CS3213: Foundations of Software Engineering

In-class Lecture and Exercises

Bioblitz

- 112 observations
- 62 species
- 5 observers
- 49 identifiers

Pink-necked Green-Pigeon (dhrsai)

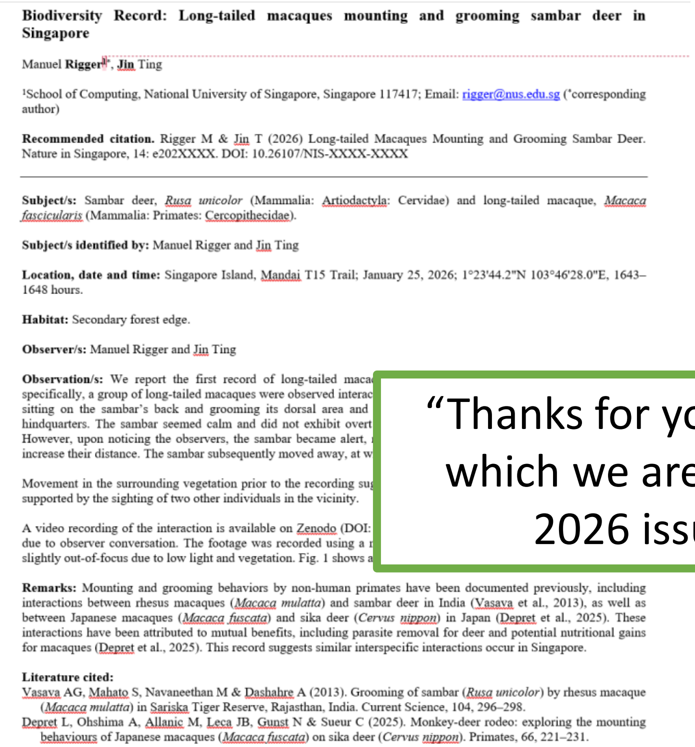
Giant Forest Ant (j_os_h_in)

Slaty-breasted Rail (mrigger)

Long-tailed Macaque (j_os_h_in)

https://www.inaturalist.org/projects/nus-cs3213-2025

Long-tailed Macaques Grooming Sambar Deer



Biodiversity Record: Long-tailed macaques mounting and grooming sambar deer in Singapore

Manuel **Rigger**[1]*, **Jin** Ting

[1]School of Computing, National University of Singapore, Singapore 117417; Email: rigger@nus.edu.sg (*corresponding author)

Recommended citation. Rigger M & Jin T (2026) Long-tailed Macaques Mounting and Grooming Sambar Deer. Nature in Singapore, 14: e202XXXX. DOI: 10.26107/NIS-XXXX-XXXX

Subject/s: Sambar deer, *Rusa unicolor* (Mammalia: Artiodactyla: Cervidae) and long-tailed macaque, *Macaca fascicularis* (Mammalia: Primates: Cercopithecidae).

Subject/s identified by: Manuel Rigger and Jin Ting

Location, date and time: Singapore Island, Mandai T15 Trail; January 25, 2026; 1°23'44.2"N 103°46'28.0"E, 1643–1648 hours.

Habitat: Secondary forest edge.

Observer/s: Manuel Rigger and Jin Ting

Observation/s: We report the first record of long-tailed maca
specifically, a group of long-tailed macaques were observed interac
sitting on the sambar's back and grooming its dorsal area and
hindquarters. The sambar seemed calm and did not exhibit overt
However, upon noticing the observers, the sambar became alert,
increase their distance. The sambar subsequently moved away, at w

Movement in the surrounding vegetation prior to the recording su
supported by the sighting of two other individuals in the vicinity.

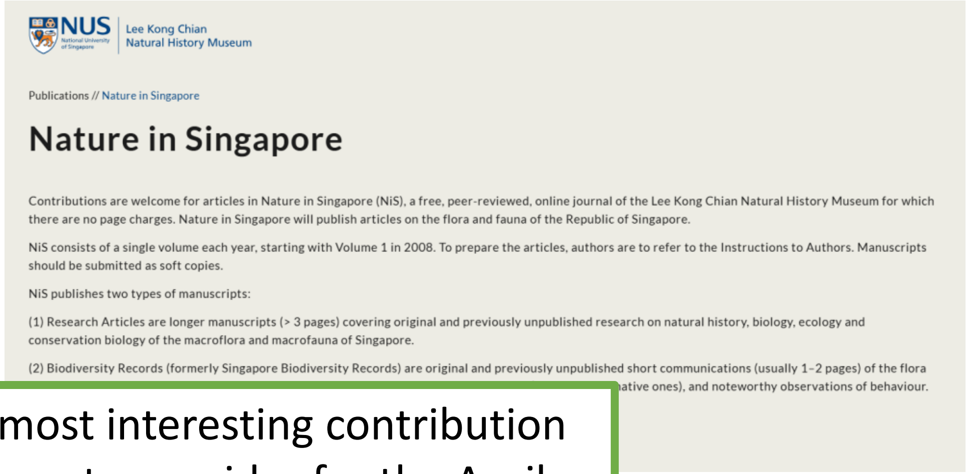
A video recording of the interaction is available on Zenodo (DOI:
due to observer conversation. The footage was recorded using a
slightly out-of-focus due to low light and vegetation. Fig. 1 shows a

Remarks: Mounting and grooming behaviors by non-human primates have been documented previously, including interactions between rhesus macaques (*Macaca mulatta*) and sambar deer in India (Vasava et al., 2013), as well as between Japanese macaques (*Macaca fuscata*) and sika deer (*Cervus nippon*) in Japan (Depret et al., 2025). These interactions have been attributed to mutual benefits, including parasite removal for deer and potential nutritional gains for macaques (Depret et al., 2025). This record suggests similar interspecific interactions occur in Singapore.

Literature cited:

Vasava AG, Mahato S, Navaneethan M & Dashahre A (2013). Grooming of sambar (*Rusa unicolor*) by rhesus macaque (*Macaca mulatta*) in Sariska Tiger Reserve, Rajasthan, India. Current Science, 104, 296–298.

Depret L, Ohshima A, Allanic M, Leca JB, Gunst N & Sueur C (2025). Monkey-deer rodeo: exploring the mounting behaviours of Japanese macaques (*Macaca fuscata*) on sika deer (*Cervus nippon*). Primates, 66, 221–231.



NUS National University of Singapore | Lee Kong Chian Natural History Museum

Publications // Nature in Singapore

Nature in Singapore

Contributions are welcome for articles in Nature in Singapore (NiS), a free, peer-reviewed, online journal of the Lee Kong Chian Natural History Museum for which there are no page charges. Nature in Singapore will publish articles on the flora and fauna of the Republic of Singapore.

NiS consists of a single volume each year, starting with Volume 1 in 2008. To prepare the articles, authors are to refer to the Instructions to Authors. Manuscripts should be submitted as soft copies.

NiS publishes two types of manuscripts:

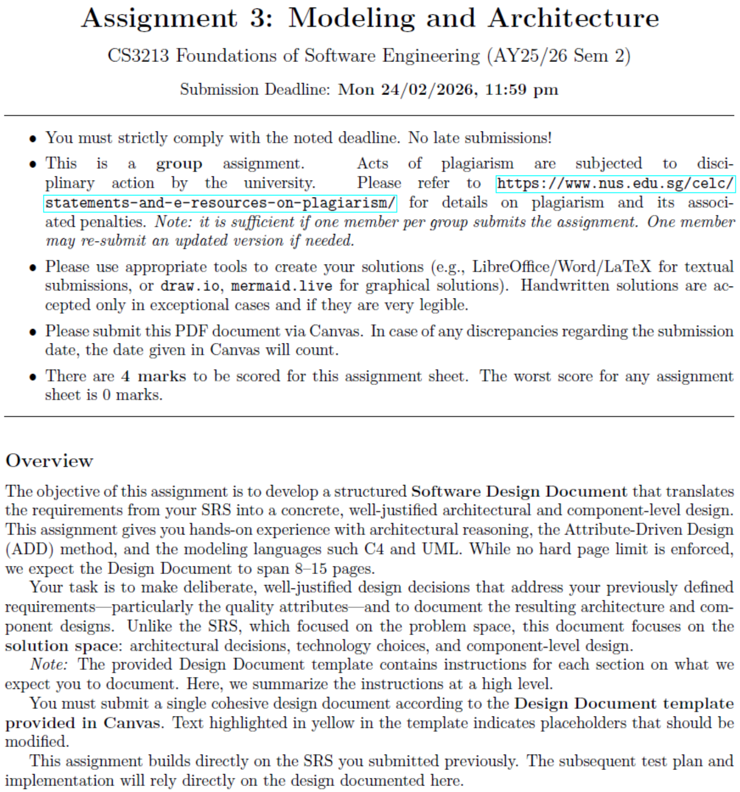
(1) Research Articles are longer manuscripts (> 3 pages) covering original and previously unpublished research on natural history, biology, ecology and conservation biology of the macroflora and macrofauna of Singapore.

(2) Biodiversity Records (formerly Singapore Biodiversity Records) are original and previously unpublished short communications (usually 1–2 pages) of the flora
ative ones), and noteworthy observations of behaviour.

"Thanks for your most interesting contribution which we are happy to consider for the April 2026 issue of Biodiversity Records."

RBL Tree Planting (~120 food plants)

Assignment



Assignment 3: Modeling and Architecture

CS3213 Foundations of Software Engineering (AY25/26 Sem 2)

Submission Deadline: **Mon 24/02/2026, 11:59 pm**

- You must strictly comply with the noted deadline. No late submissions!
- This is a **group** assignment. Acts of plagiarism are subjected to disciplinary action by the university. Please refer to `https://www.nus.edu.sg/celc/statements-and-e-resources-on-plagiarism/` for details on plagiarism and its associated penalties. *Note: it is sufficient if one member per group submits the assignment. One member may re-submit an updated version if needed.*
- Please use appropriate tools to create your solutions (e.g., LibreOffice/Word/LaTeX for textual submissions, or `draw.io`, `mermaid.live` for graphical solutions). Handwritten solutions are accepted only in exceptional cases and if they are very legible.
- Please submit this PDF document via Canvas. In case of any discrepancies regarding the submission date, the date given in Canvas will count.
- There are **4 marks** to be scored for this assignment sheet. The worst score for any assignment sheet is 0 marks.

Overview

The objective of this assignment is to develop a structured **Software Design Document** that translates the requirements from your SRS into a concrete, well-justified architectural and component-level design. This assignment gives you hands-on experience with architectural reasoning, the Attribute-Driven Design (ADD) method, and the modeling languages such C4 and UML. While no hard page limit is enforced, we expect the Design Document to span 8–15 pages.

Your task is to make deliberate, well-justified design decisions that address your previously defined requirements—particularly the quality attributes—and to document the resulting architecture and component designs. Unlike the SRS, which focused on the problem space, this document focuses on the **solution space**: architectural decisions, technology choices, and component-level design.

Note: The provided Design Document template contains instructions for each section on what we expect you to document. Here, we summarize the instructions at a high level.

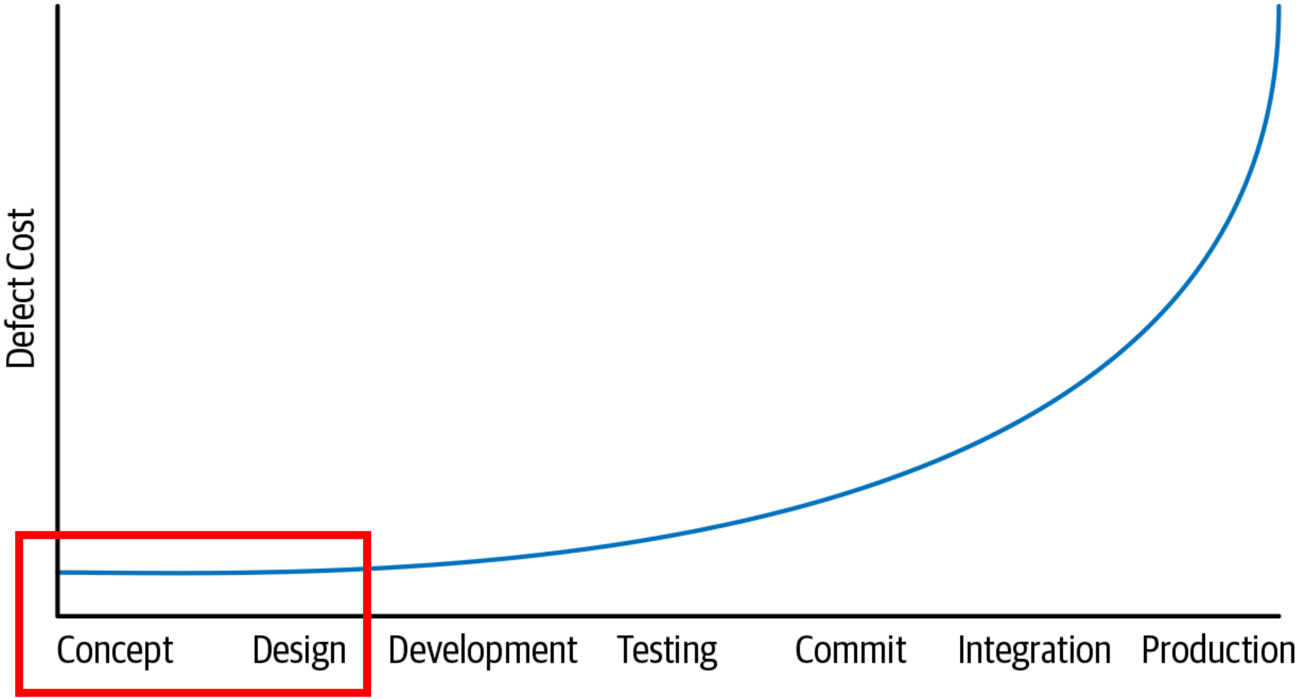
You must submit a single cohesive design document according to the **Design Document template provided in Canvas**. Text highlighted in yellow in the template indicates placeholders that should be modified.

This assignment builds directly on the SRS you submitted previously. The subsequent test plan and implementation will rely directly on the design documented here.

