Software Engineering featuring the Zachman Taxonomy

Pia Stoll, Anders Wall
Industrial Software Systems
ABB Corporate research
pia.stoll@se.abb.com,
anders.wall@se.abb.com

Christer Norström
Computer Science and Electronics
Mälardalen University
christer.norstrom@mdh.se

August 28, 2009

Abstract

Software engineering of today must consider organizational- and business issues as well as architectural issues for fast manufacturing of software. The semantics in a taxonomic scheme including organizational-, business- and architecture artifacts would help software engineers define explicit relations between software engineering artifacts at all software design-levels.

In this report, we present the Software Engineering Taxonomy, derived from the Zachman Enterprise Architecture Framework. The Software Engineering Taxonomy proved to be able to classify all software engineering artifacts from the IEEE Software Engineering Book Of Knowledge (SWEBOK) published 2004.

The Software Engineering Taxonomy also proved to give useful insights into how customer sites and development sites may interact for fast innovation exemplified with the companies Apple (AppStore) and Google. The taxonomy also proved to be useful for process analysis which is shown for the Scrum process.

1 Introduction

This report investigates the possibility of classifying software engineering artifacts for industrial software systems. The classification should include artifacts related to business and organization and therefore three Enterprise Architecture frameworks were considered. The three frameworks were: the Zachman framework, the Department Of Defense Architecture Framework (DODAF) [1] and The Open Group Architecture Framework (TOGAF) [2].

The discipline of enterprise architecture is commonly considered to have its birth in an academic article by John Zachman published 1987 by the research oriented IBM Systems Journal [3]. Zachman saw the growing complexity of information software system that extended in scope and complexity to cover an entire enterprise. He stated that decentralization of system resources without structure results in chaos and argued for the need of information system architecture. Zachman searched for an objective independent basis upon which to build a framework for information system architecture and resolved to be inspired by classic architecture.

Enterprise architecture as defined by the Federal Architecture Working Group (FAWG) [4] is: a strategic information asset base and describes the mission (i.e. the business), the information necessary to perform the mission and the technologies necessary to perform the mission, and the transitional processes for implementing new technologies in response to changing mission needs. An enterprise[1] architecture includes a baseline architecture[2], target architecture[3], and a sequencing plan[4].

According to James N. Martin [5] enterprise architecture deals with "Getting to the Future" and has drivers and outcomes. The enterprise architecture is according to Martin a means for transforming enterprise objectives into business plans and mission needs.

In the mid 1990s the DoD determined that a common approach was needed for describing its architectures, so that DoD systems could efficiently communicate and inter-operate during joint and multinational operations. The interoperability aspects of the DODAF is reflected in its architectural views which are focused on describing what's being communicated and how in the Operational View (OV) of the DODAF. The Systems View (SV) of DODAF identifies the systems that support the OVs and the Technical View (TV) describes the criteria for each required system that will satisfy the interoperability requirements. DODAF is as such not an architecture development method or a classification framework, it's an architecture description development framework focused on describing interoperability aspects of systems of systems.

TOGAF[5] is developed and maintained by members of The Open Group, working within the Architecture Forum. The original development of TOGAF Version 1 in 1995 was based on the Technical Architecture Framework for Information Management (TAFIM), developed by the US Department of Defense (DoD). The DoD gave The Open Group explicit permission and encouragement to create TOGAF by building on the TAFIM, which itself was the result of many years of development effort and many millions of dollars of US Government investment.

TOGAF is more ambitious in scope than its defense counterpart, DODAF. TOGAF organizes architectures into four domain levels: Business architecture - defines business strategy, governance, organization, and key business processes; Application architecture - specifies individual application systems to be deployed; Data architecture - defines structure of an organization's logical and physical data assets and associated data management resources; and Technology architecture - specifies software infrastructure intended to support the deployment of core, mission-critical applications.

As this report was searching for a enterprise architecture artifact classification framework, not an enterprise architecture description development framework or in-house information system architecture development framework, it resorted to study the Zachman framework in more detail as the Zachman taxonomy is a light-weight ontology that classifies enterprise architecture artifacts as described in Section 2.

The remainder of this report is organized as follows; Section 2 describes the Zachman Framework, Section 2.1 describes the Software Engineering Taxonomy and the classification of the SWEBOK software engineering artifacts, Section 2.3 and Sec-

[1] Enterprise - an organization supporting a defined business scope and mission. An enterprise includes interdependent resources (people, organizations, and technology) who must coordinate their functions and share information in support of a common mission.

