CS3203: Software Engineering Project

SPA Development Process and API Design

By: Dr. Bimlesh Wadhwa

CS3203 Project Iterations

- Understand requirements
- Form teams
- Get a consultation slot

- Subset of BasicSPA Implementation
- Extensive Testing
- Submit code
- Submit draft of the report

- Basic + Some of Advanced SPA implementation
- Submit report
- Demo progress

- Full SPA implementation
- Extensions
- Extensive testing
- Submit code and report

- Present your project
- Q&A
- Discussion of test cases

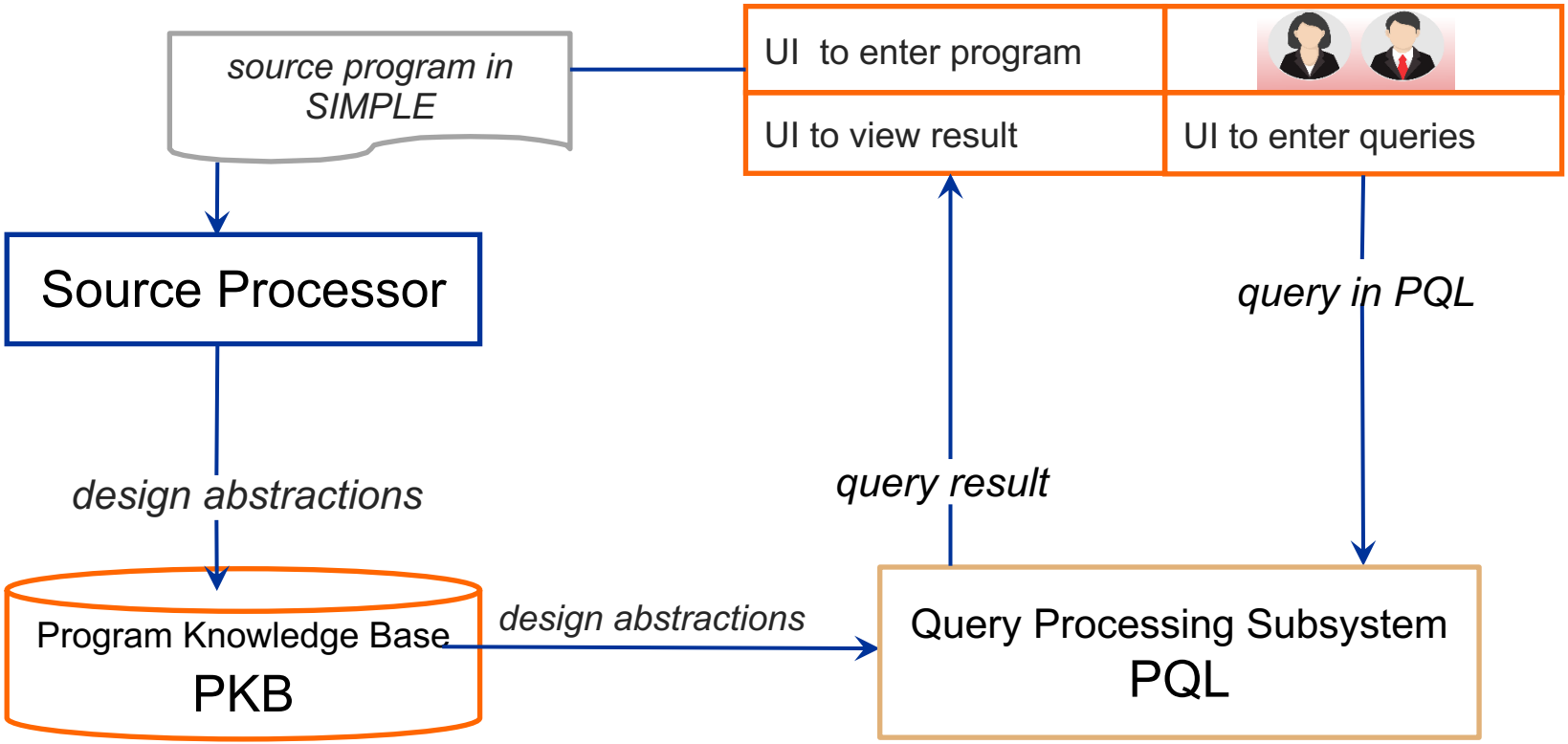
Week 1 → Week 13

SPA Development Process

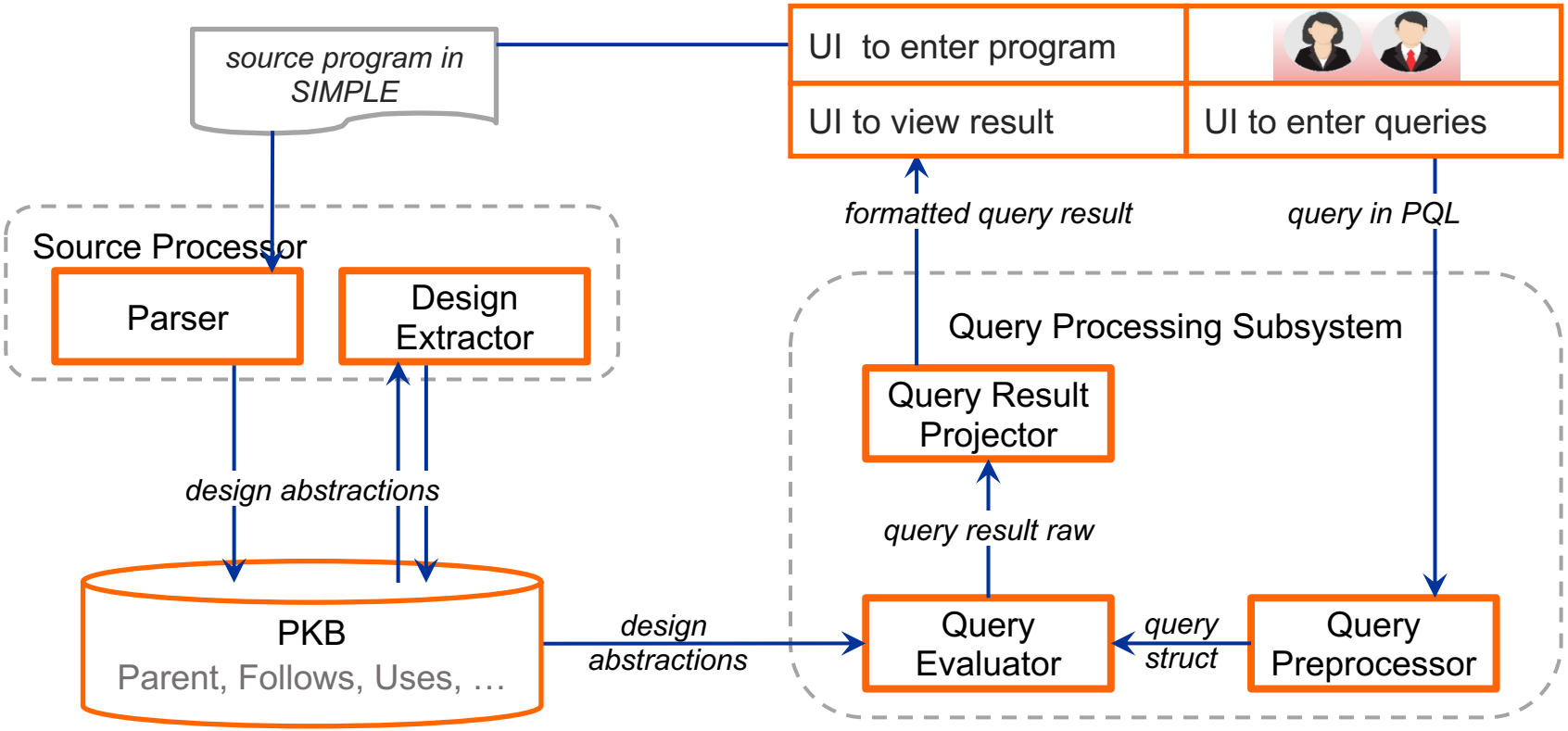
- Iterative model
  Breadth-first approach
- Organize the team, communication channels;
  Project schedule - phases, tasks, roles, milestones
- Work on methods and tools; SPA vocabulary;
  requirements, design models, C++, test cases

