Prioritizing Component Compatibility Tests via User Preferences

ICSM 2009

Il-Chul Yoon
lyoon@cs.umd.edu

Joint work with Alan Sussman, Atif Memon, Adam Porter
University of Maryland

Motivation

- Software systems are designed for large number of configurations
  - Developers work on just a few instances
- Difficulties for software testers
  - Hard to test all configurations
  - Limited computing resources
  - Configurations are not equally important

Rachet

- Process and infrastructure for testing configurations
  - Define configurations for a software system
  - Decide how to sample configurations
  - Produce a set of configurations satisfying the criteria
  - Test configurations on a cluster of machines
    - realized in a client/server architecture

Modeling Configuration Space

- ACDM= (CDG, Ann)
  - Annotated Component Dependency Model
  - CDG captures dependencies
  - Ann: component versions & constraints
- Testing all possible configurations is very expensive
  - ~ 3500 configurations
  - ~ 3 hours per configuration
  - ~ 1.2 yr on a machine
Sampling Criteria

- Only test relations between components with direct dependency
  - A DD-instance
    - component version to build
    - dependency for the build
  - Produce configurations that test each DD-instance at least once
    - Configuration
    - an ordered set of DD-instances

Build Test Plan

- Create test plan by combining configurations
  - Share effort to build common prefixes

Execute Test Plan

- Server assigns paths to nodes to clients
- Client builds components on a VM
- VM states are cached for reuse

Limitations of Our Previous Work

- Did not take into account developer preferences
  - Tested lowest-cost configurations first
  - Developers have different levels of interest in different components and their versions
- Assumed unlimited testing time
  - In worst case, developers must wait until full test plan completed before obtaining most desired results
  - Developers want to see more important/desirable results within limited time
Considering Preferences in Test Plan Execution

- Impractical for a developer to manually give an order across all configurations
  - Need a systematic method to prioritize configurations
  - Test configurations most important to developers first

- A solution approach that considers preferences
  - Capture developer preferences
  - Modify plan execution algorithm

Capturing Developer Preferences

- Component preferences
  - User specifies partial order between components
  - Rachet automatically produces a total order

- Version preferences
  - Could use any form specifying relative user interest across versions
  - Implementation uses positive integer for each version – more highly preferred versions have larger values

Prioritizing Configurations

- Define preference vector for each component version
  - e.g., (0,0,0,0,0,3,0) for F_3

- Annotate plan nodes with preference vector
  - Sum up preference vectors of component versions in the prefix (a path from the root in the test plan)
  - Gives a total ordering over all prefixes

Preference-based Plan Execution

- Goal: Test important configurations earlier

- A simple strategy
  - Input: a test plan and a client
  - Sort plan nodes not yet tested by preference
  - Return the path for the first node – the one with highest preference not yet tested
Experimental Setup

- Subject System: InterComm
  - Middleware to support coupled scientific simulations
- Test scenarios
  - Prefer configurations to build InterComm (paths to leaf nodes in test plan) with more recent component versions
- Simulation parameters
  - Used data from actual experiment
  - Number of clients: 4
  - Plan execution strategy
    - cost-based vs. preference-based
- Measurement
  - Config completion time, results, preference

Preference-based vs. Cost-based

- Preference-based quickly produces results for configurations with higher interest
- But, total plan execution takes longer
  - From lower reuse of local build effort in clients

Preference-based Plan Execution

- Secondarily, consider cost-related factors
  - Increase reuse of cached prefixes
  - Reduce redundant effort across clients
- A strategy that accounts for cost/performance
  - Input: a test plan, a client, window size (W)
  - For each node not yet tested, compute
    - L: additional work required when reusing cached prefix in the client
    - S: number of clients that have cached prefixes of the node
  - Sort nodes by preference
  - Consider the first W nodes and return the path for a node with
    - min(L)
    - otherwise, min(S)

Experiment Setup

- Test scenarios
  - Prefer configurations to build InterComm with newer component versions
  - Prefer configurations to build InterComm with specific components
- Simulation parameters
  - Number of clients: 4
  - Plan execution strategy
    - cost-based vs. preference-based
  - Window size
    - 1,16,256, n (plan size)
- Measurement
  - Config completion time, results, preference
Effect from Increased Window Size

- With larger window size, cost-related factors are considered more heavily
  - To increase reuse locality
  - And total plan execution time decreases

---

Related Work

- Consider user preference for test-case prioritization
  - Prefer user-interested pair-wise interactions [Bryce05]
  - Consider user preference to prioritize system test cases [Srikanth05]
  - Share basic idea

- Continuous integration [XP]
  - Integrate changes frequently and test build on field configurations
  - Rachet can provide automated support for continuous integration

- Support of automated configuration testing
  - Test lab automation via virtualization [by IBM, Rational, VMware]
  - Configurations are tested on manually customized VMs
  - No mechanism to produce / prioritize configurations
Conclusions and Future Work

- Presented a method to prioritize the order to test configurations
  - A simple way to specify developer preferences
  - A plan execution algorithm that considers preferences and cost

- Results from empirical studies
  - Help developers to obtain results for preferred configurations quickly
  - Balance preference and total plan execution time

- Future work
  - Incrementally test configurations with component evolution
  - Other types of coverage criteria
  - Extension to functional and performance testing