Requirements Specifications and Recovered Architectures as Grounded Theories

Daniel M. Berry[1] (dberry@uwaterloo.ca),
Michael W. Godfrey[1] (migod@uwaterloo.ca),
Ric Holt[1] (holt@uwaterloo.ca),
Cory J. Kapser[1] (cjkapser@uwaterloo.ca), and
Isabel Ramos[2] (iramos@dsi.uminho.pt)

[1] Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada
[2] Departamento de Sistemas de Informação, Universidade do Minho, 4800 Guimarães, Portugal

Abstract. This paper describes grounded analysis (GA) as a method to discover grounded theories (GTs) be subjected to later empirical validation. The paper shows that a good instance of Requirements Engineering (RE) or Architecture Recovery (AR) is an instance of GA for the purpose of discovering the artifacts that RE or AR produces. Therefore, these artifacts are also GTs.

1 Grounded Theory

Grounded analysis (GA) [1] is a method for developing grounded theories (GTs) [2], each of which consists of a collection of hypotheses, about a human process. In the 1960s, discomfort was growing with the application of traditional statistical methods to understanding and explaining social phenomena. GA was developed in response to this discomfort to provide a means gather detailed empirical evidence for theory that could be later subjected to traditional statistical empirical validation using controlled experiments or other means. GA is an adaptive research process for finding emergent theory that could not be anticipated in advance of the research. The researcher adapts the research process based on what he[3] has learned from the data he has seen so far in order to pursue data that support the emergent theory. Therefore, not only is the theory emergent, but also the process and the set of data that are sought are emergent, as the researcher learns more and more about the phenomena involved and, thus, what data should be sought. Glaser [3] claims that everything is potentially data to the GA practitioner.

The steps of the GA process are:

1. **Data collection**: collecting data about the phenomena to be modeled from a representative population,

[3] An arbitrary practitioner of GA is without loss of generality assigned the male gender and an arbitrary requirements or architectural analyst is without loss of generality assigned the female gender.

2. **Coding**: coding the data in order to understand and categorize them,
3. **Sampling**: sampling the data by focusing on some categories,
4. **Memoing**: recording the data about categories found to be important into memoranda,
5. **Sorting**: sorting the memoranda by categories, and
6. **Writing up**: writing up the hypotheses that have been developed.

Brower and Jeong [1] provide more detailed kinds of coding, and some, e.g., Dick [4] add a note-taking step between Items 1 and 2.

Steps 2 through 4 repeat until a set of hypotheses deemed worth of testing empirically is formed. While the steps are numbered in a particular order—the order even makes sense, because nothing can be written up until there is something to be written up—the reality is that dynamism reigns. In the middle of doing one step, one might see the opportunity for information requiring initiation of a different step. Hence, the steps can and do happen simultaneously.

A GA practitioner immerses himself in an instance of the method, observing, with as little prejudice as is possible, what is happening, and drawing conclusions supported by his ongoing observations. Ideally, the GA practitioner should begin the GA process with no hypotheses that he hopes to prove, in order to avoid being swayed (1) to see things that are not there and (2) to miss things that are there. In reality, totally avoiding opinions is impossible, but he should be aware of the opinions he does form, in order to keep himself honest. Moreover, he must clearly state his opinions in any write-up so that others can understand from where his decisions came [5].

Section 2 argues that Requirements Engineering and Architecture Recovery *are* GAs and thus that requirements specifications and recovered architectures are GTs. Section 3 describes related work, and Section 4 concludes the paper.

2 RE and AR as GAs

It has occurred to us that:

- Requirements Engineering (RE) is using GA to discover and construct requirements of the computer-based system (CBS) that a client needs and wants, and
- Architecture Recovery (AR)[4] is using GA to discover and construct the architecture of the CBS being examined.

A consequence of this observation is that

- the requirements specification that results from a RE effort is a GT, and
- the recovered architecture that results from an AR effort is a GT.