SPA Requirements to Functional SPA

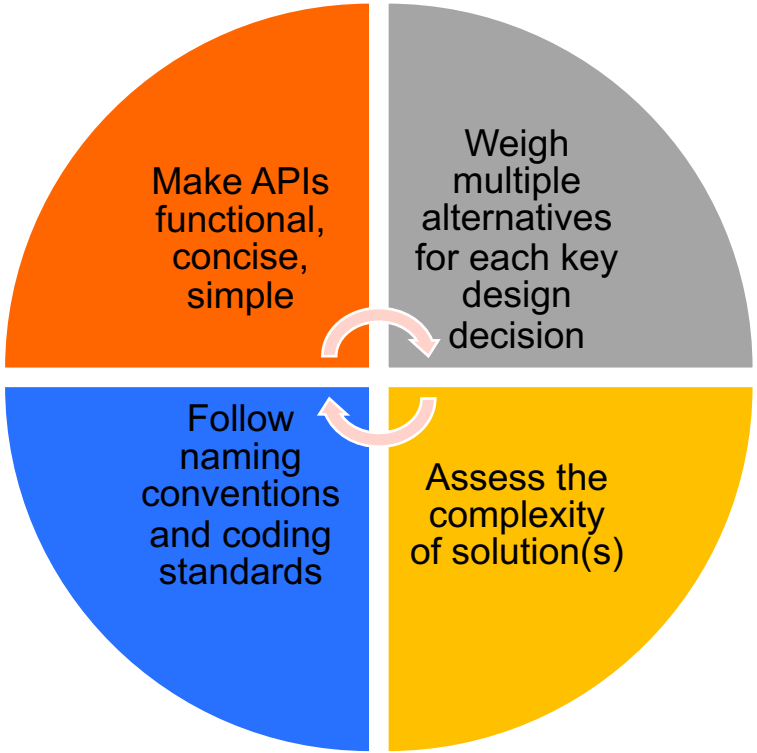
SPA – High Level View



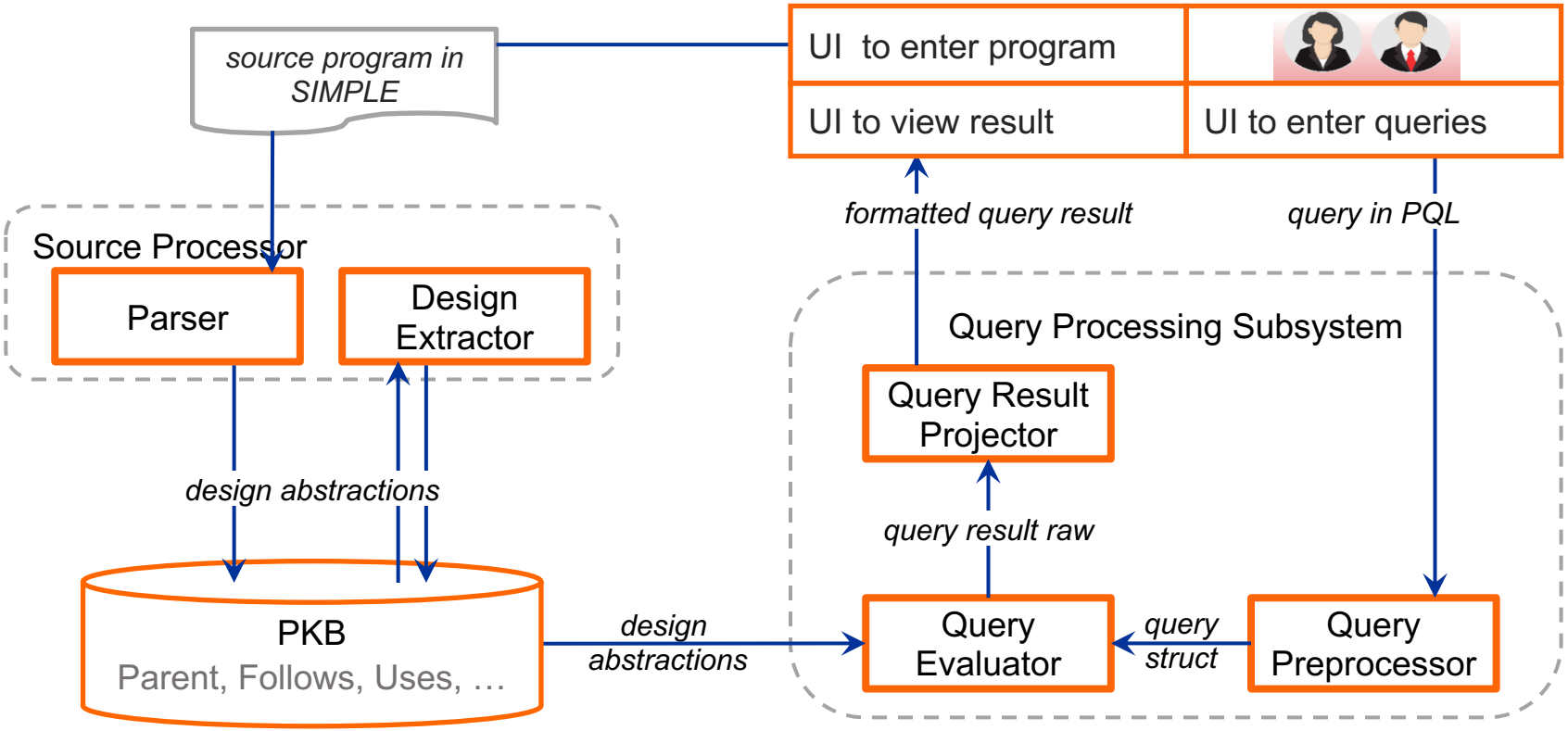
Identify Components



SPA Design Strategies



SPA Components



Component

(interchangeable with 'module' for CS3203)

- a well-defined **component**
- provides services *(computational elements)* to other components.

These structures do the work within your project. They interact with various other components, return results, and save data.

Module : Modules are basically a logical unit existing during design time. Examples : package classes, database tables. You use these module structures as a base for the components in the architecture.

Component: Components are physical units , exist during runtime.
Examples: threads, processes, objects.

API-first approach

An API-first approach : design and development of an application programming interface (API) before the implementation.

