASM)
 - peer-to-peer, platform independent

Model Based Development of Distributed Embedded Systems

F. Kordon

University of Curie, Paris

- objects make lousy distributed systems
 - thick “middleware” is slow & makes changes difficult
 - hey this guy had a class diagram with unnamed associations

Software Engineering for Modern Distributed Systems

Picco

Politecnico di Milano

- Previous major steps dictated by increasing dynamicity, decentralization, flexibility
- Next steps driven by scale, mobility due to reconfiguration (mobile), huge scale, pervasive
- Shift from system integration to federation
- Design distributed systems that grow or shrink dynamically: design a federation of component

Model Generation for Legacy Systems

B. Steffen

Universität Dortmund

- Person introducing new technology must be prepared to take 90% of the pain
- A computer telephony app with mandatory new (minor) release every 2 months; much/most effort spent on regression testing
- “Stable states” vs. internal states (gray)
- 4000 states, 6000 edges, massively folded
- from objects to rules, hiding details, partial ordering

Module Test Driver + Output Analyzer Generator

V. Berzins

Naval Postgraduate School

- Spec -> MTDOAG -> MTDOA
- User provides test input generator
- Human effort behind testing becomes virtually independent of the number of test cases

View Integration

M. Wirsing

LMU Munchen

- Different views for different stakeholders/viewpoints
- Separate view from viewpoint
- View translation
- Exotic algebra for all this

Knowledge Structuring and Representation in Requirement Spec.

G. Reggio

University of Genova

formalism \neq method

- UML, with underlying formal semantics
- include good ideas from classic work
- UML deficiency: class diagrams show what messages can be received, not messages can/will be sent.

High Assurance Systems Require Goal Orientation

A. van Lambsweerde University of Lo...

- detection/fixing defects generate other defects
- strong guarantees require formal methods, but requirements engineering needs lightweight methods
- requirements elaboration hard, software+environ, conflicting concerns, unexpected behaviors
- goals need to be refined until assignable to single agents

Automation of Sys Dev Using Natural Language Proc and 2 Level Grammar

B.S. Lee

University of Alabama

- phases of software development have problems: linguistic ambiguities, inconsistency among phases, poor reusability of req. docs, and slow prototyping.

Reliable Agent Systems

Dave Robertson

University of Edinburgh

- Agents are about confluence. It is the blend of symbolic reasoning, formal specification, concurrency and maybe mobility
- Small and nimble, loosely connected, maybe disposable, fault tolerant but low dependable, domain specific, close to people

Reliable Agent Systems

Dave Robertson

University of Edinburgh

- It is believed to be impossible now for medical practitioners to know all they should know.
- Automated support is being made available in e.g. oncology via open source components accessed via web portals
- Agent=(dialog strategy, d.s. translation into message passing, decision procedures)

How to Reach Software Ultra Reliability?

M-C Gaudel University de Paris-Sud / CNRS

- for critical systems, need $<10^{-7}$ failures/hour
- what can be concluded from a period of failure free behavior? paradox: you can predict weak reliability, but not high reliability
- state of program = approximation of state of system; design for error margins and random perturbations
- goal is to compromise between reliability and efficiency, i.e. can't *over* constrain system to be safe

Performance

M Simonetta

University of Venice

- integration of quantitative analysis of software sys
- feedback from quantitative analysis early in life cycle
- predict quantitative behavior
- stochastic processes used in performance model
- EQN and LQN
- model transformation: spec model -> perf. model

Model Checking Complex Software- a Memory Perspective

Rangarajan

Honeywell

—

Architecture Based Model Driven Software and System Development for Avionics and Safety Critical Systems

Lewis

??

—

System-of-Systems Development from an Object-Oriented Paradigm

Caffall

??

—

Module Dependences in Software Design

D. Jackson

MIT

- key motivation in software design is decoupling
- reducing dependences between modules

Parnas' Uses Model

D. Jackson

MIT

- module A uses module B when correct working of A depends on correct working of B
- if A uses B, and B uses C, then A uses C
- but the whole point of dependence analysis is to remove transitivity

New Model

D. Jackson

MIT

- module = executable syntactic unit
- A s-uses B: A depends only on B satisfying spec S;
A views B through S
- A module may be used by multiple modules, via
different specifications
- A name-uses B if A's text mentions B

Panel: Radical Innovations in SE

Panel votes on which papers in the proceedings are actually “radical innovations in S&S engineering”. Proceedings will become a LNCS publication (probably) for most of the papers.

Maybe 6-8 papers in a special journal issue?

70% of the papers were on pervasive computing

Documentation as part of the design process DH

Evolution, not maintenance