AN APPROACH FOR SOFTWARE ARCHITECTURE BY UNDERSTANDING VALUE REQUIREMENTS, DEVELOPING VALUE PROPOSITION, AND SUBSEQUENTLY REALIZING VALUE

Anand Kumar, Doji Samson Lokku, Nikhil Ravindranath Zope

Desk No: 313, Tata Research Development and Design Centre, 54 B Hadapsar Industrial Estate, Pune 411013, India.

Email: anand.ar@tcs.com, doji.lokku@tcs.com, nikhil.zope@tcs.com

ABSTRACT

Software Architecture is both a design activity (process) as well as the schema of fundamental things about a system (work product). As a design activity, architecture is the act of creating a representation of an unknown and original object whose properties (like technical aspects, formal and spatial structures) must be well enough understood in advance. As a work product, software architecture is the structure of the components of a system, their interrelationships, externally visible properties of those components and principles and guidelines governing their design and evolution over time. Handling this duality and realizing architectural designs that improve the value of the solution within cost limitations; provisioning for evolution over the system lifetime; considering the needs of all stakeholders; and ensuring that the system is well matched to its environment are the typical responsibilities of Software Architects.

The outcome of Software Architecting process is Software Architecture. Traditionally, this process provides general guidance to the Software Architect and utilizes an envelope of practices and design patterns that govern the Software Architecture creation. Its purpose is to aid the Software Architect to synthesize a solution that satisfies the requirements and it is the responsibility of the Software Architect to identify the right practices/patterns necessary for creating an appropriate solution. While most of the existing practices look at developing an Architecture that satisfies the requirements identified by the Software Architect, we propose a value understanding, value proposition and value realization based approach for Software Architecting that is based on the value co-creation system that exists in the software development and usage life cycle.

In this paper, we discuss about the theoretical framework necessary for such a Value based approach. This theoretical framework is based on insights arrived at by asking four questions that needs to be answered for the software to succeed economically. These four questions are:

a) What are the benefits and how to discover, diagnose and understand these benefits?

b) What are the carriers for achieving these benefits? How can one derive these carriers of value?

c) What are the cumulative net benefits that should be delivered by the software?

d) How does one compose and deliver the software so as to realize these benefits?

The basis of the framework is the values viewpoint for creating and describing software. We illustrate our theoretical framework and approach by architecting a task automation system.

Keywords – Value, Quality, Value Understanding, Value Carriers, Value Proposition, Value Realization, Software Architecture, Software Architecting, Value based Approach

1. Introduction

Software architecture as a discipline and its work products has been gaining prominence over the last couple of decades. In Software architecture practice, the emphasis on good software architectures has resulted in the need for a systematic approach for architectural design. According to Garlan, *"Good architectures are increasingly based on architectural styles (like client-server, blackboard, event-based, messaging, object-based architectures) that typically specify the design vocabulary, its usage and semantic assumptions. However, practice has shown that each architectural style is appropriate for certain purposes, but not for others"*. According to ISO 42010, *"The complexity of man-made systems has grown to an unprecedented level. This has led to new opportunities, but also increased challenges for organizations that create and utilize systems."*

Software Architecture is both a design activity (process) as well as the schema of fundamental things about a system (work product). As a design activity, architecture is *"The act of creating a representation of an unknown and original object whose properties (like technical aspects, formal and spatial structures) must be well enough understood in advance"*. As a work product, software architecture is *"The structure of the components of a system, their interrelationships, externally visible properties of those components and principles and guidelines governing their design and evolution over time"*. Handling this duality and realizing architectural designs that improve the value of the solution within cost limitations; provisioning for evolution over the system lifetime; considering the needs of all stakeholders; and ensuring that the system is well matched to its environment are the typical responsibilities of software architects.

The definition of a software in terms of its architecture and constituent elements depends on the software's stakeholders' interests and responsibilities. It includes assessments and decisions to select the elements that comprise the software, reference models, various properties of discourse, operational concepts, principles and guidelines. Once the software is realized and put to use, users and other stakeholders observe the effects of the software on the problem area that is addressed by the software. These effects occur because of the various software quality characteristics as a result of which users and other stakeholders experience benefits.

While the objective of software architecting is to create a representation of an unknown software whose properties must be well enough understood in advance, the problem of software architecting involves specification of the proposed software and prediction of its properties before its embodiment. It involves understanding how the resources necessary for realizing the software is organized formally, semantically and how it is represented and how these representations can be acted upon. It is a group activity involving classes of people working in unison.

2. Current Software Architecting practices

According to Maier, the four most important methodologies in the process of architecting are characterized as normative, rational, participative, and heuristic. The normative technique is solution-based; it prescribes architecture as it "should be;" that is, as given in handbooks, civil codes, and in pronouncements by acknowledged masters. The rational technique is method/rule based; it prescribes architecture as scientific and mathematical principles to be followed in order to obtain a solution to the stated problem. Both the normative and rational methods are analytic, deductive, experiment-based, easily certified, well understood, and widely taught in academia and industry. The participative technique is based on obtaining consensus between multiple stakeholders and addressing the complexities created by these stakeholders; it prescribes architecture as creating a system in which widespread cooperation is necessary. The heuristics technique is based on "common sense," that is, on what is sensible in a given context. Contextual sense comes from collective experience stated in as simple and concise a manner as possible. Both the participative and heuristics methods are nonanalytic, inductive, difficult to certify, less understood, and is seldom taught formally in either academia or industry.

According to Jamshid Gharajedaghi, the systems methodology is a holistic language of interaction and design developed to face the dilemma of systems where the whole is becoming more and more interdependent while the parts display choice and behave independently. He conceived an iterative methodology based on an iterative process of applying simple rules to architect systems. Iterations of structure, function, and process in a given context would examine assumptions and properties of each element in its own right, then in relationship with other members of the set. Subsequent iterations would establish validity of the assumptions and successively produce an integrated design of software. Accordingly, the concerns that he addresses in his iterative process are: a) Function - Concerns of goal/purpose of software, b) Structure - Concerns of resources while the software is used, c) Process - Concerns of work while the software is used and d) Context - Concerns of environment while the software is used.

According to Faislander, software architecting is a creative activity that bridges the gap between imagination (art) and the tangible reality (Science). Accordingly, defining the architecture of a system consists of imagining, in a structured way, solutions based on concepts that are supported by universal principles and rules, using predefined generic constructs, patterns and heuristics. This is achieved by transforming the requirements model of a system to a logical and physical architecture using intermediate models. These intermediate models are functional and behavioural models which are assigned to system elements of the physical architecture. The logical and physical architecture definitions are arrived at thru multiple iterations so as to produce a consistent definition.

According to the Software Engineering body of Knowledge, four main methods help guide the software architecting process. The function oriented (structured) design approach is a classical method in which decomposition of a system centers on identifying the major software functions and then elaborating and refining them in a top-down manner. In the Object oriented design approach, inheritance and polymorphism play a key role in developing a set of objects that satisfy the requirements of the software. The data structure oriented design approach is another method in which decomposition of a system centers on identifying the data structures that the software manipulates and then developing the software's control structure based on these data structures. The component based design approach addresses issues related to providing, developing and integrating software components (which are the loci of computation or data store) so as to compose the desired software.

