Homework #1

ADD v.2, TOGAF v9.1 and IEEE 42010:2011

Team: Caliware

|  |  |  |
| --- | --- | --- |
| Timothy Cioffi-Dinkel | Anthony Farina | Joanna Hang |
| timothycioffi@csu.fullerton.edu | afarina@csu.fullerton.edu | johang@fullerton.edu |
|  |  | sig.jpg |

|  |  |
| --- | --- |
| Lourdes Lopez | David Sullivan |
| lourdeslopez@csu.fullerton.edu | dsullivan@fullerton.edu |
| Untitled.png |  |

Date Submitted: 3/3/2016

CPSC545: Software Design Architecture

Spring 2016

Dr. Chang-Hyun Jo

# Revision History

|  |  |  |
| --- | --- | --- |
| **Date** | **Title** | **Team Member** |
| 02/08/16 | ADD | Timothy Cioffi-Dinkel |
| 02/09/16 | TOGAF | David Sullivan, Joanna Hang |
| 02/09/16 | IEEE | Anthony Farina, Lourdes Lopez |
| 02/24/16 | Comparison | Everyone |

# Table of Contents

Contents

[Homework #1 1](#_Toc444710460)

[Revision History 2](#_Toc444710461)

[Table of Contents 3](#_Toc444710462)

[Summary 4](#_Toc444710463)

[ADD V.2 4](#_Toc444710464)

[Introduction 4](#_Toc444710465)

[Plan, Do, Check 5](#_Toc444710466)

[Inputs 5](#_Toc444710467)

[Outputs 6](#_Toc444710468)

[The Checklist 7](#_Toc444710469)

[ADD Revised 8](#_Toc444710470)

[Conclusion 8](#_Toc444710471)

[TOGAF Version 9.1 10](#_Toc444710472)

[Part I: Introduction 10](#_Toc444710473)

[Part II: Architecture Development Method 10](#_Toc444710474)

[Part III: ADM Guidelines and Techniques 12](#_Toc444710475)

[Part IV: Architecture Content Framework 13](#_Toc444710476)

[Part V: Enterprise Continuum & Tools 14](#_Toc444710477)

[Part VI: TOGAF Reference Models 14](#_Toc444710478)

[Part VII: Architecture Capability Framework 15](#_Toc444710479)

[IEEE 42010:2011 16](#_Toc444710480)

[Architecture Descriptions 16](#_Toc444710481)

[AD Elements and Correspondence 17](#_Toc444710482)

[Stakeholders and Concerns 18](#_Toc444710483)

[Architecture Viewpoints 18](#_Toc444710484)

[Architecture Views 19](#_Toc444710485)

[Architecture Models and Model Kinds 19](#_Toc444710486)

[Architecture Decisions and Rationale 19](#_Toc444710487)

[Life Cycle 19](#_Toc444710488)

[Architecture Frameworks 19](#_Toc444710489)

[Architecture Description Language 20](#_Toc444710490)

[Uses of Architecture Descriptions 20](#_Toc444710491)

[Conclusion 21](#_Toc444710492)

[Compare ADD v.2 with TOGAF v9.1 and IEEE 42010:2011 22](#_Toc444710493)

[Goals 22](#_Toc444710494)

[ADD Goals 22](#_Toc444710495)

[TOGAF Goals 22](#_Toc444710496)

[ADD and TOGAF Goals Comparison 22](#_Toc444710497)

[IEEE 42010 Goals 22](#_Toc444710498)

[ADD and IEEE 42010 Goals Comparison 22](#_Toc444710499)

[Ideas 23](#_Toc444710500)

[ADD Ideas 23](#_Toc444710501)

[TOGAF Ideas 23](#_Toc444710502)

[ADD and TOGAF Ideas Comparison 23](#_Toc444710503)

[IEEE 42010 Ideas 23](#_Toc444710504)

[ADD and IEEE 42010 Ideas Comparison 24](#_Toc444710505)

[Steps 24](#_Toc444710506)

[ADD, TOGAF and IEEE 42010 Steps Comparison 25](#_Toc444710507)

[Work-products 27](#_Toc444710508)

[ADD, TOGAF, IEEE 42010 Work Products Comparison 28](#_Toc444710509)

[Pros and Cons 29](#_Toc444710510)

[ADD v2 29](#_Toc444710511)

[TOGAF 30](#_Toc444710512)

[IEEE 42010 31](#_Toc444710513)

[When/how to use each better in the field 31](#_Toc444710514)

[Lessons Learned 32](#_Toc444710515)

[Timothy Cioffi-Dinkel 32](#_Toc444710516)

[Anthony Farina 32](#_Toc444710517)

[Joanna Hang 32](#_Toc444710518)

[Lourdes Lopez 33](#_Toc444710519)

[David Sullivan 33](#_Toc444710520)

[Group Experience 33](#_Toc444710521)

[Reference List 34](#_Toc444710522)

[Team Charter 35](#_Toc444710523)

[Team Evaluation 39](#_Toc444710524)

# Summary

## ADD V.2

### Introduction

The ADD method, or the Attribute-Driven Design method, is a method of architectural design which recursively breaks down a system to develop a system based on a software's quality attribute requirements (Software Engineering Institute, 2006). The ADD method follows a “Plan, Do, Check” approach to design in which system attributes are selected, the attributes are satisfied by instantiating elements of that attribute, and the result is verified and refined to best fit the desires of the stakeholders.

This design method utilizes inputs and outputs for a system in order to have a specified qualification for software architecture. The inputs for a system include: functional requirements, or what a system must provide for the user; quality attributes, the way in which a system must show various properties; and design constraints, the constraints a system will face and the decisions on what the systems base design need are. The outputs include: responsibilities, data collected by the system; roles, related responsibilities; software element, a way to fulfill the roles and responsibilities; property, information about an element being developed; and relationship, a way in which two elements interact.

Once these inputs and outputs are discussed and documented, the recursive checklist is followed which would inevitably lead to a software architectural design, ready to be placed into development. This checklist follows an eight step process or which six are recursively produced as needed until the design is completed.

The first step, step zero, is outside of this scope but is where stakeholders prioritize their requirements for a software. Step one ensures that there is enough information to begin the iterations. Step two chooses which part of the software is going to be iterated through steps three through seven. Step three ranks requirements, depending on their importance to the design, and groups them by priority; these are called architectural drivers. Step four selects the best design for each of the drivers, selects the best pattern for each driver, and also selects alternatives if constraints arise. Step five selects one instance from each elements in step four and assigns responsibilities to these elements. Step six enacts the solution to a functional requirement and records findings. Lastly, step seven verifies that each input is assigned to an elements and it has been refined. Once it has been recursively implemented until a full system is developed, the ending result will be an architectural system ready for development which satisfies the inputs and outputs produced by stakeholders.

This method of design has been naturally revised by ADD practicing individuals who seek to make ADD easier to learn, understand, and apply to other teams who seek a powerful way of architectural design.

### Plan, Do, Check

The ADD design method utilizes a Plan, Do, Check cycle in order to produce a well-functioning elements of a system that satisfies the needs of the stakeholders. For the “Plan” portion of this cycle the inputs to the system design are gathered and then the elements needed for the system are selected. This allows for the desires of stakeholders to be heard by the designers and allows for the designers to decide on which elements will fit into the system and which elements will best satisfy the desires, as not all desires can be met in one design.

The “Do” portion of the cycle takes the elements decided on in the planning section and instantiates the elements into the design to satisfy the inputs of the stakeholders. This ensures that the elements requested by the stakeholders are implemented into the system.

The “Check” of the cycle analyzes the instantiated elements and verifies that they do indeed produce what the requirements are and seeks ways for the element to be refined for another iteration of the design of that element.

### Inputs

The design requires various inputs as to direct the way the system should behave. The inputs include: functional requirements, design constraints, and quality attributes.