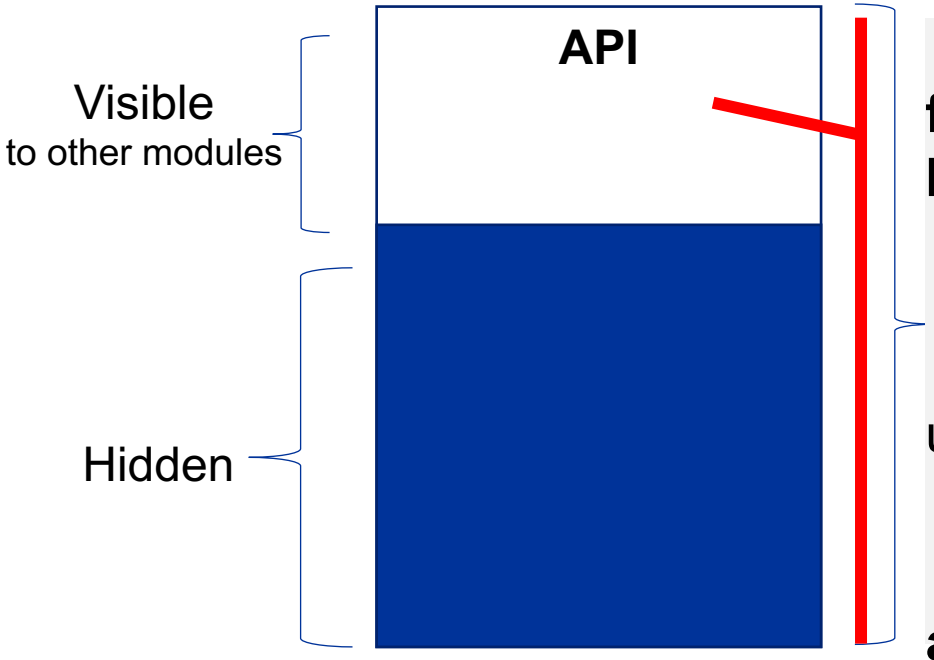
Code-first approach : your main focus is on “coding component functionality” not the “component interface”.

-Code (concrete) drives the implementation of the API(abstract)
API might need to be packed into place to accommodate the behavior of the code e.g. you can’t retrieve information the way you want to due to the way access is granted or the way a database is structured.

-User of API feels annexed on, like an afterthought to the code.

-A disjointed developer experience : team is more vulnerable to bottlenecks.

Specifying API

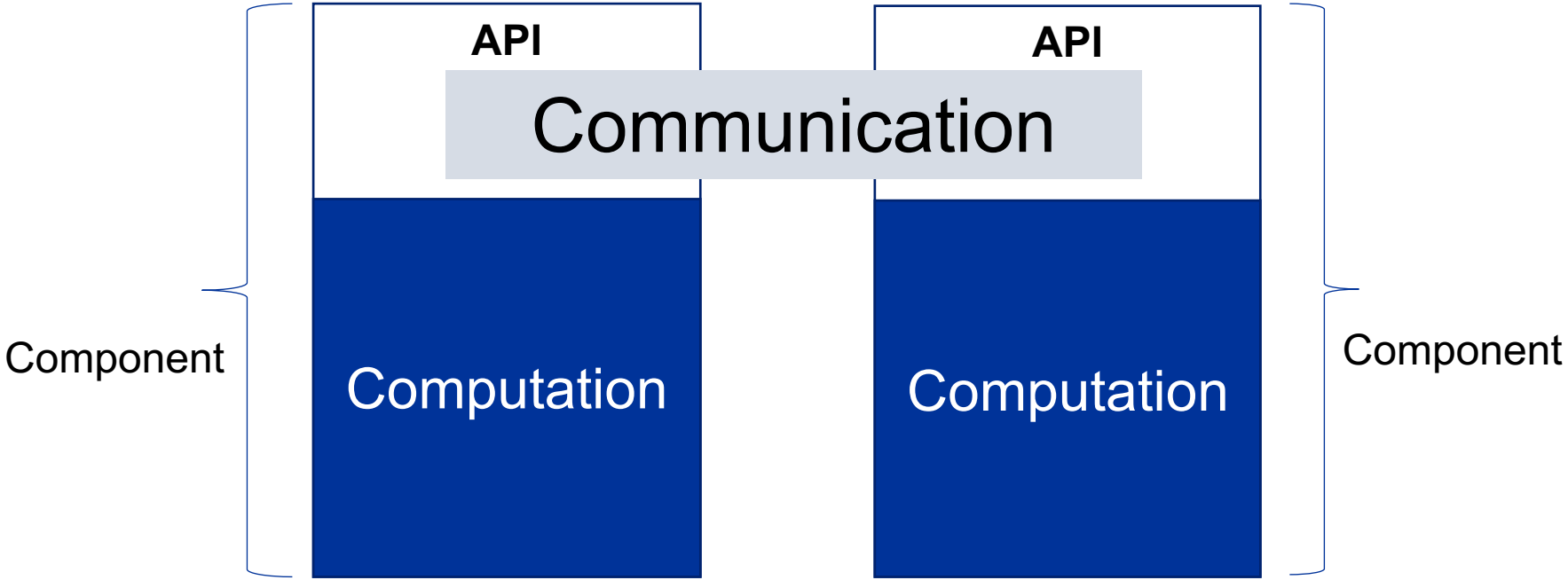


Interaction and information **flow from/to component boundaries**

It should be **well-defined**.
It should not reveal unnecessary details.

It acts as a firewall **preventing access to hidden parts**.

