

Formal Methods, to use or not to use?

Tobias Bannwart
Anil Kandrical
Jef Van Loon

Overview

1. Introduction: The Ultimate Goal of Software Engineering
2. How can we use Formal Methods?
3. Benefits of Formal Methods
4. Disadvantages, Risks and Limitations of Formal Methods
5. Transfer of Formal Methods from Science to Practice
6. Conclusion

Introduction: The Ultimate Goal of Software Engineering

- [Build better and cheaper software (Polak)

- "better" refers to the qualitative aspects (e.g. efficiency, correctness)

- "cheaper" refers to the overall costs (e.g. development, production)

- [Can formal methods help?

How can we use Formal Methods?

— [Verification

- formal proof that software is a correct realization of a specification
- full proofs rarely carried out

— [(Intensive) study of key problems

- mathematical study of selected difficult aspects of the system
- avoids costs of complete formal specification
- risk of overlooking security relevant aspects

— [Refutation

- not trying to prove that the system meets the requirements
- based on computable properties or execution of the model

Benefits of Formal Methods

- [Models can be transformed to code (Polak)

- [Prove the correctness of a model or the satisfaction of certain properties (e.g. safety and liveness properties)

- [Identification of inconsistencies in the requirements

- [The "second time phenomenon" (Berry)

- performing a task for the second time leads to better results (based on the assumption that programming is a form of formal specification)

Disadvantages, Risks and Limitations of Formal Methods

- [Expensive in terms of time and money

- [Model checking requires simplification

- simplification might hide existing problems

- [Don't offer help in identifying requirements (elicitation)

- "There's no point to using exact methods where there's no clarity in the concepts and issues to which they are to be applied" (von Neumann/Morgenstern)

Transfer of Formal Methods from Science to Practice

Communication gap between theoreticians and practitioners (Polak)

- theoreticians: programs are mathematical objects that never fail, if we get the specification right
- practitioners: theoreticians deal with toy examples, we build large scale systems

Educational, social and managerial aspects (Polak)

- educational: software engineers could use theoretical results, if they knew how
- social: software engineers are reluctant to take outside advice
- managerial: change is perceived as risky (potential impacts on schedules and costs)

Conclusion

- [**Useful in a limited domain**

- broad application is not economic

- [**How much formality?**

- as much as needed, as less as possible

- [**No silver bullet (again)**

- no method for all application domains

- not all applications require formal specification

Q&A

Thank you.