Functional requirements are the specifications of functions that the stakeholders desire from a system. The system must perform the base-line conditions of the stakeholders wants, as they are the ones who have the most to lose for a design being faulty, and without these the system cannot be developed as it will not fit the desires of those investing the most into a project. Examples of this include:

* This system will allow a user to add a credit card
* This system will allow a user to have a checkout area for purchases
* This system will allow you to have an account with a personalized password

Design constraints are specifications of elements that must be met in the finalized version of the system. This differs from functional requirements because functional requirements are what a system must allow a user to do while design constraints are what the system must be able to utilize in order for user satisfaction. Examples include:

* The system will run on Windows, Mac, and Linux platforms
* The system will use a SQL database
* The system will run using Java as the architectural programming language.

Quality attributes are the timing requirements for which a system must be constrained to. These requirements guarantee that a system will not take too long to develop, the patches and updates will not take too long to push through, and that the system itself utilized proper timing for processes to take place. Examples include:

* The system will push through a command in under one second
* The system will be update the first of every month
* The system will not use more than 20% of the CPU’s power
* The system will be tested every night for fault and bugs.

All of these inputs are needed in order to have a base for which the system can be developed on. With the understanding of what a system must seek to accomplish, designers can then integrate these requirements into the architecture of the system.

### Outputs

The ADD method’s inputs develop the basis for a systems architecture, planning the stakeholder’s requirements as well as the technical functions of the system, the outputs provide a system with the means to take the input elements and break them down to fully complete each element.

The method expects five outputs from a design of the architecture, the software element, role, responsibility, property, and relationships.

* **The responsibility** - the information for which an element in the input provides to the system. For instance if an input was for a system to allow the user to add a credit card to their profile, the responsibility would be the credit card information gathered from that user.
* **The role** - a collection of responsibility which share a common trait; for instance information about a user’s name, birthdate and email address all share common information which is the user’s identity, therefore the information gathered for these functionalities would be a role.
* **A software element** - any part of the design which completes a role or responsibility. If a responsibility is for the system to take a user’s name as their profile, then creating a database to store and recover this data would fulfill that responsibility, therefore the database would be an element of the system.
* **The property** - these are the characteristics of each element. For instance, an element such as a row in the database would have a name (i.e. user\_name), a type (sql), and a quality attribute (pull a user name in one second). These becomes the building blocks for an element to be designed.
* **The relationship** - how the element will associate with other elements. Such as this row being correlated to a user’s e-mail address (user\_mail) in order to match a user’s name with their identity.

These outputs are expected by using the ADD method as it provides the designers with way on how to solve each requirement or input to the design, allowing for a smooth means for the design of the architecture.

### The Checklist

The ADD method follows a checklist, a sequence of steps that are iterated, based on each element that is desired to be designed, until the design of that element is complete and the entirety of the system is completed.

The steps include nine steps for which two through seven are iterated based on the needs of the designers. These steps include: prioritizing the requirements, confirm there is sufficient requirements information, choosing an element to decompose, identify candidate architectural drivers, design concepts that satisfy the drivers, instantiate architectural elements and allocate responsibilities, define interfaces for elements, verify and refine requirements, repeat two through seven until the next element is needed for decomposition.

1. **Prioritize the requirements** - The stakeholders will gather together and prioritize which requirements will need to be put into the software and in what order the requirements need to be fulfilled.
2. **Confirm there is sufficient requirements information** - Once the stakeholder’s present their list, the designers then choose what elements will impact the structure the most and which elements are the highest in priority to fit these impacts. Elements that are not prioritize will need to be sent back to the stakeholders for further analysis. This allows for the designers to understand what the stakeholder’s want in the software and how the elements the stakeholder’s desire will impact the system as a whole.
3. **Choose an element of the system to decompose** - The designers will choose an element, of highest priority, for which they will break down and attempt to integrate into the system’s design. If this is the first iteration of this step, the entire system must be looked at and designed first.
4. **Identify candidate architectural drivers** - Once the element is select, the architectural drivers will need to be selected. These are the requirements for which an element is required to have, again ranked for significance, chosen by the stakeholders. A second prioritization then occurs where the stakeholders priority is listed next to the impact for which the driver will have on the system as a whole (H,H H,L L,H L,L). The designers will then select a few of these drivers based on their priority, prioritizing those with high priority from stakeholders first and low to high impact on the system to those that the stakeholders do not take a priority to but require higher impacts to the system.
5. **Choose a design concept that satisfies the architectural drivers** - This is where the design begins. The concept for the design needs to satisfy the drivers selected from step four. To do this the concept needs to follow six steps.
   1. See any issues that will face from implementation of the driver, as to allow for solutions to any issues that may arise.
   2. Look at alternatives to addressing the concerns that arise from a driver. For instance, if the driver is “to select a coding language for the system” a list of various languages and their benefit/cons would be created.
   3. Select the best solution to adequately satisfy the driver based on the benefit and cons drawn in “b”.
   4. If the driver has overlapping solutions, consider these overlaps and possible benefits from the similarities.
   5. See alternative architectural insights from these different solutions as a learning opportunity for better ways of designing an architecture.
   6. Ensure there is no issues with the selected concept.
6. **Instantiate architectural elements and allocate responsibilities** - Step six materializes the elements and assigns responsibilities to these elements. The designer must take one instance of a type as to begin breaking down the elements and easily assigning responsibilities to those elements. The design decisions are documented and if additional requirements are needed, additional element types are created and given to the different elements.
7. **Define interfaces for instantiated elements** - The next step is to define what the elements should be providing to the system overall or to other areas of the system, the restrictions this element may possess, and also what types of measures were put into place to handle bugs or errors that may arise from changes to the code.
8. **Verify and refine requirements and make them constraints for instantiated elements -** Then the designer refines the requirements for each of the children elements, ensuring that the requirements, quality attributes and design constraints fulfill the elements desired output.
9. **Step 8: Repeat Steps 2 through 7 for the next element of the system to decompose -** Lastly, steps two through seven are repeated for the next element that is wished to be designed.

### ADD Revised

Like every aspect of development, this method has been refined and revised to fit a more modern and sleek approach. Those who utilized this system in their everyday development decided to revise the old ADD method. To do this they took the old inputs and outputs and made them more clear to designer, they renamed and renumbered steps in the checklist to streamline the process, offered how to perform each step in greater detail than what was given before, and lastly gave what each step’s overall design decision should be.

Doing this allowed for ADD to be usable by more people and easier to use for those who desired its process.

### Conclusion

The ADD method is process for which a team can easily design the architecture for a system. It bases its design off of the software’s quality attributes in order to develop a design based off of the desired outcomes from stakeholders. The ADD method utilizes stakeholder inputs including: functional requirements, design constraints, and quality attributes; as well as outputs of: roles, responsibilities, properties and relationships, in a recursive process to develop a completed architectural design.

This process breaks the desires into elements, then given drivers to complete for each element. These elements are solved based on their intended outcome. Once the outcome for the element is achieved the process is repeated for a new elements until all of the elements are completed and the design is finished.

## TOGAF Version 9.1

TOGAF, an architecture framework, is a detailed method and set of tools for developing and maintaining enterprise architecture. TOGAF is developed and maintained by The Open Group. TOGAF focuses and aligns business issues and needs with IT. It is a flexible framework that can be applied and tailored to any industry, and it is also complementary as it can be used with other enterprise frameworks. By following the TOGAF method, the architecture team can identify key areas for change that will create more effective and efficient IT operations. Improving the enterprise architecture will support the delivery of the overall business strategy and increased ROI, and it results in a more agile organization that is more responsive to change.

TOGAF deals with four types of architecture: business, data, application, and technology architecture. Business architecture defines business strategy and governance. Data architecture describes the logical and physical data assets and how data is accessed. Application architecture describes the applications that supports the business functions and its interactions. Technology architecture includes the software and hardware infrastructure that supports the applications. Overall, TOGAF consists of seven parts, all of which are described below.

### Part I: Introduction

Part I contains a high level introduction that introduces core concepts of TOGAF, defines terms used throughout TOGAF, and provides release notes about the different versions of TOGAF.