According to Inverardi and Wolf, a Chemical Abstract Machine model can be adopted to architect a system. This model corresponds to an operational specification of software and does not bias the software towards any particular computation model. In this method, software architecture is fashioned after chemicals and chemical reactions. The states of the machine are chemical solutions where floating molecules can interact according to a stated set of reaction rules. This method helps define the structure and abstract behaviour of the software in a modular form which can then be refined by the addition of molecules (software design elements) and transformation rules for composing the identified molecules together. They can use this method to compose specifications from parts and define rules for this composition which can then be subjected to correctness analysis and comparative analysis.

According to Luckham, Vera and Meldal, software architecture is a specification of the components of a system and the communication between them. They propose three alternative concepts of architecture: Object connection architecture, Interface connection architecture and plug and socket architecture. An object connection architecture consists of the interfaces and connections of a system represented in the form of an object oriented systems comprising a) object oriented interfaces, b) set of modules that conform to these interfaces, c) connections between the modules. Interface connection architecture defines all connections between the modules of a system using only the interfaces which specify both the provided and required features. Plug and socket architectures are interface connection architectures in which interfaces are allowed to provider services wherein a service denotes a number of individual connections between interface features.

According to Garlan and Allen, software architecture is an interconnected collection of object instances. This postulation provides a formal basis for reasoning about the properties of the system. Accordingly, Software architecture is a hierarchical structure of components, connectors and configurations. To support this formulation, they provided a formal abstract model of system and its behaviour. The vocabulary of this formulation is defined by the basic architectural types: Components (the set of components in the architecture), Connector (set of connectors in the architecture), Port (set of ports of a component), Role (set of roles of a connector), Configuration (set of components and connectors that together form a configuration) and Binding (the glue to the connector).

According to Garlan, the central architectural challenge in mobility is to maximize the use of available resources and to minimize user distraction and drains on user attention. He proposes the concept of personal Aura as an approach to marshal the computational resources necessary to support a user's tasks in their mobile devices. He proposes an architectural framework that provides the necessary features and interfaces required at application and system level so that the user is protected from the heterogeneity of the computing environments and resource variety in their environments. Further, the various user tasks in their mobile environment are described as coalitions of abstract services which need to be available for the user to perform their tasks when they are mobile. The key ingredients of this approach are the explicit representation of user tasks as collection of services, context observation that allows task configuration, environment management for resource monitoring and adaptation.

According to Guttag, a software system comprises of a state and a set of mechanisms for changing and extracting information from the state. External world interactions are again mechanisms for manipulating the state. Accordingly, the current state of a system is the result of the mechanisms that were performed on the initial state. As a result, the architecture of a system comprises of two specifications, a) the possible software system states (or a superset of these states) and b) the mechanisms that deal with the state. Accordingly, the architect needs to decide on the abstractions that should be developed and the functionalities of the operators associated with these abstractions. According to Guttag, such an approach will help the architect to clarify when an architectural decision had to be made, what are the alternatives and what are the ramifications of the architectural choices that were made.

According to Kruchten, Software architecture deals with the design and implementation of the high level structure of the software. It is the result of assembling architectural elements so as to satisfy the functionality, performance and other non-functional requirements of the software system. Kruchten proposes a model composed of multiple views or perspectives for describing the architectural decisions made by the architect. These views/perspectives are: Logical view which is the object model of the design, Process view which captures the concurrency/synchronization aspects of the design, Physical view which describes the mapping of the software into hardware, Development view which describes the static organization of the software in the developing environment and the use cases/scenarios view which captures the decisions made by the above views. Each view is defined by using a set of elements of the form {components, containers, connectors} and is described by a blueprint.

Boehm proposes a value based approach to software engineering which is applicable for software architecture as well. According to Boehm, a value based approach helps by providing new perspectives, tools, skills and success criteria for the software engineering/architecture activities. Accordingly, Boehm proposes key foundational elements as part of his value based approach. These seven key elements are: Benefits realization analysis, Stakeholder Value Proposition Elicitation and Reconciliation, Business Case Analysis, Continuous Risk and Opportunity Management, Concurrent System and Software Engineering, Value Based Monitoring and Control and, Change as Opportunity. Collectively these key elements for a framework for value based software engineering/architecture.

According to Clements and Bass, software architecture is the result of a set of business and technical decisions. There are many influences at work in its design and the realization of these influences will change depending on the environment in which the software architecture is required to perform. Further, they propose a set of activities that are involved in creating the software architecture. These activities are: a) Creating the business case for the system, b) Understanding the requirements, c) Creating or selecting the appropriate architecture, d) Documenting and communicating the architecture, e) Analysing or evaluating the architecture, f) Implementing the system based on the architecture and, g) Ensuring that the implementation conforms to the architecture. The various architecture activities have comprehensive feedback relationships with each other which enable an architect to converge onto a good architecture.

3. PROPOSED VALUE UNDERSTANDING, VALUE PROPOSITION AND VALUE REALIZATION BASED SOFTWARE ARCHITECTURE APPROACH

Value Understanding, Proposition and Realization based Software Architecting (VURSA) approach enables architects to look beyond requirements. It is based on identifying the set of concerns an architect needs to consider and address in order to deliver value to stakeholders. It proposes that architects should:

- Move from addressing requirements to delivering value to stakeholders
- Move from performing a series of activities to achieving qualities
- Concern itself with character of the software which is above and beyond its common use
- Understand what is beneficial to stakeholders in tackling a specific problematic situation
- Go beyond benefits and look at net-benefits to stakeholders
- Work towards delivering these net-benefits

VURSA approach is based on the premise that purpose of the software is to deliver value to stakeholders and software architecture ensures that this purpose is achieved. The subsequent sections will discuss about the conceptual foundations and the various perspectives that the VURSA approach brings to the fore. The basis for this approach is understanding the benefits expected by the software users; deriving the net benefits that the software should deliver (including benefits to the providers) to various stakeholders; correlating the net benefits to the quality characteristics that the software must possess; developing software components that host these quality characteristics; composing the software components to realize the software and finally, using the software in such a way that the perceived value is realized by all stakeholders. In essence every stakeholder who is affected by the software collectively performs activities that enable all the stakeholders to realize the respective value from the software.

4. THE INTRICACIES OF THE SOFTWARE ARCHITECTING APPROACH

Software Architecting is an intellectual works that involve the participation of many stakeholders (developer side stakeholders, user stakeholders, governance stakeholders, etc) in ways that dictate and constraint the purpose and the design of the software as a whole. A software has to be conceived to accommodate all the affected stakeholders needs, perceptions, own goals and measures of value which are often diverse and incompatible. One way to ease such a situation is through economics. The VURSA approach is based on insights arrived at by asking four different questions that needs to be answered for the software to succeed economically. These four questions are:

a) What are the benefits and how to discover, diagnose and understand these benefits?

Anyone who gets affected by the software or participates in it in any way is a beneficiary. For the software to be beneficial to someone, it is necessary to figure out the benefits in terms of (1) usefulness in satisfying a user need,

(2) relative importance of the need being satisfied, (3) availability relative to when it is needed, and (4) the cost of ownership. These economic considerations dictate user satisfaction and user's ability to satisfy their purposes.

b) What are the carriers for achieving these benefits? How can one derive these carriers of value?

For the software to live-up to the expectations in terms of delivering benefits, it should exhibit the appropriate software quality characteristics. Quality characteristics determine how well the software is able to create benefits and it defines how the software architect addresses a given scenario. They differentiate software from each other and measure excellence in a chosen dimension. Quality attributes are generally considered important for obtaining a software of good quality—various "ilities" (maintainability, portability, testability, traceability), various "nesses" (correctness, robustness), including "fitness of purpose."

c) What are the cumulative net benefits that should be delivered by the software?

