SOA & API Testing

Shreyata Sugandhi

Corporate Trainer & Consultant

shreyata@gtsedutech.com

https://www.linkedin.com/in/shreyatasugandhi/

Agenda

SOA

<u>What is SOA?</u>

- Service-oriented architecture (SOA) is an approach used to create an architecture based upon the use of services.

- **In SOA**, application components provide services to other components via a communications protocol, typically over a network.

  Ex: Web Services, BPM, EAI

SOA

Why SOA?

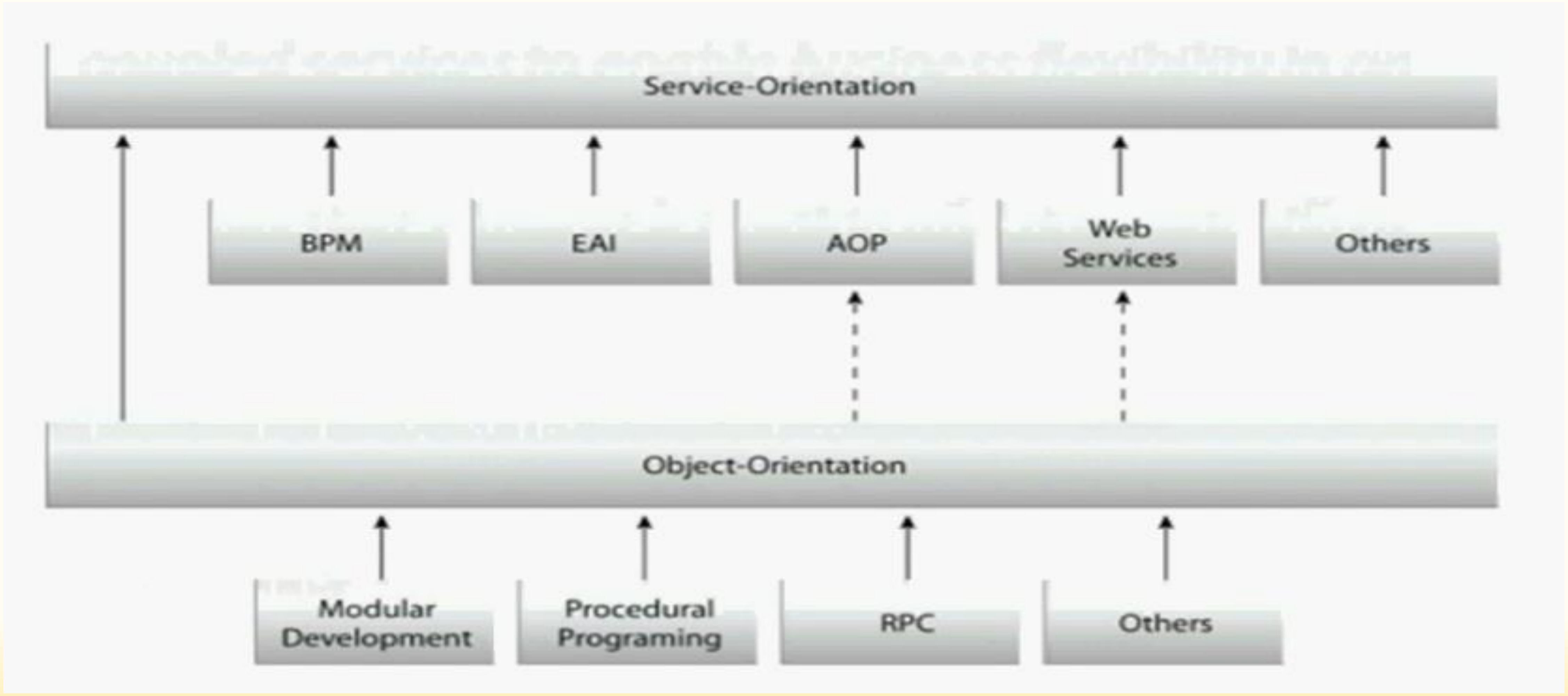
- Load balancing
- Event handling
- Supports homogeneous and heterogeneous technology
- ESB
- Collection of loosely coupled services

SOA

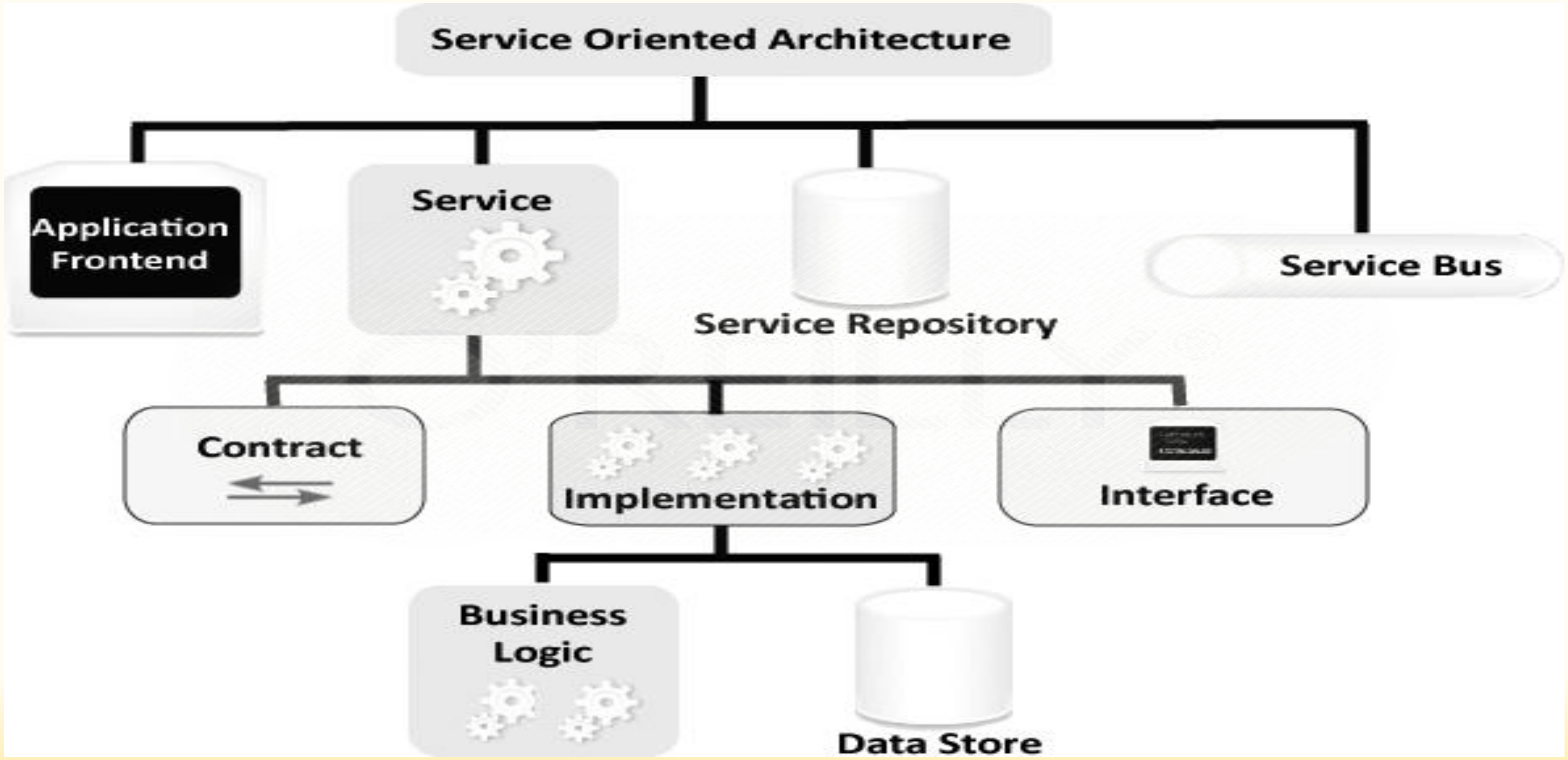
Benefits:

- Reducing Integration Expense
- Increasing Asset Reuse
- Increasing Business Agility
- Reduction Of Business Risk