Component interactions



Example: Parser – PKB interaction to build AST

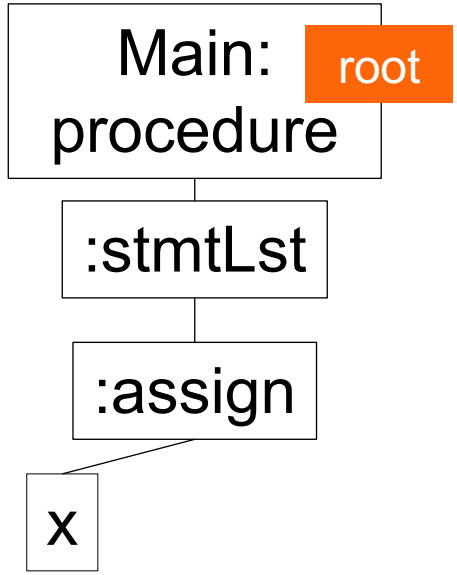
Shall we come up with an **abstract AST API** based on **our understanding of essential properties of AST**?

Ok, So I will give you **AST API - a set of interface operations to AST**. You will be able to work with AST using AST API.

Example: Parser – PKB interaction to build AST

```
procedure Main {
    x = y + z;}
```

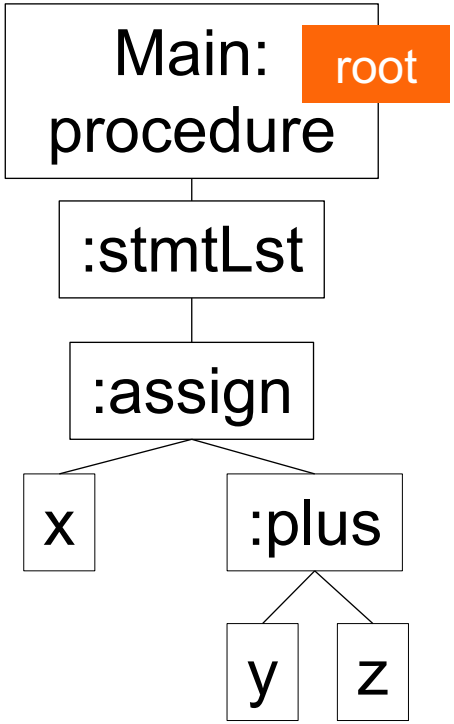
1. Create MainNode for procedure Main
2. Mark MainNode as the root of AST
3. Create a stmtLstNode for stmtLst
4. Link stmtLstNode as first child of MainNode
5. Create assignNode for assignment stmt
6. Link assignNode as first child of stmtLstNode
7. Create xNode for variable x
8. Link xNode as first child of assignNode



Example: Parser – PKB interaction to build AST

```
procedure Main {
    x = y + z;}
```

9. Create plusNode for “+”
10. Link plusNode as right sibling of xNode (second child of assignNode)
11. Create yNode for “y”
12. Link yNode as first child of plusNode
13. Create a zNode for “z”
14. Link zNode as right sibling of yNode (second child of plusNode)



Example: Parser – PKB interaction to build AST

1. Create MainNode for procedure Main
   - ***TNode CreateTNodeProc()*** – creates AST node for procedure & returns reference to it
2. Mark MainNode as the root of AST
   - ***SetAsRoot (TNode root)*** – sets TNode as the root of AST
3. Create a stmtLstNode for stmtLst
   - ***TNode CreateTNodeStmtLst()*** - creates StmtLstNode & returns its reference
4. Link stmtLstNode as first child of MainNode
   - ***SetFirstChild (TNode P, C)*** – sets C as the first child of P

Simplify & Refine API

- Simplify: you may design a few operations for creating AST nodes for various design entities e.g. procedure, stmtLst, assign, var

  createTNodeProc(), createTNodeStmtLst(), etc.

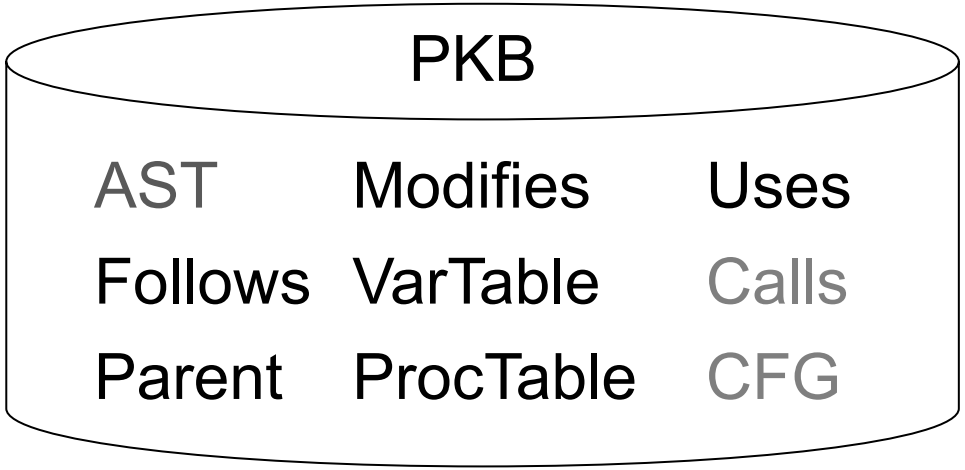
  An alternative could be a more abstract or a more general API:
  createTNode(DE) – creates an AST node for a DE

- Refine: Describe normal and abnormal behavior (exceptions, errors, wrong input, pre-conditions)

Consolidate API

also called module facade

Follows_API	Uses_API	VarTable API	AST API	.....