### Part II: Architecture Development Method

The TOGAF Architecture Development Method (ADM), which is essentially the core and heart of TOGAF, is a step-by-step guide for developing an enterprise architecture. The ADM is a generic method that can be coordinated with other business and IT frameworks. It is an iterative process, and the ADM is broken into several phases listed below.

***Preliminary Phase***

The Preliminary Phase describes the overall initial organization and planning for an enterprise architecture. The goal for the preliminary phase is to determine the desired Architecture Capability and how to establish it. This is accomplished by understanding the current business environment, gaining commitment from high level management, agreeing on scope, establishing the principles, establishing governance, and customizing TOGAF.

***Phase A: Architecture Vision***

Architecture Vision is the view of architecture of the proposed enterprise architecture. It shows what stakeholders are involved and their key concerns. The purpose of the Architecture Vision is to clarify the agreement of the architectural effort to addresses key business requirements. Essentially, the Architecture Vision connects the business strategies and goals with the architecture. This initial description will be further developed in subsequent phases. Expectations and constraints of the project are developed here.

***Phase B: Business Architecture***

The first architectural activity is to develop the Target Business Architecture that will be the foundation of the other domains (Data, Application, and Technology). The contents in the Business Architecture include: organization structure, business goals and objectives, functions, services, processes, roles, and any correlations of organization and functions. Business models are used to map the high-level business requirements to more detailed ones. Examples of business models are Activity Models, Use-Case Models, and Class Models. These views should be built to communicate to stakeholders how their concerns related to the Business Architecture.

**Steps in Phase B, C and D**

1. Select reference models, viewpoints and tools
2. Define Baseline Architecture Description
3. Define Target Architecture Description
4. Perform gap analysis
5. Define candidate roadmap components
6. Conduct formal stakeholder review
7. Finalize the Architecture
8. Create Architecture Definition Document

The architecture team checks what available resources from the Architecture Repository can be used and new building blocks are introduced. Reference models are selected, then a baseline and target is created. Afterwards, a gap analysis is performed and activities are prioritized. An analysis is conducted to determine the implications of the proposed Architecture. Finalized Architecture are created after conducting a stakeholder review.

***Phase C: Information Systems Architectures***

The Information Systems Architecture is a fundamental part of an IT system that processes the types of information and applications and the relationships between them. The goal is to develop the Target Data Architecture, which pertains to data management, and the Target Application Architecture. Data Architecture describes how data is created, transformed, distributed, migrated, secured and archived. Applications Architecture describes the vertical sector of the organization’s industry.

***Phase D: Technology Architecture***

Technology Architecture addresses the logical and physical applications and data components, including: hardware, software, and communications technology. The architecture team considers what Technology Architecture resources are available in the Architecture Repository such as: TOGAF Technical Resource Model (TRM), generic technology models from the organization’s industry, and models from Common Systems Architectures.

***Phase E: Opportunities and Solutions***

Here, initial implementation planning begins as the major projects, approach, priorities, and dependencies are identified. The transition to accomplish the Target Architecture includes: Architecture Roadmap, Work Packages, Transition Architectures, and the Implementation and Migration Plan. The transition is a timeline of logical grouped changes that shows the status between the Baseline and Target Architectures in the scheduled projects.

***Phase F: Migration Planning***

Phase F assesses the dependencies, costs and benefits of the Implementation and Migration Plan. It checks to see the Migration Plan will impact any of the four management frameworks: Business Planning, Enterprise Architecture, Portfolio/Project Management or Operations Management. Using Business Value Assessment Technique, the criterion of business value within the organization for each work package is established (Performance Evaluation Criteria, Return-on-Investment Criteria, Business Value, Critical Success Factors, Measures of Effectiveness, and Strategic Fit). The plan must estimate the cost, opportunities, resources and time required for each project. It must prioritize the migration projects by conducting a cost/benefit assessment and risk validation.

***Phase G: Implementation Governance***

Implementation Governance ensures the implementation conforms to the Target Architecture. This is the phase where the development towards the Target Architecture begins. Here, an implementation program is established along with a deployment scheduled as referred by the Architecture Roadmap.

***Phase H: Architecture Change Management***

Architecture Change Management ensures that the architecture is maintained and executed with a flexible, responsive, and adaptable change management process all while achieving the business value. Change management determines if a new architecture cycle needs to be initiated (a new ADM) or if change shall be allowed after the architecture deployment (an update). The change is dependent on classifying it into categories: simplification, incremental, or re-architecting change. To determine if the change shall result in an update or new ADM, analyze the amount of stakeholders that the change can impact.

***Requirements Management***

Requirements Management is interconnected in all the phases as it dynamically drives the architecture, and ensures that every stage validates business requirements. Approved requirements are found in the Requirements Repository and Architecture Requirements Specification.

### Part III: ADM Guidelines and Techniques

The ADM Guidelines and Techniques is a collection of guidelines and techniques to applying TOGAF and the ADM. Guidelines include how to apply the iteration, levels, security, and Service-Oriented Architectures (SOAs). Techniques described help support the tasks in the ADM.

***Iteration***

Projects may return to previous ADM phases in order to update work products with new information (iteration).

* Architecture Development Iteration
* Transition Planning Iteration
* Architecture Governance Iteration
* Architecture Capability Iteration

***Levels***

Levels provide a framework for dividing the Architecture Landscape into three levels of granularity.

* **Strategic Architecture** provides an organizing framework for executive level decision making
* **Segment Architecture** provides an organizing framework for architecture roadmaps at a program or portfolio level
* **Capability Architecture** provides an organizing framework for the development of architectural roadmaps realizing capability increments.

ADM supports architectures at different levels based on the following characteristics of the Architecture Landscape. Breadth, depth, time period, and recency are dimensions used to define and limit the scope of an architecture.

* **Breadth:** A specified subject matter area segments
* **Depth:** For broad subject areas, less detail is needed. For specific subject matter, more detailed architecture is needed
* **Time:** For a specified breadth and depth the broader and less detailed, the longer period it can be useful.
* **Recency:** After an architecture is approved, it will need to be maintained or else it will decrease in accuracy.

***Security***

Security guidelines address the security considerations during the TOGAF ADM. Security concerns for architecture domains include authentication, authorization, audit, assurance, availability, asset protection, administration, and risk management.

***Service-Oriented Architecture (SOA)***

Service-Oriented Architecture (SOA) is an architecture that supports service-orientation. It represents a business activity that can is repeatable, self-contained, composed of other services, and a black box to consumers.

### Part IV: Architecture Content Framework

The Architecture Content Framework describes the TOGAF content framework, which can be used as a stand-alone framework or in conjunction with external frameworks. The content metamodel shows the different types of building blocks within the architecture and the relationships between them. The content framework work products include Architectural Artifacts, Architecture Deliverables, and Building Blocks.

* **Architectural Artifact** - An architectural artifact describes architecture parts like catalogs, matrices, diagrams.
* **Architecture Deliverable** - An architecture deliverable is a work product that was a contractually agreed upon by stakeholders that may take in the form of: output of projects, documentation, or in the Architecture Repository as a reference model, standard, or snapshot of the Architecture Landscape.
* **Building Block** - A building block is a potentially reusable component of business, IT, or architectural capability and can relate to the architecture or solution and can be
  + **Architecture Building Blocks (ABBs)** - ABBs describe the capability requirements and influence the shape of the SBBs.
  + **Solution Building Blocks (SBBs) -** SBBs are the individual components implemented to fulfill the capability requirements.

### Part V: Enterprise Continuum & Tools

Enterprise Continuum and Tools show the different types of architectures that are typically developed by companies. It contains architecture descriptions, models, building blocks, patterns that exist within the enterprise and the IT industry. For example, highly generic architecture such as the TOGAF Technical Reference Model (TRM), those specific to IT models such as web services architecture, or specific to an industry.