Recap

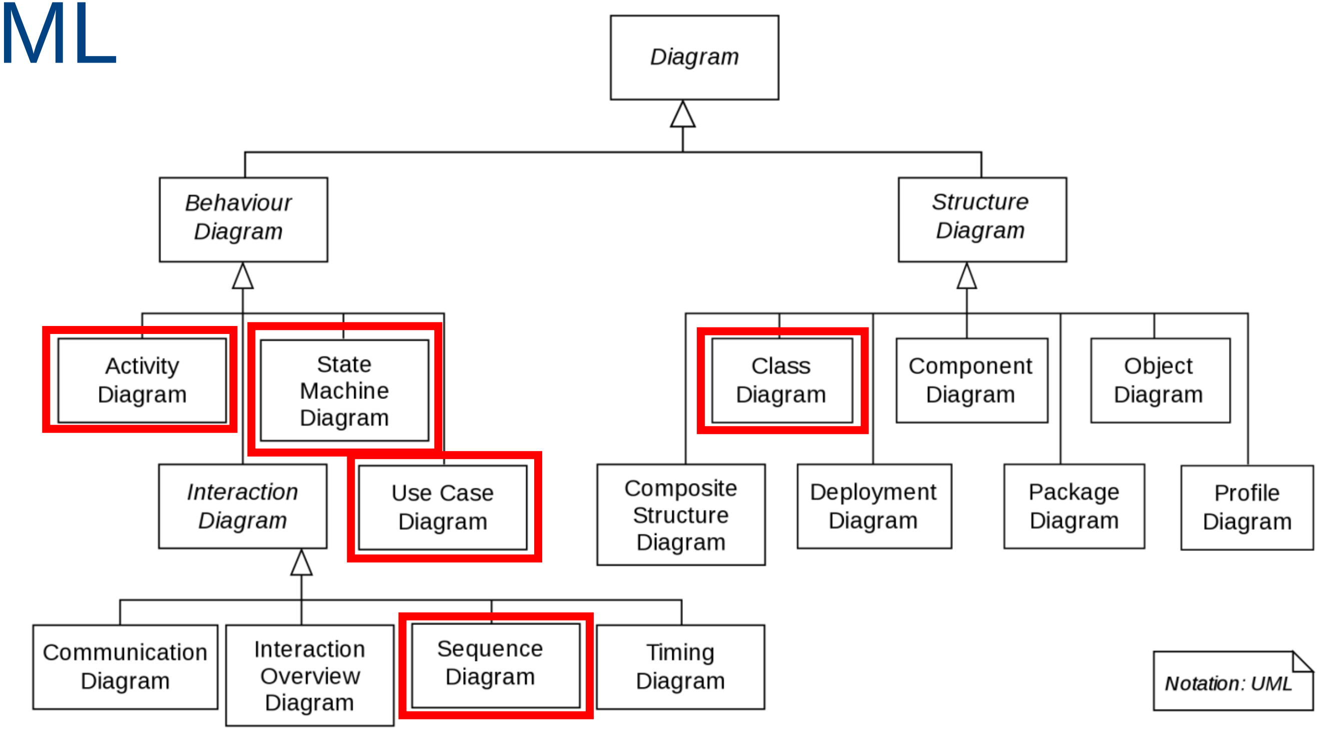
UML. C4. Architecture. Tactics and Patterns.

Addressing and Prevent Defects Early On



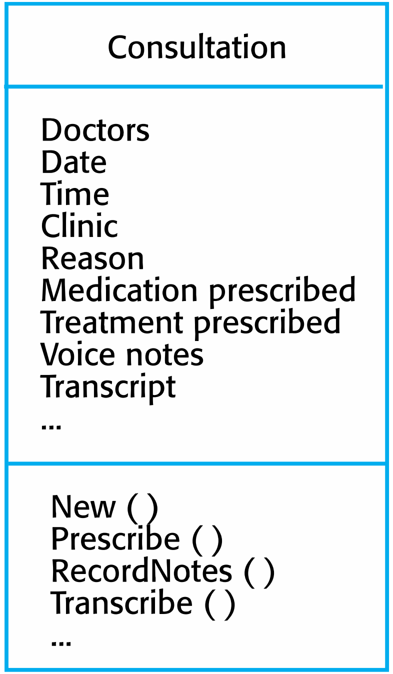
https://abseil.io/resources/swe-book/html/ch01.html

UML



https://en.wikipedia.org/wiki/Unified_Modeling_Language

Consultation Class

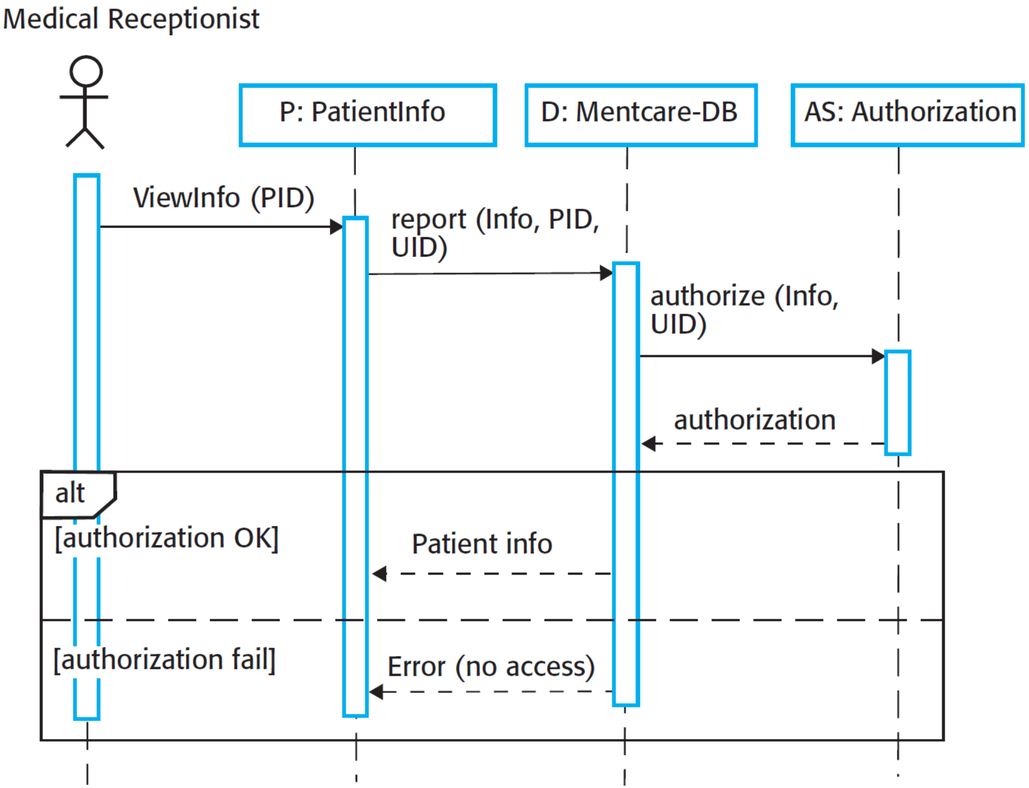


Consultation

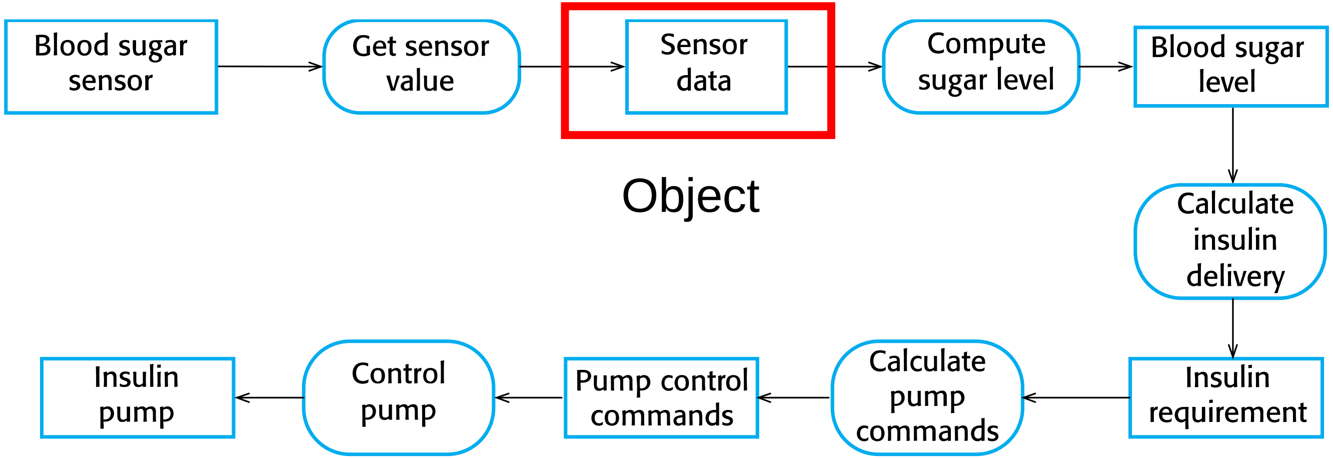
Doctors
Date
Time
Clinic
Reason
Medication prescribed
Treatment prescribed
Voice notes
Transcript
…

New ( )
Prescribe ( )
RecordNotes ( )
Transcribe ( )
…

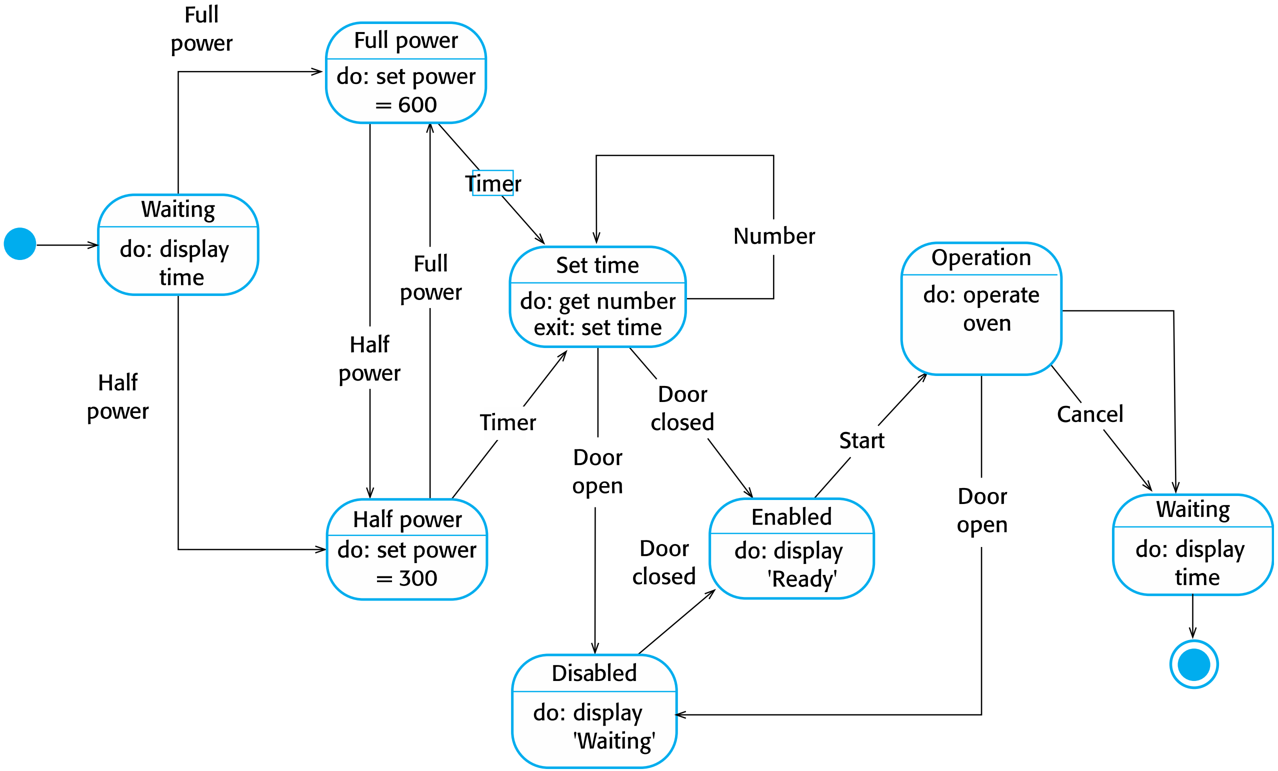
Sequence Diagrams



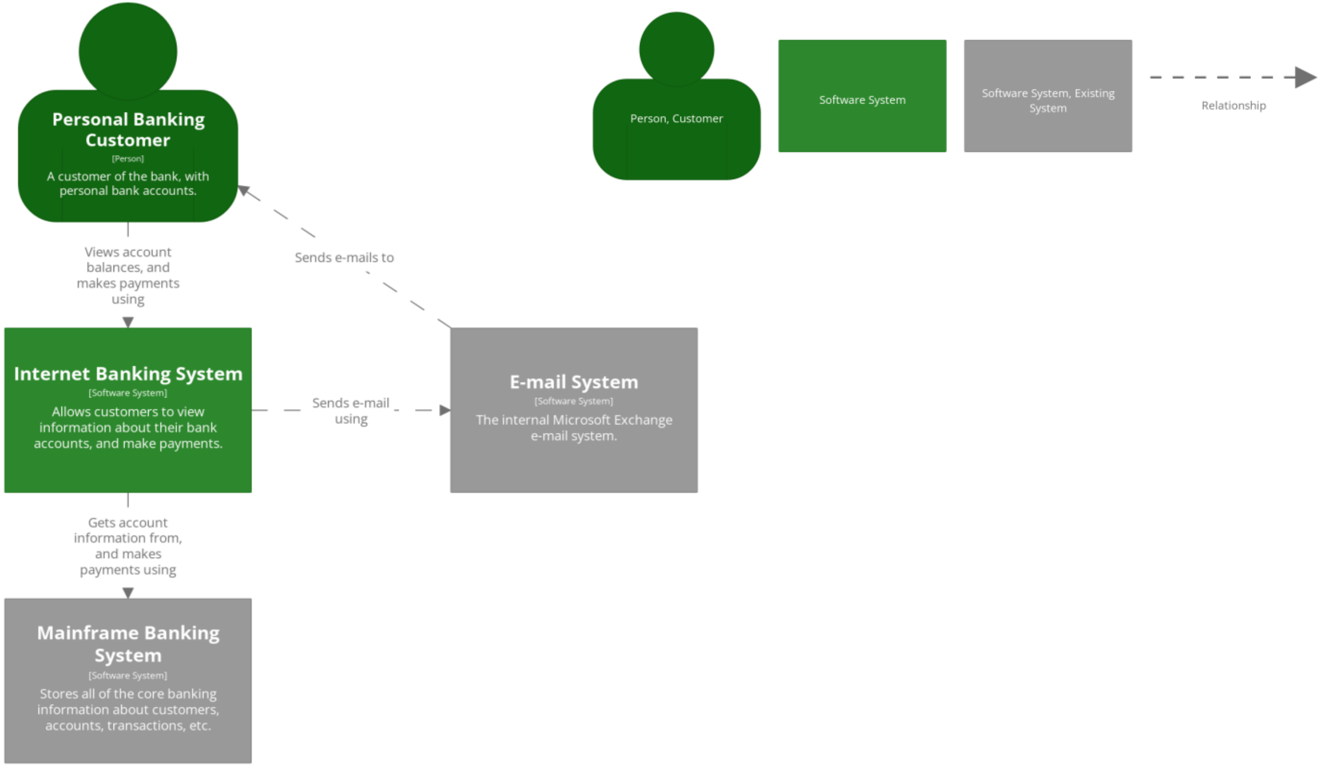
Activity Diagram



State Machine Diagram



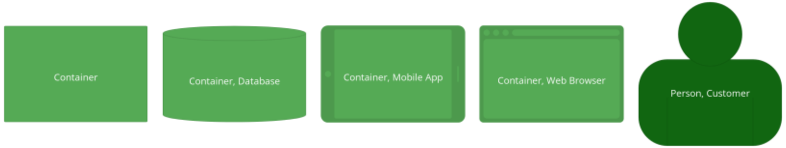
C4: System Context Diagram



Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.
Views account balances, and makes payments using
Sends e-mails to
Internet Banking System
[Software System]
Allows customers to view information about their bank accounts, and make payments.
Sends e-mail using
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.
Gets account information from, and makes payments using
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.
Person, Customer
Software System
Software System, Existing System
Relationship

