Application Reference Architecture

Table of Contents

Document Generation Date: 2022-04-24 22:04

[1] Martin Fowler - Bounded Context

[2] Appendix B to Directive on Service and Digital - Mandatory Procedures for APIs

[3] Martin, J. Principles of object-oriented analysis and design. (Prentice-Hall, 1993

Introduction

This document outlines the Application Reference Architecture (ARA) as it applies to our department.

What is architecture in general?

- *Architecture is the stuff you can't Google.* - Mark Richards, O'Reilly
- *Architecture is the decisions that you wish you could get right early in a project, product or project lifecycle* - Ralph Johnson & Martin Fowler
- *Architecture is about the important stuff, whatever that is.* - Ralph Johnson & Martin Fowler

The Application Reference Architecture (ARA) borders on what many would consider an enterprise reference architecture. This document, the ARA, attempts to provide an overview of the enterprise environment with a focus on application architecture elements. - Application architecture describes the behaviour of applications used in a business, focused on how they interact with each other and with users. It is focused on the data consumed and produced by applications rather than their internal structure. In application portfolio management, applications are mapped to business functions and processes as well as costs, functional quality and technical quality in order to assess the value provided." - *Wikipedia - Application Architect.* - Enterprise architecture documents the whole

architecture and all important elements of the respective organization, covering relevant domains such as business, digital, physical, or organizational; and ii) the relations and interactions between elements that belong to those domains, such as processes, functions, applications, events, data, or technologies." - *Wikipedia - Enterprise Architect.*

This document documents: - existing application architecture within our department - guidelines for technical leaders

This document is intended for: - technical design leads - technical development team

Out-of-Scope

- This document is neither a vision, nor a strategy nor a roadmap document.
- This document is neither nor a department culture nor an project management and development process document.
  - Strategy: What we will and will not do, and how govern resources.
  - Culture: People, Processes (Organization / Teams), Communication
  - Development Process: Processes, Tools

Definitions

Architecture

Application Architecture (GC EARB) Application Architecture consists of the interaction of applications with each other and with users. It focuses less on internal mechanics and specific programming and more on overall design on how data is consumed and created by the system. It views the interactions between applications, databases, middleware to ensure scalability, reliability, availability and manageability.

Application Architecture (RedHat) An application architecture describes the patterns and techniques used to design and build an application. The architecture gives you a roadmap and best practices to follow when building an application, so that you end up with a well-structured app. [Redhat - CNA - What is Application Architecture].

Application Architecture (TOGAF) A description of the structure and interaction of the applications as groups of capabilities that provide key business functions and manage the data assets. [Application Architect - Wikipedia.]

Architecture Quotes

: - *Architecture is the decisions that you wish you could get right early in a project, product or project lifecycle* - Ralph Johnson & Martin Fowler - *Architecture is*

about the important stuff, whatever that is. - Ralph Johnson & Martin Fowler - *Architecture is the stuff you can't Google.* - Mark Richards

Architecture Style (TOGAF)

: The combination of distinctive features related to the specific context within which architecture is performed or expressed; a collection of principles and characteristics that steer or constrain how an architecture is formed.

Types of Architecture The overall architecture of an enterprise can be described by integrated sub-architecture domains. These are:

- Business Architecture
- Application Architecture
- Information Architecture
- Technology Architecture
- Security Architecture
- Privacy Architecture and
- Data Architecture

Architecture Characteristics:

- Architecture characteristics are the aspects the system must do that is not directly related to the domain functionality. These are often called non-functional requirements but should be considered as Quality Requirements.
- An architectural characteristics meets three criteria:
  1. specifies a non-domain (*non-functional*) consideration,
  2. influences some aspect of the design, and
  3. is critical/important to the application's success. A few are listed below
- Examples of architectural characteristics:
  - operational characteristics: availability, business continuity, performance, recoverability, robustness, scalability, elasticity.
  - structural characteristics: configurability, extensibility, installability, reusability, localization, maintainabilty, portability, supportability, upgradeability.
  - cross-cutting: authentication, authorization, legal, privacy, security, supportabilty, usability, achievability, compatibilty, accessibility, interoperability.
  - See *Neal Ford's Presentation with List of Quality Attributes* for more information

Application

Application An application, application program or application software is a computer program designed to help people perform an activity

API An application programming interface (API) is a connection between computers or between computer programs. It is a type of software interface, offering a service to other pieces of software. An API may be a web-service call (REST API, ...) or a software library or framework (function calls, methods, libraries, ...).

Front-End & Back-End In simple application terms, the front-end of an application is concerned with the presentation to the end-user. In simple application terms, the back-end interacts with the data access layer.

Web Service An API to invoke a service over a network. Many different standards exist for web service APIs (Service Oriented Architecture - SOA and SOAP, REST API, CORBA). gRPC is newer option (2015) using HTTP and ProtoBuf; while more complex than REST APIs, offers programatic interface description language. gRPC is often used in micro-service architectures.

Governance:

Governance: Governance is derived from the Greek word kubernan "*to steer*" is an important responsibility of the architect role. As the name implies, the scope of architecture governance covers any aspect of the software development process that architects (including roles like enterprise architects) want to exert an influence upon. For example, ensuring software quality within an organization falls under the heading of architectural governance because it falls within the scope of architecture, and negligence can lead to disastrous quality problems.

