ICSE 2012

- June 6
  - Keynote 1
  - Cost Estimation for Distributed Software Project
  - Characterizing Logging Practices in Open-Source Software
  - Combine Functional and Imperative Pgrm for Multicore Sw: Scala & Java
  - Sound Empirical Evidence in Software

Testing
    - Identifing Linux Bug Fixing Patch
    - Active Refinement of Clone Anomaly Reports
- June7
    - Keynotes 2: Sustainability with Software - An Industrial Perspective
        - Green IT
        - What can we do?
        - Green by IT
    - On How Often code is cloned across repositories
    - Graph-based analysis and prediction for sw evolution
        - graph are everywhere
        - predictors
        - Conclusion
    - What make long term contributors: willingness and opportunity in OSS
        - approach
        - summeray
    - develop of auxiliary functions: should you be agile?
        - experiment
        - research questions
        - result
    - Static Detection of Resource Contention Problems in Server-side

script
- Amplifying Tests to Validate Exception Handling Code
- A tactic-centric approach automating traceability of quality concerns

June 6

Keynote 1

finance is not money

Cost Estimation for Distributed Software Project

PM PMPM PM

Context-specfic solutions needed!

Early user paticipation is key!

Characterizing Logging Practices in Open-Source Software

Common mistakes in logging messages

logloglog

9027

45%	
27%	
26%	verbosity

2%	

logLogEnhancer

text

code clone log clone

Combine Functional and Imperative Pgrm for Multicore Sw: Scala & Java

ScalaJavaJVM

- **Java:**
  -
  -
  -
  - Wait/Notify

- **Scala:**
  -
  - Actors,
  - lists, filters, iterators
  - while

- , OO
- import java.* java
- auto type inferance

4

scala java38%Testdebug

testdebug

scaladebugdebug

scalajava2.6%15.2%

-
  - scalajava
  - **4**
    - scala 7s @ 4 threads
    - java 4si @ 8 threads
    - **median**
      - 83s scala
      - 98s java
  - 32core: best scala 34s @ 64 threads
-
  - javascalability
- **scala**

    - 45%
    - 85%

-
    - 23%scala
    - 77%java

  multi-paradigram are better

Sound Empirical Evidence in Software Testing

Test data generation

Large Empirical Studies - not always possible

For open source software - big enough

Identifing Linux Bug Fixing Patch

- **current practice:**
    - manual

- **Current research:**

    - keywords in commits
    - link bug reports in bugzilla

Try to solve classification problem

- **issue**
    - pre-identified
    - post-identified

- **data**
    - from commit log

- **feature extraction**
    - text pre-process stemmed non-stop words

- model learning

research questions

Active Refinement of Clone Anomaly Reports

motivating

- code clones, clone groups
- clone used to detect bugs

- anomaly : inconsistent clone group many anomaly clone are note bug, high false positive

approach

- reorder by sorted bug reports

June7

Keynotes 2: Sustainability with Software - An Industrial Perspective

Sustainability

- **Classic View: Idenpendent view with overlap**
  - Social
  - Environment
  - Economic

- **Nested viw**
  - **Environment**

        - **Social**
            - Economic

Triple bottom line

- **economic**
  -global business, networks , global econ

- **env**
    - natural res, climate change, population grow

- **social**
    - awareness, connectivity, accountability

Green IT

- **reduce IT energy**
    - more than 50% cooling - doing nothing

- **mini e-waste: not properly recycled**
    - 80% in EU
    - 75% in US

- foster dematerialization

In-Memory Technology: Expected Sustainable Benefits

What can we do?

- consider all software lifecycle phases in your design
- avoid energy expensive behavior in your codes
- design lean architectures

Green by IT

- 2% green IT
- 98% green IT

On How Often code is cloned across repositories

Line based hashing code clone detection

never do anything harder than sorting

hashing a window of 5 lines of normalized (tokenized) code, dropping 3/4 of the hashing

ccfinder3, 4presionrecall

14%	type1
16%	type2
17%	type3 (not really type2)

Graph-based analysis and prediction for sw evolution

graph are everywhere

- internet topology
- social net
- chemistry
- biology

in sw - func call graph - module dependency graph

developer interaction graph - commit logs - bug reports

experiment 11 oss, 27~171 release, > 9 years

predictors

- **NodeRank**
  - similar to pagerank of google
  - measure relative importance of each node
  - **func call graph with noderank**
    - compare rank with severity scale on bugzilla
  - **correlation between noderank and BugSeverity**
    - func level 0.48 ~ 0.86 varies among projects.
    - model level > func level
- **ModularityRatio**
  - cohesion/coupling ratio: IntraDep(M)/InterDep(M)
  - forecast mantencance effort
  - **use for**
    - identify modules that need

redesign or refactoring

- **EditDistance**
  - bug-based developer collaboration graphs
  - ED(G1,G2)=|V1|+|V2|-2|V1 V2|+|E1|+|E2|-2|E1E2|
  - **use for**
    - release planning
    - resource allocation

graph metrics

- **graph diameter**
  - average node degree indicates reuse

- clustering coefficient
- assortativity
- num of cycles

Conclusion

"Actionable intelligence" from graph evolution

- studie 11 large long-live projs
- predictors
- identify pivotal moments in evolution

What make long term contributors: willingness and opportunity in OSS

OSS don't work without contributors form community

mozilla (2000-2008)

10^2.2 LTC <- 2 order -> 10^4.2 new contributors <- 3.5 order -> 10^7.7 users

gnome (1999-2007)

10^2.5 LTC <- 1.5 order -> 10^4.0 new contributors <- 3.5 order -> 10^6.5 users

approach

- read issues of 20 LTC and 20 non-LTC
- suvery 56 (36 non-LTC and 20 LTC)
- extract practices published on project web sites

summeray

- Ability/Willingness distinguishes LTCs
- **Environment**

- **macro-climate**
  - popularity
- **micro-climate**
  - attention
  - bumber of peers
  - performance of peers

regression model

newcomers to LTC conversion drops

actions in first month predicts LTCs

- 24% recall
- 37% precision

develop of auxiliary functions: should you be agile?

a empirial assessment of pair programming and test-first programming

can agile help auxiliary functions?

experiment

- pair vs solo
- test-first vs test-last
- students vs professors

research questions

- r1: can pair help obtain more correct impl
- r2: can test-first
- r3: dst test1 encourage the impl or more test cases?
- r4: does test1 course more coverage

result

- **test-first**
  - higher coverage
  - non change with correctness

- **pair**
  - improve on correctness
  - longer total programming time

Static Detection of Resource Contention Problems in Server-side script

Addressed the race condition of accessing database or filesystem of PHP

Amplifying Tests to Validate Exception Handling Code

A tactic-centric approach automating traceability of quality concerns

tactic traceability information models