https://c4model.com/diagrams/system-context

C4: Container Diagram



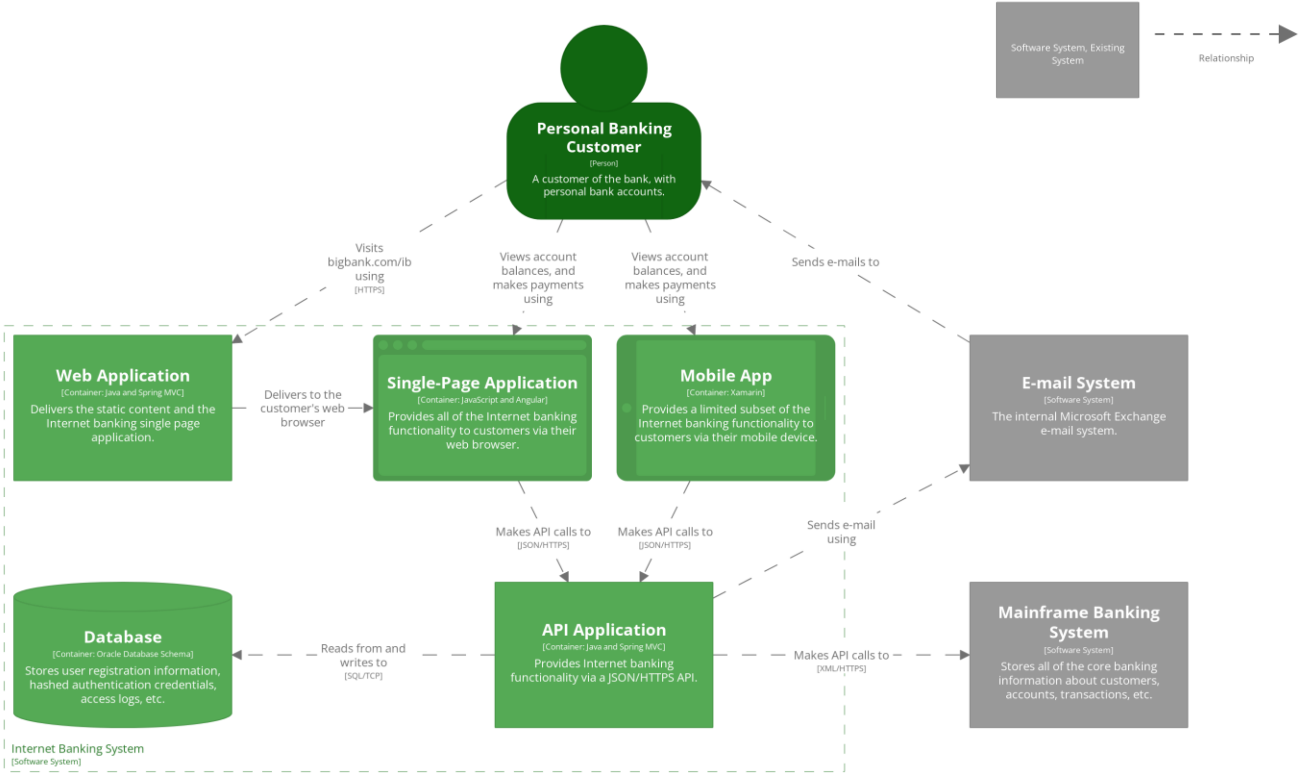
Container

Container, Database

Container, Mobile App

Container, Web Browser

Person, Customer



Software System, Existing System

Relationship

Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.

Visits bigbank.com/ib using
[HTTPS]

Views account balances, and makes payments using

Views account balances, and makes payments using

Sends e-mails to

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

Delivers to the customer's web browser

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Sends e-mail using

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

Reads from and writes to
[SQL/TCP]

API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.

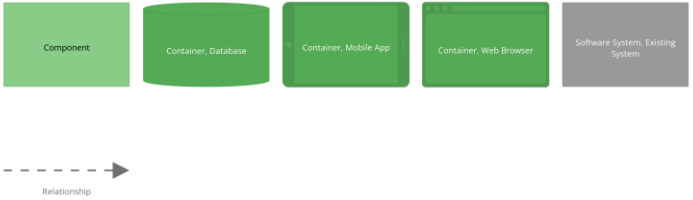
Makes API calls to
[XML/HTTPS]

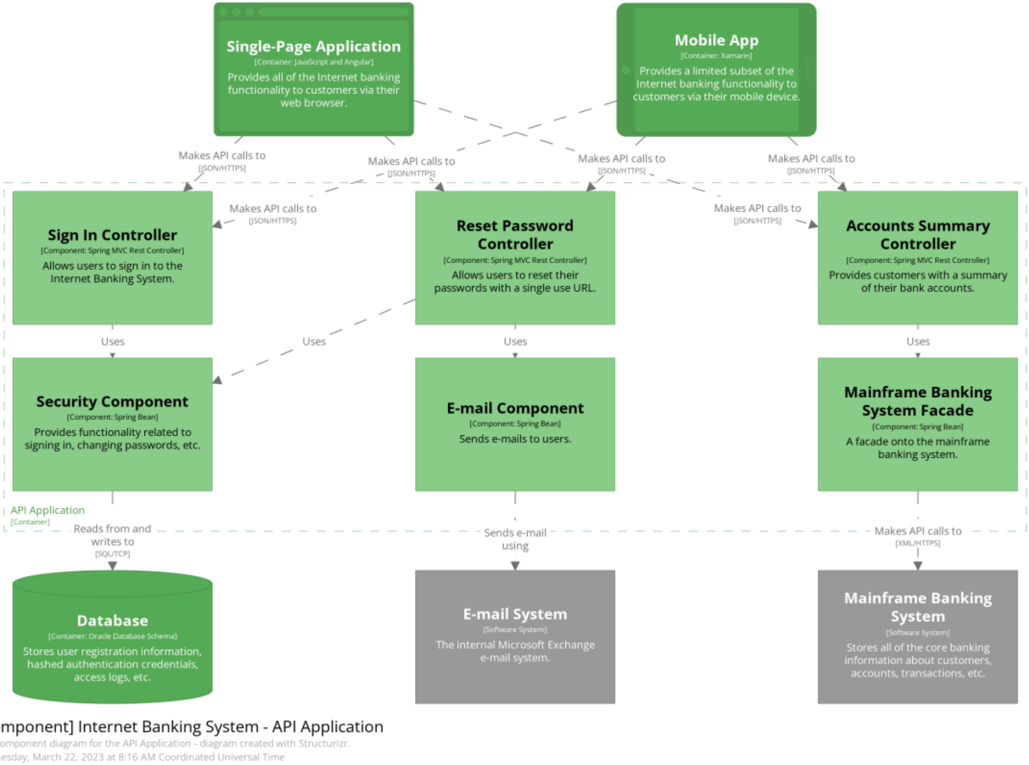
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Internet Banking System
[Software System]

https://c4model.com/diagrams/container

C4: Component Diagram





Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.

Reset Password Controller
[Component: Spring MVC Rest Controller]
Allows users to reset their passwords with a single use URL.

Accounts Summary Controller
[Component: Spring MVC Rest Controller]
Provides customers with a summary of their bank accounts.

Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.

E-mail Component
[Component: Spring Bean]
Sends e-mails to users.

Mainframe Banking System Facade
[Component: Spring Bean]
A facade onto the mainframe banking system.

API Application
[Container]

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

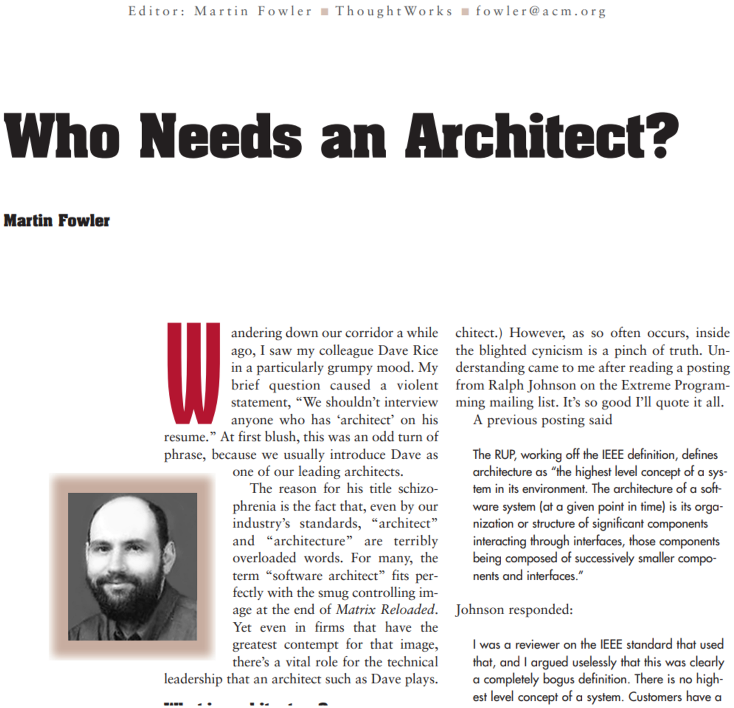
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

[Component] Internet Banking System - API Application
The component diagram for the API Application - diagram created with Structurizr.
Wednesday, March 22, 2023 at 8:16 AM Coordinated Universal Time

https://c4model.com/diagrams/component

What's Architecture?



Editor: Martin Fowler ■ ThoughtWorks ■ fowler@acm.org

Who Needs an Architect?

Martin Fowler

Wandering down our corridor a while ago, I saw my colleague Dave Rice in a particularly grumpy mood. My brief question caused a violent statement, "We shouldn't interview anyone who has 'architect' on his resume." At first blush, this was an odd turn of phrase, because we usually introduce Dave as one of our leading architects.

The reason for his title schizophrenia is the fact that, even by our industry's standards, "architect" and "architecture" are terribly overloaded words. For many, the term "software architect" fits perfectly with the smug controlling image at the end of *Matrix Reloaded*. Yet even in firms that have the greatest contempt for that image, there's a vital role for the technical leadership that an architect such as Dave plays.

chitect.) However, as so often occurs, inside the blighted cynicism is a pinch of truth. Understanding came to me after reading a posting from Ralph Johnson on the Extreme Programming mailing list. It's so good I'll quote it all.

A previous posting said

> The RUP, working off the IEEE definition, defines architecture as "the highest level concept of a system in its environment. The architecture of a software system (at a given point in time) is its organization or structure of significant components interacting through interfaces, those components being composed of successively smaller components and interfaces."

Johnson responded:

> I was a reviewer on the IEEE standard that used that, and I argued uselessly that this was clearly a completely bogus definition. There is no highest level concept of a system. Customers have a

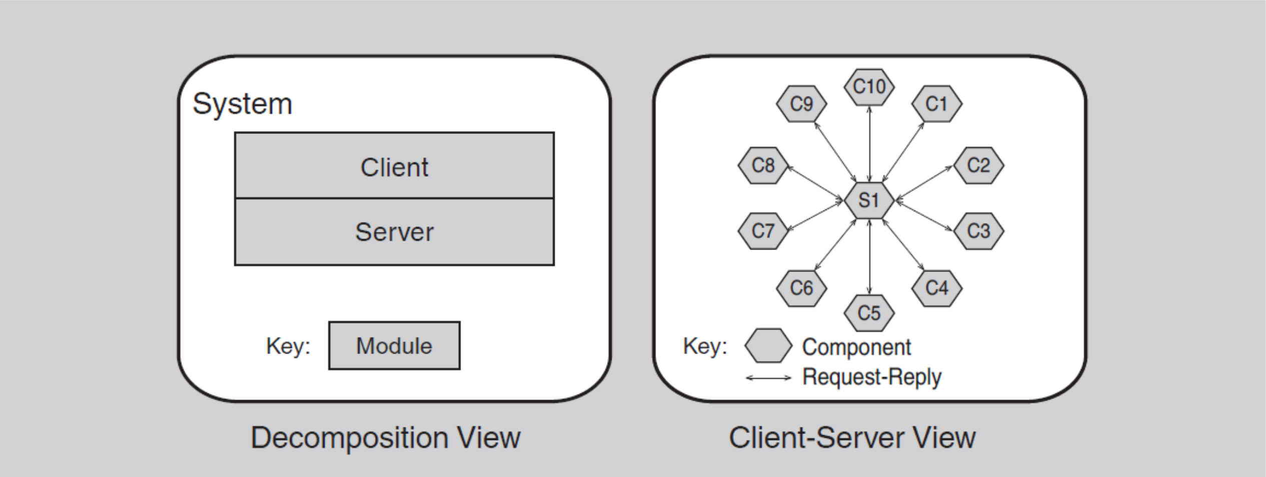
https://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf

Three Kind of Structures

- *Component-and-connector (C&C) structures*
- *Module structures*
- *Allocation structures*

Focus on the way components interact with each other at run time

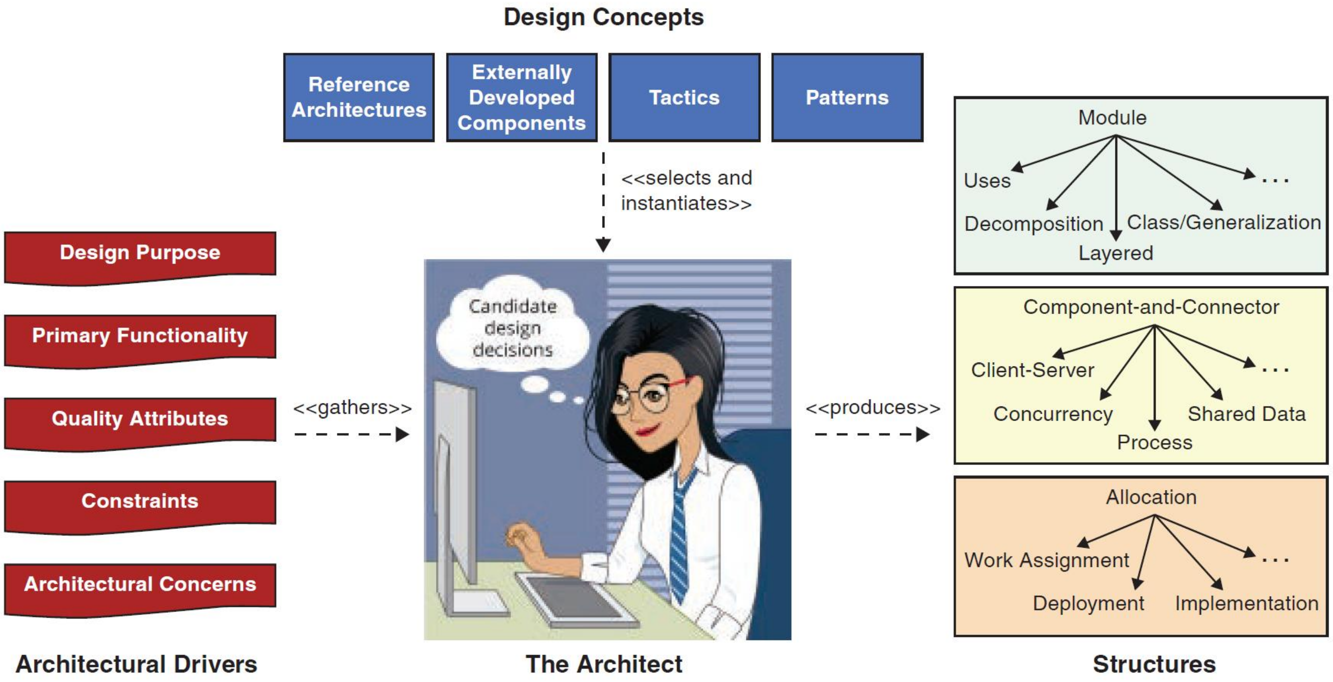
Which Structure(s) to Model?



