UbiCheck: An Approach to Support Requirements Definition in the Ubicomp Domain

Rodrigo O. Spínola, Felipe C. do R. Pinto and Guilherme H. Travassos
PESC-COPPE/UFRJ
Cidade Universitária, Centro de Tecnologia, Bloco H,
Sala 319, Rio de Janeiro, RJ, Brazil
+55 21 2562-8672
{ros, felipecrp, ght} @cos.ufrj.br

ABSTRACT
Ubiquitous computing brings a set of characteristics that are not commonly found in conventional software projects. One of the consequences is an increase in the software development complexity. Additionally, traditional software engineering techniques are not usually adequate to support the development of this system category as they do not cover specific characteristics of this domain. Therefore, this work presents *UbiCheck* - an approach to support requirements definition in the ubicomp domain, including the results of an initial observational study that indicated such approach can be feasible.

Categories and Subject Descriptors
D2. Software Engineering: Requirements/Specifications.

General Terms
Documentation, Experimentation, and Theory.

Keywords
Requirement Engineering, Requirement Definition, Ubiquitous Computing, Empirical Study.

1. INTRODUCTION
The insertion of defects throughout the development of conventional software projects is a recurring issue and it should not be different when dealing with ubiquitous software projects. The effort spent by software organizations with rework ranges on average from 40% to 50% of the total project development effort [4]. Wheeler *et al.* [5] found it is possible to verify that rework tends to grow as the development progresses. One of the main reasons for this is the increase in the effort to correct defects in the final activities of the development process as a result of defects inserted and propagated from initial development activities such as the requirements specification.

In this scenario, requirements definition represents a crucial phase in software development. Its main goal is to develop requirements specification that is complete, consistent and non-ambiguous, becoming the basis for an agreement between all stakeholders involved in the software project [1]. Several techniques and methods have been proposed to deal with these issues [2]. However, most of them support the development of conventional software projects which can reduce its efficacy and efficiency when working with specific application domains [3].

Ubiquitous software projects have a set of characteristics associated to the domain of ubiquity that cannot be commonly found in conventional software projects [13]: service omnipresence, context sensitivity, adaptable behavior, experience capture, device heterogeneity, universal usability, fault tolerance, scalability, quality of service, and privacy and trust. Therefore, one of the main ubiquitous computing (ubicomp) challenges goals is to understand how to build non-invasive computing services and make them available in the environment for the users [6][7].

In this sense, Ducatel *et al*. [8], Niemela and Latvakoski [9] and Sakamura [10] reported that ubiquitous software projects can bring new challenges not addressed by current software engineering techniques. For instance, in searching for new signs of how to deal with the building of ubiquitous software projects 11 works were found in the technical literature related to approaches that support the definition of ubiquity requirements (requirements associated with the ubiquity characteristics) [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [27]. These works can identify in 8 approaches (Table 1) related to ubiquity characteristics.

Although requirements definition may be considered a fundamental software project development activity due its impact on quality and costs, the support to this activity is still limited when we consider ubicomp domain. For instance, analyzing the works on Table 1, some issues can be observed: (1) the approaches deals with a subset of the ubiquity characteristics (adaptable behavior, context sensitivity, fault tolerance, and privacy and trust); (2) only three approaches define a set of guidelines about requirements definition; (3) the identified approaches do not explore the ubicomp domain knowledge to support their execution ; (4) only one work [27] describes a process definition and a technique detailing that can be used to support the execution of its activities.

SAC'10, March 22-26, 2010, Sierre, Switzerland.

Thus, to improve the quality of ubiquitous software and reduce costs associated to rework, we believe that it is fundamental to support the software engineer on the ubiquity requirements specification activity. In this context, this paper presents *UbiCheck* - an approach based on ubicomp domain knowledge to support the definition of ubiquity requirements in ubiquitous software projects. It is also summarized the evaluation of *UbiCheck* and the resulting improvements.

Besides this introduction, this paper has 4 other sections. Section 2 presents *UbiCheck*. Next, Section 3 discusses the initial evaluation of the approach through an observational study. Section 4 presents the improvements in the *UbiCheck*. Finally, Section 5 provides the final considerations to this work.

