Software Engineering and Programming Languages Research at The Ohio State University

More details at http://www.cse.ohio-state.edu/~weide/se

Presented by Paul Sivilotti (http://www.cse.ohio-state.edu/~paolo)

Software Engineering: Research Focus

- Practical methods and foundational theories for the construction of high-confidence software
  - “Software errors cost the U.S. economy $59.5 billion annually (about 0.6% of GDP)” [NIST 02]
  - “The nation depends on software that is often fragile, unreliable, and extremely difficult and labor-intensive to develop, test, and evolve. Our ability to construct...the complex software systems that lie at the core of our economy is painfully inadequate.” [PITAC 99]

- Theme: modularity
  - Algorithms, specifications, design principles, code, languages, compilers, tests, tools, ...

Overview of People

- RSRG: Bruce Weide, Bill Ogden, Tim Long, Wayne Heym, Paolo Bucci
- Neelam Soundarajan
- Paul Sivilotti
- CETI: Rajiv Ramnath and Jay Ramanathan
- PRESTO: Nasko Rountev

Courses: www.cse.ohio-state.edu/cgi-bin/syllabus-view.cgi

- 788: Nasko Rountev, Winter 2010
  - Prerequisite: undergraduate discrete math
- 763: Distributed systems (Paul), Spring 2010
- 756: Compilers (Nasko), Spring 2011
- Applied courses
  - Capstone project courses: 758, 762, 772
  - Enterprise architecture: 794J, 794K; email ramnath@cse
  - Applied component-based software: 668 (Winter 2010), 767 (Spring 2010), 794R (Spring 2010)
  - Undergraduate software engineering: 757
RSRG Vision: Verified Software

- RSRG: Resolve/Reusable Software Research Group
- Addressing Hoare’s Verifying Compiler Grand Challenge:
  - “Build a compiler that will generate object code for a program component if and only if it can mathematically prove that the component will behave as specified.”
- Research issues:
  - Mathematical/logical foundations for software
  - Specification and programming language(s)
  - Software design (“design for verification”)
  - Tools to support envisioned verified software paradigm
- http://www.cse.ohio-state.edu/~weide/rsrg

Vision for a Verified Software Paradigm

- Basic day-to-day programming process (“main loop”)
- Consultative software design/development process
- Demo to Illustrate Ideas/Tools
  - http://resolve.cse.ohio-state.edu:8080/ResolveVCWeb
  - Simple example:
    - QueueConcatenate.rc (contract specification)
    - IterativeQueueConcatenate.rr (one realization)
  - Look at other examples on your own:
    - “Tracing tables”
    - Unicode (human-readable) verification conditions

Neelam: Reasoning About Pgm. Behavior

Key questions:
- How do we specify precisely the expected behavior of various types of programs/systems?
- How do we test actual systems to see if they meet their specs?
- How do we show/verify that systems meet specs?

What types of systems?
- Object-oriented systems, aspect-oriented systems, distributed systems …
- http://www.cse.ohio-state.edu/~neelam

Reasoning About Designs

What about the design underlying systems?

System designs often in terms of **design patterns**

So: Can design patterns – usually described informally via examples – be specified precisely?

Advantages of doing so:
- Can verify system impl. is consistent with its intended design
- Can test system to check this consistency
- Can ensure, as system evolves, that its design integrity is preserved
Reasoning About Designs (contd)

Potential Risks:
- Trying to formalize patterns may result in loss of flexibility: Very bad since flexibility is the key source of patterns’ power

Surprising Result:
- It is possible to formalize patterns in such a way that we get precision without any loss of flexibility!
- In fact: The approach being developed often lets us identify additional dimensions of flexibility

Example: the Observer design pattern

Observer Design Pattern (Gang of Four Book)

Example: Observer is a pattern used when a set of observers, ob1, ob2, …, need info about current state of a subject, sub

Pattern: When state of sub changes, invoke Update() on ob1, ob2, … These calls update ob1, … states appropriately

Question: When can state of ob1 change & how?

Std Answer: Only due to calls to Update()

New Ans (based on formalism): Also at other points – as long as certain simple conditions are met!

Details: In Neelam’s 885 talk

PRESTO: Nasko Rountev
- PRESTO: Program Analyses and Software Tools
- The intersection of software engineering, programming languages, and compilers
- Run-time analyses
  - Effective algorithms: uncover useful information from millions of run-time events
  - Efficient algorithms: reasonable run-time overhead
- Compile-time analyses
  - Semantic foundations: e.g., for different languages, for different levels of abstraction
  - Analysis algorithms: e.g., trade-offs between cost and precision - useful precision for 1 million lines of code?

- http://www.cse.ohio-state.edu/~rountev/presto

Examples
- Run-time analyses for testing and debugging
  - Memory leak detection
  - Checkpointing for long-running applications
- Run-time analyses for understanding and evolution
  - Analysis of parallelism
- Foundations of compile-time analysis
  - Theoretical foundations for scalable interprocedural dataflow analysis in the presence of large libraries
  - Analysis of objects in distributed Java applications
- Compile-time analysis for understanding/evolution
  - Reverse engineering of UML sequence diagrams

Memory Leak Detection for Java Programs
- Multiple factors for analysis of containers
  - Overall memory consumption
  - Memory taken up by an individual container
  - Staleness of a container
- Ongoing: collaboration with IBM T.J. Watson Research on memory/performance analysis of real-world enterprise Java applications (e.g., object churn)
Run-time Analysis to Find Parallelism

- Given a sequential program, how can we evolve it to take advantage of multi-core hardware?
  - Automatic compile-time parallelization is limited
  - Use programmer intelligence together with useful tools
- Tool for characterization of potential parallelism
  - Instrument the program to track run-time data dependencies and control dependencies
  - What is the earliest completion time with unlimited resources?
  - Low instrumentation overhead
- Can we actually achieve this upper bound in practice?

