

**Copyright**

**By**

**Paul Leonardo Taylor**

**2009**

**Implementing Software Architecture Practices  
in a new environment**

By

**Paul Leonardo Taylor**

**Report**

Presented to the Faculty of the Graduate School

of the University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the degree of

**Master of Science in Engineering**

The University of Texas at Austin

August, 2009

The Report committee for Paul Leonardo Taylor  
Certifies that this is the approved version of the following report

**Implementing Software Architecture Practices  
in a new environment**

APPROVED BY

SUPERVISING COMMITTEE:

Supervisor: \_\_\_\_\_

Suzanne Barber

\_\_\_\_\_

Thomas Graser

**Implementing Software Architecture Practices  
in a new environment**

by

Paul Leonardo Taylor, MSE

The University of Texas at Austin, 2009

SUPERVISOR: Suzanne Barber

During a discussion with the head of the software infrastructure team about the need for software architecture practices at Temple-Inland Company, the manager responded by noting since the company is not a software development company “there are no real benefits to implementing software development practices in the company”. This is an approach taken by many companies whose software development activity is primarily undertaken to support business activities such as the case with manufacturing or financial companies.

This paper examines the process of implementing software architectural practices into an organization. The information contained here should be useful to small startup software companies who might assume that it is too costly to incorporate software architectural practices into their current development process. This paper should also benefit large organizations who primarily view software as solutions for short term immediate support and not in terms longer term strategic goals. Software development teams with projects that suffer from cost overruns, scheduling problems and user dissatisfaction should also find this information useful.

## Table of Contents

1. Introduction .....	1
2. Purpose of Software Architecture .....	2
3. Obstacles to Implementing Software Architecture .....	3
3.1. Misconceptions .....	3
3.2. Common Architectural problems.....	6
4. Capability Maturity Model Integration .....	9
4.1. Level 1: Initial .....	10
4.2. Level 2: Managed .....	10
4.3. Level 3: Defined.....	11
4.4. Level 4: Quantitatively managed .....	11
4.5. Level 5: Optimizing the process .....	12
5. Software Architecture Implementation Process .....	13
5.1. Determine Organization Readiness .....	14
5.2. Create Software Architecture Implementation Plan .....	16
5.3. Implementation Phase.....	18
5.3.1. Implementing Software Engineering in a Small Workgroup.....	20
5.3.2. Software Architecture Roles .....	21
5.3.3. Software Architecture Tools .....	29
5.4. Software Architectural Program Assessment .....	33
5.5. Evaluating Software Architecture .....	33
6. Conclusion.....	36
References .....	37
VITA.....	39

## List of Tables

Table 1: Project Document Repository .....	19
Table 2: Functional requirements Template.....	22
Table 3: Site Requirements Template.....	22
Table 4: Non-functional Requirement Template .....	23
Table 5 : Non-Functional Requirement.....	23

## List of Figures

Figure 1: Five Levels of Capability Maturity Model Integration.....	9
Figure 2: Architect roles in business-software development .....	25
Figure 3: SEI Architecture Expert (ArchE) Tool.....	31

## **1. Introduction**

Software is an essential and critical component of most of today's businesses. Software architecture is becoming more crucial to the success of software development in many levels of a company. The benefits of software architecture are evident to companies who develop and sell software. Companies that only use software to support their main line of business may not think about the need for software architecture practices in the same way, however it is in their best interest to develop their software systems in the most efficient and cost effective way.

The focus of this report is on the process of introducing software architecture practices to businesses that do not currently use software architecture practices. This report will help companies determine their readiness to implement software architecture practices. This report outlines some of the signs to look for such as cost overruns, inefficient use of resources, late project delivery and products that are late to market. Software architecture can help improve market share by allowing for faster time to market of software products (1). This is necessary in competitive markets and in areas where the product is rapidly changing such as cell phone markets where products are virtually obsolete as soon as they hit the market. Software architecture also produces software that is easier to support, maintain and modify. It is easier to develop a software product line if the process involves software architecture practices (2).

This paper focuses on the preconditions for implementing software architecture and how to prepare a company for the necessary changes in process. The paper then examines the necessary roles of software engineering such as a project champion, requirements engineer, developer and architects. It then discusses how following these steps can make a software engineering department more efficient by

taking advantage of reuse and knowledge transfer. It also discusses where software architecture fits into the overall goals of the organization.

The rest of this paper is organized as follows. In section 2, the paper provides a rationale for software architecture, in which the paper looks at the ways in which an organization or team will benefit from software architecture. Section 3 presents some common misconceptions and obstacles to software engineering and provides information on how these problems can be mitigated. Section 4 introduces the Capability Maturity Model Integration (CMMI) which shows the five levels of maturity for the software development process. Understanding the CMMI process is important for any team or business that develops software systems because it allows them to identify the state of their development process and how it can be improved. Section 5 describes the process of implementing a software architecture program into an organization or team. This is broken down into the following four categories; determining an organizations readiness for a software architecture program; creating an implementation plan; implementing the new program and evaluating the program's effectiveness. The paper offers a conclusion in section 6.

## **2. Purpose of Software Architecture**

Companies have to become lean as they focus on improving areas of their business that are inefficient and waste resources. Lots of time, effort and money are lost due to inefficient practices. Companies are taking notice and are trying to find ways improve their software process. Companies can make to step towards improving their software development process by applying proven software architecture practices.

All software development teams currently apply some aspects of software architecture as defined by Perry (3). The goal of software architecture is to improve the development process by offering an appropriate view of the system throughout the software development life cycle to each of the stakeholders, reduce maintenance cost, improve documentation and allow for reusability of the system or the techniques used to develop the system. In order to successfully introduce software architecture practices to a new team or organization there are a number of considerations that needs to be made, such as the size of the team or organizations and the organization structures. A good architectural program has the capability to be applied to projects of different sizes throughout the organization. The architectural program can be thought of as a set of tools that can be customized to fit each project. A good software architecture plan will scale well to larger projects. The cost of implementing a new software architecture plan can be great however the cost can be offset by reuse on future projects.

