DRAFT - ICT Project Guidance

Development:  
Custom Service Development Patterns

Version:

0.1

Author:

Sky Sigal, Solution Architect

## Description

<TODO>

## Synopsis

<TODO>

## Contents

[Description 1](#_Toc145232973)

[Synopsis 1](#_Toc145232974)

[Contents 2](#_Toc145232975)

[Synopsis 3](#_Toc145232976)

[Introduction 3](#_Toc145232977)

[Heading Level 3 3](#_Toc145232978)

[Heading Level 4 3](#_Toc145232979)

[Appendices 4](#_Toc145232980)

[Appendix A - Document Information 4](#_Toc145232981)

[Images 4](#_Toc145232982)

[Tables 4](#_Toc145232983)

[References 4](#_Toc145232984)

[Review Distribution 4](#_Toc145232985)

[Audience 4](#_Toc145232986)

[Structure 4](#_Toc145232987)

[Diagrams 4](#_Toc145232988)

[Terms 5](#_Toc145232989)

## Introduction

BOSSCARD/ RAID: Background [], Objective, Options, Scope[In/Out], Stakeholders [Users], Constraints, Assumptions, Risks, Dependencies, Decisions, Deliverables.

### 

Developers are provided a governance framework to develop at pace, without resorting to external Governance.

Developers – as a group -- must self-direct their work using the guiding Principles that impact the solution.

## Principled Design

The Principles that the architect must be guided by and adhere to where possible and appropriate include:

* AoG obligations:
  + NZ Privacy Principles
  + NZ Digital Standards Principles
  + NZ Digital Service Principles
  + NZ Data and Information Management Principles
* This organisations obligations:
  + Architecture Principles
  + Integration Principles

These principles are outlined in an earlier section, as well as the Appendices.

## Principled Development

The number of principles that a developer must adhere to are fewer:

* **Principled Design First:** Developers must understand that the system designer is constrained to be guided by the above principles and not make changes to designs without consulting the system designer.
* **Automated Test First:** Developers must not develop new functionality before developing the automated tests to demonstrate their correct functioning.
* **The Project Principles**: the developer must adhere to the following previously listed Principles:
  + Pipeline First
  + Secure First
  + Web Interface API First
  + Multiple Channels First
  + Universal User First
  + Smallest Visual Interface First
  + Succinct, Summarized, Actionable, Forgiving, Views First
  + Forgiving Behaviour First
  + YAGNI – as long as it is not used to defer required work.

### Technical Decision Registry

Deviations from the above guiding Principles should be very rare (or else the Principles need to be updated) must be recorded in a Technical Decision Registry.

Thumbs up sign  
**Tip:**  
It is recommended to make the Technical Decision Registry an easily accessible section of the project ALM’s Wiki.   
  
Keep in mind that it may need to be sometimes formatted to print and present to Governance boards.

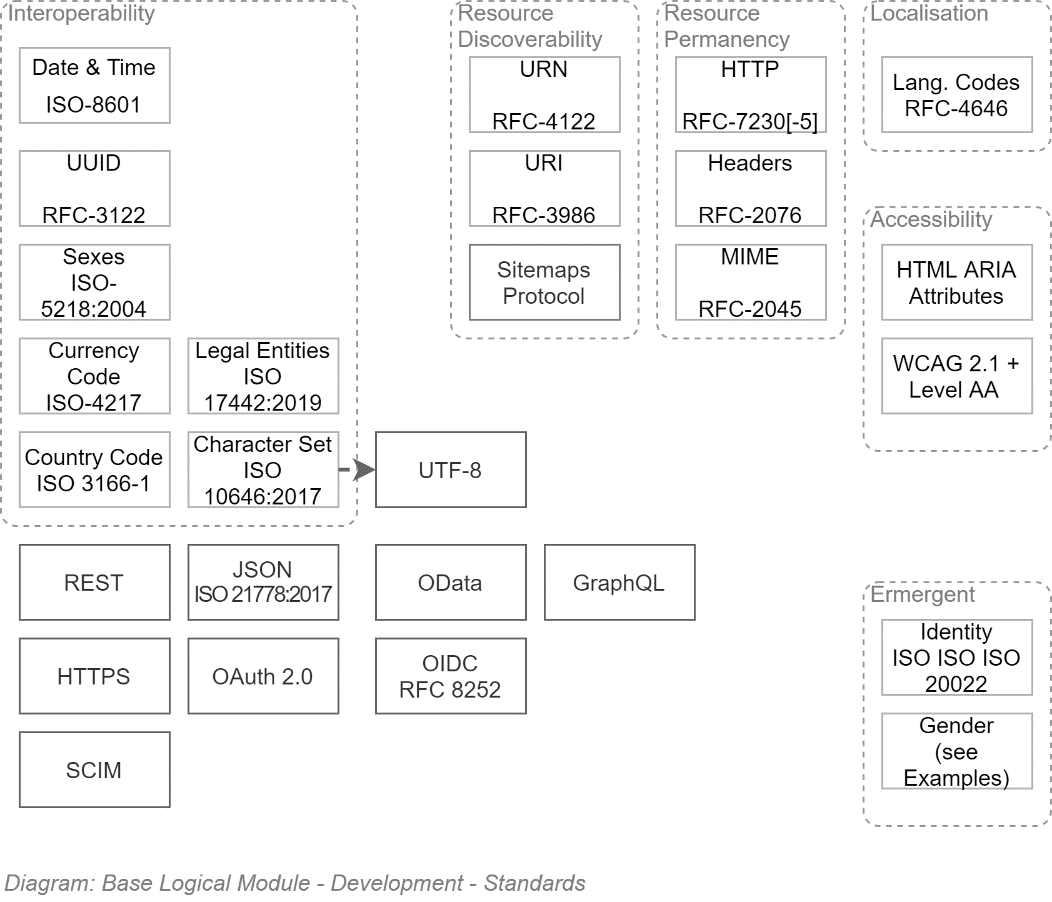
### Standards

As a government agency we are mandated to always use international standards before inventing home-brew solutions.

As per the NZ Digital Standards:

* *“Standards compliance is consistently applied. Agencies are obliged to responsibly self-regulate standards compliance. Where policy demands, an agency or a supplier must comply with a standard and provide evidence to verify or certify compliance.”*
* *“Adopting international standards should be first. If one is not suitable, consider a NZ profile of an international standard, rather than developing local NZ or NZ government standards.”*

The following are core standards expected to be used within any and all system used within this organisation.