API Specification Format

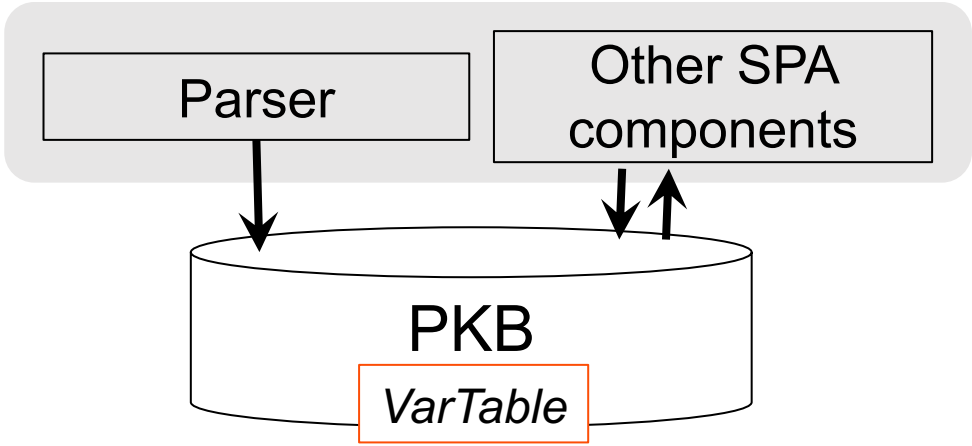
Name {
Overview: purpose/responsibility of the API

Public interface: interface operations
  Operation header: return-value op-name (type parameter(s))
    <pre-condition>, if any
     Description: - describe what the operation does
                       - describe both normal and abnormal behavior
}

Example: VarTable

- VarTable stores all program variables

Index	Variable Name
1	x
2	y
3	z
4	i



Example: VarTable

VarTable *Overview:* VarTable stores variable information from the Parser.
API
INDEX insertVar (**STRING** varName); ***Description***: If varName is not in the VarTable, inserts varName into the VarTable and returns its index.
STRING getVarName (**INDEX** ind); ***Description***: Returns the name of a variable at VarTable [ind] If 'ind' is out of range, error (or throw exception)
INDEX getVarIndex (**STRING** varName); ***Description***: If varName is in VarTable, returns its index; otherwise, returns -1 (special value)

VarTable API (refined)

VarTable *Overview:* VarTable stores variable information from the Parser.
API
INDEX insertVar (**STRING** varName); ***Description***: If varName is not in the VarTable, inserts varName into the VarTable and returns its index. Else returns its index the table remains unchanged.
STRING getVarName (**INDEX** ind); ***Description***: Returns the name of a variable at VarTable [ind] If 'ind' is out of range, error (or throw exception)
INDEX getVarIndex (**STRING** varName); ***Description***: If varName is in VarTable, returns its index; otherwise, returns -1 (special value)
INTEGER getSize (); ***Description:*** Returns the number of variables in the table

Abstract vs Concrete APIs

- Abstract APIs
  - Conceptual, symbolic, no implementation concerns
- Concrete API
  - Public interfaces of the classes implementing modules

Example

Abstract API	Concrete API
STMT_SET getModifies (**VARINDEX** var)	**STMT_SET** getModifies (**VARINDEX** var) **typedef int VARINDEX;** **typedef int STMT_SET [30];**
TNODE_SET getFollows* (**TNODE** n)	**TNODE_SET** getFollowsStar (**TNODE** n) struct TNODE {…} typedef TNODE TNODE_SET [100]
VAR_SET getUses (**STMT** s)	**VAR_SET** getUses (**stmt** s) typedef int stmt; typedef string VAR_SET[100];

Example
(Abstract API)

- **BOOLEAN** `setFollows (`**STMT_NO** `stmt1,` **STMT_NO** `stmt2)`
  Stores the Follows(stmt1, stmt2) information into the PKB. Returns TRUE if information is added successfully or if information already exist in PKB, returns FALSE otherwise.
- **BOOLEAN** `isFollows (`**STMT_NO** `stmt1,` **STMT_NO** `stmt2)`
  Returns TRUE if Follows(stmt1, stmt2) holds and the information is stored in the PKB, returns FALSE otherwise.
- **LIST_OF_STMT_NO** `getStmtsFollowedBy (`**STMT_NO** `stmt2)`
  Returns a list of statement numbers s_1, s_2, …, s_n which are followed by stmt2, i.e. Follows(s_1, stmt2), Follows(s_2, stmt2), …, Follows(s_n, stmt2) holds.
- **LIST_OF_STMT_NO** `getStmtsFollows (`**STMT_NO** `stmt1)`
  Returns a list of statement numbers s_1, s_2, …, s_n which follows stmt1, i.e. Follows(stmt1, s_1), Follows(stmt1, s_2), …, Follows(stmt1, s_n) holds.