[2] Baseline architecture - the architecture as it is today, also called as-is architecture

[3] Target architecture - the (planned) future architecture, also called to-be architecture or goal architecture

[4] Sequencing plan - the strategy for changing the baseline architecture to the target architecture, also called the transition plan

[5] http://www.opengroup.org/architecture/togaf9-doc/arch/

tion 2.4uses the Software Engineering Taxonomy from Section 2.1 to analyze the cases: AppStore, Google and Scrum, and Section 3 presents the conclusions of the work with the Software Engineering Taxonomy and its usefulness for the software engineering discipline and future work.

2 Zachman Framework

In a joint article [6] published 1993, Sowa and Zachman explain that the Zachman framework links the concrete things in the world (entities, processes, locations, people, times and purposes) to the abstract bits in the computer. The Zachman framework is not a replacement of programming tools, techniques, or methodologies but instead, it provides a way of viewing the system from many different perspectives and how they are all related. The framework logic can be used for describing virtually anything. The logic was initially perceived by observing the design and construction of buildings. Later it was validated by observing the engineering and manufacture of airplanes. Subsequently it was applied to enterprises during which the initial material on the framework was published.

The objective to use the Zachman framework [7] [3] [8], according to Zachman, is to reduce time-to-market for anything substantive and the way to do this is to create primitive re-usable model descriptions designed to be reused in more than one implementation (composite) anywhere in the enterprise. Zachman further states that the only way of managing the impact of change is to manage vertical and horizontal relationships in the framework.

According to Zachman, “Architecture” is the set of descriptive representations relevant for describing a complex object (actually, any object) such that the instance of the object can be created and such that the descriptive representations serve as the baseline for changing an object instance. Descriptive representations (of anything) typically include “Abstractions” classifying the description focus: Inventory Sets(What), Process Transformations(How), Network Nodes(Where), Organization Groups(Who), Timing Periods(When), Motivation Reasons(Why). Descriptive representations also include “Perspectives” classifying the description usage: Scope Concepts, Business Contexts, System Logic, Technology Physics, and Components. The relevant descriptive representations would necessarily have to include all the intersections between the Abstractions and the Perspectives (Figure. 1). “Architecture” would be the total set of descriptive representations (models) relevant for describing the complex object and required to serve as a baseline for changing the complex object once it is described. Zachman’s complex object is the enterprise, but principally he states that the complex object can be any object.

The Zachman framework is a structure, not a methodology for creating the implementation of the object. The Zachman Framework does not imply anything about how architecture is done (top-down, bottom-up, etc). The level of detail is a function of a cell not a function of a column. The level of detail needed to describe the Technology Physics perspective in row four may be naturally high but it does not imply that the level of detail of the row one descriptions should be lower or the opposite.

The framework is normalized, that is adding another row or column to the framework would introduce redundancies or discontinuities. Composite models and process composites are needed for implementation. A composite model is one model that is comprised of elements from more than one framework model. For architected implementations, composite models must be created from primitive models and diagonal

Abstraction / Perspective	INVENTORY SETS (WHAT)	PROCESS TRANSFORMATIONS (HOW)	NETWORK NODES (WHERE)	ORGANIZATION GROUPS (WHO)	TIMING PERIODS (WHEN)	MOTIVATION REASONS (WHY)
SCOPE CONTEXTS (Strategists)	e.g. Inventory Types	e.g. Process Types	e.g. Network Types	e.g. Organization Types	e.g. Timing Types	e.g. Motivation Types
BUSINESS CONCEPTS (Executive Leaders)	e.g. Business Entities & Relationships	e.g. Business Transform & Input	e.g. Business Locations & Connections	e.g. Business Role & Work	e.g. Business Cycle & Moment	e.g. Business End & Means
SYSTEM LOGIC (Architects)	e.g. System Entities & Relationships	e.g. System Transform & Input	e.g. System Locations & Connections	e.g. System Role & Work	e.g. System Cycle & Moment	e.g. System End & Means
TECHNOLOGY PHYSICS (Engineers)	e.g. Technology Entities & Relationships	e.g. Technology Transform & Input	e.g. Technology Locations & Connections	e.g. Technology Role & Work	e.g. Technology Cycle & Moment	e.g. Technology End & Means
COMPONENT ASSEMBLIES (Technicians)	e.g. Component Entities & Relationships	e.g. Component Transform & Input	e.g. Component Locations & Connections	e.g. Component Role & Work	e.g. Component Cycle & Moment	e.g. Component End & Means

Figure. 1: The Zachman Framework

composites from horizontally and vertically integrated primitives. The structural reason for excluding diagonal relationships is that the cellular relationships are transitive. Changing a model may impact the model above and below in the same column and any model in the same row.

