Software Engineering Challenges- IT Services Industry Perspective

Santonu Sarkar
Software Engineering Technology Labs (SETLabs)

Infosys Global Presence

Infosys Corporate Presence

• 42 offices, 10 GDC, 7 India offices

Americas	APAC	Europe
USA Canada	Australia China Hong Kong India Japan Mauritius UAE	Belgium Denmark Finland France Germany Italy Norway Spain Sweden Switzerland The Netherlands United Kingdom

Infosys Internship Program

Interns worked in the current project so far

1. Tsinghua University, Beijing China
2. University of Mannheim, Darmstadt University- Germany
3. University of New South Wales- Australia
4. University of Urbana Champagne, IL
5. California Institute of Technology *
6. University of Southern California *
7. Cornell University
8. ETH Zurich

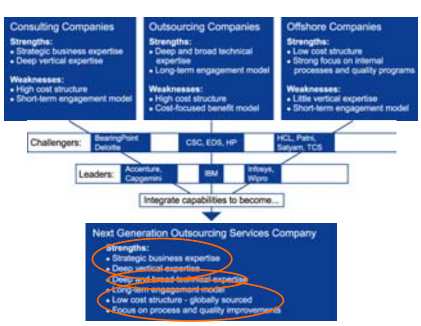
Awards and Accolades

- 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 CMM 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

Agenda

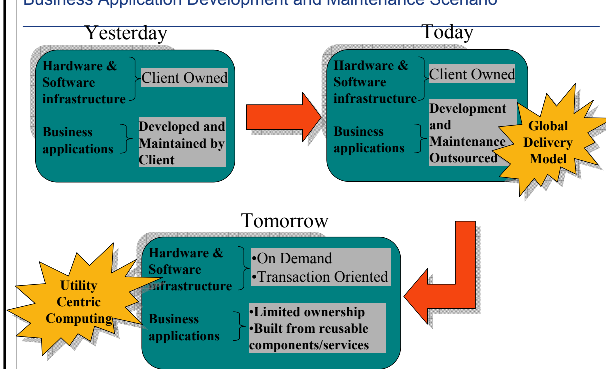
- ❑ Software Engineering Challenges
- ❑ Software Engineering Research Agenda at SETLabs
- ❑ Research Projects- Overview
- ❑ A Research Project- Detailed

Next Generation Outsourcing of Software Services

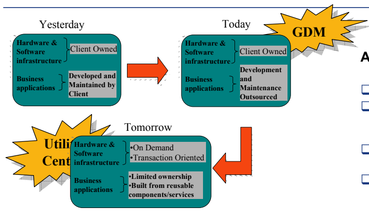


Source: AMR

Business Application Development and Maintenance Scenario



Business Application Development and Maintenance- Complexities



Complexities

Application Development

- Large in size, distributed
- Never built from scratch: uses existing components, commercial products
- Needs to integrate with the environment
- Rapid development as requirements evolve

Application Maintenance

- Large in size
- Poor or no documentation, test cases
- Unavailability of original authors
- Gross deviation of the code from the original architectural intent
- Labor intensive maintenance

Additional Complexities

- Need to evaluate whether a business application can be transformed to a utility based model
- How can the business value of these applications be delivered to end users through mobile devices
- Culture and change management

Distributed Software Development and Maintenance- Today

Distributed locations

- Delivering projects with less cost

Shortage of talent

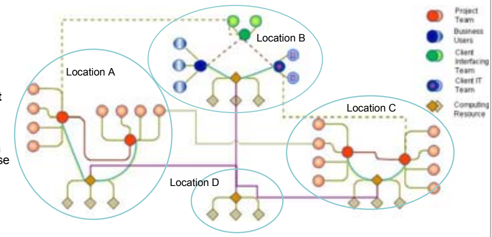
- Delivering projects with less staff
- Delivering with limited expertise

Margin pressures

- Delivering by spending less effort

Successive team handovers

- Retaining knowledge across team transitions



- How can the engineering productivity be improved- accuracy, faster turn around time?
- How can the collaboration among people, computing resources can be made structured, precise and effective?
- How can the development/maintenance be made assembly line and less dependent on human expert?
- How can one ensure various quality traits like performance, security, modularity..?
- How can one improve application re-engineering – program comprehension, modularization, service-orientation..?

