Project 9 - Smarter Testing of Evolving Software Systems

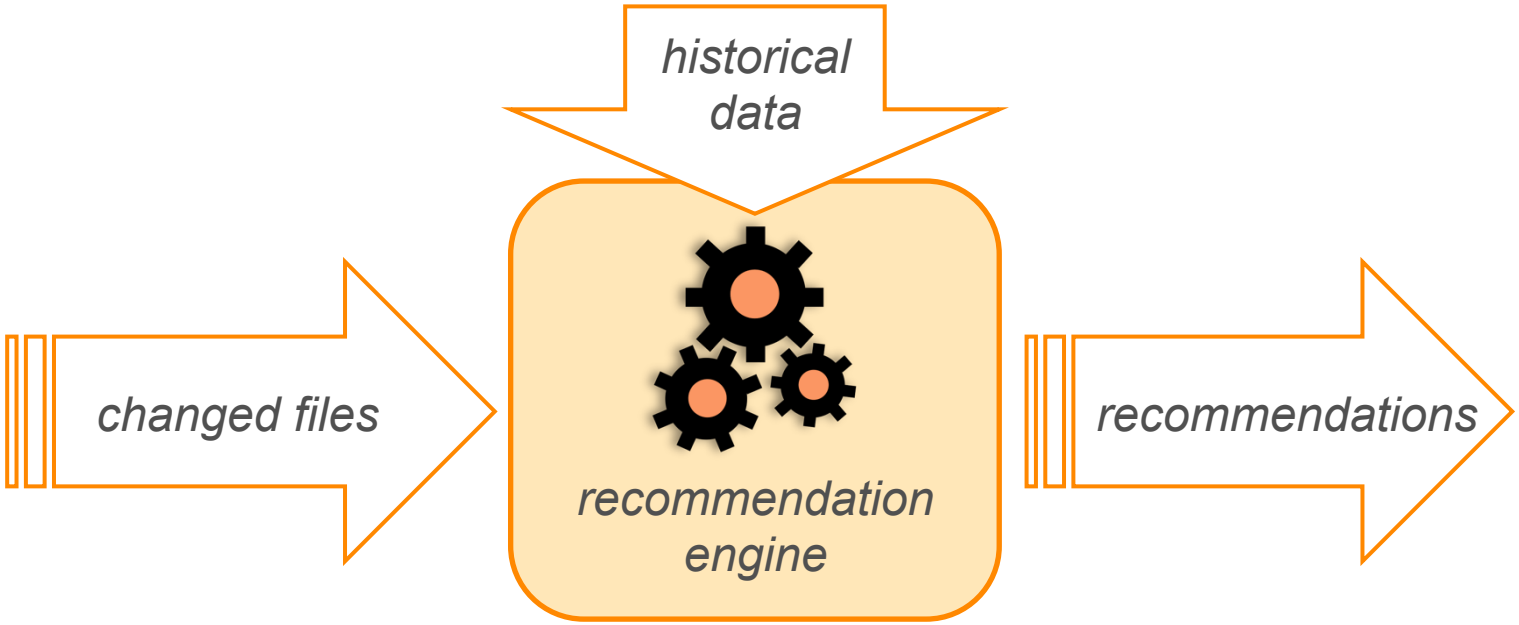
Leon Moonen, Thomas Rolfsnes,
Stefano Di Alesio, Razieh Behjati

Overview

- (short) project direction
- project status
- overview of plans for 2016

Test Recommendations based on Evolutionary Coupling

- *frequent pattern mining* on development metadata (e.g. in versioning and issue mgnt. systems)
  - two files that frequently change together have a dependency
  - if one is changed, impact/regression of other should be tested
  - *"other customers that watched X, also enjoyed Y"*



Project 9 Status (1/2)

- HaRT: History Based Recommendations for Testing
  - initial prototype delivered earlier in 2015 (demo March UPW)
  - Carl (KM) uses this prototype in current testing efforts
- started generalization to CSC context
  - made core recommendation engine independent of partner
    - adapters to `git` versioning system (CSC) and `track` (KM)
  - analyzed bug/issue linking in CSC versioning data
    - *original hypothesis: multiple changes/commits that address the same issue should be seen as one large combined transaction*
  - ⚡ for project analyzed, only 14% of commits linked to bugs/issues
    - dependent on project culture; CSC also has 100% linked projects
  - no problem to make recommendation w/o linking changes
    - ➢ needs *qualitative evaluation* (and *gold standard*)