Table 1. Related Works and Ubiquity Characteristics

Ubiquity Characteristics	Identified Approaches							
	[15]	[16]	[17] [18] [19]	[20]	[21]	[22]	[23] [24]	[27]
Service Omnipresence								
Context Sensitivity		■	■				■	■
Adaptable Behavior		■	■					■
Experience Capture								
Device Heterogeneity								
Universal Usability								
Fault Tolerance						■		
Scalability								
Quality of Service								
Privacy and Trust						■		

2. UBICHECK

UbiCheck is a checklist-based approach to support requirements definition in the ubicomp domain. According to Laitenberger *et al*. [25], checklists can be used with the purpose of guiding software engineers throughout the execution of a task. Thus, it is expected that the using of *UbiCheck* would help software engineers during the requirements definition activities increasing the efficiency and effectiveness of such activity by reducing omission defects and execution time.

UbiCheck composes a framework to support the definition and verification of ubiquity requirements for ubiquitous software projects [14]. The framework is based on a body of knowledge regarding ubicomp [14]. This body of knowledge comprises two elements: (1) **characteristics,** identifying the main concerns to be dealt when working with ubiquitous software projects and; (2) **factors,** defining how each characteristic can be covered in terms of functionalities.

This body of knowledge composes the framework's conceptual core. In general, the framework supports the following stages: (1) Body of knowledge configuration; (2) Software projects characterization; (3) Software project's ubiquity requirements identification and specification, and; (4) Revision of the defined software projects' ubiquity requirements.

Figure 1 represents the stages and activities of *UbiCheck*. As can be observed, the approach is composed by 4 sequential activities grouped in 2 main steps regarding ubiquity requirements: (1) Configuration and (2) Definition ubiquity requirements. The step 1 is usually executed in the initial definition of the *Ubicheck's* body of knowledge. It also can be used to allow the improvement of this knowledge for a specific organization. On the other hand,

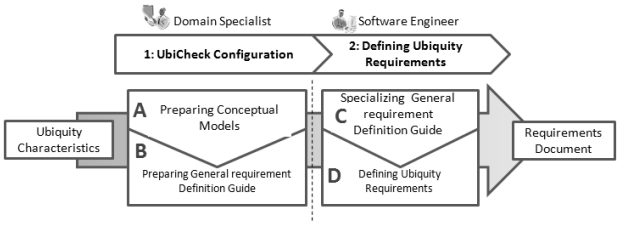

Figure 1. *UbiCheck* overview.

the step 2 is usually performed once for each ubiquitous software project.

2.1 Step 1: *UbiCheck* Configuration

The goal of this step is to configure the body of knowledge from the conceptual core (ubiquity characteristics and their respective factors) to support the requirements definition activity. From this conceptual core, it is possible to define: (1) **Conceptual Models:** represent the main information regarding each ubiquity characteristic considering their attributes, relationships, and services; (2) **General requirement Definition Guide (GDG)**: checklist based on ubicomp domain to support the identification of ubiquity requirements in software projects.

To illustrate these artifacts generation process, we will use the conceptual core fragment about the characteristic *experience capture* presented below:

Information Capture
Capturing system user interaction information
Capturing experiences automatically
Information Reasoning
Analyzing interaction patterns
Analyzing relationship among public and private experiences
Information Management
Store captured information
Represent users' activities
Represent users' needs and preferences

2.1.1 Activity A: Preparing Conceptual Models

This activity consists of generating, for each ubiquity characteristic, a model representing the main information regarding ubiquity characteristics according to their respective factors.

The ubiquity conceptual models have been developed as UML class diagrams with the purpose of representing ubiquity characteristics' concepts and relationships in a graphical and structured format. Additionally, aiming at consistency, the models are generated based on a meta-model defining the types of elements and valid relationships used to represent the ubiquity characteristics and factors. A model fragment for the *experience capture* characteristic is presented in Figure 2. We believe such representation can increase the information understandability related to different ubiquity characteristics by software engineers.