Technical Debt

Technical debt is somewhat misunderstood in within IT and our department. Technical debt goes beyond having aging end-of-life applications. Below are some common definitions used to describe technical debt:

- Technical debt (also known as design debt or code debt, but can be also related to other technical endeavours) is a concept in software development that reflects the implied cost of additional rework caused by choosing an easy (limited) solution now instead of using a better approach that would take longer. [Technical Debt - Wikipedia].
- Technical Debt is a term coined thirty years ago by Ward Cunningham:

  Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still

under the debt load of an unconsolidated implementation, object-oriented or otherwise. [Ward Cunningham - 1992].

Note: Ward Cunningham is one of the authors of the Agile Manifesto

- The sum of time and effort one has to pay to keep up with the tools one is 'married' to.
- Changes required that are not completed are considered debt, and until paid, will incur interest on top of interest, making it cumbersome to build a project. As a change is started on a codebase, there is often the need to make other coordinated changes in other parts of the codebase, system, solution or documentation. Although the term is used in software development primarily, it can also be applied to other professions.
- Technical debt is a concept in programming that reflects the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution. - *Technical Debt - Technopedia*
- Technical debt is commonly associated with extreme programming, especially in the context of refactoring. That is, it implies that restructuring existing code (refactoring) is required as part of the development process. Under this line of thinking refactoring is not only a result of poorly written code, but is also done based on an evolving understanding of a problem and the best way to solve that problem.
- Technical debt may also be known as design debt.
- When taking short cuts and delivering code that is not quite right for the programming task of the moment, a development team incurs Technical Debt. This debt decreases productivity. This loss of productivity is the interest of the Technical Debt. - *Technical Debt Metaphor - Agile Alliance*

Business

The applications we develop support business models. When an application is part of a business transformation or digital transformation initiative it is important to have a clear understanding of the businesses strategic direction. Some common artifacts to communicate this are[4]:

1. Business Glossary
2. Organizational Map
3. Business Capability Model (BCM) : identify and score capabilities against good system design quality attributes {performance, scalability, stability, monitorability, extensibility, security}

[4]Hewitt, Eben. Semantic Software Design: A New Theory and Practical Guide for Modern Architects, 2020. - *ISBN 978-1-4920-4594-6*

4. Process Maps and Re-engineer Processes : Consider value streams. Model process using BPMN.
5. Define the Metrics : Identify what metrics can help assessment and reflection on desired business outcomes. These metrics must be possible to measure and communicate.
6. Understand the Governance Model: Governance is a meta-process. In your value stream, ask how decisions are made, who the authorities are, what roles they have, and what relevant review boards are. Operational scorecards.
7. Business Architecture in Applications : What business strategy does this application map to? Why does this project/application matter? What new capabilities are you creating? What major uses cases are performed? Who are the audiences?

Business Capability Model (BCM)

A common way for the business to communicate what the organization needs and does is through a business capability model (BCM)[5]. There are many uses for a BCM. Product owners can use a BCM to drive convergence in technology and business processes to enterprise standards. Regular review of aligning the BCM with the department strategy and vision can allow enterprise architects and business architects to identify and prioritize the corresponding IT initiatives with business needs. Internal committees, working groups and forums can collaborate to identify reusable business process and push for adoption across the organization. Business capabilities, processes, information flows and value streams should be assessed routinely based on efficiency, priority, and complexity.

Our department has a draft Business Capabiltiy Map (BCM) describing the main capabilities required to fulfill our mandate. To help support the business our technology teams provide a broad range of IT capabilities. Our IT department supports many networks both nationally and internationally. Within the IT department, our software development team supports an extensive catalog of applications.

Process Maps, Information Flows and Value Streams

Information Flows [6] is a business view of how information flows between business responsibility centres. *The main purpose of an information flow diagram is so that sources that send and receive information can be displayed neatly and analysed..*

Introduced in Lean (1950's) a value stream is a set of actions (workflow) to produce value [7]. Value Stream Mapping is visual tool introduced in Lean Management methodology to display the value stream with define icons to show

[5] Wikipedia - Business Capability Model
[6] Wikipedia - Information Flow Diagram
[7] Wikipedia - Value Streams

delays and inventory stages. An example value stream might be recruitment "street to seat", "hire to retire" and "procure to pay".

A Process Map[8] defines the standard business process, and who is responsible for the activity.

Business Governance

The health of our portfolio needs to improve as identified in our Corporate Risk Profile (CRP). Several leadership principles have been established over the years to provide guidance when addressing business needs. Key principles relating to governing and directing architecture and application design are:

1. **Rationalization**: With a large backlog of valuable business requests and opportunities application functionality must be constantly rationalized. During the software development phase, requirements must be rationalized against the original approved project scope and other compete business needs. The costs of increment application development, both in project costs and ongoing costs must be carefully understood. This is the process of rationalizing business needs and can include the senior management team when necessary. [*See Guidance - Rationalization for more information*]
2. **Executive Lead / Change Management**: Projects and programs need executive sponsors who are committed to the change management and rationalization required to allow IT to develop a product.
3. **Business Architecture and Artifacts**: The business plays a key role in shaping the application. Business architecture (capabilities, value streams, information flows, organization model) are essential for successful analysis of the business needs during application development. Significant architecture re-work and design waste result if these are unavailable. As our department adapts agile methodologies, incremental value in the project can be obtained by the agile team including maturing business artefacts as part of the product backlog.

Business and Technology Environment

Our Information Technology (IT) operates in a complex and constrained environment due to the sensitivity of information managed. Awareness of the legislative and departmental directives and policies is crucial at the outset of application development. A common phrase used in DevOps is to *shift-left* quality attributes like security and privacy. Key non-functionality quality requirements derived from our environment should be considered at the outset (e.g., official languages act, accessibilty act, information management). The non-functional requirements should be realistic and follow the SMART guidelines (Specific, Measurable, Achievable, Realistic, Time-Bount)[9].

