

Overview of Software Architecture and Software Estimation

Raffi Tikidjian

Thomas Tan

Joshua Garcia

Vu Nguyen

Daniel Popescu



Outline

- ❑ Overview of Software Engineering
- ❑ Software Architecture
- ❑ Software Estimation



What is Software Engineering

- ❑ Software Engineering is concerned with theories, methods and tools for professional software production
- ❑ Goals:
 - Produce high quality software product
 - On time
 - In budget
 - Satisfy needs people involved (customers, owners, users, developers, etc)
 - Enable highest Return of Investment (ROI)

Why Software Engineering

- ❑ Software is everywhere nowadays
- ❑ Software production is difficult
 - Software construction is human-intensive
 - Software is malleable
 - Software is intangible
 - Software size and complexity are increasing, for example, Windows Vista:
 - has over 50 mil lines of code
 - 2000 software developers
 - 5 years of development



Software Architecture



- ❑ Architecture is a set of principal design decisions about a software system
- ❑ Three fundamental understandings of software architecture
 1. Every application has an architecture
 2. Every application has at least one architect
 3. Architecture is not a phase of development



Faithful Implementation

- ❑ All of the structural elements found in the architecture are implemented in the source code
- ❑ Source code must not utilize major new computational elements that have no corresponding elements in the architecture
- ❑ Source code must not contain new connections between architectural elements that are not found in the architecture

Architecture Compliance Checking

- ❑ Architecture compliance is the degree to which the implemented architecture conforms to the planned architecture
 - Late architecture evaluation (i.e., after the implementation)
- ❑ Dynamic compliance checking
 - With run-time traces
 - Addressing behavioral architectural views
- ❑ Static compliance checking
 - Without executing the source code
 - Addressing structural architectural views

Static Architecture Compliance Checking Approaches

(1)

```
package hello;

import world.World;

public class Hello {
    public void caller() {
        World world = new World();
        world.variable = "Hello World.";
        world.methodCall();
    }
}

package world;

public class World {
    public String variable;

    public void methodCall() {
        System.out.println(variable);
    }
}
```



❑ Extraction of relevant facts from source code

- Structural elements (e.g., in Java: packages, classes, methods)
- Dependencies among the structural elements (e.g., in C: includes, function calls, variable accesses)

❑ Aggregation of facts about a system in a repository/fact base and abstraction to components

Static Architecture Compliance Checking Approaches (2)

□ Application of the static compliance checking approach

- Convergence
 - A relation between two components that is allowed or was implemented as intended
 - implementation is compliant to the architecture
- Divergence
 - A relation between two components that is not allowed or was not implemented as intended
 - implementation violates architecture
- Absence
 - A relation between two components that was intended but not implemented
 - no implementation found

Architecture Compliance Checking Approaches



- ❑ Three static architecture compliance checking approaches have been identified
 - Reflexion models
 - Relation conformance rules
 - Component access rules

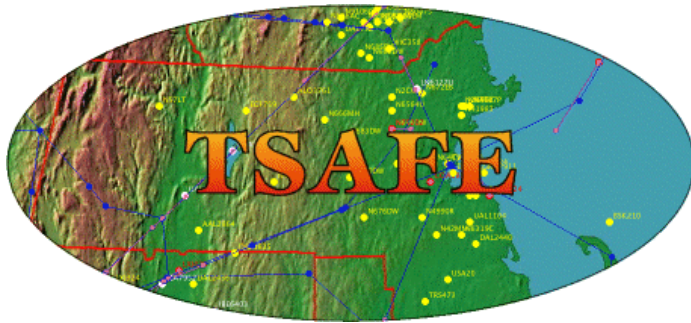
- ❑ These approaches are a sound quality assurance instrument in architecture development
 - Benefits of each approach have been shown in several case studies
 - But
 - What are the commonalities, differences, advantages and drawbacks of each approach?
 - When choose approach A instead of approach

Case Study - TSAFE



- ❑ TSAFE is a prototype of the Tactical Separation Assisted Flight Environment as specified by NASA Ames Research Center

- TSAFE checks aircraft flights, predicts future trajectories, and displays results on a map
- TSAFE was turned into a testbed by Fraunhofer Center Maryland (FC-MD) to be used for software technology experimentation