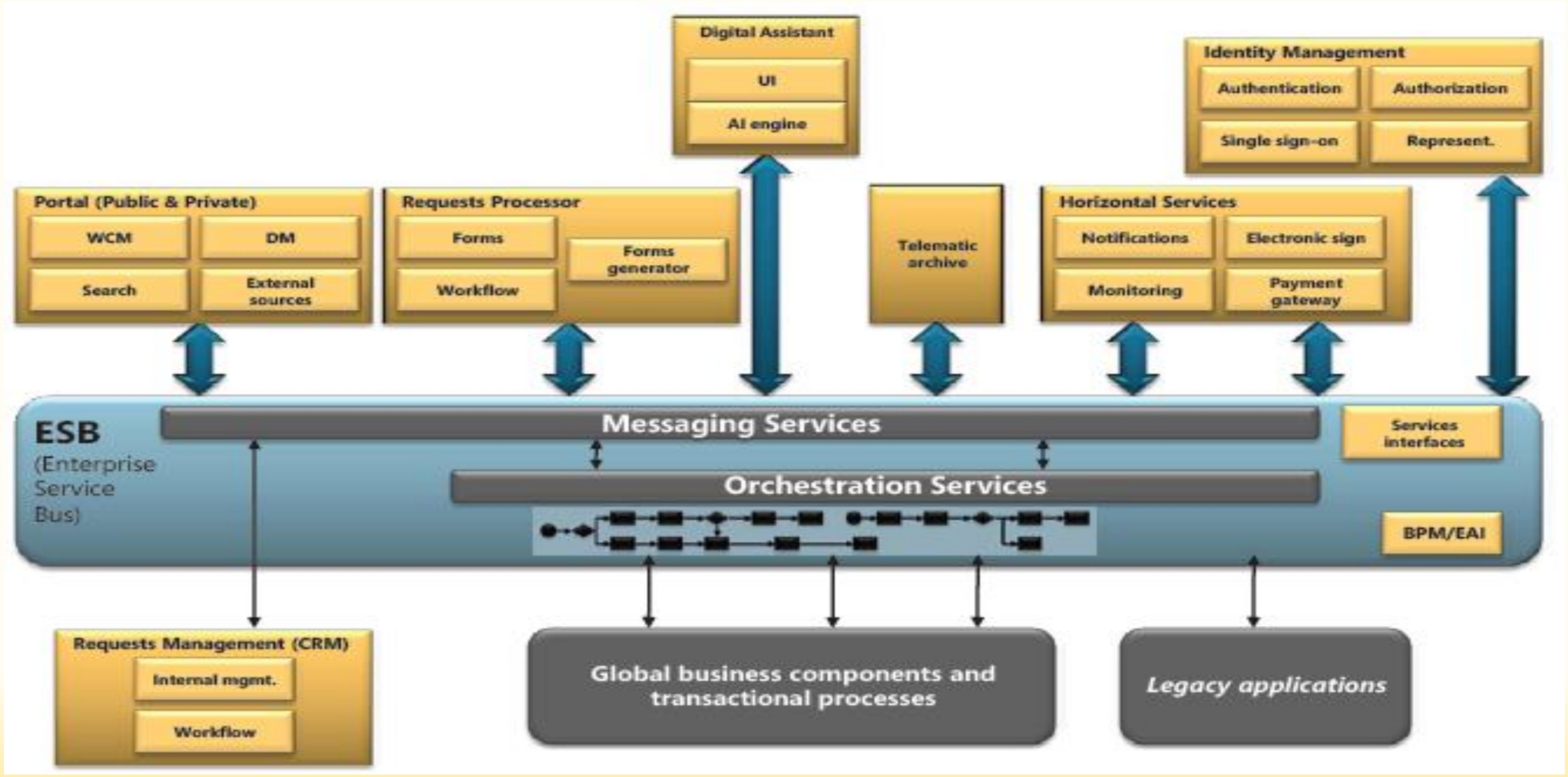
Evolution of SOA



SOA Orientation

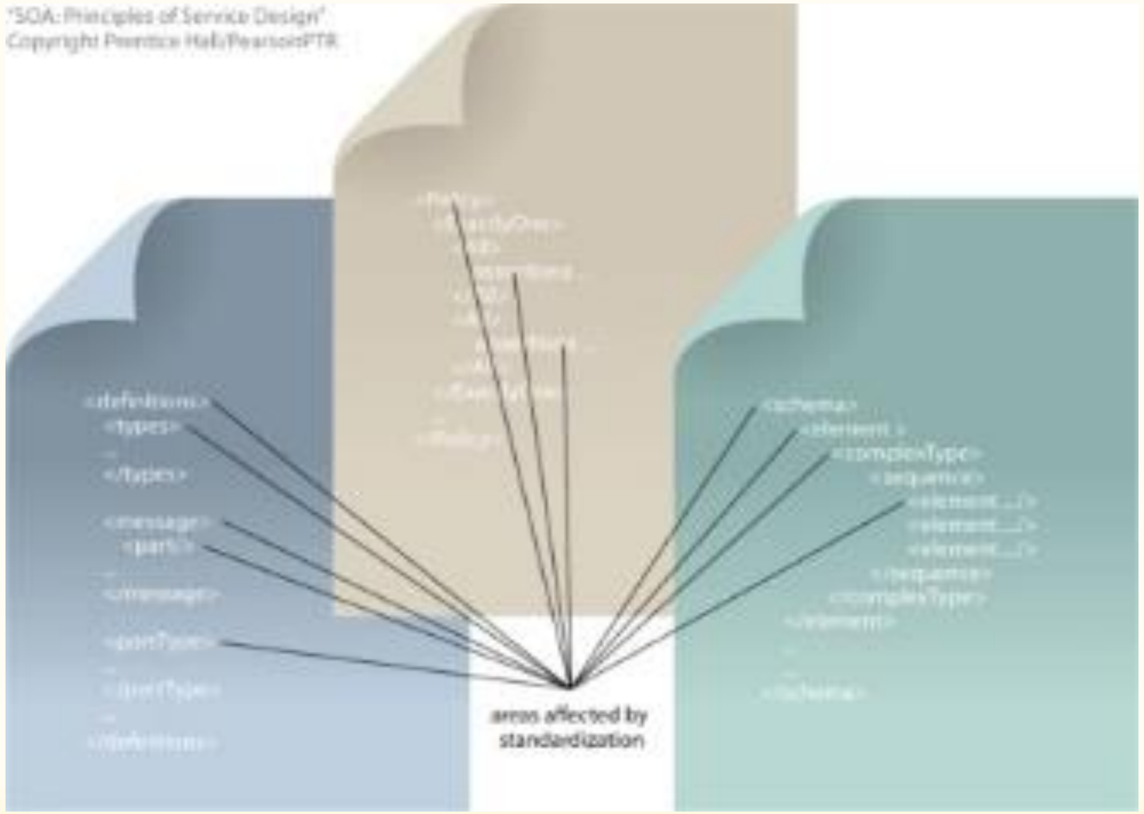


Architecture



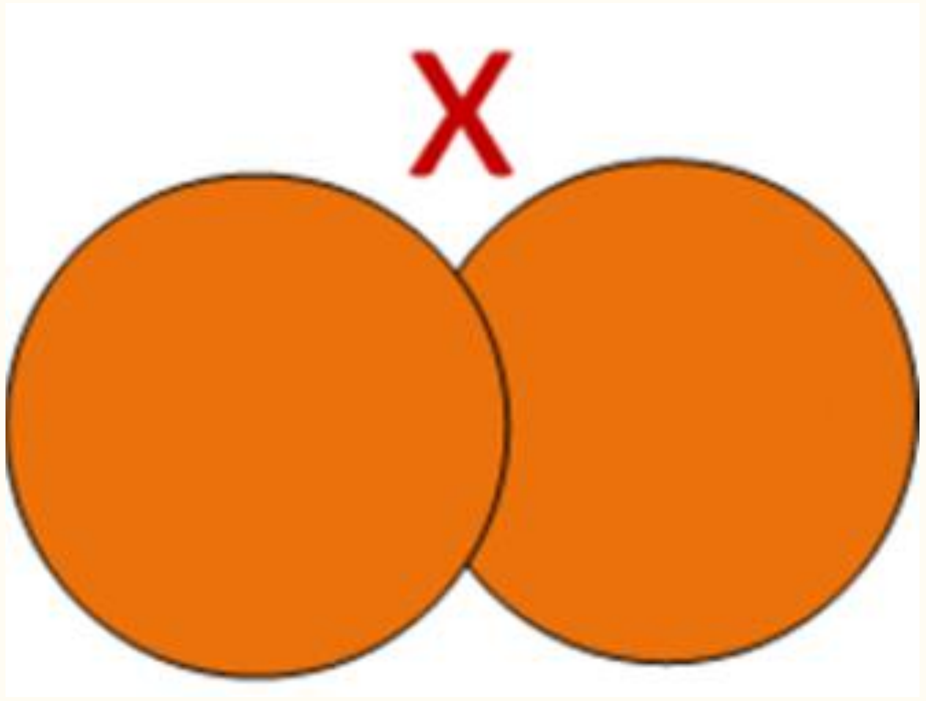
Principles of SOA

- Standardized Service Contract -- Services adhere to a service-description.



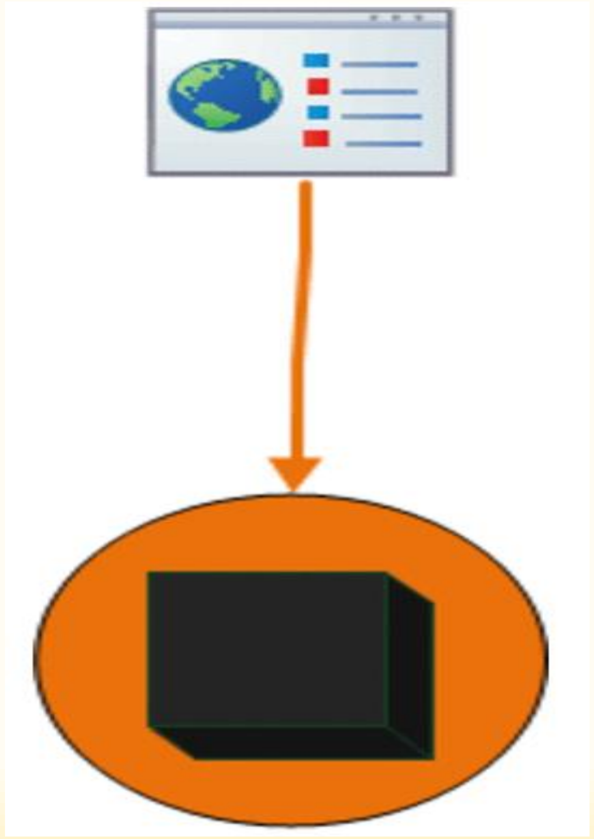
Principles of SOA

- Loose Coupling -- Services minimize dependencies on each other.



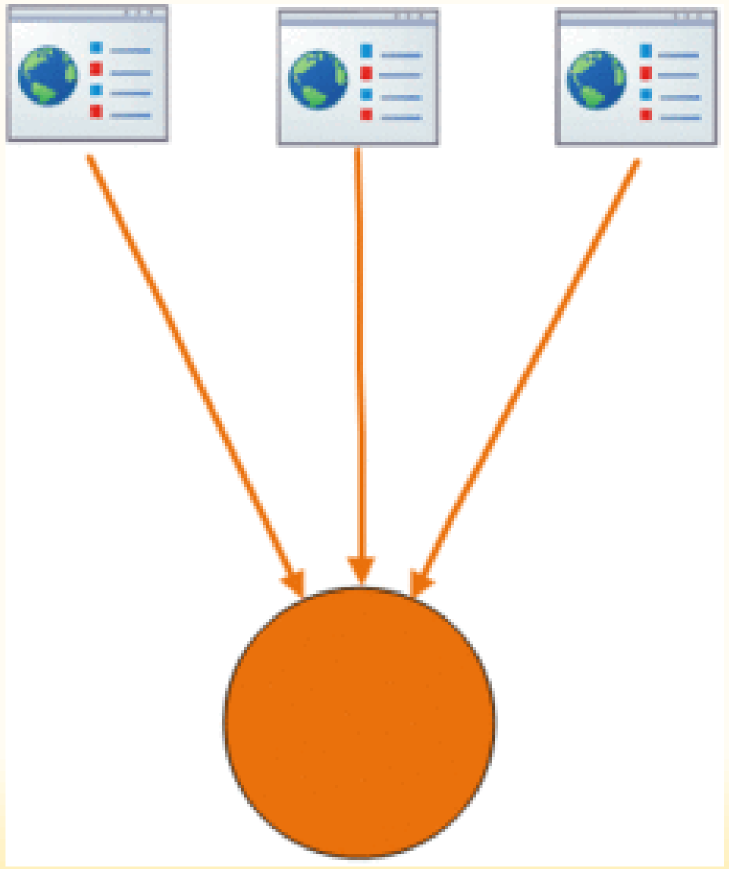
Principles of SOA

- Service Abstraction -- Services hide the logic they encapsulate from the outside world.