[8] Wikipeida - Business Process Maps
[9] Wikipedia (SMART) Requirements

Discovery

There are many initiatives within our department that require enterprise and domain architecture effort to recommend the path forward.

1. Identity and Access Management (IdAM):
   Analyze existing identity and access management options to provide multi-domain identity and access to highly compartmentalized information.

2. Enterprise Integration & Interoperability:
   Analyze steps to mature our ability create a composable enterprise recommended by Gartner.[10]. This guide recommends creating Reference Architectures which is modular. The modules can be composed and indepently improved. "*The framework is based on the ability to assemble and reassemble various digital assets and business elements for real-time adaptability and resilience in the face of uncertainty.*". The guide identifies the need for business strategy documents, roadmaps and business architecture deliverables to inform reference architecture creation.

3. Enterprise Search:
   Gartner calls the broader enterprise search an Insight Engine. Gartner - Critical Capabilities for an Insight Engine. [Gartner Magic Quadrant - Insight Engines]. Key terms include; connectors, touch points, integrations. Popular open-source solutions like Solr and Elastic support API integrations for adding and removing content with structured-metadata. A key to the success of enterprise search is the ability to structure the index information with metadata. This enables discover and faceted searches.

- [ ] Enterprise Taxonomy : A deliverable within the Information Management Modernization program (IMmod)

1. Multi-Security Zone Applications:
   Our directorate has been asked to to move workloads to lower security zones. As a consequence business processes may span security zones. The cross-domain-solution has been identified as an enabler technology. What overall application, data, information and security architecture is needed to realize these benefits.

2. Managing Media Our department manages multimedia (images, audio and video files) as well as file-types on a diverse range of applications. Media management can be addressed be a Media Management Platform and a Digital Experience Platform (DXP). Industry leaders include OpenText, Oracle and Salesforce. While some Content Management Systems (CMS) also support DXP features many new market entrants are SaaS-based and require cloud connectivity (e.g, Sanity.io). OpenText DXP() is not in the top-magic quadrant; however it deserves consideration due to GCdocs.

- OpenText DXP

[10] Gartner- Ignition Guide to Building Reference Architectures for a Composable Business

- Opentext Why you Need a DXP: *Orchestrating a cohesive, contextual experience that meets brand standards, achieves business goals across all channels and touchpoints, while it delights the recipient, is a massively difficult task.*

Features of a DXP: - Content Management System - Media Asset Management - Digital Asset Management (media and non-media content) - Headless DXP / CMS : Provide back-end features expose media assets via API's.

TODO - reference Confluence ITOD Dependencies document TODO - Add Enterprise Interoperability to ITOD Dependencies

Application Characteristics and Styles

TOGAF defines Architecture Style as *the combination of distinctive features related to the specific context within which architecture is performed or expressed; a collection of principles and characteristics that steer or constrain how an architecture is formed.*. Depend ding on the desired architecture characteristics and different style will be chosen.

- The GC Digital Operations Strategic Plan[11] indicates the priorities for services/applications should be *developing and delivering services that, by design, put users first by being accessible, inclusive, secure and easy to use, and that respect privacy and choice of official language.* This is mostly focused at services the public consumes, versus, services and applications our internal public servants use.

Application Characteristics

As part of the analysis and design some high-level characteristics of the application should be assessed. Some of these attributes may be official documented as part of the project and application development, and others may have to be assumed or derived for requirements.

Department Application Characteristics

[11] Gregor Hohpe and Bobby Woolf, Enterprise Integration Patterns (Boston: Addison-Wesley, 2003).

Attribute	Description	Note
Criticality	How critical is this application to the business. This is sometimes referred to as Tier-1, 2, 3.	The department lacks an official list of application criticality. Based on criticality, and TBS guidance, critical applications must have certain quality components like a business continuity plan (BCP) and a Disaster Recover Plan (DRP). This need to maintained and practised like fire alarms on a regular basis.
Security Profile	Based on the security triad of Confidentiality, Integrity and Availability (CIA) and indicating the impacts of integrity and availability to the organization (High, Medium, Low).	Common profiles are PBMM (Protected-B, Medium, Medium) and TSHH (Top Secret, High, High). The security profile can help guide development of quality requirements (non-functional requirements)
Information Classification	What classification of information is managed by the system	Unclassified, Confidential, Protected A/B/C, Secret and Top Secret are common security classifications
IM Repository Type	Identifies whether the information in this system is transitory or corporate.	Based on the repository type additional requirements relating to managing the information through its lifecycle are required. Reference Guideline on Service and Digital

Attribute	Description	Note
Information Business Type	Our department treats operational information different from administrative information.	The distinction is unclear, and there are few guidelines to help projects to help manage this distinction. Applications are categorized as managing operational or administrative information. For example, CW is administrative, CWOPS is operational (however only extremely limited operational information is permitted in CWOPS).

Quality / Non-Functional Characteristics

Identifying the key quality attributes of the system is required to chose an effective architecture style. Trade-offs between complexity, scalability, observability, reliability and other attributes is required. No single architecture style is suitable for all applications.