Agenda

- Software Engineering Challenges
- Software Engineering Research Agenda at SETLabs
- Research Projects- Overview
- A Research Project- Detailed

Improving Engineering Productivity- Research Agenda

Automation

- Business Process Model Centric Requirement Analysis
- Model Driven Design
- Software Quality Assurance
- Large System Development, Comprehension, Modernization

Collaboration

- Collaboration Platform for community based requirement analysis
- Virtualization

Assembly

- Service Oriented Architecture
- Software As a Service

Improving Engineering Productivity- Research Agenda

- Business Process Model Centric Requirement Analysis
  - Tool for Requirement Analysis
  - Creation of Reusable Artifacts
- Model Driven Design
  - Use of higher order languages - Architecture Modeling
  - Model Driven Testing
- Software Quality Assurance
  - Capacity Analysis and Performance Assessment
  - Application Security
- Large System Development, Comprehension, Modernization
  - Metric driven approach to measure the work-product quality
  - Fast Comprehension
- Collaboration Platform for community based requirement analysis
- Virtualization
  - Environments, test-beds that can be accessed uniformly everywhere and enable location-independence of many activities
- Service Oriented Architecture
  - Service Semantics, Orchestration, Differential QoS
- Software As a Service
  - Prototypes and reusable framework

Other Research Areas at SETLabs (Illustrative)

Dynamic Processes	Malleable Architecture	Personalized Information	Pervasive Infrastructure	Transformational IT Management	Platforms
Business Process Management Enterprise Collaboration Knowledge Engineering/ Ontology	Web Services SOA Software Architecture Grid Computing	Enterprise Content Management Business Intelligence Data warehousing	Mobile Computing Sensor Networks RFID	Outsourcing IT Strategy Solution Methodologies Regulatory compliance	J2EE .Net

Agenda

- ❑ Software Engineering Challenges
- ❑ Software Engineering Research Agenda at SETLabs
- ❑ Research Projects- Overview
- ❑ A Research Project- Detailed

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 errors, testing effort

❑ **Process Execution and Analysis Platform**
- A customizable solution for process modeling (executable level), process orchestration and workflow management

❑ **Capacity Assessment and Performance Engg**
- Comprehensive framework for engineering and managing performance and capacity with in-house tools developed for comprehensive performance management

❑ **Large System Development and Management**

❑ **Application Security**
- Secure application development process
- Authentication solution using mobile devices
- AOP based application vulnerability detection

Performance Engineering Framework

Requirement Elicitation
- NFR Capture
- NFR Validation

Architecture & Design
- Architecture Scalability Assessment
- Infrastructure Sizing
- Performance Test Strategy Definition

Build
- Preliminary Load Testing

Systems Integration Testing
- Application Scalability Assessment
- Load Test
- Stress & Volume Test
- Endurance Test
- Performance Prediction
- Capacity Planning

Deployment

Post Deployment (Maintenance)
- Workload Forecasting
- Capacity Upgradation Roadmap

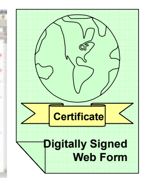
- ❑ A poorly performing software costs billions of dollars to the business
- ❑ Well engineered application for performance is much less costly than post-deployment refactoring

Application Security Research

Process – Secure SDLC
- Security Requirement Capture
- Security Test Cases
- Threat Model and Counter measures
- Testing
- Security Prescriptions- J2EE/.NET

Authentication

Secure Web Interface



Application Vulnerability Detection
- Framework based on Aspect Oriented Programming principles to thwart web application security vulnerabilities

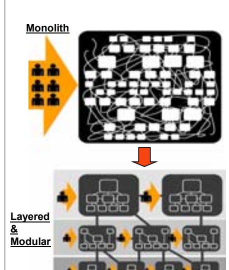
- ❑ Gartner analysis: 75% of the malicious attacks on the Web are due to application level security holes
- ❑ Problems increase with more and more business applications are becoming web-based

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

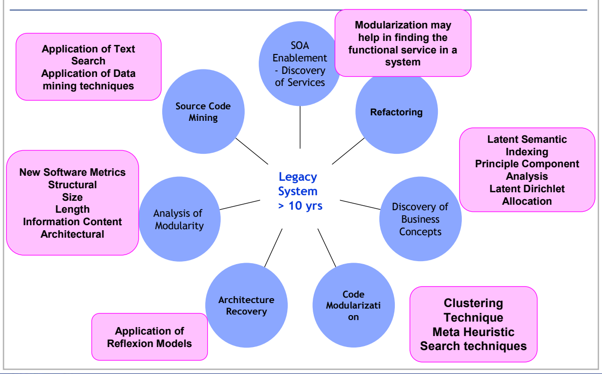
Project Overview

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 domain modules which are independently testable and releasable
  - ❍ To implement tool to assist developers to build modular code
  - ❍ To assist migration into a Service Oriented Architecture

Large System Modernization Research



Research Trend Analysis

- Program complexity metrics (Halstead, Cyclomatic, MI, Caper Jones) based on are not suitable for modularity analysis
  - They can be used to assess the maintainability of a system in terms of how complex the system is to understand
  - They don't indicate how modular a system is (Parnas 1972). For instance, a system may consist of modules which are loosely coupled having well defined API functions but each module may be functionally very complex to understand.
- Classical 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 A 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 calls server functions several times but that does not mean that client-server functions should be combined together to form a module!

