EEWC Doctoral Consortium Research Proposal Software User Acceptance Testing based on Enterprise Ontology principles

R.B.J. (René) Ceelen

Radboud University Nijmegen
Institute for Computing and Information Sciences Nijmegen
The Netherlands, r.ceelen@science.ru.nl

November 10, 2020

1 Introduction

Almost everyone experiences the enormous influence of information technology (IT) on the arrangement of organizations and society. The reliability of IT is highly critical for organizations, and the awareness has grown that IT is a compelling means to achieve higher performance. But lots of IT initiatives fail. Research over a lengthy period did not prove any positive relationship between IT investments and measurable improvements in enterprise performance. Maizlish and Handler[1] mentioned that the root cause is failure to define success criteria at the start of the project. Hoogervorst[2] said that a core reason for strategic failures is the lack of coherence and consistency among the various components of an enterprise. The higher the degree of fit among the various components of the enterprise, the more effectively the enterprise is likely to operate. Standish Group[3] is reporting that the main cause of failure in IT projects is the lack of user involvement. The essence of IT development with its complexity, conformity, changeability, and invisibility[4] makes introducing defects into the software construction a fact. Boehm[5] concluded that the main economic problem from the perspective of software testing is the exponential growth of costs by finding software issues near to the acceptance phase. Summarizing the reasons mentioned above, designing enterprise unity and integration is fundamental and does not come 'incidentally.' When designing the enterprise, the software engineering aspects and its software acceptance must also be designed using the same models and engineering aspects. Our claim in this research is that when the whole system from enterprise engineering, software engineering, and acceptance engineering is using the same 'language' less IT initiatives fail and will be used for the right purpose.

2 Background

2.1 Software Testing

In the early 1970s, the level of professionalism associated with software testing increased significantly by publishing about testing, international conferences, and the development of national standards. The first article about software testing was published in 1950[6]. The first formal conference on software testing was organized by Bill Hetzel and held at the University of North Carolina in June 1972, and the first software testing book was published in 1973. This book is the testing analog of a software architecture specification. Its purpose is to focus attention on the overall organization of the test set and the linkage of test set elements to software requirements and design components.

Software testing is the process of evaluating a system or component during or at the end of the development or implementation process to determine whether it satisfies specified requirements.

2.2 Enterprise Engineering

Enterprise engineering is the whole body of knowledge regarding the development, implementation, and operational use of enterprises, as well as its practical application in engineering projects[7]. An enterprise can be interpreted as a company, organization, business, or governmental institution which is highly complex and highly organized. One of its characteristics is that management is often only interested in what the enterprise should realize, not in how that should be accomplished. Enterprise engineering intends to address the design perspective in a formal, methodological way[7][2] to get answers in how to achieve the what. Enterprise Engineering is built on the following three foundational pillars: Enterprise Ontology, Enterprise Architecture, and Enterprise Governance. Each of these fields has the following defined goal:

1. *Intellectual manageability* - To bring organizational changes, one needs to have insight and overview. This implies a well-devised systematic reduction of complexity (Enterprise Ontology).

2. *Organizational concinnity* - For an enterprise to be a coherent and consistent whole, its parts must be arranged in a skillful a harmonious way. This implies a well-devised design (Enterprise Architecture).

3. *Social devotion* - EE takes a human-centered view on enterprises. This implies a well-devised distribution of authority and responsibility (Enterprise Governance) [7][2].

2.3 Enterprise Ontology

Coping with current and future challenges in organizations, a conceptual model of the enterprise is needed that is coherent, comprehensive, consistent, and concise, and that only shows the essence of the enterprise, abstracted from irrelevant details [8]. Thus allowing to achieve intellectual manageability, which is one of the general goals of Enterprise Engineering. By *coherent* we mean that the distinguished aspect models constitute a logical and truly integral whole. By *comprehensive* we mean that all relevant issues are covered, that the whole is complete. By *consistent* we mean that the aspect models are free from contradictions or irregularities. By *concise* we mean that no redundant matters are contained in it, that the whole is compact and succinct. And last that this conceptual model is *essential*, showing only the essence of the enterprise abstracting from all realization and implementation issues. This total conceptual model is called an *ontological model* [8]. The goal is to understand the essence of the construction and operation of systems, more specifically, of enterprises.