Application Architecture Styles

Architectural style is defined as a set of characteristics and features that make a building or other structure notable or historically identifiable. Architecture styles are been established and evolved over the years. Some common application architecture styles are[12]:

- Distributed: Microservices Architecture : pros (reliablity, modularity, elasticity, +++), cons: (cost, complexity, ...)
- Distributed: Orchestration - Service Oriented Architecture (~2005) :
  - pros: (good elasticity, fault tolerance, scalability), cons: (complexity, testability, cost, ...).
  - cons: A big weakness of SOA was the use of a common platform for all services deployed (e.g., Oracle SOA Suite, IBM WebSphere, DataPower, MessageBroker). SOA also required stateful services and

[12]Building Microservices - Sam Newman

sharing of context (tight-coupling).

  - note, SOA promised loose-coupling, scalability and fault tolerance Josuttis, N. M. SOA in practice. (O'Reilly, 2007)] however these were difficult to achieve with SOA.

- Distributed: Event Driven Architecture : pros (fault tolerant, modular, good cost), cons: (complexity, testability,)
- Monolithic: Layered: 3-tier/N-Tier/Client-Server : pros (simplicity and cost), cons: (scalability, fault tolerance, deployability, testability, modularity)
- Monolithic: Pipeline: pipelines & filters : pros (simplicity and cost), cons: (scalability, performance, . . .)

Application Architecture Guidance

Goal: Reduce Technical Debt

1. Rationalization.

In cooperation with the business, business governance and other stakeholders development of functional and non-functional requirements must be rationalized. There are many strategies to rationalize development to ensure the project can be completed on time, in an agile manner. Some strategies for recognized industry leaders include:

- "Reduce features, focus on the priorities" - Basecamp - Feature Selection - Start with No
- Apple had suffered from lousy engineering management and Steve Jobs was answering a negative question about a removed feature. Steve Jobs response was that "*Focusing is about saying no. When you say no, you piss off people.*"- Steve Jobs - Focus on Saying No - 1997:
- Are we staying true to the vision? : Basecamp - Priorities - Whats the Big Idea:
- Prioritize, Focus on what you really want to deliver), Flexibility : Scope flexibility. It's better to make half a product than a half-assed product. - Basecamp - The Starting Line - Fix Time and Budget, Flexibility on Scope:
- *How does a project get to be a year behind schedule? One day at a time.* - Fred Brooks 1979 Software Project Management - The Mythical man Month. the mythical man-month - 1975 - isbn

1. Reuse / Buy / Build.

Prior to a business case or project proceeds to development, any new application creation should be discussed with other stakeholders (e.g, TMO - Transformation

Management Office, BRMO - Business Relationship Management Office, TRB - Technology Review Board and the AWG - Architecture Review Board). If an new application is justified, the options analysis should consider the TBS Digital Standards and GC EARB Application Architecture Standards [13]. The following priorities for options analysis:

- Reuse: Attempt to reuse what we currently own, or what other government departments / partners are using.

- Buy: Buy solutions and integrate into our enterprise architecture
- Build: As a last resort, custom build a solution. This should be limited to business capabilities and processes that are unique to our department. Executive approval (Department Architecture Review Board) required.

1. Document & Exercise Backup & Recovery

All applications, regardless or criticality, must have a documented backup and recovery procedure. This needs to be exercised on a regular basis (at least annually) and must be done prior to deployment to production.

Business critical applications require a BCP and DR plan to be documented and reviewed on a regular basis.

- [ ] Enterprise Architecture : Formally identify the criticality of applications and record this in the department's official configuration management database (CMDB). Note: *As of this writing the CMDB is not the official source of truth for the list of critical applications. The project should clearly identity if this application is critical.

1. Build Less and Stay Lean

BaseCamp has a few short-narratives on ways to stay-competitive; which can be adopted to our department attempting to reduce technical debt.

- Basecamp - The Starting - Build Less:
  - Less features
  - Less options/preferences
  - Less people and corporate structure
  - Less meetings and abstractions
  - Less promises
- Basecamp - Stay Lean - Less Mass: - less "Thick process" - less "Long-Term Roadmaps" (supported as by our ITSS Study - Ian Lovsion 2017) - less of "The past ruling the future"

[13] GC EARB EA Standards and Application Architecture

Goal: Reduce Content Duplication with URL Design and Search

Content (information) is duplicated within applications and across technologies. The causes of this have not been formally documented, however some factors leading to users copying content are the lack of *trust* in being able to find or access the content in the future. This can be paraphrased as *I need a local copy for me or my team*. This leads to copies of information on shared-drives and transitory and corporate applications.

Some historical examples that have led to this "clone-and-own" culture include:

- Link Rot: Application upgrades making links to content fail. Deep Linking is the use of a hyperlink that links to a specific, generally searchable or indexed, piece of web content on a website. For example, a link to a specific case, request or document.
- Access: Users are concerned that the content may disappear due to the content owner removing, renaming or modifying user-access. This is difficult to address at the application layer, and requires enterprise information and access-management governance.

1. URL Lifecycle When supporting Deep Linking design must take into account the lifecycle of the link, ned that the content may disappear due to the content owner removing, renaming or modifying userand its ability to function through upgrades. Consider patterns such as Permalink and Data Object Identifier (DOI). When provide a link to a user for reference, identify this should be a trusted-link which survives upgrades/replacements.
2. URL Design Define a URL strategy for the application, including an inventory or URL's provided. Define the manner in which URLs are clean, friendly and pretty Clean URL; support *http://example.com/name* as opposed to *http://example.com/index.php?page=name*.
3. Enterprise Search Enterprise search will definitely help in enabling users to find the information they should have access to. This is a major long-term initiative.