- ❑ We statically evaluated two variants (original prototype and a redesigned version) of TSAFE to illustrate the three approaches

- Architectural models, architectural rules, source code and documentation were available
- Architects and developers could be interviewed to obtain further information (if necessary)

Reflexion Models



- ❑ Architectural components are mapped onto source code elements provided in a low-level source code model by human-based mapping
- ❑ These mapping are based on regular expressions and a high-level component typically comprises a high number of source code elements
- ❑ The architectural model contains the planned or intended relations among components
- ❑ At evaluation time, the mappings are resolved
- ❑ Relations among architectural components are marked as convergences, divergences or absences
- ❑ The original idea was introduced by [Murphy, Notkin, and Sullivan, 2001]

Principle of Reflexion Models

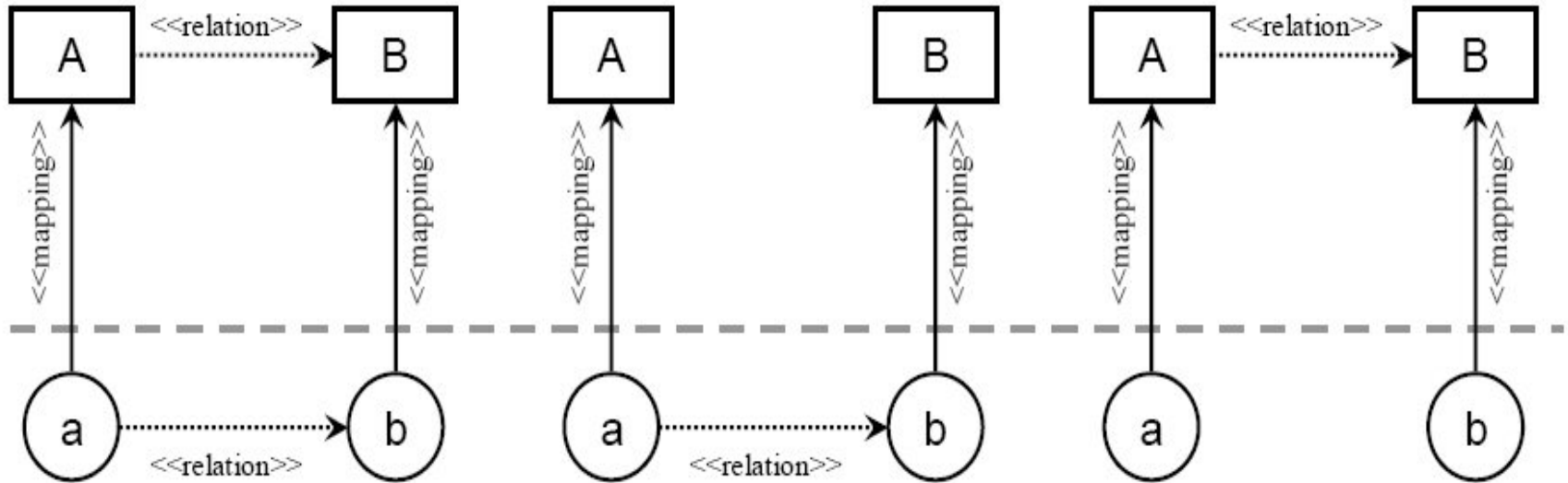


High Level Model

convergence

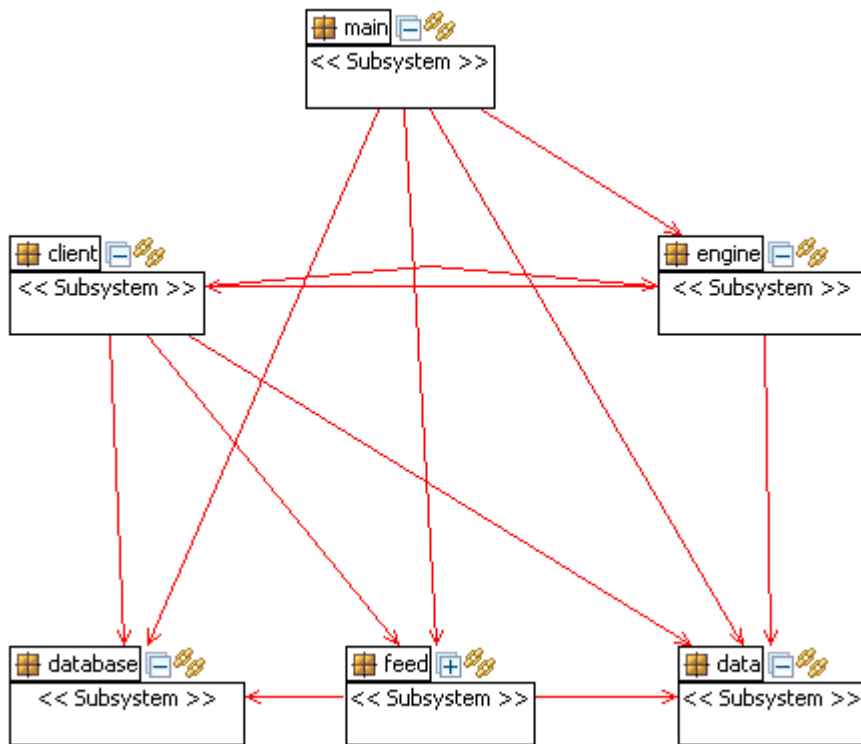
divergence

absence