2.4 Generic System Development Process

As an additional refinement in the field of Enterprise Ontology towards the function and construction of systems we use the Generic System Development Process (GSDP) [9]. According to Dietz [9], the development of a single homogeneous system of any type can be understood as an instance of the GSDP. The design concepts and processes are shown in figure 1.

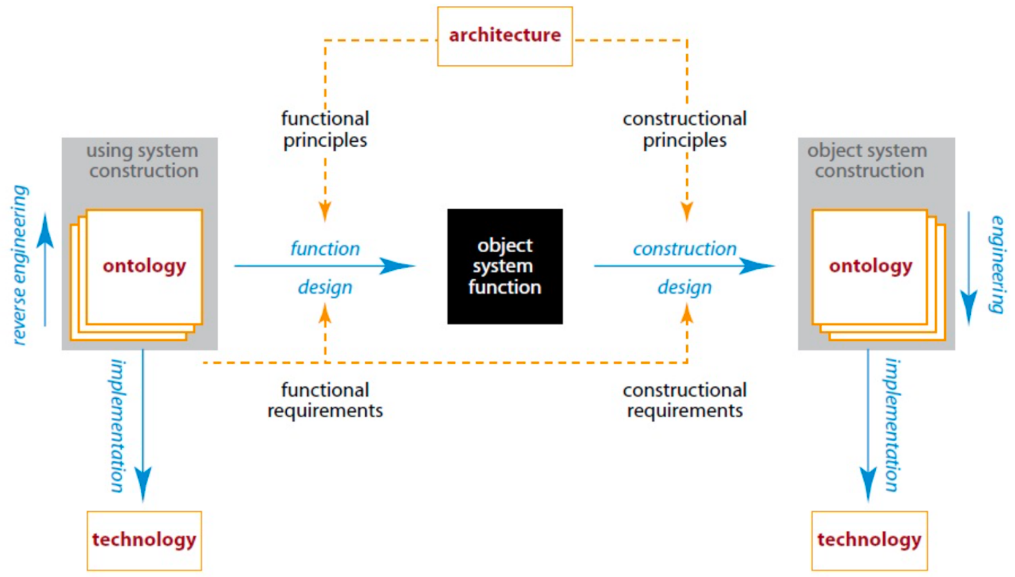

Figure 1: Generalized System Development Process [9]

Figure 1 shows the object system (OS) understood as the system to be developed, and the using system (US) understood as the system that is going to use the services (functionality) of the OS. The development of an OS is considered to consist of a design (1), engineering (2), and an implementation phase (3).

1. *The design phase* comprises a function design and construction design. Function design, the first step in the plan of the OS, starts from the construction of the US and ends with the function of the OS. Function design delivers the requirements of the OS, so a black-box model of the OS. This black-box model clarifies the behavior of the OS in terms of (functional) relationships between input and output of the OS. The function model of the OS does not contain any information about the construction of the OS. Construction design, the second step in the design of the OS, start with the specified function of the OS, and it ends with the construction model of the OS. Construction design bridges the mental gap between function and construction, which means establishing a correspondence between systems of different categories: the category of the US (where the purpose of the OS is defined) and the category of the OS. Construction design delivers an ontology, the highest level white-box model, of the OS. This white-box model clarifies the internal construction and operation of the OS in terms of collaboration between its elements to deliver products to its environment. By an ontological model of a system, we understand a model of its construction, which is entirely independent of how it will be implemented using some underlying technological infrastructure.

2. *The engineering of a system* is the process in which several white-box models are produced, such that every model is fully derivable from the previous one and the available specifications. Engineering starts from the ontological model, creates a set of subsequently more detailed white-box models, and ends with the implementation model.

3. *By implementation* is understood the assignment of technological means to the elements in the implementation model, so that the system can be put into operation.