Basecamp - The Starting - Build Less [. . .] less means: - Less features - Less options/preferences - Less people and corporate structure - Less meetings and abstractions - Less promises

[. . .] Basecamp - Stay Lean - Less Mass: - less "Thick process" - less "Long-Term Roadmaps" (supported as by ITSS Study - Ian Lovsion 2017) - less of "The past ruling the future"

Goal: Composable Enterprise, Composable Applications

A composable application is a key pattern in micro-services. In our current environment and infrastructure environment, the focus should be on designing

*single purpose services" on virtual machines. Applications should be thought of as thin user interfaces on top of this collections of services. The design of the services/APIs is important to success of the project and application. Some strategies to help in the design and communication of the service-architecture are:

Domain Drive Design (DDD) / Bounded Context[14]

DDD is useful for large transformation and modernization projects like HR and ERP modernization. A bounded-context breaks the large domain into a cohesive boundary. Within this bounded-countext services can be designed and exposed. Refer to *Domain Driven Design*[15] for details on this concept.

GC Directive on Service and Digital - Standards on APIs[16]

The Direcitve on Service and Digial provides high-level guidance on API design which should be implemented: - Build APIs against the business requirements - Work with the developers who are expected to consume your API - Expose APIs using industry accepted open standards

Decouple User Interfaces

Design the web UI to work across ~~mobile devices, tablets, and~~ desktops at a minimum. Text is ~~striked-out~~ to indicate we currently develop applications for use on a known standard-provisioned desktop with two-monitor.

1. APIs, and the consuming services and applications should have *high-cohesion* and *losse-coupling*. This is especially important as software communicates across business domains. Application Programming Interfaces (APIs) should be used to reduce; especially at the high-level interactions between components.[17][18][19]. Architectural patterns to support composable applications include:

- High Cohesion: *The Fundamentals of Software Architecture*[20], in Chapter 3 on Modularity, describes how to measure modularity. Cohesion can be measured in terms of functional, communication, procedural, logical and other dimensions.

[14] Martin Fowler - Bounded Context

[15] Evans, Eric. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison-Wesley, 2004.

[16] Appendix B to Directive on Service and Digital - Mandatory Procedures for APIs

[17] Reduce Coupling - Martin Fowler IEEE 2002.

[18] Amazon-AWS Bezos API's - Expose Data and Functionality through service interfaces

[19] Mastering API Architecture

[20] Fundamentals of Software Architecture : Richards, Mark, and Neal Ford. Fundamentals of Software Architecture: An Engineering Approach. First edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly, 2020.

- Low Coupling : Use data access layers between application business logic and the database layer. Coupling is more difficult to understand and requires assessment of connascence[ˆGuidance-14}]. Some common guidelines are:
  - break system and APIs into encapsulated replacements
  - minimize any connascence (depedecny relationships between objects) between systems

SOLID[21]

SOLID is five design principles supporting composable applications:

- S - Single Responsibility Principle. Gather together things that change for the same reason, and separate things that change for different reasons. Good system design means that we separate the system into components that can be independently deployed. A class should only have a single responsibility, that is, only changes to one part of the software's specification should be able to affect the specification of the class. - *reference: 97 Things Every Programmer Should Know #76 - 2010.*

  If a class has more than one responsibility, then the responsibilities become coupled. Changes to one responsibility may impair or inhibit the class' ability to meet the others. This kind of coupling leads to fragile designs that break in unexpected ways when changed.

1. O - Open–closed principle - Software entities and components should be open for extension, but closed for modification.
2. L - Liskov substitution principle - Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program. Design by Contract.
3. I - Interface segregation principle - Many client-specific interfaces are better than one general-purpose interface.
4. D - Dependency inversion principle - One should "depend upon abstractions, *not* concretions.

12-Factor Application

The *12-Factor Applications*[22] was defined by Heroku in 2011 as a means to define attributes or a successful Software as a Service (SaaS) application with portability and resilience characteristics. These characteristics are good goals for the software architecture to achieve.

1. Codebase: There should be exactly one codebase for a deployed service with the code base being used for many deployments.

[21] Martin, J. Principles of object-oriented analysis and design. (Prentice-Hall, 1993
[22] 12-Factor Application - Heroku - 2011

2. Dependencies: All dependencies should be declared, with no implicit reliance on system tools or libraries.
3. Config: Configuration that varies between deployments should be stored in the environment.
4. Backing services: All backing services are treated as attached resources and attached and detached by the execution environment.
5. Build, release, run: The delivery pipeline should strictly consist of build, release, run.
6. Processes: Applications should be deployed as one or more stateless processes with persisted data stored on a backing service.
7. Port binding: Self-contained services should make themselves available to other services by specified ports.
8. Concurrency: Concurrency is advocated by scaling individual processes.
9. Disposability: Fast startup and shutdown are advocated for a more robust and resilient system.
10. Dev/Prod parity: All environments should be as similar as possible.
11. Logs: Applications should produce logs as event streams and leave the execution environment to aggregate.
12. Admin Processes: Any needed admin tasks should be kept in source control and packaged with the application.

Develop an API Strategy

API's are a critical component of our technology stack. As applications and technology more-and-more through API's we need to mature our API strategy. The API Strategy should address concerns such as: - API Discovery / Catalog: How can developers discover integrations (*TODO*) - API Testing: automated testing, performance testing, stubbed-out testing. - API Standards follow GC Standards on API guidance, align with NZ API Guidance & Resources & UK API Technical & Data Standards guidance. These are written to support integrated digital processes across departments and agencies; however their guidance is relevant for internal integrations.

