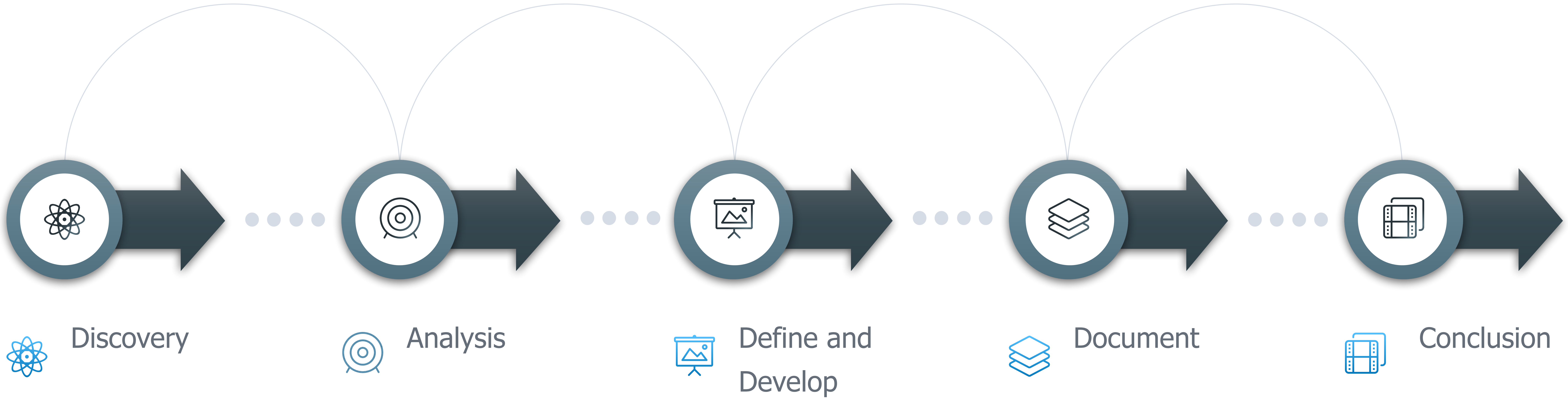
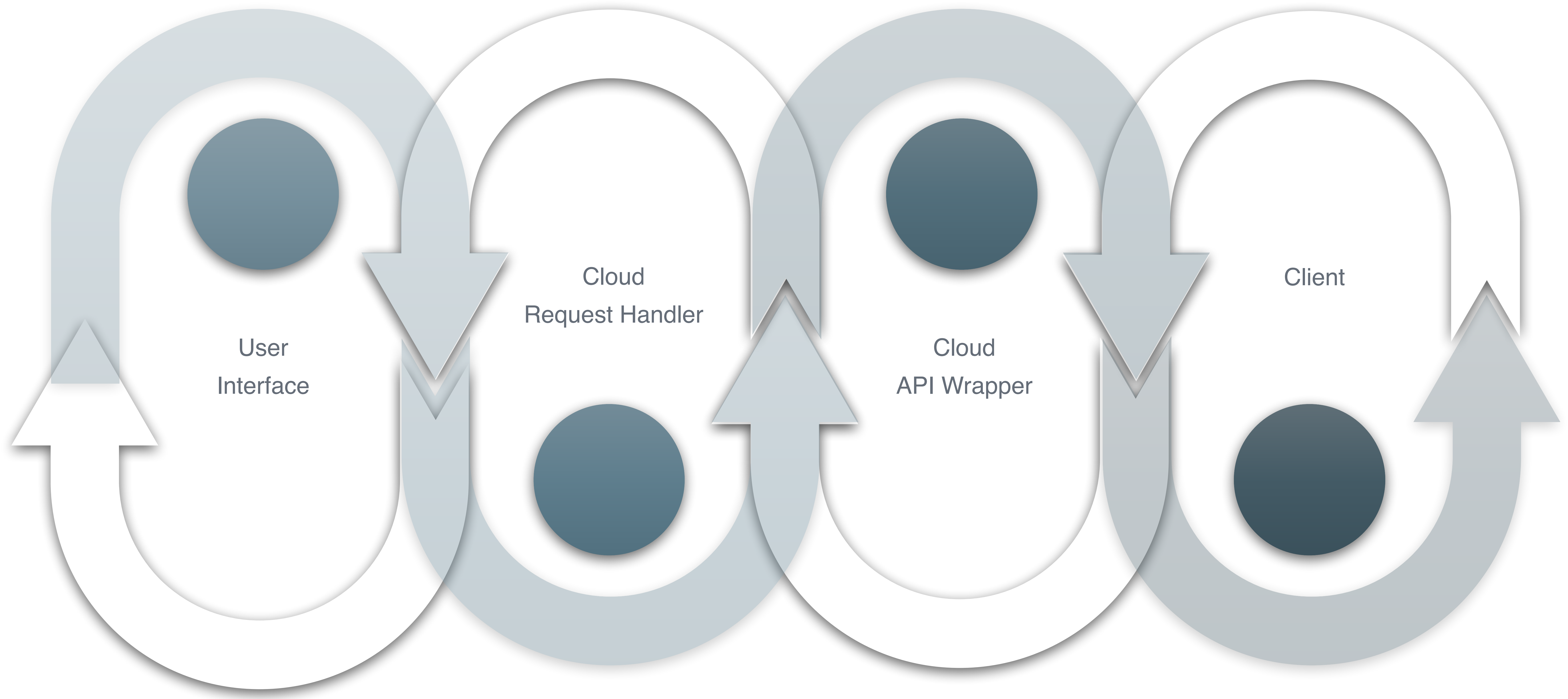
Our API Assessment Process