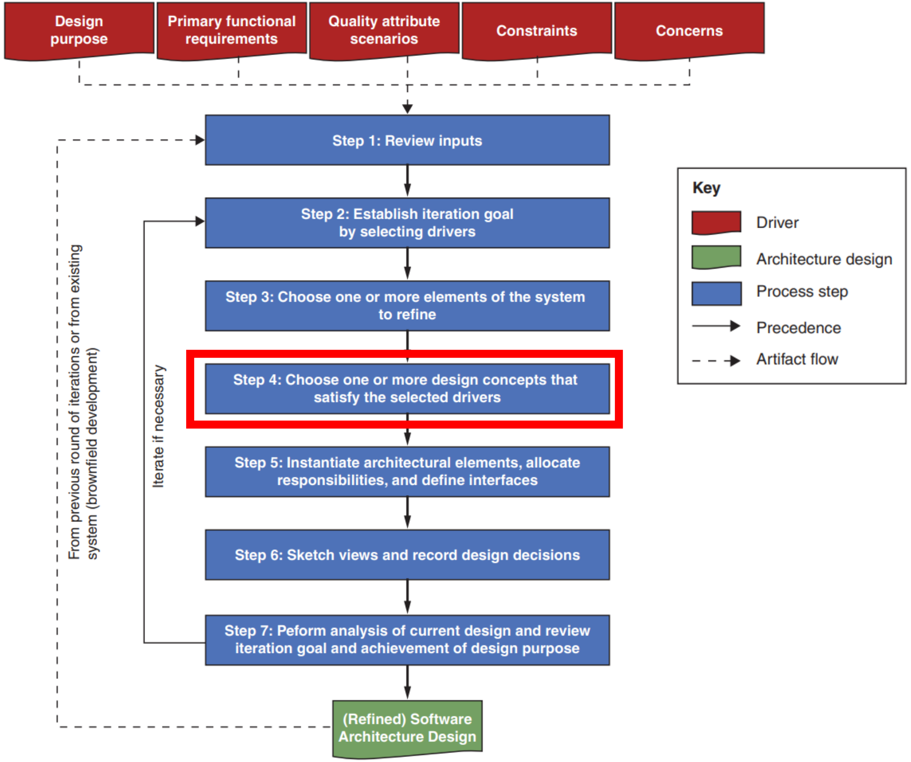
Decomposition View

Client-Server View

Approaching Architecture



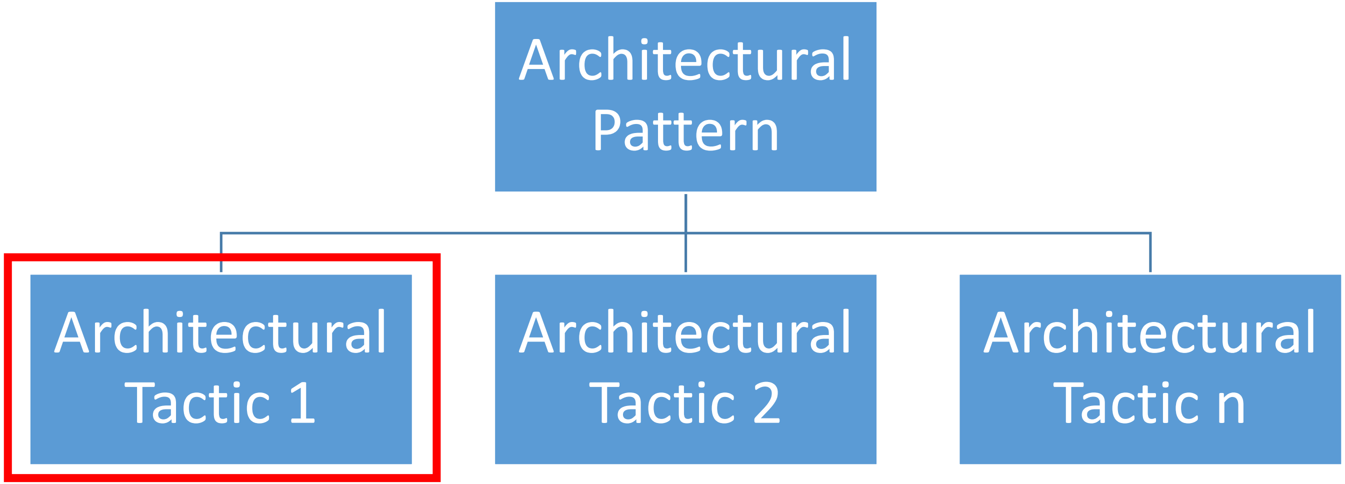
Attribute-Driven Design (ADD)



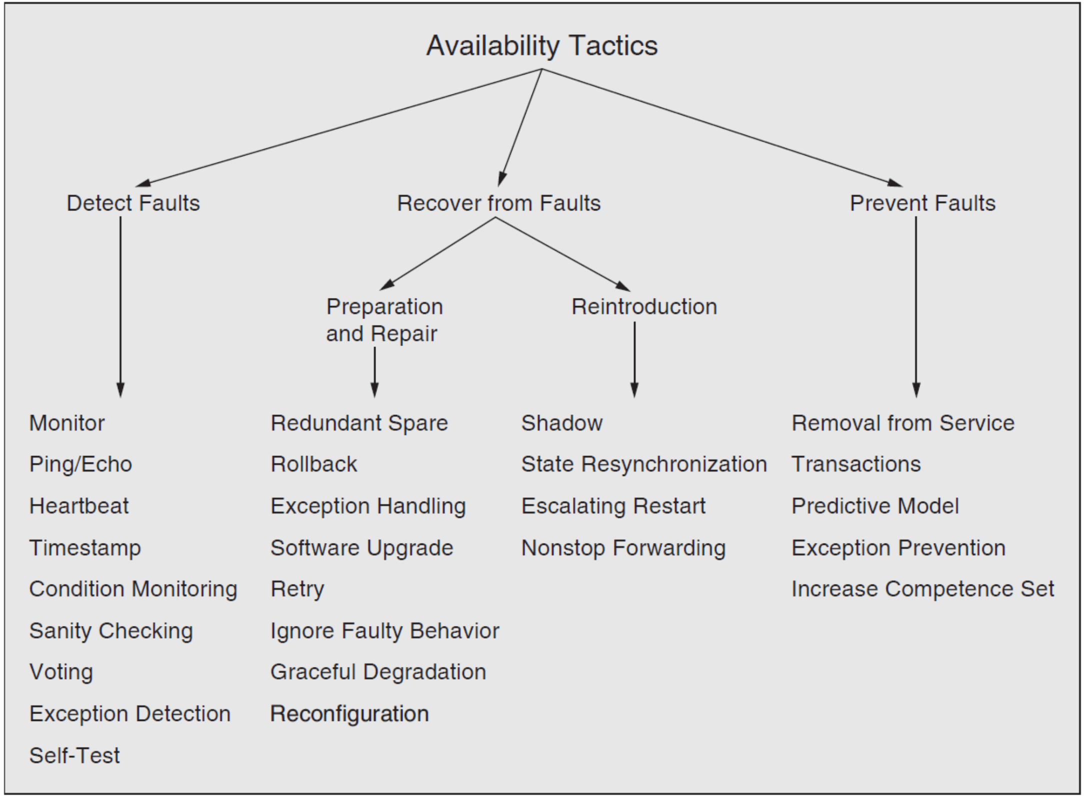
Design purpose
Primary functional requirements
Quality attribute scenarios
Constraints
Concerns
Step 1: Review inputs
Step 2: Establish iteration goal by selecting drivers
Step 3: Choose one or more elements of the system to refine
Step 4: Choose one or more design concepts that satisfy the selected drivers
Step 5: Instantiate architectural elements, allocate responsibilities, and define interfaces
Step 6: Sketch views and record design decisions
Step 7: Peform analysis of current design and review iteration goal and achievement of design purpose
(Refined) Software Architecture Design
From previous round of iterations or from existing system (brownfield development)
Iterate if necessary
Key
Driver
Architecture design
Process step
Precedence
Artifact flow

Architectural Tactics and Patterns

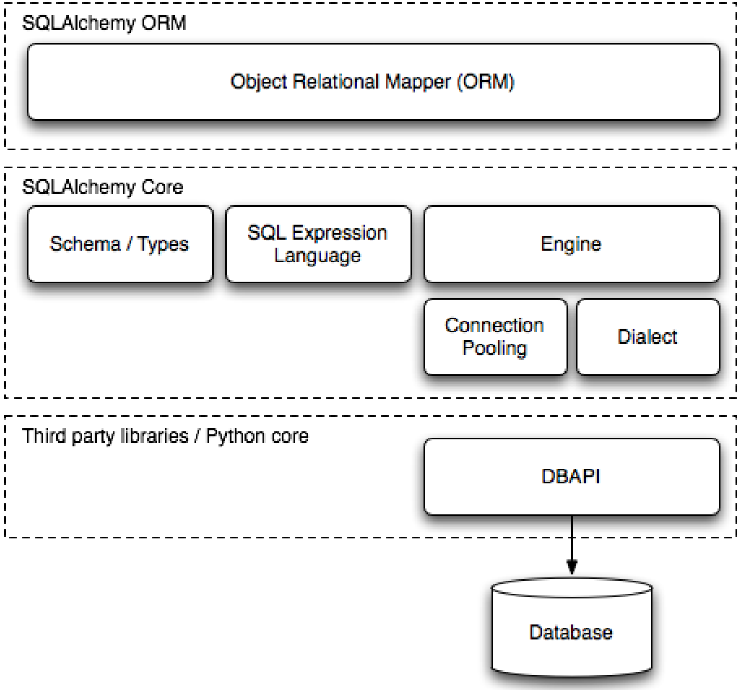
assign decision that influences a quality attribute



Availability Tactics



SQLAlchemy



https://aosabook.org/en/v2/sqlalchemy.html

ADD Exercise

DBMSs

SQLite

DuckDB

PostgreSQL

MariaDB

TiDB

We want to ensure these important systems’ reliability!

CockroachDB

MySQL™

Unit Testing: MySQL

zlob_print.test

```
--source include/have_debug.inc
--source include/have_innodb_max_16k.inc

set global innodb_compression_level = 0;
create table t1 (f1 int primary key, f2 longblob)
   row_format=compressed, engine=innodb;
set debug='+d,innodb_zlob_print';
insert into t1 values (1, repeat('+', 1048576));
set debug='-d,innodb_zlob_print';
select f1, right(f2, 40) from t1;
drop table t1;
set global innodb_compression_level = default;
```

zlob_print.result

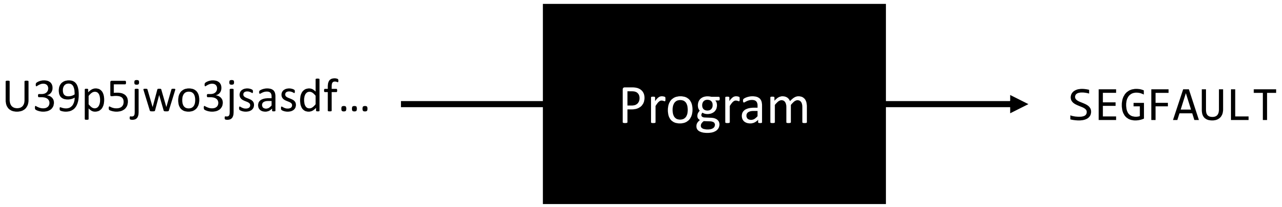
```
set global innodb_compression_level = 0;
create table t1 (f1 int primary key, f2 longblob)
row_format=compressed, engine=innodb;
set debug='+d,innodb_zlob_print';
insert into t1 values (1, repeat('+', 1048576));
set debug='-d,innodb_zlob_print';
select f1, right(f2, 40) from t1;
f1              right(f2, 40)
1
                +++++++++++++++++++++++++++++++++++++++
+
drop table t1;
                            ssion_level = default;
```

We will cover this in our testing week. But can we do better?

https://github.com/mysql/mysql-server/blob/8.0/mysql-test/suite/innodb/t/zlob_print.test

https://github.com/mysql/mysql-server/blob/8.0/mysql-test/suite/innodb/r/zlob_print.result

(Black-box) Fuzzing

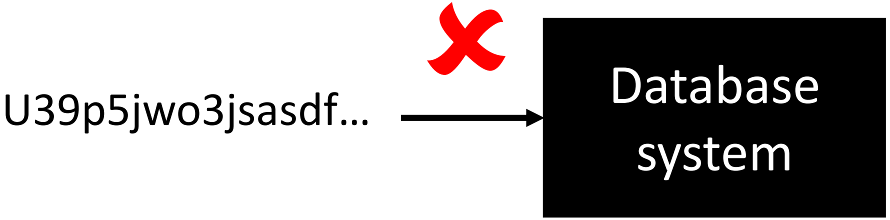


U39p5jwo3jsasdf…

Program

SEGFAULT

(Black-box) Fuzzing



U39p5jwo3jsasdf…

Database system

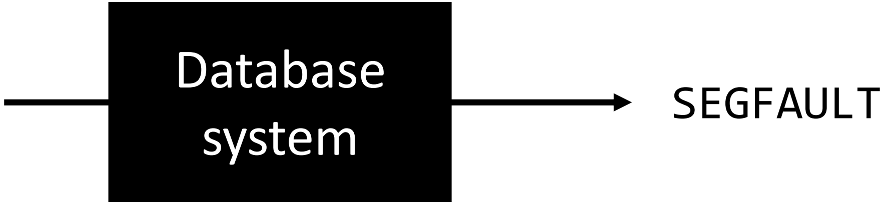
Such invalid input will be rejected without reaching any deep parts of the database system