- As we mature with our API Strategy, and enterprise approach to APIs for the following is important:
  - API Documentation(https://www.gov.uk/guidance/how-to-document-apis): discover, affordances (understand how to use API), integration with API. Examples: GOV.UK Frontend, Stripe API, Mailchimp.
  - API Protocols: Leverage protocols and languages like gRPC and GraphQL for integrations.
  - API Management: As the number of components, micro-services and integrations grow, the need for an API management layer to provide orchestration and API lifecycle management increases. API management provides a single point of entry for all connected systems and services. Helps developers (IT, client-authentication, authentication,

business-citizen) develop to APIs.

- References:
  - Wikipedia API Management
  - Gartner Ensure API Management includes Cloud and Microservices
  - Gartner Human Capital Management (HCM) Integration Strategy - 2020
  - Gartner Choose Integration Technology - 3 Patterns of Integration:
    1) Data Consistency across platforms (ERP, CRM, Billing, ...)
    2) Multistep Process / Pipeline
    3) Composite Service

    Identify integration needs (Application Integration, Data Integration, API Lifecycle Management, Integration Platform, BPM (Pega, ..), Master Data Management, Message Oriented Middleware (ESB, Streaming, ...), Robotic Process Automation(RPA)

Event-Driven Process and Streaming

Event Driven Architectures are useful for distributed, asynchronous, scalable and performant systems Leverage events as a core principle. Publish these events, subscribe to these events (streaming data flows).

- event-based data flows for batch and real-time processing
- message-oriented over transactions. An interesting video presentation on "*Why to use Events by Avdi Grimm - Nordic JS No Return: Moving beyond transactions*".

Composable ERP and HR Enterprise - Gartner

Gartner - The Future of Enterprise Resource Planning (ERP) is Composable[23] defines a *Composable ERP* as an adaptive technology strategy that enables the foundational administrative and operational digital capabilities for an enterprise to keep up with the pace of business change. This strategy delivers a core of composable applications and, as a service, software platforms that are highly configurable, interoperable, and flexible to adapt to future modern technology.

The key Gartner recommendations are valid for ERP and HR modernization initiatives: - [ ] Define business capabilities through road mapping exercises - [ ] Reduce reliance on customizations and proprietary-ERP - [ ] Continuously deliver incremental business value in a modular mode - [ ] Build an ERP team with wide-ranging skills (Adopt a DevOps for ERP). Ensure adequate skill development and training

Goal: Testability, Testable Applications and Automation

Testing applications and groups of applications effectively and efficiently requires analysis and design. The application development needs to include capabilities to

[23] Gartner - The Future of ERP is Composable

facilitate testability. Testing scopes vary based on developer testing and quality assurance testing. QA testing often involves elaborate efforts to setup a system (install, configure and provision) for a single test case.

- Automation: Adopt theMicrosoft - Shift Left Testing) DevOps of automation while shifting-left the integration and quality testing. Automation should accommodate the CI/CD concepts, as well as the ability to provision and validate tests across multiple environments. The following principles are copied from Microsoft Shift Left Testing[24]
  - Write tests at lowest level possible. Favour unit tests over functional tests. When functional tests fail, consider if unit tests should be more comprehensive.
  - Write-once, run anywhere (DRY - Do Not Repeat yourself): Tests should be written to work in any environment (Dev, Sig, Prod).
  - Design Product for Testability. Discuss how the system is testable during peer-reviews and Technology Review Board (TRB) reviews.
  - Test Code is a product. Treat the software used to automate testcases as code. The code is version-controlled, and discoverable (i.e., it exists in close proximity to the application code
  - Test ownership follows application software ownership. The software development team owns creating automated tests for not only unit-tests but boundary/integration tests.

Recommendation:

- [ ] SDLC Checklist: FY 22/23: The Quality Assurance Working Group and the SDLC Working Group should consider formalizing the above principles and guidance as part of the new SDLC process, milestones and checklist.

Adopt development methodologies like test driven development (TDD) that predates DevOps. How TDD abd DevOps are related is well described in the article '*TDD for a DevOps World*[25] - summary: - TDD is clearly a quality enhancing practice. I - TDD is a really good way to mitigate the risks of defects - TDD is increases the chances of actually achieving the resilient and rugged code that needs to withstand the increasing demands of a DevOps world where expectations are much higher.

Goal: Future Proof Technology

The MACH Aliance was announced in December 2021. AWS, MongoDB and others are associated with this alliance. This alliance defines the strategy, allow container and micro-service focused, still identifies key-concepts for use within our data-centre applications; namely modular, API-based and headless (i.e.,)

This manifesto is "*Future proof enterprise technology and propel current and*

[24] Microsoft - Shift Left Testing
[25] TDD for a DevOps World

future digital experiences". MACH aligns with the GC EARB directions[26]:

- **M**: Micro-Services (Modular): Individual pieces of business functionality that are independently developed, deployed, and managed. A swappable architecture.
- **A**: API: All functionality is exposed through an API.
- **C**: Cloud: SaaS that leverages the cloud, beyond storage and hosting, including elastic scaling and automatically updating.
- **H**: Headless: Front-end presentation is decoupled from back-end logic and channel, programming language, and is framework agnostic.

Our constraints may limit our ability to leverage the cloud. For on-premise constrained systems DevOps practices to automate updating and Kubernetes auto scaling should be prioritized.

A view of how MACH applies to guidance and industry patterns is below.

