Software Engineering Challenges- IT Services

Industry Perspective

Santonu Sarkar
Software Engineering Technology Labs (SETLabs)

Awards and Accolades

- One of the World’s Most Respected Companies - Financial Times-PwC annual survey
- One of the top 3 IT services company in the world - Business Week
- “Infosys is a role model for companies everywhere in financial transparency.” - Forbes
- First Indian company to publish its financial statements in conformity with US GAAP
- Amongst the first companies in the world to be certified at CMMI Level 5
- Global benchmark used for on-time, on-budget, on-quality delivery in IT Services
- Over 95% business from existing clients

- 1.4 million resumes in FY 06; one in every 65 applicant selected
- Training: certified to be equivalent to BS in Comp. Sc. in the US
- Scaling up: capability to train 14,000 employees in a year at Mysore
- Employees of 41 nationalities
- Amongst the “100 best places to work for in IT” in the US
- First company to leverage the Global Delivery Model
- First Indian company to list on NASDAQ
- First Indian company to offer a comprehensive ESOP plan

Next Generation Outsourcing of Software Services

Business Application Development and Maintenance Scenario

Yesterday

- Client Owned
- Developed and Maintained by Client

Today

- Hardware & Software Infrastructure
- Business Applications
- Client Owned
- Development and Maintenance Outsourced

Tomorrow

- On Demand
- Transaction Oriented
- Limited ownership
- Built from reusable components/services

Global Delivery Model
Software Quality Assurance

- Business Process Model Centric Requirement Analysis
- Model Driven Design
- Software Quality Assurance
- Large System Development, Comprehension, Modernization
- Collaboration Platform for community based requirement analysis
- Virtualization
- Service Oriented Architecture
- Software As a Service

Other Research Areas at SETLabs (Illustrative)

<table>
<thead>
<tr>
<th>Dynamic Processes</th>
<th>Malleable Architecture</th>
<th>Personalised Information</th>
<th>Pervasive Infrastructure</th>
<th>Transformational IT Management</th>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Process Management</td>
<td>Web Services</td>
<td>Mobile Computing</td>
<td>Outsourcing</td>
<td>JOEE</td>
<td>Net</td>
</tr>
<tr>
<td>Enterprise Architecture</td>
<td>Content Management</td>
<td>Service Networks</td>
<td>Strategy</td>
<td>Solution</td>
<td></td>
</tr>
<tr>
<td>Software Architecture</td>
<td>Business Intelligence</td>
<td>RFI</td>
<td>Methodologies</td>
<td>Regulatory</td>
<td></td>
</tr>
<tr>
<td>Knowledge Engineering Ontology</td>
<td>Grid Computing</td>
<td>Warehousing</td>
<td>Compliance</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Agenda**

- Software Engineering Challenges
- Software Engineering Research Agenda at SETLabs
- Research Projects- Overview
  - Research Project- Detailed (Active Collaboration with Purdue University)
    - Current Work
    - Work in Progress
    - Tool

**Software Engineering Research at SETLabs - An Illustrative List**

- InFlux - Requirements Engineering solution
  - Comprehensive RE methodology covering Requirements elicitation, analysis
- InFlux Application Architecture solution
  - Comprehensive methodology for blueprinting an application architecture based on high level business and technical requirements
- Intelligent Production Support Platform
  - Reduces turn around time for incident diagnosis through system and diagnostic knowledge embedded inside an expert system
- Automated test-case generation from requirements
  - Ontology based test case composer (under development) with an aim to reduce in test-case error, testing effort

**Performance Engineering Framework**

- A poorly performing software costs billions of dollars to the business
- Well engineered application for performance is much less costly than post-deployment refactoring

**Application Security Research**

- Gartner analysis: 75% of the malicious attacks on the Web are due to application level security holes
- Problems increase with more and more business applications are becoming web-based

**Project Overview**

A large (millions of lines of code) business application with long shelf life undergoes continuous modification, often in an ad-hoc manner and finally becomes extremely difficult to understand, manage and enhance

- To increase comprehensibility, analyzability (e.g. changeability) of a large software system, the source code can be grouped into modules.
- A module is a implementation unit of software that provides a coherent unit of functionality.
- Goal
  - To implement a tool to analyze the system from modularity perspective, and help users to reorganize the system into a set of smaller modules which are independently testable and maintainable.
  - To implement tool to assist developers to build modular code
  - To assist migration into a Service-Oriented Architecture
Large System Modernization Research

Current Approach

Research Trend Analysis

Notion of Modularity

- A module can be defined variously, but generally must be a component of a larger system, and operate relatively independently from the operations of the other components
- Should possess well-specified abstract interfaces
- Should provide orthogonal, high-cohesion
- Portability: Module independence

Benefits

- Extensibility
- Reusability

Decomposability: Are larger components decomposed into smaller components?