### **3. Obstacles to Implementing Software Architecture**

Software architecture implementation can be extremely challenging (4). There are three main reasons for this complexity. The first reason is that existing corporate standards and culture needs be incorporated into the software architecture process. The second major challenge facing software architecture implementation is getting the business stakeholders buy-in. This needs to be done using concrete facts about the benefits of software architecture to the business. All discussions needs to focus on the specific befits to the company such as cost savings and improvements in Return on Investments (ROI). As companies face the pressures to cut cost and reduce spending, they are being forced to do more with fewer resources. A good software architecture process is a great way to cut cost and operate more efficiently with fewer resources in order to keep the business operating effectively. The third challenge posed by implementing a software architecture program is in developing an effective implementation roadmap.

#### **3.1.Misconceptions about Software Architecture**

There are some common misconceptions about software architecture (5) that can impede its widespread usage. These misconceptions are commonly cited by developers, managers and executives as justification for not using software architecture practices. Part of the confusion could be attributed to the multiple different definitions cited by experts (3) (6) (7). Below we explore some of the misconceptions and how these misconceptions may be addressed.

- Software architecture is often thought of as just component based development.

While it is true that components are the basic building blocks of a software system, it is not enough to just say that software architecture is just concerned with components. Component development suggests a bottom up approach for software development; however comprehensive software architecture must incorporate managing the development process and providing views that are appropriate for the targeted audience.

- Software Architecture is the same as design

It is true that software architecture involves many aspects of design, however, architecture is much more than design. Design is focused on what needs to be done or what needs to be implemented, software architecture however involves not just the design decisions but it is also concerned with how the decisions are made. There are three views to software architecture; conceptual, implementation and execution. Software architecture focuses on other aspects of the system development process such as cost, maintenance, performance and reliability

- Software Architecture is just the infrastructure specification

Thinking about the architecture only in terms of the infrastructure is just one aspect of the architecture. The infrastructure is a very important part of the software architecture. If the focus is set too narrowly on the infrastructure then the end product could fit the infrastructure however it fails to solve the problem.

- Software architect is the job of one individual

A good architecture team can be more efficient than a single architect. It is best to have a single leader that sets the direction of the team (8). Small companies or teams may not be able to afford a team of architects. If the architecture plan is well documented and accessible then a small team with a single architect may be able to perform as efficiently as larger companies with dedicated software architecture teams.

- Software Architecture cannot be validated

There are many methods and tools available to measure the improvements and successes of developing a software architecture program. This includes improved and more efficient use of resources. This can be especially evident when the architecture is reused. Additional software architecture metrics include risk management, usefulness of the documentation and issues tracking.

### **3.2. Common Architectural problems**

- Scope Creep

Teams that are tasked with developing new system may find it instinctive to please their customers at all cost. Often this includes trying to accommodate any and all feature request proposed by the customer throughout the life of the project. Managers need to ensure that the project has a clear vision in order to manage the scope. The project deliverables should be approved by the key stakeholders (9). Some companies use a change management process to communicate the financial and scheduling cost of incorporating the change into the project.

- Not enough users input

Requirements engineer may sometimes interview only a limited number of end users in the requirements gathering process. It may also be tempting to focus more on the requirements specified by the person paying for the system. The requirements should go through rigorous requirements analysis process where requirements are determined based on how well they satisfy the user's needs and the company's priorities.

- Quality Attributes

Another common mistake is to focus only on the functional requirements of the system while ignoring the non-functional requirements such as security, performance, usability and maintainability. Many

projects fail because their quality attributes does not meet customer satisfaction (10) (11). Software architecture requires that both the functional and non-functional requirements of the system be addressed. It is important to consider quality attributes early in the process. Some quality attributes such as performance and scalability can be used to track the progress of the project.

- Architectural Diagrams

Software architecture is not just a network of box and line diagrams describing the system. While architectural diagrams are important tools for describing the system they are generally very technical in nature and requires explanations of the symbols used to specify the architecture. This level of decomposition may be suited for a technical stakeholder such as a developer or tester; however there are other consumers of the software architecture such as management and the user community which needs an appropriate view of the software architecture.

- Ignoring Requirements

Teams without a mature software architecture process will often spend a significant amount of time gathering and refining requirements. However when they get to the development phase they tend to ignore the requirements and develop the system the way they think it should be as opposed to how it has been specified by the requirements (12). This often leads to a system that does not satisfy the customers' needs. Development teams will sometimes focus on the technology rather than the requirement and then try to designing the system around the technology instead of choosing technology solutions to solve the problem efficiently.

- Non Specific Infrastructure Specifications

The solution space must be explicitly specified in the software architecture. It can be easy to generalize the specification of platform or Commercial off-the-shelf (COTS) solutions, such as specifying that Oracle database is required for the solution instead of specifying what version or build of the software is required. Hardware and software platform requirements templates are good tools for gathering hardware and software specifications.

- Back out Plan

Many software projects are implemented to replace existing system. It is important to create a comprehensive rollback plan as part of the development process. Sometime the rollback plan could suffer if there are problems with the project schedule. The blackout plan should be accounted for in the project schedule.

## 4. Capability Maturity Model Integration

It is helpful for managers to think about the state of their own software development process in terms of the Capability Maturity Model Integration (CMMI). This section provides an overview of the CMMI process. CMMI was published in 2001 by the Software Engineering Institute at Carnegie Mellon University (13) as an update to the earlier Capability Maturity Model (CMM) process (14). The CMMI process integrates product and process development with people and other process improvement initiatives across a project, a division, or an entire organization. There are five levels to the software development process as specified by CMMI; initial process, managed process, defined process, quantitatively managed process and optimized process. Each level requires that the requirements for the previous level be satisfied.