SQL Generators: Expressions

```
import random

COLUMNS = ["age", "salary", "bonus", "dept_id"]
CONSTANTS = list(range(1, 101))
COMPARISONS = ["=", "<", ">", "<=", ">=", "<>"]
LOGICAL_OPS = ["AND", "OR"]
UNARY_OPS = ["NOT"]

def gen_sql_expr(depth=2):
    if depth == 0:
        if random.choice([True, False]):
            return str(random.choice(CONSTANTS))
        else:
            return random.choice(COLUMNS)

    choice = random.choice(["leaf", "unary", "binary", "comparison"])

    if choice == "leaf":
        return gen_sql_expr(0)

    elif choice == "unary":
        return f"{random.choice(UNARY_OPS)} ({gen_sql_expr(depth - 1)})"

    elif choice == "comparison":
        left = gen_sql_expr(depth - 1)
        right = gen_sql_expr(depth - 1)
        return f"({left} {random.choice(COMPARISONS)} {right})"

    else:  # binary logical
        left = gen_sql_expr(depth - 1)
        right = gen_sql_expr(depth - 1)
        return f"({left} {random.choice(LOGICAL_OPS)} {right})"

# Example usage
for _ in range(5):
    print(gen_sql_expr(depth=2))
```

(Black-box) Fuzzing

```
CREATE TABLE t0(c0 INT);
INSERT INTO t0 VALUES (1);
CREATE INDEX i0 ON t0(c0);
SELECT * FROM t0
WHERE c0 > 'Hello';
```



Syntactically valid SQL statements are more likely to trigger bugs in the database systems and might cover corner cases overlooked by manually-written test cases

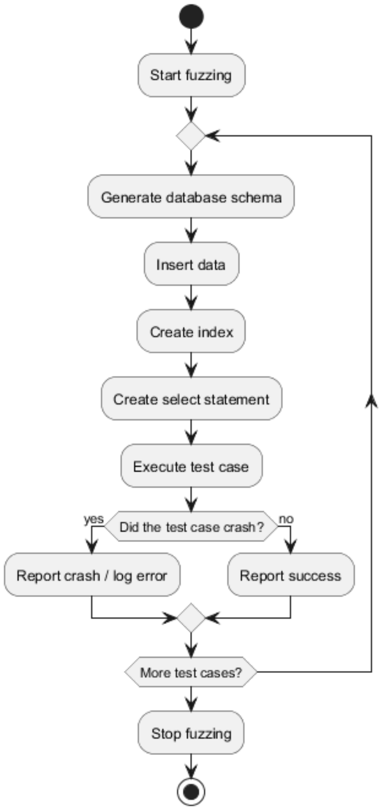
Illustrating Test Case Generation

How can we use UML diagram to illustrate the process of generating, executing, and validating a SQL test case? What is the appropriate diagram type?

```sql
CREATE TABLE t0(c0 INT);
INSERT INTO t0 VALUES (1);
CREATE INDEX i0 ON t0(c0);
SELECT * FROM t0
WHERE c0 > 'Hello';
```

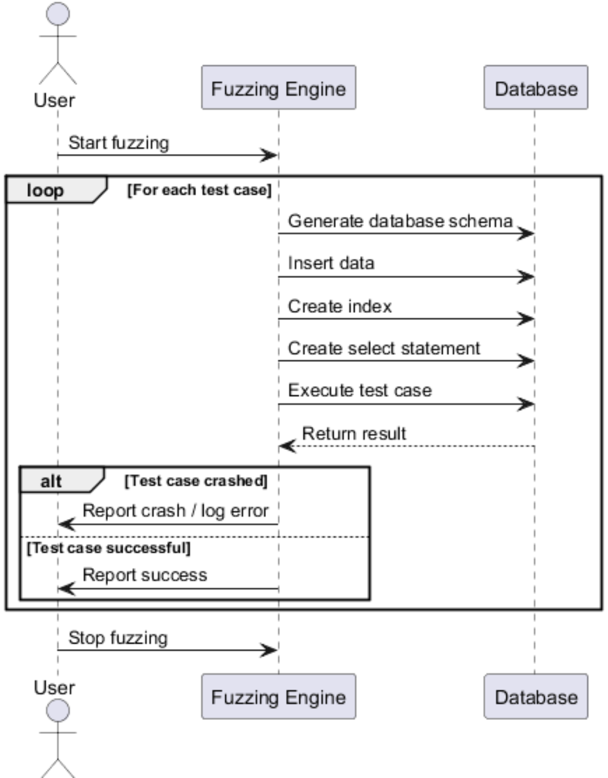
https://pollev.com/manuelrigger

Illustrating Test Case Generation



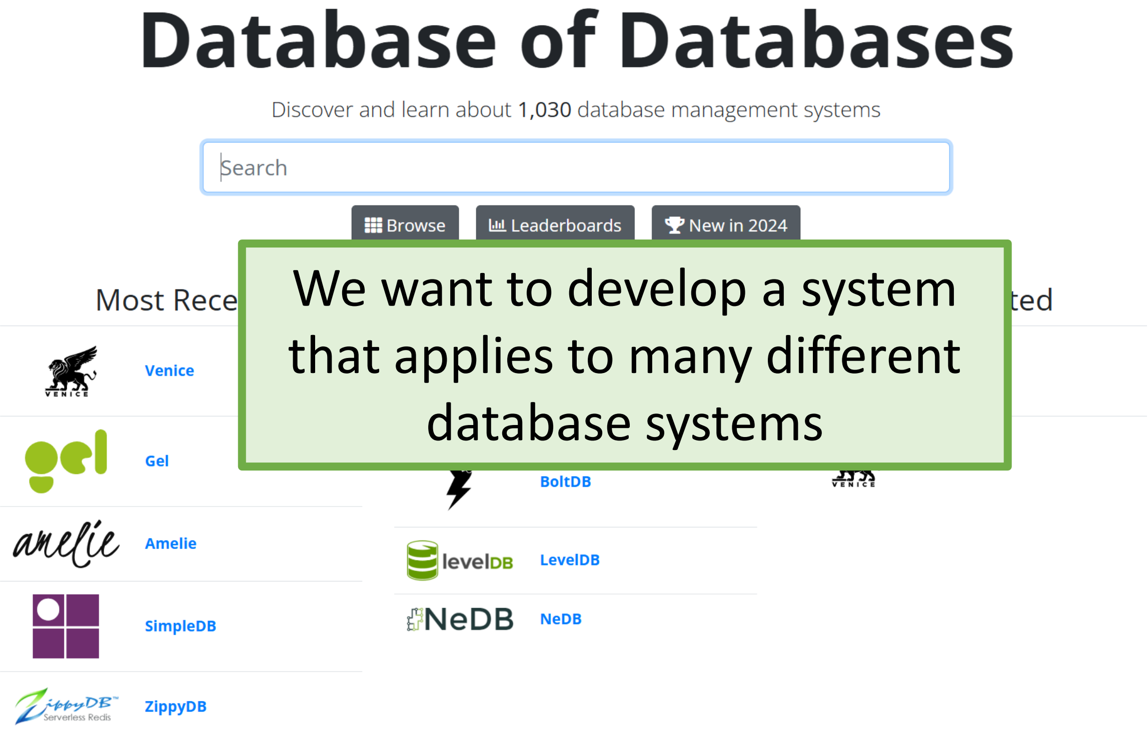
- Good for illustrating a workflow
- Easier to represent branching and conditions

Illustrating Test Case Generation



- Focused on actors/objects
- Makes order of sequences clear

Many DBMSs Exist!



Database of Databases

Discover and learn about **1,030** database management systems

Search

Most Rece

ted

Venice

Gel

Amelie

SimpleDB

ZippyDB

BoltDB

LevelDB

NeDB

We want to develop a system that applies to many different database systems

SQL Dialects

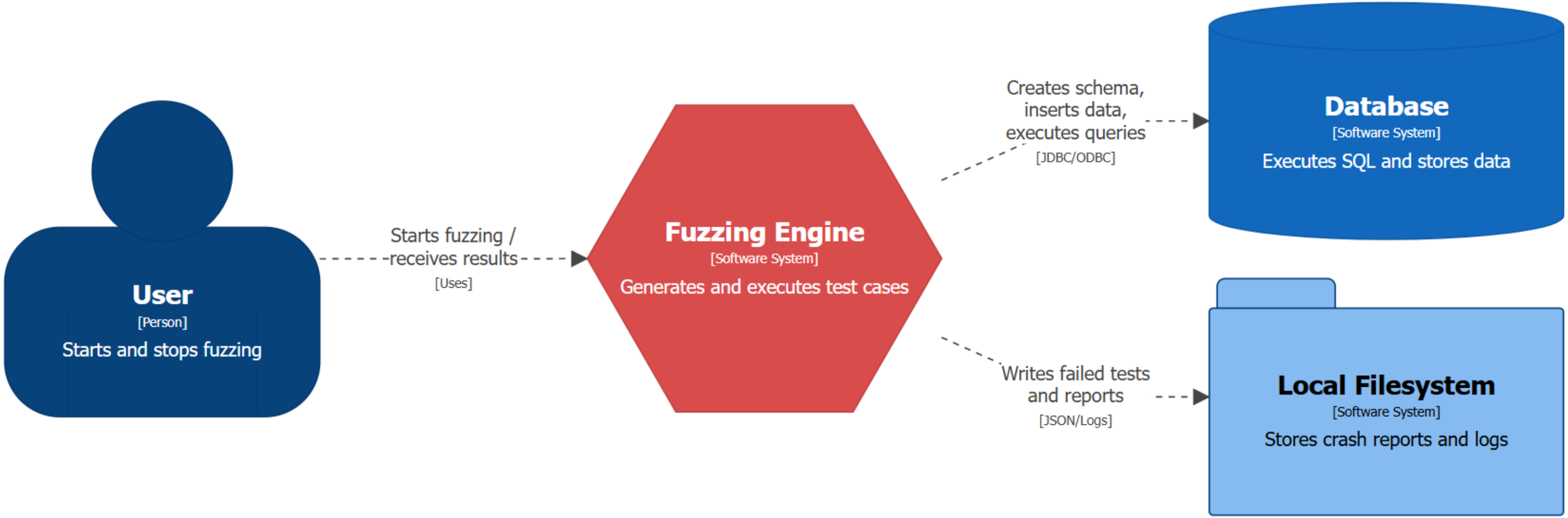
The SQL Dialect Problem

A multitude of SQL dialects exist that differ in terms of syntax and semantics

C4 Context Diagram

How can we visualize the system and its context as a C4 context diagram?

C4 Context Diagram



User
[Person]
Starts and stops fuzzing

Starts fuzzing / receives results
[Uses]

Fuzzing Engine
[Software System]
Generates and executes test cases

Creates schema, inserts data, executes queries
[JDBC/ODBC]

Database
[Software System]
Executes SQL and stores data

Writes failed tests and reports
[JSON/Logs]

Local Filesystem
[Software System]
Stores crash reports and logs

Quality Attributes?

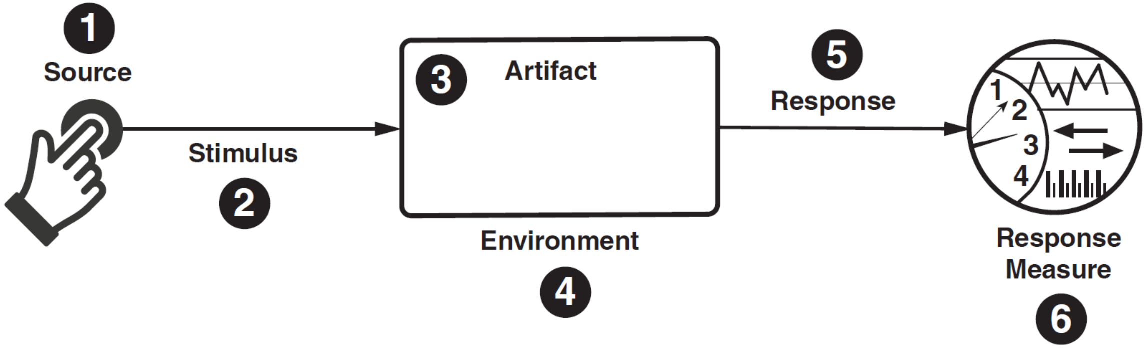
What quality attributes do you think are most important?

https://pollev.com/manuelrigger

Quality Attributes!

- **Scalability:** fully utilize the CPU resources
- **Performance**: the higher the test-case throughput, the quicker and more bugs we can find
- **Modifiability/Extensibility**: we want to easily add support for new SQL dialects/database systems
- **Effectiveness**: the test inputs should be effective
- **Reproducibility**: can replay test-cases and sequences

Performance



1. Source: User of the fuzzer engine
2. Stimulus: executes engine to continuously generate and execute test inputs on a server-client target database system using default options
3. Artifact: test-case generation pipeline and execution scheduler
4. Environment: normal operation on a single, dedicated machine with 64 CPU cores and 512 GB of RAM, with the database system running on the same machine
5. Response: the fuzzer generates and executes test inputs with minimal per-test overhead, fully utilizing the hardware resources
6. Response Measure: ???

Performance

What could be possible response measures?

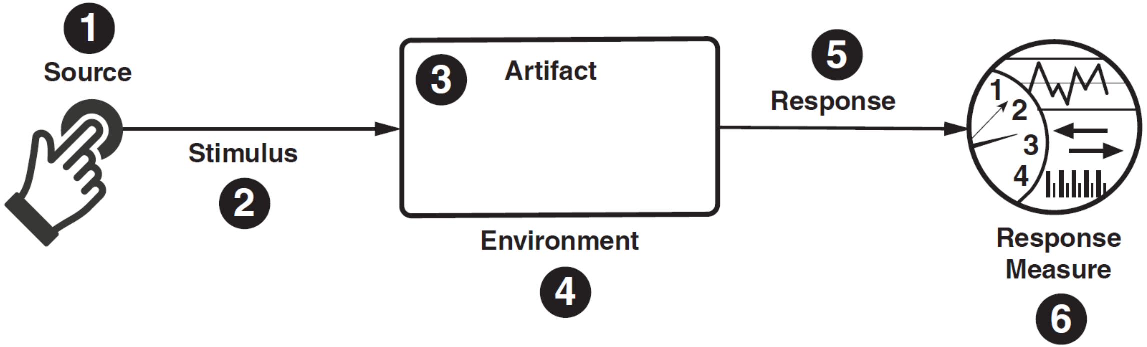
https://pollev.com/manuelrigger

Performance: Response Measures

- Test cases per second (per CPU core/over all CPU cores)
- Latency (≤ 50 ms per test case)
- CPU utilization
- I/O throughput metrics