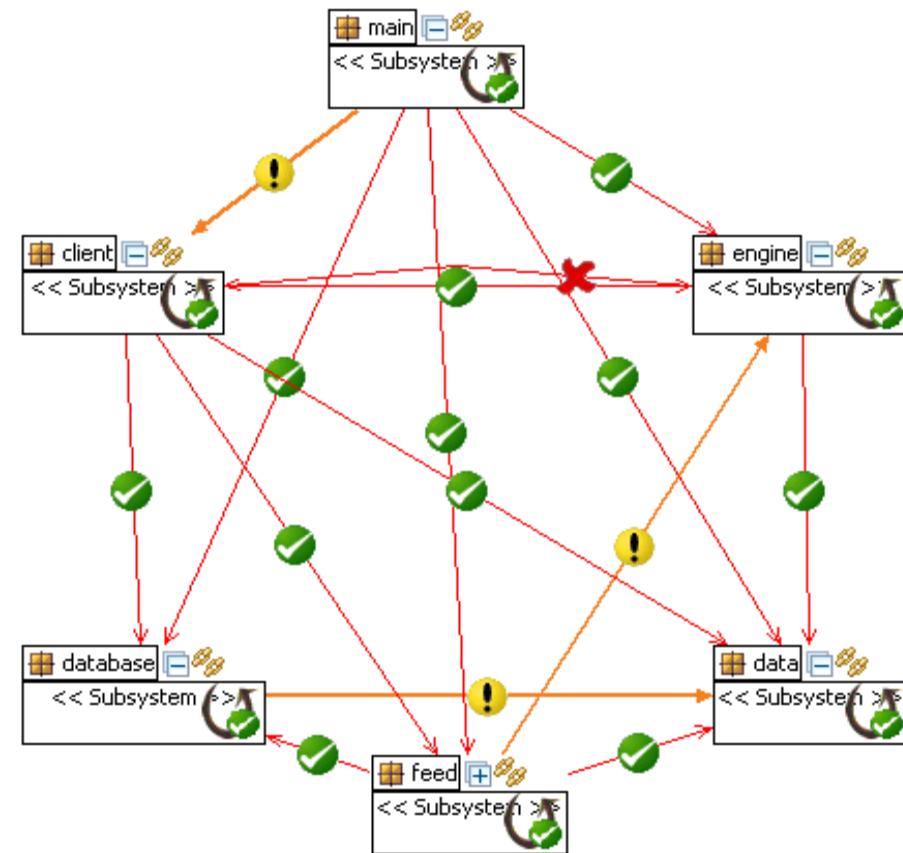


Reflexion Model Example

- ❑ TSAFE I – intended subsystem dependencies



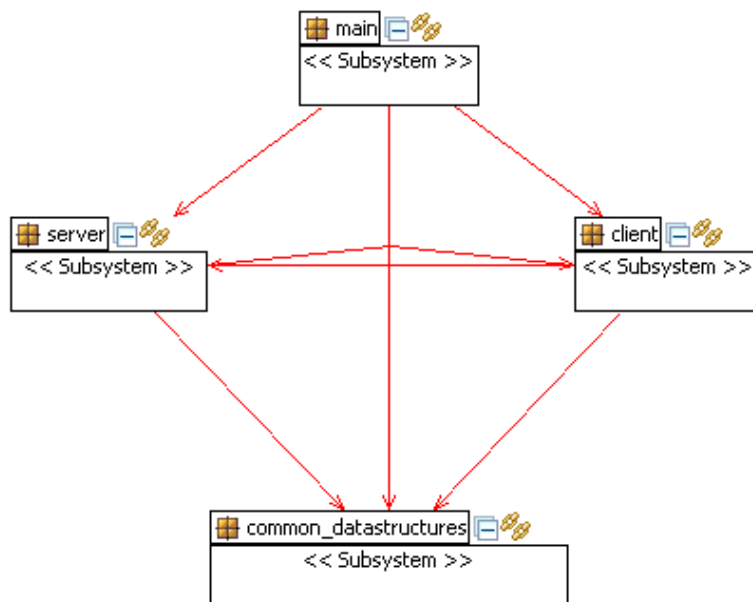
- ❑ TSAFE I – compliance checking results



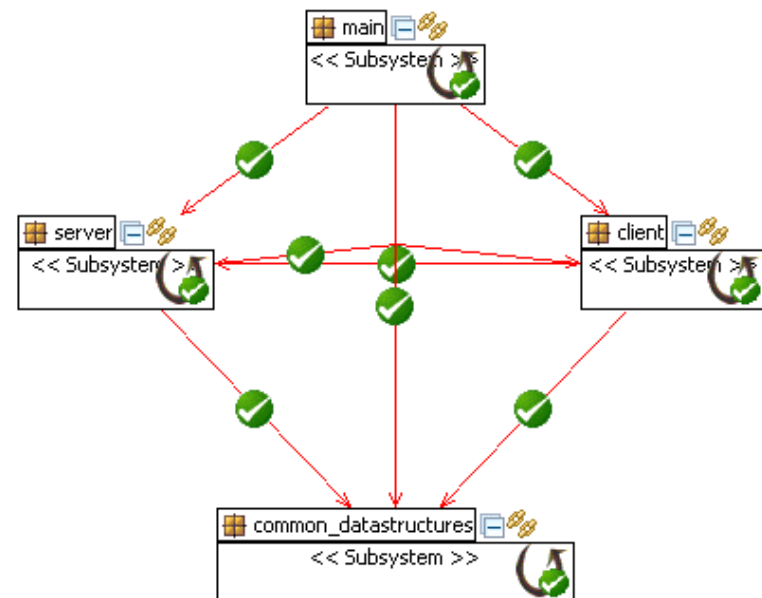
Reflexion Models – Compliance Checking Results

TSAFE II

- ❑ TSAFE II – intended subsystem dependencies



- ❑ TSAFE II – compliance checking results



Relation Conformance Rules

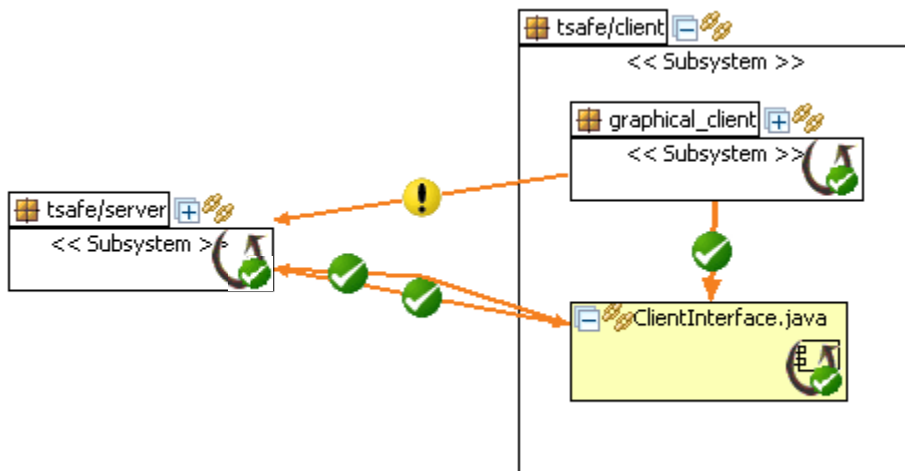
- ❑ A relation conformance rule defines an allowed or a forbidden relation between two components