Software Architecture deals with the design of the software. The process of creating the value proposition of the software starts with understanding the quality characteristics, value creation context and problem situation that needs to be addressed; diagnosing it using archetypes, patterns and existing models; and synthesizing an approximate symbolic (or) mathematical (or) conceptual (or) physical representation. In summary, a value proposition is a multi-faceted artefact produced by the design process and composed of relatively independent and orthogonal facets/models of the software.

d) How does one compose and deliver the software so as to realize these benefits?

Essentially, Value proposition is used to create a scheme of things of the software elements and to provide the criteria for composing these software elements. It is used to verify and validate the various software components and their interconnections. This feedback is then used in the next iteration of the software design cycle, particularly when problems or challenges are identified. Software system is the output that is created by the software realization process which transforms design inputs to an acceptable output. This system is characterized by quality attributes that are of value to its users. When this system is put to use, it creates value and related experience to the users.

5. THE FOUR PERSPECTIVES

The insights obtained from answering the four different questions serve as four different interrelated perspectives as shown in fig 1. While two perspectives focus on knowledge concerns, the remaining two focus on operational concerns. The two knowledge perspectives are 'value understanding' (Q1 in fig 1) and the 'value carriers' (Q2 in fig 1). The two operational perspectives are the 'Value Proposition' (Q3 in fig 1) which is the specification of the value delivered by the service and 'Value Realization' (Q4 in fig 1) which is the instantiated design for the particular problem context. Together, these four perspectives and their interrelationships provide a structure that enables the architect to comprehend the dynamics of Value based Software Architecture. These four perspectives, as illustrated in fig 1, address how value to stakeholders can be traced from Value understanding; correlations to value carriers; specification as Value propositions; and subsequent realization as software.

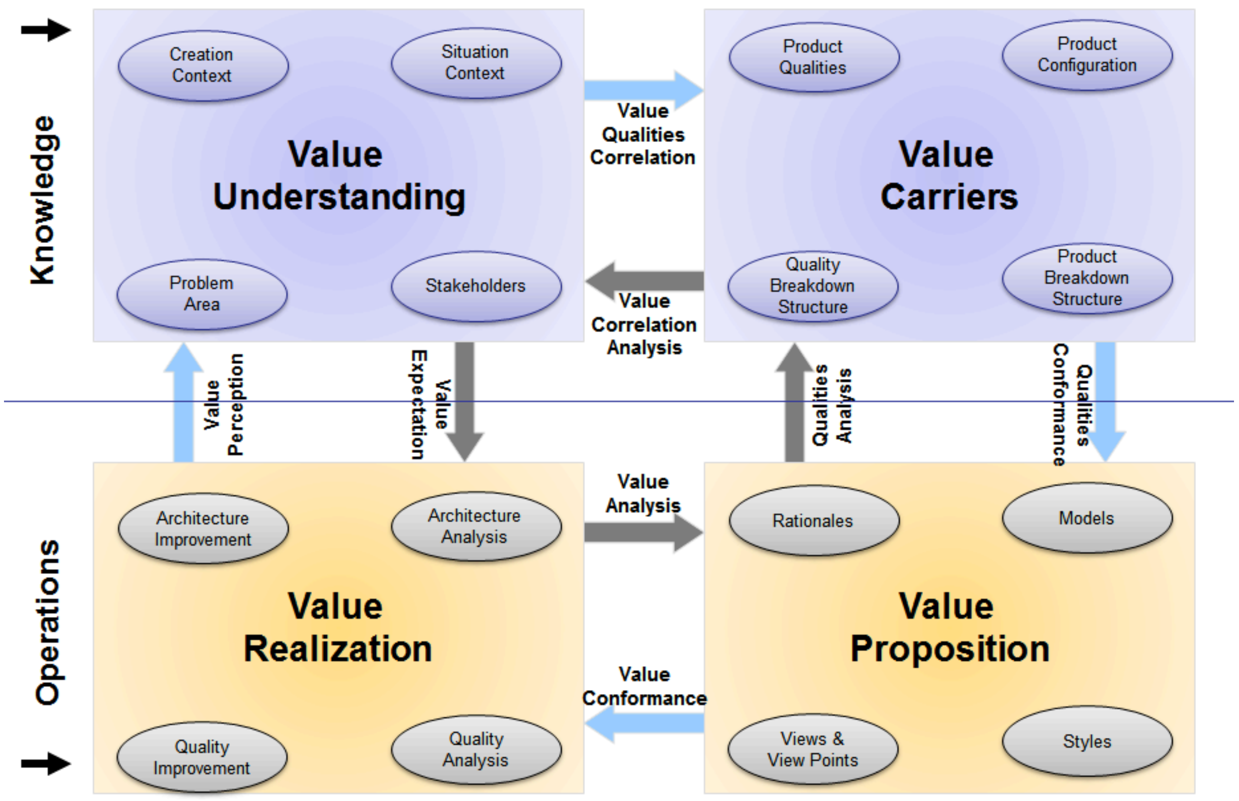

FIG 1: VALUE UNDERSTANDING, PROPOSITION AND REALIZATION BASED SOFTWARE ARCHITECTING (VURSA) APPROACH

The following sections provide some initial steps that can be taken for each of the four perspectives of the VURSA approach. While discussing each of these perspectives, a few guidelines are introduced and these guidelines briefly touch upon the nature of activities to be performed under each of the chosen perspectives.

i) UNDERSTANDING VALUE REQUIREMENTS

Value is a measure of worth (e.g., benefit divided by cost) of a specific product by a customer, and potentially other stakeholders and is a function of (1) the software's usefulness in satisfying a customer need, (2) the relative importance of the need being satisfied, (3) the availability of the software relative to when it is needed, and (4) the cost of ownership to the consumer [22]. All the things that contributes significantly to the stakeholder in terms of achieving their goals, plans, improvements, developments needed for growth, etc is considered as appealing to a stakeholder serves as the perceived/possible value.

For the software to succeed in creating value for any one stakeholder, it must create value for all whose contributions are critical to software's success. The failure to satisfy any one critical stakeholder creates risks of compensatory actions that lead to a value delivery failure, thus to the satisfaction of none. It is therefore necessary to understand the value that each stakeholder expects at different points in time. In the value understanding perspective, since value is the goal, the key question is; "How to discern value, not only after the software is developed but before the software is developed, and especially when value changes?" The solution is to understand stakeholders value in a given value creation context, explicate this context to define a set of possible solutions, abstract and identify the purpose of the system and explicate this to quality specifications. Table 1 list down the guidelines for the Value understanding perspective to consider.

TABLE 1: GUIDELINES FOR UNDERSTANDING VALUE REQUIREMENTS

- Identify Stakeholders
    - Identify all of those stakeholders who get affected by the software.
- Understand Stakeholders Value Creation Context
    - Understand value creation processes of stakeholders.
- Understand Problem Context
    - Understand how stakeholders perform their work processes to deliver outcomes.
- Define Problem Space
    - Work out the underlying purpose. Agree on problem abstractions.
- Define Solution Space
    - Analyse problem context and work out a collection of feasible solution concepts.
- Define Value Creation Context
    - Analyse customer's value creation processes and identify what value delivery is possible.

ii)

iii) IDENTIFYING VALUE CARRIERS

Once the Value that needs to be delivered is discovered, diagnosed and understood, it is necessary to understand the ways/means by which value delivery is possible. All the things that should exist in the software and contributes to the realization of value to stakeholders, in other words the carriers of value, are considered as the various qualities of the software. Quality characteristics of a software is the degree to which the software satisfies the stated and implied needs of its various stakeholders, and thereby enables value realization. These Quality Characteristics is a function of (1) usage of the Software, (2) impact of the software on its stakeholders, (3) measure of the degree of satisfaction of user needs, (4) and measure of the capabilities of the software that benefits its users. It measures the excellence of the software in a chosen dimension and is the basis for satisfying its stated purpose.