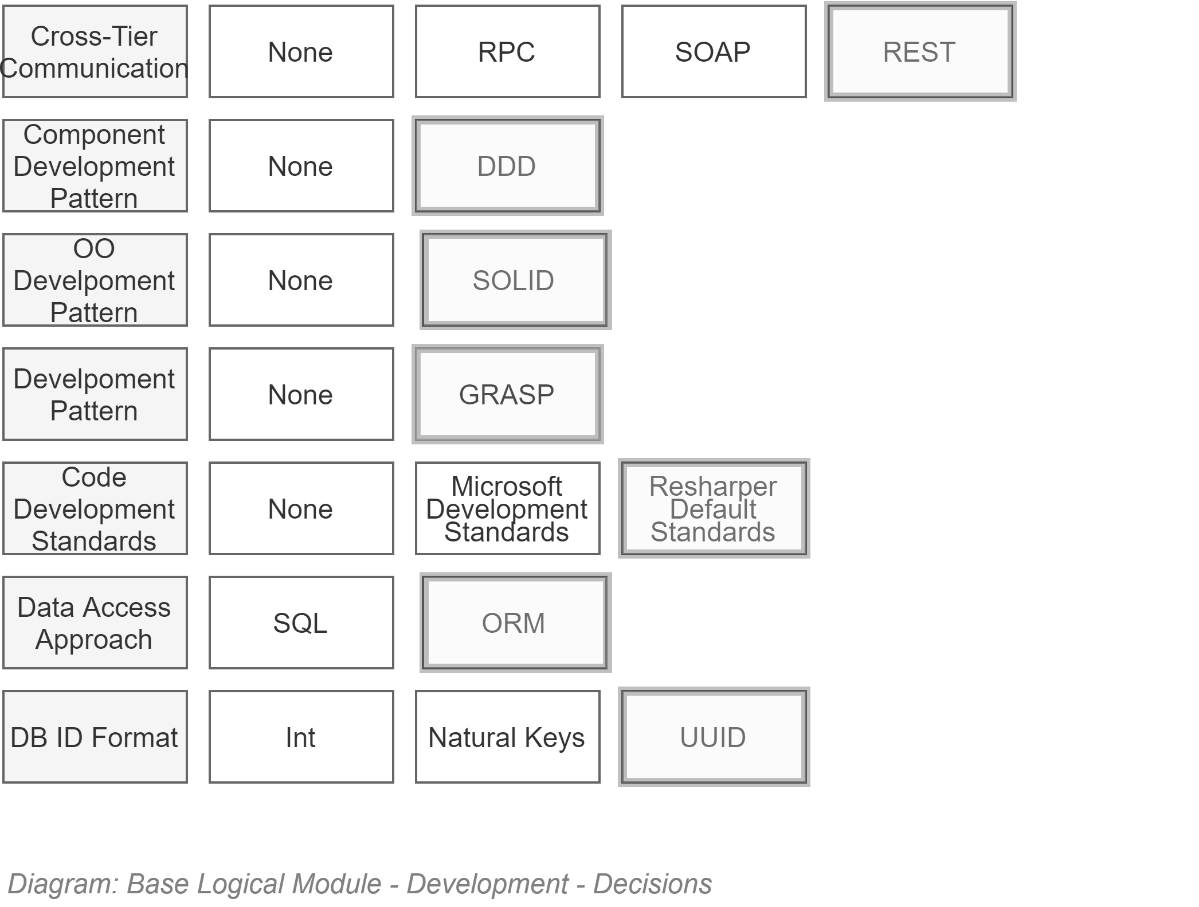


The above is less difficult that it may first appear. At a high level, the gist of above diagram is:

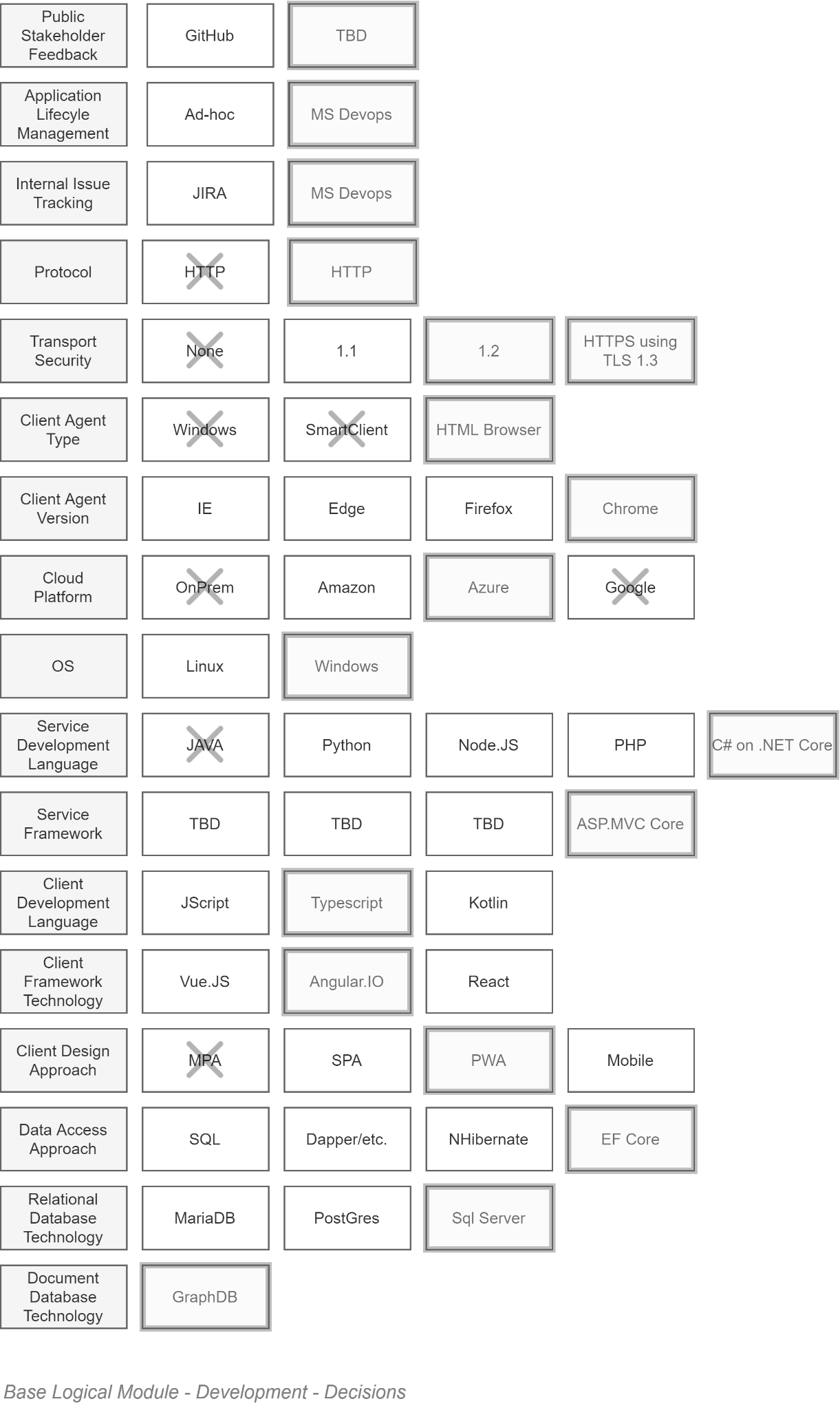
* Use *Application Logic* (*not* Database!) generated UUIDs for IDs
* Persist values using ISO standards.   
  Avoid home brew solutions for values (e.g.: "NZ". Or "N.Z." Or "New Zealand" all look like reasonable decisions, until systems need to synchronize and match against 3 different possible values)
* Persist text without stripping macrons, umlauts, etc. Accept international characters.
* UTF-8 is the standard for web pages now. Don't use anything else.
* Use the most precise Mime definition available (JSON is not TEXT, for example).
* Use international standards to make UIs accessible to visually impaired users.
* Use OIDC to authenticate Users. Use OAuth to authorise systems.
* Use OData over REST over HTTPS for APIs. GraphQL. Is an acceptable duplication of OData (although not as a replacement).
* And ensure Support staff can provision lots of users and groups, using SCIM.

Really not difficult at all once you get into it (SCIM probably can be done just by buying a library to help out).