### Capability Maturity Model – Integrated

Level	Focus	Process Areas	Result
<b>5 Optimizing</b>	<b>Continuous process improvement</b>	Organizational Innovation & Deployment Causal Analysis and Resolution	<b>Productivity &amp; Quality</b>
<b>4 Quantitatively Managed</b>	<b>Quantitative management</b>	Organizational Process Performance Quantitative Project Management	
<b>3 Defined</b>	<b>Process standardization</b>	Requirements Development Technical Solution Product Integration Verification Validation Organizational Process Focus Organizational Process Definition Organizational Training Integrated Project Management Risk Management Decision Analysis and Resolution	
<b>2 Managed</b>	<b>Basic project management</b>	Requirements Management Project Planning Project Monitoring & Control Supplier Agreement Management Measurement and Analysis Process & Product Quality Assurance Configuration Management	
<b>1 Initial</b>	<b>Competent people and heroics</b>		

Figure 1: Five Levels of Capability Maturity Model Integration (15)

#### **4.1. Level 1: Initial**

Level 1 is generally referred to as chaotic (16) which implies that there are no process controls employed. The development process tends to be poorly controlled. It also tends to be reactive to issues instead of anticipating potential issues. Organizations and teams that are at this level do not create risk management plans or have any formal process of governance for their projects. Because of the lack of formality, teams are not able to take advantage of past success or apply lessons learned to future projects. They also become extremely dependent on specific individuals as all the knowledge and experience tends to be possessed by specific individuals.

#### **4.2. Level 2: Managed**

Level 2 processes are characterized on a project by project basis. These organization or teams generally have some project management process that they follow. Some of the process management includes requirements management, risk management and project governance. The standards and practices may differ for each application of the process as it pertains to managing risk, testing and quality analysis. The process may also be different between projects in areas such as planning, cost analysis, project controls and contingency plans.

### **4.3. Level 3: Defined**

CMMI Level 3 specifies that in addition to satisfying the requirements of Level 2 the organization or team must take a proactive approach to process management. The processes are well defined and documented. These include standards, procedures, tools, and methods applied to the development process. This helps to establish consistency among projects across the company. These standards are developed by the organization and are refined and improved over time. The process can be tailored to fit individual projects while keeping in line with the standards and guidelines set out by the company. Because the standards and guidelines are based on past success it also means that managers will be able to more accurately budget time and resources for a project by providing better estimates. The company must also be proactive with their verification, compliance and governance processes.

### **4.4. Level 4: Quantitatively managed**

CMMI Level 4 requires that the level 3 requirements be satisfied and that the processes be measured and controlled. The process can be measured and controlled using statistical and other quantitative techniques such as Key Performance Metrics (KPM). The information gathered from these techniques are then incorporated into the process and used to further improve the process.

#### **4.5. Level 5: Optimizing the process**

Level 5 requires continual process improvement based on a quantitative analysis of issues that are common across all projects. It requires that the process be continually revised to address both incremental changes that could result from procedural or governance changes and major changes such as technological innovations or legal changes. The effects of the changes to the process are measured to further improve the process.

## **5. Software Architecture Implementation Process**

Sometime it may be necessary for a company to go through rapid software changes. This could be a result of a new acquisition that requires consolidation of software systems; required technology changes to keep up with new or changing markets brought on by technological advancements or it may be imposed on the company by some governing agencies to meet some hard deadline such as the 2009 Digital television (DTV) Act passed by congress and required of television broadcasters (17). It is often the case that, software becomes a major bottleneck whenever these changes require new or updated software systems.

Business software exists to support the business and its activities, or to help change the way business is performed. If it does not explicitly support or help change the business, it does not matter how technically brilliant the software is, its value lies in its capability to increase the productivity and efficiency of the business. New technologies can drastically change the way in which business is performed. This can have the effect of reducing the amount of resources needed or improving potential customer base. Such was the case with web 2.0 and web services which allowed retailers to reach a wider customer base through ecommerce.

The goals of the software must be aligned with the goals of the business. Software that has differing goals from the business put undue restriction on management in performing their functions. An example of this is where the business is expanding into newer markets and requires more infrastructure support to handle the additional load; however the software solutions were selected to reduce storage cost with no contingencies for scaling. Having the software systems aligned with the business needs reduces risk and makes it easier for the software systems to change with the business which is a critical component of modern businesses.

### **5.1. Determine Organization Readiness**

Companies will need to analyze data from past projects in order to determine if they could benefit from improvements to their software development process. These include data such as the length of the projects, the required resources and how well the projects stick to their budgets. The expertise of available personnel should be taken into account when considering the cost of software process improvement. Companies can choose to train existing employees as software architect or hire temporary contractors if the talent does not exist in house.

An internal assessment should be performed to assess the organization or team is readiness for a software architectural program. The thought of applying software architecture practices to their development process could cause some teams to resist the change as they view it as additional or sometimes unnecessary work. Some teams may welcome the idea of change especially if they have worked on project that were plagued with schedule delays, cost overruns or customer dissatisfaction with the software products. Managers should identify areas of their processes that could benefit from software architecture so that they can be tracked for improvements. Additional funding may be needed

base on the extent of the software process improvement. Below is a list of questions that could be helpful in assessing the need of a software architectural program

1. What is the number of ongoing projects?
2. What is the project backlog?
3. How often do the projects go over budget?
4. How often do the projects have cost overruns?
5. Who are the customers?
6. What are the common customer complaints from past or current projects?
7. How are the software projects managed for quality?
8. How do the customers determine software quality?
9. Are the customers involved in the project development process?
10. Are the software estimates correct?
11. How is project success measured?
12. What are the obstacles to improving the software productivity and quality?

## **5.2. Create Software Architecture Implementation Plan**

The next step is to develop a software architecture implementation plan. This plan should include the goals for the outcome of the implementation as well as a list of required documentation such as communication guidelines, requirements gathering procedures, and review guidelines. It is also important to document the quality assurance process. These processes require continual revisions and updates to ensure that the process is working as effective as possible.

The process improvement analysis should begin with an assessment of the software development practices that are currently being used by the company. This can be an informal assessment that compares the current practices to established practices such as those specified in the CMMI framework. These practices should be surveyed and assessed for strengths and weaknesses. This can be done in a simple brain storming session in which all team members participates. The focus of this meeting should be on areas that cause bottlenecks on previous projects, resources that are highly taxed and projects that have huge cost overruns. Employees can write ideas on index cards in order to take the spotlight off any individual team member. This approach should highlight a convergence of the most common problems experienced by team members. One of the issues that will undoubtedly arise is the problem of having multiple solutions to the same problem. This can be addressed by proper documentation and using a properly indexed document repository that supports reuse. The main reason for the group meeting to get some consensus of the most pervasive issues so that they can be addressed first

The action items from the brainstorming sessions can be documented and made accessible to the team members. The deliverables needs to be prioritized so that team members can focus on items that will have the greatest impact on improving the software development process. The document needs to be updated whenever progress has been made on implementing the software architecture changes. The organization should write clear development guidelines and expected deliverables for each stage of the development process.