Critical Understandings

EcoSystem, Problem, People

Assessment Flow



SWOT Examination

Strengths

Existing applications satisfy business need.

Excellent portfolio of promotional capabilities.

Weaknesses

Security through obscurity.

No deployment methodology.

Dated technology.

Opportunities

Refactor APIs as products.

Leverage wholesale application community to drive business.

Secure application by leveraging existing infrastructure.

Threats

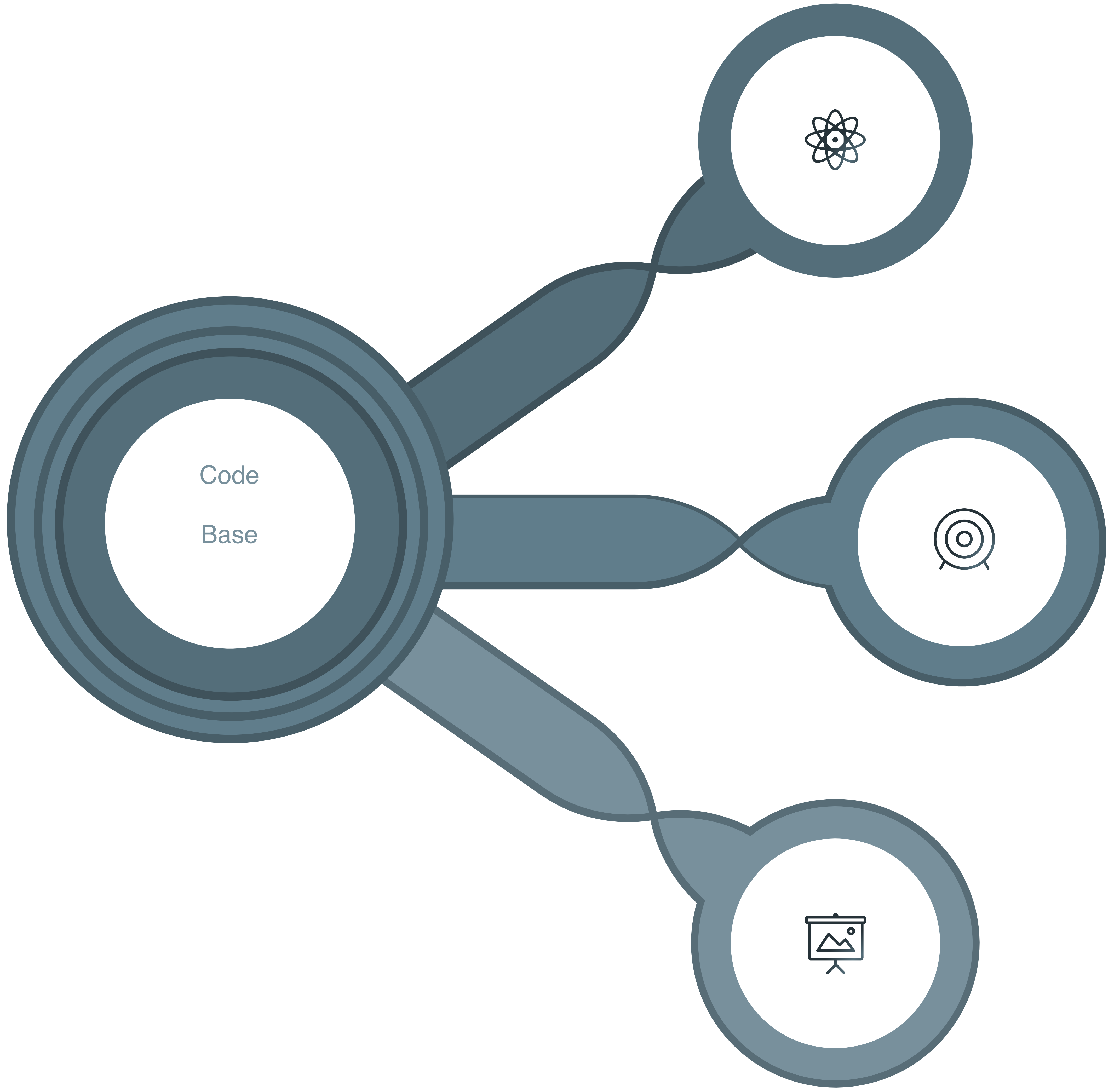
Security and policy.