#### Protocols



#### Development Technologies



TODO: Fix Diagram (HTTP/HTTP…)

Organised by deployment Tier, the above is as follows:



#### Maintenance Latest Version as Standard BAU

For maintainability, interoperability and security reasons the system must be maintained to update to the latest version of each standard when 20% of the market is using it, or within a year – whichever is shortest.

These updates must be a BAU operation, and not require a request from project stakeholders to initiate the work.

## Constraints

### Regulations

TODO

### Principles

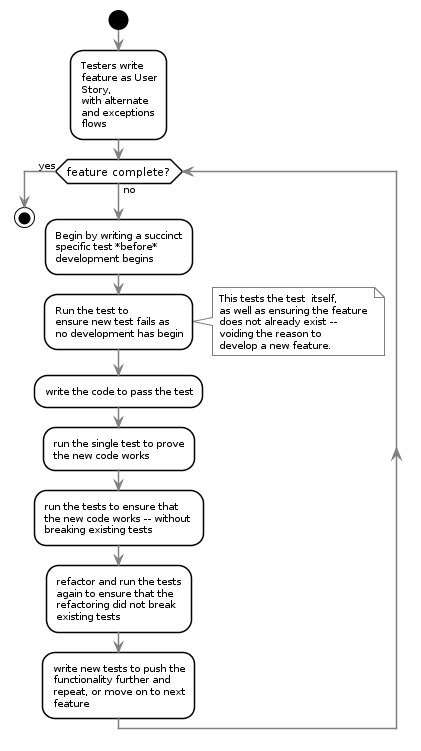
TODO

## Best Practices

### Test Driven

Developers MUST follow an automated Test-Driven Development process.

The following is an example workflow:



### Documented

Source code must be documented to facilitate reviews as well as decrease the risk of the market being unable to deliver capable maintenance specialists to take over and continue the work.

**Note**  
A long-running debate has existed in the development world, implying that code comments within code is not required if the code is clean and self-describing. The argument is idealistic and simply wrong. Code is written to communicate with computers. Comments are written to communicate with humans. You would not say that a book written in French, if written well enough, should not require assistance to allow an English person who is not 100% fluent to read it). Secondly, there are many types and levels of stakeholders in a project, not all access the code. Produce documentation of what is going on, elsewhere.

### Loose Coupling, High Cohesion

Where possible, use SOLID development patterns – specifically Liskov's Substitution and Dependency Inversion design principles.

**Note:**In C#, this is implementable as:  
- Using a class' constructor to inject– typed as Interfaces -- any service dependencies needed.

### Real-time Notification over Delayed Batch notification

Where applicable, prefer real-time notification messages between systems of changes, over the use of delayed scheduled updates (i.e., no reliance on after hour “fat” ETL operations).

**Note:**Use Queued Command patterns to process batch or CPU costly operations as close to real-time while limiting the impact on more prosaic transactional operations.

## Protocols

## Development Patterns

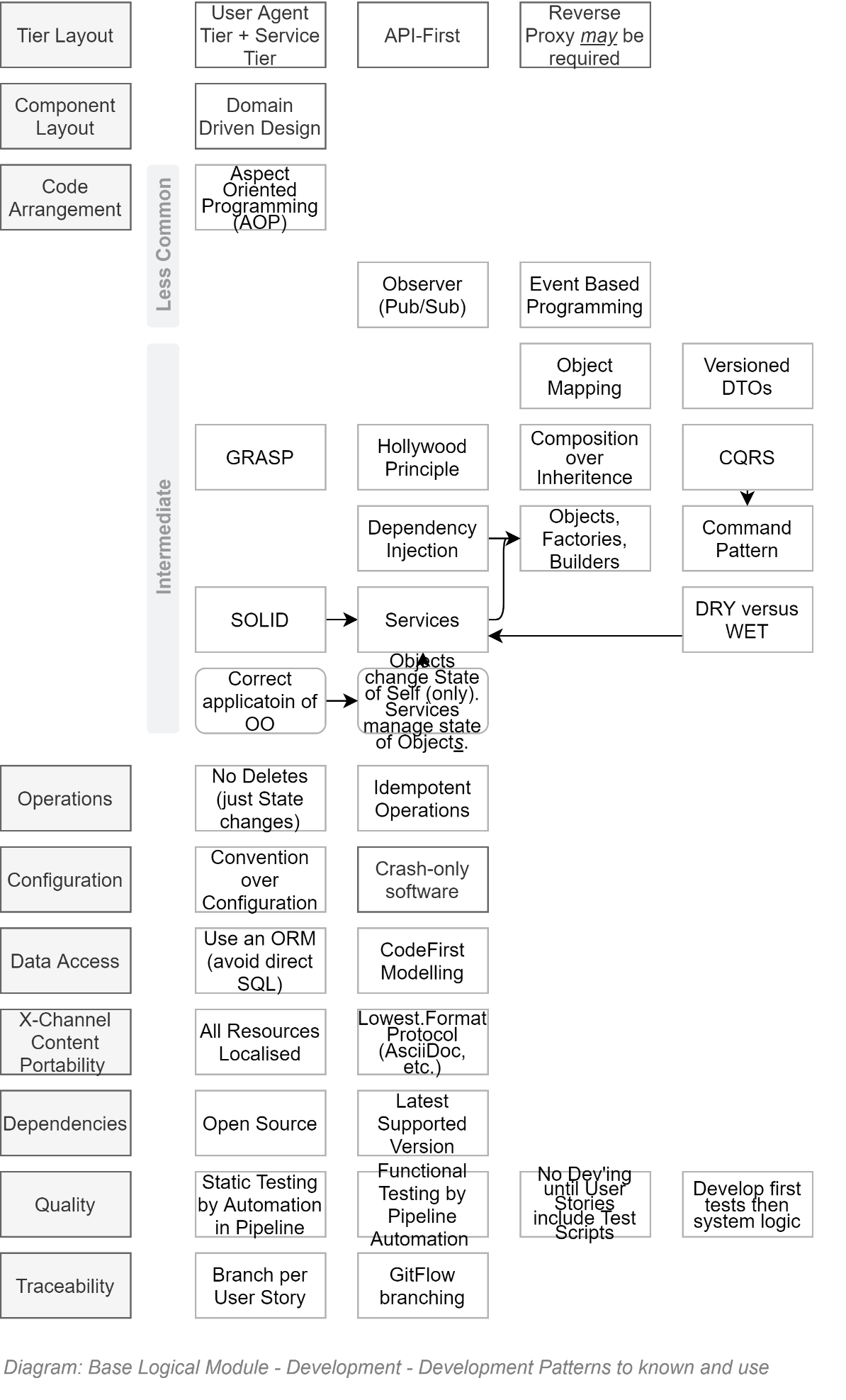
Distributed Version Control Service

The use of a distributed Version control system is a key productivity driver.

Competent ALM Services – as referenced in the Delivery View –providing

Use of

Continuous Delivery Service



### Pull-Requests

The Agile Manifesto was put together by top-notch developers. Even for them – and certainly for everyone else -- review of code submissions is a valuable process to protect the code base.

Pull-Requests for Review of changes must be issued before branches are merged into target branches.