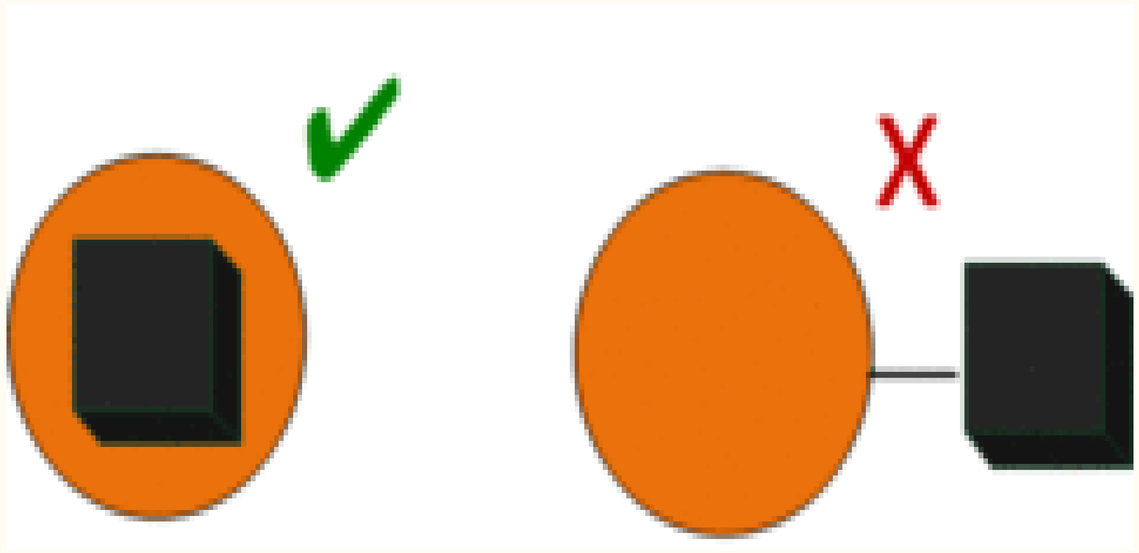
Principles of SOA

- Service Reusability -- Logic is divided into services with the intent of maximizing reuse.



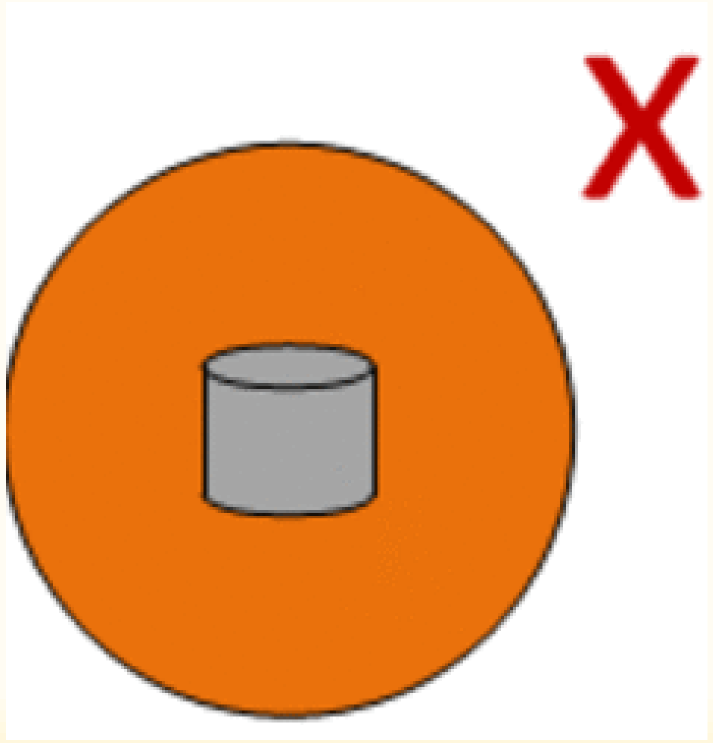
Principles of SOA

- Service Autonomy -- Services should have control over the logic they encapsulate.



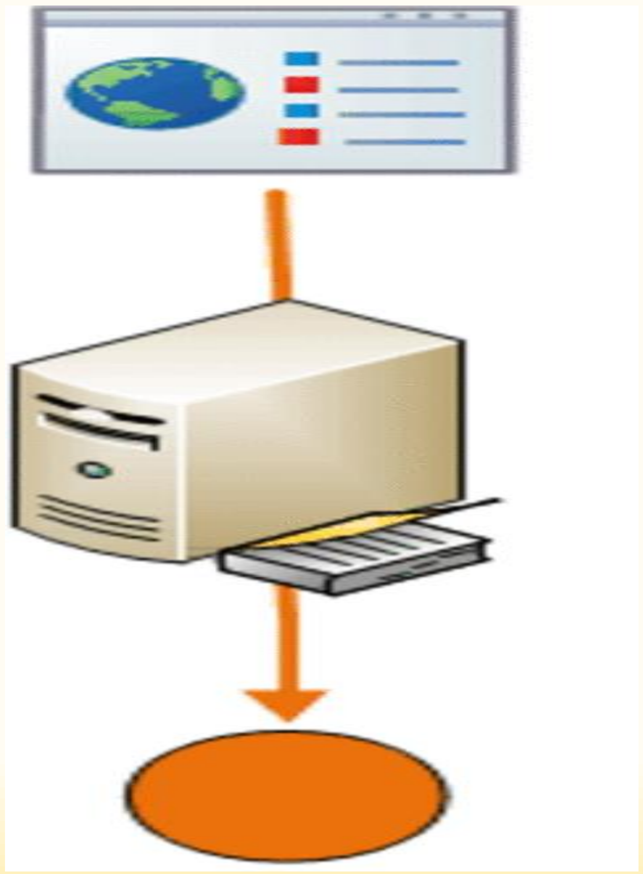
Principles of SOA

- Service Statelessness -- Ideally, services should be stateless.



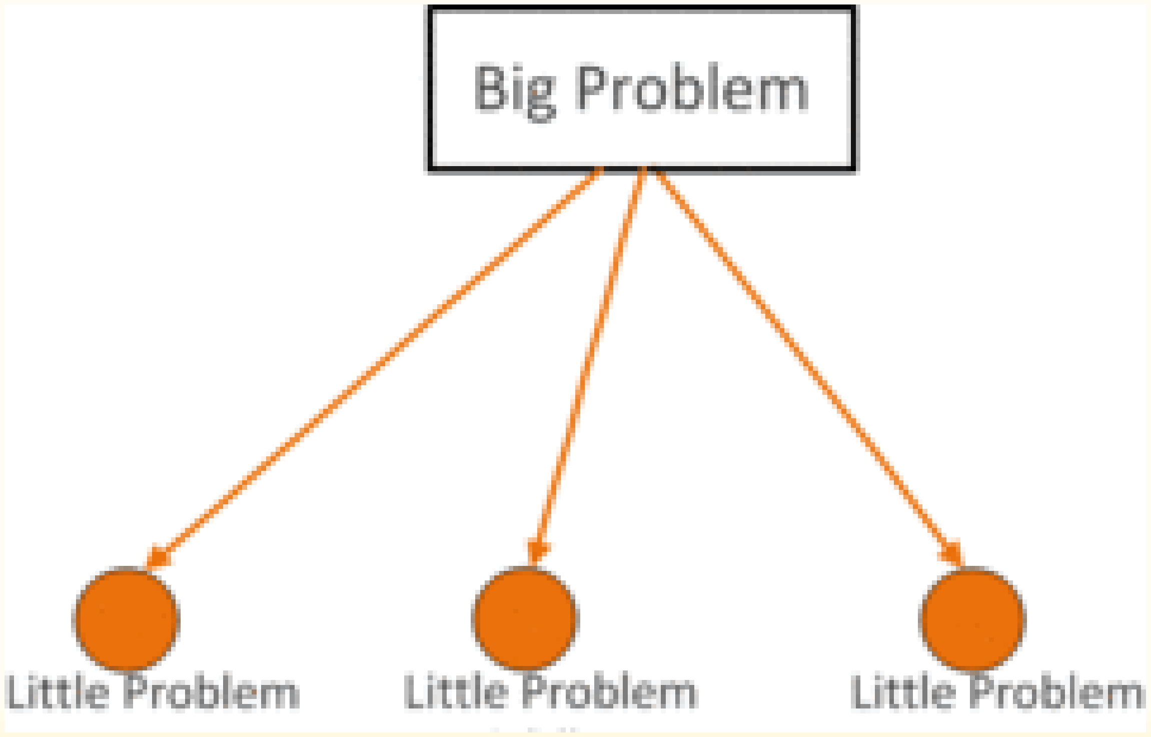
Principles of SOA

- Service Discoverability -- Services can be discovered (usually in a service registry).



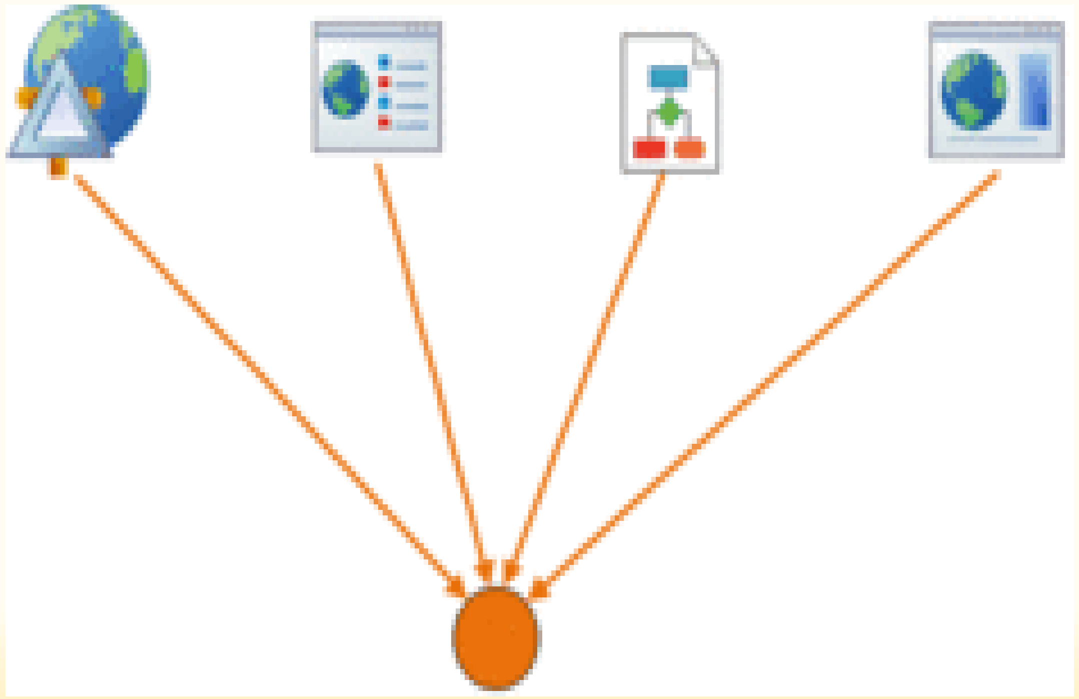
Principles of SOA

- Service Composability -- Services break big problems into little problems.