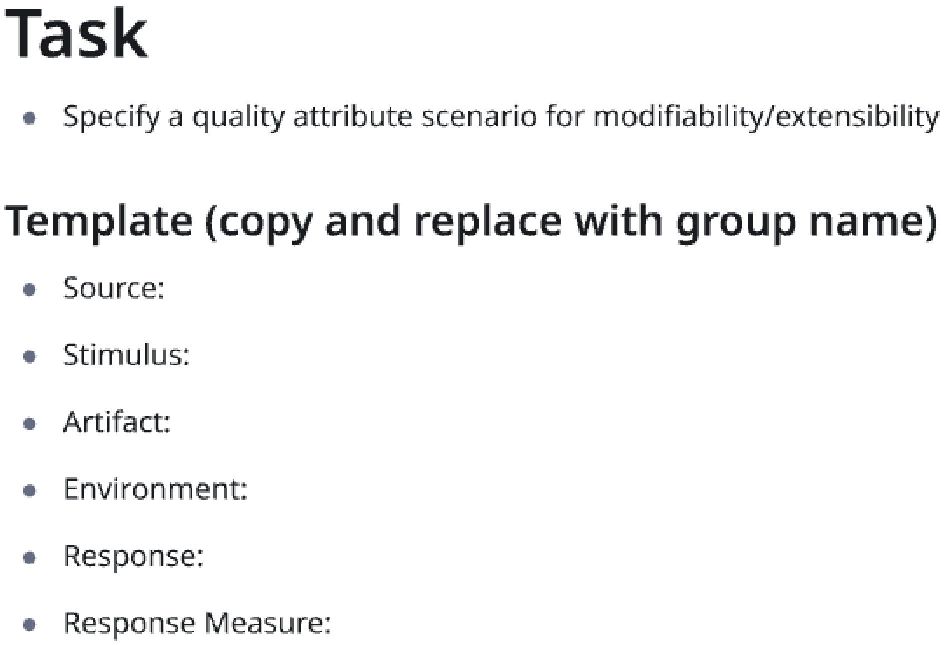
> *“Generate a Python program that connects to a SQLite systems and repeatedly generates a fixed sequence of SQL statements to create a table, insert 30 rows, create an index, and execute a SELECT statement on it. The program should output the throughput of the executions per second after running the program for 5 minutes.”*

Extensibility



1. Source:
2. Stimulus:
3. Artifact:
4. Environment:
5. Response:
6. Response Measure:

Quality Attributes



Task

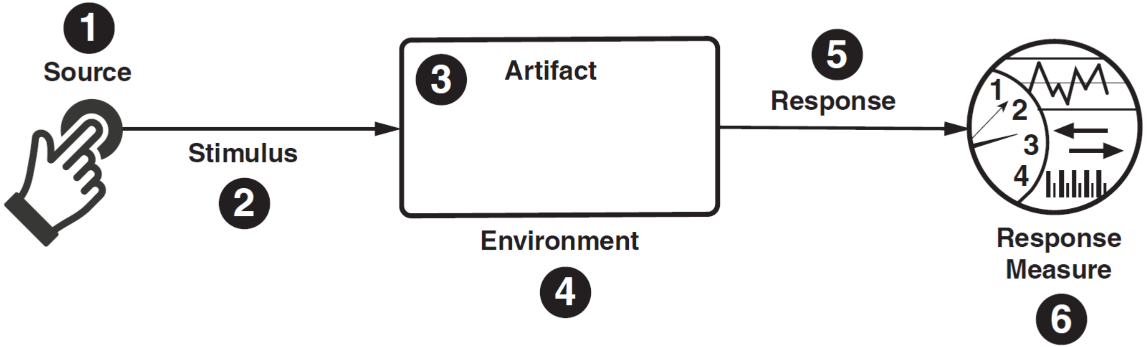
- Specify a quality attribute scenario for modifiability/extensibility

Template (copy and replace with group name)

- Source:
- Stimulus:
- Artifact:
- Environment:
- Response:
- Response Measure:

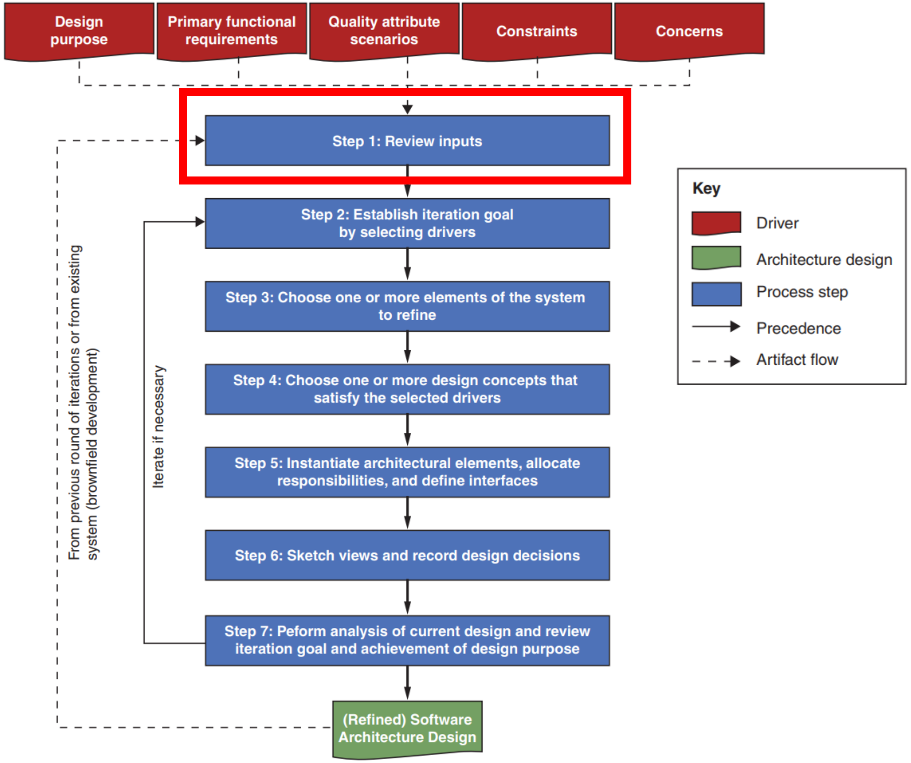
https://miro.com/app/board/uXjVGH3ZE1A=/?share_link_id=780348082206

Extensibility



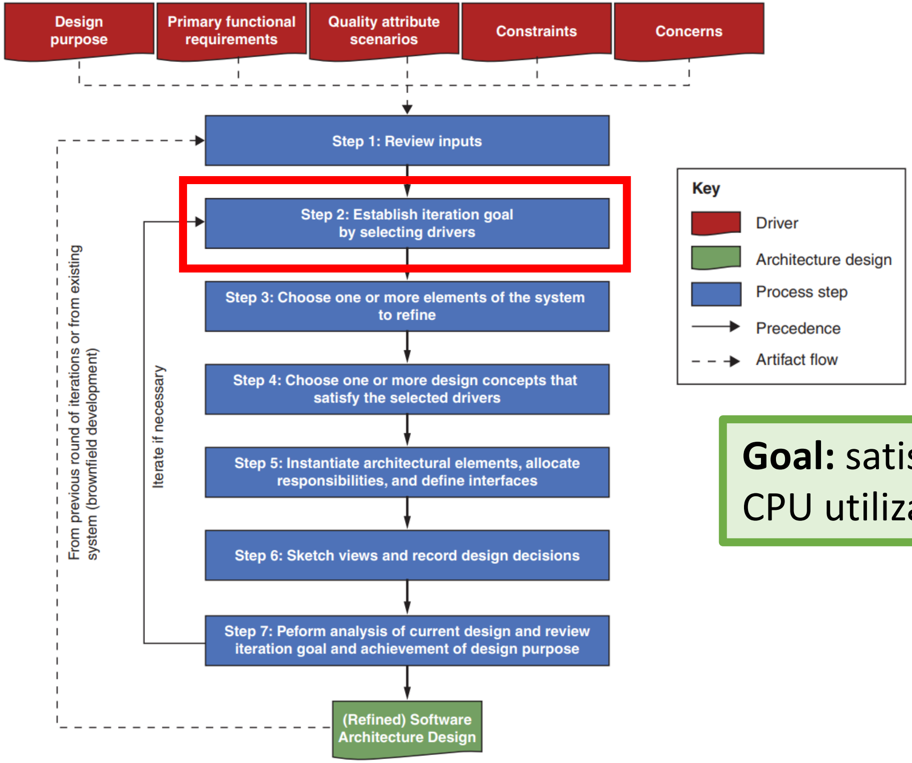
1. Source: fuzzer developer
2. Stimulus: wants to add a new SQL dialect
3. Artifact: fuzzing engine
4. (Environment: during normal development)
5. Response: can add the SQL dialect by extending the fuzzing engine
6. Response Measure: required code changes are restricted to additions only, no changes to the commonly used fuzzer engine core or other dialect implementations are required, and additions are encapsulated

Attribute-Driven Design (ADD)


Design purpose
Primary functional requirements
Quality attribute scenarios
Constraints
Concerns
Step 1: Review inputs
Step 2: Establish iteration goal by selecting drivers
Step 3: Choose one or more elements of the system to refine
Step 4: Choose one or more design concepts that satisfy the selected drivers
Step 5: Instantiate architectural elements, allocate responsibilities, and define interfaces
Step 6: Sketch views and record design decisions
Step 7: Peform analysis of current design and review iteration goal and achievement of design purpose
(Refined) Software Architecture Design
From previous round of iterations or from existing system (brownfield development)
Iterate if necessary
Key
Driver
Architecture design
Process step
Precedence
Artifact flow

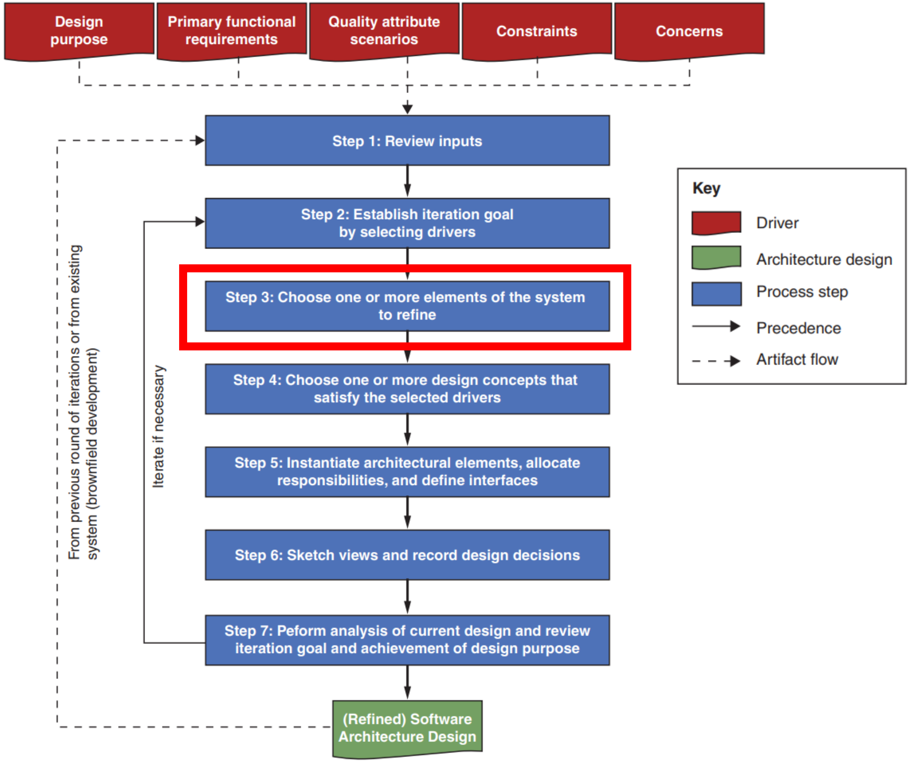
- **The purpose of the design round**: initial design
- **Primary functional requirements**: generate test inputs efficiently
- **Primary quality attribute scenarios**
- **Constraints and concerns**

Attribute-Driven Design (ADD)


Design purpose
Primary functional requirements
Quality attribute scenarios
Constraints
Concerns
Step 1: Review inputs
Step 2: Establish iteration goal by selecting drivers
Step 3: Choose one or more elements of the system to refine
Step 4: Choose one or more design concepts that satisfy the selected drivers
Step 5: Instantiate architectural elements, allocate responsibilities, and define interfaces
Step 6: Sketch views and record design decisions
Step 7: Peform analysis of current design and review iteration goal and achievement of design purpose
(Refined) Software Architecture Design
From previous round of iterations or from existing system (brownfield development)
Iterate if necessary
Key
Driver
Architecture design
Process step
Precedence
Artifact flow

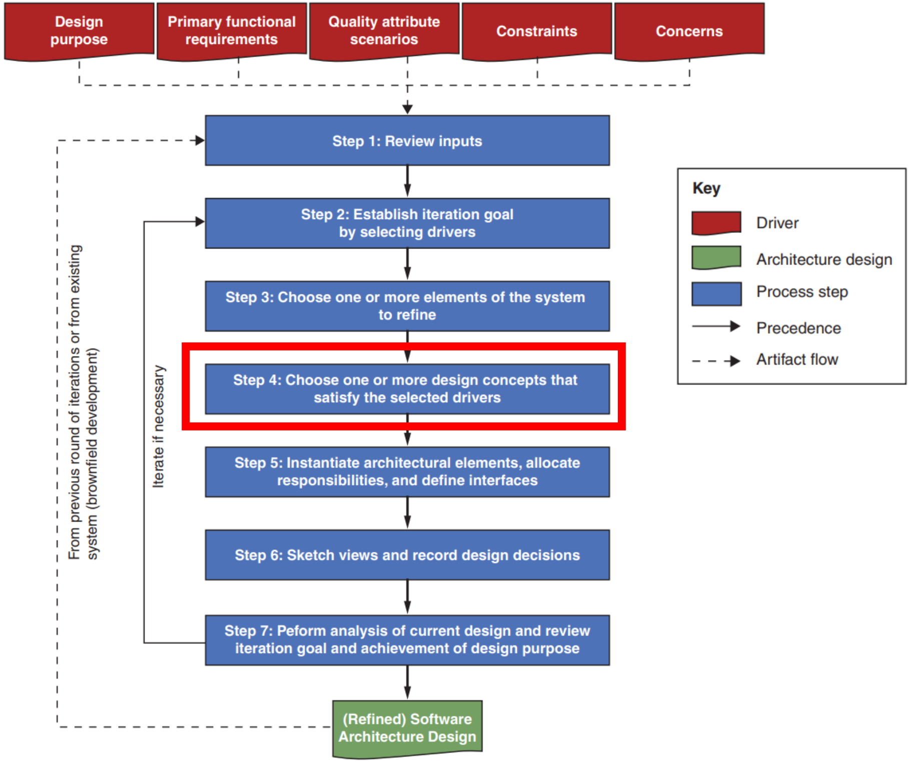
Goal: satisfy scalability scenario (e.g., > 80% CPU utilization on each CPU)

Attribute-Driven Design (ADD)

