Panel on Reverse Engineering and Architecture Evolution (14:05–15:50)

Alessandro Maccari. The goal of this panel is to narrow the gap between reverse engineering and architecture research communities[1]. At the moment, they do not know anything about each other; researchers from one field do not have adequate knowledge of the other etc.

Arie van Deursen. Informed everybody that in the February of the next year there will be a Dagstuhl seminar on software architecture, so attendants should be already waiting for invitations ☺

The goal of his position statement was to present SIMPLE – Simple Incremental Method for Product Line Evolution. The main notion of SIMPLE is *package*, which is a basic development unit that could be built and deployed independently. The method is also supported by a tool "autobundle". From the developer's point of view, at every "evolution point" – be it horizontal, vertical or some combination of both (the term "diagonal" was proposed for this situation, though I think that it should be called plain or usual evolution) – you should rethink the organization of the product (or product line), or recover required variability. Still, there are problems, such as cross-cutting features, for instance, turning logging on and off would affect many other features.

Question from me: what are these evolution points and how can we tell whether this is an evolution point or not?
Answer: usually, the evolution decisions are made just before releasing the next version of software. It does not make sense to make these decisions more often (definitely this is not an everyday process).

Claudio Riva. Started his position statement with the question: how many people in the audience had papers in both communities? Turned out that quite few of us did. Why there is such a gap between these communities?

Probably, the problem could be partially attributed to the fact that software architecture community could not express clearly what they want to reengineer. But the same problems are also on the other side – reverse engineering tools do not try to abstract anything tangible from the source code. This could be illustrated by the results of Dagstuhl seminar on reengineering exchange formats that was held last year. Everybody quickly agreed on exchange formats for AST and even call/dependence graphs, but there was no consensus on the format of the architecture. Even worse, there was strong disagreement on the nature of the artifacts that should represent architecture. So we should find a way to work on bridging the gap between Software Architecture and Reverse Engineering communities.

Claudio also proposed Feature-Oriented Reverse Engineering as a viable approach to architecture recovery. Everybody understands what feature is! Features also represent assets in product lines, features are reused etc.

Elliot Chikofsky noted that WICSA and WCRE seminars most probably will be held together in 2003, so if anybody finds an answer to these questions, then there is still about a year and a half to write a paper that could address both audiences.

Question from the audience: what is a feature? **Arie answered** that at the moment, there is no formalization of this notion, and everybody relies on their intuitive understanding of what "feature" is. Still, even without formalization, this notion helps to come up with useful results, so probably that's OK for now to leave it as is. **Alessandro noted**: in NOKIA the feature is defined as something that the customer is willing to pay for.

Derek Rayside presented his position statement "Polymorphism as a Problem" and said that he can even assert that polymorphism is THE problem in the moder software development. We have to deal with polymorphic calls at all levels, starting with architecture to reverse engineering to metrics to source code.

---

[1] Actually, the first goal of the panel that Alessandro mentioned was "to recover from my hangover". I couldn't help but mention it here, because we didn't find out whether this goal was accomplished ☺

Derek also presented several approaches to polymorphic analysis – Rapid Type Analysis, dealing with parametric polymorphism etc.

Galal Hassan Galal "Reverse Software Architectonics". Why do you reverse architect?

From the books on "usual" (non-software) architecture, we can learn that there exists principle of architecting for adaptability. The idea is that some things are less likely to change than the others (for instance, the site of the building is unlikely to change; structure of the building is also difficult to change etc.)

Question from me: I don't think that there are a lot of stable things in software (even the hardware platform or programming language may change during the lifetime of a software system). What could be chosen in software architecture as "stable parts" (equivalent to site and structure in usual architecture)?
Answer: It seems that the most stable thing in software is domain concepts, so we have to identify these concepts and then we will be able to identify them in most software architectures. Another advice is to cross-check two different views (software system and the application domain). Their intersection is quite likely to give you the "most stable" part of the system.

Tarja Systä commented that "designing for change" might be very difficult even if you specially invest in doing so. Recently there was a paper[2] by Tommi Mikkonen and Peeter Pruuden describing the project, in which they tried to achieve greater flexibility in their design, because they anticipated changes. Nevertheless, they had to rewrite the whole system four times in order to keep up with changing requirements.