For manufacturing a process composite would be necessary. The process composite describes the working process of creating the model descriptions of the composite model, typically ending with the descriptions of the components in the Component Assemblies perspective, e.g. a service or framework. A third dimension of the framework, called science, has been proposed by O'Rourke et al. [9]. This extension is known as the Zachman DNA (Depth iNtegrating Architecture). In addition to the perspectives and aspects the z-axis is used for classifying the practices and activities used for producing all the cell representations.

The Zachman Framework has been used almost exclusively for information system modeling. The reason is to find in the integration between the Business Concepts perspective and the System Logic perspective. Zachman prescribes that the Scope Contexts perspective and the Business Concepts perspective should describe the enterprise's own scope and business. The System Logic perspective, the Technology Physics perspective, and the Components perspective should describe the system's support of the enterprise's Scope Contexts perspective and Business Concepts perspective. For an enterprise that builds an in-house information system this is natural since information systems's goal is to support in-house business processes. Enterprises building information systems to support their own business processes has control over all five perspectives in line with Zachman's argumentation that the enterprise should only enter controllable model descriptions into the framework.

Being in control of a model translates into having the power to change the model description. If a person is in control of the model describing the system's technology processes, this person has the power to change e.g. the classes and their relations in the system.

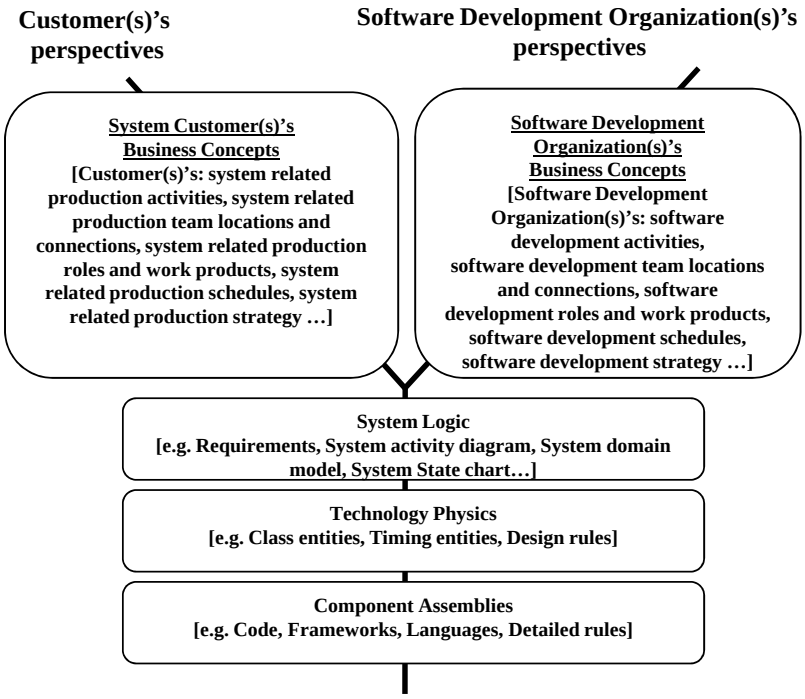

Figure. 2: The Customer's and the Software Development Organization's perspectives

2.1 Software Engineering Perspectives

In order to be able to use the Zachman framework for software engineering artifacts, two basic assumptions were done:

1. The classification framework, derived from the Zachman framework, describes the software development organization and the descriptions of the customer's enterprise that relates to the usage of the system

2. The classification framework, derived from the Zachman framework, is three-dimensional where site is the third dimension. The site might be the software development organization, external development organization or the customer's enterprise as long as the site has a part in the system usage or system development.