By constructively resolving the value requirements, through quality characteristics the software must possess; we can arrive at the design of the software without any recourse to technical artifices, wherein it is evident that the stakeholder value concerns have been addressed. As a result, we can perceive the suitability and adequacy of the functionality and appreciate the capacity of the resources needed to cater to the demands of all the non-functional quality characteristics. The key question is; "*How to establish the quality characteristics that the software must have in order to be acceptable to all of its stakeholders?*" The solution is to establish mappings between the benefits (value) desired by the various stakeholders and the proposed quality characteristics of the software, discerning priorities to resolve conflicts between stakeholder desires. Table 2 lists down the guidelines for the Value carriers' perspective to consider.

TABLE 2: GUIDELINES FOR IDENTIFYING VALUE CARRIERS

- Redefine Stakeholders Value Requirements
    - Redefine what value the software creates for its stakeholders.
- Define Solution Configuration
    - Establish what the software needs to deliver.
    - Define Software Quality characteristics and its correlation to Value
    - Identify essential and distinguishing attributes that deliver the intended value
- Develop Product breakdown Structure
    - Develop a form based on layered abstractions with each layer chosen based on knowledge domains involved.
- Develop Quality breakdown Structure
    - Prioritize, delineate and segregate Quality characteristics across different levels of understanding.
- Develop Delivery Processes
    - Develop stakeholders' responsibility satisfaction processes and associated management processes so

as to achieve the identified purpose.

iv) VALUE PROPOSITION

Value proposition defines a collection of value objectives that needs to be realized by the software. These objectives are achieved by careful organization of the software components. This organization structure may include both structural and behavioural aspects, properties, software qualities, software elements, and their interrelationships, etc. Value proposition provides a general capability to express the software as a hierarchy of relevant value delivery structures. It is used by various stakeholders who create, utilize and manage software to improve communication and co-operation, enabling them to work in an integrated, coherent fashion.

Creating the Value propositions involves stating the purpose of the software; stating the desired value adding qualities from an understanding of the user needs; stating the software components that comprise the design; identifying styles to be used; planning and preparing the design methodology to be adopted; stating the parts and interconnections, creating spatial structures that represent these parts; and representing these structures as multiple views. The key question is; "*How does the designer formulate their value proposition? How do they establish the link between design abstractions and Value proposition that is to be developed?*" The solution is to express the design abstractions as models and transform the models to multiple layers of understanding based on the knowledge domain and descriptions involved (eg. Infrastructure layer, physical layer, deployment layer, operational layer, etc), which are essentially different perspectives/views in understanding a software. Table 3 lists down the guidelines for the Value proposition perspective.

TABLE 3: GUIDELINES FOR DEVELOPING VALUE PROPOSITION

- Identify Design Styles
    - o Identify common patterns that characterize the system as a whole
- Define Design rationales
    - o State the reasons for the choices made
- Define design models
    - o Models can be used to structure, identify, analyze and synthesize design
- Create Design views and view points
    - o Frame specific solution concerns; establish conventions for realization
- Translate to Value Proposition by creating appropriate architecture description
    - o Utilize available approaches (Adopt standardized (agreed-upon) definitions and perspectives to describe the software architecture)
    - o Express identified solution configuration, delivery processes, qualities decomposition using Architecture Description languages.
- Qualities Conformance
    - o Trace identified qualities; expected value to Value Proposition.

v) VALUE REALIZATION

The purpose of the Value Realization perspective is to instantiate the software architecture for a problematic situation. For this it is necessary to transform the specified behaviour, interfaces and implementation constraints into realization actions that create the software according to the usage processes. The software is developed by processing the information appropriate to the selected processes and by employing appropriate technical specialties or disciplines. This process results in the satisfaction of specified structural/behavioural requirements through verification and stakeholder requirements through validation.

In order to realize value, it is necessary to address requirements of all stakeholders. The failure to satisfy any one critical stakeholder creates risks of compensatory actions that lead to a value delivery failure. In the Value Realization perspective, since successful software usage is the goal, the key question is; "*How to utilize people, conceptual and technology resources in such a way so as to conform to the Value proposition and provide an* *intervention in a problematic situation?*". Table 4 lists down the guidelines to follow for the Value realization perspective.

TABLE 4: GUIDELINES FOR VALUE REALIZATION

- Value Realization
    - Instantiate the architecture for a specific instance.
- Product Analysis and Improvement
    - Trace Architecture Description to its development.
- Qualities Analysis and Improvement
    - Analysis of delivered qualities against desired qualities and appropriate improvement
- Value Analysis and Improvement
    - Analysis of delivered value against desired value and appropriate improvement

6. CASE STUDY – DESIGN OF A TASK AUTOMATION SYSTEM

Maintaining integrity across information systems requires a disciplined usage of the systems, often requiring a role-playing user to login into multiple systems and carefully use them to assure integrity. The crossover between systems is manual, tedious and prone to error. The challenge is to ensure the automatic flow of information between different information systems across the organization without or with very minimal manual intervention. In this case study, we discuss about an integration approach that automates IT tasks and uses this automation to bridge information systems, leading to a possibility of completely avoiding manual intervention. In the subsequent sections, we discuss about the architecture of such a system using VURSA approach.

6.1 IDENTIFIED STAKEHOLDERS

Anyone who gets affected by the system or participates in the system in any way is a stakeholder. This association can occur anytime during the life time of the system. The probable stakeholders for the task automation system are: Individual, Manager, Project, Organization, Recipients and the Infrastructure Team.

6.2 UNDERSTANDING STAKEHOLDERS VALUE

Software users perform IT tasks to integrate different information systems. Over multiple iterations they would like to continuously reduce time, effort and cost involved in integrating these systems. Therefore "**reduce cost"** in performing the IT tasks in information systems is the minimal value that the software users would look forward to. Suppose, the IT tasks for integrating information systems are performed at a higher speed then the business latency would reduce. Similarly, the accuracy of the data transfer between information systems is 100% then there will be less effort impended in asserting data integrity. We can treat all this extraneous value requirements as "Enhancing Value to Customers". Table 5 lists down the Value understanding in this situation.