The standard GA steps can be applied directly to RE and to AR. All that are changed are the subjects examined and the artifacts produced. As with any other GA effort, it is best that the requirements analyst or architectural analyst avoid having preconceived ideas of the outcome.

[4] Architecture recovery is a major and essential component of reverse engineering, whose common acronym, "RE", is identical with that for "requirements engineering". However, reverse engineering includes steps that are not considered in this paper and is regarded as outside the scope of this paper. In this paper, "RE" means only "requirements engineering".

2.1 RE as a GA

RE has as its purpose to discover requirements for a CBS to be built by developers at the behest of a client for the benefit of users [6]. In RE for a CBS, the requirements analyst initially has a vague notion of the CBS's requirements, i.e., what the CBS is supposed to do. By reading requests for proposals (RFPs), vision documents, and other written materials supplied by the client of the CBS, by talking with the client, users, or both, of the CBS, the requirements analyst begins to build a mental model of the CBS to be built. Each mental model must be both validated and refined by asking questions of the client and users. The questions asked at any time are determined by the mental model that has emerged so far. That is, the requirements analyst asks follow-up questions to clarify what he has learned already and to test emerging hypotheses.

While the typical requirements analyst may not specifically follow the six steps of GA, she normally does every step in some form, possibly in a different order and possibly in parallel, *as is allowed in traditional GA*.

The RE variants of the steps of the GA process are:

1. **Data collection**: collecting requirement ideas from (1) an RFP; (2) vision documents; (3) interviews of clients and users; (4) client and user reactions to draft use case descriptions, draft software requirements specification (SRS) sections, models, prototypes, etc.; (5) etc.,
2. **Coding**: (1) classifying requirements as functional or nonfunctional; (2) ranking requirements by necessity, desirability, feasbility, costs, etc.; (3) determining stakeholders affected by and affecting each requirement; (4) clustering requirements into feature groups; (5) etc.,
3. **Sampling**: asking customers and users follow up questions about the various codings of requirements ideas,
4. **Memoing**: writing stories, use cases, SRS sections, etc.,
5. **Sorting**: sorting the memoranda by categories, and
6. **Writing up**: writing up the final SRS.

With the application of GA, RE for a CBS becomes an interpretive and collaborative effort to develop a contextual and in-depth GT about the CBS that a client needs and wants. The CBS's requirements should be constructed jointly by the developers and the client and users so that the clients and users will be motivated to support and use the CBS when it is finally built [7]. As with any other GT, this GT must be validated. This validation is done by having the client, the users, and other stakeholders accept the requirements specification as specifying their collective requirements.

Note that it is understood in RE that requirements are both discovered, by elicitation, and constructed, by invention [6, 8].

2.2 AR as a GA

AR has as its purpose to determine a useful and reasonable model of the software architecture of an existing CBS [9]. Although AR is sometimes called "architecture extraction", that term is misleading, in that an explicit architectural model of a CBS commonly

exists neither in the actual CBS nor in its documentation. Moreover, the architecture often does not exist even in the minds of the developers. AR typically begins by searching for hints or descriptions of the architecture, such as might exist in any documentation of the CBS. Often, no such hints or descriptions exist, and the search may be carried out by interviewing the CBS's software architect, if any; developers; and other key stakeholders. The source code of the CBS may be analyzed manually, using fact extractors that automatically create a graphical representation of the code, or both.

The architectural analyst carrying out this analysis generally begins understanding neither the target architecture nor the best way to discover this architecture. Rather, she follows a process which is essentially that of GA. This process gathers more and more data about the CBS and develops, in an emergent fashion, what is hoped to be an increasingly useful and detailed model of the architecture of the CBS [10]. Involving developers in this process helps in two ways: The developers can provide intimate knowledge of the implemented CBS and at the same time, can direct the creation of a model that is more likely to be useful to the developers. As the AR process proceeds, the analyst makes decisions on the fly, (1) that modify the process to deal better with the data gained so far and (2) that refine the emergent model of the CBS's architecture.