Despite the fact that those models can increase the understandability of the conceptual core, they lack a clear vision about the importance of each represented element. Therefore, besides a graphical view, each characteristic model is also represented by a hierarchical structure (tree of elements). The criteria based on coupling between the models' elements to construct the hierarchy were adapted from [26]. For instance, considering the model fragment showed in Figure 2, we can

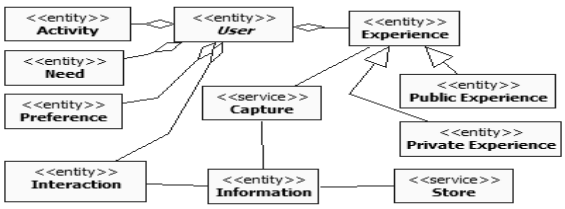

Figure 2. Fragment of Experience Capture Conceptual Model.

define the hierarchical representation presented in Figure 3. This set of information will be used to support the creation of the checklists in the Activity B: Preparing GDG.

2.1.2 Activity B: Preparing GDG

In this activity, the General requirement Definition Guide is created based on the tree of elements defined in the Activity A. Basically, a question is created for each element of this tree aiming at to support the requirements definition.

Once the elements represented in the elements tree are considered important in ubiquitous software projects, the goal of those questions is to guide the software engineer to insert the model's elements into the requirements specification document.

As a result, we have a GDG covering all ubiquity characteristics and their respective factors evaluated as required for the software project. Figure 3 shows the GDG questions preparation from a branch of a tree of elements for the characteristic *capture of experience*.

It is important to highlight that the conceptual core transformation into conceptual model elements, organizes it considering the trees of elements, and the GDG definition associated with the trees of elements allows us to create traces amongst the different transformations. This scenario allows a consistent evolution of the mapped knowledge and the body of knowledge specialization for specific projects needs. Moreover, once the body of knowledge has been configured, it will be only changed whether new factors or ubiquity characteristics can be identified in the technical literature. This feature reduces the need to have an ubicomp domain specialist to support the development activities in the software project.

Finally, it is important to observe that Step 1 allows the incremental improvement of techniques to define ubiquity requirements through ubicomp domain knowledge maintenance and the artifacts that support the software engineer activities.

After Ubicomp domain knowledge configuration, we can go one step forward.

2.2 Step 2: Defining Ubiquity Requirements

The goal of this Step is to calibrate the GDG to use it in a specific software project. The specialization process and how to use the specialized guide to support the requirements definition are explained in the following subsections.

Tree	Question
User	What are the relevant users?
-- **Interaction**	What are the relevant users' interaction?
-- -- **Information**	What are the relevant information about users' interaction?
-- -- -- **Capture**	How to capture information about user interaction?
-- -- -- **Store**	How to store information about user interaction?
-- **Activity**	How the user activities are considered?

Figure 3. Elements Tree Fragment for Experience Capture Characteristic.

2.2.1 Activity C: Specializing GDG

In this activity, the General requirement Definition Guide (GDC) is transformed into a **S**pecialized requirement **D**efinition **G**uide (**SDG**). The SDG construction process is based on a checklist to characterize ubiquitous software projects defined by Spínola *et al*. [13]. The characterization checklist identifies the relevant ubiquity characteristics and factors for a particular software project (a ubiquitous software project does not need to explore all ubiquity characteristics). Next, the traces among the characterization checklist questions and GDG questions are used to transform the GDG into SDG.

2.2.2 Activity D: Defining Ubiquity Requirements

In this activity, the software engineer uses the SDG to support the ubiquity requirements elicitation and their specification.

As described in Section 2.1.2, SDG is checklist based having a series of questions indicating what type of ubiquity information the software engineer should be aware during the requirements elicitation activity. The SDG does not indicate how to proceed to execute the requirements elicitation (for instance: interviewing, questionnaire, or brainstorming).

Figure 4 illustrates a fragment of a SDG for the *experience capture* characteristic. This specialized guide comprises: (1) Instructions about its usage; (2) Ubiquity characteristics present on the current software project; (3) Relevant information about ubicomp as defined on Activity A – Configure *UbiCheck*; (4) Questions about relevant ubiquity information that should be identified by the software engineer.