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 have high cohesion and low coupling (Meyer)
- Benefits
  - Extensibility
    - well-defined, abstract interfaces
  - Reusability
    - low-coupling, high-cohesion
  - Portability
    - hide machine dependencies

How modular is a system? (Meyer)

- **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?

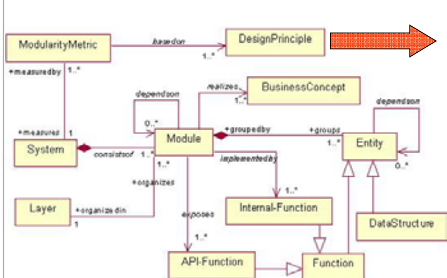
Notion of Modularity- Baldwin and Clark

- 'Design Structure Matrix' To define a modular structure
- Generic, can be applied in various industries
- Modularity operators and net option value for each modularity operators
- 'Design Structure Matrix' can be applied to Software Design:
  - Sullivan et al (2001) models dependencies among modules, module interfaces and key data structures
  - Sangal et al (2005) models hierarchical subsystem organization using DSM
- Modularity operators are abstract and generic. It is a non-trivial task to define precise semantics in the context of software design
- Several matrix operations can be applied on a design structure matrix to compute various system dependencies
- Sangal et al have used DSM structure to represent layered architecture

	1	2	3	4	5
Entity-1	X				X
Entity-2			X		
Entity-3	X				
Entity-4			X	X	
Entity 5	X				X

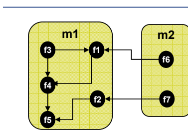
	1	11	12	2	3	31	32
Module-1							
submod11			X	X			
submod12		X					X
Module-2						X	
Module-3							
submod31							X
submod32						X	

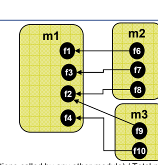
Current Approach



1. A module offers a well-defined coherent set of services
2. Interaction through well-defined API
3. To a large extent each module should be independently compliable
4. Modules should be easily extendible and testable with minimal effect on others
5. Acyclic dependency among modules and layered organization of modules
6. Uniformity in module size and observance of module size bounds

Modularity Metrics [Sarkar et al API-Based and Information-Theoretic Metrics for Measuring the Quality of Software Modularization– IEEE TSE 2007]





Module Interaction Index
- Inter-module interactions should ideally happen only through a set of designated API functions. For a well modular system, all external calls are routed through API functions and the metric will be 1.
- In reality external calls happen through non API functions due to bad programming practice, thus this metric will not be 1 in real life systems.

Non-API closeness Index
- Ideally the non-API internal functions ideally should never be exposed to the external world and this number should be close to 1

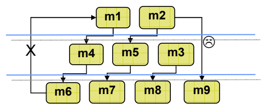
Explicit Dependency Index
- Ideally the inter-module interactions should happen through functions, not through indirect means such as global variables. This index measures the violation of this principle.

Max API Usage
- Max(#API functions called by any other module) / Total number of API functions, 0 otherwise
- Average over all modules for which the above is non-zero
- Should be close to 1
- All most all of the published API functions in a module would be used by some other module, thereby making the fraction close to 1Non-API closeness Index

Module size uniformity
- Enforces equal sized modules within an acceptable standard deviation

Module size boundedness
- Ensures that an individual module size should not deviate too much from some acceptable module numbers, discourages monolithic modules

Modularity Metrics…..



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.

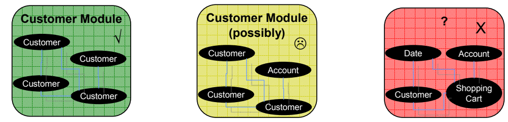
Normalized Cumulative Component Dependency Index[1]

- 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.

Metric based on stability of modules in Layers

1. *John Lakos, "Large Scale C++ Software Design". Addison-Wesley, 1996*
2. *Les Hatton, "Reexamining the Fault Density-Component Size Connection", IEEE Software 1997*

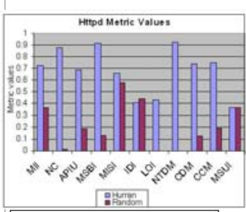
Modularity Metrics…..