Developer attraction to dated technology.

Legal liability.

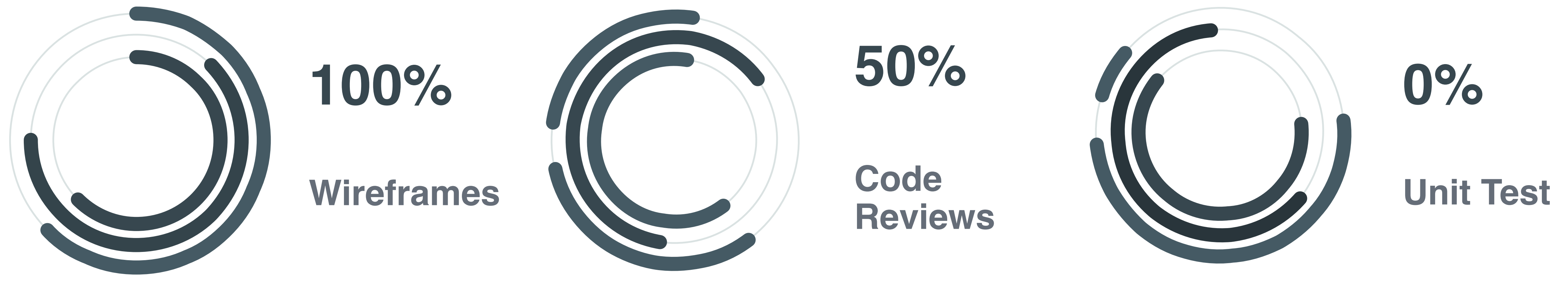
Code Readiness

Is there enough information for a developer to begin coding and testing?