3. OBSERVATIONAL STUDY

The goal of this study was **to analyze** *UbiCheck*, **with the purpose of** characterize, **with respect to** its applicability, **into the context of** ubiquitous software projects **from the point of view of** software engineering students.

In this study, applicability relates to verify if software engineers could understand and use *UbiCheck* in a ubiquitous software project. For this, one of the two observable behaviors is expected: **B1**: *UbiCheck* cannot be used by software engineers to support ubiquitous software development; **B2**: *UbiCheck* can be used by software engineers to support ubiquitous software development.

3.1 Instrumentation Planning

As instrumentation, the following forms were elaborated:

(1) Consent and Subject characterization (personal data, academic degree, and experience level on software projects);

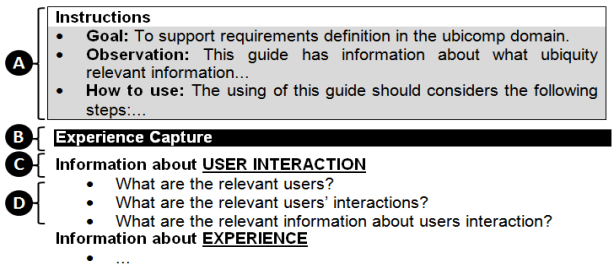

Figure 4. Fragment of a SDG.

(2) *Ad hoc* Task Execution: this form intends to collect information (effort in time scale) about the requirements definition activity execution without *UbiCheck* support;

(3) *UbiCheck* Task Execution: this form intends to collect information (effort in time scale) about the requirements definition activity execution with *UbiCheck* support;

(4) *UbiCheck* Evaluation: it aims at to identify the potential and benefits in using the proposed approach;

(5) Problems and Issues: this form's goal is to identify the problems and/or issues associated with *UbiCheck* using;

Besides, two ubiquity scenarios complement this study instrumentation: **(Scenario 1)** the tracking of inventory items while being transported through the university campus; **(Scenario 2)** the tracking of loaned bikes in a university campus.

3.2 Execution

The population of this study was represented by Master and PhD students in the software engineering area. A set of 8 subjects from an Object Oriented Software Engineering course took part. They were grouped in 3 teams (A-3, B-3, and C-2 subjects). Each team received the two ubiquity scenarios to increment the requirements specification of an Inventory Management System. The requirements definition of each scenario was made sequentially according to the distribution presented on Table 2.

Table 2. Scenario Distribution and Effort to define the ubiquity requirements.

	1st Iteration (*ad hoc*)		2nd Iteration (with *UbiCheck*)	
Team A	Scenario 1	300 min	Scenario 2	240 min
Team B	Scenario 1	240 min	Scenario 2	120 min
Team C	Scenario 2	180 min	Scenario 1	180 min

During the study execution, the subjects signed the consent form and filled in the forms that allowed us to capture the qualitative data about *UbiCheck* applicability.

3.3 Results and Limitations

The empirical study execution allowed us to observe some positive features and drawbacks of the proposed approach. The positive features are:

- The subjects reported that *UbiCheck* helps in the ubiquity requirements definition activity. They said the questions in *UbiCheck* have led them to think about important issues in ubicomp domain that they normally do not capture in the requirements document.
- *UbiCheck* allowed for this group an effort (in time) reduction on average of 23,3% to define ubiquity requirements (see Table 2).

On the other side, the subjects also reported some issues (improvement opportunities) of using *UbiCheck*:

- Ubicomp domain represents a knowledge intensive area but the proposed approach did not provide enough support for some terms used in the checklist. This caused a misunderstanding of some concepts.
- It was not clear the relationship between the expected answers of some questions in *UbiCheck* and the sections of a requirements specification document. Thus, the software engineers faced some issues on defining specifics ubiquity requirements.

In general, the results suggested that *Ubicheck* could be possible to support the behavior B2. However, some threats to validity were observed considering this particular study:

- The study was executed with different academic degree , experience and knowledge levels graduate students;
- The two used scenarios were bit similar. This fact may have influenced the reduction of the effort involved in the 2nd iteration of the study;
- Population's size does not allow a satisfactory statistical treatment.

Therefore, it could be interesting to replicate this study to obtain more indication about the observed behavior increasing the confidence in results.