Principles of SOA

- Service Interoperability -- Services should use standards that allow diverse subscribers to use the service. This is considered so obvious these days that it is often dropped as a principle.



Web Services

- Web services are the piece of software that makes itself available over the internet.
- Web services use a standard XML messaging system for data sharing and storing. XML is portable, machine and language independent.
- Web services works on W3C (World Wide Web Consortium) protocols

<ins>Types of Web Services:</ins>

- XML RPC
- SOAP
- REST

Features of Web Services

- Loosely coupled
- Reusability
- Open Standards
- Interpretability
- Software as service
- Secured
- Scalable

Ex: Payment gateways, CRM applications

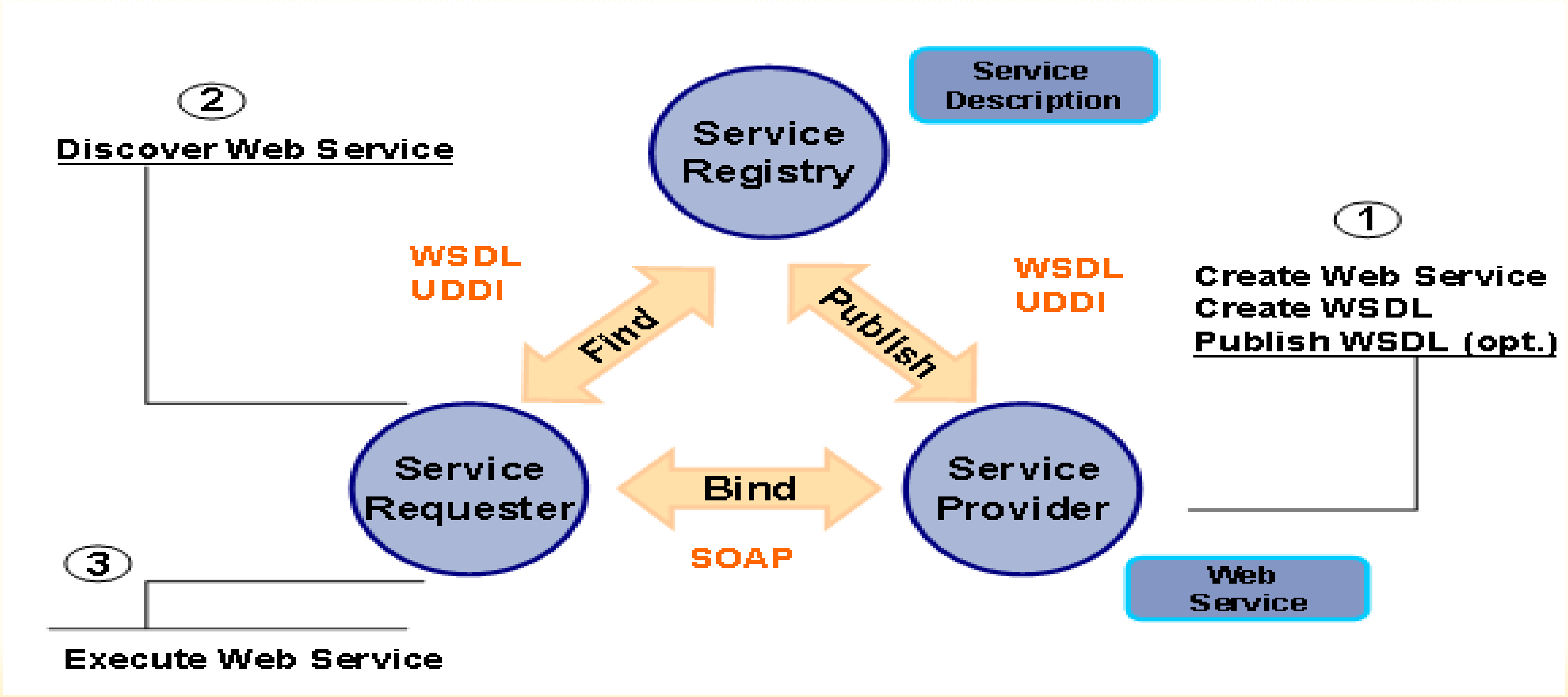
SOAP vs REST

SOAP	REST
Simple **O**bject **A**ccess **P**rotocol	**RE**presentational **S**tate **T**ransfer
Highly secured	Less secured
Used for large amount of data transactions	Used for small amount of data transaction
Havier than REST	Lighter than SOAP
Slower than REST	7 times faster than SOAP
Difficult to implement	Easy to implement
Works with HTTP, HTTPs, JMS, SMTP protocols	Works with only HTTP & HTTPS protocols
XML used for SOAP is call WSDL	XML used for REST is called as WADL
SOAP builds an XML based protocol on top of HTTP or sometimes TCP/IP	REST is defined by HTTP, URI, Media Formats and Application Specific Coordination Protocol

WSDL & WDAL

WSDL	WDAL
Web Services Description Language	Web Application Description Language
WSDL defines contract between client and service and is static by its nature.	WDAL defines contract between client and service is somewhat complicated and is defined by HTTP, URI, Media Formats and Application Specific Coordination Protocol.
A machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns	A machine-readable XML description of HTTP-based web applications used for describing the functionality offered by a web service
www.w3.org/TR/wsdl20/	www.w3.org/Submission/wadl/
WSDL is flexible in service binding options (for example, services can be offered via SMTP mail servers) it did not originally support HTTP operations other than GET and POST	It is not as flexible as WSDL (no binding to SMTP servers), but it is sufficient for any REST service and much less verbose.

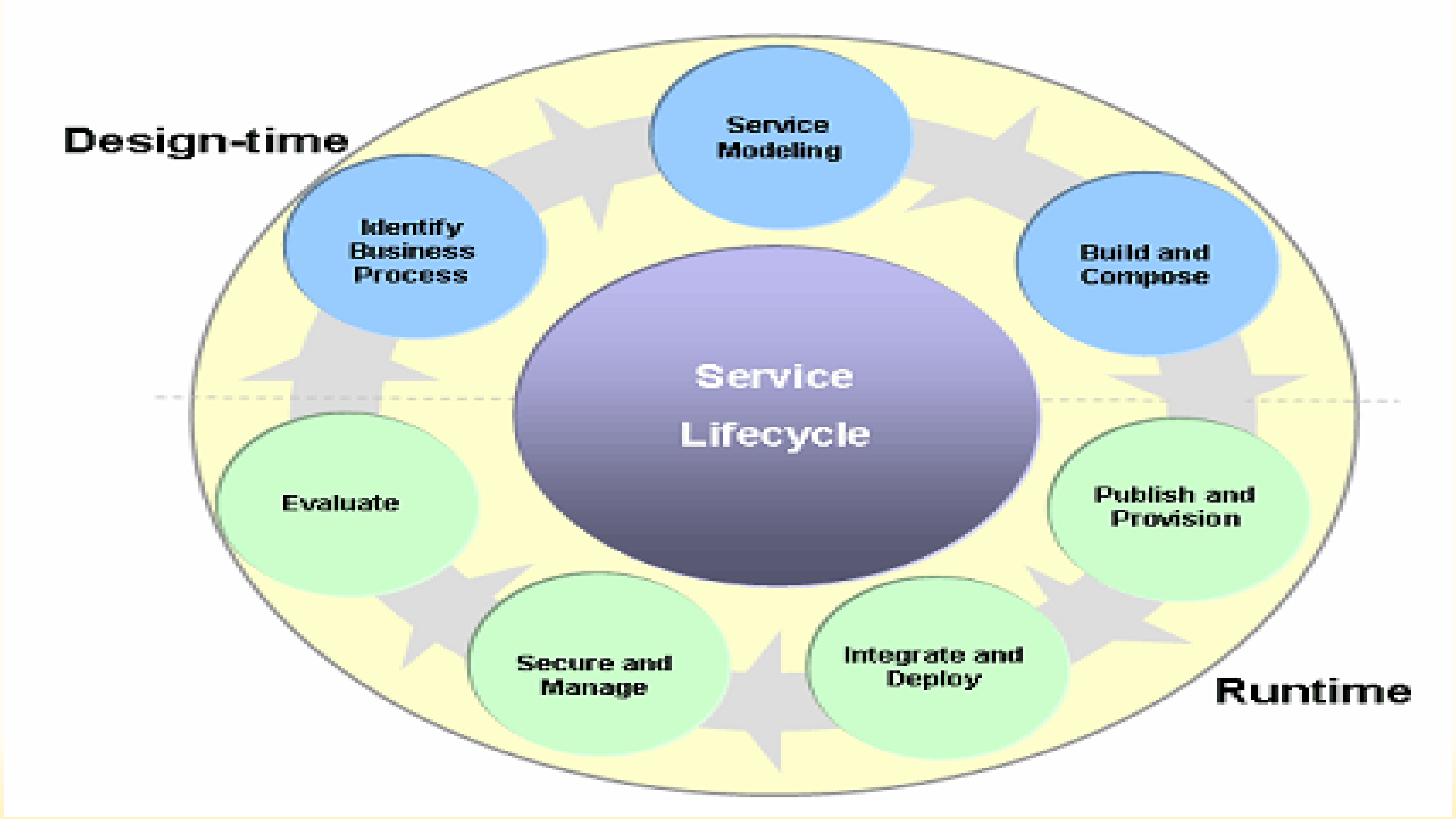
Web Services Architecture



Service Registry & Repository

- An SOA registry supports the UDDI (Universal Description, Discovery and Integration) specification
- An XML- (Extensible Markup Language) based registry that was developed for the purpose of making systems interoperable for e-commerce
- The SOA registry expands on UDDI by facilitating enhanced and continuous revision of content. Such content can exist as XML documents, process descriptions and information about potential business partners.
- An SOA registry allows a participating enterprise to discover and use current and relevant information more quickly than is possible with UDDI alone.