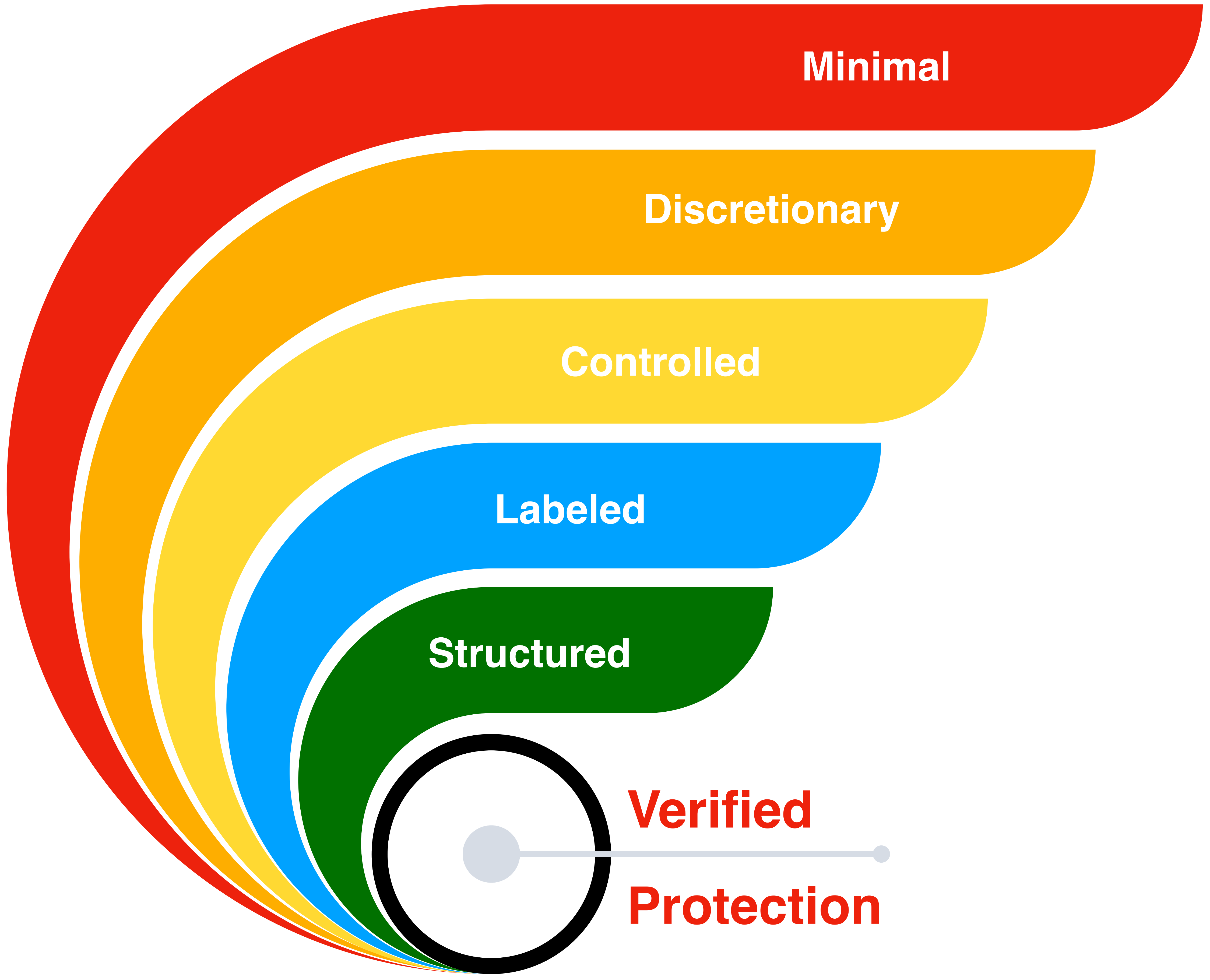
Code **Start**

Instead of measuring code quality, readability, the attempt here is to measure if the developer has enough information to proceed through the DevOps lifecycle.