TABLE 5: VALUE UNDERSTANDING

Reduce cost	**Enhance Value**
• Reduce time and effort impended by agents to perform their tasks	• Improve efficiency of business
• Reduce complexity of work done by agents	• Reduce business latency
• Increase the productivity and throughput of agents	• Improve creativity of agents
• Reduce/Remove non-value adding tasks	• Experience certainty in delivering results
	• Do more with less

performed by agents • Reduce overheads in tasks performed by agents • Increase effectiveness of agents • Complete tasks on or within schedule (Reduce rework) • Increase operational effectiveness of computing resources • Eradicate bottlenecks and inefficiencies	• Assert quality of deliverables • Small, Light-weight but powerful • Address more than 80% of IT tasks • Eradicate non value adding activities

6.3 IDENTIFIED STAKEHOLDERS VALUE CREATION CONTEXT

Some of the value creation context adopted by the stakeholders in this problem situation could be a) On demand, on time delivery, b) Increase effectiveness, productivity and throughput, c) Within budget, no additional costs, d) Less waste in terms of bottlenecks and inefficiencies, e) Less business latency and, f) Experience Certainty

6.4 IDENTIFIED PROBLEM SITUATION

Every organization has a patchwork of information systems made up of new and old, compatible and not-so-compatible software. Information systems are part of the enterprise infrastructure that supports business. They provide the functionality needed by the Organization. Each system internally supports the transactions, and sometimes the processes to realize this functionality. The presumption is that there exists a smooth flow of information from one system to another system. However, it is not hard to see that there could be duplication of data, and other kinds of overlaps between these systems. It is also possible that data in one system is needed by transactions in other system. This situation is ripe for loss of integrity of these systems due to lack of consistency between these systems in relation to the organization. Maintaining integrity across systems requires a disciplined usage of the systems, often requiring a role-playing user to login into multiple systems and carefully use them to assure integrity. The crossover between systems is manual, tedious and prone to error. This is the problematic situation that we need to address.

6.5 PROBABLE SOLUTION SPACE

The conceptual solution for a task automation system could be to *"Create an IT robot that would emulate human activities in a computer, run automated IT tasks, perform these IT at a higher rate, high endurance, reliability, precision and speed, organize tasks for integration, repeatability and scalability and manipulate and interact with IT systems to facilitate automation"*. This could be achieved by creating a comprehensive collaborative environment for automating human tasks in which many tools work together to provide the desired task automation.

The various steps involved in this task automation are: a) Define people process steps straddling across IT systems for integration, b) Identify manual activities of human actors integrating information in IT systems, c) While enacting, identify activities that can be performed by machine and automate them, d) Gradually increase the level of automation till all the steps are automated and, e) When the entire process is completely automated, systems are bridged by automation.

6.6 PROBABLE SOLUTION VALUE CREATION CONTEXT

We introduce automation as a means to reduce/remove human efforts impended in maintaining information integrity. Towards this end, we would like to design the task automation system in which, human activities that straddle across information systems are identified and automated. When all activities are completely automated, information systems are bridged by automation. Such automation increases accuracy, reduces latency, increases efficiency, improves productivity and reduces hardware burden imposed by manual processes. In such a case the probable value creation context due to the automation of manual integration processes could be: a) Increased agility to Expand, b) Customer Satisfaction and Delight, c) Improved business value, d) Moving up the value chain, e) Lower costs, increased throughput and f) Higher yield and margins.

6.7 Probable Solution Characteristics

Based on the identified solution value creation context, we can arrive at a set of quality characteristics of the solution. These quality characteristics dictate the configuration of the solution as a whole. Table 6 illustrates these quality characteristics.

TABLE 6: SOLUTION CHARACTERISTICS

Must have	**Can Have**
• Perform Transactions • Assert Confidentiality and Accuracy • Ensure Reliability, precision while performing the transaction • Ensure Certainty and Compliance • Support Fault tolerance and Recovery • Reduce Latency	• Aesthetics, Simplicity, Changeability **Excitement/Delight** • Speed of Transacting • Scalability of Transactions • Productivity improvement • Cost reduction and standardization

6.8 Probable Solution Configuration

Based on the identified solution concept, solution value creation context and solution characteristics, we have arrived at a solution configuration and associated processes for the task automation system (referred as eScript). As shown in fig 2, we utilize a reference scripting environment (comprising a language configuration and an operational semantics configuration) to create the necessary infrastructure for the task automation system. The task that needs to be automated is expressed as a process program in the form of a machine executable script which when enacted will facilitate the necessary task automation. Fig 3 is an illustration of the associated processes for this specific solution configuration.

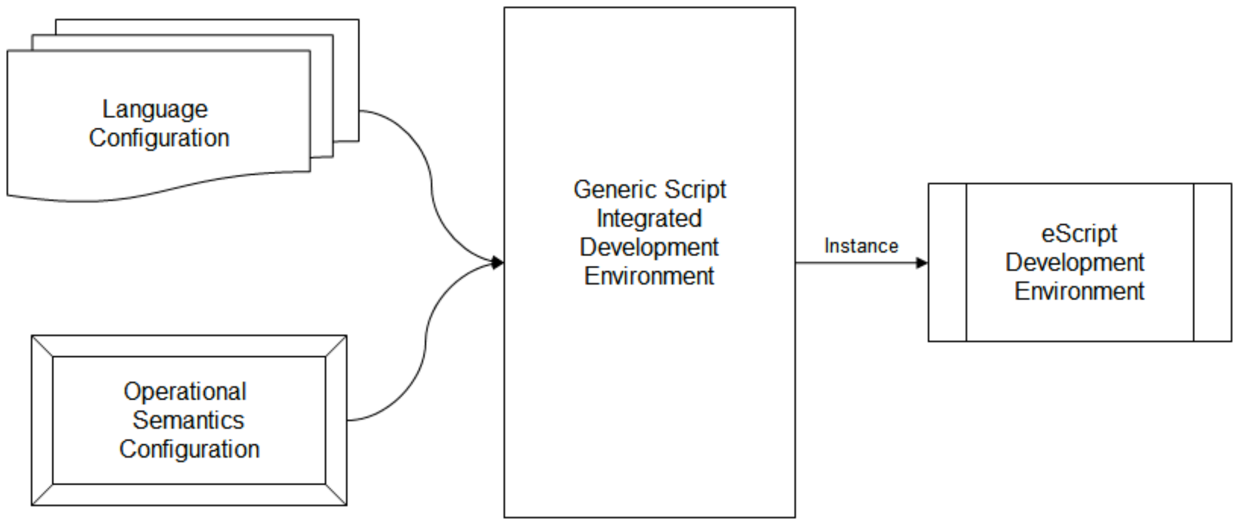
Language Configuration
Operational Semantics Configuration
Generic Script Integrated Development Environment
Instance
eScript Development Environment