* **Enterprise Continuum** - classes of assets that are not directly used in ADM architecture development but contextual asses: policies, standards, strategic initiatives. The enterprise continuum explains how generic solutions can support organizational requirements and consists of the Architecture Continuum and the Solutions continuum.
* **Architecture Continuum** - represents the structure of Architectural Building Blocks (ABBs). These are reusable architectural assets that and are used to guide elements in the Solutions Continuum. The architecture continuum is a useful tool to discover commonality and eliminate unnecessary redundancy.
* **Solutions Continuum** - defines what is available in the organization for re-use as Solution Building Blocks (SBBs). These are results of agreements between customers and business partners.

### Part VI: TOGAF Reference Models

The TOGAF Reference Model contains the TOGAF Technical Reference Model (TRM) and Integrated Information Infrastructure Reference Model (III-RM). TRM focuses on the Application Platform space. TRM provides a taxonomy to define the components and a conceptual structure in the foundation of the architecture. III-RM focuses on the Application Software space.

**TRM Entities**

* **Application Software** - There are two categories of Application Software:
  + Business Applications - Applications that are specific to the organization’s industry
  + Infrastructure Applications - Widespread or commercial off-the-shelf Applications
* **Application Platform** - a combination of services that support underneath the Application Software
* **Communications Infrastructure** - basic services to interconnect data for the transfer of data

**TRM Interfaces**

* **Application Platform Interface** - a well-defined interface between Application Software and an Application Platform that both sides conform to for providing platform services.
* **Communications Infrastructure Interface** - an interface between the Application Platform and the Communications Infrastructure over a network.

### Part VII: Architecture Capability Framework

Architecture Capability Framework shows how to organize the framework. If it is implemented any capability with an organization would require the design of the four domain architectures: Business, Data, Application, and Technology. The Architecture Capability Framework provides guidelines for and Enterprise Architecture Board for making decision regarding architecture and governance on an enterprise-wide level. This provides guidance and effective usage of resources for meeting the organization’s strategic objectives.

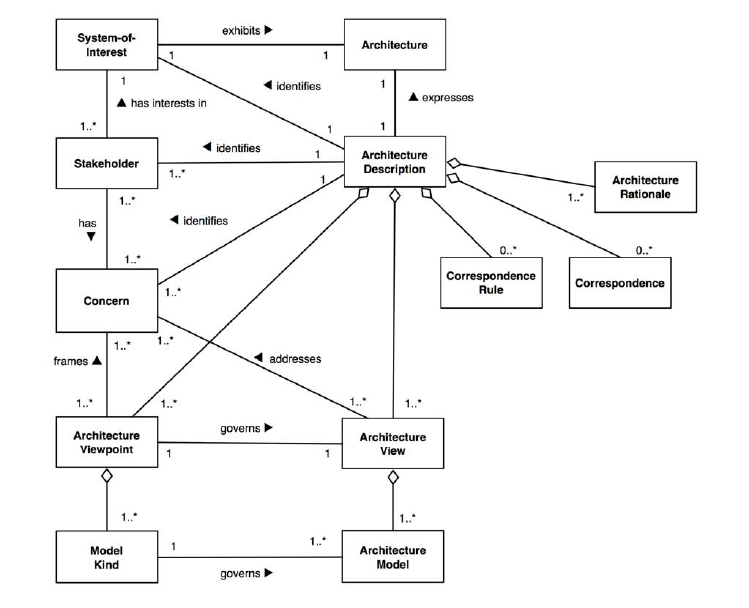
## IEEE 42010:2011

The ISO/IEC/IEEE 42010 is an international standard created by the International Organization for Standardization, the International Electrotechnical Commission, and the Institute of Electrical and Electronics Engineers. The 42010 standard “addresses the creation, analysis and sustainment of architectures of systems through the use of architecture descriptions” (ISO/IEC/IEEE 42010:2011, 2011). The main goal for the standardization of the definition(s) of architecture description and its components is to make it easier for software architects to create, analyze, and sustain architecture systems. When all definitions are understood, it makes is significantly easier to understand architecture systems especially when trying to analyze a system created by two or more different architects.

### Architecture Descriptions

Architecture Descriptions are work products that are used to represent an architecture. They are composed of one or multiple components such as concerns, stakeholders, architecture viewpoints, architecture frameworks, architecture description languages as well as architecture terms and concepts. These components are formerly knowns as AD elements. The ISO/IEC/IEEE 42010 standard provides requirements for architecture descriptions but does not define a format or process for creating architecture descriptions. This is out of scope for this standard. The standard also provides definitions and insights about the individual AD elements and their correspondences (relations) with each other. This document will summarize these components.

Overall, architecture descriptions should define the system and provide the necessary additional information. Requirements for the additional information will be defined by the organization or project beforehand. The figure below is describes a model of an architecture descriptions components (ISO/IEC/IEEE 42010:2011, 2011).



The ISO/IEC/IEEE 42010 standard specifies that an architecture description shall contain an overview, information regarding stakeholders and their concerns in regards to the system, a definition for each viewpoint, architecture views and the corresponding model for each viewpoint. Architecture descriptions shall also contain information specifying the reasoning behind architecture decisions.

### AD Elements and Correspondence

As stated in the previous section, AD elements are the base components of architecture descriptions. These AD elements have relationships with other AD elements. This relationship is called a correspondence and is ruled by correspondence rules.

Some types of AD elements:

* Stakeholders
* System concerns
* Architecture viewpoints
* Architecture views
* Model kinds
* Architecture models
* Architecture decisions
* Rationale

### Stakeholders and Concerns

To achieve the goal of architecture descriptions, the description must first identify the stakeholders and their concerns.The stakeholders to be identified will vary by project but at the very minimum, there are eight stakeholders that must be considered. To name a few, the users of the system, the operators of the system, and acquirers of the system are all stakeholders that must be considered. Along with identifying stakeholders, there are five concerns that need to be considered. These concerns must be associated with the stakeholders who hold these concerns. These stakeholders and concerns should be “framed by” an architecture viewpoint.

### Architecture Viewpoints

Architecture viewpoints are very central components in architecture descriptions. Viewpoints are used for two main purposes. The first purpose of a viewpoint is to create/define a perspective in which architecture views are displayed and explained. This perspective includes conventions such as languages, notations, model kinds, design rules, modeling methods, analysis techniques, and other operations. Architecture views then uses a viewpoint to explain the architecture of the system. The second purpose of an architecture viewpoint is to “frame” the stakeholder concerns. Overall, the architecture viewpoint should include:

1. One or more concerns framed by the viewpoint.
2. Typical stakeholders for concerns framed by this viewpoint.
3. One or more model kinds used in this viewpoint. For each model kind, the conventions should be specified.
4. References to its sources

### Architecture Views

Architecture views are directly related to architecture viewpoints. A view depicts the architecture based on the stakeholders concerns. The ISO/IEC/IEEE 42010 standard specifies that there shall only be one view per viewpoint. The architecture view should include:

1. Information that identifies the view
2. Reference information to the viewpoint
3. Architecture models
4. Information regarding any known issues related to the view

Architecture views are governed and composed of architecture models and model kinds either directly or indirectly through architecture viewpoints. These components are detailed in the next section.

### Architecture Models and Model Kinds

Model kinds are very important components to architecture descriptions because they are directly related to architecture views and viewpoints. Model kinds define the type of conventions that the architecture model must adhere to. Some examples of model kind conventions are data flow diagrams or state transition models.

An architecture model is composed of a model kind and are a part of one or more architecture views. The architecture model should be able to focus on all of the concerns that are framed by the architecture viewpoint.

### Architecture Decisions and Rationale

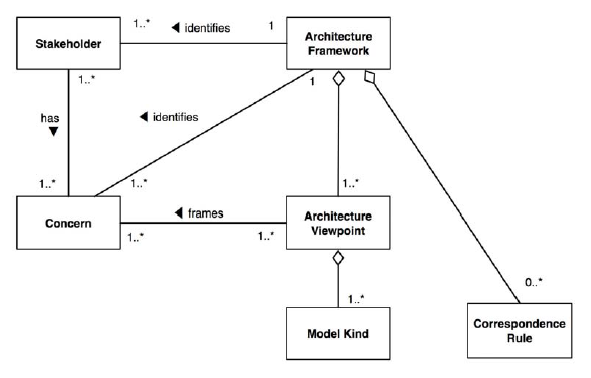
Architecture decisions are documented within the architecture description as well as the reasoning behind why a decision was made. This information becomes useful when the team is analyzing the architecture for modifications, maintenance or reuse.