### **Development Guidelines**

The organization or team needs to specify some development guidelines that can be applied to all their software development projects. The development guideline document is a document that contains the development steps agreed upon by the team. It specifies all the artifacts that must be produced from the project. The development guidelines serve to manage the expectations of the stakeholders of the development process. The process of setting development guidelines is an ongoing process that must be updated to reflect changes in the goals of the business. Below is an example of some categories of software development guidelines that can be useful to product development.

1. Requirement Acquisition Process
2. Specify stakeholders expectation
3. Document integration guidelines
4. Testing Plans
5. Quality Assurance Plans
6. Change Control process
7. Define User Interface Standards
8. User Manual and Documentation Guidelines

### **5.3. Implementation Phase**

The implementation phase consists of developing the documentation specified in the implementation plan. Organizations work on projects of different sizes so it may be useful to classify the documentation by the size of the project. These documents should be kept in a document repository that is accessible to project team members. The implementations phase must also specify as much information about targeted metrics that projects should aim to achieve. It should also specify the tools that are available for the software architecture. Software metrics can be collected about the current process so that they can compared to future projects in order to see the benefits of these techniques.

Some new skills may be needed that requires outside training. Additional specialized skills may not be necessary unless that are some direct industry specific knowledge to be gained. It may also be useful to have training sessions within the team where stronger team members can train weaker team members in areas such as product prototyping, design, better coding practices and application. This will ensure that the time is spent productively and applied to current projects.

It is better to systematically apply the new software architecture techniques, instead of trying to get it all implemented at once. Tools should be selected that will be most beneficial to ongoing projects.

Below is an example of software development and architecture documents that can be kept in a document repository.

## Project Management Templates

Document <i>(right-click document title to save a copy and use as a template)</i>	Phases					Executive or BOD Approval
	Initiate	Design	Develop	Implement	Close	
<b>Getting Started</b>						
Initiating a Project	•					
PMO Contacts						
Stakeholders Checklist	•	•	•	•	•	
IS&T Glossary	•	•	•	•	•	
PMI Glossary	•	•	•	•	•	
Information Systems Request (ISR)	•					
<b>Small Projects</b> <i>Less than \$1M and between 200 and 2000 hours</i>						
Stakeholder Checklist	•					
Project Plan Template	•	•			•	
Project Plan Instructions	•	•			•	
Project Questionnaire	•					
Requirements Matrix	•	•				
Estimation Workbook	•					
Scope Change Request	•	•	•	•		
Project Logs (Issues, Risk, Decisions, Action Items)		•	•	•	•	
Project Team Room (logs)		•	•	•	•	
Lessons Learned					•	
<b>Large Projects</b> <i>\$1M or more, or 2,000 hours or more</i>						
Stakeholder Checklist	•					
Project Plan Template	•	•			•	
Project Plan Instructions	•	•			•	
Project Questionnaire	•					
Requirements Matrix	•	•				
Infrastructure Support Checklist	•					
Project Risk Assessment	•	•	•			•
Project Logs (Issues, Risk, Scope Change)	•	•	•	•	•	
Estimation Workbook	•					
Executive Summary	•					•
Scope Change Request	•	•	•	•		
Project Financial Evaluation (PFE)	•					•
Project Schedule	•	•	•	•		•
Phase Exit Checklists	•	•	•	•	•	
Functional Design Specification Template		•				
Functional Design Specification - Instructions						
Technical Design Specification Template		•				
Technical Design Specification Instructions						
Implementation & Deployment Plan			•	•		
Salary Capitalization Project Guideline	•	•	•	•	•	
Test Plan Template			•			
Test Plan Instructions			•			
Project Closure Signoff						•
AFE Forms (Accounting Office)	•					

Table 1: Project Document Repository (18)

### **5.3.1. Implementing Software Engineering in a Small Workgroup**

The emphasis of this section is to show how selected software architecture practices can be applied to small team. Small software development companies or workgroups do not have the resources to develop a fully structured software architecture program. They are usually challenged with having too many project backlog and not enough resources to finish them. Small teams have few employees to perform all the software engineering tasks. Learning new software architecture skills can be time-consuming while trying to meet deadlines for customers and managers. The managers tend to think that their projects are too small to justify software engineering methods or costly project management and tracking tools.

Process improvement can be achieved using structured development methods (19) and practical quality assurance on selected project instead of pursuing state-of-the-art architecture techniques for all projects at once. Applying software architecture practices to selected project avoids overworking the team members while gaining valuable software architecture skills that can be expanded to new projects. Teams that employ this gradual improvement approach to software architecture can gain the skills and documentation they need at an affordable cost while taking advantage to existing resources.

### 5.3.2. Software Architecture Roles

- **Requirements Gathering**

Requirements' gathering is an important part of the software development process. The quality of the requirements directly impacts the outcome of the project. Companies should develop a requirements gathering plan to keep track of all the artifacts that are required during the requirements gathering phase. It is best to have one team member responsible for managing the requirement if it is a small team. Larger companies may have an entire team dedicated to managing requirements. Some useful tools for tracking requirements include spreadsheets such as Microsoft Excel or Openoffice.org. These tools can be used to document the required task and provide updates as necessary. Additional requirements gathering tools are available such as Microsoft SharePoint which provides a shared workspace, Rational ROSE, etc.