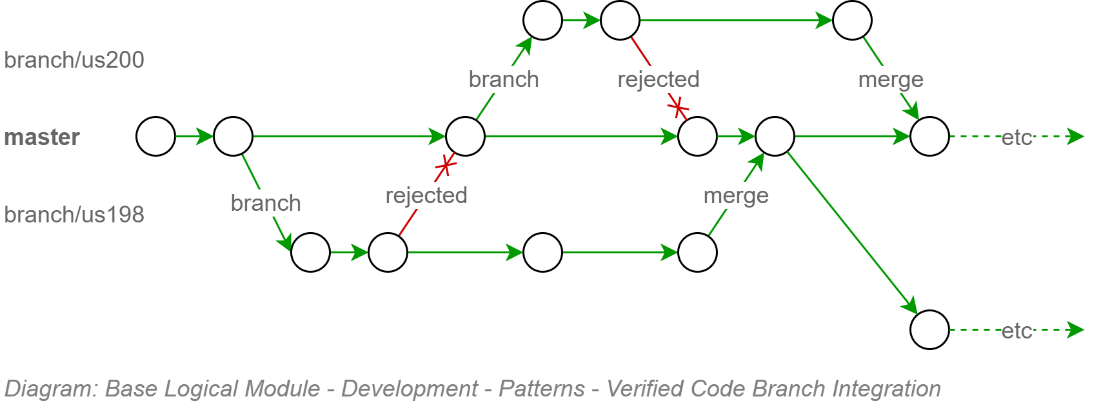
Mature code integration pipelines can be configured to automate reviews of code formatting, code complexity, unit testing, functional testing, etc – basically taking care of the minutia -- before the submitted code is reviewed by a human team member for opportunities to optimise the code.

**Note:**   
if the continuous pipeline has been configured to automate the review of format, complexity, unit testing, and functional testing, review by a human can often be skipped – most apps are not required to be race cars (they really just need to be functionally correct, while meeting quality expectations).

### Verified Code Branch Integration

The Version Control Service employed to deliver the solution performs \*Continuous Integration\* activities and verifies submitted code feature branches before integrating the code with the protected `master` branch.

If the Version Control Service rejects the code due to it failing tests -- or a `Pull Request` reviewer (see elsewhere in this View) has manually rejected the submission -- the developer has to fix it and try again before the Version Control Service will allow the submitted feature branch to be integrated with the protected `master` branch:



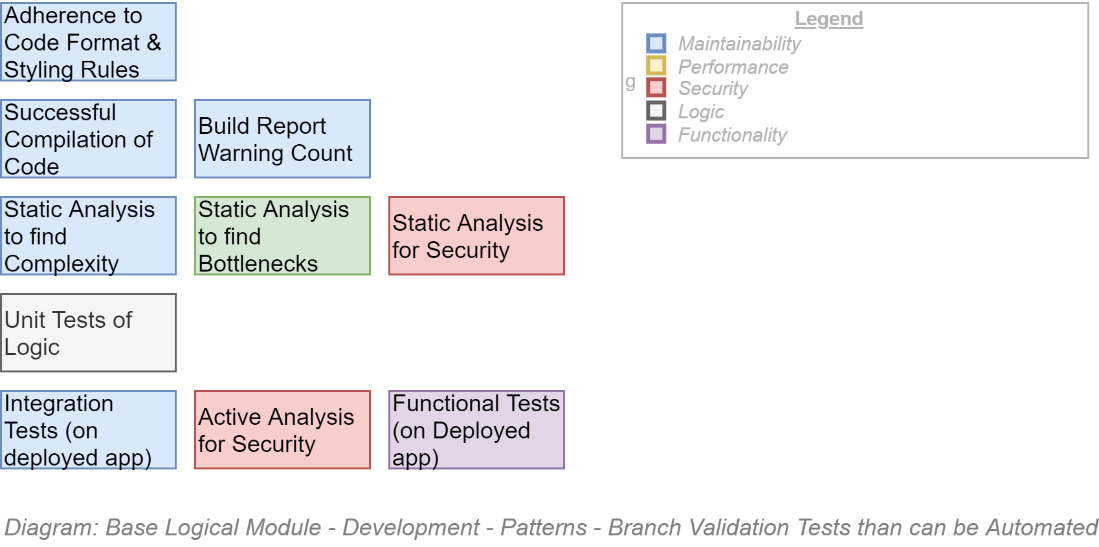
This upfront effort configuring the continuous integration pipeline to protect the code base from getting polluting, protects other developers from downloading poor code and working around it – only to have to remove the work arounds when the original poor code is fixed.

**Tip:**  
This is no different than the kinds of real-world safeguards that would be put in place if we were talking about a communal water reservoir, with everyone tasked with filling it up, and everyone drinking from it. Unless catching typhoid was an objective.

### Automated Tests

Static Tests can be performed on the source code itself, or the components created by compiling the source code. Dynamic tests can only be applied once the components have been deployed to a target environment.

The following categories are recommended for testing by automation.



The project is not expected to automate every test possible before beginning to develop the information system – but the project manager and dev team lead are expected to put aside resource time to implement in an ongoing continuous manner till they are completed.

**Note:**  
The importance of these automated tests are two-fold:   
- ensuring the quality of code during the main upfront development phase,  
- putting in place before handover to different (support) developers the automated safeguards required to ensure the initial quality of the product does not degrade over service’s whole service lifecycle duration.