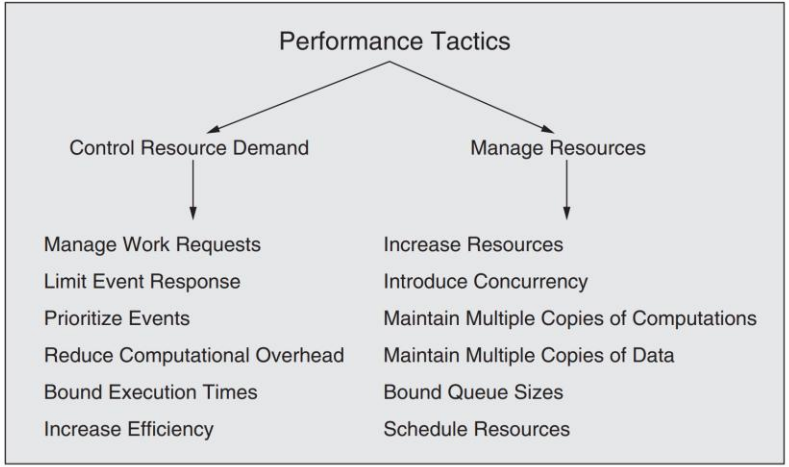

Design purpose
Primary functional requirements
Quality attribute scenarios
Constraints
Concerns
Step 1: Review inputs
Step 2: Establish iteration goal by selecting drivers
Step 3: Choose one or more elements of the system to refine
Step 4: Choose one or more design concepts that satisfy the selected drivers
Step 5: Instantiate architectural elements, allocate responsibilities, and define interfaces
Step 6: Sketch views and record design decisions
Step 7: Peform analysis of current design and review iteration goal and achievement of design purpose
(Refined) Software Architecture Design
From previous round of iterations or from existing system (brownfield development)
Iterate if necessary
Key
Driver
Architecture design
Process step
Precedence
Artifact flow

Greenfield project: we choose a top-down approach by decomposing the overall system

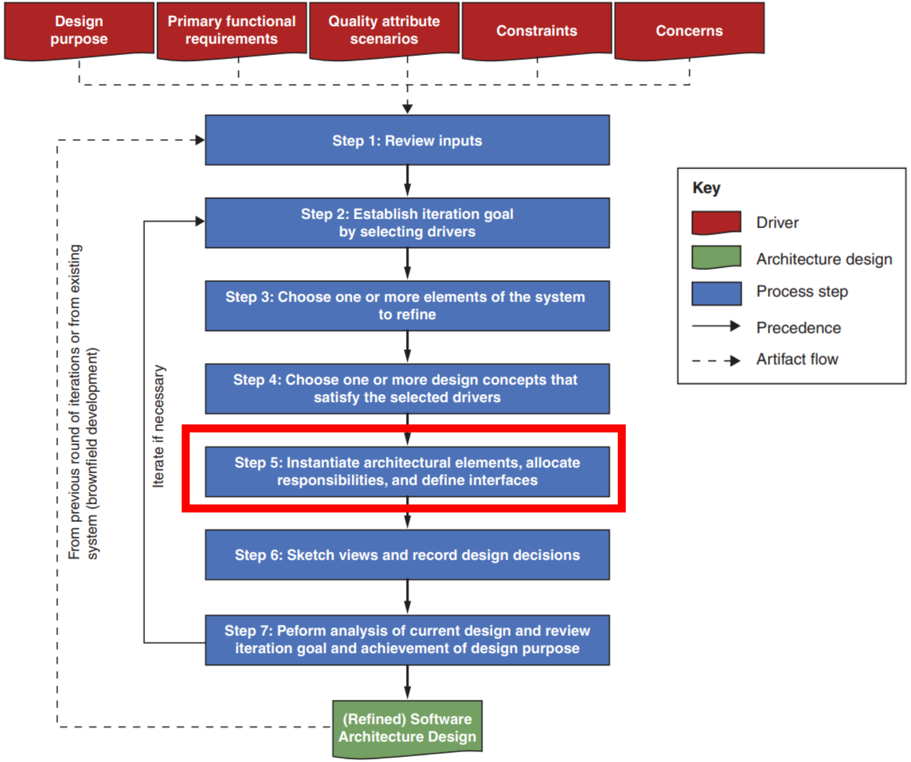
Attribute-Driven Design (ADD)


Design purpose
Primary functional requirements
Quality attribute scenarios
Constraints
Concerns
Step 1: Review inputs
Step 2: Establish iteration goal by selecting drivers
Step 3: Choose one or more elements of the system to refine
Step 4: Choose one or more design concepts that satisfy the selected drivers
Step 5: Instantiate architectural elements, allocate responsibilities, and define interfaces
Step 6: Sketch views and record design decisions
Step 7: Peform analysis of current design and review iteration goal and achievement of design purpose
(Refined) Software Architecture Design
From previous round of iterations or from existing system (brownfield development)
Iterate if necessary
Key
🟥 Driver
🟩 Architecture design
🟦 Process step
→ Precedence
- - → Artifact flow

Which tactics and/or patterns would be applicable?



Attribute-Driven Design (ADD)


Design purpose
Primary functional requirements
Quality attribute scenarios
Constraints
Concerns
Step 1: Review inputs
Step 2: Establish iteration goal by selecting drivers
Step 3: Choose one or more elements of the system to refine
Step 4: Choose one or more design concepts that satisfy the selected drivers
Step 5: Instantiate architectural elements, allocate responsibilities, and define interfaces
Step 6: Sketch views and record design decisions
Step 7: Peform analysis of current design and review iteration goal and achievement of design purpose
(Refined) Software Architecture Design
From previous round of iterations or from existing system (brownfield development)
Iterate if necessary
Key
Driver
Architecture design
Process step
Precedence
Artifact flow

In this class, we will only present a high-level design

Design A: Concurrent Fuzzer Threads

- Tactic: *Introduce Concurrency*
- Idea: each separate fuzzer thread generates test inputs separately, executes them, and inspects the results

Design B: Schedule Resources

- Tactic: *Schedule Resources*
- Idea: the fuzzer generates test inputs, and sends them to a pool of workers, each of which executes a test case, sends them to a separate database instances, and checks the result

Illustration

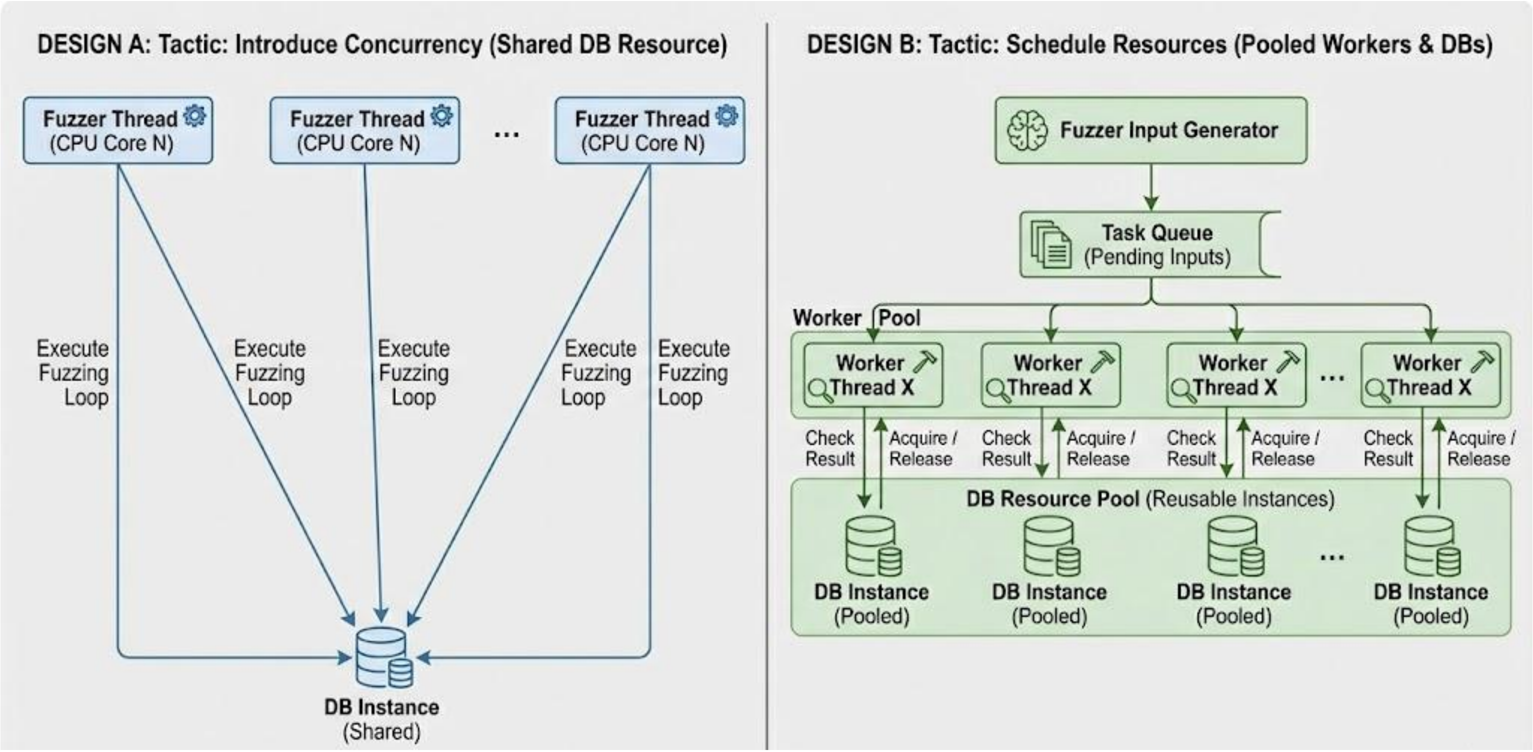


Image by Gemini

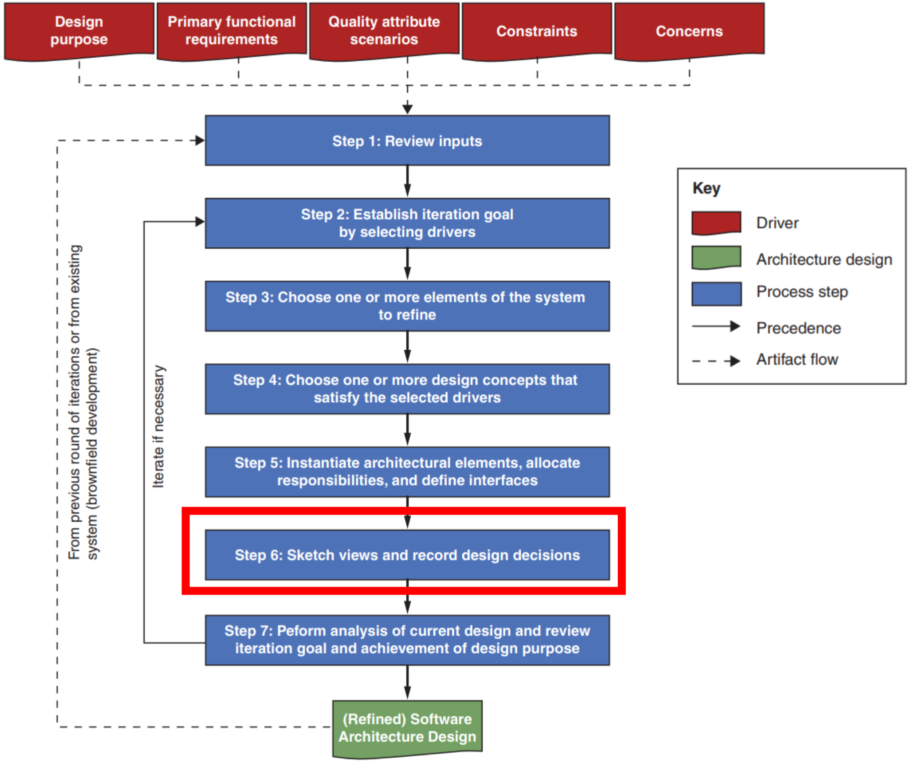
Discussion: Designs

Which design would work better? Why?

Discussion: Design A vs. Design B

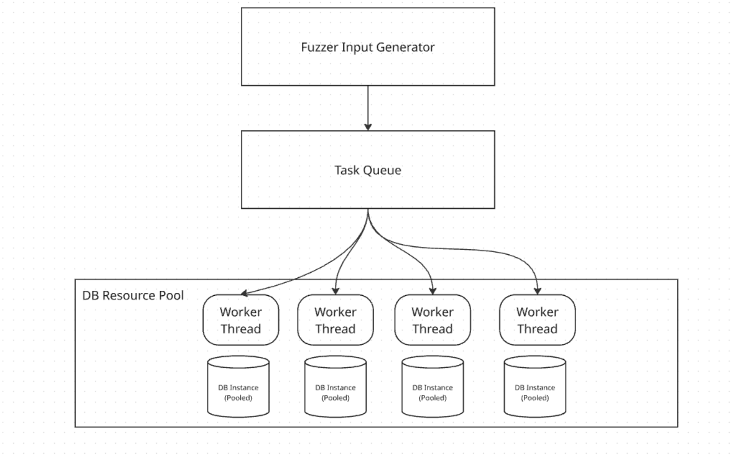
- Design A might be better if the generation and result inspection process is the bottleneck
- Design B might be better if the database system is the bottleneck
- Prototyping to evaluate!
  - Go for Design B

Attribute-Driven Design (ADD)



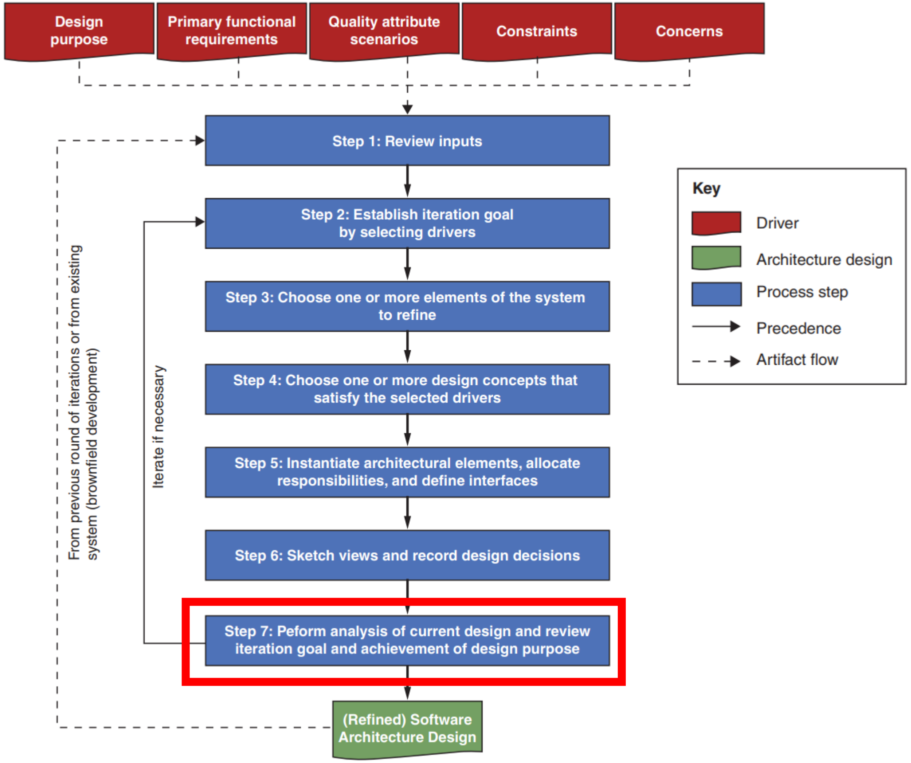
Design purpose
Primary functional requirements
Quality attribute scenarios
Constraints
Concerns
Step 1: Review inputs
Step 2: Establish iteration goal by selecting drivers
Step 3: Choose one or more elements of the system to refine
Step 4: Choose one or more design concepts that satisfy the selected drivers
Step 5: Instantiate architectural elements, allocate responsibilities, and define interfaces
Step 6: Sketch views and record design decisions
Step 7: Peform analysis of current design and review iteration goal and achievement of design purpose
(Refined) Software Architecture Design
From previous round of iterations or from existing system (brownfield development)
Iterate if necessary
Key
Driver
Architecture design
Process step
Precedence
Artifact flow