The requirements gathering process is fundamental to the success of the projects. It is important to develop a requirements gathering plan that involves input from all the stakeholders. Requirements acquisition sessions should be scheduled with the end users. Use case scenarios and stories are helpful methods of understanding how the requirements described by the user, solve the business need. It is also important to restate the requirements to the end user to ensure that the requirements engineer captures the requirement correctly. Templates have proved very useful in requirements gathering (20). Most end users are not technical experts so they cannot be expected to know the technical details of the system. Templates are visual in nature and they help to guide the end user into providing the information that is most relevant to the project. They also ensure that the requirements engineer asks the right questions of the end users during the requirements acquisition process. Below are some example documents templates that can be used to elicit functional and non-functional requirements

<b>Req. Id</b>	<i>&lt;requirement name&gt;</i>
<b>Version</b>	<i>&lt;current version&gt;</i>
<b>Author</b>	<i>&lt;author&gt;</i>
<b>Source</b>	<i>&lt;source of requirement&gt;</i>
<b>Description</b>	<i>&lt;Precise description of the requirement&gt;</i>
<b>Precondition</b>	<i>&lt;use case precondition&gt;</i>
<b>Sequence of events</b>	<i>Ordered sequence of actions</i>
<b>Post Condition</b>	<i>&lt;post-condition of the events&gt;</i>
<b>Exceptions</b>	<i>Sequence of exception&gt;</i>
<b>Performance</b>	<i>&lt;Response Time, etc&gt;</i>
<b>Importance</b>	<i>&lt;importance of the requirement&gt;</i>
<b>Urgency</b>	<i>&lt;urgency of the requirement&gt;</i>
<b>Comments</b>	<i>&lt;additional comments about the requirement&gt;</i>

**Table 2: Functional requirements Template**

<b>Req. Id</b>	<i>&lt;requirement name&gt;</i>
<b>Version</b>	<i>&lt;current version&gt;</i>
<b>Author</b>	<i>&lt;author&gt;</i>
<b>Source</b>	<i>&lt;source of requirement&gt;</i>
<b>Purpose</b>	<i>&lt;purpose for the requirement&gt;</i>
<b>Description</b>	<i>&lt;Precise description of the requirement&gt;</i>
<b>Data</b>	<i>&lt;specific data relevant to the requirement&gt;</i>
<b>Importance</b>	<i>&lt;importance of the requirement&gt;</i>
<b>Urgency</b>	<i>&lt;urgency of the requirement&gt;</i>
<b>Comments</b>	<i>&lt;additional comments about the requirement&gt;</i>

**Table 3: Site Requirements Template**

<b>Req. Id</b>	<i>&lt;requirement name&gt;</i>
<b>Version</b>	<i>&lt;current version&gt;</i>
<b>Author</b>	<i>&lt;author&gt;</i>
<b>Source</b>	<i>&lt;source of requirement&gt;</i>
<b>Description</b>	<i>&lt;Precise description of the requirement&gt;</i>
<b>Importance</b>	<i>&lt;importance of the requirement&gt;</i>
<b>Urgency</b>	<i>&lt;urgency of the requirement&gt;</i>
<b>Comments</b>	<i>&lt;additional comments about the requirement&gt;</i>

**Table 4: Non-functional Requirement Template**

<b>Req. Id</b>	<i>RN-56 –System Reliability</i>
<b>Version</b>	<i>2.0 (5/15/2009)</i>
<b>Author</b>	<i>Paul Taylor</i>
<b>Source</b>	<i>Don Smith</i>
<b>Description</b>	<i>The system shall be online 97% of the time with a maximum of 30 minutes consecutive downtime</i>
<b>Importance</b>	<i>Vital</i>
<b>Urgency</b>	<i>Urgent</i>
<b>Comments</b>	<i>n/a</i>

**Table 5 : Non-Functional Requirement**

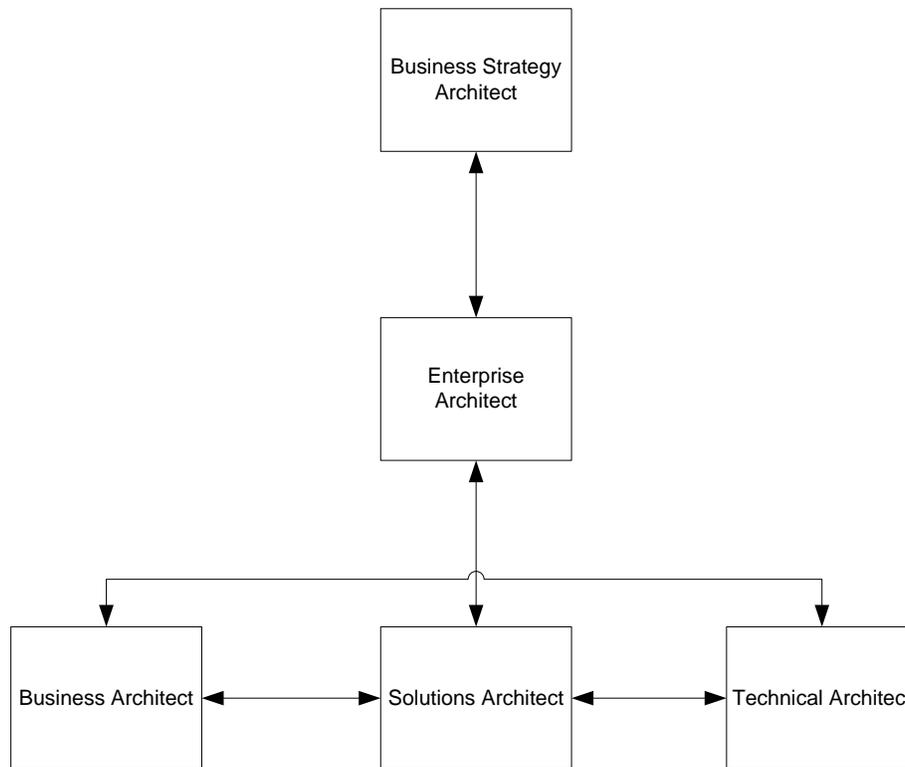
- **Software Architect**

Software architects need to have a competent understanding of the business that they support.

Software architects are generally IT specialists with limited knowledge of business matters. The software architect should remain focused on how each software decision aligns with the goals of the business.

The software architect must be someone who understands the business, its structure and what kind of problems affects the company. The architect must be able to show how each technical decision aligns with the business goals.