4. UBICHECK IMPROVEMENTS

The obtained results allowed us to evolve *UbiCheck* based on the improvement opportunities reported in prior section. Thus, the following enhancements are present on *UbiCheck 2.0*:

- A glossary of terms has been created and attached to the GDG/SDG;
- For each question of the GDG/SDG, it was clearly defined what is expected to be answered;
- For each question of the GDG/SDG, it was defined how the related concept should be described in terms of the requirements specification document (specification item). For this, a set of concepts usually present in a requirements specification document (for instance, functional requirement, use case description, actor, and business rule) was defined and associated with the corresponding GDG/SDG questions.

Figure 5 illustrates a SDG new version fragment for the *experience capture* characteristic. Besides the elements previously shown in Figure 4 (A, B, C, and D), this new guide version includes: (E) Link to glossary of terms; (F) Additional information about answers expectation, and; (G) Suggestion about where a concept should be described into the requirements specification document.

5. CONCLUSION

This paper presented *UbiCheck*, a checklist based approach to support the requirements definition in the ubicomp domain. Besides, the importance of a body of knowledge regarding ubicomp to support ubiquitous software development was also discussed. *Ubicheck* is supported by such a body of knowledge.

The development of *Ubicheck* follows an experimental based methodology. In this paper, the initial study to evaluate the use of *Ubicheck* has been summarized. These initial results were useful to indicate a possible *Ubicheck´*s feasibility and to allow its evolution. However, this study should be replicated aiming at to confirm that *Ubicheck* can support software engineers on specifying ubiquity requirements.

Currently, we are working on the planning of a new study to assess the actual benefits of *UbiCheck* associated to the reduction

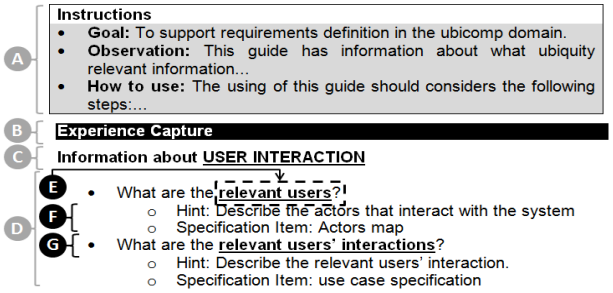

Figure 5. Fragment of a SDG.

of omission defects inserted during the definition of ubiquity requirements in ubiquitous software projects development.

6. ACKNOWLEDGMENTS

Our thanks to CNPq (Grant 75459/2007-5), CAPES, and FAPERJ for the financial support.

7. REFERENCES

[1] IEEE, 1998, "IEEE Recommended Practice for Software Requirements Specifications - Description, Standard 830", IEEE Press.

[2] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in ICSE '00: Proceedings of the Conference on The Future of Software Engineering. New York, NY, USA: ACM Press, 2000, pp. 35-46.

[3] Oliveira, K. M., Zlot, F., Rocha, A.R. C., Travassos, G. H., Silva, C. G. M., Menezes, C. S., (2004). Domain Oriented Software Development Environment. Journal Of Systems And Software, v. 72, n. 2, p. 145-161.

[4] Boehm, B. W., Basili, V.R., 2001, "Software Defect Reduction Top 10 List.". IEEE Computer 34: 135-137.

[5] Wheeler, D.A., Brykezynski, B., Meeson, R.N., 1996, Software Inspections: An Industry Best Practice, IEEE.

[6] Weiser, M. "The Computer for the 21st Century". Scientific American 1991, pp. 94-104.

[7] Abowd, G., 1999. Software engineering issues for ubiquitous computing. Proceedings of the 21st International Conference on Software Engineering.

[8] Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., Burgelman, J.-C. (2003). "Ambient Intelligence: From Vision to Reality". IST Advisory Group Draft Rep.

[9] Niemela, E., Latvakoski, J., 2004. Survey of requirements and solutions for ubiquitous software, Proc. of the 3rd Int. Conf. on Mobile and Ubiquitous Multimedia, p.71-78, College Park, Maryland.