Project 9 Status (2/2)

- evaluated using *'re-enactment':*
  - try to complete known commits based on partial commit
  - ➢ identified major *shortcoming* of state-of-the-art mining algo (*Rose*)
    - only produces results in ~25% of cases for KM and CSC
    - much lower for linux, httpd (~10%); git, mysql, webkit (~15%)
- *TARMA:* (family of) new algorithms that address this issue
  - produce *results in all cases*
  - variants impose requirements on patterns
  - currently evaluating precision: *never worse than Rose*
  - \+ consistently better than alternative mining approach using SVD
- just in: new collaborator, KM Dynamic Positioning
  - generalization to other major version mgmt / issue tracking (TFS)
  - *"parsable"* historical test execution data available *(gold standard)*

Shared Topics for AWP 2016

- *use software analytics to direct testing efforts*
  - where are *bugs/issues clustered* in the system (over time)
  - components/files/methods with high *change rates (code churn)*
  - monitor test execution data & continuous integration data (CSC)
  - *combine* with recommendation results for *prioritization*
- investigate techniques that mimic *"aging of evidence"*
  - architecture of software system is known to change over time
  - parts that used to have a dependency may no longer have one
  - decrease impact that old co-changes have on recommendation w.r.t. more recent ones

KM related topics for AWP 2016 (1/2)

- (continued) evaluation of automated recommendations with respect to KM's expert opinion (Carl)
  - define *protocol* for more systematic tracking

change ID	date	expert opinion	HaRT output	delta	added to testplan + why	removed from testplan + why	HaRT error + why

  - target new testing efforts in Q1 2016
- create fine-grained mapping between tests and methods
  - add runtime traceability / dependency tracking for KM-PP
  - *hypothesis*: path profiling using coverage analysis instrumentation
  - challenge: getting data out of embedded system(s)
    - highly specialized to KM context, Sim researchers no expertise, 3rd party?
    - creates code coverage info that is of interest to KM
  - compare path profiles for number of tests
    - analysis to remove 'noise' (startup code, setup/teardown of test fixtures

KM related topics for AWP 2016 (2/2)

- transfer our approach to Dynamic Positioning (KM-DP)
  - develop adapter to TFS (+ system specific use)
  - evaluate performance in new context
    - compare to KM-PP / CSC / open source systems
  - DP enforces issue linking during commit, and TFS supports hierarchical organization of tasks / issues (work items)
    - ➢ *assess benefits of using of these to combine commits*
    - (concrete use of hierarchies is project specific; make abstraction)
  - can historical test execution data establish a gold standard?
    - *"could we have recommended tests that failed"*
  - can we exploit large level of automated (unit) testing?

CSC related topics for AWP 2016 (1/2)

- *granularity:* refine from file level to method level
  - enables more fine-grained (re)testing
  - also of interest to KM: supports ‘chopping up’ large test procedures
  - requires (lightweight) code analysis
  - evaluate benefits/drawbacks of fine-grained recommendations
- extend prototype to context of developer driven testing
  - developers needs, usage scenarios, and desired integration
  - evaluate recommendation quality in new context (usability/impact)
    - ‘developer satisfaction' interviews
    - compare teams with and without *‘treatment’*
  - from recommending tests to recommending change
    - *“other developers that changed these methods, also changed …”*

CSC related topics for AWP 2016 (2/2)

- compare to source code analysis based advise
  - CSC currently evaluating Coverity Test Advisor
    - commercial tool for impact analysis/test prioritization
  - compare their 'advise' to HaRT recommendations
  - *hypothesis*: HaRT will find more *semantic* associations
    - also language independent and finds cross-language dependencies
    - but these are complementary techniques

thanks!

Leon Moonen
leon.moonen@computer.org
https://leonmoonen.com/