API Security Level

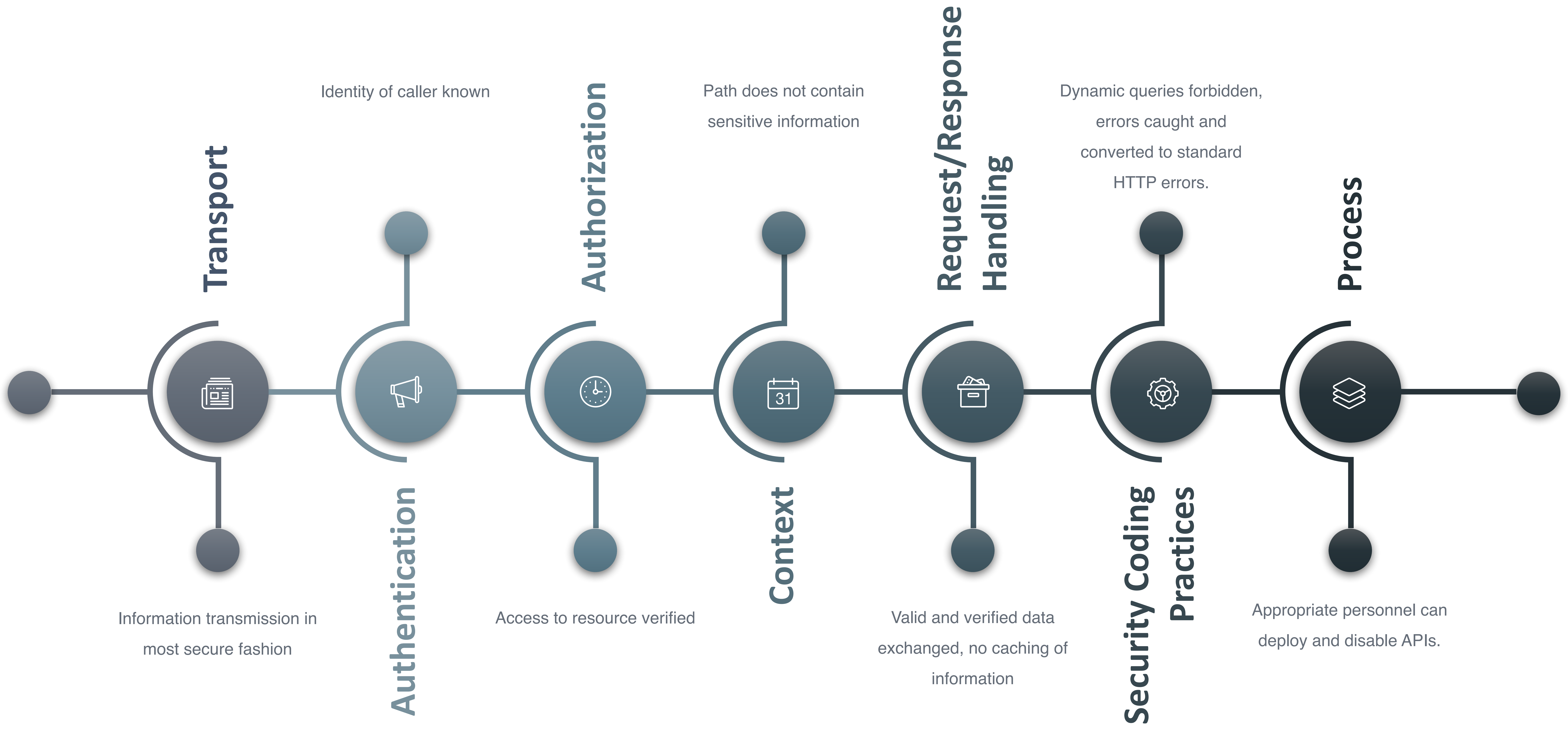
The ability to protect information systems and assets while delivering business value through risk assessments and mitigation strategies.



Risk	Security Level Description
High	**Level 1 - Minimal Protection** **Zero Characteristics, Evaluation Failed**
Somewhat High	Level 2 - Discretionary Protection Identification, Authentication, Assurance Minimal, DAC
Medium	Level 3 - Controlled Access Auditing Capable, Security Testing
Somewhat Low	Level 4 - Labeled Security MAC for objects, Labeled Object Access, Stringent Security Testing, Security Model
Low	Level 5 - Structured Protection High level design, Layered Abstraction, Tamperproof functions, Admin guide, Design Documentation
No Risk	Level 6 - Verified Protection