- ❑ A relation conformance rule contains a source component and a target component defined by regular expressions
 - The relation rule "A* must exist B*" defines that a relation must exist from components, which name starts with an A, to components having a name starting with B

- ❑ Relation conformance rules allow that leaf level components can be checked (ignoring their super components)

- ❑ One relation conformance rule can cover multiple relations of different, multiple components

Relation Conformance Rules – Compliance Checking Results TSAFE II



□ TSAFE II – Conformance Rules

- No classes but the *ServerInterface* class shall access classes in the *client* subsystem
- No classes other than the *ClientInterface* class shall access classes from the *server* subsystem

Component Access Rules

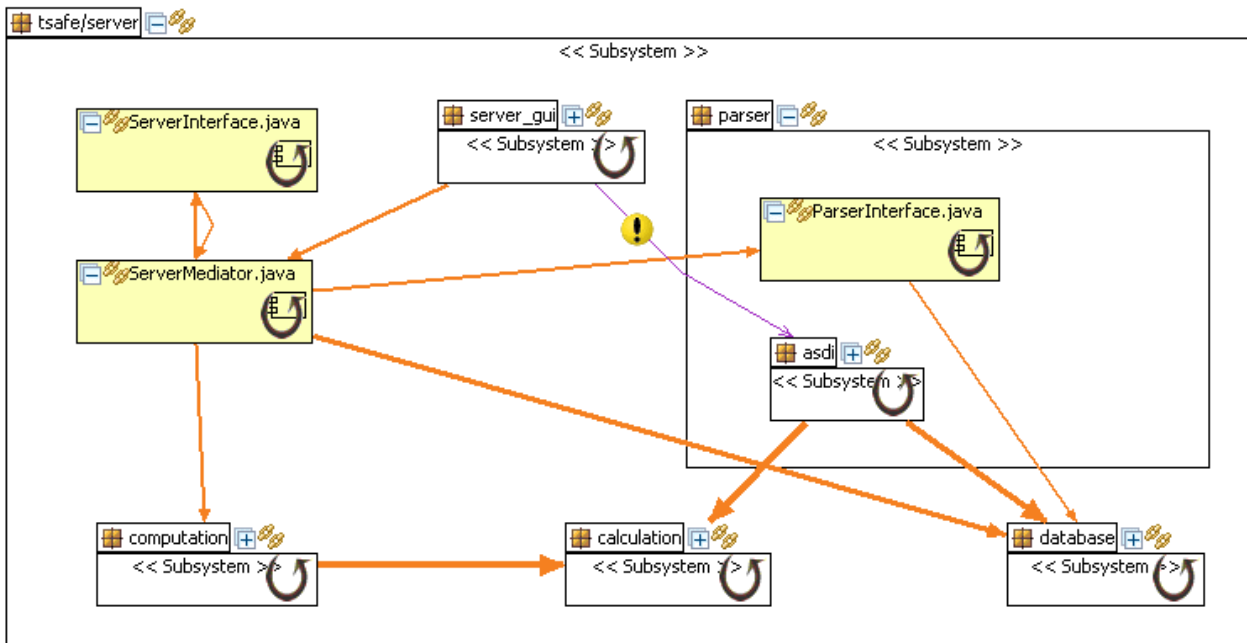


- ❑ Component access rules define simple ports of a component, which other components are allowed to use
- ❑ They help increase information hiding of components on a level, which might not be supported by the implementation language itself
 - In Java, public methods within a component are accessible from all other components in the system
- ❑ Component access rules encapsulate components and open only certain ports for component interaction
- ❑ Component access rule concerns only one component.

Component Access Rules – Compliance Checking Results TSAFE II

❑ TSAFE II – Access Rule

- Only the *ParserInterface* class shall be accessed from any other component



Static Architecture Compliance Checking



- ❑ Static architecture compliance checking are a sound instrument in architectural development
- ❑ Demonstration of three approaches derived from practical experiences in industrial and academic case studies
 - Reflexion models
 - Relation conformance rules
 - Component access rules



An Overview of Software Estimation



Center for Systems and Software Engineering

What is Software Estimation

- ❑ Software Estimation is a process that produces effort, time, cost, etc. for developing software systems BEFORE the work has been done
- ❑ Estimation is a subjective process
- ❑ The most subjective process is GUESS-TIMATE

Why Estimation is Important?