3 User Acceptance Testing (UAT)

3.1 Software Testing

As said software testing is the process of evaluating a system or component during or at the end of the development or implementation process to determine whether it satisfies specified requirements. IEEE[10] defines "verification" and "validation" as follows:

- Verification: The process of determining whether or not the product of a given phase of the software development cycle fulfill the requirements established during the previous phase. ("Am I building the product right?")
- Validation: The process of evaluating software at the end of the software development process to ensure compliance with software requirements. ("Am I building the right product?")

Test granularity[11] refers to the fineness of a test's focus. A fine-grained test case allows the tester to check low-level details, often internal to the system. A coarse-grained test case provides the tester with information about general system behavior. In figure2 the test granularity spectrum and ideal testers are presented.

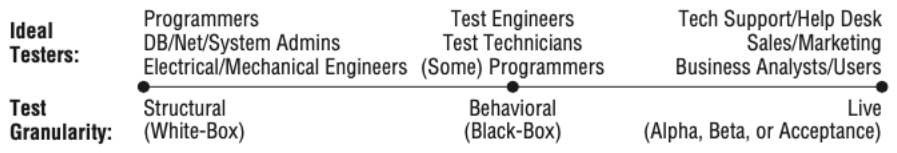

Figure 2: The test granularity spectrum and ideal testers [11]

There are two main classes of software testing, black-box testing, and white-box testing. The essential difference between the two categories is that black-box testing ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions.

White-box testing takes into account the internal mechanism of a system or component[10] and the purpose is to find bugs in low-level structural elements such as lines of code, database schemas, chips, subassemblies, and interfaces. The tester bases structural tests on *how* a system operates.

Black-box testing focuses on the behavior of the system or component and the purpose is to find bugs in high-level operations, such as major features, operational profiles, and customer scenarios. Testers can create black-box functional tests based on *what* a system should do. Behavioral testing involves a detailed understanding of the application domain, the business problem that the system solves, and the mission the system serves.

Live tests are also part of black-box testing and involve customers, content experts, early adopters, and other end users. The purpose is to do the final test, before the system or component goes into a production environment.

3.2 User Acceptance Testing

User Acceptance Testing (UAT) is black-box testing consisting of comparing a software system to its initial requirements and the current needs of its end-users or, in the case of a contracted program, to the original contract [12]. It is a crucial step that decides the faith in an information system. The outcome provides a quality attribute for the customers (or end-users) to determine whether to accept or reject the software product. It is, therefore, categorically different from other types of testing where the intent is principally to reveal errors [13].

The primary guide to software acceptance testing is the system requirements document, and the primary focus is on usability and reliability and behavior of the system from the perspective of the end-user [12]. There is currently no industry-wide standard for software acceptance testing. Therefore, in practice, most testing methods are based on best practices. The widely used methods, like ISTQB, do not adopt the Enterprise Engineering perspective, and thus do not use ontological models as the starting point for acceptance testing.

3.3 Automated Testing

The software development environment has shifted significantly in the recent years with the introduction of Agile and Devops. This new focus on higher speed and the shift to smaller, more frequent releases, has a great influence on testing. In the last years test automation has become part of the testing fundaments.

Test automation or 'automatic testing' is an automated test process where predefined and set values are controlled by a 'robot' in a particular information system or subsystem.

The testrobot has many advantages as long as the instructions and data sets given to the robot are correct. This robot also comes with its disadvantages. First, all instructions that should be given to the robot should be concretely identified and 'explained' to that robot. Secondly, the robot can make high-speed comparisons, but during manual testing the scope beyond test execution is broader. The human perspective is much more aware and intuitive and therefore 'automatically' looks beyond the test case with its implicit observation. Third, it's difficult to move the acceptance process to a robot. The definition of acceptance tests is to establish that the organization is ready for the use of the software. Therefore it is crucial to test manually, with all its intuitions, implicit observations and determination if an organization is actually ready to use the software within a healthy balance with automated testing. The "fit for purpose or fit for business" test and decision must be done by humans.