The AR variants of steps of the GA process are:

1. **Data collection**: (1) collecting any reports that may document the CBS's architecture or aspects of it; (2) interviewing key stakeholders about the architecture; (3) inspecting the source code, manually or with tool support; (4) interacting with the running CBS, often using an interactive debugger or other instrument; (5) etc.
2. **Coding**: classifying collected information as essential or coincidental to the architecture, determining aspects of the CBS which have importance to the stakeholders and to the architecture, preliminary division of the CBS into upper level subsystems, etc.
3. **Sampling**: (1) probing the source code, or any preliminary graph model, to see if any proposed decompositions are reflected in the actual implementation; (2) asking stakeholders if a proposed decomposition is useful and intuitive; (3) etc.
4. **Memoing**: (1) writing up preliminary descriptions of modules or components; (2) preparing preliminary diagrams of module or component interactions, as determined thus far; (3) etc.
5. **Sorting**: (1) collecting and sorting the various data, descriptions and diagrams, along with collected motivations, toward formulating an model of the overall architecture; (2) etc.
6. **Writing up**: writing up a description of a determined model of the architecture, including motivating rationale, top-level decomposition into subsystems, description and documentation of those subsystems, and further descriptions and decomposition as appropriate to the CBS.

Thus AR is a collaborative process for developing a GT about the architecture of a CBS. Some elements of the GT, e.g., the code facts, are discovered by examining the CBS, and some other elements of the GT, e.g., the architecture, are constructed by thinking about the discovered facts. This GT must be validated by showing the recovered architecture to whatever developers are available for consultation.

3 Other Work

GA has been used extensively to develop theories explaining social processes of all kinds [e.g., 2, 3, 11, 12], even in technical disciplines such as software engineering [e.g, 5, 13, 14], RE [e.g., 7, 15–18], and AR [e.g., 19–22]. We call these uses of GA *methodological uses* because they study methods. While there is much empirical work, including using GA, about RE and AR methods, in order to understand RE and AR, to the authors' knowledge, there is no other work that specifically desccribes either RE or AR as an empirical method itself. However, Galal and Paul [16] did describe one part of RE as a GA when the presented GSEM (Grounded System Engineering Methodology), a grounded analysis method for "developing qualitative scenarios against which statements of requirements can be evaluated."

4 Conclusions

RE for a CBS can be viewed as a GA for the purpose of discovering the CBS's requirements. AR for a CBS can be viewed as a GA for the purpose of discovering the architecture of the CBS. In brief, GA provides a systematic description of the processes of RE and AR, which might otherwise seem to be random searches. Consequently, the requirements specification emerging from a RE effort or the recovered architecture emerging from an AR effort is a GT that must be subjected to validation in a manner appropriate for the artifact.

The emergence of the information that RE or AR normally finds is consistent with considering RE and AR as GAs. In each of RE and AR, not only is the final product of the activity emergent, but also the process by which the final product is discovered is emergent. This observation says that any attempt to standardize RE or AR methods is unlikely to succeed.

Author Berry has often said in his RE courses that each problem seems to beget its own RE method. Certainly, he never predetermines how he will discover any particular client's requirements. He listens and adapts his methods to the emerging situation. Our reading of the requirements engineering textbooks by Gause and Weinberg [8] and by Robertson and Robertson [6] suggests that each of these authors operates in the same way. Cockburn [23] agrees for the entire lifecycle, not just RE.