SOA Lifecycle



SOA Testing Techniques

- **Black Box** - Black Box testing refers to the technique of testing a system *with no knowledge* of the internals of the system, no access to source code and system architecture. A Black Box tester typically tests the functionalities provided by the web services.
  - Advantages - Efficient Testing, Unbiased Testing, Non intrusive, Easy to execute
  - Disadvantages - Localized Testing, Inefficient Test Authoring, Blind Coverage

- **White Box** - White Box testing refers to the technique of testing a system with knowledge of the internals of the system, access to source code and system architecture. A White Box tester typically analyzes source code, derives test cases from knowledge about the source code, and finally targets specific code paths to achieve a certain level of code coverage.
  - Advantages - Increased Effectiveness, Full Code Pathway Capable, Early Defect Identification, Reveal Hidden Code Flaws
  - Disadvantages - Difficult To Scale, Difficult to Maintain, Cultural Stress, Highly intrusive

SOA Testing Techniques

- **Grey Box** - Gray Box testing refers to the technique of testing a system with limited knowledge of the internals of the system, access to detailed design documents with information beyond requirement documents. Gray Box tests are generated based on information such as state-based models or architecture diagrams of the target system.
    - Advantages - Offers Combined Benefits, Non Intrusive, Intelligent Test Authoring, Unbiased Testing
    - Disadvantages - Partial Code Coverage, Defect Identification,

SOA Testing Methodology

- Governance Testing - ensuring that each new and existing service conforms to the standards, policies and objectives of an organization for the entire life of that service

- Service-component-level Testing / Unit Testing - to test that the code not only successfully compiles, but the basic functionality of the components and functions within a service are working as specified

- Process Testing - Process/Orchestration testing ensures services are operating collectively as specified

- System Testing – defined business requirements and has met the defined business acceptance criteria

SOA Testing Methodology

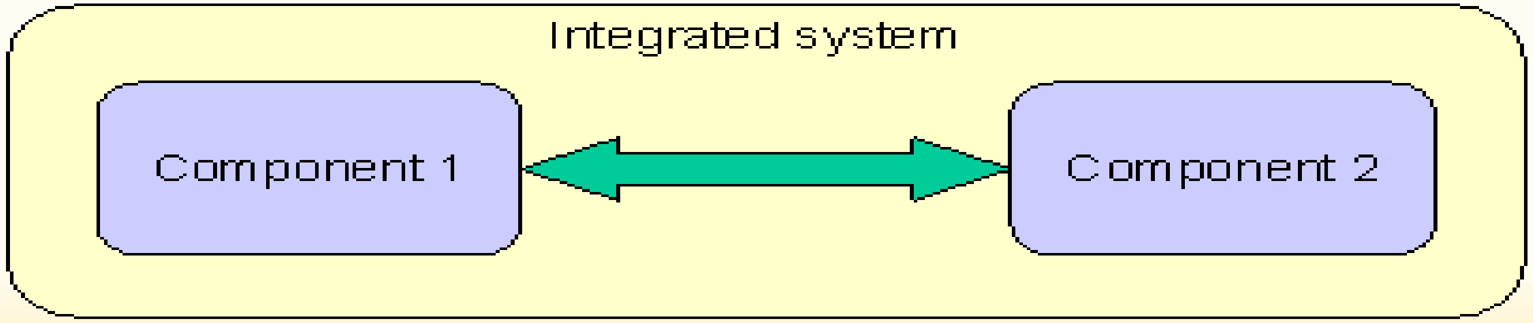
- <u>Integration Testing</u> - determine if interface behavior and information sharing between the services, are working as specified

- <u>Security Testing</u> – to ensure protection from unauthorized software

- <u>Service level Testing</u> - Service reuse will demand each service is delivered from this level/phase of testing with a comprehensive statement of quality and even a guarantee. This includes Functional Testing, Performance Testing and Security Testing.

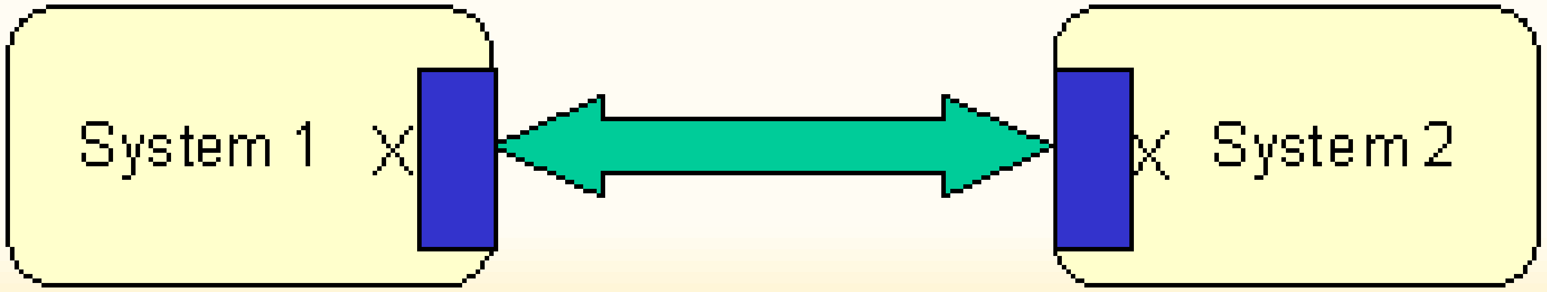
Integration Testing

- **Integration** is the act of forming, coordinating or blending into a functioning or unified whole.
- Testing the act of forming coordinating or blending is **Integration Testing**
- Integration is where a system is subsumed within another either conceptually or physically.
- ESB's and Enterprise portals are more akin to integration systems.



Interoperability Testing

- **Interoperation** is the ability of two or more systems or components to exchange information and to use the information that has been exchanged.
- Interoperation is a peer to peer sort of thing where the two systems interoperate between themselves sharing information (which may be state and so can change processes in one another).
- To test end-to-end functionality between (at least) two communicating systems is as required by the standard(s) on which those systems are based is called **Interoperability Testing**.



Compatibility Testing

- **Compatibility** is concerned with the ability of two or more systems or components to perform their required functions while sharing the same environment.
- Compatibility is not concerned with interoperability.
- **Compatibility testing** is a non-functional testing conducted on the application to evaluate the application's compatibility within different environments.
- There are two types of Compatibility Testing
  - **Backward compatibility** - determine if changes to an interface will affect existing users of the interface
  - **Forward compatibility** - a system that is designed to it fits with planned future versions of itself.

Performance Testing

Def

A non-functional test to validate capacity and reliability.

Need

To ensure a good user experience (fast and error free) under any user load.

Types of Performance Tests

- **Load Testing** - Checks the application's ability to perform under anticipated user loads. The objective is to identify performance bottlenecks before the software application goes live.

- **Stress Testing** - Involves testing an application under extreme workloads to see how it handles high traffic or data processing .The objective is to identify breaking point of an application.

- **Endurance / Soak Testing** - is done to make sure the software can handle the expected load over a long period of time.

- **Spike Testing** - tests the software's reaction to sudden large spikes (hikes and drops) in the load generated by users.

Types of Performance Tests

- **Scalability Testing** - The objective of scalability testing is to determine the software application's effectiveness in "scaling up" to support an increase in user load. It helps plan capacity addition to your software system.

- **Volume Testing** - Under Volume Testing large no. of. Data is populated in database and the overall software system's behavior is monitored. The objective is to check software application's performance under varying database volumes.

- **Isolation Testing** - Isolation testing is not unique to performance testing but involves repeating a test execution that resulted in a system problem. Such testing can often isolate and confirm the fault domain.

Security Testing

- Security is set of measures to protect an application against unforeseen actions that cause it to stop functioning or being exploited. Unforeseen actions can be either intentional or unintentional.
- Security Testing ensures, that system and applications in an organization, are free from any loopholes and weaknesses that may cause a big loss.

Types of Security Testing

- **Vulnerability Scanning**: This is done through automated software to scan a system against known vulnerability signatures.
- **Security Scanning:** It involves identifying network and system weaknesses, and later provides solutions for reducing these risks. This scanning can be performed for both Manual and Automated scanning.
- **Penetration testing**: This kind of testing simulates an attack from malicious hacker. This testing involves analysis of a particular system to check for potential vulnerabilities to an external hacking attempt.
- **Risk Assessment:** This testing involves analysis of security risks observed in the organization. Risks are classified as Low, Medium and High. This testing recommends controls and measures to reduce the risk.

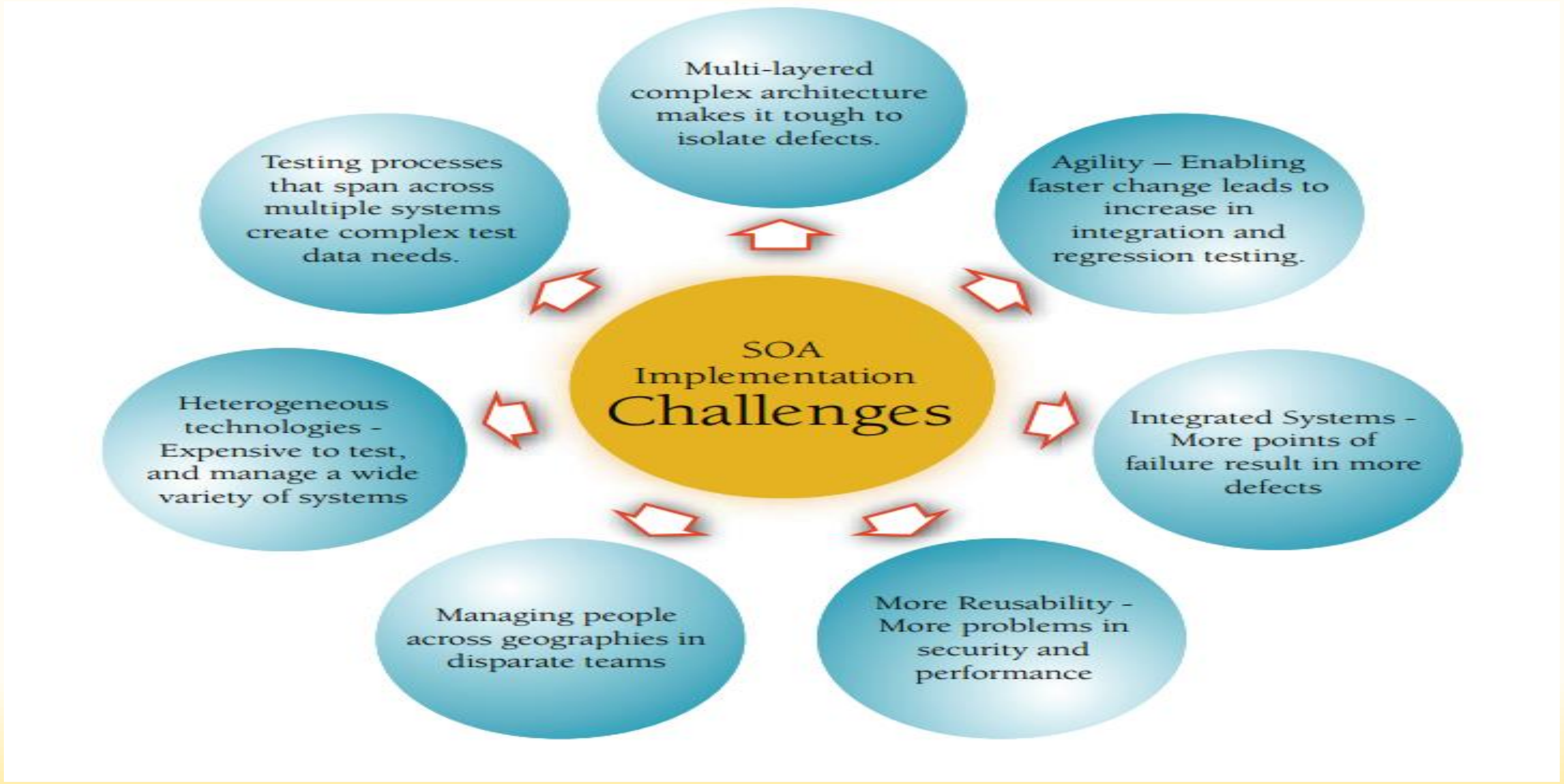
Types of Security Testing

- **Security Auditing:** This is internal inspection of Applications and Operating systems for security flaws. Audit can also be done via line by line inspection of code