Composability: Are larger components composed from smaller components?

Understandability: Are components separately understandable?

Continuity: Do small changes to the specification affect a localized and limited number of components?

Protection: Are the effects of run-time abnormalities confined to a small number of related components?

Notion of Modularity - Baldwin and Clark

- Design Structure Matrix: To define a modular structure
- Generic, can be applied in various industries

Modularity operators are abstract and generic.

They can be used to assess the maintainability of a system in terms of how complex the system is to understand.

Program complexity metrics (Halstead, Cyclomatic, MI, Caper Jones) based on

Clasical coupling-cohesion metrics are based on function dependency (or some other structural dependency). Can they be used for partitioning a code to a set of modules?

- If a function calls a function B, then both A and B should be considered 'cohesive' & belong together in the same module. But using function call dependencies as the sole basis for modularization runs counter to the very spirit of what is meant by modules in modern code writing. Modules pull together functions not because they call one another, but because they serve similar purposes with respect to the rest of the software.

- In a typical client-server system, client functions call server functions several times but that does not mean that client-server functions should be combined together to form a module!

Modularity Metrics

- Low API usage
- High cohesion
- No complex interactions within modules
- Modular size uniformity
- No external module size greater than an acceptable standard deviation

Module Interaction Index

- Non-API function calls between modules
- Non-API function calls are related to each other

Non-API function calls may be hidden through similar names which confuse the function caller.

Non-API cohesion index

- Low cohesion means that the API functions should be tightly coupled
- Non/API functions should not be exposed to the external world and this number should be zero

Non-API size uniformity

- Ensure that no individual module size is greater than an acceptable standard deviation

Recall that modularity is the art of decomposing software into modules that are easy to test, maintain, and understand.

Software Design: A set of modules which are loosely coupled having well defined API functions but each module may be functionally very complex to understand.

Module size boundedness

- Ensures that an individual module size should not deviate from an acceptable module number, measured against the number of modules.
Layer Organization Index
- Checks whether layering principles have been honored
- Discourages skipping of layers when a module at a given layer calls another module in the layer below it. It also discourages calls to the upper layer with high penalty. In an ideal case, this index should be close to 1.

Normalized Cumulative Component Dependency Index
- A module with fewer numbers of dependencies is comparatively more testable than a module with large number of dependencies.
- A module needs to be tested whenever the same module or any of the modules it depends on is modified. Therefore in a system where there is cyclic dependency between modules, a change in one module can potentially require the entire system to be tested. This value in ideal condition should be < 1. A high number (>> 1) indicates that testing effort is high.

Concept Domination Index and Concept Coherency Index
- Modules that have ‘functionally similar’ entities (functions, files, data structures) are considered to be functionally cohesive.
- Measures the violations of such principles.
- Based on Information Theory: A module with functionally similar entities has less Entropy and more Information Content.
- Uses keyword based similarity detection.

Metric based on stability of modules in Layers

Experiments
- Metric values - human and random modularization

Work in Progress
- Automated Modularization
- Identification of Architectural Layers
- Identification of Domain Concepts

Modularization of Software
- Objective
  - Re-modularization of large legacy SW systems by re-grouping all entities (functions/files) into a set of modules without changing the content of functions.
  - Each entity should belong to exactly one module.
  - Achieve a modularization which will be considered as ‘good’.

- Goodness of Modularization can be determined by
  - Measuring the modularity metrics
  - Measuring how close the modularization is to the human
  - Using various distance measures

- Expert who developed or maintains the system
- Knows the system
- Provides drawbacks
- Subjective judgment
- Expertise in architecture

- Independent software architecture expert
- Needs time to understand
- Less biased

- The solution space for real-life system is huge
- With 1000 files and 20 modules, the number of solutions could be of the order of $2^{1990}$

Approximate Algorithm - Tabu Search (TS)
- Objective to be maximized
- Local optimum
- Global optimum

- Idea of Tabu Search
  - After an entity has been moved from module A to B, initial to move it back to A for a certain number of iterations
  - Maintain a set of non-dominated solutions while algorithm runs
  - Drop solutions which are dominated by others
  - Use non-dominated solutions as starting point for finding better solutions
  - Let user pick one of the non-dominated solutions after the algorithm has terminated

- Approximate algorithms

<table>
<thead>
<tr>
<th>Solution</th>
<th>Metric 1</th>
<th>Metric 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>2 (dominated by 3)</td>
<td>0.5</td>
<td>0.6</td>
</tr>
<tr>
<td>3</td>
<td>0.8</td>
<td>0.7</td>
</tr>
</tbody>
</table>
Experiment conducted

- The modularity metrics—specifically the concept-based metrics have been used as objective functions.
- Different mix of metrics have been tried as multi-objective functions.
- All the solutions are much better than random modularization (‘bad’).
- But the produced modularizations are not similar to the reference modularization with regard to the distance measures MoJo.
  - Reference modularization was created manually based on system documentation, human modularization.
  - Concept-based metrics showed good values for reference modularization.
  - When the solutions obtained by Tabu search are measured using Concept metrics they showed better result.
- The metrics that are good for measuring the deterioration of modularization may not be a good objective function for modularization.