### Life Cycle

Architecture analysis, development and maintenance occurs throughout the system lifecycle. Architecture descriptions provide a method for a team to analyze a common representation of the architecture throughout the lifecycle. Having an architecture description allows the team to effectively communicate possible changes to the architecture and analyze their impact.

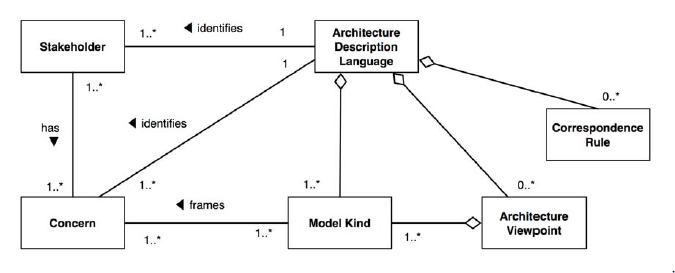
### Architecture Frameworks

Architecture frameworks are used to create architecture descriptions along with developing methods to model the architecture and creating communication lines between organization and projects. The figure below shows all of the parts of an architecture framework (ISO/IEC/IEEE 42010:2011, 2011). These parts include concerns, stakeholders, viewpoints, and correspondence rules.



### Architecture Description Language

Architecture description languages (ADLs) are very similar to architecture frameworks in that they define concerns and stakeholders. But where the two differ is that ADLs define model kinds which can be “organized into viewpoints”. The figure below shows the parts of an architecture description language (ISO/IEC/IEEE 42010:2011, 2011).



### Uses of Architecture Descriptions

Architecture descriptions have several uses for the stakeholders including:

* Design
* Development
* Analysis
* Review
* Evaluate/compare alternative architectures
* Documentation
* Maintenance and support
* Communication
* Compliance
* Training

### Conclusion

In conclusion, the ISO/IEC/IEEE 42010 standard specifies the practices, methods, and components of architecture descriptions. These practices, methods, and components include architecture frameworks, stakeholders, concerns, architecture description languages, architecture viewpoints, model kinds, and architecture views. With this standard, software architects, stakeholders, organizations, etc. should have a better understanding of creating, analyzing, and interpreting software architecture.

# Compare ADD v.2 with TOGAF v9.1 and IEEE 42010:2011

## Goals

### ADD Goals

ADD is a software architecture design method which goal is to produce a design with the main focus being quality attributes of a software. These quality attributes are geared towards aspects such as user experience, timing issues of a software and overall proper functionality of the software.

### TOGAF Goals

TOGAF is a framework to build an enterprise architecture and offers a repository of best practices. It emphasizes an organization’s business goals as the driver for architecture. The goal of TOGAF is to institute an industry standard with a generic enterprise architecture method that can be used on its own or complementary with other frameworks. It shall achieve the Boundaryless Information Flow vision.

### ADD and TOGAF Goals Comparison

Both ADD and TOGAF create an architecture design iteratively through following a structured process. ADD focuses on technical architecture in the application domain, whereas TOGAF is related to enterprise architecture that addresses the business goals of utilizing IT resources. TOGAF is an architectural process that starts with business architecture to set the foundation for the data, application, and technology architectures.

### IEEE 42010 Goals

IEEE provides standards on what information is required within an architecture description. An architecture description is a documentation of the architecture. The architecture description can be used throughout the software lifecycle to aid in the design, analysis, and maintenance of the architecture. IEEE 42010 does not provide a standards for the process of designing an architecture.

### ADD and IEEE 42010 Goals Comparison

While ADD provides a method for the creation of an architecture design, IEEE 42010 is a standard for documenting the design in the form of an architecture description. IEEE 42010 standards for documenting architecture descriptions can be used in combination with ADD, since IEEE 42010 standards do not provide a method for designing an architecture. Both methodologies are centered on designing and viewing the architecture based upon stakeholder concerns and quality attributes.

## Ideas

ADD and TOGAF are architecture frameworks whereas IEEE is an architecture description standard. ADD is a methodology to create software architectures based on quality attributes. TOGAF is a high level approach for developing an enterprise architecture.

### ADD Ideas

The ADD method follows a “Plan, Do and Check” system in which the designers plan which quality attributes, which design constraints or which requirement the wish to address. Next, they instantiate this plan in order to satisfy the requirement, and last, check to ensure the requirement was actually satisfied and works in the system.

### TOGAF Ideas

Here are the main ideas of TOGAF:

* Architecture Development Method (ADM) - The enterprise-wide architecture is developed iteratively through several phases to refine the enterprise architecture and business requirements.
* Architecture Content Framework - Architectural work products that are produced in the ADM phases: Deliverables, Artifacts, and Building Blocks
* Enterprise Continuum - The methods for classifying the architecture and solution artifacts across the enterprise which fall into a continuum from generic to specific
* TOGAF Technical Reference Model (TRM) - a generic framework of services that is used to be combined with other industry frameworks to guide to build the organization’s architecture
* Architecture Capability Framework - a definition of the organizational structures, skills, roles, and responsibilities required to establish and support the enterprise architecture

### ADD and TOGAF Ideas Comparison

Both ADD and TOGAF methods are developed iteratively and move gradually from more general to more specific details of the architecture. The ADD process follow “Plan, Do, Check” in order to decompose the entities from a quality perspective. TOGAF uses generic frameworks as a starting point and follows a process to tailor it to the enterprise. Both approaches take steps to refine the requirements that result an improved architecture that is flexible to change. ADD ideas focus on following a process to produce a software architecture to maximize efficiency and quality. TOGAF focuses on developing the enterprise architecture that takes into account various perspectives of the enterprise to improve use of IT resources.

### IEEE 42010 Ideas

IEEE 42010 standards for documentation of an architecture within an architecture description can be used while designing, analyzing, and maintaining an architecture. Practicing a standardized method for documentation will facilitate communication regarding an architecture. Stakeholders will have a clear understanding of where to find information about the architecture and rationale for decisions made in regards to the architecture design.

### ADD and IEEE 42010 Ideas Comparison

Both ADD and IEEE 42010 standards are centered on quality attributes. ADD provides a process for designing an architecture based on quality attributes and IEEE 42010 standards provides a method for documenting architecture description from the perspective of stakeholders’ fundamental concerns. Both methodologies can be used in combination for architecture design and maintenance of the architecture throughout the system lifecycle.

## Steps

Below is a list of the major steps that were identified for ADD v2, TOGAF V9.1, and IEEE42010:2011.

|  |  |  |
| --- | --- | --- |
| **ADD V2** | **TOGAF V9.1 (ADM)** | **IEEE42010:2011** |
| Prioritize requirements  Step 1: Confirm there is sufficient requirements information  Step 2: Choose an element of the system to decompose  Step 3: Identify candidate architectural drivers  Step 4: Choose a design concept that satisfies the architectural drivers  Step 5: Instantiate architectural elements and allocate responsibilities  Step 6: Define interfaces for instantiated elements  Step 7: Verify and refine requirements and make them constraints for instantiated elements | Preliminary Phase  Phase A: Architecture Vision  Phase B: Business Architecture  Phase C: Information Systems Architectures  Phase D: Technology Architecture  Phase E: Opportunities and Solutions  Phase F: Migration Planning  Phase G: Implementation Governance  Phase H: Architecture Change Management  Requirements Management | Architecture Description  -Identify stakeholders  -Identify concerns  -Document relation between stakeholders and concerns  Viewpoints (one or more)  -each concern must be framed by at least one viewpoint  Views (one or more)  -one architecture view for each viewpoint  -Models  Consistencies and Correspondence  Document rationale for architecture key decisions |

### ADD, TOGAF and IEEE 42010 Steps Comparison

ADD Step 1, confirmation of sufficient requirements information, is similar to select phases in TOGAF’s ADM. In both ADD and TOGAF’s Preliminary Phase, business requirements are listed. In ADD, the prioritization occurs in this step; however, TOGAF prioritizes migration projects in Phase F: Migration Planning. Also, TOGAF is unique as it suggests a continuous process to refine the requirements. In the ADM, this is known as the requirements management. IEEE 42010 standard does not provide a process for gathering requirements; however, requirements are captured as concerns for example: reliability, security, maintainability, etc. IEEE 42010 standards also require the documentation of the relation between concerns and stakeholders. Architecture views allow for a stakeholder to have a view of the entire system for each concern.