4. Research Objectives and Questions

As mentioned, lots of IT initiatives fail. Research over a lengthy period did not prove any positive relationship between IT investments and measurable improvements in enterprise performance. The reason can be various, but user involvement has a significant impact on the success and adaption of a new information system. And in particular, user involvement plays an essential role in User Acceptance Testing (UAT), which is black-box testing and a crucial step that decides the faith in a system or component [12]. As said, there is no industry-wide standard for UAT. Therefore, in practice, most testing methods are based on best practices, or even worse, MsExcel is used for this kind of testing.

Our main objective is to develop a method (way of thinking, way of modelling, way of working, way of organizing and way of supporting) that enable practitioners to design their UAT approach in a structured manner. This method will reduce organizational complexity and perform more efficient and more effective user acceptance test.

The focus of this dissertation is in the first place designing a method for UAT using Enterprise Ontology (EO) to achieve this objective. This leads to our central research question:

How can we construct UAT using Enterprise Ontology to create a UAT method that is structured, repetitive, adequate, and that will support an effective and efficient user acceptance test?

To answer this central question, we have formulated the following sub-questions:

1. What are the requirements for an adequate UAT?
2. What aspects of EO can be used to define a robust UAT?
3. How to create a UAT method that is structured, repetitive, and adequate?

5. Research Approach

This design science research approach seems most suited to our research. The design science paradigm has its roots in engineering and the sciences of the artificial. It is fundamentally a problem-solving paradigm. It seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, management, and use of information systems can be effectively and efficiently accomplished [14]. Figure 3 shows the process model of the Design Science Research Methodology (DSRM) [15].

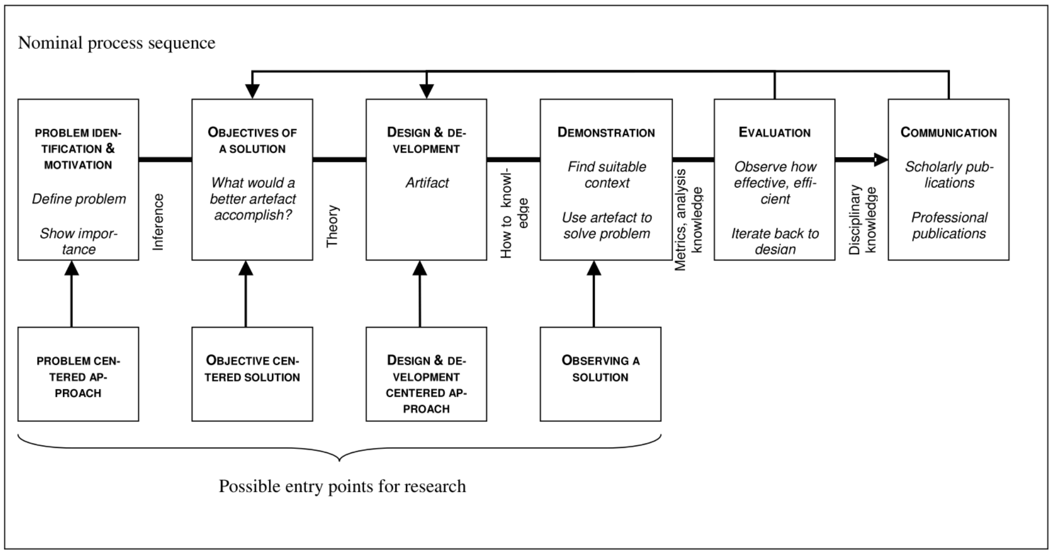
Nominal process sequence
PROBLEM IDENTIFICATION & MOTIVATION
Define problem
Show importance
Inference
OBJECTIVES OF A SOLUTION
What would a better artefact accomplish?
Theory
DESIGN & DEVELOPMENT
Artifact
How to knowledge
DEMONSTRATION
Find suitable context
Use artefact to solve problem
Metrics, analysis knowledge
EVALUATION
Observe how effective, efficient
Iterate back to design
Disciplinary knowledge
COMMUNICATION
Scholarly publications
Professional publications
PROBLEM CENTERED APPROACH
OBJECTIVE CENTERED SOLUTION
DESIGN & DEVELOPMENT CENTERED APPROACH
OBSERVING A SOLUTION
Possible entry points for research

Figure 3: shows the process model of the Design Science Research Methodology (DSRM) [15]

The entry point for this research is the problem-centered approach. Our theoretical basis for designing the UAT artifact is the Ψ theory and follow all phases from defining objectives until the evaluation of the observations and communication phase.