Businesses can solve this problem of aligning software architect with the business goals by specifying multiple architecture roles. The organization structure described below defines the five architectural roles that can be used to align software architecture with the business goals (21). These roles are business strategy architect, enterprise architect, business architect, solutions architect and technical infrastructure architect. It is important to note that these roles can be filled by individuals that perform each role, a team or a single individual that perform all the roles.



**Figure 2: Architect roles in business-software development**

- **Business-Strategy Architect**

The functions of the Business strategy architect must be closely aligned with the long term goals of the business. This role focuses on the business strategy rather than the IT strategy by working closely with upper management to ensure that the strategic decisions have the proper technical solutions and support. In today's market many business decisions are driven by technology even in companies that are not traditionally considered technology companies. The business strategy architect role is therefore part of the team that determines the direction of the business or work closely with upper management to communicate how different technology solutions will impact the company. They are able to provide

insight into what technologies are right for the business and the appropriate time to implement these solutions to maximize the benefits to the business.

- ***Business Architect***

The Business Architect Association (BAA) defines a business architect as follows (22)

*A Business Architect is the member of the team whose primary responsibility is to take the "big picture" future view of the structure of the organization. A Business Architect is generally responsible for both ongoing and project-based work, and provides services in important situations and strategic implementation.*

The job of the business architect is to provide the structure which is used by the software architect to develop the software solution. The business architect works in conjunction with the business to gather requirements that address the business need and are in line with the company goals. A business architect should possess the following skills; strong strategic and analytical skill, extensive knowledge of the business, strong marketing skills along with good communication skills to be able to communicate effectively between the management and IT. The business architect is the person that is responsible for ensuring that the software is in line with the company needs. The solution developed by the software architect should map directly to the specifications supplied by the business architect.

- ***Solution Architect***

The solutions architect is essentially the application architect. The solution architect is responsible for designing the technical structure of the application based on the requirements specified by the business analyst. The solutions architect should possess analytical skills and is capable of transferring the requirements specified by the user into a layered view of the system. The solutions architect must be careful not to make assumptions about the business needs but should instead work with the business analyst to clarify unclear requirements. There are a number of tools that are available to design the software architecture some of which is discussed in section 5.3.3.

- ***Technical-Infrastructure Architect***

The Technical Infrastructure architect is responsible for the technical infrastructure that will support the application. This includes areas such as hardware, operating system, network system and system software. The requirements for the technical architecture are derived from the non-functional requirements specified by the user. These include specifications about timing, performance, security, logging and error services.

- ***Enterprise Architect***

The role of the enterprise architect is to connect all of the other architectural roles. An enterprise architect is someone that understands both the needs of business and the technical requirements of the system and is capable of being an advocate for both. The Enterprise architect is responsible for aligning the project with the goals of the organization and must therefore be capable of maintaining the big picture and how the project impacts the organization.

- ***Project Champion***

In addition to the role of the software architect, the role of the project champion deserves special mention. The project champion role is to be an advocate for the user community. The software solution should satisfy user requirements and therefore having a project champion working with the development teams ensure that the needs of the users are constantly being accounted for. A project champion will be able to communicate conflicting requirements back to the end users. The project champion will also help to prioritize competing quality attributes for individual projects.

- ***Software Quality Assurance***

Quality Assurance is an important role in software architecture. Quality Assurance is the last line between software products and customers. A software assurance program must be set up to ensure that the product that the customers receive meets a set of minimum acceptable quality standards. The complexity of software systems make it impossible to release a perfect software system. Setting quality assurance standards allows managers to effectively determine when the product is acceptable for customers. Software Quality standards include setting a value for number of new bugs uncovered per thousand lines of code based on past experienced.

### **5.3.3. Software Architecture Tools**

There are a number of tools and methodologies that are available for implementing a software architecture program. These tools were developed to track the development process and allow for timely analysis so that corrective measures can be taken to ensure a smooth software implementation. These tools are available to support different areas of the software architecture ecosystem. There are specific tools that allow non technical management visibility into the architecture process as well as tools tailored for developers and other stakeholders. The following section examines some of the tools that are available to help with the implementation.

- ***Architecture-Based System Evolution***

Most software systems are designed with the understanding that they will evolve over time. This could result from changes in the business mission and goals requiring that the software adapt to a new business model or changes in the technology to which the business must adapt in order to remain competitive. The Architecture-Based System Evolution method is a means by which to ensure that the system evolves and continues to serve the goals and mission of the business throughout its lifetime. As the business change the software architecture needs to be redesigned to improve quality and address deficiencies with the system. Business can develop variations of the Architecture Base System evolution that fits their specific software model such as high availability critical system (23) or automated system (24).

- ***Architecture Competence Assessment***

Architecture Competence Assessment can be used to assess a business's ability to perform and sustain architecture-centric development and software evaluation that serves the company's goals. It is used to determine if a company has the ability to grow, use, and sustain the knowledge and skills that is necessary for continued architecture-based development. The company must rely on proper documentation of the software architecture process and continual analysis in order to ensure that their software architecture is aligned with the company goals (25).

- **Architecture Expert (ArchE)**

The Architecture Expert (26) tool is a rule-based architecture design assistant tool that helps software architects analyze software architecture for quality attributes. The ArchE shows a meta-model that provides visibility into competing requirements and helps the software architect manage tradeoffs among quality attributes such as knowledge of quality attribute models, how to analyze software architecture for its quality attribute properties, and how to manage tradeoffs among the quality attributes. ArchE can be used to generate some design alternatives and present them to the architect in order to help with their design decisions. Below is an example of a quality tradeoff analysis proposed by ArchE.