- ❑ Knowing things beforehand is always important
- ❑ Business development:
 - Investment decisions
 - Allocate resources (e.g. money, time, people) for investments
 - Project bidding dilemma
 - Overestimate → failure to get contracts → work lost → financial lost
 - Underestimate → failure to complete projects → financial lost
- ❑ Project management:
 - Project planning: effort, schedule, staff
 - Client negotiations
 - Trade-offs: change requests, risk management decisions, etc.



and Local Calibration



COCOMO II Overview

- ❑ COnstructive COst MOdel (COCOMO) was invented in 1981 by Dr. Barry Boehm at USC
- ❑ COCOMO II was developed in mid-1990s to adapt to new changes in current and future practices
- ❑ COCOMO II model is one of the most widely-used cost models in the industry, especially by US DoD contractors (Boeing, Northrop Grumman, Aerospace, etc.)
- ❑ COCOMO is continuously supported by Dr. Boehm's teams at Center for Systems and Software Engineering:
 - Annual international conference in COCOMO
 - Emerging extensions are published regularly



COCOMO II Effort Formula

$$PM = A \times Size^E \times \prod_{i=1}^N EM_i \quad (\text{Eq. 1})$$

Where $E = B + 0.01 \times \sum_{j=1}^5 SF_j$

PM – Effort in person-month

Size – Size of projects or modules

A – Constant, currently $A = 2.94$

B – Constant, currently $B = .91$

SF_j – Five scale factors, determined by estimators

EM_i – Effort multipliers, determined by estimators

COCOMO II Schedule Formula

$$\mathbf{TDEV = C \times PM^F \times SCED\%} \quad (\text{Eq. 2})$$

$$\begin{aligned} \text{Where } F &= D + 0.2 \times 0.01 \times \sum_{j=1}^5 SF_j \\ &= D + 0.2 \times (E - B) \end{aligned}$$