MACH	Description	ARA Guidance	Industry Patterns
M - Microservices	Individual *pieces* of business functionality can be deployed and managed	Composable Applications	Microservices Architecture Style
A - API	Functionality exposed via API	Composable Applications	GC API-First, GC Standards on API
C - Cloud	Leverage SaaS to its fullest including scalability and automation	Security / Policy Restricted	Aligned to MACH
H - Headless	Decouple front-end from back-end	Cohesive, Loosely-Coupled Applications, Services and APIs	

Aside: An interesting article, *MACH Sitecore Architecture*[27] on how a Content Management System (CMS) is attempting to become MACH-compliant; with discussion on impacts to CMS features like editors, and the use of technologies like JAMstack.

[26] GC EARB EA Standards and Application Architecture
[27] MACH Sitecore Architecture

Goal: User Experience

Our user experience can be improved by looking at modern applications and their integration into varying computing platforms (desktop, mobile, tablet). Some modern experiences can include: - Push Notifications: Business fit-for-purpose notifications using the Push API and integrated into the Windows Operating System experience. Replace mindset of email-based notifications into a notification platform with end-user ability to control notifications. - Sharing Content Across Platforms: Ability to share content across platforms similar to sharing news and social-media content. Allows the ability to communicate effectively in different channels (intranet, CMS, ...). *oEmbed* is one standard for sharing content across platforms with a linkable visual.

Goal: Accessibility

The Accessible Canada Act received Royal Assent on June 21, 2019, and came into force on July 11, 2019.[Reference]. Our department has no formal policies on accessibility. In light of no policy, applications should strive to achieve WCAG 2.1 Level AA. This goal changes by application, and development must ensure they are aware of the business requirements for accessibility.

WCAG 2.1 Level AA (Double-A) implies: - Media: Captions are present on live video. When appropriate, there exists audio description of what's happening on streaming media. - Markup: Ability to resize text without breaking layout. Language is declared in document. - Design: A minimum contrast of 4.5:1 among elements. Heading tags (h1,h2,h3, etc.) are present and emerge from content organically. - Forms: If an error is present on a form, the website will: suggest ways to fix it, the user may withdraw and resubmit the form, or the form prompts a confirmation. - Navigation: Pages can't be nested or unintentionally obfuscated unless part of a step-by-step process, such as an application or feed result. Navigation follows a semantic structure and is repeated on pages.

A basic checklist of minimal accessibility requirements are:

- [ ] Applications should meet WCAG 2.1 Level AA success criteria.
- [ ] Applications should allow users to self-identity accessibility needs.

Creating an Architecture Strategy : Guidance

The architectural strategy for a program; whether they are renewal efforts (ERP, HR, IM, Collaboration) or greenfield (Case Management) should follow methodologies proven to be successful.

The guidance below is a summary of **Technology Strategy Patterns**.[28]

[28] Building Microservices - Sam Newman

Apply Patterns to Formulate a Strategy

- Context: Trends, Constraints, Stakeholders
- Understand: Research, analyse and understand your stakeholders, the environment and the technology landscape.
- Options: Identify options in the products, services and technology roadmaps
- Analysis: Analyse options.
- Recommendation: Make recommendation and obtain approval.

Concerns of an Architect

- Contain entropy: Show a path in a roadmap; garnering support for that vision through communication of guidelines and standards; and creating clarity to ensure efficiency of execution and that you're doing the right things and doing things right.
- Specify Non-Functional Requirements / Quality Requirements: The "..ility" list. scalability, availability, maintainability, manageability, monitorability, extensibility, interoperability, portability, security, accessibility, observability, conformity (laws, directives). Wikipedia - Quality Attributes
- Determine trade-offs: Identity the least-bad option.

1. Patterns / Tools Many tools exist to perform options analysis. Most start with creating an hypothesis and then validating (or invalidating) it formally.

Some tools suggested are: - Logic Tree - SWOT - Strengths, Weaknesses, Opportunities and Threats

Corporate (Enterprise) Context

Use design-thinking in the cross-functional project team to arrive at appropriate architectures .
The steps in design thinking are: 1. Define the Problem 1. Make Observations : determine users, observe user's actions, create personas 1. Form Insights 1. Frame Opportunities 1. Generate Ideas 1. Refine Solutions 1. Try Prototypes

Design thinking principles:

1. Human-centricity
2. Showing, not-telling
3. Clarification
4. Experimentation
5. Collaboration

Patterns

Patterns are known, proven solutions. Patterns help us communicate architecture and design to each other.

Drive strategy with patterns.

Patterns are used at different layers and perspectives; such as:

- application architecture patterns
- software design patterns : Decorator, Factory, Visitor, Pub/Sub, ..
- user experience patterns

Application Architecture Styles

Big Ball of Mud - Anti-Pattern:

A "Ball of Mud" application is an application without structure, software making direct database calls, with no concerns for design. In 1997, Brian Foote and Joseph Yoder, coined this the Big Ball of Mud:

A Big Ball of Mud is a haphazardly structured, sprawling, sloppy, duct-tape-and-baling-wire, spaghetti-code jungle. These systems show unmistakable signs of unregulated growth, and repeated, expedient repair. Information is shared promiscuously among distant elements of the system, often to the point where nearly all the important information becomes global or duplicated.

The overall structure of the system may never have been well defined.

Our department has an abundance of *ball of mud* applications.

Other useful architecture patterns/styles are described in "Application Architecture Styles".

Software Design Patterns

In software engineering, a software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. There are many great books on software design patterns. Wikipedia's Software Design Patterns[29] is a good resource.

Famous design pattern books include:

- 1995 - Design Patterns: Elements of Reusable Object-Oriented Software[30] : Define a set of creational, behavioural and structural software patterns like builder, factory, facada, bridge, mediator, command and observer.