The assumptions are illustrated in Figure 2. With these assumptions, the system development's Business Concepts perspective will describe the software development artifacts, e.g. software development activities, software development team locations and connections, software development roles and work products, software development schedules, and software development strategies. The models in the customer's Business Concepts perspective will describe the customer's production related to the need of system support. The resulting classification framework is called the Software Engineering Taxonomy.

The models in the Software Engineering Taxonomy might be shared across development and customer sites but it is the software development organization that controls

the degree of openness. For example if the customer is an active member in the requirements handling team at the software development organization, then the requirements handling activity is shared across sites. This would mean that the model with the Business Concepts perspective and Process Transformations abstraction is partly shared since this model contains the requirements handling activity. Another example of shared models across sites is the open source development. In open source development, several software development sites share software development model descriptions across sites. Not only the development activities can be shared across sites, but also the testing activities. Google lets their customers test the Google software applications before the final release, which makes the customers part of the test team.

2.2 Software Engineering Descriptions

The Software Engineering Body Of Knowledge, SWEBOK, guide has the objective to promote a consistent view of software engineering worldwide and was published 2004 [10]. SWEBOK has references to a very large number of software engineering theories. The IEEE SWEBOK has divided the software engineering domain into a set of knowledge areas; software requirements, software design, software construction, software testing, software maintenance, software configuration management, software engineering management, software engineering process, software engineering tools and methods, and software quality. The knowledge areas act as knowledge for the persons working in that specific area. In contrast to the work described in [11] [12] the classification of the SWEBOK software engineering artifacts in this report does not try to reflect the software engineering professions but instead seek a classification approach similar to the building engineering [13] .

The software engineering artifacts are not physical like in the building engineering but differ in their descriptions, not in their physical dimensions. A software engineering description can be very complex, e.g. a domain model including a large set of entities and their relations, and it can be less complex, e.g. the listing of reports. Some of the software engineering artifacts from SWEBOK could be descriptions of their own. For example, the artifact “System Class diagram” could be a complete description of the model with the Inventory Sets abstraction and the Technology Physics perspective.

The process used to classify the descriptions from the SWEBOK was:

1. Anything resembling an artifact was extracted from the SWEBOK.
2. The artifact duplicates were removed when all of the artifacts were extracted.
3. The non-duplicate artifacts were analyzed and grouped according to their descriptions. For example, the “Release Schedule” artifact was grouped together with “Construction Schedule”, “Project Schedule and milestones”, and “Test Schedule”.
4. The grouped artifacts were translated into general model descriptions. For example, the group of schedules in the previous step was generalized into the model description “Schedules for projects, releases and processes”.
5. The general model descriptions were classified according to the perspectives and abstractions from the Software Engineering Taxonomy. When performing the classification it was important to distinguish between the customer’s perspectives and the development organization’s perspectives. For example, the model

description “Schedules for projects, releases and processes” was classified with a Timing Periods abstraction and Software Development Organization’s Business Concepts perspective.

The software engineering artifacts which describe the development of a software system and the system itself were expected to be straight-forward to classify in Zachman’s System Logic perspective since this perspective contains model descriptions of software system architecture, e.g. use cases, activity diagrams, requirements.

It was also expected that Zachman’s Business Concepts perspective would be difficult to use for Software Engineering artifacts since this perspective is typically used to model the customer’s business processes in need of system support, e.g. production processes. The classification showed that the development organization’s Business concepts as e.g. software testing, development schedules, prototype analysis etc could easily be classified in the Software Engineering Taxonomy when the taxonomy perspectives were the software development organization perspectives.

Non of the SWEBOK software engineering artifacts described the customer’s Scope Contexts or customer’s Business Concepts perspective and hence the classification of the SWEBOK artifacts is done using only the software development organization’s perspectives. The resulting classification of the SWEBOK artifacts is two-dimensional and shown in Figure 3 and in Figure 4.