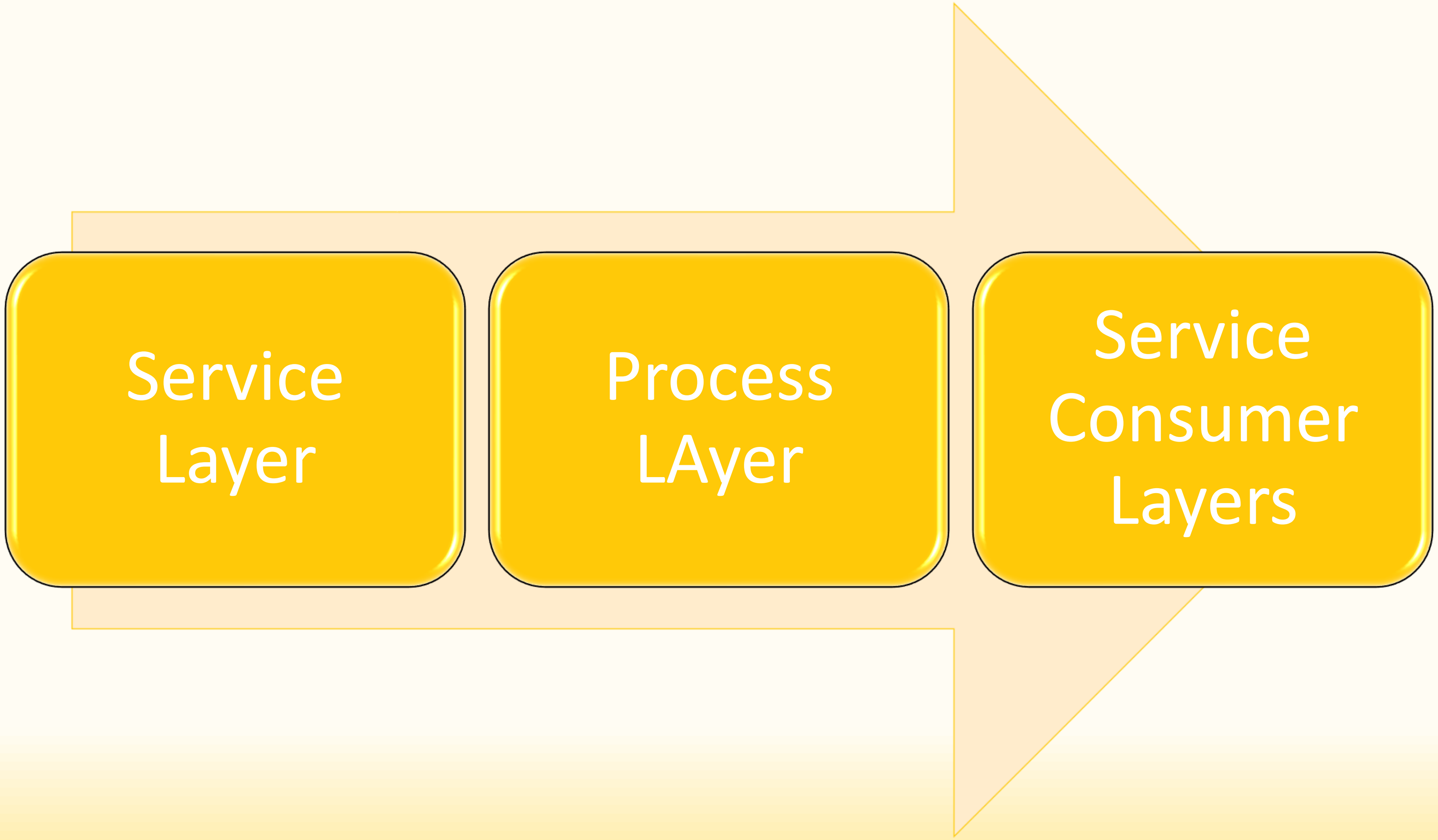
- **Ethical hacking:** It's hacking an Organization Software systems. Unlike malicious hackers ,who steal for their own gains , the intent is to expose security flaws in the system.

- **Posture Assessment:** This combines Security scanning, Ethical Hacking and Risk Assessments to show an overall security posture of an organization.

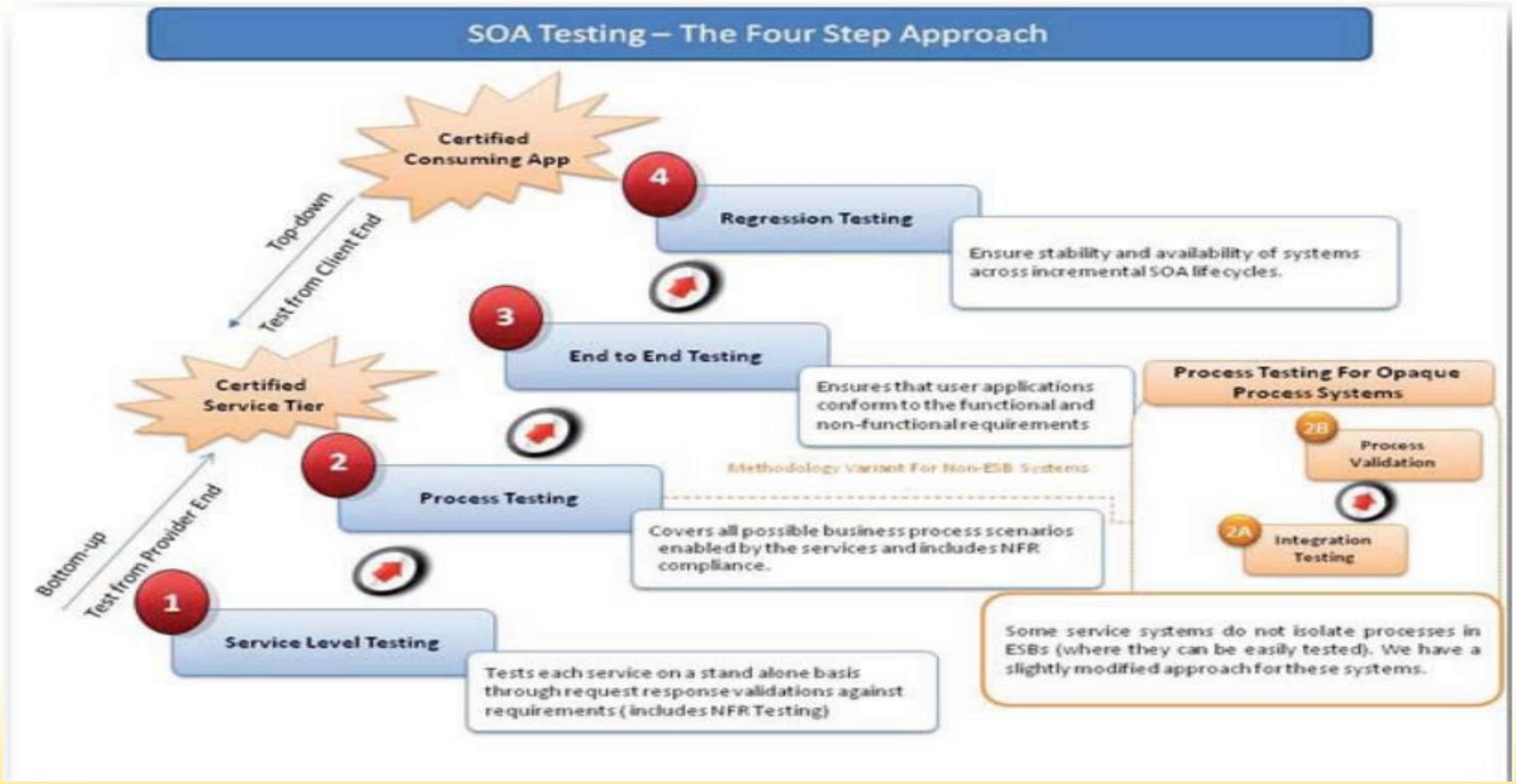
Challenges in SOA Testing


SOA Implementation Challenges
Multi-layered complex architecture makes it tough to isolate defects.
Agility – Enabling faster change leads to increase in integration and regression testing.
Integrated Systems - More points of failure result in more defects
More Reusability - More problems in security and performance
Managing people across geographies in disparate teams
Heterogeneous technologies - Expensive to test, and manage a wide variety of systems
Testing processes that span across multiple systems create complex test data needs.