Paul Sivilotti

- High-confidence distributed systems
- Many aspects to “high confidence”:
  - Security, correctness, robustness, reliability
- Many challenges (for distributed systems)
  - Partial synchrony, partial failure, loose coupling, dynamic assembly
- Research projects to address these challenges:
  - Theory
  - Tools
  - Algorithms
  - http://www.cse.ohio-state.edu/~paolo

1-Theory: Maximality

- When is a program correct?
  - Answer: When it “satisfies” its specification: when all program behaviors are permitted by specification
- But many specifications are nondeterministic
  - E.g., starting number for TCP/IP sequence numbers: implementations are free to make any choice
- Consequences of this free choice
  - Security holes (network fingerprinting) and testing problems (masking: false negatives)
- Theory of maximality: a program is capable of all behaviors permitted by specification
- Challenges: non-compositional; non-implementable under some conditions

2-Tools: Defense Against Code Injection Attacks

- Common structure for Web applications
  - Accept input from user (string)
  - Generate an SQL query from this input (string)
- Program is generating a program!
  - User input is just a string (weak typing)
  - Malicious user input causes generated program to do surprising things
    - Known as a “code injection” attack
- Our approach:
  - Has the flexibility of weak typing (easy to use)
  - Has the robustness of strong typing (safe)

3-Algorithms: Failure Locality

- Faults should be contained to small neighborhoods
  - But: in an asynchronous system, failure can not be reliably detected
- Example with a classic resource allocation problem
  - A graph of processes competing for resources
  - Neighbors not allowed to use the resource simultaneously
- For this (important) class of problems
  - For most algorithms, a single process failure can result in the entire system starving
  - Even for optimum algorithms, the best achievable failure locality is surprisingly bad
- Research questions:
  - Trade-offs between synchrony and failure locality
  - Trade-offs between performance (e.g., message complexity) and failure locality

NSF Industry-University Collaborative Research Center
Rajiv Ramnath and Jay Ramanathan
http://www.ceti.cse.ohio-state.edu

- Affiliated with the Center for Experimental Research in Computer Systems (CERCS) at Georgia Tech
- Leverage over 50 inter-disciplinary researchers & practitioners
- 35+ Affiliated Companies
  - CERCS for Enterprise Transformation and Innovation (CETI)
Enterprise Systems Need to Support the Changing Face of Business

Non-Routine Requests
Adaptive Complex Enterprise (ACE)

Increasing complexity

Uncertainty and competitive pressures
Product requirements uncertainty
Large number of request types
Non-routine requests
Demand unknown
Project management uncertain
Process flow uncertain
Transaction type unknown
Required capabilities unknown
Transaction sequencing unknown
Availability of capabilities unknown
Virtualization & delayed binding

Embrace variation

CETI Program Areas

- ACE: Modeling and Analysis Frameworks for Adaptive Complex Enterprises
  - Lean behaviors, Resilience, Pinpointing areas of innovation
- Knowledge-based collaboration systems for Communities of Practice
  - Portals for enabling collaboration, dissemination, self-service
- Integrated Development Environments for Adaptive Complex Environments
  - Developing, managing and monitoring the ACE
- Software Engineering Education
  - Agile and structured SOLC, ITIL, Technology Strategy and Management, Enterprise System Architecture Design and Evaluation
- Work @ CERCS:
  - Virtualization, Information Fusion, Middleware, Autonomic computing

CETI - Selected Accomplishments

- DARPA - Workflow for the Virtual Enterprise
- City of Columbus
  - “Sense and Respond” Strategic Plan
  - 311 architecture validation, charge-back model, Evaluation of 311 for non-routine use
- OSUMC
  - Case study in implementing Lean methods in IT: Improvement of the PC delivery process
- Nationwide Insurance
  - Analysis framework for Capacity Management
  - EA Framework for Innovation Management
  - Collaboration tools for the Architecture Review process
  - Data Center power conservation
- Enterprise Systems and Cloud Computing Laboratory
- Securities Exchange of China
  - Executive session in collaboration with Fisher College
- Over 40 Capstone projects in collaboration with Industry

CETI Research

- Focus on
  - Problems identified by industry, researched by CETI
  - Integrating computer science, systems engineering and business
  - Knowledge consolidation and curriculum development
  - What works, and what is needed to make things work
- Through industry projects as the research vehicle
  - Graduate students do MS (and some PhD) theses on complex industry problems (currently about 15 students)
  - Direct value is delivered to the sponsoring industry organization
  - Employees are mentored and trained during project delivery so that knowledge is retained and leveraged many folds
- Aimed at workforce development for those who need skills in:
  - Technology management
  - Enterprise architecture roles
  - Where globalization is causing a huge impact
- Utilizing work from from Georgia Tech, Other IUCRS, ITSMF, TOGAF, SEI and others

Questions?