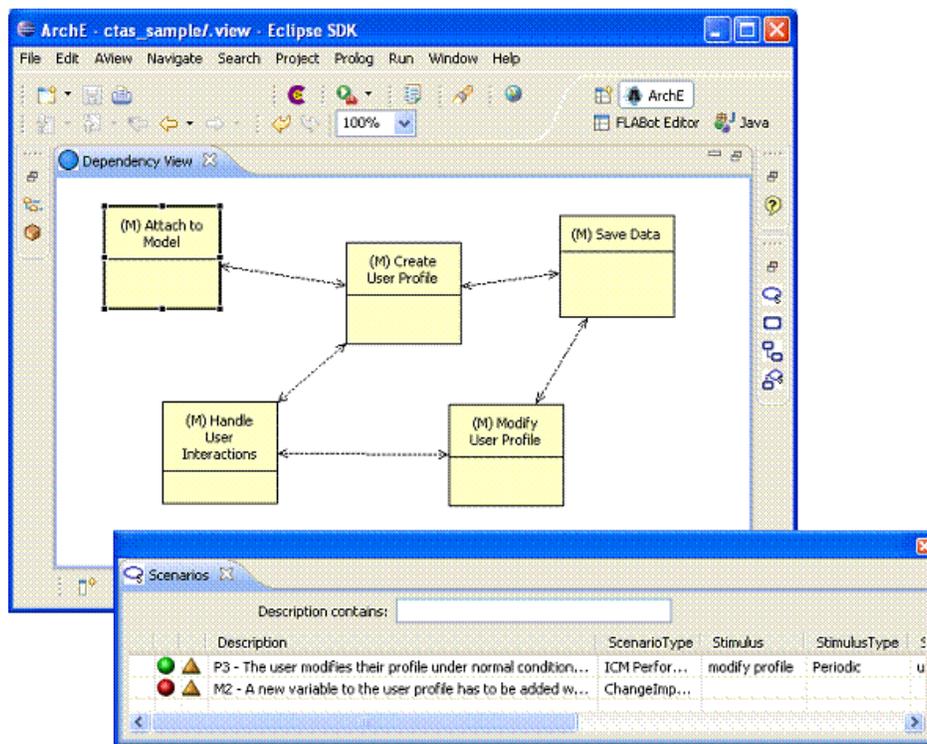


Figure 3: SEI Architecture Expert (ArchE) Tool

- ***Attribute-Driven Design***

The Attribute-Driven Design (ADD) is a method of designing software architecture based on the functional and quality attribute specified in the requirements. These include specifications about attributes such as performance, security, maintainability, reliability, availability and usability. The ADD method enables software architects to understand the effects of quality attributes early into the architectural process and allow architects to present different options to fit a solution. They also allow for an architectural solution be customizable to fit varying product lines.

The ADD method is a top-down recursive decomposition method in which architectural significant requirements are chosen at each level of the process. The requirements are analyzed for how quality attributes aligns with the business objectives. The child object and components are chosen to satisfy the specifications of the higher priority qualities. The outputs for each stage of the ADD are passes and input for the next level.

- ***Cost Benefit Analysis Method***

The Cost Benefit Analysis Method (CBAM) is used to determine the cost benefit implications of architectural decisions while managing uncertainty in future business and mission goals. It allows for a more concrete analysis of budget estimates and allows managers to make more informed decision about the viability of the project. The Cost Benefit Analysis Method consists of choosing architectural strategies and assessing the benefits based on the quality, cost, schedule and desirability. The CBAM will allow stakeholders of choose an architecture design that provides the highest level of satisfaction to their needs.

#### **5.4. Software Architectural Program Assessment**

The software architecture assessment phase is an ongoing phase. This is where actual data from metrics collected from projects implemented using the software architecture program is analyzed in order to identify their benefits. Information gathered through this assessment should be incorporated into the software architectural program in order to ensure continual improvements.

#### **5.5. Evaluating Software Architecture**

Problems found and fix in the architectural phase are cheaper to fix than later in the development process (27). This is why it makes sense to try and identify and fix problems with the architecture early into software development process. Software architecture evaluation methods can be used to determine if the right architecture has been chosen for the project and whether or not the architecture solves the problem. Software architecture evaluations can help to avoid risk by catching problems early that could later prove costly to the project. Software Architecture Metrics and other evaluation tools can be used to assess software architecture quality attributes such as reusability, reliability, interoperability, portability and maintainability. Architectural evaluation can occur at any stage in the creation process. The process of analyzing the architectural decisions can begin as soon as the decisions are made. It is better to make the architectural evaluation process an integral part of the architecture creation process. This will eliminate the need for a full blown architectural review at the end of the process. The architectural evaluation should be performed by the evaluation team as well as key stakeholders of the system such as the project champion and other project decision makers. The following section gives a brief overview of some useful techniques for software architecture evaluation.

- ***Active Reviews for Intermediate Designs (ARID)***

Active Reviews for Intermediate Designs (28) is a method for reviewing architectural specifications for system when there are more information available than just a high level specification , however the architecture may not be fully documented. The ARID will help provide some early insights into how well the architecture is adhering to the requirements specification. This method is a low-cost way to ascertain whether or not the preliminary design specification is suitable for the application domain. This is done by mapping use cases specified by the stakeholders to the initial design specifications in order to ensure that the stakeholder needs are satisfied.

- ***Software Architecture Analysis Method***

Software Architecture Analysis Method (SAAM) was developed in 1994 as a method for describing and analyzing the properties of software architectures (29). SAAM allows for the specification of acceptable quality attributes of the architecture, thus they can be compared with the actual attributes of the architecture to determine if the user requirements are satisfied by the architecture. A prioritized list of attributes can be used to guide decisions. The SAAM methodology also allows for evaluation of risk and mapping of the quality attributes to an architectural specification. The Architecture Tradeoff Analysis Method discussed below evolved out of the Software Architecture Analysis Method and can be used to provide additional analysis of the architecture.

- ***Architecture Tradeoff Analysis Method***

Architecture Tradeoff Analysis Method (ATAM) provides a method of evaluating software architectures relative to the quality attribute goals specified by the business unit. The ATAM process works by bringing stakeholders together throughout the architectural creation process to ensure that the architecture meets the required specifications. The ATAM exposes architectural risks with the system design that are not in line with the company's goals and objectives. It allows for the prioritization of system goals in order to provide a standard mechanism for handling conflicting goals. The ATAM provides a catalog of the architectural approaches used so that they may be available for reuse as part of the common knowledge base. This in turn serves to improve the quality of the architectural documentation and contributes to the refinement of the architectural process. Key architectural decisions are specified by the ATAM as sensitivity points or trade-off points (30). This specifies properties, components or relationships that are critical to achieving specified quality attributes, such as the level of confidence in a security encryption scheme or the maximum acceptable downtime for the system per month.