FIG 2: SOLUTION CONFIGURATION

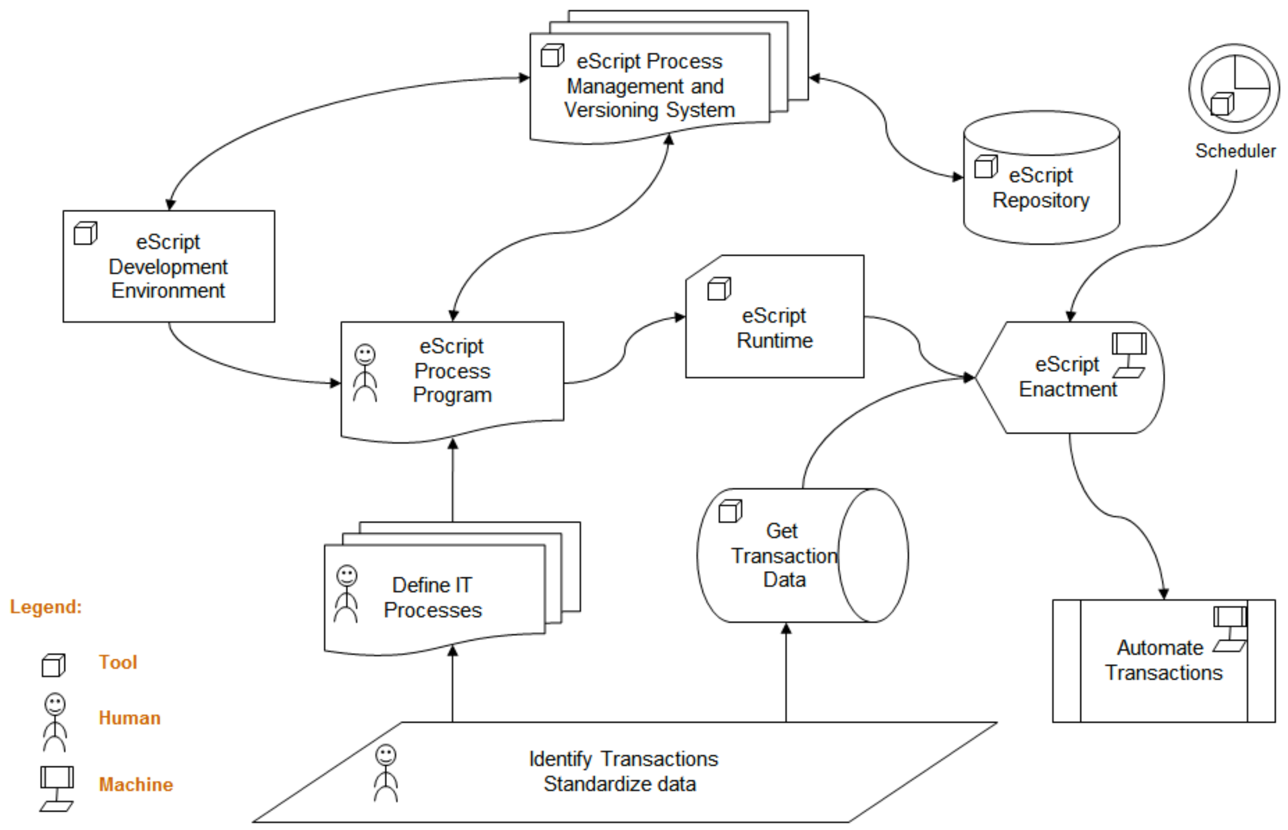

FIG 3: PROCESSES IN THE SOLUTION CONFIGURATION

6.9 Probable Solution Qualities Breakdown Structure

For the identified solution qualities, we adapt a structured systematic approach to breakdown the qualities into different layers wherein each layer addresses a specific concern. The different layers are selected based on the different knowledge domains involved in the creation of the software solution. For illustration purposes, we consider 7 layers as illustrated in Table 7.

Table 7: Solution quality Characteristics Break Down Structure

Levels of Understanding	Functionality	Reliability	Usability	Efficiency	Maintainability
Problem Space	Perform Process Descriptions	Cope with statistical variance	Increase value for impended effort	Desired value for impended effort	Cope with change
Architecture	Conceptual integrity	Certainty and Accountability of outcome	Effectiveness and efficiency of usage	Reduce undesirable features	Modularity, Modifiability, Changeability
Engineering	Measure of Capacity of the solution	Measure of consistency and repeatability	Measure of Elegance and clarity	Measure of Speed and space	Measure of adaptability to changing environment
Construction	Structure for validation	Perform under stated conditions and Constraints	Fitness for purpose and time to use	Reduction in complexity	Localization of change
Deployment	Allow existence of multiple roles and responsibilities	Safety in deployed environment	Utilization of deployed environment	Utilization of available resources	Allow existence of multiple versions
Initialization	Define operational parameters	Multiple conditions and Constraints	Ease and simplicity of use	Maximal and minimal performance characteristics	Multiple configurations
Operational Environment	Ability to perform stated service	Ability to survive, sustain and handle failures	Accomplishment of Process Descriptions	Increase Productivity	Variability in usage

6.10 Probable Solution Configuration Breakdown Structure

For the identified solution configuration, we adapt a structured systematic approach to breakdown the system configuration into different components wherein each component addresses a specific concern. The different components are selected based on the different knowledge domains involved in the creation of the software solution. For illustration purposes, we consider the components shown in figure 4.

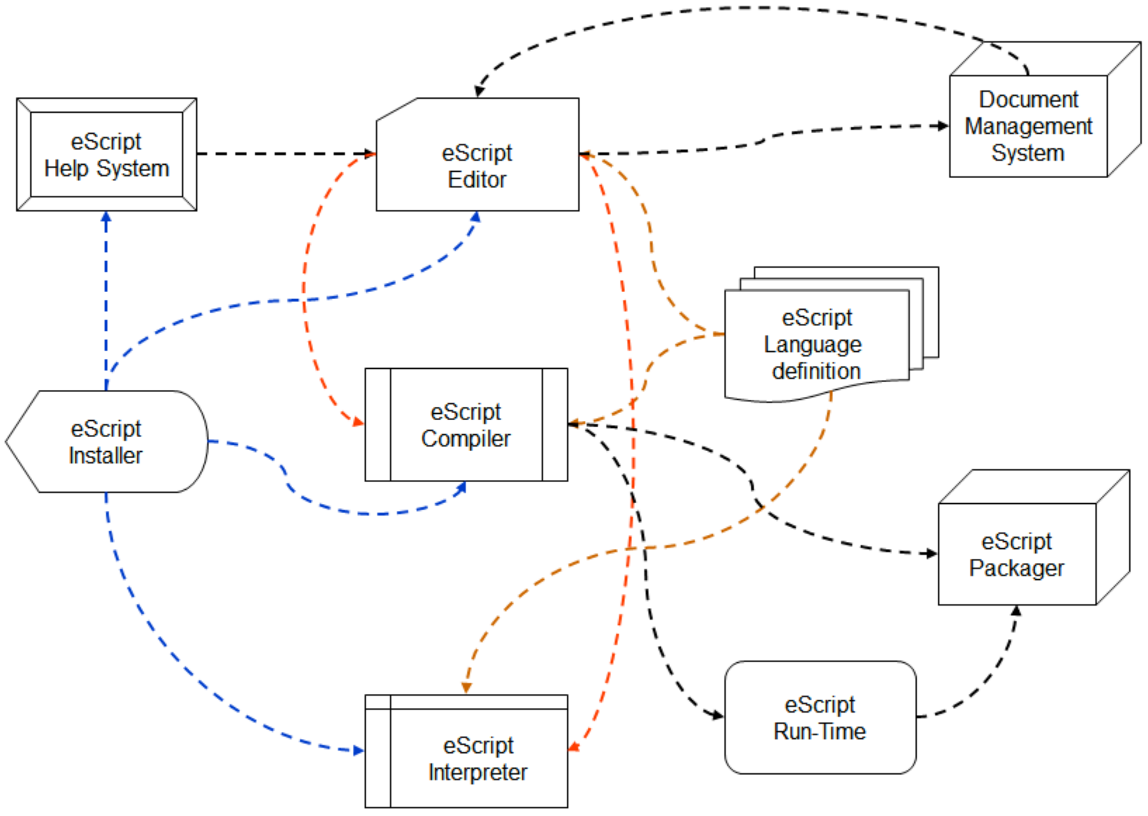

FIG 4: SOLUTION CONFIGURATION BREAKDOWN STRUCTURE

6.11 REPRESENTING THE ARCHITECTURE OF THE SOLUTION USING SYSML

Architecture models serve as specifications of a system. They allow the different configurations of configuration items to be captured in an analogous way, such that certain properties of the system can be understood, organized, managed and emulated. Since only architectural properties are captured, architecture models reduce the complexity of system specification. Architecture models are approximations and they capture different properties of the system. It is a scaled down version and is built with all essential details. Fig 5 illustrates such an architecture model for our task automation system.