Problem identification and motivation

Almost everyone experiences the enormous influence of information technology (IT) on the arrangement of organizations and society. The reliability of IT is highly critical for organizations, and the awareness has grown that IT is a compelling means to achieve higher performance. But lots of IT initiatives fail. When designing the enterprise, the software engineering aspects and its software acceptance must also be designed using the same models and engineering aspects. Our claim in this research is that when the whole system from enterprise engineering, software engineering, and acceptance engineering is using the same 'language' less IT initiatives fail and will be used for the right purpose.

The objective of the solution

The objective is to develop a UAT artifact that includes methods, methodologies, and modeling techniques that can be used to design an adequate UAT approach.

Design and development

The main artifact is UAT. In this step, they are determined, and prototypes are developed.

Demonstration

After developing the proof of concept prototypes of the artifact, we will test the artifact internally with experts and, where possible and appropriate, on client cases.

Evaluation

In the evaluation step, we verify in case studies whether the UAT artifact leads to a more effective UAT approach with a higher adoption by its end-users.

Communication

In the last phase, we will publish articles with the results of the main artifact and related case studies.

6. Current status and planning

a. Ph.D. Project

This dissertation will be supervised by prof. dr. H.A. (Erik) Proper, prof. dr. J.B.F. (Hans) Mulder and prof. dr. J. (Jan) Dietz.

b. Planning

The Ph.D. research is planned for three years, starting in April 2020 and ending in April 2023. The years 2020 and 2021 are dedicated to build the artifact and perform two case studies. The results will be published in academic articles. In the last year, the focus will be on gathering all the results and writing the final thesis.

c. Published

In the book Enterprise Ontology – a human-centric approach to understanding the essence of organization [8] an article is published "DEMO enhanced software testing" in Chapter 20, paragraph 5.

References

[1] B. Maizlish and R. Handler, *IT Portfolio Management Step-by-Step*. Wiley, Hoboken, 2005.

[2] J. A. P. Hoogervorst, *Enterprise Governance and Enterprise Engineering*. Springer, Berlin, 2009.

[3] Standish Group, "Chaos reports," 1994.

[4] E. Brooks, "No silver bullet: Essence and Accident in Software Engineering," *IEEE Computer*, vol. 20, no. 4, pp. 10–19, 1987.

[5] B. Boehm and C. Englewood, "Software Engineering Economics," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 1, pp. 4–21, 1984.

[6] A. M. Turing, "Computing machinery and intelligence," *Machine Intelligence: Perspectives on the Computational Model*, no. 1950, pp. 1–28, 2012, doi: 10.1093/mind/lix.236.433.

[7] J. L. G. Dietz, *Enterprise Ontology - Theory and Methodology*. Springer, Berlin, 2006.

[8] J. L. G. Dietz and H. B. F. Mulder, *Enterprise Ontology – a human-centric approach to understanding the essence of organisation*. Springer, Berlin, 2020.

[9] J. L. G. Dietz, *Architecture: Building Strategy Into Design*. Sapio, 2008.

[10] IEEE, *Standard Glossary of Software Engineering Terminology*, vol. 121990. ieee1990, 1990.

[11] R. Black, *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing*. 2009.

[12] C. Kaner and J. Bach, "Paradigms of black-box software testing," *16th International Conference and Exposition on Testing Computer Software, Washington, DC*, 1999.

[13] P. Hsia and D. Kung, "Software requirements and acceptance testing," *Annals of Software Engineering*, pp. 291–317, 1997.

[14] A. R. Hevner, "Design science 97," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004, doi: 10.1007/BF01205282.

[15] K. Peffers, T. Tuunanen, and C. Gengler, "The design science research process: a model for producing and presenting information systems research," *Journal of Management Information Systems*, no. May 2014, pp. 83–106, 2006.