

Response Paper

Towards Automatically Generating Summary Comments for Java Methods (Sridhara et al.)
Embodied Social Proxy: Mediating Interpersonal Connection in Hub-and-Satellite Teams
(Venolia et al.)
An Automated Procedure for Identifying Poorly Documented Object Oriented Software
Components (Pendharkar)

Rafael Kallis

March 18, 2017

This week's articles have faced the issues of social distance among distributed colleagues and lack of adequate documentation in object oriented code. Venolia et al. have introduced ESP, a videoconferencing terminal dedicated to a specific satellite worker for realtime communication, which aimed to improve interactions of meeting participants. The ESP seems to show improvements by allowing the hub to associate to the remote person with a device, giving him a form of physical presence. Additionally, the ESP features a display and multiple cameras which allow the remote worker to have a better perception of the hub's surroundings and interactions. One feature the ESP failed to include is gaze awareness, the ability to allow hub members to determine where the remote worker precisely is looking at.

Sridhara et al. have shared their contributions on another spectrum of problems. By generating descriptive summaries for functions in natural language, they aim to decrease the effort of understanding a maintenance task and the code related to it. Whilst their generator produces highly qualitative documentation, it is limited to code written in Java. Additionally, their

generator doesn't support class or global variable documentation. I still believe that their contribution is significant, since the issue of programming language specificness can be tackled in the future. If it is possible with Java, what holds us back making other language specific generators? Nevertheless, the possibility of parameterising the comment generator based on the user's, or company's, preferences would also be quite important.

Pendharkar's work also contributes in the same area. He proposed an artificial neural network (ANN) backed procedure which can be used to determine poorly documented software components in a quick and unbiased way. His ANN takes as an input various measures, such as number of methods, sub-classes, events and UI elements and computes the difference between actual and predicted source and documentation lines of code. The differences are then used to determine optimal, over or under documentation of a component. In contrast to Sridhara et al, Pendharkar's procedure also takes under consideration classes and system components, such as global variables.

In my opinion, the combination of Pendharkar's work with the contributions of Sridhara et al, has potential. Allow me to propose three scenarios, which combine both procedures and could aid providing better documentation or make documenting code more efficient.

- The comment generator can be initially neglected and if Pendharkar's procedure detects poor documentation, the comment generator can be used.
- Initially running the automatic comment generator, then by using Pendharkar's procedure determine possible weaknesses in the generated documentation, which can be manually improved.
- Assuming customization of the Sridhara et al. comment generator is possible, Pendharkar's procedure could help determining the optimal parameters for the comment generator, allowing us to have an even more effective automatic code generator.