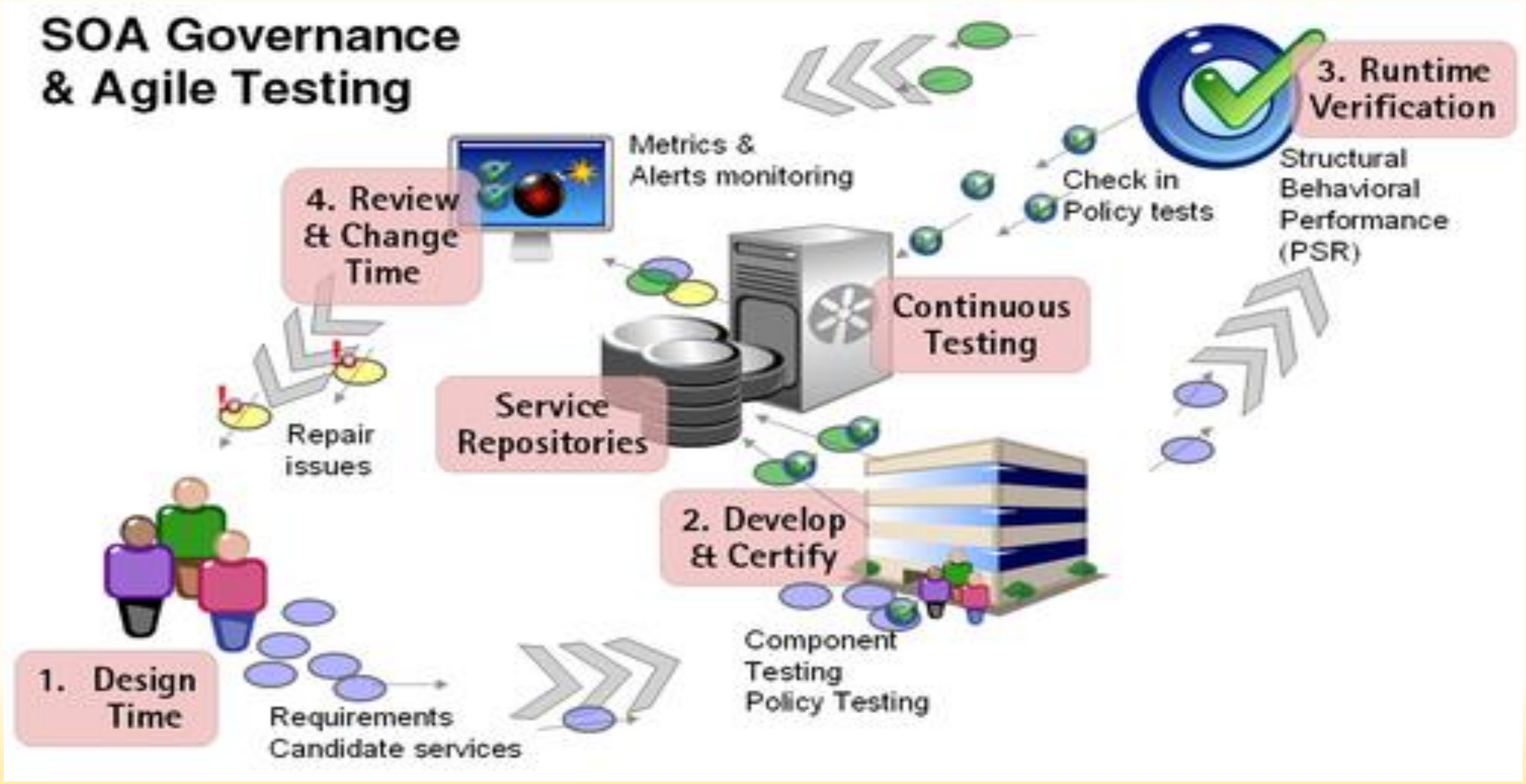
Testing Process



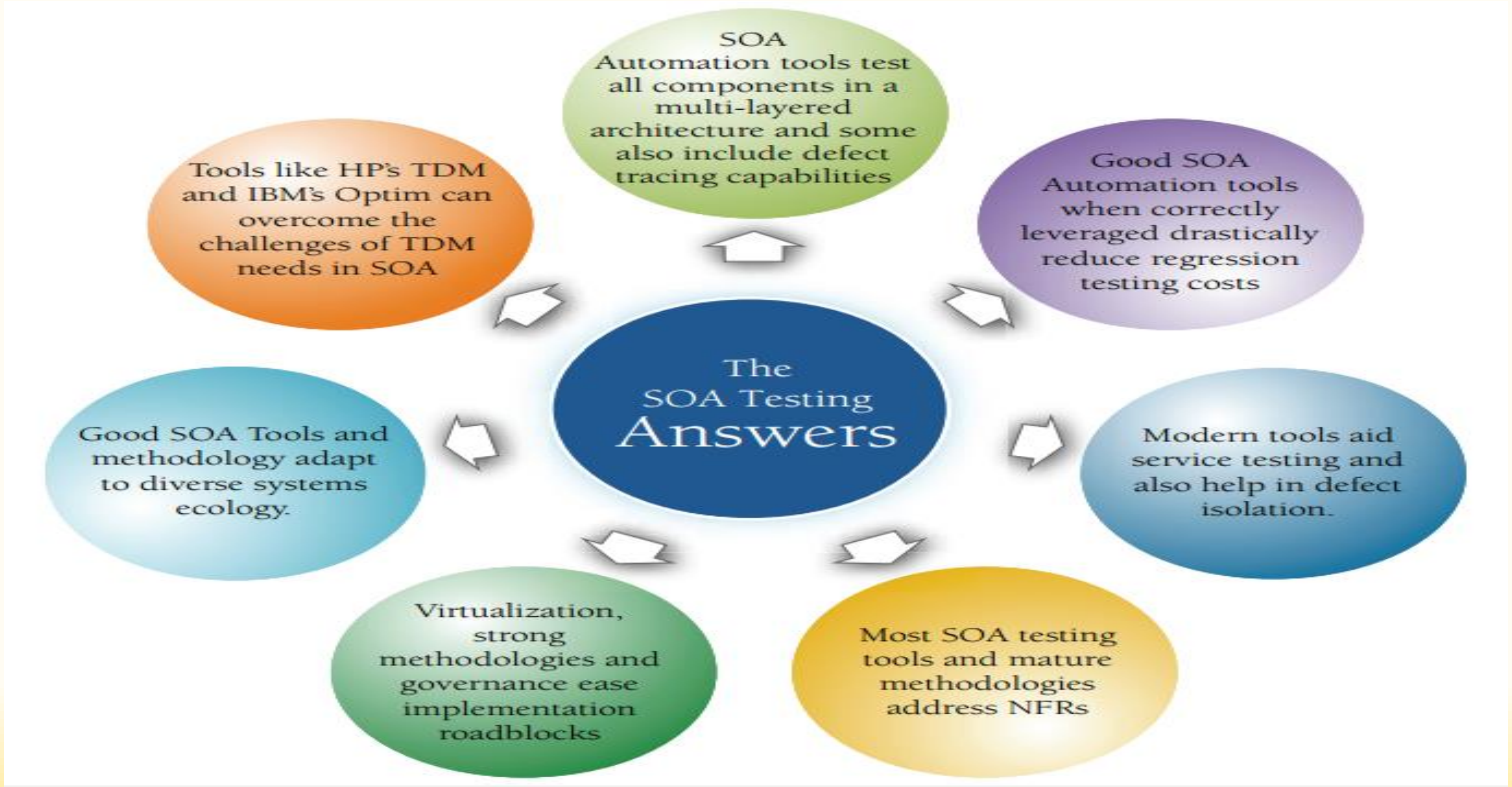
SOA Testing Approach


SOA Testing – The Four Step Approach
Certified Consuming App
Top-down
Test from Client End
Certified Service Tier
Bottom-up
Test from Provider End
4
Regression Testing
Ensure stability and availability of systems across incremental SOA lifecycles.
3
End to End Testing
Ensures that user applications conform to the functional and non-functional requirements
2
Process Testing
Covers all possible business process scenarios enabled by the services and includes NFR compliance.
1
Service Level Testing
Tests each service on a stand alone basis through request response validations against requirements (includes NFR Testing)
Methodology Variant For Non-ESB Systems
Process Testing For Opaque Process Systems
2B
Process Validation
2A
Integration Testing
Some service systems do not isolate processes in ESBs (where they can be easily tested). We have a slightly modified approach for these systems.

SOA & Agile Testing Life Cycle


SOA Governance & Agile Testing
Metrics & Alerts monitoring
3. Runtime Verification
4. Review & Change Time
Check in Policy tests
Structural Behavioral Performance (PSR)
Continuous Testing
Service Repositories
Repair issues
2. Develop & Certify
1. Design Time
Component Testing Policy Testing
Requirements Candidate services

SOA Testing Solution


The SOA Testing Answers
SOA Automation tools test all components in a multi-layered architecture and some also include defect tracing capabilities
Good SOA Automation tools when correctly leveraged drastically reduce regression testing costs
Modern tools aid service testing and also help in defect isolation.
Most SOA testing tools and mature methodologies address NFRs
Virtualization, strong methodologies and governance ease implementation roadblocks
Good SOA Tools and methodology adapt to diverse systems ecology.
Tools like HP's TDM and IBM's Optim can overcome the challenges of TDM needs in SOA

What is API and API Testing?

- **API (Application Programming Interface)** is a computing interface which enables communication and data exchange between two separate software systems.

- **API TESTING** is a software testing type that validates Application Programming Interfaces (APIs).

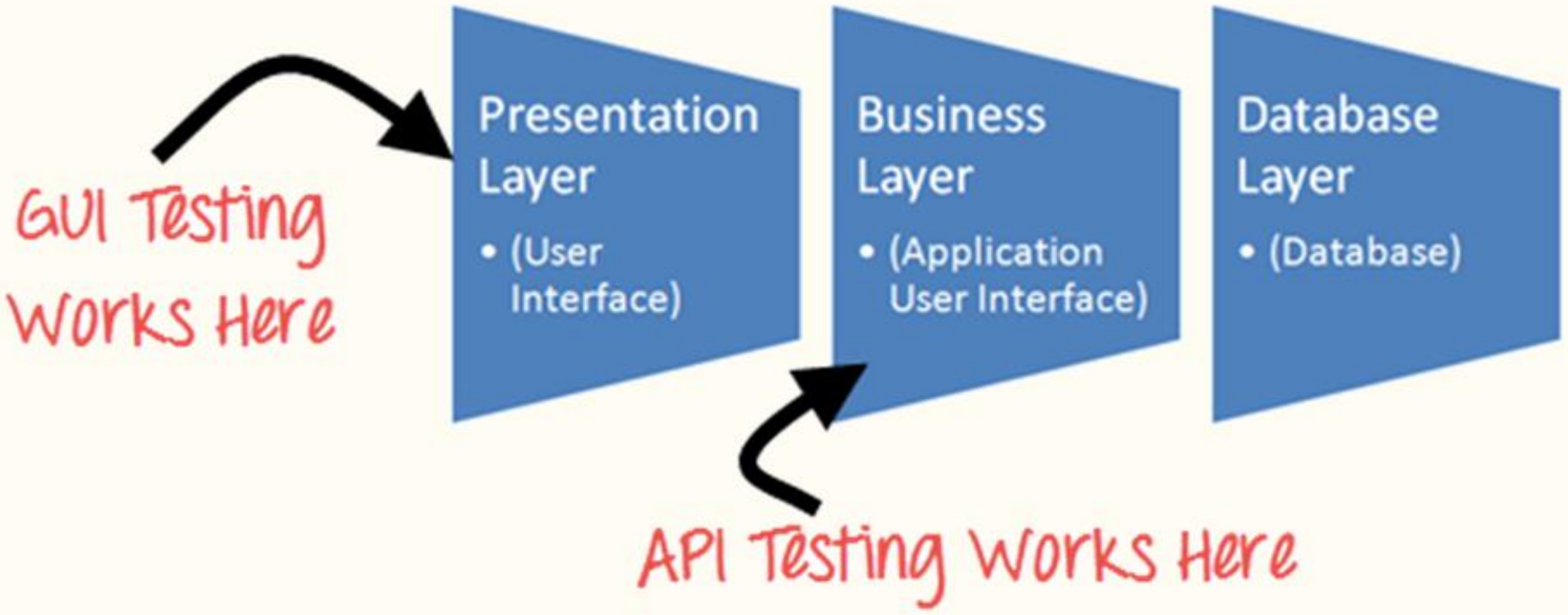
- The purpose of API Testing is to check the functionality, reliability, performance, and security of the programming interfaces.