Identification of Business Topics

- When a software system is small, one can understand its functional architecture by manually browsing the source code. For large systems, one employs structural information and analysis techniques such as:
  - Call graph.
  - Control and data flow, data slicing, chopping.
  - Model checking.
- These techniques help a little to comprehend the functional intent of the system.
- An important step to comprehend the functionality is to identify the embedded business topics around which the high-level components (or modules) have been implemented.
  - Customers and Loans in a Banking Application.
  - SSL Encryption in Apache Web Server.
  - Buffered Storage in PostgresSQL database.
  - but not in text editors.

Experiment

- We are running systems that consist of 10 Million Lines of Code or even more.
- Too many modules exist to understand and manage the whole system. Therefore, an additional organization of the modules is needed.

Architectural Layering in a large software system- Problem Context

- Layers completely partition a set of software, and each partition constitutes a virtual machine - with a public interface - that provides a cohesive set of services. (+ strict ordering)
- Collection of modules constitutes a layer and layers are typically stacked.

Layered architecture

- Bass et al. "Documenting Software Architectures":
  - Layers help to bring quality attributes of modifiability and portability to a software system.
    - In theory, if something is changed within a lower layer it can be hidden. It helps to manage complexity and to communicate the structure, because of its simplicity.
    - Testability is increased, especially for the top and bottom layer, because only one mock layer can be used to test these two layers as a test driver.

How to identify layers in a system?

Identification of Top and Bottom modules

- For every module we can compute the Fan-In / Fan-Out ratio.
- Assumption:
  - Sink candidates have a high Fan-In and a low Fan-Out and therefore belong to the bottom layer.
  - Driver candidates have a low Fan-In and a high Fan-Out and therefore belong to the top layer.

Attribution-based Grouping of modules:

1. Initial assignment of some modules to the layers (the driver and sink candidates).
2. For each remaining module calculate the attribution of each module to one of the layers.
3. If a module is highly attracted to a certain layer, then assign it to this layer.
4. If a module is similar attracted to multiple layers, let the user decide.

Identifying Semantic Information

- Semantic information is found in the names of identifiers in the Source Code. Often it leaves hints at what the code is doing in a human-readable form. For instance, "proxy", "http" while implementing an http-proxy. Similarly Address, Street, Zip are related to Address.
- Such meaningful keywords can be found in:
  - Functions and their parameters.
  - Variables declarations and use.
  - Files.
  - Classes and types.
  - Comments.
- Assuming that the meaningful keywords do exist in program elements, is it possible to correlate these keywords to meaningful clusters?
- Applying Natural language processing techniques
  - Use of computers to analyze and index text written by humans.
  - Applications:
    - Search engines.
    - Identifying topics in scientific papers (Griffiths and Steyvers, 2004).
Latest Semantic Analysis

- It computes combinations of words having similar meanings. This reduces noise and deals with synonymy (two keywords have the same meaning, even approximately so)

  **Algorithm**
  - The number of times keyword k appears in source code document d is placed in matrix X[k,d].
  - Singular Value Decomposition computes a lower dimensional approximation Y of X.
  - The keyword-keyword matrix YTY contains similarity between keywords. Similarly, the document-document matrix YTY contains similarity between documents.
  - A similarity group between the keywords (YTY) can be treated as a topic.

- Does not deal to polysemy (keyword has more than one meaning)
  - Hard partitioning of keywords
  - Do we know how the topics are distributed across files? - No

Latent Dirichlet Analysis

- A document/file is a mixture of topics.
- Topics are a mixture of keywords.
- A software system S is a corpus of M source code files, S = {f_1, ···, f_M}, with N unique terms W = {w_1, ···, w_N}.
- Each file f_d, d = 1 ··· M has a multinomial distribution θ_d over T topics, and each topic θ_j, j = 1 ··· T has a multinomial distribution Φ_j over W.
- The topic extraction aims to find out a set of suitable Φ_j using a Dirichlet distribution.

Summary

- Software Engineering challenges
  - Scale
  - Geographical Distribution
  - Low cost
  - Fast development using COTS products

- Software Engineering Research is extremely crucial for the organization

- Research focus:
  - Automation
  - Collaboration
  - Assembly

- Large Software System Analysis
  - Metrics based evaluation
    - Various projects are executed on
      - Modularization
      - Analysis
      - Concept Extraction

Thank you