Abstraction → / Software Development Organization Perspective ↓	INVENTORY SETS (WHAT)	PROCESS TRANSFORMATIONS (HOW)	NETWORK NODES (WHERE)
SCOPE CONTEXTS	[Reports, Standards, Stakeholders, Tools, Products]	[Requirement Handling/ Design/ Construction/ Testing/ Maintenance/ Configuration Management/ Engineering Management]	[Internal & External Development Team Networks, Supplier Networks]
BUSINESS CONCEPTS	[Estimations, Prototypes, Analysis, …, Decisions and their Relations]	[Activities for: Requirement Handling/ Design/ Construction/ Testing/ Maintenance/ Configuration Management/ Engineering Management]	[Internal & External Development Team Locations & Connections]
SYSTEM LOGIC	[System Domain Model]	[System Activity Diagram]	[System Deployment Diagram]
TECHNOLOGY PHYSICS	[Development View, Class Diagram]	[Logical View, Interface Specification]	[Physical View, Deployment Diagram]
COMPONENT ASSEMBLIES	[Database Configuration, Build Configuration]	[Algorithms, Languages, Code Modules, Frameworks]	[Communication Protocols, Port Configurations]

Figure. 3: The SWEBOK software engineering artifacts classified in the Software Engineering Taxonomy. The figure shows the three first columns.

The generalized model descriptions are enclosed by brackets in each cell of the Software Engineering Taxonomy in Figure 3 and Figure 4. For example, the cell with

ORGANIZATION GROUPS (WHO)	TIMING PERIODS (WHEN)	MOTIVATION REASONS (WHY)	← Abstraction ↓ Software Development Organization Perspective
[External Regulatory Bodies, Internal & External Development Teams]	[Internal & External Development Releases, Global Economy Events, …]	[Business Goals, Process Scopes, Policies, Culture, Principles, Missions]	SCOPE CONTEXTS
[Internal & External Development Team Roles & their Work Products]	[Schedules for Projects/ Releases/ Processes]	[Strategies for: Processes'/ Projects'/ Staffing/ System and Projects' Objectives]	BUSINESS CONCEPTS
[System Use Cases]	[State Charts]	[Requirements, Constraints, Qualities]	SYSTEM LOGIC
[User Interfaces]	[Sequence Diagrams]	[Design Rules, Design Principles]	TECHNOLOGY PHYSICS
[Security Control, Safety Control]	[Concurrency Model]	[Explicit Design Rules/ Design Principles Configuration]	COMPONENT ASSEMBLIES

Figure. 4: The SWEBOK software engineering artifacts classified in the Software Engineering Taxonomy. The figure shows the three last columns.

the Inventory Sets abstraction and Scope Contexts perspective contains descriptions of reports, stakeholders, standards, tools and products used by the software development organization. The cell with the Process Transformations abstraction and Scope Contexts perspective contains model descriptions of processes for requirement handling, design, construction, testing, maintenance, configuration management, and engineering management. In the Software Engineering Taxonomy, the processes do not dictate the classification; they are a part of the classification scheme. The Scope Contexts- and Business Concepts perspectives with the Process Transformations abstraction got a large number of artifacts classified since SWEBOK contains a large amount of process descriptions and activity definitions for the processes. The models description are instantiated for each software development organization. For example, an organization doing Scrum [14] processes would instantiate the model with the Business Concepts perspective and Process Transformations abstraction with descriptions of typical Scrum activities: "Sprint Review", "Planning", etc.

2.3 Apple and Google Process Composite Models

We have re-engineered the interactions between development sites and customer/utilization sites into our Software Engineering Taxonomy for two companies: Apple (AppStore) [[15], [16], [17]] and Google [18]. The companies are world-leading [19] in establishing new ways of interacting with their customers during software development and

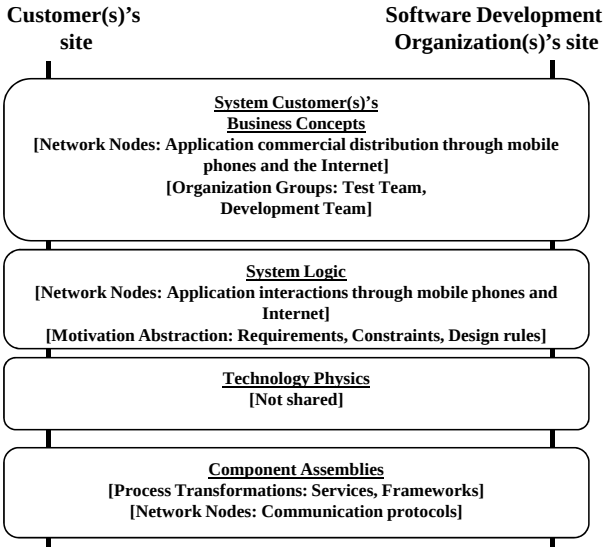

Figure. 5: Bridges between Customer Site(s) and Development Site(s) for Apple and Google.

therefore highly interesting for creating composite models which bridge the gap between customer/utilization site(s) and development organization site(s) in the Software Engineering Taxonomy.