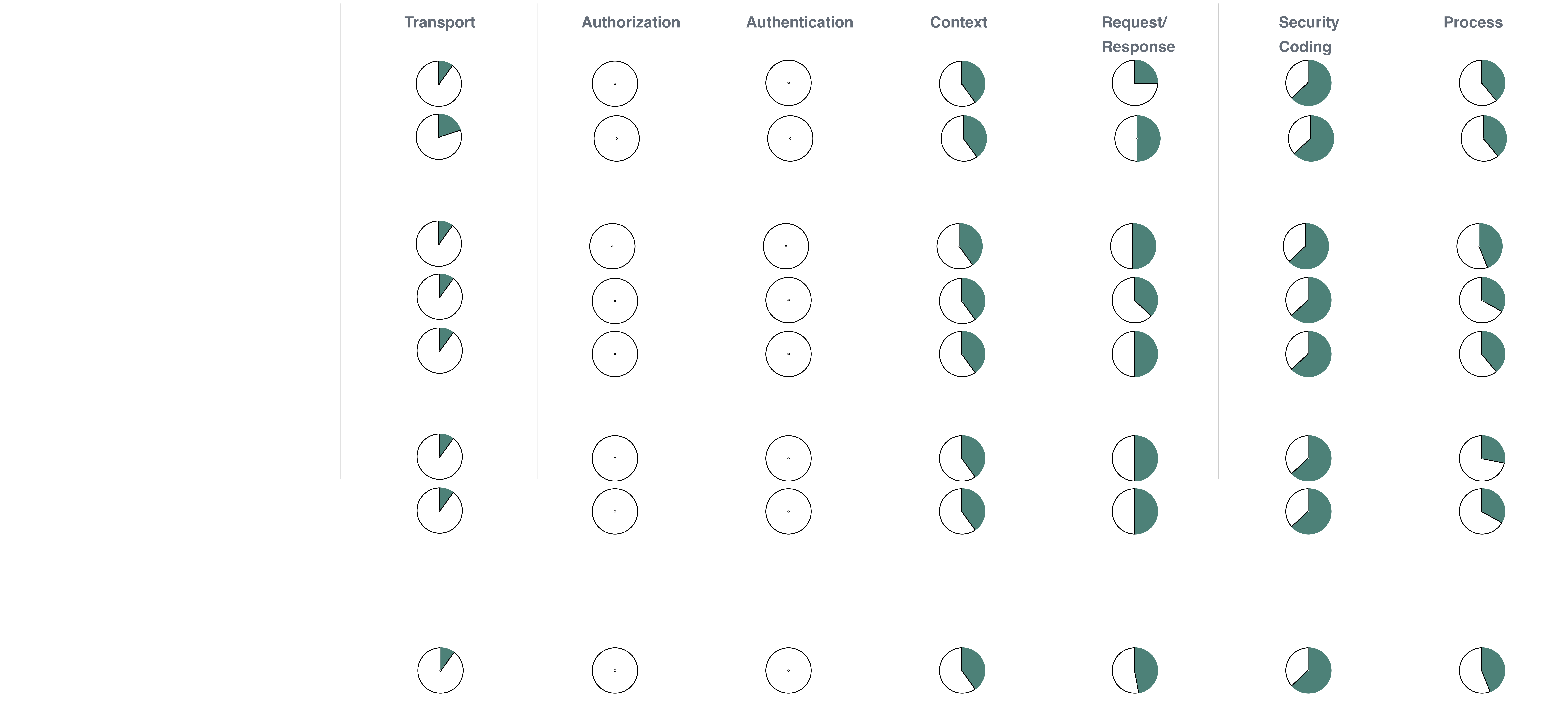
We are here

API Security Audit

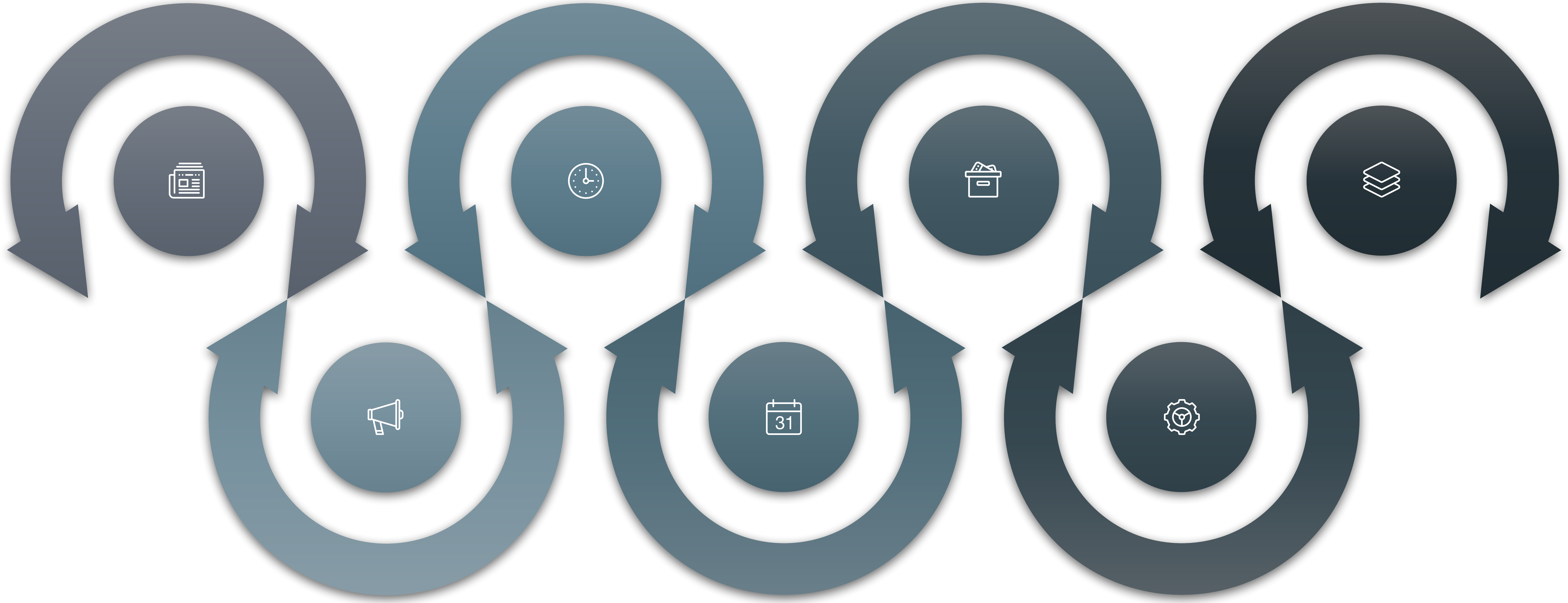

Transport
Information transmission in most secure fashion
Authentication
Identity of caller known
Authorization
Access to resource verified
Context
Path does not contain sensitive information
Request/Response Handling
Valid and verified data exchanged, no caching of information
Security Coding Practices
Dynamic queries forbidden, errors caught and converted to standard HTTP errors.
Process
Appropriate personnel can deploy and disable APIs.
31

Current State Measurements

Degree of security



Future State



Secure end to end
connection

Authenticate First, then
authorize,
Allow only enough access

Log all API
Interaction to get
end-to-end usage
view

Configure and
adopt standards,
avoid coding
security

Layer API security
IP whitelisting
Two way SSL/TLS
TLS version enforcement
Geo Filtering
Filter out bots

Use simplest method of
authentication, authorization
for API. Don't implement your
own.

Leverage API
policies

Incorporation Strategy

Step 1 — Transport

- All communication to network edge use TLS
- All communication from edge to backend use TLS
- Trusted issued certificates, checked for validity

Step 2 — Gateway

- Place all APIs behind a gateway
- Utilize gateway Policy
- Establish token use

Step 3 — Policy

- Caching
- Security
- Mediation

Step 4 — Authentication

- API Gateway is only authorized caller to backend
- No part of API operates without validating identity of caller
- No caching of credentials

Step 5 — Authorization

- Backend system validates the provider claim set
- Return error on authorization failure
- All code paths require authorization by default.

Step 6 — Process

- Code passes security audit
- Code base consolidated into deployment process
- Unit test written

API Security Yardstick

All communications to the edge of the network are secured using the most recent version of TLS with perfect forward secrecy and a modern cipher set.
All communications from the edge of the network to backends are secured using the most recent version of mutual TLS with perfect forward secrecy and a modern cipher set.
No export grade encryption algorithms are used.
Certificates are issued by a trusted PKI.
Certificates are checked for revocation on every call.
Certificates are checked for validity period on every call.
Certificates are checked against presented server names for every call.
Certificates uniquely identify a specific system (i.e., wildcard certificates are not used).
Certificates are properly controlled to ensure their private keys are protected.