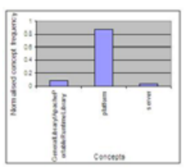
Concept Domination Index and Concept Coherency Index

- Modules that has '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

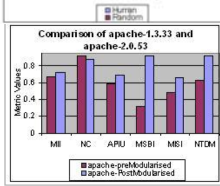
Experiments



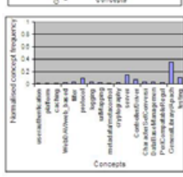
Metric values-human and random modularization



Concept Distribition-Module beos



Metric Value Comparison



Concept Distribution-Random Modularization

Work in Progress

- Automated Modularization
- Identification of Architectural Layers
- Identification of Domain Concepts

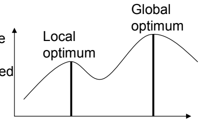
Modularization of Software

- Objective
  - Re-modularization of large legacy SW systems- Regroup all entities (functions/files) into a set of modules without changing the content of functions/files
    - 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
  - Possible drawbacks
    - Self-interests
    - Used to existing structure
    - Subjective judgment
    - Blindness
- 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 20^{1000}
- **Approximate algorithms**

Approximate Algorithm-Tabu Seach (TS)



Solution	Metric 1	Metric 2
1	0.8	0.4
2 (dominated by 3)	0.5	0.6
3	0.6	0.7

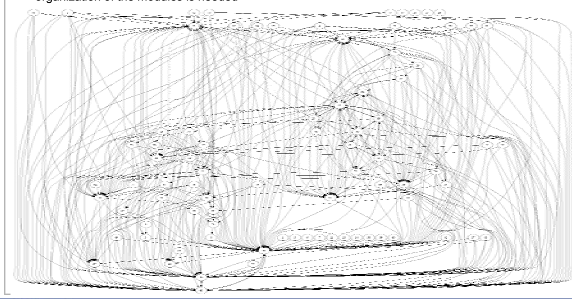
- Idea of Tabu Search
  - After an entity has been moved from module A to B, **forbid to move it back** to A for a certain number of iterations
  - Maintain a set of non-dominated solutions while algorithm runs
  - Drop solutions which are dominated by others
  - Use non-dominated solutions as starting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

Experiment conducted

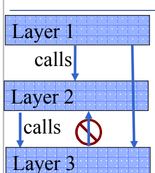
- The modularity metrics – specifically the concept based metrics have been used as objective functions
- Different mix of metrics have been tried as multi-objective functions
- All the solutions are much better than random modularization ('bad')
- But the produced modularizations are not similar to 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

Architectural Layering in a large software system- Problem Context

- We are running systems that consists 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



Layered architecture

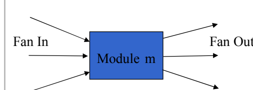


Bass et al. "Documenting Software Architectures":

- 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.
- 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?

Experiment



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**

Attraction 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 attraction of each module to one of the layers
3. If a module is highly attracted to a certain layer then assign this module to this layer
4. If a module is similar attracted to multiple layers, let the user decide.

Andreas Christl et al "Equipping the Reflexion Method with Automated Clustering," *WCRE 2005*, pp. 89-98

Attraction Value = f(#calls within a layer, #calls to a lower layer, #back calls to higher layers)

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 PostgreSQL database
    - but not in text editors

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
  - Variable 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)

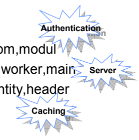
Latest Semantic Analysis

- 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 YY^T contains similarity between keywords. Similarly, the document-document matrix Y^TY contains similarity between documents.
  - A similarity group between the keywords (YY^T) 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

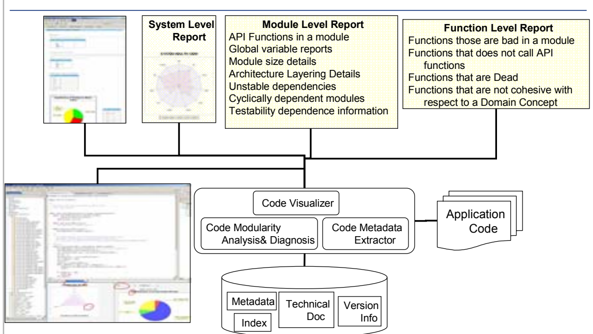
- A document/file is a mixture of topics.
- Topics are a mixture of keywords.

- A software system S is a corpus of M source code files, S = {$\mathbf{f}_1, \cdots, \mathbf{f}_M$}, with N unique terms W = {$w_1, \cdots, w_N$}
- Each file $\mathbf{f}_d$, d = 1 $\cdots$ M has a multinomial distribution θ^d over T topics, and each topic t_j , j = 1 $\cdots$ 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.



- Topic 1 = auth,config,mod,dbm,modul
- Topic 2 = thread,child,signal,worker,main
- Topic 3 = cach,size,mmap,entity,header

Tool Architecture



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