TDEV – Time to develop, calendar month

C – Constant, currently $C = 3.67$

D – Constant, currently $D = .28$

SCED% – Percentage of schedule compression

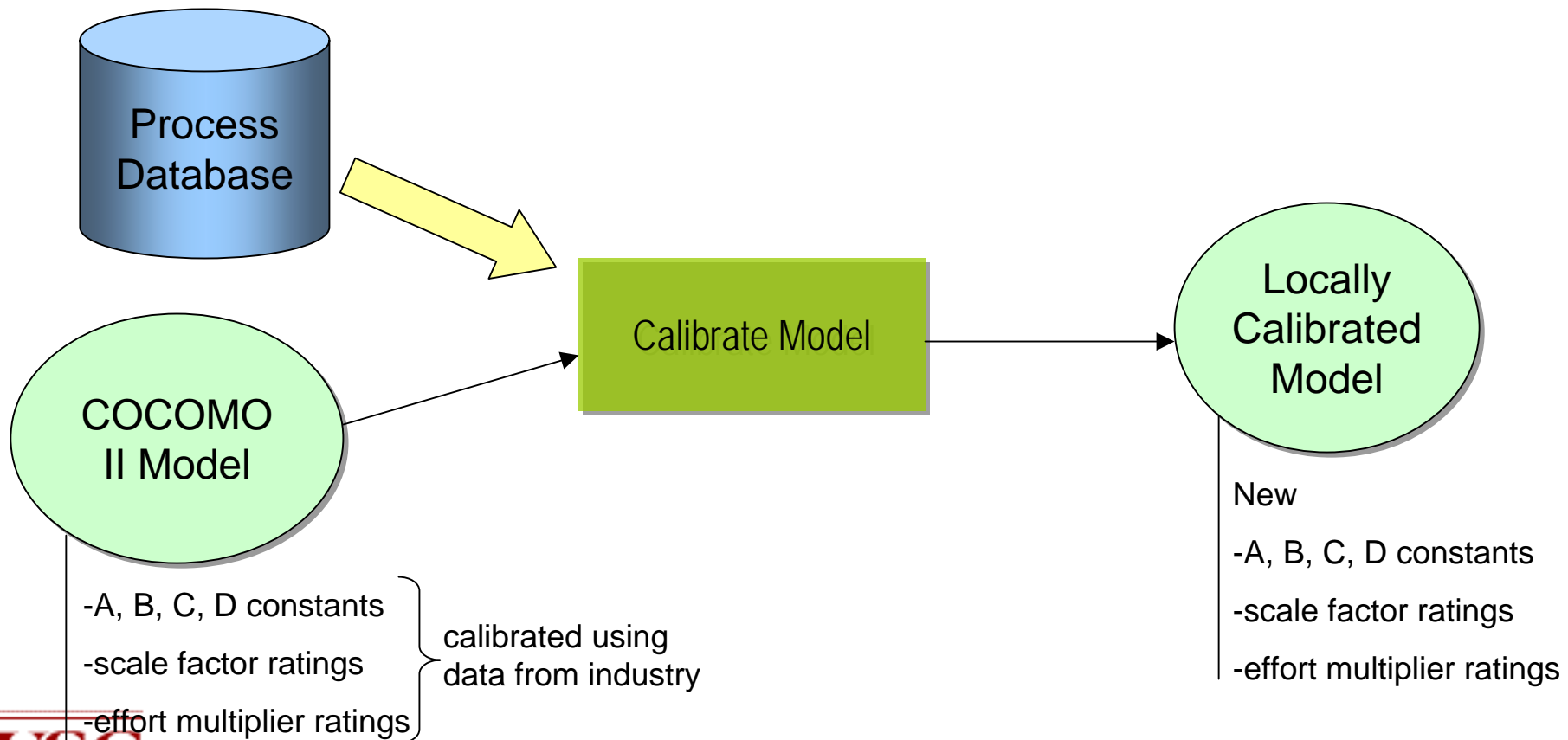
**Note: C and D can be calibrated*

COCOMO II Cost Drivers

- ❑ Cost drivers are variables considered to be source of cost variations
- ❑ Classified into two categories:
 - Scale factors: 5
 - Effort multipliers: 17 for Post-Architecture Model, 7 for Early-Design Model
- ❑ Each cost driver is rated into six levels (some cost drivers have 4 or 5)
 - Very Low (VLO)
 - Low (LO)
 - Nominal (NOM)
 - High (HI)
 - Very High (VHI)
 - Extra High (XHI)