The architecture of the task automation system is designed to be extensible. Here, syntactic extensions are facilitated by plug-in interface and semantic extensions are facilitated by plug and play interface. This architecture pattern is recursive and is applicable for all sub-systems. In our model, symbol is used to express plug-in interface and symbol is used to express plug and play interface. Both interfaces have well-defined schema definitions and there exist protocols that control the instantiation of these interfaces. At the core of this platform is a central controller. This controller is based on Von-Neumann architecture and its primordial purpose is to interpret stored specifications and create job definitions for sub-controllers.

The central controller utilizes pre-defined protocols and schema to interact with the sub-controllers. The sub-controllers are also based on Von-Neumann architecture and their purpose is to interpret the job definition available in their job queues and perform appropriate actions that commensurate with their domain.

For our discussion purposes, we restrict the specifications for the central controller to be of 3 kinds; the first being the specification for expressing IT tasks and stringing these tasks together as an IT process; the second is the interpretation of the IT process and requesting the sub-controllers to perform appropriate actions and the last is the analysis of the IT process to draw meaningful inferences. The various sub-controllers that exist in our model are catered to support these 3 kinds of specifications. A typical specification for creating IT process would be:

a) Identify the type of Task (using the Task Controller) and tag a label to it.

b) Associate a work-item to the identified task (using the work-item controller) and tag a label to it.

c) Bind work-item to action that needs to be performed in the IT system (using IT system controller).

d) Create a collection of tasks to be performed by repeating the above steps.

e) Define the sequence of tasks and the flow of Work-items (using the Sequence Controller).

f) The outcome of performing this job is the creation of an IT process specification

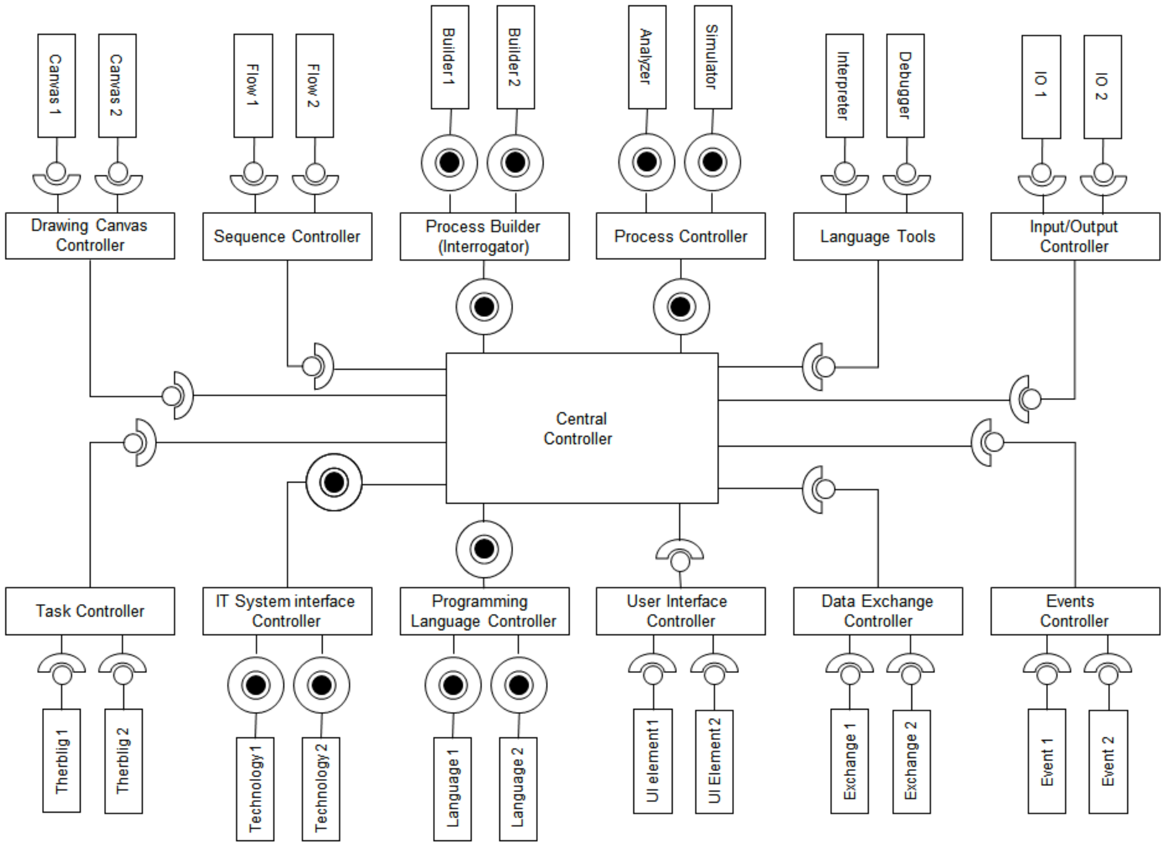

FIGURE 5: SYSML REPRESENTATION OF TASK AUTOMATION SYSTEM

7. Summary

The basis of the VURSA approach is the "values viewpoint" for creating and describing software. It includes identifying the need for architects to understand value creation processes, creating a separation of value and quality concerns across levels, creating appropriate form/structures for realizing value and establishing traceability between value understanding, value proposition and value realization. The core ideas behind VURSA, as illustrated in Table 8, are the four different perspectives and the corresponding activities to be performed against these different perspectives.

Table 8: Summary of VURSA

Perspective	Guidelines
Value Understanding	• Identify Stakeholders o Identify all of those stakeholders who get affected by the software. • Understand Stakeholders Value Creation Context o Understand value creation processes of stakeholders. • Understand Problem Context o Understand how stakeholders perform their work processes to deliver outcomes. • Define Problem Space o Work out the underlying purpose. Agree on problem abstractions. • Define Solution Space o Analyse problem context and work out a collection of feasible solution concepts. • Define Value Creation Context • Analyse customer's value creation processes and identify what value delivery is possible.
Value Carriers	• Redefine Stakeholders Value Requirements o Redefine what value the software creates for its stakeholders. • Define Solution Configuration o Establish what the software needs to deliver. o Define Software Quality characteristics and its correlation to Value o Identify essential and distinguishing attributes that deliver the intended value • Develop Product breakdown Structure o Develop a form based on layered abstractions with each layer chosen based on knowledge domains involved. • Develop Quality breakdown Structure o Prioritize, delineate and segregate Quality characteristics across different levels of understanding. • Develop Delivery Processes o Develop stakeholders' responsibility satisfaction processes and associated management processes so as to achieve the identified purpose.
Value Proposition	• Identify Design Styles o Identify common patterns that characterize the system as a whole • Define Design rationales o State the reasons for the choices made • Define design models o Models can be used to structure, identify, analyze and synthesize design • Create Design views and view points o Frame specific solution concerns; establish conventions for realization

	• Translate to Value Proposition by creating appropriate architecture description o Utilize available approaches (Adopt standardized (agreed-upon) definitions and perspectives to describe the software architecture) o Express identified solution configuration, delivery processes, qualities decomposition using Architecture Description languages. • Qualities Conformance o Trace identified qualities; expected value to Value Proposition.
Value Realization	• Value Realization o Instantiate the architecture for a specific instance. • Product Analysis and Improvement o Trace Architecture Description to its development. • Qualities Analysis and Improvement o Analysis of delivered qualities against desired qualities and appropriate improvement • Value Analysis and Improvement • Analysis of delivered value against desired value and appropriate improvement

8. ACKNOWLEDGEMENTS

Our thanks to Prof. Kesav Nori, Dr. VCS Prasad (Ex-TCS Advisor), Sekhar Vadari, and Dr. Swaminathan of Tata Consultancy Services for seeding the thoughts on Value and Qualities based Design.