[29] Wikipedia - Software Design Patterns

[30] Design Patterns: Elements of Reusable Object-Oriented Software : Gamma, Erich, ed. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series. Reading, Mass: Addison-Wesley, 1995.

- 2003 - Fowler - Patterns of Enterprise Application Architecture[31] : Defines many categories of patterns including Domain, Data Source, Web Presentation, Distribution, Offline, and Session State Patterns. Specific patterns include data gateway, model-view-controller, and client session state.

User Interface Patterns

Many patterns exist for a successful user-experience (search, navigation, filters, comparisons, grids, ...) - Blueprints by CodePros - Patterns : The Blueprints are a collection of basic and minimal website concepts, components, plugins and layouts with minimal style for easy adaption and usage, or simply for inspiration.

- Google Material UI : An exensive library of UI patterns including Search, Navigations, Onboarding and other common use-case patterns.

Other Patterns / Laws

Business Patterns

Caveman Pattern Reliving old problem experience with new projects even though the risk is extremely low. (O'Reilly Fundamentals of SW Architecture)

Conway's Law Conway's Law asserts that organizations are constrained to produce application designs which are copies of their communication structures. This often leads to unintended friction points. The 'Inverse Conway Maneuver' recommends evolving your team and organizational structure to promote your desired architecture. Ideally your technology architecture will display isomorphism with your business architecture.

- Isomorphism : In sociology, an isomorphism is a similarity of the processes or structure of one organization to those of another, be it the result of imitation or independent development under similar constraints. In our application development context, it implies, that different technology branches, division and sections develop their products under similar constraints.

Cloud Design Patterns

1. Microsoft Azure Cloud Design Patterns
2. AWS Prescriptive Guidance Patters : 2,000 pages of patterns for cloud, DevOps, communication, testing, governance, IoT, Security, Servierless, Spark, ETL, A/B Testing, Canary Testing, ...

[31] Fowler - Patterns of Enterprise Application Architecture : Fowler, Martin. Patterns of Enterprise Application Architecture. The Addison-Wesley Signature Series. Boston: Addison-Wesley, 2003.

Microservices Patterns

Sam Newman's book, "Building Microservices, Defining Fine-Grained Systems"[32] defines many pattnerns, including patterns to migrate from monoliths to microservices. These patterns include:

- Strangle Fig Pattern: . You intercept calls to the existing system—in our case the existing monolithic application. If the call to that piece of functionality is implemented in our new microservice architecture, it is redirected to the microservice.
- Saga Pattern
- References legacy Enterprise Integration Patterns (EIP)[33]
- Backend for Front-End (BFF) : similar to an aggregating gateway, but limited to a single user-interface. For example have a BFF for an Android vs Applie client.

References for microservices patterns:

- Sam Newman Patterns
- Martin Fowler

References

Software

Architecture

- Richards, Mark. & Ford, Neil. Fundamentals of software architecture: an engineering approach. (O'Reilly, 2020)

Design

- [1.Vernon, V. Implementing domain-driven design. (Addison-Wesley, 2013)] http://www.worldcat.org/isbn/9780133039900

Patterns

- Hewitt, E. Technology strategy patterns: architecture as strategy. (O'Reilly, 2018)
- Design patterns: elements of reusable object-oriented software. (Addison-Wesley, 1995).
- Hewitt, E. Technology strategy patterns: architecture as strategy. (O'Reilly, 2018).. Analysis, Strategy Creation and Communication Patterns. Audience is technical leads and architects attempting to recommend a strategy.

[32] Building Microservices - Sam Newman

[33] Gregor Hohpe and Bobby Woolf, Enterprise Integration Patterns (Boston: Addison-Wesley, 2003).

Principles

- Martin, J. Principles of object-oriented analysis and design. (Prentice-Hall, 1993)

Government of Canada

- Digital Standards - Playbook : Provides aspirational guidance around key themes; Design with Users, Iterate and improve frequently, Work in the open by default, Use open standards and solutions, Address security and privacy risks, Build in accessibility from the start, Empower staff to deliver better services, Be good data stewards, Design ethical services, Collaborate widely. Also available as GitHub Pages - *Digital Playbook - GitHub Pages*. These digital standards are common standards used internationally and align well with these *Digital Principles*.
- Directive on Service and Digital : Defines EARB assessment, API use, Network use, IT provisions standards (minimum screen size, .ERP standard, . . .)
  - Standards on APIs - also as Appendix B to Directive on Service and Digital - Mandatory Procedures for APIs
  - GC EARB EA Standards and Application Architecture
- Policy on Service and Digital : Defines role of TBS CIO and Deputy Heads. Defines roles of SSC, PSPC, LAC, CSE departments.
  - These replaced *Directive on Management of Information Technology - Archived 2020-03-31*
  - GC Information Management Guidelines - 1996
- GC Digital Oprations Strategic Plan - 2021-2024 - by Marc Brouillard a/-CIO (pre new CIO). Highlights accomplishments - EARB, Digital Strategic Plan, Open Government Plan, signed Digital Nations Charter (Denmark, Canada, Israel, Mexico, UK, Estonia, Korea, Uruguay, New Zealand, . . .). 1) Modernize IT via APM, 2) Improve Services, 3) Implement Enterprise, 4) Transform the Institution.
  - GC Digital Strategy - DOSP aligns with digital strategy.
  - GC Digital Operations Strategic Plan - 2018-2022 : User-centric, open, accessible, modern technology, digital leaders, digitally enabled.
- GC TBS Information Management Strategic Plan - 2017-2021: Includes strategic goals and objectives.
- GC Cloud Adoption Strategy