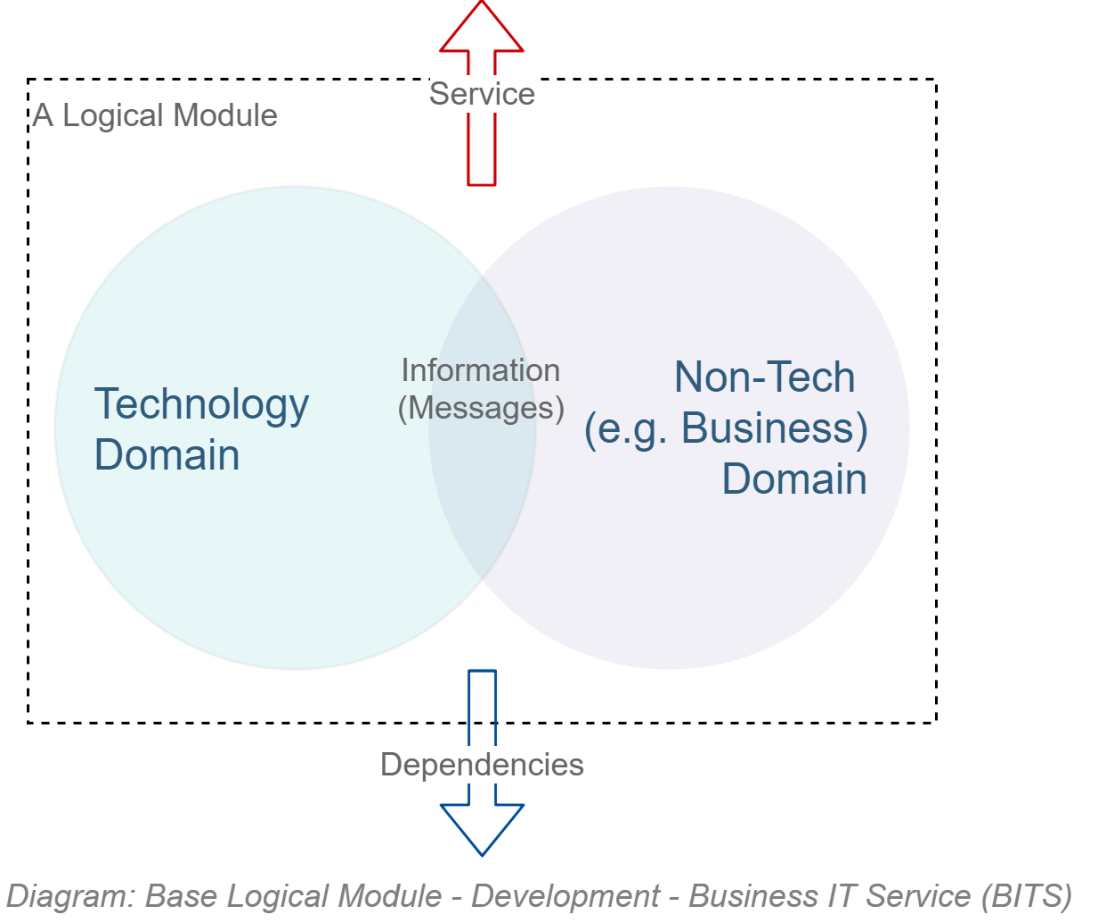
### Component Design Patterns

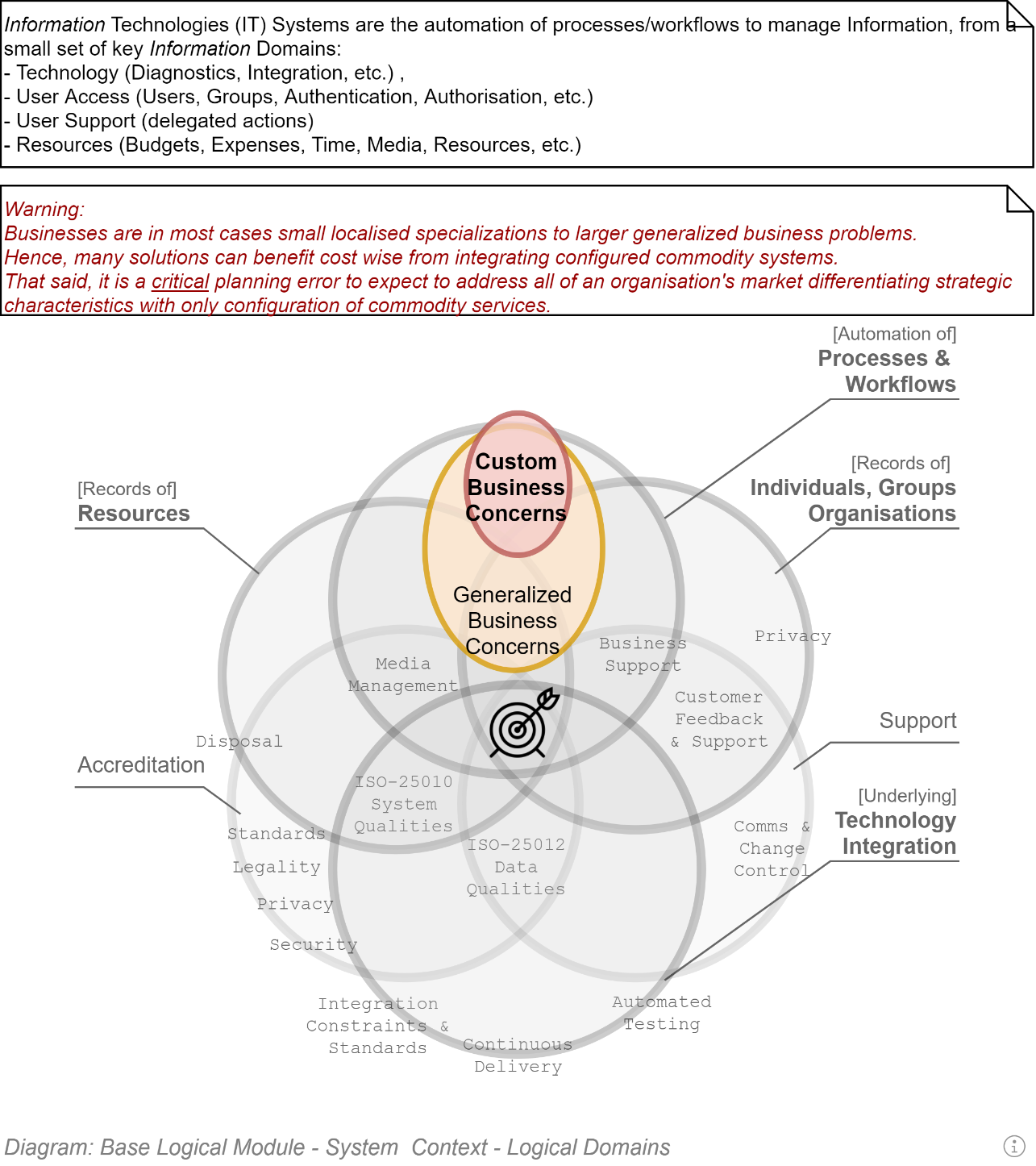
Domain Integration (DI) Service Design

Domain Integration is the recognition that Services are the composite result of integrating multiple Domains around their unique information.

#### BITS

A Business IT Service, in its simplest form, can be envisaged as the merging of two domains:



Reality is a little more complex: the actual number of domains involved in an IT product is higher:  


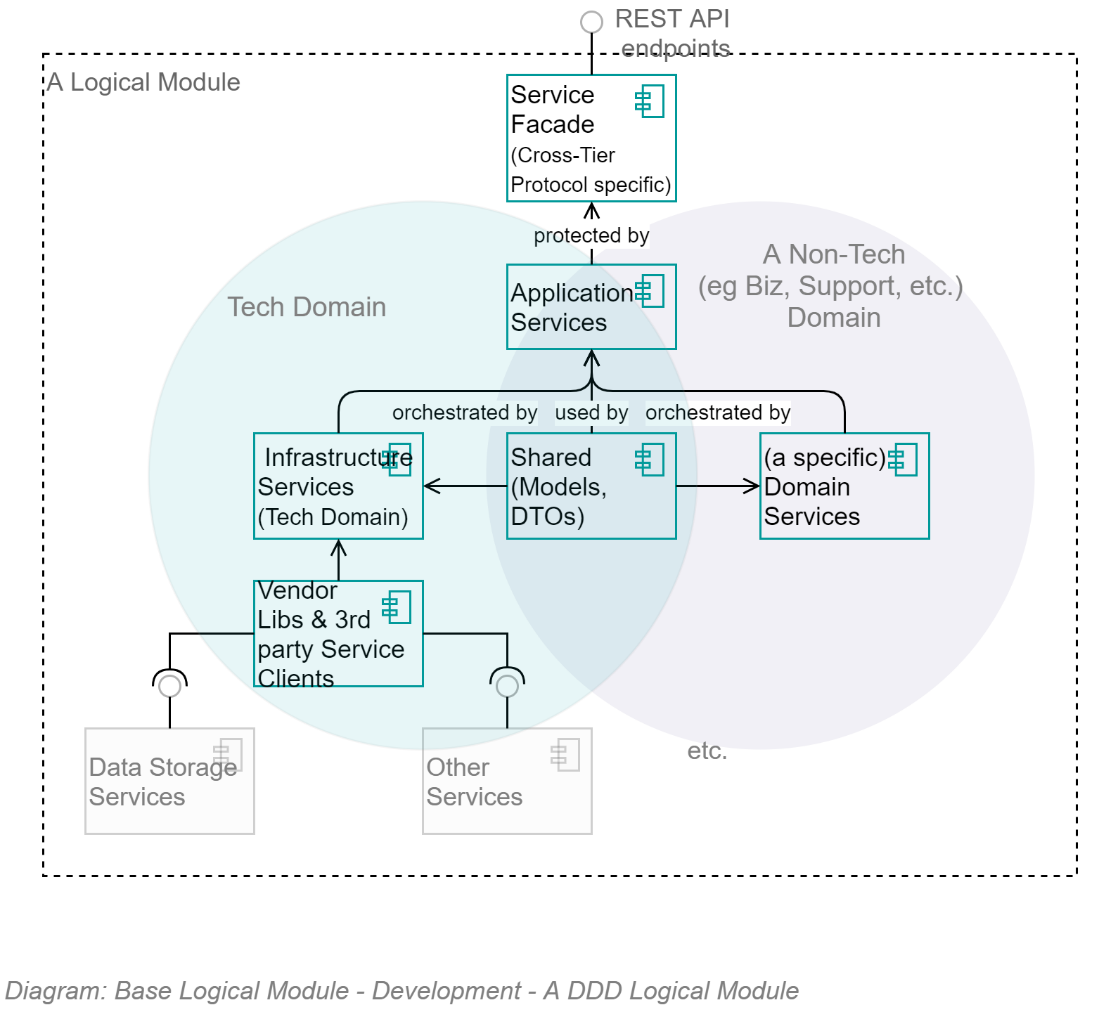
But that just means a couple more Logical Modules.