Step 2 in ADD focuses on decomposition of a system element on the current architecture. This differs greatly from TOGAF because TOGAF goal is to create an enterprise architecture. If any improvements are to be made to an existing architecture within TOGAF, the changes must be proposed, agreed upon, and decided if the architecture team will go through another ADM cycle. The only decomposition that occurs in TOGAF is deciding if building blocks can be broken down into further building blocks. Decomposition of the architecture is not in the scope of IEEE 42010.

Step 3 in ADD, identifying architectural drivers, requirements are ranked a second time for the elements that are decomposed. Again, TOGAF does not focus on decomposing software elements. The ranking in ADD depends on the impact on the architecture. This is similar to TOGAF’s Migration Planning where the team does a cost/benefit analysis of the implementation plan. The selected requirements in ADD are called candidate architectural drivers. However, in TOGAF, the business goals drive the overall architecture. Within the IEEE 42010 standards, architecture drivers are captured within an architecture description as concerns. IEEE 42010 standards do not provide a process for ranking concerns, however, not all concerns are captured within the architecture description. IEEE 42010 standards specify the documentation of concerns that are considered fundamental to the stakeholders.

Step 4 in ADD chooses a design concept that satisfies the architectural drivers. The major elements and relationships are chosen by selecting the design constraints and quality attribute requirements. In order to develop the architecture in TOGAF, reference models, viewpoints, and tools need to be selected in the beginning of Phases B, C, and D. This is where the design of the Target Architecture begins. ADD heavily analyzes the relationships between the elements, which is similar to TOGAF’s Content Framework and Reference Model. The TOGAF Content Framework contains three categories that show the relationships between them: deliverables, artifacts, and building blocks. Also, the TOGAF Reference Model shows the existing relationships between the entities. IEEE 42010 does not provide a method for choosing a design concept based on the concerns, however, the architecture is documented within a viewpoint or multiple viewpoints based on stakeholder concerns. IEEE 42010 standards require the documentation of key architecture decisions. Rationale regarding the choice of a design concepts would documented within the architecture description as a key decision.

Step 5 in ADD instantiates architectural elements and allocates responsibilities. ADD divides functionalities according to their types from the functional requirements so that every parent element has a sequence of child elements. TOGAF is not as detailed in this process, but TOGAF similarly groups functionalities by differentiating the areas of architecture into these domains: business, data, application, and technology. To implement changes in TOGAF, they are logically grouped together in Phase E: Opportunities and Solutions. IEEE 42010 standards does not provide a process for dividing functionality, however an architectural view can depict more than one concern. The relation between concern and architectural view can be found within the viewpoint of an architectural description allowing for the stakeholder to easily identify which view corresponds to a concern of interest. IEEE 42010 standard does not provide a method for grouping concerns into viewpoints.

Step 6 in ADD defines the interfaces for the instantiated elements. The services and propertiesrequired, known as the element’s interface, are defined. This concept is similar to defining the Architecture Principles in TOGAF. For example, technology independence has an implication for commercial off-the-shelf applications. It is also similar to the concept of a viewpoint within the IEEE 42010 standards in that a viewpoint defines the contents of a view. It differs, in that ADD is defining an architectural element and a viewpoint within IEEE 42010 is defining an architectural description element.

Step 7 in ADD verifies and refines requirements and makes them constraints for instantiated elements. This step is similar to TOGAF’s continuous process of requirements management by ensuring the requirements are met. IEEE 42010 does not provide a process for refining requirements, however, an architecture description can be used to refine requirements. Stakeholders can use a view, and the models contained within the view, for a particular requirement or concern and analyze the architecture for improvements.

Step 8 in ADD repeats Step 2-7 until the next element is decomposed. In TOGAF, the process is similar as both ADD and TOGAF are iterative. In addition, breadth, depth, time period, and recency are dimensions used to define and limit the scope of an architecture. IEEE 42010 standards for an architecture description is not an iterative process; however, the architectural description should be used and updated throughout the system lifecycle.

## Work-products

The following table is a list of the inputs, outputs, processes, work-products for ADD, TOGAF and IEEE 42010. Note, TOGAF contains only the major inputs, outputs, work-products, and processes.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **ADD** | **TOGAF** | **IEEE42010:2011** |
| **Inputs** | * Stakeholder desires   + Functional Requirements   + Quality Attributes * Development issues   + Design Constraints | * Request for Architecture Work * Architecture Principles * Reusable Architecture Building Blocks * Product Information * New Technology Reports * Architecture Requirements Specification | * System of interest * Stakeholders * Concerns * Relation between stakeholders and concerns * Rationale for key decisions |
| **Outputs** | * Completed Software Element * The Data and Functionality Required to Complete the Software Element * Definition of the Element * Relationship between the Element and Other Parts of the System | * Statement of Architecture Work * Business Scenario/Architecture Vision * Business Architecture * Technology Architecture * Impact Analysis * Architecture Contract * Requirements Impact Statement | * Architecture Description containing inputs as well as viewpoints, views, models, and correspondence rules. |
| **Processes** | * Iterative | * Iterative   + Architecture Development Iteration   + Transition Planning Iteration   + Architecture Governance Iteration   + Architecture Capability Iteration | * Continuous documentation |
| **Work Products** | * A overall, rough architectural design that fits the desires of the stakeholders within the priories allotted | * Deliverable * Artifact * Building Block | * Architecture View * Architecture Models |

### ADD, TOGAF, IEEE 42010 Work Products Comparison

The three main inputs of ADD are functional requirements, quality attributes, and design constraints. Functional requirements specify the stakeholder needs and conditions to meet the requirements in the system (Software Engineering Institute, 2006). These descriptions can be found in TOGAF’s Architecture Requirements Specification. The quality attributes in ADD refer to a system’s various properties such as buildability, availability, portability, performance, security, testability, usability, and capacity. These can also be found in TOGAF’s Architecture Requirements Specification. The design constraints in ADD can be found in TOGAF’s Request for Architecture Work, Building Blocks, and Architecture Requirements Specification. The three main inputs of ADD can be found throughout the ADM in the different phases and their required inputs and resulting outputs. IEEE 42010 standards provide detailed specifications for the content that is required for architecture descriptions, although not specified as inputs, the main content to initiate an architecture description can begin with identifying a system of interest, identifying stakeholders, identifying concerns, defining of the relations between stakeholders and concerns, and providing rationale for key decisions.

The main outputs of ADD are completed software element, the data and functionality required to complete the software element, definition of the element, relationship between the element and other parts of the system. In TOGAF, these are found in the Content Metamodel and Technical Reference Model. Once an architecture description if fully developed, it will include all of the required items specified within the IEEE 42010 standards as well as additional suggested content. At a high level, an architecture description will include stakeholders, concerns, views, viewpoints and models. Each concern should be captured within a view such that a stakeholder will have a view of the entire system from the perspective of the concern.

ADD’s process is iterative and the process steps were mentioned earlier. After Steps 2 through 7 can be repeated. TOGAF is also an iterative process, but contains a suggestion of iteration cycles as the phases are grouped together in related activities (Architecture Development Iteration, Transition Planning Iteration, Architecture Governance Iteration, and Architecture Capability Iteration). IEEE 42010 standards for documenting an architecture description indicate that the architecture description should be used and updated throughout the lifecycle of the system of interest. The document should be updated as the architecture is updated.