References

1. Brower, R.S., Jeong, H.S.: Grounded analysis: Beyond description to derive theory from qualitative data. In Yang, K., Miller, G.J., eds.: Handbook of Research Methods in Public Administration, Boca Raton, FL, USA, Auerbach, Taylor & Francis (2008) 823–839
2. Glaser, B.G., Strauss, A.L.: The Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine, Chicago, IL, USA (1967)
3. Glaser, B.G.: Basics of Grounded Theory Analysis: Emergence vs. Forcing. Sociology Press, Mill Valley, CA, USA (1992)
4. Dick, B.: Grounded theory: a thumbnail sketch. Technical report, Graduate College of Management, Southern Cross University, Lismore, NSW, Australia (2005, Accessed November 2007) `http://www.scu.edu.au/schools/gcm/ar/arp/grounded.html`.

5. Walsham, G.: Interpretive case studies in IS research: Nature and method. European Journal of Information Systems **4** (1995) 74–83
6. Robertson, S., Robertson, J.: Mastering the Requirements Process. Second edn. Addison-Wesley, Harlow, UK (2006)
7. Ramos, I.M.: Aplicações das Tecnologias de Informação que Suportam as Dimensões Estrutural, Social, Política e Simbólica do Trabalho. PhD thesis, Departamento de Informática, Universidade do Minho, Guimarães, Portugal (2000)
8. Gause, D., Weinberg, G.: Are Your Lights On? How to Figure Out What the Problem REALLY Is. Dorset House, New York, NY, USA (1990)
9. Chikofsky, E.J., Cross, J.H.: Reverse engineering and design recovery: A taxonomy. IEEE Software **7** (1990) 13–17
10. Holt, R.: Software architecture as a shared mental model. In: Proceedings of the Tenth International Workshop on Program Comprehension (IWPC). (2002)
11. Jeong, H.S.: A Grounded Analysis of the Sensemaking Process of Korean Street-Level Fire Service Officials. PhD thesis, Public Administration, Florida State University, Tallahassee, FL, USA (2006)
12. Pershin, G.: Adoption of Policies that Permit Community Colleges to Grant Bachelor Degrees in Florida. PhD thesis, Public Administration, Florida State University, Tallahassee, FL, USA (2006)
13. Carver, J.: The use of grounded theory in empirical software engineering. In: Empirical Software Engineering Issues. Critical Assessment and Future Directions. LNCS 4336, Berlin/Heidelberg, Germany, Springer (2007) 42
14. Coleman, G., O'Connor, R.: Using grounded theory to understand software process improvement: A study of irish software product companies. Information and Software Technology **49** (2007) 654–667
15. Power, N.: A Grounded Theory of Requirements Documentation in the Practice of Software Development. PhD thesis, Dublin City University, Dublin, Ireland (2002)
16. Galal, G.H., Paul, R.J.: A qualitative scenario approach to managing evolving requirements. Requirements Engineering Journal **4** (1999) 92–102
17. Calloway, L.J., Knapp, C.A.: Using grounded theory to interpret interviews. Technical report, School of Computer Science and Information Systems, Pace University, New York, NY, USA (1995, Accessed November 2007) `http://csis.pace.edu/~knapp/AIS95.htm`.
18. Lang, M., Fitzgerald, B.: Web-based systems design: a study of contemporary practices and an explanatory framework based on "method-in-action". Requirements Engineering Journal **12** (2007) 203–220
19. Briand, L.C.: The experimental paradigm in reverse engineering: Role, challenges, and limitations. In: Thirteenth Working Conference on Reverse Engineering (WCRE). (2006) 3–8
20. Kapser, C.J., Godfrey, M.W.: "Cloning considered harmful" considered harmful. In: Thirteenth Working Conference on Reverse Engineering (WCRE). (2006) 19–28
21. Sillito, J., Volder, K.D., Fisher, B., Murphy, G.: Managing software change tasks: An exploratory study. In: Proceedings of the International Symposium on Empirical Software Engineering (ISESE). (2005) 23–32
22. Sillito, J., Wynn, E.: The social context of software maintenance. In: Proceedings of the Twenty-Third IEEE International Conference on Software Maintenance (ICSM). (2007) 325–334
23. Cockburn, A.: Selecting a project's methodology. IEEE Software **17** (2000) 64–71