## **6. Conclusion**

The software development process is very complex. Software architecture is widely used to manage this complexity. Continued research on tools and techniques are making it easier for business to adopt software architecture techniques into their software development process. Software architecture has drastically changed the way that people think about software development. This is evident from Fred Brooks suggestion in the original publication of The Mythical Man-Month in 1975 suggesting them we should plan on building one system to throw away (31) to his revised advice in the 20th anniversary edition published in 1995 where he suggest using process management techniques to avoid a throw away system (8). This paper discussed the aspects of the software architecture process and how it can be used to improve the software development process in companies or organizations that do not currently employ software architecture practices.

## References

1. **Len Bass, Paul Clements, Rick Kazman.** *Software architecture in practice.* s.l. : Addison-Wesley, 1998.
2. **P Clements, L Northrop.** *Software product lines.* s.l. : Software Engineering Institute Carnegie Mellon, 2001.
3. *Foundations for the study of software architecture.* **DE Perry, AL Wolf.** s.l. : ACM SIGSOFT Software Engineering Notes, 1992.
4. *Overcoming Obstacles in Implementing SOA.* **Lublinsky, Boris.** s.l. : InfoQ, 2008.
5. **Kruchten, Philippe.** *Ten Common Misconceptions about Software Architecture.* s.l. : Rational Software Corp., 1998.
6. *An introduction to software architecture.* **D Garlan, M Shaw.** s.l. : Advances in Software Engineering and Knowledge Engineering, 1993.
7. *Abstractions for software architecture and tools to support them.* **M Shaw, R DeLine, DV Klein, TL Ross, DM Young, G.** s.l. : IEEE Transactions on Software Engineering, 1995.
8. **Jr, FP Brooks.** *The Mythical Man-Month.* s.l. : Addison Wesley, 1995.
9. **RL Purvis, GE McCray.** *Project assessment: A tool for improving project management.* s.l. : Information Systems Management, 1999.
10. **Reel, JS.** *Critical success factors in software projects.* s.l. : IEEE software, 1999.
11. **Charette, RN.** *Why software fails.* s.l. : IEEE spectrum, 2005.
12. **A Stellman, J Greene.** *Applied Software Project Management, 1st Edition.* 2005.
13. *Capability maturity model® integration (CMMI SM), Version 1.1.* **CP Team - Pitsburg.** s.l. : Software Engineering Institute, 2001.
14. *An Analysis of SEI Software Process Assessment Results 1987-1991.* **Dave Kitson, Steve Masters.** s.l. : IEEE COMPUTER SOCIETY, 1992. CMU/SEI-92-TR-024.
15. **Andrea O. Salas, Joanne L. Walsh, James C. Townsend.** *Software Engineering Process Group.* s.l. : NASA, 2003. FY03 ASCAC Representation.
16. *Capability Maturity Model for Software (Version 1.1).* **M.Paulk, B. Curtis , M. Chrissis , C. Weber.** 1993. CMU/SEI-93-TR-024.
17. *Section 3002 of the Digital Transition and Public Safety Act of 2005.* 47 U.S.C. § 309(j)(14). 2005.

18. **Templeinland**. *Project Management Document Repository*. 2008.
19. **Wiegers, KE**. *Creating a software engineering culture*. 1996.
20. *A Requirements Elicitation Approach Based in Templates and Patterns*. **AD Toro, BB Jiménez, AR Cortés, MT Bonill**. s.l. : Ibero-American Workshop on Requirements Engineering, 1999.
21. *Business Improvement Through Better Software Architecture*. **Sten Sundblad, Per Sundblad**. s.l. : Microsoft Architect Journal, 2007.
22. *Business Architecture: An Emerging Profession*. **Paul A. Bodine, Jack Hilty**. s.l. : Business Architects Association Institute, 2009.
23. *Architecture-based runtime software evolution* . **P Oreizy, N Medvidovic, RN Taylor**. s.l. : Proceedings of the 20th international conference on Software, 1998.
24. *An architecture-based approach to self-adaptive software*. **Oreizy, P. Gorlick, M.M. Taylor, R.N. Heimhigner, D. Johnson, G. Medvidovic, N. Quilici, A. Rosenblum, D.S. Wolf, A.L.** s.l. : Intelligent Systems and their Applications, IEEE, 1999.
25. *Evaluating the Software Architecture Competence of Organizations*. **Bass, L., et al.** s.l. : Seventh Working IEEE/IFIP Conference on Software Architecture , 2008.
26. *Preliminary Design of ArchE: A Software Architecture Design Assistant*. **Felix Bachmann, Len Bass, Mark Klein**. s.l. : CMU/SEI, 2003.
27. *Introduction to software engineering*. **Leach, Ronald J.** 2000.
28. *Active Reviews for Intermediate Designs*. **Clements, Paul C.** s.l. : CMU/SEI, 2000.
29. *SAAM: A method for analyzing the properties of software architectures*. **R Kazman, L Bass, M Webb, G Abow**. s.l. : IEEE COMPUTER SOCIETY, 1994. ISSN 0270-5257.
30. *Evaluating a Software Architecture*. **Paul Clements, Rick Kazman, Mark Klein**. s.l. : Addison Wesley, 2001.
31. **Brooks, FP**. *The mythical man-month: essays on software engineering*. 1975.

## VITA

Paul Leonardo Taylor was born in Kingston, Jamaica on March 1, 1979, the son of Adlin Taylor and Lascelles George Taylor. After completing his work at Camperdown High School, Kingston, Jamaica, He migrated to the United States of America to attend Arkansas Tech University in Russellville, Arkansas where he received the Bachelor of Science in May, 2002. He was employed as an Information Technology Specialist at Dell Computer Corporation in 2003. During the years that followed he was employed as a Software Engineer at Temple-Inland Inc located in Austin, Texas. In August 2007 he enrolled into the Cockrell School of Engineering Graduate School at the University of Texas at Austin.

Permanent Address:                   13509 Briarcreek Loop  
  
  Manor, Texas 78653

This report was typed by Paul L. Taylor