9. REFERENCES

1. Maier M, E. Rechtin, 2009, The Art of Architecting, 3rd Edition, CRC Press
2. Alain Faisandier, 2012, Systems Architecture and Design, Volume 3, Singerygy' Com
3. ISO/IEC, 42010: 2011, Systems and Software Engineering - Architecture Description
4. Jack Ring, 1998, Value seeking approach for Systems Engineering, IEEE-SMC, Conference Proceedings
5. Noriaki Kano, 1996, Guide to TQM in Service Industries, Asian Productivity Organization
6. SE Handbook Working Group, INCOSE, 2011, Systems Engineering Handbook V 3.2.2
7. Barry Boehm, Value based Software Engineering, ACM Sigsoft, Software Engineering Notes Vol 28, no 2
8. K Anand, Kesav V Nori, Nitin K Reddy, Raman A, 2008, User Value driven Software Development, Information Systems 2008, Barcelona, Spain
9. K Anand, Doji Samson, Nikhil Zope, 2012, Value driven approach for Services Design, 56th Conference of International Society of Systems Science, San Jose, USA
10. Doji Samson Lokku, 2011, Value Distilled: A framework based approach to establish the trace of Technology Value in the context of Engineering Management, Eurocon 2011
11. David Garlan, Mary Shaw, 1994, An introduction to soft-ware architecture, Advances in Software Engineering and Knowledge Engineering, Volume I, World Scientific Publishing Company

12. Dewayne E Perry, Alexander Wolf, 1992, Foundations for the study of software architecture, ACM SigSoft, Software Engineering Notes Vol 17, No 4
13. IEEE Computer Society, 2004, Guide to Software Engineering Body of Knowledge, SWEBOK
14. ISO/IEC, 15288:2008, Systems Engineering - System Life Cycle Processes, ISO and IEC, 2008
15. SE Handbook Working Group, INCOSE, 2011, Systems Engineering Handbook V 3.2.2
16. Womack, James P., and Daniel T, Jones, 1996,Lean Thinking, New York: Simon & Schuster
17. John Reekie, Rohan McAdam, A Software Architecture Primer, Angophora Press
18. Zachman, John A., "A Framework for Information Systems Architecture," IBM Systems Journal, vol. 26, no. 3, 1987; IBM Publication G321-5298
19. Kano, N., 1984, Attractive quality and must-be quality, The Journal of the Japanese Society for Quality Control, April, pp. 39-48.
20. Kesav V Nori and N. Swaminathan, A Framework for software product Engineering, Asia-Pacific Software Engineering Conference, Bangalore, Dec6-8, 2006.
21. Bill Hallier, Space is the Machine, A configurational theory on Architecture, eBook
22. Philippe Kruchten, 1995, Architectural - Blueprints, The 4 + 1 View Model of Software Architecture, IEEE Software 12 (6), pp 42-50
23. Systems Engineering and Cybernetics Centre, 1999, Multi-Modeling Approach to Enterprise Analysis and Modeling, Quality Management System Guidelines
24. Dale D. Meredith, Kam W. Wong, Ronald W Woodhead, Robert H. Wortman, Design and Planning of Engineering Systems, Prentice Hall
25. Bill Hillier and Adrian Leaman, 1974, How is design Possible, Journal of Architectural and Planning Research,
26. IEEE Computer Society, 2012, Guide to Software Engineering Body of Knowledge, Software Engineering Economics
27. IEEE Computer Society, 2012, Guide to Software Engineering Body of Knowledge, Software Design

28. Barry Boehm and Kevin Sullivan, Software Economics: A Roadmap, The Future of Software Engineering, http://www.cs.ucl.ac.uk/staff/A.Finkelstein/fose/finalboehm.pdf
29. Hitchins, D. 2008. "Emergence, Hierarchy, Complexity, Architecture: How do they all fit together? A guide for seekers after enlightenment." Self-published white paper. Accessed 4 September 2012. Available at: http://www.hitchins.net/EmergenceEtc.pdf.
30. Steve McConnell, 2004, Code Complete, Microsoft Press,
31. OMG, OMG Systems Modeling Language Version 1.3, http://www.omg.org/spec/SysML/20120401
32. Mary Shaw, 1994, Patterns for Software Architecture, First Annual Conference on the Pattern Languages of Programming, August 1994, vol 1, Coplien and Schmidt (eds), Addison-Wesley,1995, pp. 453-462.
33. ISO/IEC, ISO - 15939, Systems and Software Engineering - Measurement Process, ISO and IEC
34. ISO/IEC, ISO – 25010, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models, ISO and IEC
35. ISO/IEC, ISO – 42030, – Architecture Evaluation, ISO and IEC (Working Draft)
36. Abowd, Gregory; Bass, Len; Clements, Paul; Kazman, Rick; Northrop, Linda; & Zaremski, Amy. Recommended Best Industrial Practice for Software Architecture Evaluation (CMU/SEI-96-TR-025). Software Engineering Institute, Carnegie Mellon University, 1997. http://www.sei.cmu.edu/library/abstracts/reports/96tr025.cfm
37. Nico Lassing, 2002, Architecture Level Modifiability Analysis, SIKS Dissertation Series No. 2002-1, Dutch Graduate School for Information and Knowledge Systems
38. Rick Kazman, Jai Asundi, Mark Klein, 2002, Making Architecture Design Decisions: An Economic Approach, Technical Report, CMU/SEI-2002-TR-035, ESC-TR-2002-035
39. Len Bass, Paul Clements, Rick Kazman, 2003, Software Architecture in Practice, 2nd Edition, Addison Wesley publications
40. Robert Allen, David Garlan, 1996, The Wright Architectural Specification Language, School of Computer Science, Carnegie Mellon University.

41. Robert J Allen, A forma approach to Software Architecture, Doctoral Thesis, School of Computer Science, Carnegie Mellon University, 1997.

42. Paola Inverardi, Alexander Wolf, Formal Specification and Analysis of Software Architectures using the Chemical Abstract Machine Model, IEEE Transactions on Software Engineering, Vol 21, No 4, April 1995

43. Garlan, David; Monroe, Robert T.; and Wile, David, ACME: An Architecture Description Interchange Language (1997). Tepper School of Business. Paper 322. http://repository.cmu.edu/tepper/322

44. David C Luckham, James Vera, Sigurd Meldal, 1995, Three Concepts of System Architecture.

45. David C Luckham, James Vera, An Event-Based Architecture Definition Language (RapidE)

46. David Garlan, Formal modelling and analysis of Software Architecture: Components, Connectors and Events, School of Computer Science, Carnegie Mellon University

47. Nenad Medvidovic, Richard Taylor, A framework for classifying and comparing Architecture description languages, University of California at Irvine, Technical Report, January 1997.

48. Clemens Szyperski, Dominik Gruntz, Stephan Murer, Component Software: Beyond Object Oriented Programming, 2nd Edition, Addison Wesley Publications

49. Ian Sommerville, 2000, Algebraic Specification

50. Roy Thomas Fielding, Architectural Styles and the Design of Network based Software Architectures, Doctorate Thesis, University of California, Irvine, 2000.