COCOMO II Local Calibration

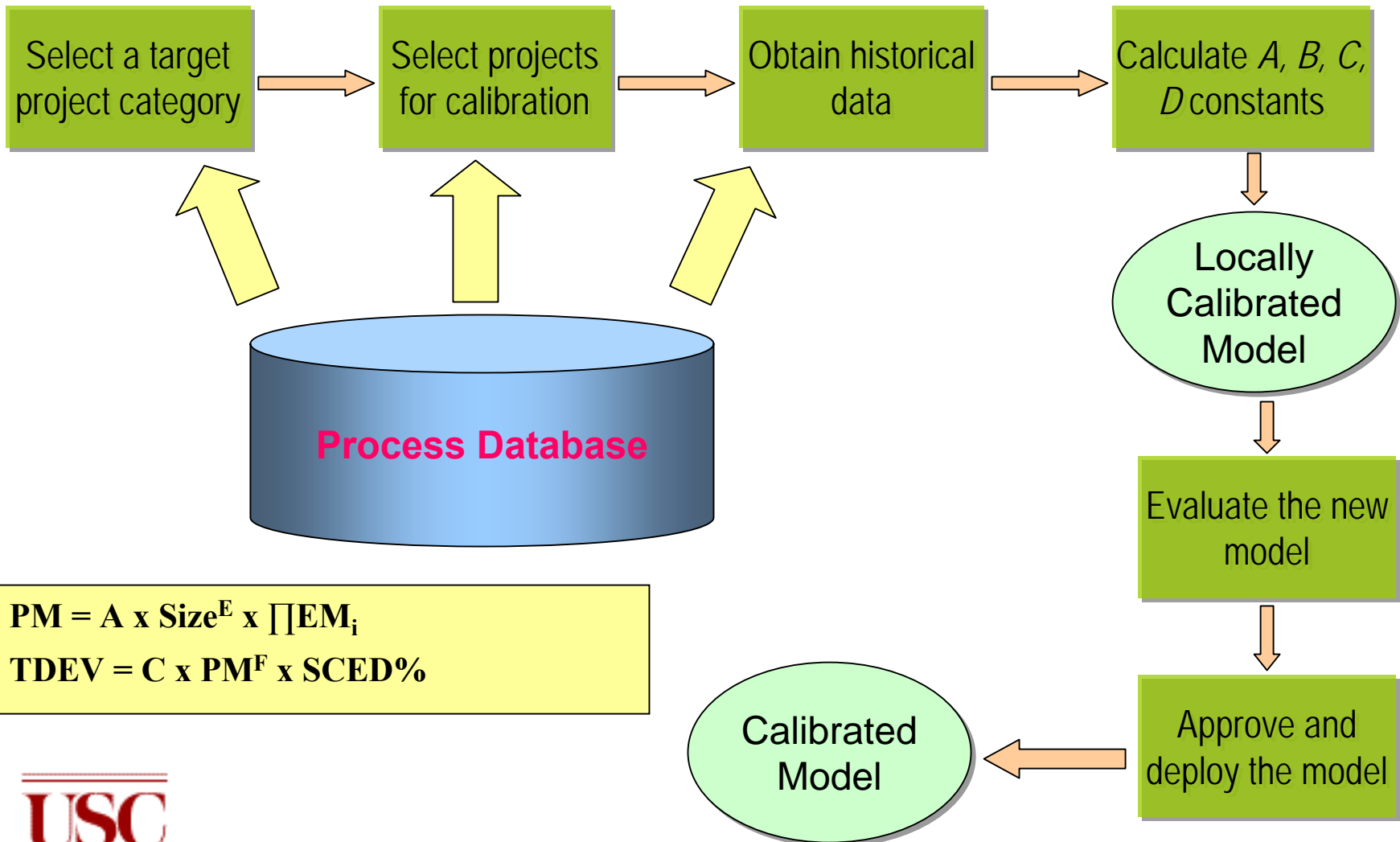
- Local Calibration is a process of building new cost model based on a basic cost model AND local data points



Why Local Calibration?

- ❑ COCOMO II Model Version 2000 was calibrated on 161 data points from industry, mainly from defense contractors
- ❑ Different organizations have different capability and environment
- ❑ Different projects have different characteristics
- ❑ SLOC is used as a size measure. Other sizing units have to be converted to SLOC

Calibration Process



$$PM = A \times \text{Size}^E \times \prod EM_i$$
$$TDEV = C \times PM^F \times \text{SCED}\%$$

Evaluate Calibrated Model



- ❑ There are two criteria for evaluating cost models
 - Accuracy
 - Consistency
- ❑ Model Accuracy is measured by the average Magnitude of Relative Error (MRE) and Prediction Level (PRED)

$$\text{MRE} = \frac{|\text{EstimatedEffort} - \text{ActualEffort}|}{\text{ActualEffort}}$$

$$\text{PRED}(p) = k / n$$

Where,

p is error level

k is the number of projects in a set of n projects whose $\text{MRE} \leq p$

Evaluate Calibrated Model (cont)

□ Model Consistency

- Evaluated using *correlation coefficient*, R^2

$$R = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{(n - 1)s_x s_y}$$

x_i – estimates

y_i – actuals

S_x – standard deviation of estimates

S_y – standard deviation of actuals

$0.9 \leq R^2 \leq 1.0$	Strong correlation. The model is consistent.
$0.7 \leq R^2 < 0.9$	High correlation. The model is consistent but needs to be reviewed and/or refined.
$0.0 \leq R^2 < 0.7$	The model should be reviewed and refined. It is not recommended for using this model.

A Case Study

- ❑ The local calibration was used to derive a local estimation model at FCG (First Consulting Group inc.)
- ❑ 8 completed projects were selected for the calibration
- ❑ Results:
 - **Effort Model:**
 - MRE = 21.45% → low estimation error
 - PRED(0.30) = 85.71% → high level of prediction
 - $R^2 = 0.87$ → high correlation
 - **Schedule Model:**
 - MRE = 24.14% → Rather low estimation error
 - PRED(0.30) = 85.71% → high level of prediction
 - $R^2 = 0.92$ → strong correlation (model is consistent)



Thank You



Center for Systems and Software Engineering