Apple has created a way to easily install applications in run-time by structuring application code into bundles [17]. The bundle structure is part of the Apple framework. Apple shares the framework but in contrast to the open source community gives external developers no access to the Apple core business logic components.

The shared composite model pattern for bridging the utilization- and development sites gap for Apple and Google is visualized in, Figure 5. The innovative integration takes place in the Network Nodes abstraction in the Software Engineering Taxonomy for both AppStore and Google.

The AppStore describes the connections of internal- and external developers, customers, and the Apple organization through the Internet and through the mobile phone network.

The customers get a test/product strategy role when they indirectly drive both the internal and external development by downloading the internally and externally developed applications. The top-ten download list is visible for customers as well as developers on the AppStore web page.

Google's Ecosystem [18] describes the global locations and connections of Google's systems, services, advertisers, and customers over locations barriers world-wide. Google makes services available for external sites to use in their applications via standard protocols. Customers get a test/product strategy role when they test beta-versions of Google's products voluntarily.

The system design and deployment are crucial but not shared since they are descriptions of the core business logic. By considering what models in the Software Engineering Taxonomy are possible to share with external sites we can find new ways of bridging the gap between utilization and development which could create faster in-

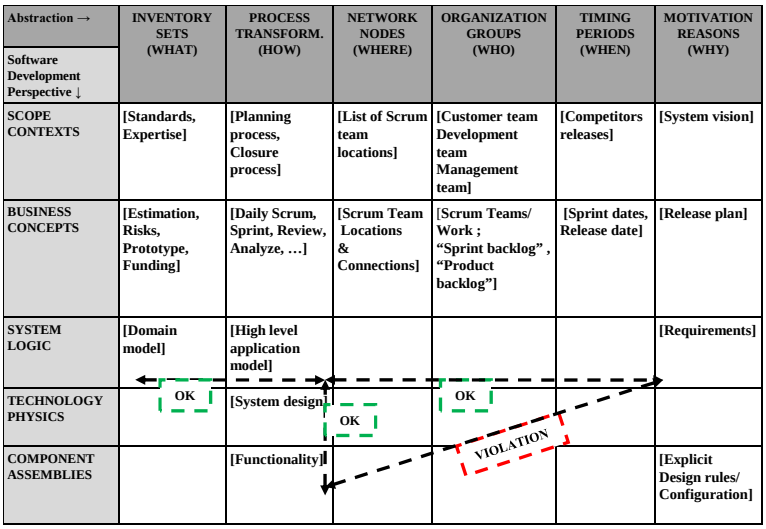

Figure. 6: Scrum reverse engineered into the Software Engineering Taxonomy

novation of new or enhanced products.

2.4 Scrum Composite Process Model

When reverse-engineering the Scrum process into our Software Engineering Taxonomy it becomes clear that the Scrum approach is rather extensive in the scope- and business perspective (Figure 6). To bridge the gap between customer/utilization site and developer site, the Scrum process [14] includes the customer and the sales organization as members of the development team. By integrating customer, management, release management, and development in a set of teams, all the teams' concerns are integrated in a dynamic team work product called "product backlog". Two important activities in the Scrum development process is the cost estimations and risk estimations.

The requirements and qualities are described in the Taxonomy cell with the Motivation Reasons abstraction and System Logic perspective. The code or program is described in the cell with the Process Transformations abstraction and Component Assemblies perspective. The composite process model, the Scrum development process as described in [14], takes a step from requirements to architectural design and domain modeling in the pregame phase. Would the Scrum composite process model have taken a direct step from requirements to code then the Zachman rule of consistency would have been violated.

An interesting approach would be to integrate explicitly formulated design rules [20], described in the taxonomy cell with the Motivation Reasons abstraction and the Technology Physics perspective. This would be an alternative way or additional step to take from requirements to code.

3 Conclusions and Future Work

The Software Engineering Taxonomy derived out of the Zachman Framework relies on two assumptions:

1. The classification framework, derived from the Zachman framework, describes the software development organization and the descriptions of the customer's enterprise that relates to the usage of the system

2. The classification framework, derived from the Zachman framework, is three-dimensional where site is the third dimension. The site might be the software development organization, external development organization or the customer's enterprise as long as the site has a part in the system usage or system development.