Finally, the work product from ADD is an overall, rough architectural design that fits the desires of the stakeholders within the priories allotted. The equivalence of this in TOGAF is the finalized architecture. However, within TOGAF, work products are defined in the Architecture Content Framework as a deliverable, artifact, and building block. The main work products within the IEEE 42010 standards for architecture description are architecture views and architecture models. Architecture views contain the architecture models. The architecture viewpoint is used as a legend for the view and describes the stakeholders, concerns, and models that are captured within the view.

## Pros and Cons

The following section pertains to the advantages and disadvantages of ADD v2, TOGAF v9.1, and IEEE 42010.

### ADD v2

|  |  |
| --- | --- |
| **ADD v2 Pros** | **ADD v2 Cons** |
| Utilizing the ADD method of design development provides a few architectural advantages.   * It allows for development to be based on stakeholder prioritization. * Quality elements and requirements focus solves system issues with base system requirements for increase chance of user appreciation and profitability * Iterative approach creates consistency and ease of timing for development. * It focuses on one part of the system after the first iteration (whole system design) creates specification of solutions. | However, ADD is not a perfect design strategy. There are some drawbacks to this type of development.   * It is only a rough sketch of an architecture is created and does not produce a completely detailed plan. * It can become too specific and developed lack system-wide design. * A clash between stakeholders and developers could become an issue if priorities are not aligned. |

### TOGAF

|  |  |
| --- | --- |
| **TOGAF v9.1 Pros** | **TOGAF v9.1 Cons** |
| Using TOGAF has a few benefits (The Open Group, 2011-b):   * It may provide increased time to market. * Since TOGAF is based on best practices, it will provide consistent business processes and information. * It is reliable and secure. * The organization will have a more efficient business operation and IT operation. * It is flexible; it can be used for any organization of any size in any industry. * It can be used alone or complementary with other frameworks. * The organization will have a better return on investment. * TOGAF reduce risks for future investment. * Overall, it is faster, simpler, and cheaper procurement. | However, there are some drawbacks to TOGAF.   * TOGAF is not a one size fits all. It requires tailoring to your organization * TOGAF is long, and complex with over 1000 pages. * It is incomplete, but always evolving. * It can be difficulty to learn. Therefore, it is recommended that it should be taught in a training course and users should get a certification. |

### IEEE 42010

|  |  |
| --- | --- |
| **IEEE:42010 Pros** | **IEEE:42010 Cons** |
| The advantages of IEEE are:   * It provides a central documentation of a systems architecture. * Architecture is captured with context, information such as stakeholder, concerns and decisions are documented * It provides stakeholders with a method for viewing the system architecture from the perspective of a stakeholder concern. | However, the disadvantage is:   * It does not provide a process for designing architecture. |

## 

## When/how to use each better in the field

ADD is best used for initial architectural designing. Once the system is designed to fit the stakeholder’s requirements, then the intricacies of the system and the documentation of developing the system need to be addressed.

TOGAF is best for organizations wishing to build an enterprise architecture. It can be used alone or can be complementary used with other frameworks. It can also be used in any industry as long as it is tailored to the organization’s needs. It is for professions that are involved in architecture projects in planning, execution, development, delivery or operation (The Open Group, 2011-b).

IEEE 42010 standards for an architecture description is best used for documenting architectures and can be used in combination with ADD. Creating standardized documentation will facilitate reuse of the architecture within an organization. It will facilitate communication between stakeholders throughout the system lifecycle and provide a method for stakeholders to view an architecture based on a concern.

# Lessons Learned

## Timothy Cioffi-Dinkel

These types of architectural design development were foreign concepts to me before this homework. My experience with the design of a system usually occurred well after the design process was completed and the architecture was ready to be developed. With this I learned how to successfully develop a system that either uses the interest of stakeholders or the interest of business and IT. ADD would be used by me in the future to design an architecture which can impress those who have the highest interest for a system. This will benefit me in a career in a senior development position as I will be able to design a software that will be what my bosses want. TOGAF will help me if I ever am part of an executive team for a business that requires a new system to be developed. I will be able to help design a system that will benefit my company and its future as well as benefit my IT department in their future needs. IEEE provided me with the insight into how documentation, when designing a system, should be done for the design of a system to help with future development as well as current development.

## Anthony Farina

Prior to this homework assignment, I had no knowledge of software architecture, architecture design, architecture framework, or the industry standards regarding architecture. This assignment has taught me about three different standards. The ISO/IEEE standard taught me about the standards of software architecture as a whole while the ADD and TOGAF standards taught me about the standards of architecture frameworks. Learning all of these standards was very exciting because I can use these standards while practicing software engineering in the real life. Working at an organization that does not have a software architect or doesn’t document a software architecture, this information will be very valuable to me. What I would like to learn more about regarding software architecture is how to actually implement and design software architecture while using these standards.

## Joanna Hang

I had no prior knowledge regarding software architecture frameworks. I learned about software architecture. I learned the differences between ADD, TOGAF, and IEEE and when it is best to apply each. ADD was relatively easy to understand and interpret as the general purpose was software decomposition. TOGAF and IEEE were more complex. I would most likely use the practices in ADD in my current workplace. Although TOGAF states that it can be applied to any organization of any size, I feel it is more suitable for larger businesses like Fortune 500 companies. I liked TOGAF’s idea of potential Reusable Building Blocks and having an Architecture Repository as it would reduce time in creating new architectures by analyzing what the organization currently developed. From IEEE, I learned about architectural standards.

## Lourdes Lopez

Prior to beginning this assignment I had very little experience with architecture design. Most projects that I have worked on mimicked the architecture of other existing applications within the organization with no consideration to the new applications unique quality attributes. Throughout the process of completing this homework assignment, I have learned about 3 techniques for architecture design that will significantly impact and improve the way I approach architecture design. I am looking forward to learning about ADD in more detail as I work through the second homework for this class. Going forward, in my professional work I can see myself using ADD for architecture design for smaller applications. TOGAF will help me look at architecture from an organizational perspective and IEEE 42010 will help me with documentation and communication with stakeholders regarding an architecture.

## David Sullivan

Before this assignment, I had no prior knowledge of architecture frameworks or methods. I learned that there can be various approaches to developing architecture. ADD could be easily implemented within a small project or organization to decompose the software architecture whereas TOGAF and IEEE 42010 are more suited for more complex projects or large organizations. TOGAF is more beneficial for organizations which need to address the concerns of a large organization because it can be adapted in combination with other industry enterprise frameworks. IEEE 42010 defines architectural components that can assist the architect to model the system architecture. As a start, I am interested applying the ADD method to decompose architectural elements. I also want to learn more about the architecture principles we discussed in TOGAF by defining our enterprise architecture and benefit from the creation of building blocks and their reuse. Also, I see value of the creating architectural documentation to understand how the business processes and technology work together and also how this documentation will be a useful tool when working with management and other stakeholders.

## Group Experience

Together as a group, we learned about software architecture, architecture design, standard for software architecture, and standards for architecture for architectural frameworks. The group learned about Attribute-Driven Design (ADD), Version 2.0 which is about recursively breaking down a system to create an architecture based on quality attributes. We also learned about TOGAF, which is an architecture framework focused on maintaining and developing enterprise architecture. The third topic that was researched was the IEEE 42010 standard, which focuses on creating and maintaining architectures through architecture descriptions. Lastly, we learned about the differences and similarities between these standards. With this knowledge the team can now improve their software engineering craft.

# Reference List

ISO/IEC/IEEE 42010:2011, ISO/IEC/IEEE 42010:2011: Systems and software engineering Architecture description (1st Edition 2011-12-01).

Software Engineering Institute. (2006). Attribute-Driven design (ADD), Version 2.0.