[10] Sakamura, K. (2006) "Challenges in the Age of Ubiquitous Computing: A Case Study of T-Engine, An Open Development Platform for Embedded Systems". ICSE 2006, China. Pages:713-720.

[11] Russel, D.M., Streitz, N.A., Winograd, T., 2005. Building disappearing computers. Com. of the ACM, pp. 42 – 48.

[12] Kitchenham, B., Dybå, T., Jorgensen, M. (2004) "Evidence-based Software Engineering", Proceedings of the 26th ICSE, Scotland, UK, v.10 n.4, pp.437-466.

[13] Spínola, R.O., Silva, J.L.M., Travassos, G.H. (2007) "Checklist to Characterize Ubiquitous Software Projects". In: XXI SBES – Brazilian Symposium on Software Engineering, João Pessoa, Brazil.

[14] Spínola, R.O., Pinto, F.C.R., Travassos, G.H. (2008) "Supporting Requirements Definition and Quality Assurance in Ubiquitous Software Project". In: 3rd ISOLA, Greece.

[15] Jorgensen, J. B., and Bossen, C. Requirements Engineering for a pervasive health care system. Proc. of the 11th IEEE Int. Requirements Engineering Conference, 2003, 55-64.

[16] Goldsby, H. and Cheng, B. H. C. Goal-Oriented Modeling of Requirements Engineering for Dynamically Adaptive System, 14th IEEE Int. Requirements Engineering Conference, 2006, 345-346.

[17] Hong, D.; Chiu, D. K. W., and Shen, V. Y. Requirements elicitation for the design of context-aware applications in a ubiquitous environment. Proc. of the 7th International Conference on Electronic Commerce, ACM, 2005, 590-596.

[18] Chiu, D. K. W.; Hong, D.; Cheung, S. C., and Kafeza, E. Adapting Ubiquitous Enterprise Services with Context and Views. 10th IEEE Int. Enterprise Distributed Object Computing Conference, 2006, 391-394.

[19] Chiu, D. K. W., Hong, D., Cheung, S. C., and Kafeza, E. Towards Ubiquitous Government Services through Adaptations with Context and Views in a Three-Tier Architecture. Proc. of the 40th Annual Hawaii Int. Conf. on System Sciences, IEEE Computer Society, 2007.

[20] Bo, C., Xiang-Wu, M., and Jun-Liang, C. An Adaptive User Requirements Elicitation Framework. 31st Annual Int. Computer Software and Applications Conf., 2007, 501-502.

[21] Xiang, J., Liu, L., Qiao, W., and Yang, J. SREM: A Service Requirements Elicitation Mechanism based on Ontology. COMPSAC, 2007.

[22] Markose, S., Liu, X.F., and McMillin, B. A Systematic Framework for Structured Object-Oriented Security Requirements Analysis in Embedded Systems. IEEE Int. Conf. on Embedded and Ubiquitous Comp., 2008,75-81.

[23] Oyama, K., Jaygarl, H., Xia, J., Chang, C. K., Takeuchi, A., and Fujimoto, H. Requirements Analysis Using Feedback from Context Awareness Systems. COMPSAC, 2008.

[24] Ming, H., Oyama, K., and Chang, C. K. Human-Intention Driven Self Adaptive Software Evolvability in Distributed Service Environments. 12th IEEE International Workshop on Future Trends of Distributed Comp. Systems, 2008, 51-57.

[25] Laitenberger, O., El Eman, K., and Harbich, T. G. An Internally Replicated Quasi-Experimental Comparison of Checklist and Perspective-Based Reading of Code Documents. IEEE Trans. Softw. Eng., IEEE Press, 2001.

[26] Lima, G. M. P. S.; Dias Neto, A. C.; Travassos, G. H.. A Heuristic Based Testing Strategy for the Identification of Class Integration Order in Object-Oriented Software. CLEI Electronic Journal, v. 11, p. 1-13, 2008.

[27] Goldsby, H.J.; Sawyer, P.; Bencomo, N.; Cheng, B.H.C.; Hughes, D. Goal-Based Modeling of Dynamically Adaptive System Requirements; Engineering of Computer Based Systems, 2008. ECBS 20008.