Certificate pinning or another suitable certificate protection method should be employed.
If OAuth is used, OAuth 2.0 is used instead of 1.0a.
OAuth or another suitable authentication mechanism to used to authenticate access to all endpoints.
The API Gateway is responsible for handling authentication; endpoints do not authenticate users.
All calling applications provide an API key in the header.
The API Gateway does not cache the results of authentication for given credentials.
The API Gateway validates that a presented Bearer token was issued to the presenting application.
Authentication results include a set of claims.
Bearer tokens are signed, then encrypted, JWTs when exposed to calling applications.
The API Gateway transmits claims to backend systems to handle domain and business logic-based authorization.
The API Gateway is the only authorized caller to backend systems, and it authenticates using a digital certificate over mutual TLS.
Remote systems cannot impersonate users without evidence that the user has authenticated and authorized this impersonation.
No part of the API delivery stack operates without validating the identity of its caller.
The API Gateway properly validates JWTs.
A 401 is returned for any authentication failure.
The API Gateway evaluates the calling application to authorize access to an endpoint.
The API Gateway evaluates the scopes of the Bearer token or an equivalent mechanism to authorize access to an endpoint.
Authorization that requires domain knowledge is implemented below the API Gateway.
The API Gateway provides claims to underlying systems to perform authorization.
Authorization results are not cached.
Backend systems validate that the provider of a claimset is authorized to provide that claimset.
If OAuth is used, users can revoke tokens without administrator assistance.
A 403 is returned for any authorization failure.
Authorization logic occurs early in the code path, before any business logic is executed.
All code paths require authorization by default.
Scope claims are used to constrain the activities permitted by an caller's provided authentication material.
Sensitive information is not visible in paths.
Sensitive information is not visible in headers.
Correlation keys are requested of clients.
If a client does not provide a correlation key, one is created by the API Gateway.
Context returned to the client that the API expects to have returned to it is cryptographically secured.
Content types of request bodies are verified.
Methods that do not accept content (such as GETs) reject requests that attempt to send such data.
Content is verified to be syntactically correct.
Content is verified against an appropriate schema, if JSON or XML.