The Open Group. (2011). *TOGAF ®, an open group standard.* Retrieved from: <http://www.opengroup.org/subjectareas/enterprise/togaf>

The Open Group. (2011). *TOGAF module 1 management overview* [PDF document]. Retrieved from:<http://www.togaf.info/togafSlides91/TOGAF-V91-M1-Management-Overview.pdf>

# Team Charter

|  |  |  |
| --- | --- | --- |
| **Course Title** | CPSC 545 Software Design & Architecture | All team members participated in the creation of this charter and agree with its content.  **Date** 01/31/2016 |
| **Instructor** | Dr. Chang-Hyun Jo |
| **Course Dates** | 01/24/2016 – 05/14/2016 |

**Team Members** (Contact Information)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Address (city, state, country) | Phone | Cell | Email |
| Timothy Cioffi-Dinkel | San Diego, CA | 951-775-6150 | 951-775-6150 | timothycioffi@csu.fullerton.edu |
| Anthony Farina | Bear, DE | 302-299-5495 | 302-299-5495 | afarina@csu.fullerton.edu |
| Joanna Hang | Fullerton, CA | 714-686-7401 | 714-686-7401 | johang@fullerton.edu |
| Lourdes Lopez | Turlock, CA | 209-205-0581 | 209-205-0581 | lourdeslopez@csu.fullerton.edu |
| David Sullivan | Fullerton, CA | 714-686-9488 | 714-686-9488 | dsullivan@fullerton.edu |

**Team Member Skill Inventory** (Areas individual members can contribute)

|  |  |
| --- | --- |
| Timothy Cioffi-Dinkel | * MySQL * Java/C++ * XML, HTML/CSS, PHP * Photoshop * Game Development, Unity/C# |
| Anthony Farina | * MySQL - Database administration and design * Java * Android Development * Microsoft Office * Google Web Toolkit |
| Joanna Hang | * SQL Server - Database administration and design * SQL Reporting Services * HTML, CSS, JavaScript, jQuery, Bootstrap, and Less * Microsoft Office * Adobe Photoshop, InDesign, Digital Publishing Suite |
| Lourdes Lopez | * Data Analytics - Complex queries using T-SQL or PL-SQL * Business Intelligence tools - Business Objects Web Intelligence, Business Objects Universe Design, Microsoft SSRS and Crystal Reports * Web Application Development with C# ASP.NET and JavaScript * Experience with agile development using Scrum methodologies * Microsoft Visio for documenting business process |
| David Sullivan | * iOS Development * Android Development * Java * C#, ASP.NET, JavaScript, Bootstrap * Microsoft Web API |

**Team Goals** (Project goals, team process goals, quality goals, etc.)

|  |
| --- |
| * Combine the unique skill sets of each of our team members, so that we can produce a successful project as well as learn from each other's areas of expertise. * Learn about the software process by reviewing course material as well as experiencing the process as we work towards our course assignments. * Improve upon our existing knowledge of software development processes so that we will develop software in a more organized manner. * Keep track of our weekly assignments and goals so that we meet our deadlines and are well prepared for upcoming assignments. * Establish and maintain strong communication between teammates in order to facilitate success. |

**Team Roles** (Define roles of each member to achieve goals)

|  |  |
| --- | --- |
| Lourdes Lopez  Team Lead | * Act as the team liaison which includes communicating with the professor and/or class regarding the team’s progress and/or asking questions * Turn in assignments, take a screenshot of submitted assignment, and e-mail team members * Collaborate with Facilitator on agendas * Manage project timeline |
| Joanna Hang  Recorder | * Take meeting notes and put them in Google Drive * Document ideas and key decisions from meetings * Ensure team members know the purpose and goals for assignments |
| Timothy Cioffi-Dinkel  Facilitator | * Inform team of any changes to meetings * Assist team lead with agenda before meetings * Coordinate video calls/start video calls * Begin/end conference calls with a check-in with each team member * Ensure all team members participate and ask everyone for input. * Cover topics on the agenda * Manage meeting times |
| David Sullivan  System Architect | * Ensure the project is within the scope of technical requirements * Review project through a technical standpoint |
| Anthony Farina  Quality Assurance | * Ensure process meets users’ quality expectations * Extract collaborative assignments from Google Drive and convert documents to appropriate document type (Word, Excel, PDF, etc.) * Perform final review by checking format and reviewing assignments * Upload final copy to Google Drive and inform Team Lead that assignment is ready to turn in |

**Ground Rules** (Meeting schedule/locations, attendance expectations, agenda, assignment completion, communication methods, etc.)

|  |
| --- |
| * All team members will check Titanium daily. * All team members will check emails at least once per day and reply within 24 hours. * All team members will meet on Tuesday’s at 6:00 PM PST using Google Hangouts. * Google Hangouts will be the main tool for communication between team members. * Google Hangouts will be installed on each member’s mobile phone. * All team members will check Google Hangouts at least twice per day. * All team member will complete collaborative assignments using Google Drive. * All team members must be respectful, honest, and clear to all other team members at all times. * No plagiarism or cheating. Each team member will proof read each other’s work to prevent this. * Group will rotate team leaders each semester. * All team members will communicate and ask questions. * All team will equally contribute, show effort, and will not leave the work to a few individuals. * All team members will be supportive, willing to help and assist other team members. * Each member must be open to constructive criticism from each other team member. * Each member is responsible for their tasks. Tasks must be completed by the deadlines set by the professor and or the group. * If emergency arises that may affect deadlines, meetings, etc., team member will at least inform the team leader or if possible the entire group via Google Hangout about the situation so the group can plan accordingly. |

**Time Commitments/Availability** (Pacific Time)

|  |  |
| --- | --- |
| Timothy Cioffi-Dinkel | * M-F 5pm-8pm * Sat-Sun all day |
| Anthony Farina | * M-F 5pm-8pm * Sat-Sun all day |
| Joanna Hang | * M-F 5pm-8pm * Sat-Sun all day |
| Lourdes Lopez | * M-F 5pm - 8pm * Sat-Sun all day |
| David Sullivan | * M-F 5pm - 8pm * Sat-Sun all day |

**Conflict Management** (What are potential conflicts that might arise among or between team members during this course? How will team members deal with these and other conflicts?)

|  |
| --- |
| * In order to avoid conflict among team members, roles and responsibilities should be equally designated. * If conflicts arise, concerns should be submitted to the agenda and will be addressed to the whole team to settle conflicts. |

**Risk Management** (What are potential barriers to the achievement of these goals?)

|  |
| --- |
| * Identify risk factors within team (meeting time conflicts , external commitments to work and family) and within the software process (training, organizational policies, configuration) * Prioritize tasks in order of importance and communicate to stakeholders * When problems arise, communicate with team and stakeholders * Actively monitor progress and reevaluate requirements |

**Team Evaluation Criteria** (List evaluation criteria that will be used to evaluate team members objectively.)

|  |
| --- |
| * Evaluate each individual team member on their own work. * Evaluate each individual team member on their participation in group discussions. * Evaluate each individual team member on their ability to collaborate on the group project. * Evaluate each individual team member on their ability to produce material in a timely matter. * Evaluate each individual team member on their ability to fulfill role. |

# Team Evaluation

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Members** | Timothy Cioffi-Dinkel | Anthony Farina | Joanna Hang | Lourdes Lopez | David Sullivan | **Total** | **Comments on Your Evaluation on Team** |
|  |
| **Evaluators** |
| Timothy Cioffi-Dinkel | 100 | 100 | 100 | 100 | 100 | **500** | Great work, solid team |
| Anthony Farina | 100 | 100 | 100 | 100 | 100 | **500** | Everyone did their part with no trouble. |
| Joanna Hang | 100 | 100 | 100 | 100 | 100 | **500** | Great work! |
| Lourdes Lopez | 100 | 100 | 100 | 100 | 100 | **500** | Great job team! |
| David Sullivan | 100 | 100 | 100 | 100 | 100 | **500** | Everyone stepped up and worked hard |
| **Total** | 500 | 500 | 500 | 500 | 500 | 2500 |  |
| **Max** | 500 | 500 | 500 | 500 | 500 | 500 |  |
| **Average** | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 500.00 |  |
| **Percent** | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% |  |
| **Signature** |  |  | sig.jpg | Untitled.png |  |  | Work was divided amongst all team members and exceeded our expectations. |
| **Comments on Your Score Earned from Team** | Fair enough | I'm happy. | Great! | Fair | I'm happy |  |  |