#### Domain Driven Development

Domain Driven Development is a well-established and successful approach to developing OO systems.

It recognises that there are disparate domains that MUST be kept distinct -- to not become one big unmanageable indistinct mess of logic across multiple domains – and handle the orchestration between them in a thin way.

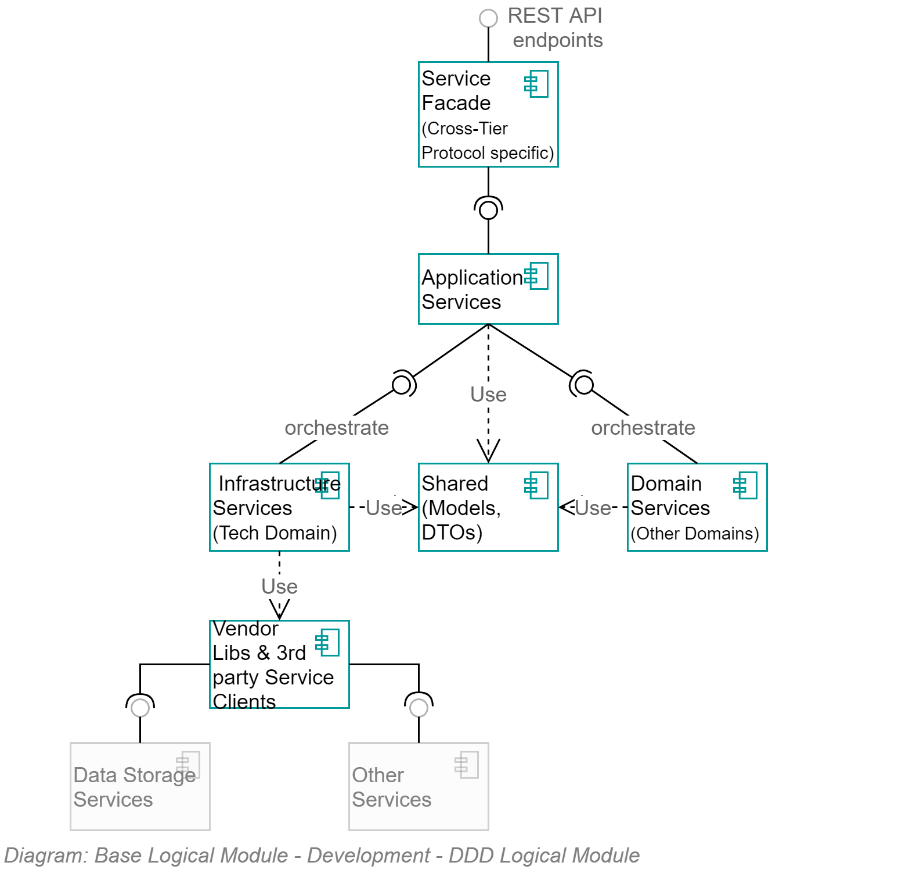
**Note:**  
Domain Driven Design is more holistic than the above short introduction to it and is well worth reading Eric Evan’s seminal work on the matter.



The above diagram demonstrates that, as per Domain Driven Development component composition pattern, all logical Modules are composed of 5 key parts:

* **Infrastructure Services:**   
  a set of infrastructure/technology domain services.  
  Note that a key benefit of the Infrastructure Assembly isolates the rest of the application from dependencies on 3rd party open source and vendor libraries by providing system specific wrappers that provided simplified, app-specific methods, returning to the rest of the system app-Specific messages.
* **[Non-Infrastructure] Domain Services:**  
  A set of (business or other non-infrastructure) domain services
* **Application Services:**  
  A set of Application Service over both of them, orchestrating calls between the two distinct domains, while keeping them separated
* **Shared Entities/Messages:**  
  A shared set of messages/entities that all three sets of services can be passed between them.
* **Service Facade Controllers & DTOs:**  
  A service facade to expose the Application Services to external users as APIs and DTOs.

The same information – this time in a straightforward classic UML Component diagram -- is as follows:



### Object Oriented Development

DDD builds upon and therefore has a dependency on developers applying correct Object Oriented (OO) development patterns.

Object-Oriented development means, in essence two things:

Object definitions are used to create Instances which have public and/or private properties and optional methods to manage their ***own*** properties. Objects do not invoke external services or change the properties of other Objects.

Vice versa, stateless singleton instances of Services definitions are used to manage multiple **other** objects (think of them as specialised helper classes).