Request bodies are not directly bound to data objects.
Properties, headers, and content in the request is ignored if the API does not understand or expect it.
Binary objects are only accepted as base64 encoded strings.
Call rates are throttled by caller and overall on an endpoint.
Errors are returned in a consistent structure.
Stack traces and other internal information about the operations of your API's backends are not returned in errors.
Proprietary information, such as server names or platforms, are not returned in response headers or bodies.
Excess information about the nature of an error, such as "invalid password" instead of "invalid or missing credentials," is not provided in errors.
Error responses only include sufficient information for a consumer to know how to correct their request.
Sensitive information is not visible in response headers.
Authentication and authorization-relevant endpoints use the no-cache directive.
Stale information is properly identified using Cache-Control headers.
All systems beyond the furthest caching layer use Cache-Control headers to disable caching.
Sensitive information is redacted, if appropriate, by data loss protection systems.
CORS should be available for any API that is likely to be accessed by JavaScript running a browser.
HSTS headers should be returned to enforce upgrades to HTTPS on all endpoints.
Responses are verified against an appropriate JSON or XML schema, if appropriate.
Responses do not contain sensitive data that the API expects the client to filter.
All SQL queries are parameterized.
Dynamic SQL in stored procedures is forbidden.
JSON is not evaluated directly as JavaScript.
External references in JSON and XML documents are not followed.
Binary code or script code in request bodies is never executed.
Security-related events are logged to a centralized security event system.
All exceptions are caught and converted to standardized errors.
All type shapes in the API's contract definition have clearly defined and explicit descriptions and validation patterns.
Developers must attest to the security of their API product, including all code within it.
Systems hosting API delivery stack components are patched regularly.
Information security personnel are part of API product teams.
APIs are tested for security flaws with penetration testing tools.
APIs are tested for common security flaws with static code analysis tools.
APIs are only developed using managed code.
Critical vulnerabilities in API delivery stack components are patched immediately.
Network isolation techniques are used to control the flow of API traffic.
Sudden changes in the usage of an API are detected and reviewed.
Security-sensitive policy in the API Gateway is maintained in shared libraries and access to change it is controlled.
Appropriate personnel can disable APIs.
Appropriate personnel can disable access to APIs by applications.
Data is marked with appropriate security classifications to identify its appropriate usage.
Developers cannot modify code in any enviroment without following the normal code progression path.
Automated test systems prevent untested code from entering environments.
Technical debt, including exceptions to these standards, is recorded and reported to management.
Non-production endpoints are not available to consumers, unless the endpoint is a well-defined sandbox endpoint exposing only test data.
APIs are properly inventoried such that the entire solution, including underlying infrastructure, are available for review and analysis.

Taliferro Tech

dba Taliferro Group
Making Things Better
06/09/2024

425.600.7066
vikki.owens@taliferro.com x102
ty.showers@taliferro.com x101
ATTN: Taliferro Group
1424 11th Ave STE 400
Seattle, WA 98122