The classification of the IEEE SWEBOK artifacts uses only the software development organization's perspectives, not the customer perspective, resulting in the classification being two-dimensional. However, the three dimensions of the Software Engineering Taxonomy can be used to describe a software development organization that shares models with external software development sites or customer sites, e.g. Google, Apple and Open Source development as described in this report. The analysis of App-Store and Google showed that the taxonomy's Network Nodes abstraction and Organization Groups abstraction columns are important for sharing models with external development- and utilization sites for faster innovation of new products.

The reverse engineering of the Scrum process into our Software Engineering Taxonomy showed that all of the Scrum artifacts can be classified and that the focal point of the Scrum is on the Scope Contexts perspective and the Business Concepts perspective of the development organization. The descriptions of the System Logic perspective and the Technology Physics perspective are thin in the Scrum process.

The Software Engineering taxonomy can serve as a reasoning framework into which artifacts and results of software engineering theories, processes and case studies might be mapped for further analysis. The consistency rules of the Zachman framework are valid for the Software Engineering Taxonomy.

It remains to do a formal validation of the Software Engineering Taxonomy. The formal validation could be in the form of a more thorough collection of software engineering artifact and their classification. Further, an expert panel could judge the classification's correctness.

References

[1] DoD. Department of Defence Architecture Framework Working Group, DoD Architecture Framework, DoDAF, version 1.0. Department of Defence, 2003.

[2] TOG. *The Open Group Architecture Framework, version 8/9, 2002/6*. The Open Group,.

[3] J. A. Zachman. A Framework for Information Systems Architecture. *IBM Systems Journal*, 26(3):276–292, 1987.

[4] R. C. Thomas. A Practical Guide to Federal Enterprise Architecture,. www.gao.gov/bestpractices/bpeaguide.pdf, 2001. retrieved July 11th 2009.

[5] J. N. Martin. *An introduction to the Architectural Frameworks DODAF/MODAF/NAF*. Course given at the Royal Institute of Technology, Stockholm, Sweden, 2006.

[6] J. F. Sowa and J. A. Zachman. Extending and formalizing the framework for information systems architecture. *IBM System Journal*, 31:590–616, 1992.

[7] J. A. Zachman. The Zachman Framework and Observations on Methodologies. *Business Rules Journal*, 5(11), 2004.

[8] J. A. Zachman. *The Zachman Framework for Enterprise Architecture; A Primer for Enterprise Engineering and Manufacturing*. Zachman International, 2003.

[9] C. O'Rourke, N. Fishman, and W. Selkow. Enterprise Architecture, Using the Zachman Framework. *Thomson Course Technology*, 2003.

[10] P. Bourque, R. Dupuis, A. Abran, and J. W. Moore. Ieee recommended practice for architectural description of software-intensive systems, 2004.

[11] O. Mendes and A. Abran. Software Engineering Ontology: A Development Methodology. Technical report, University from Quebec in Montreal, 2004.

[12] P. Wongthongtham, E. Chang, and I. Sommerville. Software Engineering Ontology for Software Engineering Knowledge Management in Multi-site Software Development Environment. http://smi.stanford.edu/projects/protege/conference/2007/presentations, 2007.

[13] ASTM. ASTM Standard C33, "Specification for Concrete Aggregates", 2003.

[14] K. Schwaber. Scrum development process. Workshop Report: Business Object Design and Implementation. 10th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications. Addendum to the Proceedings. ACM/SIGPLAN OOPS Messenger 6(4), October 1995.

[15] D. B. Yoffie and M. Slind. Apple computer, 2006, 2007.

[16] P. Tsarchopoulos. Innovation lessons from apple. *The Economist*, 2007.

[17] Apple. About bundles, 2005.

[18] B. Iyer and T. H. Davenport. Reverse engineering google's innovation machine. *Harvard Business Review*, 2008.

[19] J. McGregor. The world's 50 most innovative companies. *Business Week*, 2008.

[20] M. J. LaMantia, Y. Cai, A. D. MacCormack, and J. Rusnak. Evolution analysis of large-scale software systems using design structure matrices and design rule theory. *Harvard Business School Working Knowledge*, 2007.