Example
(Concrete API)

- `typedef int STMT_NO;`
- `typedef int LIST_OF_STMT_NO[1000];`

- `bool setFollows (STMT_NO stmt1, STMT_NO stmt2) { … }`
- `bool isFollows (STMT_NO stmt1, STMT_NO stmt2) { … }`
- `LIST_OF_STMT_NO getStmtsFollowedBy (STMT_NO stmt2) { … }`
- `LIST_OF_STMT_NO getStmtsFollows (STMT_NO stmt1) { … }`

Strategy to Systematically Discover SPA components' APIs

- Identify architectural components
- Understand associations among design abstractions.
- Understand interactions among SPA components and design abstractions.
- Discover and Document API from above associations and interactions

Strategy to Systematically Discover APIs

- Identify architectural components
- Understand associations among design abstractions.
- Understand interactions among SPA components and design abstractions.
- Discover and Document API from above associations and interactions

UML Class diagram

Strategy to Systematically Discover APIs

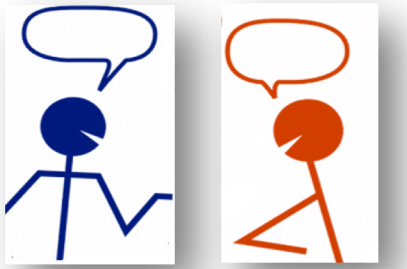
- Identify architectural components
- Understand associations among design abstractions.
- Understand interacti
  components and des
- Discover and Document API from above associations and interactions