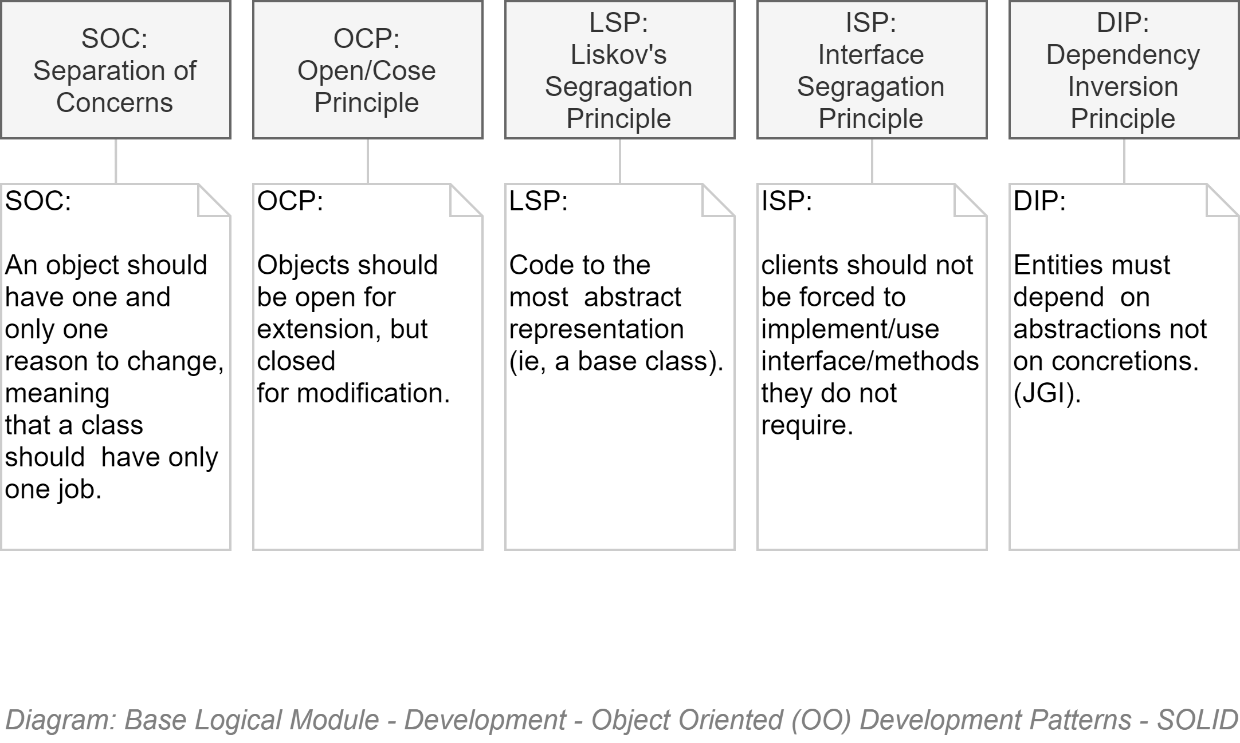
This clarity of each object type’s role is easily demonstratable as follows:

* A bank account object has a read-only Balance property and two methods: AddAmount(amount), RemoveAmount(amount) -- but no TransferTo(accountEntity) method as that would be meddling with another Entity. It also has no Save() method, as that would mean it is invoking/has a dependency on another class, in this case a Service.  
  Note: the ActiveRecord pattern is not conformant with OO theory.
* A BankingService would have no balance value, as Services are Stateless, but would have a Transfer(fromAccountEntity, toAccountEntity) to invoke the RemoveAmount(amount) on one and the AddAccount(amount) on the other.

### SOLID Object Oriented (OO) Development Patterns

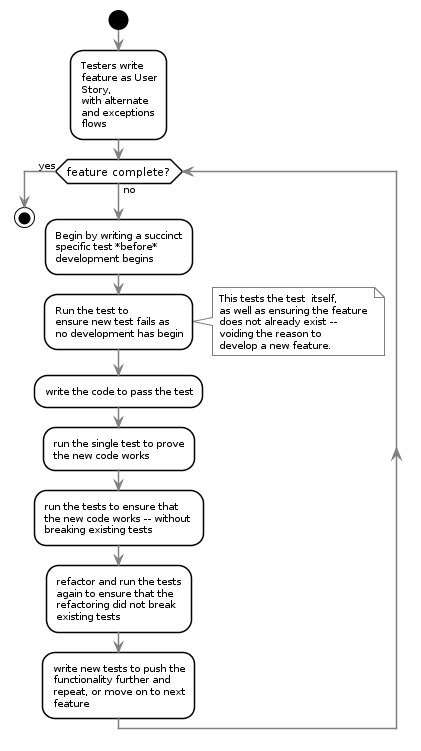
Robert C. Martin’s SOLID development patterns decrease complexity, improve maintainability and decrease the risk of non-delivery to expectations.

Irritant  
**Prohibition:**  
Developers that do not adhere to these development patterns for OO development MUST not be permitted to work on the project until they remedy this lack of experience and technical maturity.  
  
It’s a MUST, and not a SHOULD, as it is that crucial to the modularity, reuse and maintainability of any custom development.



### Test-Driven Development

TODO



## Technologies

The technologies used depend on the development platform chosen.

The following are those expected if the development platform is .NET Core

### .NET Core Based Development

TODO:

## Stakeholders

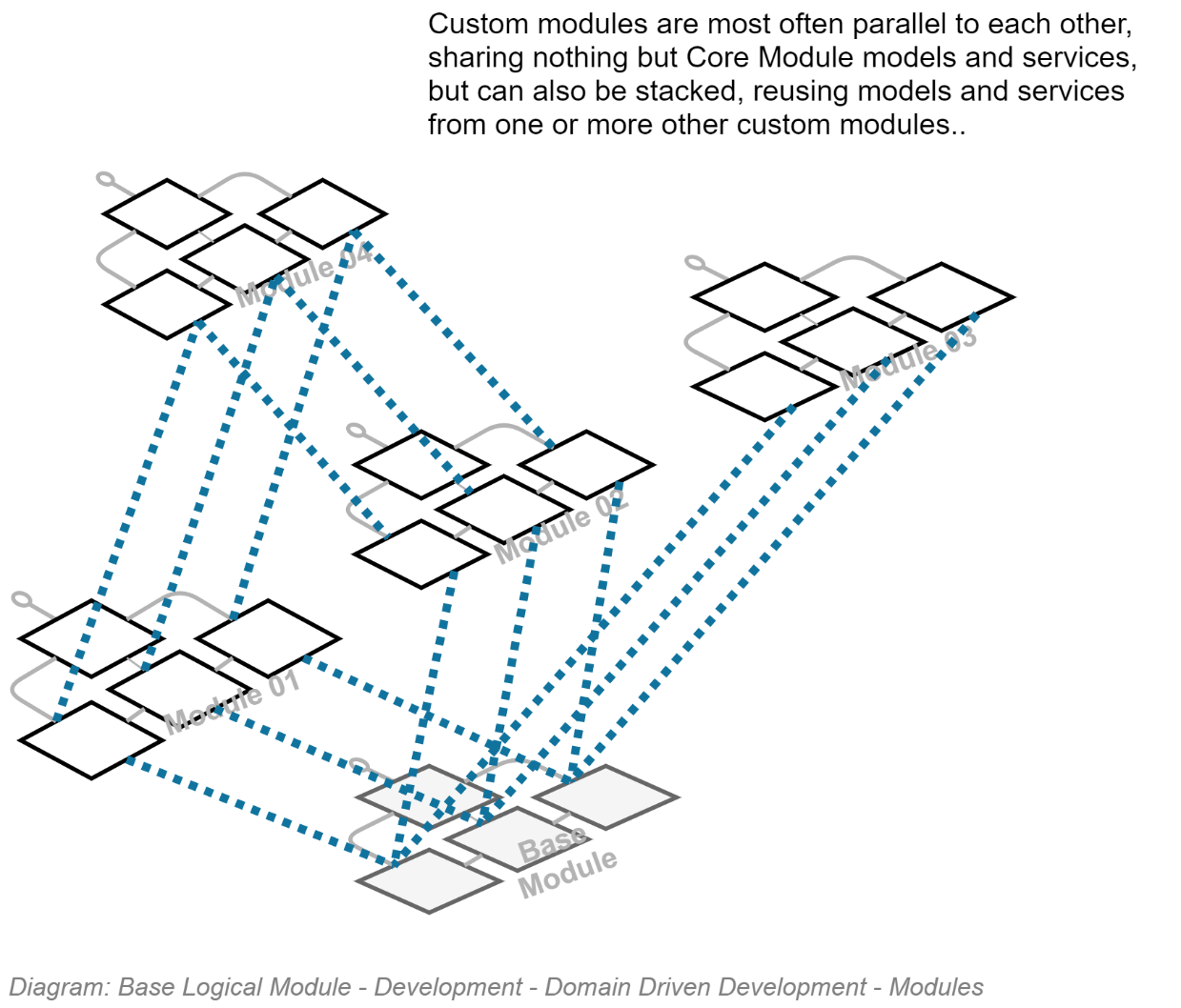
### Testers

## Components

### Packages

* Modules are a set of assemblies arranged in the classic DDD formation of the following:
* App.ModuleX.Application
* App. ModuleX.Infrastructure
* App. ModuleX.Shared
* App. ModuleX.Domain

**Tip:**  
The Base Module generally has lots happening in its Infrastructure Module and little in its Domain Assembly – whereas other Modules (Business Modules) have generally little to nothing in their Infrastructure Assembly, and more happening in their Business Domain Assembly.

