Towards Multi-Party Evolution of Social Software

Arnd Poetzsch-Heffter
University of Kaiserslautern

Barbara Paech
University of Heidelberg

1 Software Supporting Social Processes

Many modern communication and information systems are increasingly embedded into society. These systems stretch across many organizations, have very large, heterogeneous user groups, and support complex processes. Accordingly, requirements engineering methods include techniques for detailed stakeholder modeling that go far beyond the classical *role separation* into *customer*, *developer*, and *user* [AR04]. Furthermore, the system requirements often depend on rules decided by stakeholders not involved in the software development and maintenance. In particular, changes in legal regulations or laws directly call for modifications of administrative software. Even for today's software systems, the complexity of the relationship between social and administrative processes on the one side and software development and maintenance processes on the other side is often underestimated and leads to severe engineering problems. For example, the project "HochschulStart" that is aiming to build a system for application and enrollment of students at German universities has suffered from the large number of involved universities and ministries (see [Mer12]). It also illustrates the close relationship between regulations, social processes and software as well as the resulting development challenges.

We believe that an increasing number of existing and future social processes will be supported by software systems. Existing examples are Wikipedia, conference systems, social networks, and platforms for crowdsourcing and open source development. Within these software systems and platforms for social processes and activities, that we call *social software* in the following, the users get a more prominent role than in classical software:

- Interaction and content provided by users is often crucial for the success of social software. Thus, the systems have to support *mechanisms for flexible adaptation* to users' needs. However, managing such adaptations is a challenge, because many of these systems have very large user groups.
- A clear role separation between user, developer, and customer is given up. Users can modify data that affect the work of others, can grant access rights to other user groups, and might even participate in the software development. Thus, a more *detailed management of rights and policies* to handle conflicts is needed. Whereas in classical software engineering conflict resolution among user groups is part of the requirement engineering, social software needs policies and mechanisms to also resolve conflicts at run-time (e.g., Wikipedia needs a mechanism to accept articles).

In summary, social software lives in a context of heterogenous user groups with different rights and many kinds of possible conflicts. Policies are needed to resolve such conflicts. However, for maintenance and evolution of such software this is not enough. Evolution might want to restructure the stakeholder groups and change the policies without breaking systems integrity or data privacy. This becomes challenging, if different stakeholders want to be involved in the evolution process, i.e., in case of so-called *multi-party* evolution.

2 Model-based Multi-Party Evolution of Social Platforms

To address the sketched challenge of multi-party evolution, we propose an *explicit*, *integrated*, and *self-referential* modeling approach comprising at least the following models:

- a stakeholder model specifying all groups that might develop, influence or use the system and participate in its evolution
- a data model specifying the data formats and ownership aspects
- a software architecture specifying the features and components of the system
- a policy model specifying the rights and mechanism to access and modify
  - the data stored in the system
  - the features and components of the system
  - all of the models mentioned above, in particular the policy model itself

These models should explicitly be described in some formal language and maintained as part of the system. They should be integrated in the sense that there are clear consistency constraints among the models, e.g., policies are expressed in terms of the modeled stakeholders. In addition, and this is the central aspect, the models are self-referential, i.e., they also describe how to evolve the models themselves. For example, the policy model has to grant the right to modify itself to one of the groups mentioned in the stakeholder model; otherwise it cannot be modified during evolution steps. Thus, the models become first-class artifacts of the evolution process. This is similar to legislative systems that comprise laws on how to modify the law.

In summary, we argue that in addition to rules describing how social software works and how it can be evolved, we need rules about the evolution of the models and rules.

References

[AR04] Ian F. Alexander and Suzanne Robertson. Understanding Project Sociology by Modeling Stakeholders. *IEEE Software*, 21(1):23–27, 2004.

[Mer12] Peter Mertens. Schwierigkeiten mit IT-Projekten der öffentlichen Verwaltung – Neuere Entwicklungen. *Informatik Spektrum*, 35(6):433–446, 2012.