UML Class diagram

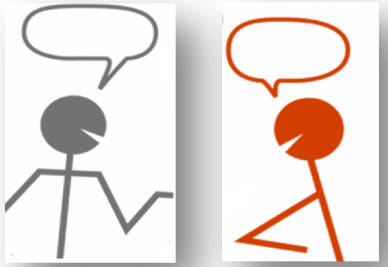
UML Sequence diagram

Tip for Designing APIs

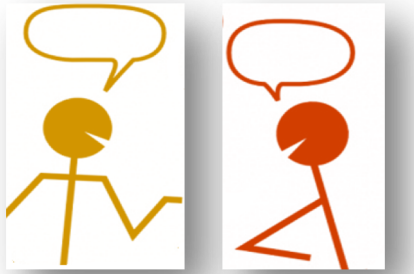
Work in pairs to discover interfaces



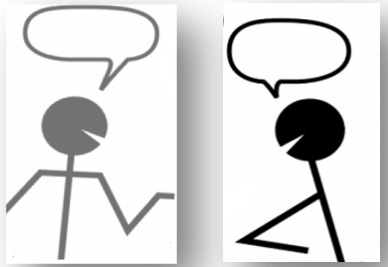
Parser → PKB



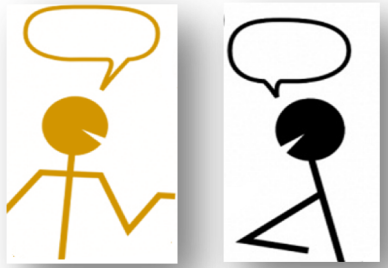
PKB → DE



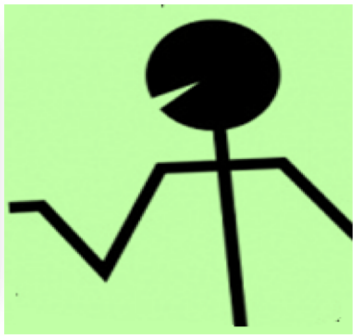
QE → PKB



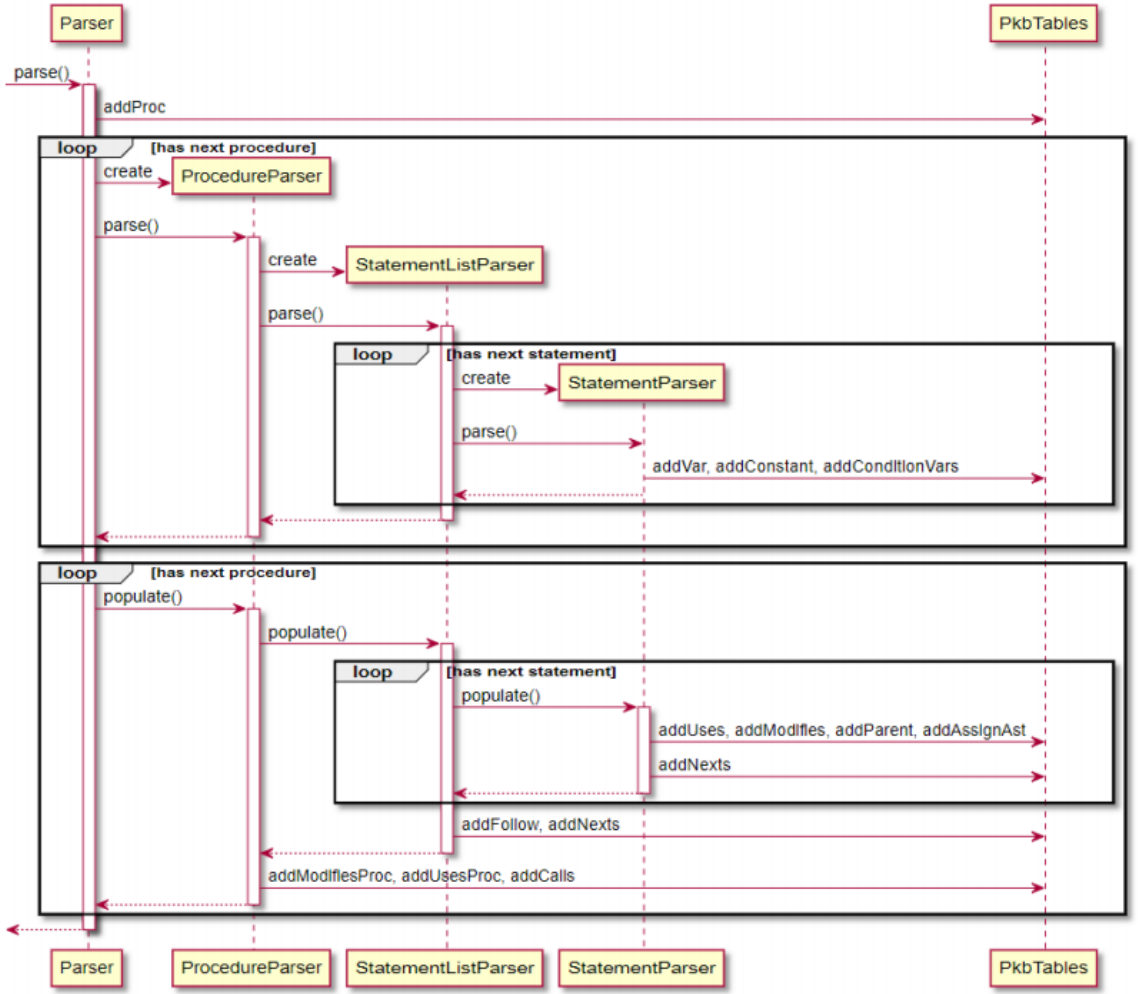
QPP → QE



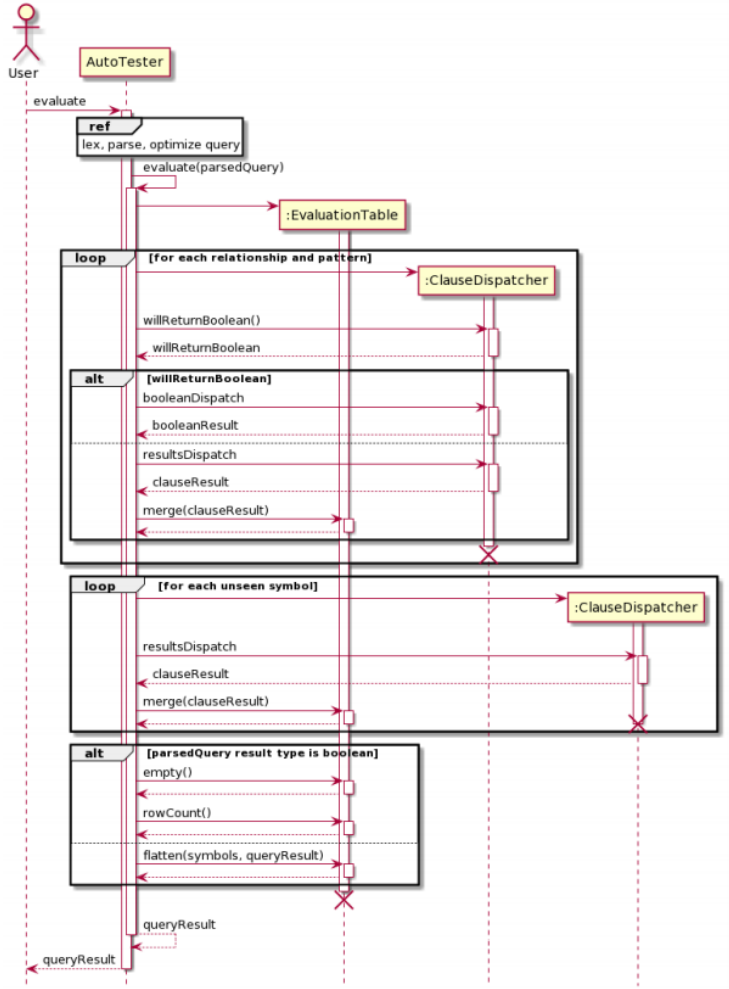
QE → QRP



Example: Interaction between Parser & PKB



Example: Interaction for Query Processing



Quality of APIs

Communication

- Components communicate in terms of APIs
- Team members communicate in terms of APIs

Common practice

- Public or Open APIs

Quality matters

- Good API: saves time
- Bad API: hinders productivity
- Incorrect APIs: project **disaster**

Summary

API-first approach : Early validation; Abstraction Layer

Decoupled Dependencies; Parallel/Faster development

APIs should be simple, concise, precise, unambiguous, complete, stable & easy to understand.

Use Standard / Consistent naming conventions & documentation format Use symbolic names for types: STRING, INTEGER