* A solution is composed of a Base Module, on which multiple Business Modules can be developed in parallel:
* 

### Classes

## Security

### Configuration Information

### Service Integration Credentials & Key Vaults

### In Transit

As per the Design Principles and summarised in the Security View, all communication between internal and external devices must be secured by encryption. Therefore, all HTTP based traffic must over HTTPS, maintained by BAU to use the latest version of the TLS standard.

### OWASP

The Open Source Foundation for Application Security Project (OWASP) has developed the API Security Top 10[[1]](#footnote-2). The latest version at this point in time (2020) is the 2019 version, the IDs of the risks being:

* API1:2019 Broken Object Level Authorisation
* API2:2019 Broken User Authentication
* API3:2019 Excessive Data Exposure
* API4:2019 Lack of Resources & Rate Limiting
* API5:2019 Broken Function Level Authorisation
* API6:2019 Mass Assignment
* API7:2019 Security Misconfiguration
* API8:2019 Injection
* API9:2019 Improper Asset Management
* API10:2019 Insufficient Logging & Monitoring

## API Development

### API Versioning

API Version information is an essential element of a complete URI.

Requests using an URL that does not include an API Version number in the URL must return an error HTTP code.

Information  
**Note:**  
The use of a term such as “LATEST” [version] is **not** an acceptable solution, as client systems will stop working as expected upon breaking changes in APIs.

Bullseye**Requirement:**  
Older versions of APIs MUST be kept functioning in parallel for a published period (e.g.: 6 months) to allow service clients time to update their code to consume the latest version.

### Notification & Redirection

Bullseye  
**Quality Requirement:**  
Older versions of APIs must indicate to consuming service clients that they are not consuming the latest version of an API.   
The rationale is that although older versions of APIs are to be kept running in parallel for a duration, this state is not permanent.

Use HTTP headers to return:

* A non-200 HTTP response code.
* A single word enumeration value to indicate the state of the API (Draft, Latest, Superseded, Removed, Retired).
* The versioned URL of the latest version of the API (if Superseded).

Information  
**Note:**As usual when an international standard is not yet available to guide development, avoid re-inventing the wheel, research for the latest best practices, present and obtain approval from the lead architect.

## Services

### Base Infrastructure Services

* The core infrastructure services offered within the Base Module, for use by itself -- and Modules dependent on the Base module -- follow classic DDD recommendations and include (but not limited to):
* **ContextService:** a service to manage the current operation’s context.
* **ImmutableHostSettingsService:** a service to manage immutable settings of the host device (wrapping access to web.config).
* **DiagnosticsTracingService:** a service to trace rolling diagnostic exceptions.
* **ExceptionManagementService:** a service to manage and record exceptions.
* **LocalizedCachingService:** a service to cache resources closest to use, in the format closest to use, by the user agent’s culture-region code.
* **ObjectMappingService:** service to manage the mapping of application entities to versioned DTO objects, and back again.
* **ValidationService:** a service to manage the object and property validation of objects.
* **AuthorisationService: a service to manage access operations by principals.**
* **BlobStorageService:** provides managed control of private and public Blob Storage.
* **TenantManagementService:** a service to manage Tenants.
* **PrincipalManagementService:** a service to manage System Principal records.
* **PrincipalRoleManagementService:** a Setting to manage relationships between System Principals and System Roles.   
    
  Important:   
  May rely on external services (e.g.: EOI, Attributes, Relationships) to provide hints as to how to allocate roles – but role allocation is per system (not monolithic/centrally controlled).
* **SessionManagementService:** a service to manage one or more current active sessions per Principal.
* **SessionOperationService:** a service to persist untamperable auditable records of Principal initiated operations within a Session.
* **MutableSettingsService:** a setting to sync across hosts mutable Settings.
* **MediaUploadService:** orchestration service to manage the upload of media.
* **MediaMalwareVerificationService:** validates uploaded media.
* **MediaMetadataService:** develops a metadata object to describe uploaded media (mimetype, extensions, dates, whether contained malware, whether it was persisted, etc.).
* **RepositoryService:** provides managed access to relational storage.
* **UnitOfWorkService:** a service to manage batch persistence across tiers.
* **NotificationService:** a service to manage notifications to and among system users.
* **IUniversalDateTimeService:** a service to return UDT datetimes.
* **ISMTPService:** a service to deliver outward bound email notifications.
* **IDbContectPreCommitService:** a service to intercept database operations to ensure cleanup and logging is handled centrally.
* **IConversionService:** a service to convert values from one type to another.
* **ITelemetryService:** a service to manage the collection and recording of telemetry.

#### Service Design

* This is a living document: as time permits further technical definitions and design of the above services will be included.

## Service Façade

The Service Façade exposes the following appropriately queryable ODATA REST Controllers.

The following list of APIs mirrors what is expressed in the Service Interoperability View.

### Base Module

* ConfigurationStepRecordController
* DataClassificationController
* ExceptionRecordController
* MediaMetadataController
* NotificationController
* PrincipalController
* RoleController
* SearchController
* SessionController
* SessionOperationController
* SystemDeveloperController
* SystemDocumentationController
* SystemInformationController
* TenantController

Note:  
The APIs are appropriately accessible – depending on the controller and data type, information will be made available depending on the Authorisation granted to the current Session’s Principal.

## Business Module's Service Facade

* The APIs that are exposed via a Business Specific Module Service Façade are addressed in a separate – thin -- SAD deliverable.

Appendices

Appendix A - Document Information

### Versions

0.1 Initial Draft

### Images

### Tables

### References

**There are no sources in the current document.**

### Review Distribution

The document was distributed for review as below:

|  |  |
| --- | --- |
| Identity | Notes |
| Russell Campbell, Project Manager |  |
| Sandy Britain, Enterprise Architect |  |
| Rodney Snell, Business & Technical Lead |  |
|  |  |

### Audience

The document is technical in nature, but parts are expected to be read and/or validated by a non-technical audience.

### Structure

Where possible, the document structure is guided by either ISO-\* standards or best practice.

### Diagrams

Diagrams are developed for a wide audience. Unless specifically for a technical audience, where the use of industry standard diagram types (ArchiMate, UML, C4), is appropriate, diagrams are developed as simple “box & line” monochrome diagrams.

### Terms

Refer to the project’s Glossary.

##### IT

: acronym for Information, using Technology to automate and facilitate its management.

##### ICT

: acronym for Information & Communication Technology, the domain of defining Information elements and using technology to automate their communication between entities. IT is a subset of ICT.

1. <https://owasp.org/www-project-api-security/> [↑](#footnote-ref-2)