This was the last position statement and we turned to discussion.

Question from Albert Zündorf: What do I do after I analyzed my architecture? What's the payback?
Answer: The idea is simply to check your architecture. Hopefully, you will find that your architecture is OK; otherwise you will have to re-architect your application in order to secure the "most stable" parts and make sure that they are singled out and formalized.

Comment from Tarja: architecture is actually more of a way to communicate between developers than reference or guidance. Reverse engineering always starts with a question, with something that you would like to find out about software. Perhaps we should do the same dealing with architecture – once you formulate the question, it becomes easier to understand what to you want from architecture.

Galal responded that he disagrees with the statement that designs are made for communication between developers. He believes that architecture usually provides guidance for developers in their small-scale decisions. He also said that the architecture is aimed at providing an overall structure, and made an analogy with water and steel: water is very flexible, but you cannot build anything from it, while the steel is not flexible, but provides adequate material for building.

Elliot said that the analogy does not hold, because in software we are dealing with water, not with steel! The software is so easy to change that you are unlikely to find anything stable.

Galal: placing software in the context helps you to freeze various aspects of software – you have documents, customers, requirements etc. They all help to form the structure of the software system, to make it more rigid.

Vaclav Rajlich asked: can you assure rigidity in advance or is it a result of the process?
Arie answered: software development is a non-linear learning process. You start with your current knowledge and then you build and learn in the process. Maybe you will understand that you made a mistake earlier, but you do not know that in advance. So yes, the structure of the system reflects the process of its development[3].

---

[2] T. Mikkonen, P. Pruuden "Practical perspectives on software evolution and architectures", In Proceedings of International Workshop on Principles of Software Evolution 2001, pp. 115-119, Vienna, Austria, September 2001.
[3] Interestingly enough, Arie answered to Vaclav's question with approximately the same phrases that Vaclav used in his own keynote talk the previous day!

Albert asked: Can you say that this architecture is better or worse than the other?
Galal: yes, we can answer by analyzing the domain. **Tarja commented** that in her opinion, such analysis makes sense only depending on the purpose. **Arie added** that there is an ATAM method for analyzing/comparing architectures (see http://www.sei.cmu.edu/ata/ata_method.html), which was developed at SEI.

Vaclav asked: this architectural committee of 12 in NOKIA that was mentioned earlier – does it work?
Alessandro answered: Yes, it does and the results are sometimes quite unexpected. For instance, we found out that some things that we believed to be sacred – weren't, and some things that we considered to be easy to change weren't so easy (e.g. new communications protocol seems to be a simple addition, because there already are so many existing protocols supported; still, in some cases it isn't so easy, because new protocol probably means new services, new user interface etc.)

NOKIA is actually quite good in solving these architectural questions. It works in the following way: if the team leader thinks that the issue that he is dealing with will not affect other teams, then he makes the decision. Otherwise he passes the problem to the next level, to the person overseeing several groups simultaneously and so on. At the top level there are people that make the most influential architectural decisions; these decisions are likely to influence every developer.

I asked whether these top-level people have any idea about what's going on below them and whether they retained any understanding of software. I also wanted to hear an example of top level decision.
Alessandro answered that everything works just fine, because the top-level decisions are on greater level of abstraction than pure software. As an example of such decisions he mentioned changing the scheme of messaging between components from function calls to asynchronous messages.

Elliot noted that the terminology used during the panel is inappropriate: reverse engineering and reverse architecting are not two different entities. The thing is that reverse engineering consists of low-level and high-level reverse engineering (the latter is what we called today reverse architecting), we just forgot about the existence of the high-level part in our discussion.

Alessandro summarized the discussion:

1. There is a wide gap between domain model (used by architects) and solution model (used by developers) – both in practice and research. We need to somehow overcome this gap.
2. We believe that architecture changes should be done in small steps – we don't want dramatic changes (Arie and Claudio mentioned this in their position statements; Galal's layered scheme is also in line with this idea)
3. Features represent a good way of describing the (potentially recovered) architecture; all the participants were quite content with NOKIA definition ("something that the customer is willing to pay for").
4. It remains unclear what is architecture good for – communication? Guidance? More research is needed in this area.