- In API Testing, instead of using standard user inputs(keyboard) and outputs, you use software to send calls to the API, get output, and note down the system's response.

- API tests are very different from GUI Tests and won't concentrate on the look and feel of an application.

- It mainly concentrates on the business logic layer of the software architecture.

API Testing


GUI Testing Works Here
Presentation Layer
• (User Interface)
Business Layer
• (Application User Interface)
Database Layer
• (Database)
API Testing Works Here

Why to Test API?

- It is to ensure the API does what it’s supposed to do.
- It is to ensure that the API can handle the load.
- It will help to detect the ways the users can mess things up.
- To ensure that the APIs work across devices, browsers, and operating systems.
- With API testing there could be costs involved due to system failure.

What do you need To Start API Testing?

- Who is your target audience?
- Who is the API user?
- Which environment(s) should the API typically be used?
- Which aspects are you testing?
- Which problems are we testing for?
- What are your priorities for testing?
- What is supposed to happen under normal circumstances?
- What could potentially happen under abnormal circumstances?
- What are the criteria for Pass or a Fail? What data is the desired output? What is the chain of events?
- Which other APIs could this API interact with?
- Who on your team is in charge of testing what?

API Testing Approach

- **API Testing Approach** is a predefined strategy.
- It doesn’t include testing of source code.
- The most important aspect are functionalities, testing techniques, input parameters and the execution of test cases.
- Perform black box testing
- Steps to be considered for API testing –
  - Understanding the functionality of the API program
  - Clearly define the scope of the program
  - Apply testing techniques such as equivalence classes, boundary value analysis, and error guessing and write test cases for the API
  - Input Parameters for the API need to be planned and defined appropriately
  - Execute the test cases and compare expected and actual results.

How to Test API?

- API testing is a form of integration testing that is performed to test the API to validate its functionality, reliability, performance, and security of the application for which API is used.
- In this testing, the APIs and the integrations they enable are tested.
- This testing is usually performed for software systems that have multiple APIs.
- Types of testing to be done –
    - Unit Testing: For testing the functionality of individual operation.
    - Functionality Testing: For testing the functionality of multiple unit tests when tested together.
    - Load Testing: For testing the functionality and performance under load conditions.
    - Error Detection: For identifying any errors such as exceptions and resource leaks.
    - Security Testing: For testing that the API is secure against any external threats.
    - UI Testing: For testing the functionality of user interface as part of end-to-end integration tests to ensure the UI functions as expected.
    - Interoperability & WS Compliance testing: This type of testing applies to SOAP APIs and it ensures conformance to Web Services (WS) Interoperability Profiles. The compliance is tested to ensure that the predefined standards are met.
    - Penetration Testing: For detecting any vulnerabilities of an application from attackers.
    - Fuzz Testing: For testing the API by giving inputs in an attempt to crash it.

Test Cases for API Testing

- Test cases of API testing are based on
  - **Return value based on input condition:** it is relatively easy to test, as input can be defined and results can be authenticated
  - **Does not return anything:** When there is no return value, a behavior of API on the system to be checked
  - **Trigger some other API/event/interrupt:** If an output of an API triggers some event or interrupt, then those events and interrupt listeners should be tracked
  - **Update data structure:** Updating data structure will have some outcome or effect on the system, and that should be authenticated
  - **Modify certain resources:** If API call modifies some resources then it should be validated by accessing respective resources

How to Test API

API automation testing should cover at least following testing methods apart from usual SDLC process

- **Discovery testing:** The test group should manually execute the set of calls documented in the API like verifying that a specific resource exposed by the API can be listed, created and deleted as appropriate
- **Usability testing:** This testing verifies whether the API is functional and user-friendly. And does API integrates well with another platform as well
- **Security testing:** This testing includes what type of authentication is required and whether sensitive data is encrypted over HTTP or both
- **Automated testing:** API testing should culminate in the creation of a set of scripts or a tool that can be used to execute the API regularly
- **Documentation:** The test team has to make sure that the documentation is adequate and provides enough information to interact with the API. Documentation should be a part of the final deliverable

Sample Test Cases

- Validate the keys with the Min. and Max range of APIs (e.g. maximum and minimum length)
- Keys verification. If we have JSON, XML APIs we should verify it’s that all the keys are coming.
- Have a test case to do XML, JSON Schema validation.
- Verify the Parse the Response data
- Verify the JSON Schema validation, Verify the Field Type, Verify the Mandatory Fields
- Valid Response headers & Negative Testcases response
- Verify that how the APIs error codes handled.
- Verify the response HTTP status code.
- Valid Response payload
- Chaining Request verification.
- Verification of APIs with Data parameters.
- End to End CRUD flows
- Database Integrity Test Cases
- File Upload Testcases

Output of an API

- An output of API could be

  1. Any type of data
  2. Status (say Pass or Fail)
  3. Call another API function.

Response to be tested

In this testing, a request is sent to the API with known to analyze the response that includes:

- Accuracy of data
- HTTP status code
- Response time
- Error codes of any errors returned by API
- Authorization checks
- Results of non-functional tests such as performance, security, etc.

Types of Bugs that API testing detects

- Fails to handle error conditions gracefully
- Unused flags
- Missing or duplicate functionality
- Reliability Issues. Difficulty in connecting and getting a response from API.
- Security Issues
- Multi-threading issues
- Performance Issues. API response time is very high.
- Improper errors/warning to a caller
- Incorrect handling of valid argument values
- Response Data is not structured correctly (JSON or XML)

Challenges of API Testing

- Main challenges in Web API testing is **Parameter Combination, Parameter Selection, and Call Sequencing**
- There is no GUI available **to test the application which makes** difficult to give input values
- Validating and Verifying the output in a different system is little difficult for testers
- Parameters selection and categorization is required to be known to the testers
- Exception handling function **needs to be tested**
- Coding knowledge is necessary for testers

Best Practices of API Testing

- API Test cases should be grouped by test category
- On top of each test, you should include the declarations of the APIs being called.
- Parameters selection should be explicitly mentioned in the test case itself
- Prioritize API function calls so that it will be easy for testers to test
- Each test case should be as self-contained and independent from dependencies as possible
- Avoid “test chaining” in your development
- Special care must be taken while handling one-time call functions like – Delete, CloseWindow, etc…
- Call sequencing should be performed and well planned
- To ensure complete test coverage, create API test cases for all possible input combinations of the API.

HTTP Methods

- GET
- POST
- PUT
- HEAD
- DELETE
- PATCH
- OPTIONS

Response codes:

1xx Informational	2xx Success
100 Continue	200 OK
101 Switching Protocols	201 Created
102 Processing (WebDAV)	202 Accepted
	203 Non-Authoritative Information
3xx Redirection	204 No Content
300 Multiple Choices	205 Reset Content
301 Moved Permanently	206 Partial Content
302 Found	207 Multi-Status (WebDAV)
303 See Other	208 Already Reported (WebDAV)
304 Not Modified	226 IM Used
305 Use Proxy	
306 (Unused)	4xx Client Error
307 Temporary Redirect	400 Bad Request
308 Permanent Redirect (experiemental)	401 Unauthorized
	402 Payment Required
5xx Server Error	403 Forbidden
500 Internal Server Error	404 Not Found
501 Not Implemented	405 Method Not Allowed
502 Bad Gateway	406 Not Acceptable
503 Service Unavailable	407 Proxy Authentication Required
504 Gateway Timeout	408 Request Timeout
505 HTTP Version Not Supported	409 Conflict
506 Variant Also Negotiates (Experimental)	410 Gone
507 Insufficient Storage (WebDAV)	411 Length Required
508 Loop Detected (WebDAV)	412 Precondition Failed
509 Bandwidth Limit Exceeded (Apache)	413 Request Entity Too Large
510 Not Extended	414 Request-URI Too Long
511 Network Authentication Required	415 Unsupported Media Type
598 Network read timeout error	416 Requested Range Not Satisfiable
599 Network connect timeout error	417 Expectation Failed
	426 Upgrade Required
	428 Precondition Required
	429 Too Many Requests
	431 Request Header Fields Too Large
	451 Unavailable For Legal Reasons
	499 Client Closed Request (Nginx)

Testing an API with GET requests

- This is most frequent type of request made by consumers of the service, so it's important to **check every known endpoint with a GET request**.

- At a basic level, these things should be validated:
  - Check that a valid GET request returns a 200 status code.
  - Ensure that a GET request to a specific resource returns the correct data. For example, GET /users returns a list of users.

- GET is often the default method in HTTP clients, so creating tests for these resources should be simple with any tool you choose.

Validation points for POST

- Create a resource with a POST request and ensure a 200 status code is returned.
- Next, make a GET request for that resource, and ensure the data was saved correctly.
- Add tests that ensure POST requests fail with incorrect or ill-formatted data.

Validation points for PUT

- Generally, when a PUT request creates a resource the server will respond with a 201 (Created), and if the request modifies existing resource the server will return a 200 (OK) or 204 (No Content).
- Repeatedly calling a PUT request always returns the same result (idempotent).
- The proper status code is returned when creating and updating a resource (eg, 201 or 200/204).
- After updating a resource with a PUT request, a GET request for that resource should return the correct data.
- PUT requests should fail if invalid data is supplied in the request -- nothing should be updated.

Validation points for DELETE

A typical test case for a DELETE request would look like this:

- Create a new user with a POST request to /users
- With the user id returned from the POST, make a DELETE request to /users/{{userid}}
- A subsequent GET request to /users/{{userid}} should return a 404 not found status code.

- In addition, sending a DELETE request to an unknown resource should return a non-200 status code.

Common API errors

- Invalid SSL certificates
- Service provider outages
- Too many redirects
- Invalid payload formats

Thank You