Softw Syst Model (2017) 16:927–928
DOI 10.1007/s10270-017-0621-x

EDITORIAL

The importance of flow in software development

Jeff Gray[1] · **Bernhard Rumpe**[2]

Published online: 5 September 2017

From social and psychological theories and studies [1], we know that there exists a mental state called "flow" that allows individuals to concentrate deeply on a specific task without noticing the surrounding environment or the time, while remaining fully aware of the current work that they are doing.

In a recent TV broadcast about cognitive brain function, several illustrative examples were given, such as a free climber who says that he is at peak performance when he completely forgets about the world and the danger associated with the climb, but fully concentrates on the rocks and all the next moves he is planning to make. It was also shown that the world record holders in speed tasks, such as stacking cubes or solving Rubic's Cube, do not use their cerebrum very intensively when executing the speed task. Only a small, but obviously efficient, part of the brain is concentrating on the task to be executed. Similar examples can be found among athletes and musicians who may occasionally get into a "groove" where performance and concentration reach a peak level.

Software developers that fully concentrate on their work also report this kind of flow, where only the relevant parts of the brain are focused on the core task. We can argue that software development is more complex and probably involves more parts of the brain than speed stacking, but it also seems that software development becomes more productive when the developer has the ability to reach flow for a large part of his or her working time.

This ability to reach and sustain flow depends partially on the developer's own circumstances; for example, whether they get enough sleep, have little to no stress at home, and lead a safe and enjoyable life. To a large extent, this ability also depends on the concrete working circumstances. Is the room quiet? Can the developers work on their own for a long time without disturbances by phones, emails, or background noise in the room? It is also important that the developer is in a constant state of focus while thinking of the important issues that enable him or her to execute their task. In the software development context, it is helpful if the tooling aids productivity and does not slow down the focus time (e.g., compilation time should not take long). Agile development methods have the potential to capitalize on the opportunity of software developers to get into the flow and provide continuous improvement to the system they are developing. Mihaly Csikszentmihalyi also argues that the flow is very helpful to engage in creative discovery of new ideas and inventions [1]. Software development can benefit from flow because the need to identify an optimal architecture and the best structure for object interactions can be a creative activity.

Agile development methods also thrive from early and immediate feedback: The software should always be able to compile and to run the associated tests. To remain in the flow, it is helpful that compilation and test executions are quick, because otherwise developers may become distracted by the temptation to read emails, go for the next coffee, take an early lunch, or engage another co-worker in a conversation. Software development tools are vitally important for productive development and keeping developers in the flow zone.

When considering the current state of tooling for model-based software development (compared to just coding), an opportunity exists for new capabilities that help developers

✉ Bernhard Rumpe
bernhard.rumpe@sosym.org

Jeff Gray
jeff.gray@sosym.org

[1] University of Alabama, Tuscaloosa, AL, USA

[2] RWTH Aachen University, Aachen, Germany

achieve flow. Currently, many tools are able to help with creating and editing large models, perform partial consistency checks, and generate code for multiple platforms. But in comparison with the available tooling for traditional general-purpose programming languages, there is still a large gap in tool capabilities. Models are often interacted with in a monolithic form, i.e., all models are processed in batch each time a code generation request is started. The time it takes to perform code generation and model checking may cause a disruption in the flow. If a code generation process (for a large industrial model, or a set of models within the project) takes longer than drinking a cup of coffee, software developers that use model-based techniques may lose their flow of concentration. They will not get the same feeling of satisfaction that would result from a better transition across the tool usage, which may hamper productivity when delays emerge. We hope that modeling tools will improve the opportunity for developers to achieve flow through improved tool implementation, but also by better modeling languages that enhance modularity and incremental compilation.

We hope that you have fun and remain in the flow when reading the articles in this issue!

1. Csikszentmihalyi, M.: Finding Flow: The Psychology of Engagement with Everyday Life. Basic Books, New York (1997)

Content of this issue

This issue contains the following twelve regular papers:

- "From software extensions to product lines of dataflow programs" by Rui Gonçalves, Don Batory, João Sobral, and Taylor Riché.
- "An approach based on the domain perspective to develop WSAN applications" by Taniro Rodrigues, Flavia Delicato, Thais Batista, Paulo Pires, and Luci Pirmez.
- "Generating process model collections" by Zhiqiang Yan, Remco Dijkman, and Paul Grefen.
- "Promoting traits into model-driven development" by Vahdat Abdelzad and Timothy Lethbridge.
- "Process mining using BPMN: relating event logs and process models" by Anna Kalenkova, Wil M.P. van der Aalst, Irina Lomazova, and Vladimir Rubin.
- "FLAME: a formal framework for the automated analysis of software product lines validated by automated specification testing" by Amador Durán, David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés.
- "Towards an integrated formal method for verification of liveness properties in distributed systems: with application to population protocols" by Dominique Mery and Mike Poppleton.
- "Model checking multi-level and recursive nets" by Mirtha Fernández Venero and Flávio Corrêa da Silva.
- "On the formal interpretation and behavioural consistency checking of SysML blocks" by Andrew Simpson and Jaco Jacobs.
- "Variability extraction and modeling for product variants" by Lukas Linsbauer, Roberto Lopez, and Alexander Egyed.
- "A model framework-based domain-specific composable modeling method for combat system effectiveness simulation" by Xiaobo Li, Yonglin Lei, Weiping Wang, Feng Yang, and Yifan Zhu.
- "A novel model-based testing approach for software product lines" by Ferruccio Damiani, David Faitelson, Christoph Gladisch, and Shmuel Tyszberowicz.