Sketch Views



- We only have a single view here, but it could also include multiple ones (*e.g.*, a decomposition view and concrete interfaces)
- Record the reason that motivated the design and the trade-offs considered

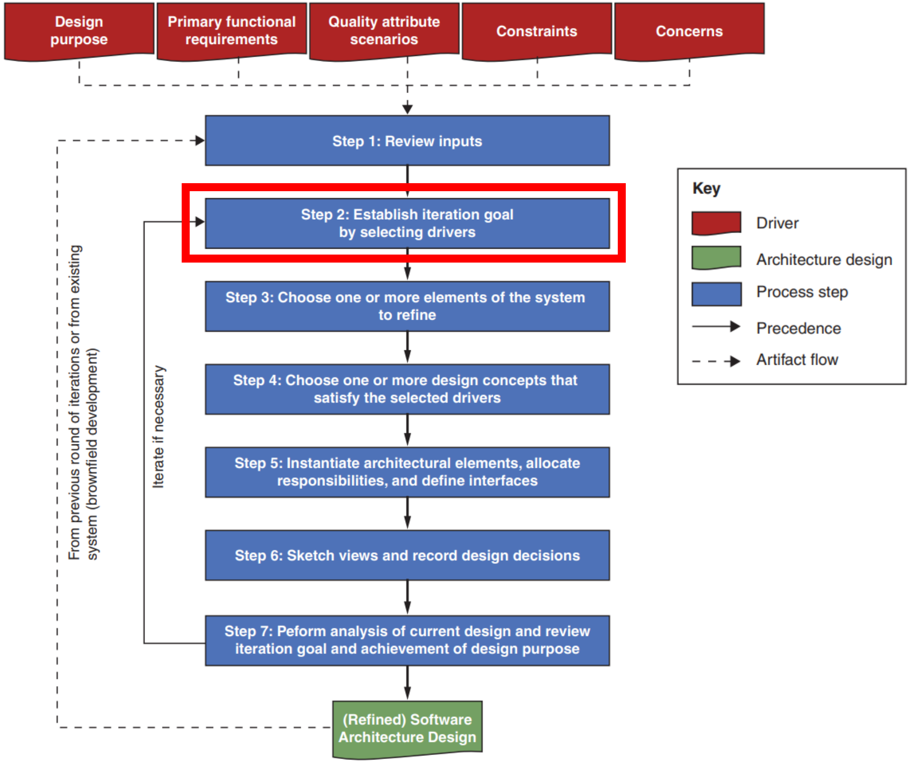
Attribute-Driven Design (ADD)


Design purpose
Primary functional requirements
Quality attribute scenarios
Constraints
Concerns
Step 1: Review inputs
Step 2: Establish iteration goal by selecting drivers
Step 3: Choose one or more elements of the system to refine
Step 4: Choose one or more design concepts that satisfy the selected drivers
Step 5: Instantiate architectural elements, allocate responsibilities, and define interfaces
Step 6: Sketch views and record design decisions
Step 7: Peform analysis of current design and review iteration goal and achievement of design purpose
(Refined) Software Architecture Design
From previous round of iterations or from existing system (brownfield development)
Iterate if necessary
Key
Driver
Architecture design
Process step
Precedence
Artifact flow

How to review?

- Yourself
- AI Chatbot
- Your colleagues
- Prototyping

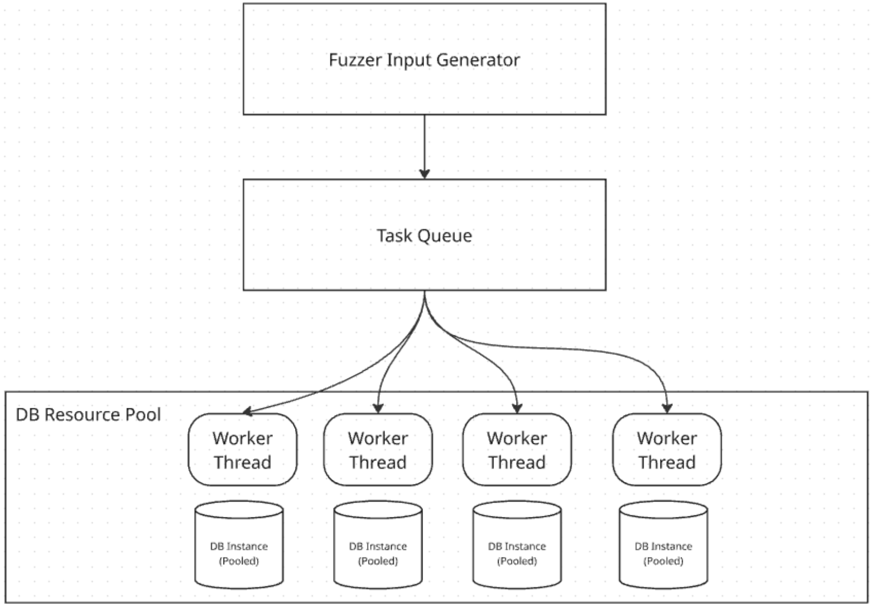
Attribute-Driven Design (ADD)


Design purpose
Primary functional requirements
Quality attribute scenarios
Constraints
Concerns
Step 1: Review inputs
Step 2: Establish iteration goal by selecting drivers
Step 3: Choose one or more elements of the system to refine
Step 4: Choose one or more design concepts that satisfy the selected drivers
Step 5: Instantiate architectural elements, allocate responsibilities, and define interfaces
Step 6: Sketch views and record design decisions
Step 7: Peform analysis of current design and review iteration goal and achievement of design purpose
(Refined) Software Architecture Design
From previous round of iterations or from existing system (brownfield development)
Iterate if necessary
Key
Driver
Architecture design
Process step
Precedence
Artifact flow

Next iteration goal: extensibility with respect to new SQL dialects

Course Exercise

Please copy and do not directly modify this figure!



Fuzzer Input Generator

Task Queue

DB Resource Pool

Worker Thread

Worker Thread

Worker Thread

Worker Thread

DB Instance (Pooled)

DB Instance (Pooled)

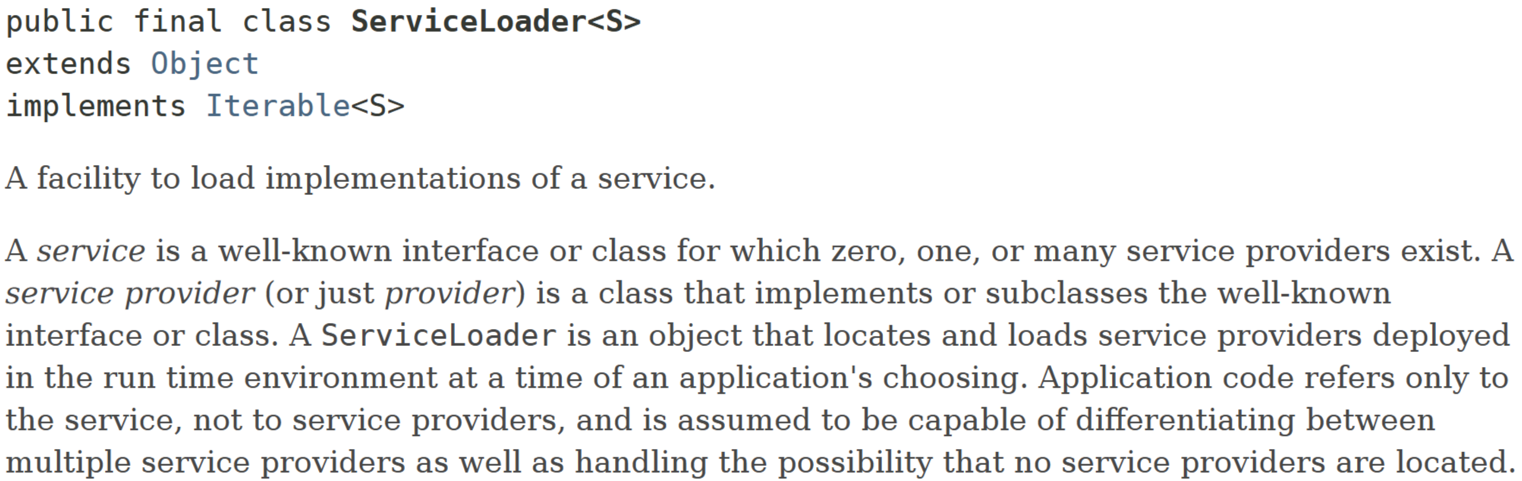
DB Instance (Pooled)

DB Instance (Pooled)

https://miro.com/app/board/uXjVGH3Q260=/?share_link_id=866544456203

Design A: Defer Binding

- Tactic: *Defer Binding*
- Idea: the fuzzer loads SQL dialect support plugins from a file/directory



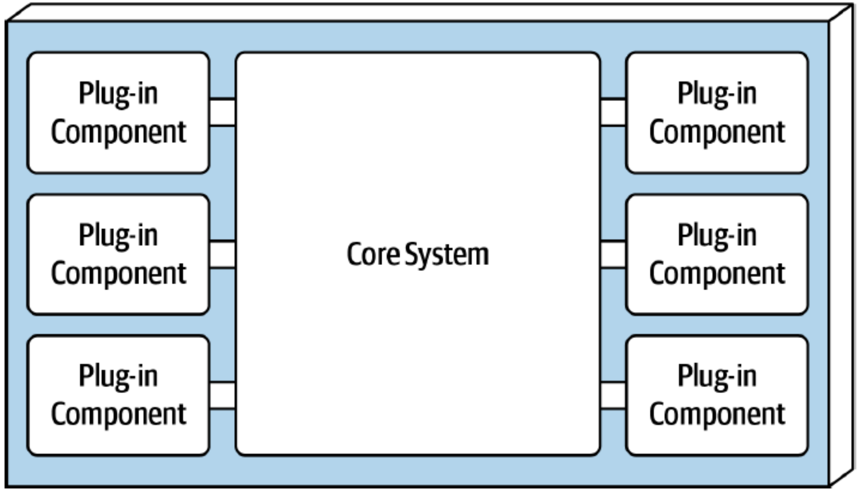
```
public final class ServiceLoader<S>
extends Object
implements Iterable<S>
```

A facility to load implementations of a service.

A *service* is a well-known interface or class for which zero, one, or many service providers exist. A *service provider* (or just *provider*) is a class that implements or subclasses the well-known interface or class. A `ServiceLoader` is an object that locates and loads service providers deployed in the run time environment at a time of an application's choosing. Application code refers only to the service, not to service providers, and is assumed to be capable of differentiating between multiple service providers as well as handling the possibility that no service providers are located.

Design A: Defer Binding

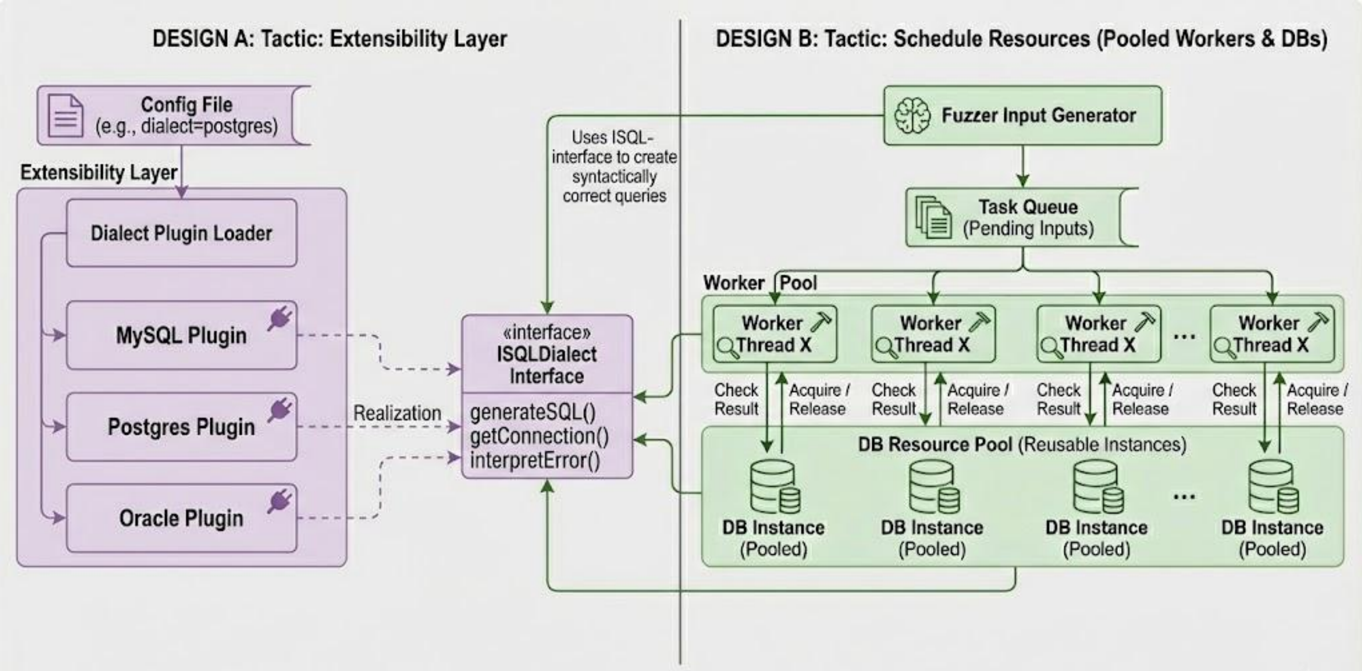
- In this case, this would basically correspond to a microkernel/plugin architecture



Design B: Abstract Services

- Tactic: *Abstract Services*
- Idea: the fuzzer engine and database pool are unaware of a specific dialect, but interact with an abstract service

Illustration: Combine Design A and B



DESIGN A: Tactic: Extensibility Layer

DESIGN B: Tactic: Schedule Resources (Pooled Workers & DBs)

Config File (e.g., dialect=postgres)

Extensibility Layer

Dialect Plugin Loader

MySQL Plugin

Postgres Plugin

Oracle Plugin

Realization

«interface» ISQLDialect Interface

generateSQL()
getConnection()
interpretError()

Uses ISQL-interface to create syntactically correct queries

Fuzzer Input Generator

Task Queue (Pending Inputs)

Worker Pool

Worker Thread X

Worker Thread X

Worker Thread X

...

Worker Thread X

Check Result

Acquire / Release

DB Resource Pool (Reusable Instances)

DB Instance (Pooled)

DB Instance (Pooled)

DB Instance (Pooled)

...

DB Instance (Pooled)

Idea: Core components (Generator, Workers, Pool) depend only on an abstract `ISQLDialect` interface. A Plugin Loader dynamically injects the concrete implementation (e.g., Postgres) at runtime based on configuration, achieving high extensibility.