



Universiteit
Leiden
The Netherlands

Static analysis of unbounded structures in object-oriented programs

Grabe, I.

Citation

Grabe, I. (2012, December 19). *Static analysis of unbounded structures in object-oriented programs*. Uitgeverij BOXPress, Oisterwijk. Retrieved from <https://hdl.handle.net/1887/20325>

Version: Corrected Publisher's Version

License: [Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)

Downloaded from: <https://hdl.handle.net/1887/20325>

Note: To cite this publication please use the final published version (if applicable).

Cover Page



Universiteit Leiden



The handle <http://hdl.handle.net/1887/20325> holds various files of this Leiden University dissertation.

Author: Grabe, Immo

Title: Static analysis of unbounded structures in object-oriented programs

Issue Date: 2012-12-19

Stellingen

behorende bij het proefschrift

Static Analysis of Unbounded Structures in Object-Oriented Programs

door Immo Grabe

1. If you do it, do it right! To reason about a program efficiently a formalism must be suited to express the features of the programming language in a natural way, i.e. do not use object activation to mock object creation but extend the formalism to deal with object creation (Chapter 2).
2. Deadlock is an iceberg! The current notions of deadlock only cover parts of a bigger problem - lifelock and mixed forms (Chapter 3 and 5).
3. More than one road leads to Rome! Futures and promises are an alternative model for concurrency. Our formalism allows for a comparison between the model featuring futures and promises and the model featuring multi-threading (Chapter 4).
4. If you preach reuse, reuse! Not only software components can be reused but also formalisms and techniques. In such cases these need not be reinvented to fit the problem setting but the problem setting can be translated to allow for the reuse of the formalism or technique (Chapter 5).
5. Most of the effort spent and progress made in theoretical computer science affects only a little part of practical software engineering.
6. Any computer scientist or software engineer facing challenging problems should have a strong background in theoretical computer science.
7. Even if the problem is understood, understanding a formalism is less than half the way to solve the problem. Experience is required to solve a problem.
8. There are bad proofs - even if they are correct. Good proofs are simple and elegant.
9. The biggest benefit - even bigger than your environmental conscience - of travelling by train is the time you can spend on reading - often even more than planned.

10. Implementing complex systems is as much a management task as a computer science problem. Contributing to the success of a project is as much an educational task as an engineering task to a computer scientist or software engineer.